SOUTHWEST RESEARCH INSTITUTE

POST OFFICE DRAWER 28510 • 6220 CULEBRA ROAD • SAN ANTONIO, TEXAS, USA 78284 • (512) 684-5111 • TELEX 76-7357

May 22, 1990

Ms. Linda Uljon
NASA - Johnson Space Center
Building #12, Room 275, FS 72
Houston, Texas 77058

Subject: Delivery of Final Report for the Research into Display Sharing for the MCC; NASA Grant Number NAG9-370; SwRI Project Number 05-2922

Dear Ms. Uljon:

Enclosed with this letter is the final report summarizing our research into Display Sharing techniques using X Windows. This report outlines, in detail, an X Windows based approach to Display Sharing. The report also describes the current Display Sharing prototype system now installed and running at both SwRI and NASA laboratories.

Steve Hugg will be contacting you to set up a date for a presentation of the report's contents. If you have any questions or concerns, please call Steve Hugg at (512) 522-2780.

Sincerely,

Melvin A. Schrader
Director
Data Systems Department

MAS:PFF:pam

cc: Steven B. Hugg
Susan B. Crumrine SBC
Paul F. Fitzgerald
Nina Y. Rosson
Stephen R. Johns
William A. Bayliss
Larry Bishop (NASA-JSC)
Mike Kearney (NASA-JSC)
NASA Scientific and Technical Information Facility (2 copies)

SOUTHWEST RESEARCH INSTITUTE
Post Office Drawer 28510, 6220 Culebra Road
San Antonio, Texas 78228-0510

RESEARCH INTO DISPLAY SHARING TECHNIQUES FOR DISTRIBUTED COMPUTING ENVIRONMENTS

FINAL REPORT

NASA Grant No. NAG9-370
SwRI Project No. 05-2922

Prepared by:
Steven B. Hugg
Paul F. Fitzgerald, Jr.
Nina Y. Rosson
Stephen R. Johns

Prepared for:
NASA Lyndon B. Johnson Space Center
Houston, TX 77058

May 22, 1990

Approved:

Melvin A. Schrader

Melvin A. Schrader, Director
Data Systems Department

TABLE OF CONTENTS

1.0 INTRODUCTION 1

2.0 THEORY OF OPERATION 2

2.1 System Operation 2
2.1.1 Displays, Windows, and Clients 2
2.1.2 Transmitting Displays From a Workstation 2
2.1.3 Receiving Displays At a Workstation 6
2.2 User Operation 6
2.2.1 Retrieve Channel Guide 6
2.2.2 Send Display Requests 9
2.2.3 Receive Display Requests 9
2.2.4 Remove Channel Requests 9
2.2.5 Remove Receiver Requests 9

3.0 SYSTEM ARCHITECTURE 15

3.1 The Display Sharing Workstation 15
3.1.1 X Server Modification 15
3.1.2 Protocol Distributor 22
3.1.2.1 PD Initialization 22
3.1.2.2 PD Processing 23
3.1.2.2.1 X Protocol Transmission 23
3.1.2.2.2 State Information Transmission and Shared Memory Requests 23
3.1.2.3 Protocol Distributor I/O Packet Structure . 27
3.1.3 Protocol Receiver 27
3.1.3.1 PR Initialization 27
3.1.3.2 PR Processing 29
3.1.3.2.1 X Protocol Reception 29
3.1.3.2.2 State Information Transmission and Shared Memory Requests 30
3.1.3.2.3 Local Expose Event Solicitation 31
3.1.3.3 Protocol Receiver I/O Packet Structure . . 31
3.1.4 Local Distribution Manager 31
3.1.4.1 LDM Initialization 31
3.1.4.2 LDM Processing 32
3.1.4.2.1 Retrieve TV Guide 32
3.1.4.2.2 Distribution Authorization Request . . . 32
3.1.4.2.3 Reception Authorization Request 32
3.1.4.2.4 Cancel Distribution on Channel 33
3.1.4.2.5 Cancel Reception on Channel 33
3.1.4.2.6 Stop Central Distribution Manager 33
3.1.4.2.7 Quit 33
3.2 Dedicated Display Sharing Host 34
3.2.1 Central Distribution Manager 34
3.2.1.1 CDM Initialization 34
3.2.1.2 CDM Processing 35
3.2.2 Protocol Multiplexer 36

3.2.2.1 PM Initialization 37
3.2.2.2 PM Processing 37
3.2.2.2.1 New Connections 37
3.2.2.2.2 Protocol Dispatching 38
3.2.2.3 Performance and Redundancy 39

4.0 USING THE DISPLAY SHARING PROTOTYPE IN MOSL 40

4.1 Equipment 40
4.2 Setup 40
4.2.1 Removal of Ford Variant Server 41
4.3 Process and Shared Memory Cleanup 42
4.4 Startup 43
4.5 Distributing a Window 44
4.6 Receiving a Window 44
4.7 Shutdown of Display Sharing 45
4.8 Restarting Display Sharing 45
4.9 Starting an Application 45
4.10 Finishing Display Sharing Session 46

5.0 DISPLAY SHARING SOFTWARE DESCRIPTION 47

5.1 Source and Destination Station Software 47
5.1.1 Server 47
5.1.1.1 Modified Server Version 47
5.1.1.2 Pseudo Modified Server Version 47
5.1.2 Protocol Distributor (PD) 48
5.1.3 Protocol Receiver (PR) 48
5.1.4 Local Distribution Manager (LDM) 49
5.2 Dedicated Host Software 49
5.2.1 Central Distribution Manager (CDM) 49
5.2.2 Protocol Multiplexer (PM) 49

6.0 DISPLAY SHARING RESEARCH TOOLS 50

6.1 Protocol Profiler 50
6.2 Stand Alone Version 50
6.3 Display Sharing Version 50

7.0 PROTOTYPE EVALUATION 52

7.1 Workstation Performance 52
7.1.1 Modified X Server Approach 52
7.1.2 Display Sharing Wedge Approach 53
7.2 Network Performance 53

8.0 OUTSTANDING ISSUES 56

8.1 Colormaps 56
8.2 Fonts 56
8.3 Expose Events 56
8.4 Delay Until Appearance 57
8.5 Multiple Windows Per Client 57
8.6 Using the Display Sharing Wedge Approach 57
8.7 Shared Memory and Semaphores 57
8.8 Optimized X Code 58
8.9 Discontinued Distribution of Window 58
8.10 Redundant Dedicated Hosts 58
8.11 Multiple Window Id's Over the Network 58
8.12 gcm12 File 59

APPENDICES

APPENDIX A WORKSTATION LOCAL SHARED MEMORY
APPENDIX B DEDICATED HOST SHARED MEMORY
APPENDIX C I/O REQUEST TYPES
APPENDIX D RPC REQUEST CODES
APPENDIX E RPC DATA STRUCTURE FOR RETRIEVE TV GUIDE
APPENDIX F RPC DATA STRUCTURE FOR DISTRIBUTION AUTHORIZATION
APPENDIX G RPC DATA STRUCTURE FOR RECEPTION AUTHORIZATION
APPENDIX H RPC DATA STRUCTURE FOR CANCEL DISTRIBUTION
APPENDIX I RPC DATA STRUCTURE FOR CANCEL RECEPTION
APPENDIX J CHANNEL MAP AND STATION STRUCTURES
APPENDIX K SERVER LISTINGS
APPENDIX L PROTOCOL DISTRIBUTOR LISTINGS
APPENDIX M PROTOCOL RECEIVER LISTINGS
APPENDIX N LOCAL DISTRIBUTION MANAGER LISTINGS
APPENDIX O CENTRAL DISTRIBUTION MANAGER LISTINGS
APPENDIX P PROTOCOL MULTIPLEXER LISTINGS
APPENDIX Q ALIASES AND SCRIPT FILES
APPENDIX R RPC RELATED INCLUDE FILES
APPENDIX S DISPLAY SHARING INCLUDE FILES

LIST OF FIGURES

2.0 Phase Two Display Sharing Prototype Configuration 3
2.1 Workstation Configuration 4
2.2 Data Flow For X Protocol Distribution 5
2.3 Data Flow For Reception of X Protocol 7
2.4 Retrieve Channel Guide Request Path 8
2.5 Send Display Request Data Path 10
2.6 Receive Display Request Data Path 11
2.7 Remove Channel Request Data Path 12
2.8 Remove Receiver Request Data Path 13
3.0 The X Window Server in Unix 16
3.1 X Graphics Request Path 17
3.2 The X Window Server With Modification 18
3.3 Multicast Data Flow 20
3.4 X Protocol Buffers 21
3.5 Get Window Attributes State Information Data Flow 25
3.6 Get Graphics Context State Information Data Flow 26
3.7 Protocol Multiplexer Data Flow 28
4.1 Display Sharing Window Layout 41

ABBREVIATIONS

Bps	Bytes Per Second
bps	Bits Per Second
CDM	Central Distribution Manager
GC	Graphics Context
GCS	Graphics Context Structure
GGCS	Get Graphics Context Structure
GWATS	Get Window Atrributes Structure
LAN	Local Area Network
LDM	Local Distribution Manager
MCCU	Mission Control Center Upgrade
MOSL	Mission Operations Support Lab
OSI	Open Systems Interconnection
PD	Protocol Distributor
PM	Protocol Multiplexer
PR	Protocol Receiver
RGB	Red/Green/Blue
RTU	Real Time Unix
TCP/IP	Transmission Control Protocol/Internet Protocol
TP4	Transport Protocol, class 4
UDP	User Datagram Protocol
WATS	Window Attributes Structure
Xgcm	X Graphics Control Module
XID	X Resource Identifier

1.0 INTRODUCTION

This report for NASA Grant NAG9-370, "Research into Display Sharing Techniques for Distributed Computing Environments, Final Report," describes the X-based Display Sharing Solution recommended in the Interim Report dated September 13, 1989. During the project period covered by this report, SwRI performed the following activities:

- Identification and evaluation of X-based alternative architectures
- Selection of an X-based architecture for further study
- Development of the selected X-based alternative prototype
- Identification of typical and atypical displays subject to Display Sharing
- Development of profiles, based on X-protocol type, of typical and atypical X displays
- Evaluation of performance degradation and network load based on the prototype

The Display Sharing Prototype includes the base functionality for telecast and display copy requirements. Since the prototype implementation is modular and the system design provided flexibility for the Mission Control Center Upgrade (MCCU) operational consideration, the prototype implementation can be the baseline for a production display sharing implementation. To facilitate the process this report contains the following discussion:

- Theory of Operation
- System Architecture
- Using the Prototype
- Software Description
- Research Tools
- Prototype Evaluation
- Outstanding Issues

2.0 THEORY OF OPERATION

The prototype developed by SwRI is based on the concept of a dedicated central host performing the majority of the Display Sharing processing, allowing minimal impact on each individual workstation, as shown in Figure 2.0. Each workstation participating in Display Sharing hosts programs to facilitate the user's access to Display Sharing, as shown in Figure 2.1, as well as the sending and receiving of displays to and from the dedicated host machine. The dedicated host machine receives a display from a source workstation and multiplexes the display to all appropriate receiving workstations.

2.1 System Operation

At the workstation level, the fundamental component of the prototype is a custom enhancement to the X window server (see Section 3.1.1). This modification allows all X protocol received by the server from a client application to be input into the Display Sharing system, if requested. The workstation user must request all access to the Display Sharing system. No unsolicited displays are allowed.

Each display selected for distribution by a workstation operator is associated with a 'channel.' A workstation which is distributing a display is known as a Source Station. Potential receiving workstation operators may browse a 'channel guide' to select and request a display (channel) for reception. A workstation which is receiving a distributed display is known as a Receiving Station.

2.1.1 Displays, Windows, and Clients

Internally in the Display Sharing system, the primary differentiation between displays is the client id. This value may be obtained from any X resource identifier (XID) by shifting the number right by 24 bits. For example, a typical XID (in hexadecimal) is 0x600001. The client id is 6 (e.g., in most X servers, this number is incremented for each new client; at the time this client was started there were 6 clients including this one). The number in the lower nibble (1) indicates the resource number. The prototype uses the client number to index a number of data structures on the local workstation and at the dedicated host. It is possible that the value of an XID on one workstation will equal that of one on another workstation since XIDs are only meant to refer to local resources. The prototype does not take this into account when identifying displays from different clients on different workstations and will not properly handle the case of two workstations distributing displays (windows) with identical XIDs. The prototype can make such a distinction with modifications.

2.1.2 Transmitting Displays From a Workstation

When the user elects to send a display, as shown in Figure 2.2, the Local Distribution Manager (LDM, see Section 3.1.4) on the Source Station sends the request, including display identification obtained from the user, to

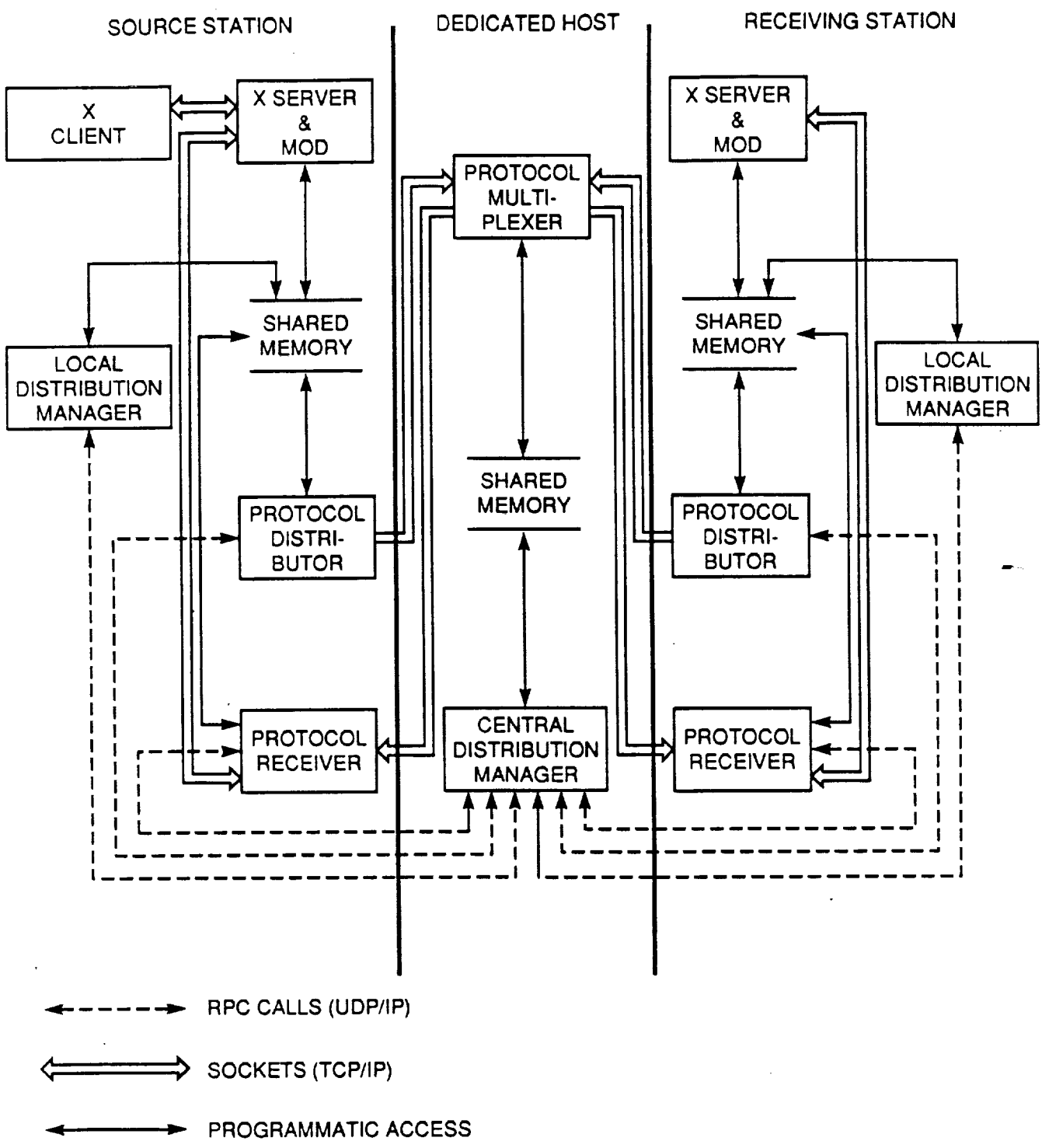

FIGURE 2.0 PHASE TWO DISPLAY SHARING PROTOTYPE CONFIGURATION

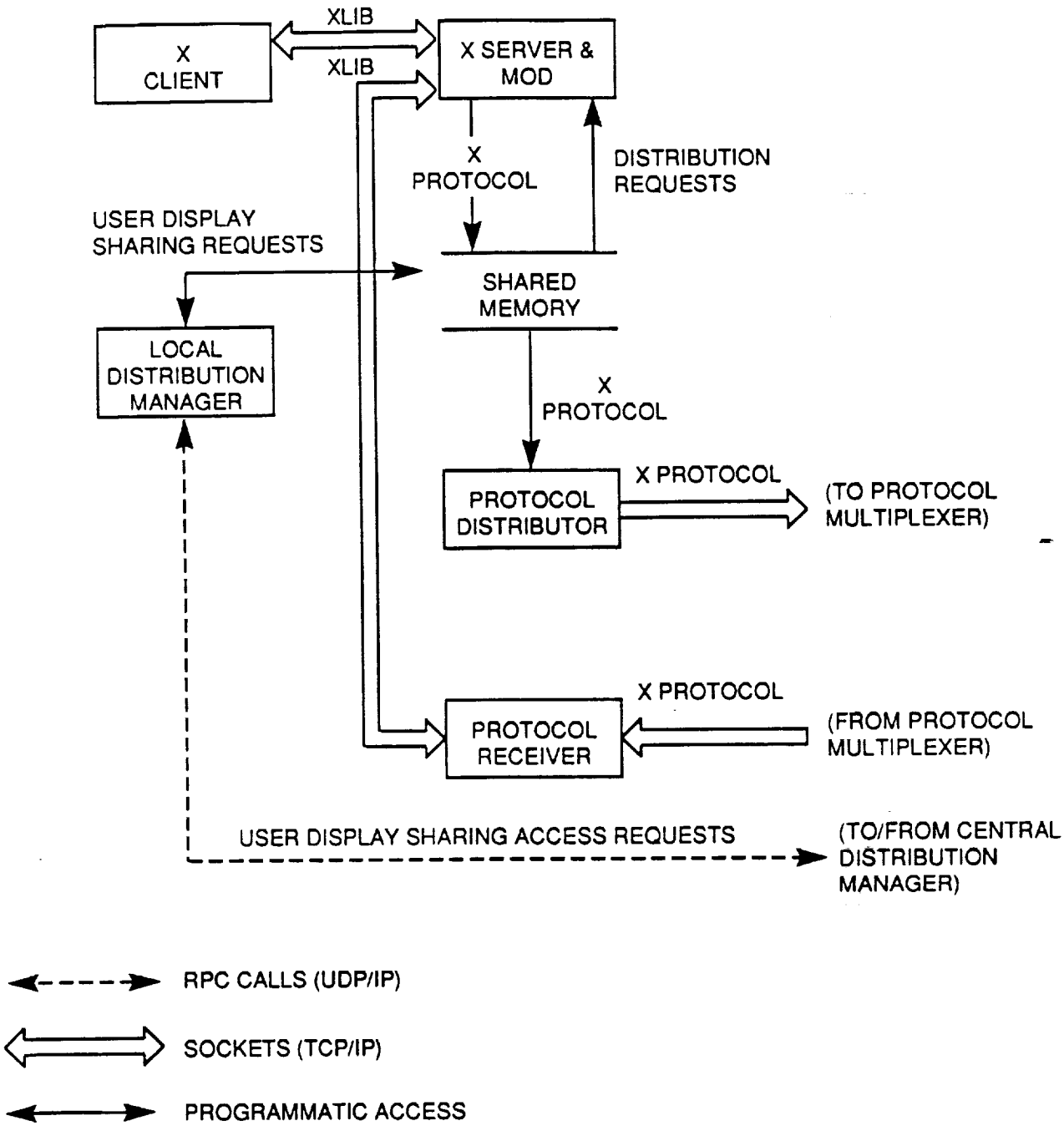

FIGURE 2.1 WORKSTATION CONFIGURATION

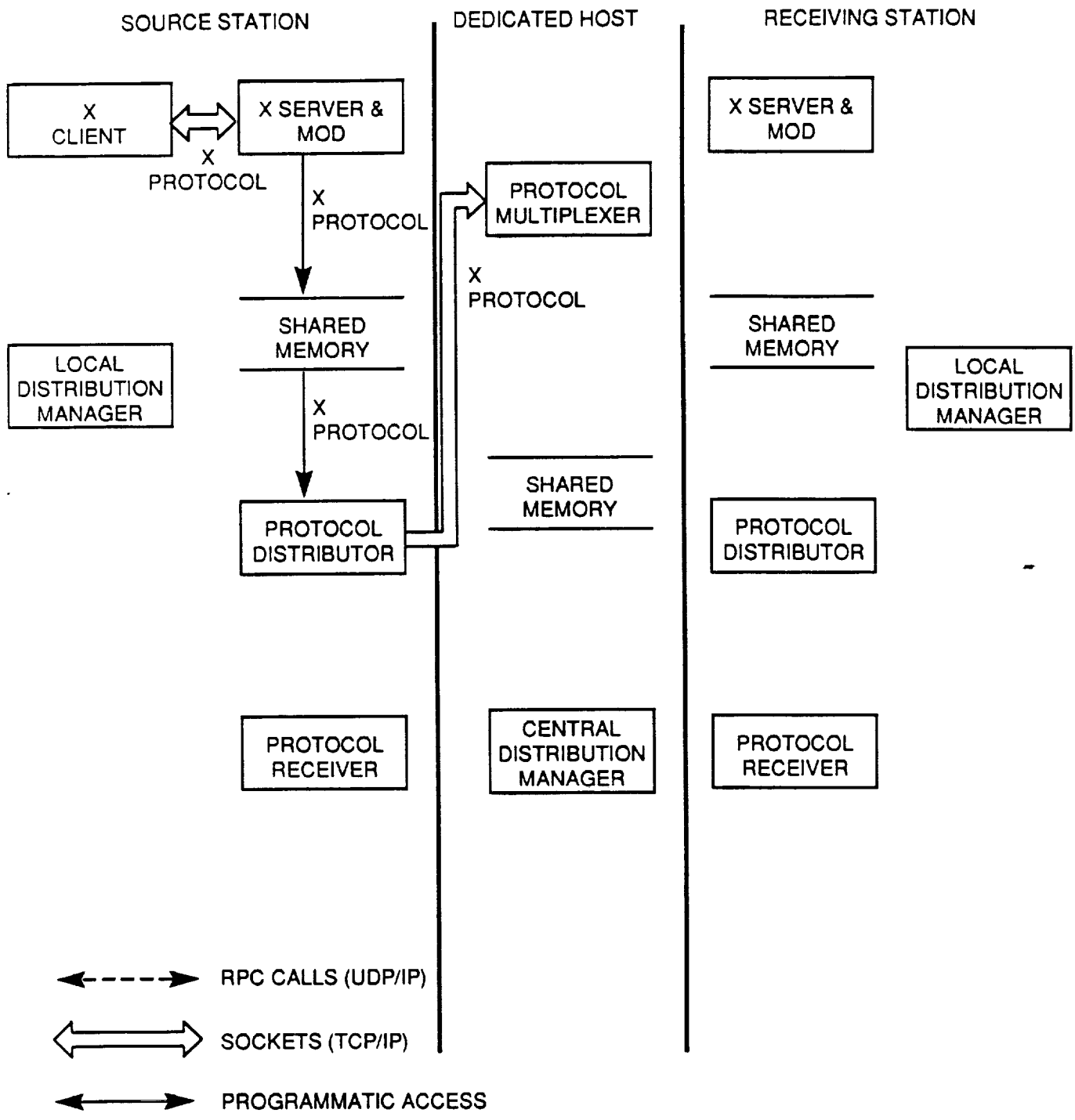

FIGURE 2.2 DATA FLOW FOR X PROTOCOL DISTRIBUTION

the Central Distribution Manager (CDM, see Section 3.2.1). The CDM mediates the request, and if approved, selects a channel to associate the display. Once the LDM has received approval for distribution the LDM notifies both the Source Station's Server Mod (see Section 3.1.1) and the Source Station's Protocol Distributor (PD, see Section 3.1.2) that this particular display is now 'wanted' for distribution.

The PD, having received notification of a new 'wanted' display for distribution, monitors a shared memory area used by all the local Display Sharing processes for communications. When X protocol from an X client, whose display is being distributed, is received by the X server modification, it is copied into that shared memory area and the PD is alerted. The PD then retrieves the X protocol from the shared memory area and sends it to the Protocol Multiplexer (PM, see Section 3.2.2).

2.1.3 Receiving Displays At a Workstation

The workstation operator may select to receive a display, as shown in Figure 2.3, from a list of channels received from the Receiving Station's LDM. The LDM sends the reception request to the CDM. The CDM mediates the request and sends the response back to the LDM for notification to the user. The CDM also requests the PM to add a new Receiving Station to the list of receivers for that particular channel.

When the PM receives X protocol from a Source Station's PD, it then sends the X protocol to every Receiving Station's PR (Protocol Receiver, PR, see Section 3.1.3) selected to receive that channel. The Receiving Station's PR then begins to receive X protocol from the PM and processes the protocol (see Section 3.1.3), and then sends the protocol to the local X server for display.

2.2 User Operation

The workstation operator has access to the Display Sharing system using a graphical user interface, accessible through a window manager selection. The user interface is driven by the local LDM, which is the initiator of all send and receive requests.

2.2.1 Retrieve Channel Guide

A retrieve channel guide request, as shown in Figure 2.4, is initiated by selecting the **Retrieve Channel Guide** option presented by the LDM (see Section 3.1.4). The LDM passes this request on to the CDM (see Section 2.1.1) who keeps the current channel guide. The CDM returns the current channel guide list to the LDM, who in turn presents this list to the workstation operator. The channel guide list is a list of active channels. The channels are represented by an alpha-numeric identification.

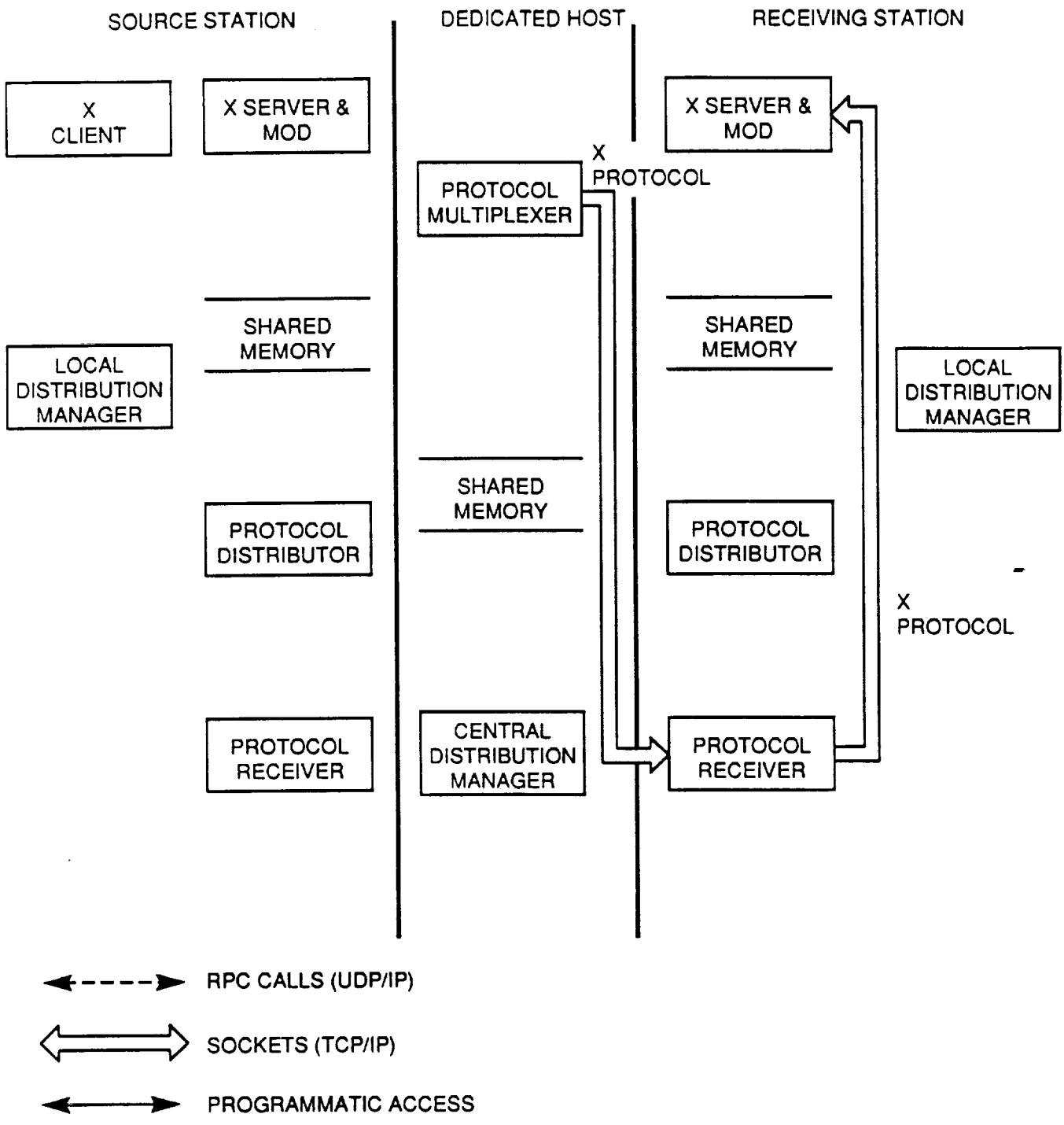

FIGURE 2.3 DATA FLOW FOR RECEPTION OF X PROTOCOL

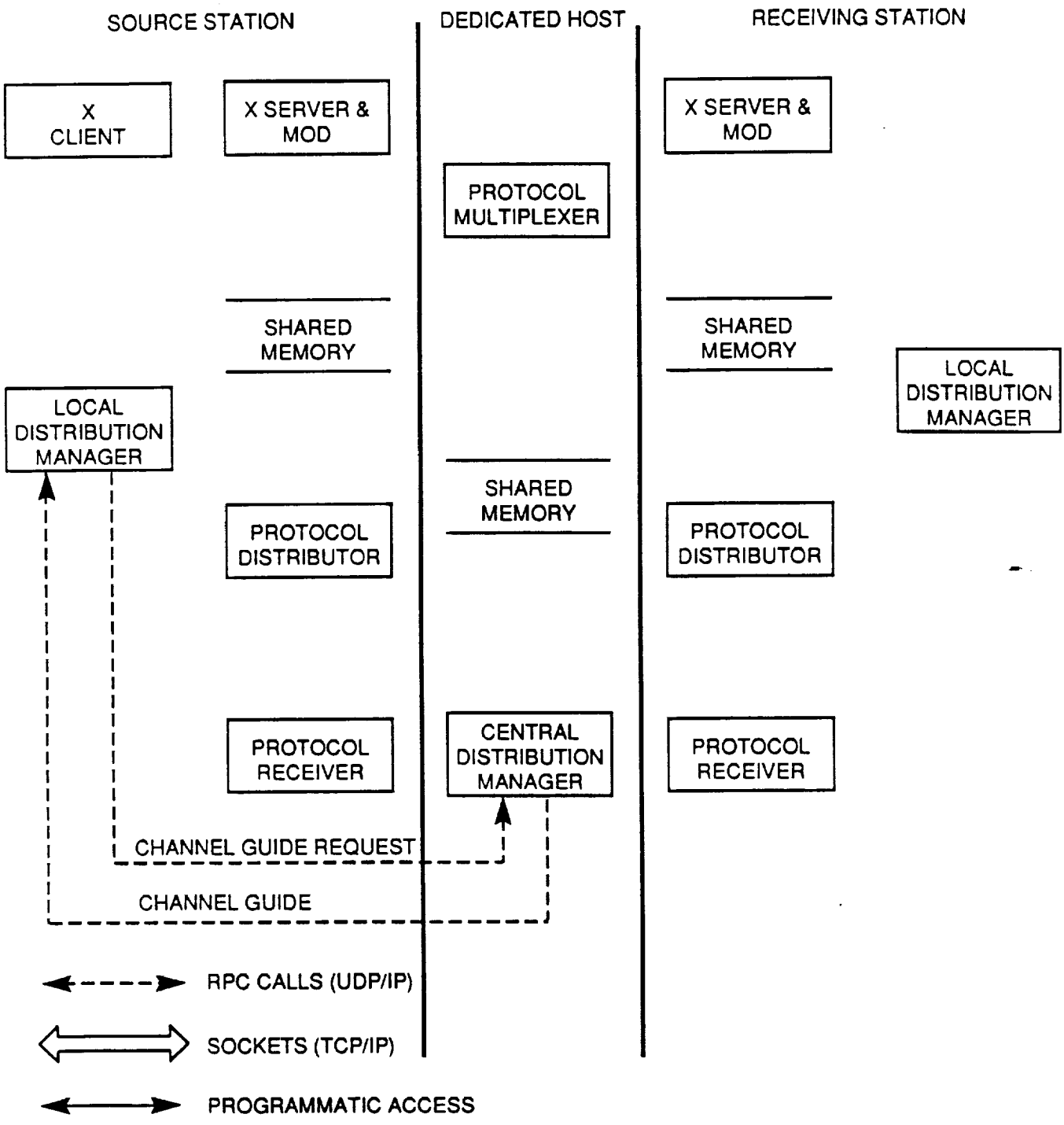

FIGURE 2.4 RETRIEVE CHANNEL GUIDE REQUEST PATH

2.2.2 Send Display Requests

A send display request, as shown in Figure 2.5, is initiated by selecting the **Distribution Authorization** option presented by the LDM (see Section 3.1.4). The operator is then prompted to use the mouse pointer to indicate which display to make available for distribution. Once the selection is made the operator is then prompted to provide an alpha-numeric identification for that display. This identification is associated with the channel selected for this display by the CDM.

When this process is complete, the LDM transmits the request to the CDM (see Section 2.1.1). If the CDM approves the request to distribute, no further operator action is required.

2.2.3 Receive Display Requests

A receive display request, as shown in Figure 2.6, is initiated by selecting the **Reception Authorization** option presented by the LDM (see Section 3.1.4). The LDM then presents the operator with a list of channels (along with an alpha-numeric identification) available for reception. The operator may select to receive one or more channels. Once the selection has been made, the LDM notifies the local PD (see Section 3.1.2) and Server Mod (see Section 3.1.1) of the channel selected. When X protocol is received for that channel by the PR (see Section 3.1.3) it is processed and passed on to the local X server. A window is then created by the PR for the requested channel's display. Note that at this point, the placement of the window on the screen is handled by the local window manager. Once the operator has placed the window on the screen, it is drawn to in the same manner as that of the original window on the Source Station.

2.2.4 Remove Channel Requests

A remove display request, as shown in Figure 2.7, is initiated by selecting the **Remove Channel** option presented by the LDM (see Section 3.1.4). This request is used to remove a display (channel) from distribution. The LDM prompts the operator to indicate which channel it wishes to stop distributing on. The LDM then notifies the local PD (see Section 3.1.2) and Server Mod (see Section 3.1.1) to no longer distribute the X protocol for the display which is associated with the selected channel. The LDM also notifies the CDM (see Section 3.2.1) that the display for that particular channel is no longer available. The CDM, in turn notifies the PM (see 3.2.2) and no further X protocol for that channel is distributed.

2.2.5 Remove Receiver Requests

A remove receiver request, as shown in Figure 2.8, is initiated by selecting the **Remove Receiver** option presented by the LDM (see Section 3.1.4). This request is used to indicate that the workstation operator no longer wishes to receive a particular channel display. The LDM prompts the operator to indicate which display (channel) to discontinue. The LDM then

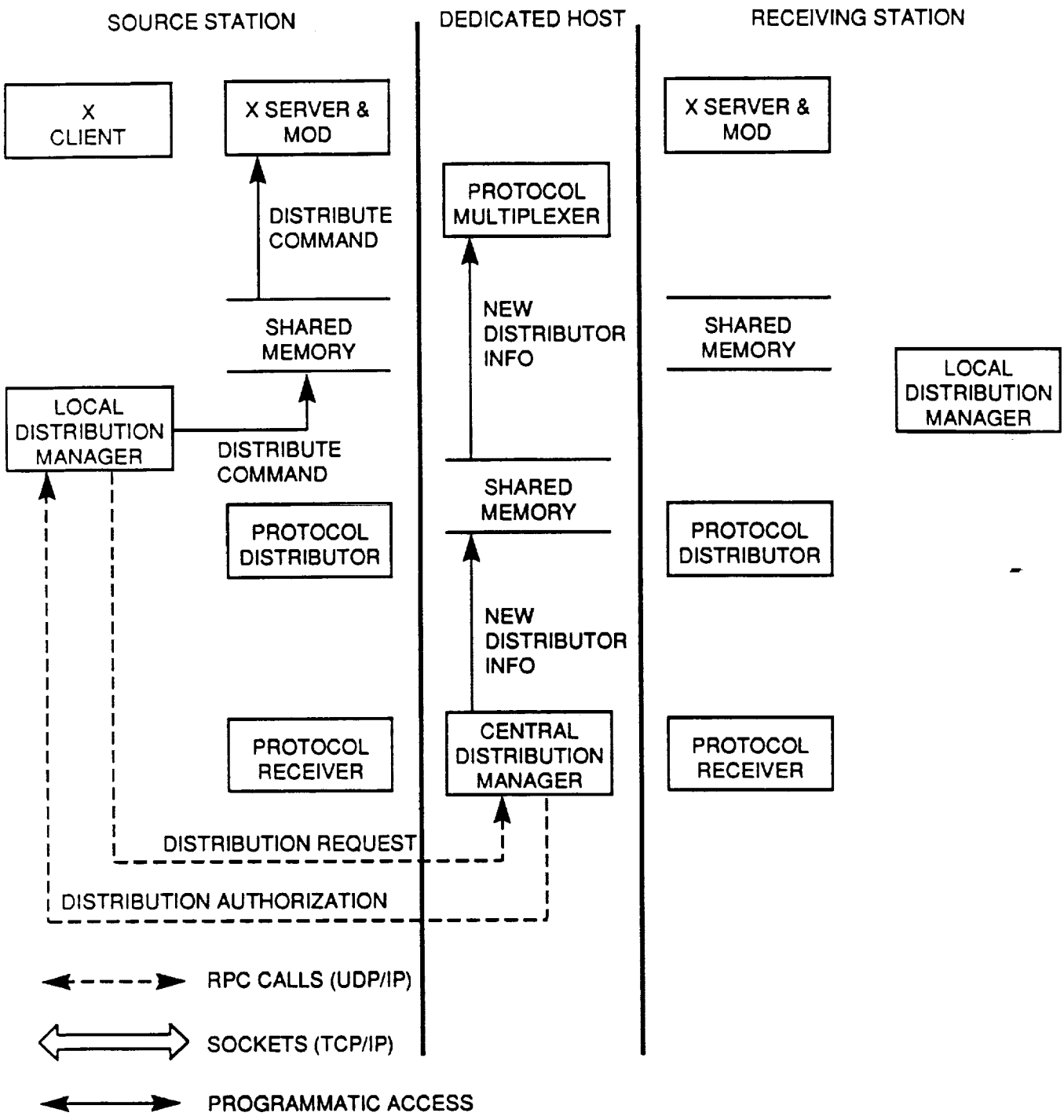

FIGURE 2.5 SEND DISPLAY REQUEST DATA PATH

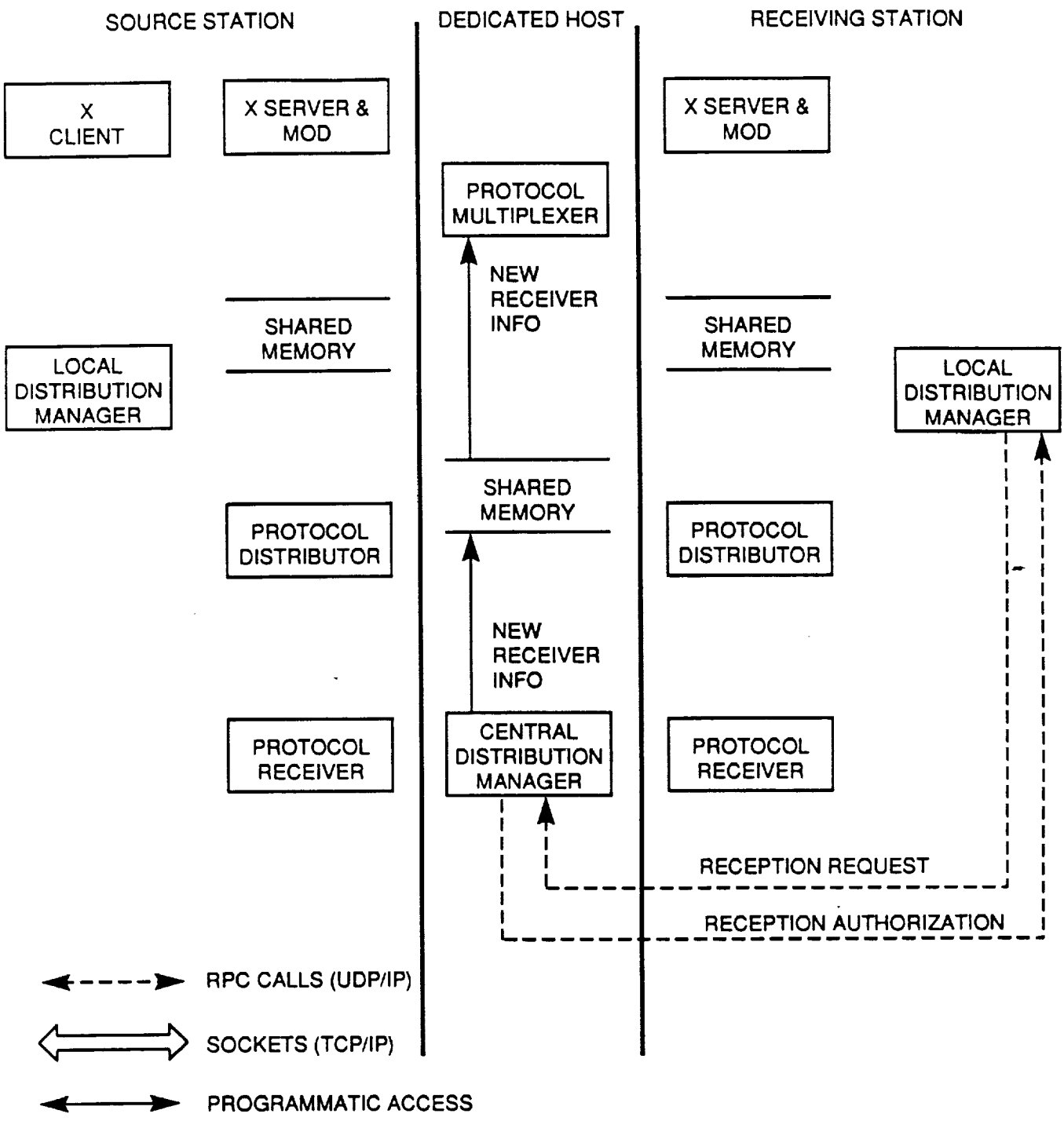

FIGURE 2.6 RECEIVE DISPLAY REQUEST DATA PATH

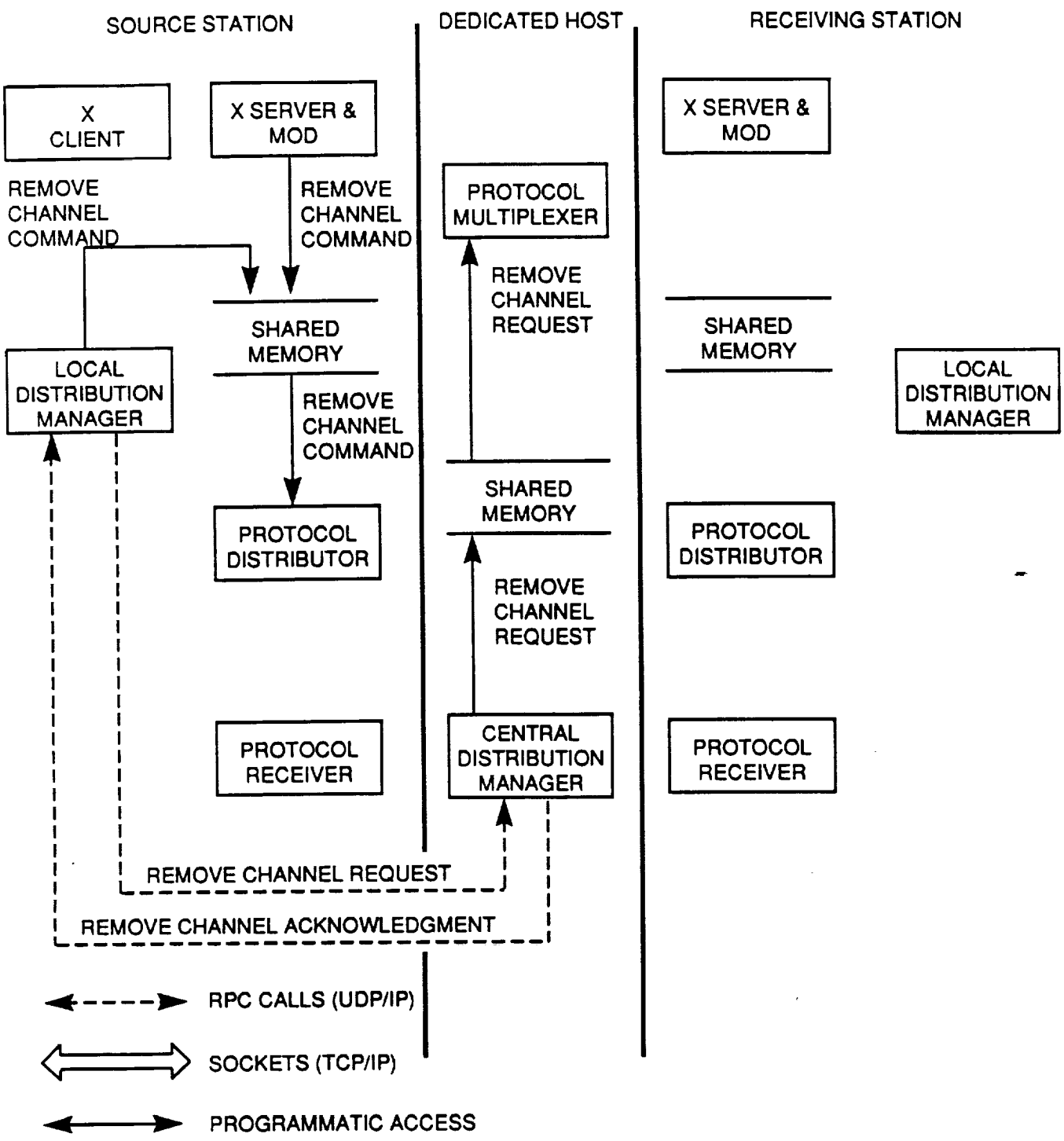

FIGURE 2.7 REMOVE CHANNEL REQUEST DATA PATH

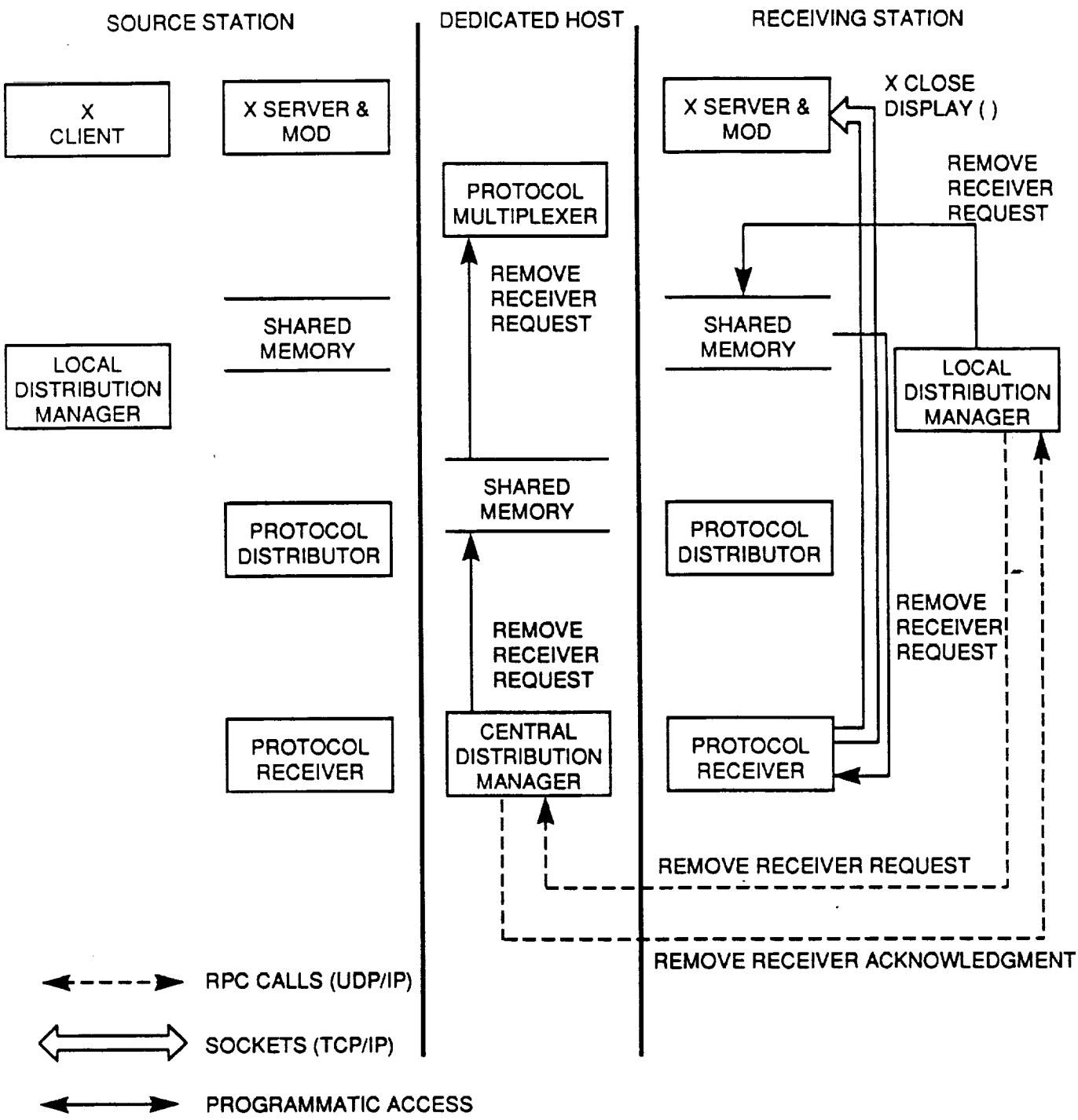

FIGURE 2.8 REMOVE RECEIVER REQUEST DATA PATH

notifies the CDM (see Section 3.2.1) to remove this particular Receiving Station from the list of Receiving Stations for the selected channel. The CDM also passes this notification on to the PM (see Section 3.2.2).

3.0 SYSTEM ARCHITECTURE

3.1 The Display Sharing Workstation

The Display Sharing Workstation, as shown in Figure 2.1, hosts three Display Sharing processes and a modified X Window Server. All of these processes work together to facilitate the distribution and reception of X protocol.

3.1.1 X Server Modification

The X Server encompasses two main divisions of labor, as shown in Figure 3.0 as follows:

- Operating System related functions and Graphics Hardware related functions. The Operating System functions handle all of the communications between client and server; for a Unix system, the base level is the reading and writing of a file descriptor. When a client connects to a server in a Unix system, it is given a data structure, called the Display structure. The display structure contains information about the server connection and the server. The structure also contains a file descriptor used to communicate with the server. As an X client makes graphics requests, Xlib turns the requests into X protocol packets and stores them in a buffer area, as shown in Figure 3.1. When the buffer area is full, it is 'flushed' to the server by writing the buffer to the Display connection file descriptor.

- On the server side, this X protocol is read from the X client's file descriptor and then acted upon by the Server. Any replies or error returns are written back to the X client using the same file descriptor. The current Display Sharing prototype uses a modification to the X server, as shown in Figure 3.2, such that the X protocol packet just read from an individual X client is stored for use by the Display Sharing system.

The modification to the server consisted of a single line of source code. The line, a C statement shown below, was added at a point in the server where the protocol packet is read from an X client:

```
(void)multicast(client,ptr,len);
```

Where:

multicast;	the name of a subroutine to be called with the following arguments:
client;	the client index whose protocol was just read. This number can be obtained by shifting any XID right 20 bits.
ptr;	a pointer to the I/O packet just read from a client.

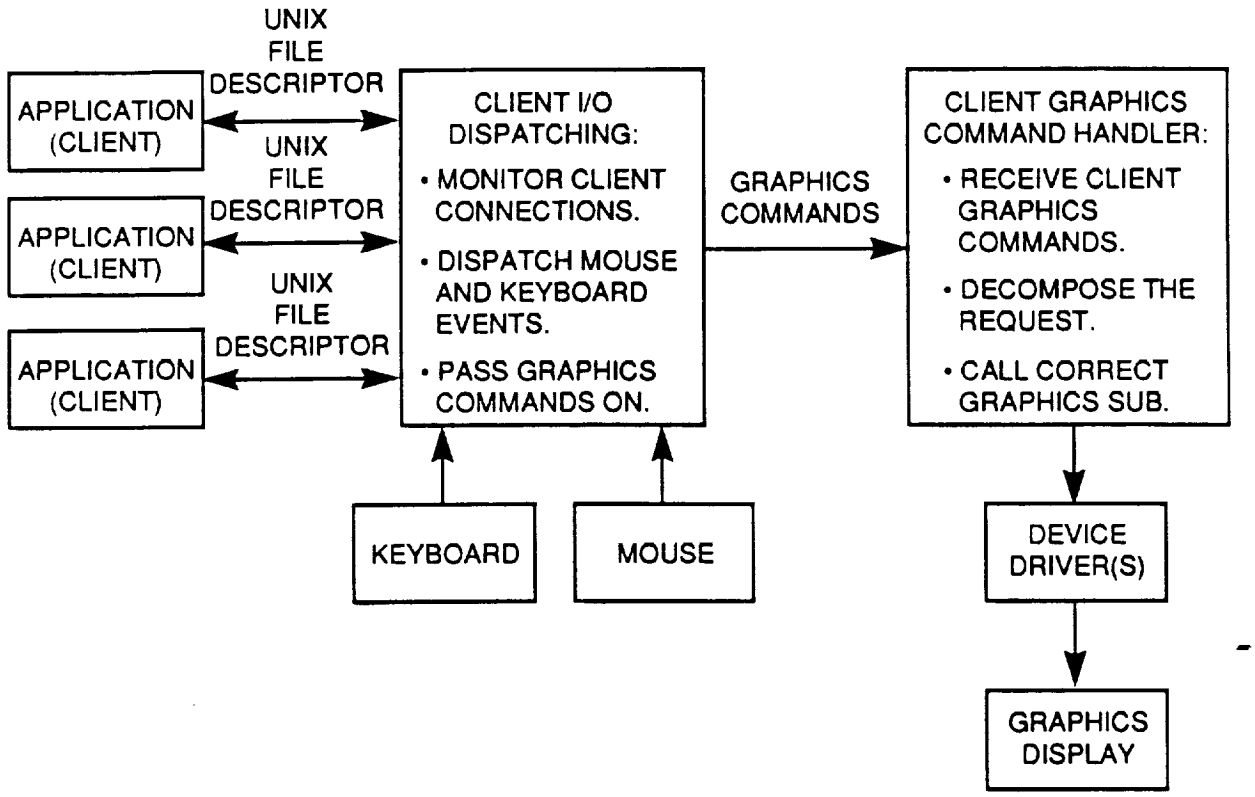

FIGURE 3.0 THE X WINDOW SERVER IN UNIX

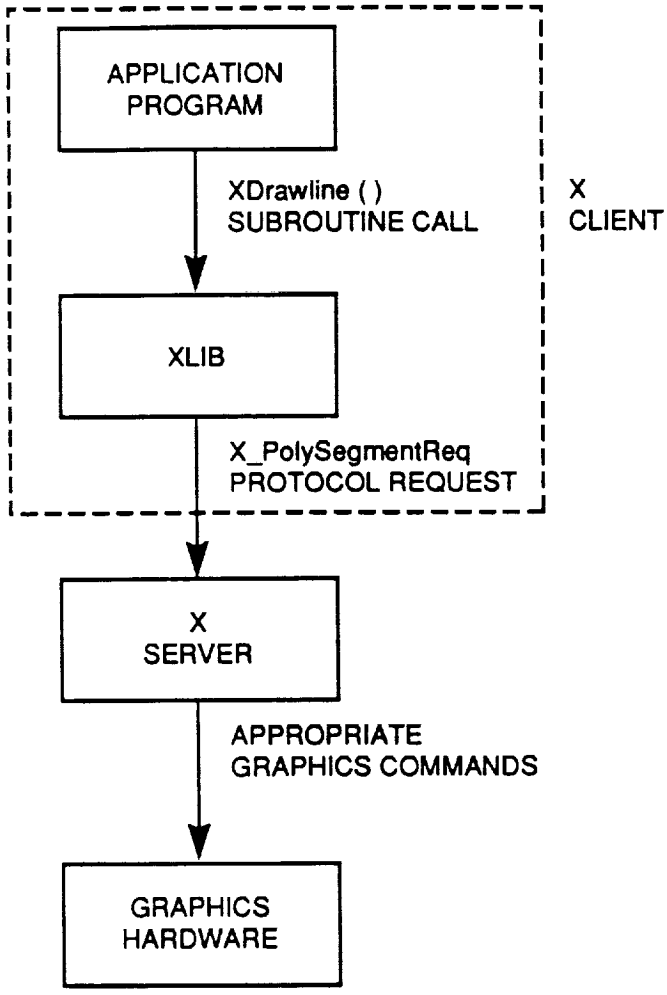

FIGURE 3.1 X GRAPHICS REQUEST PATH

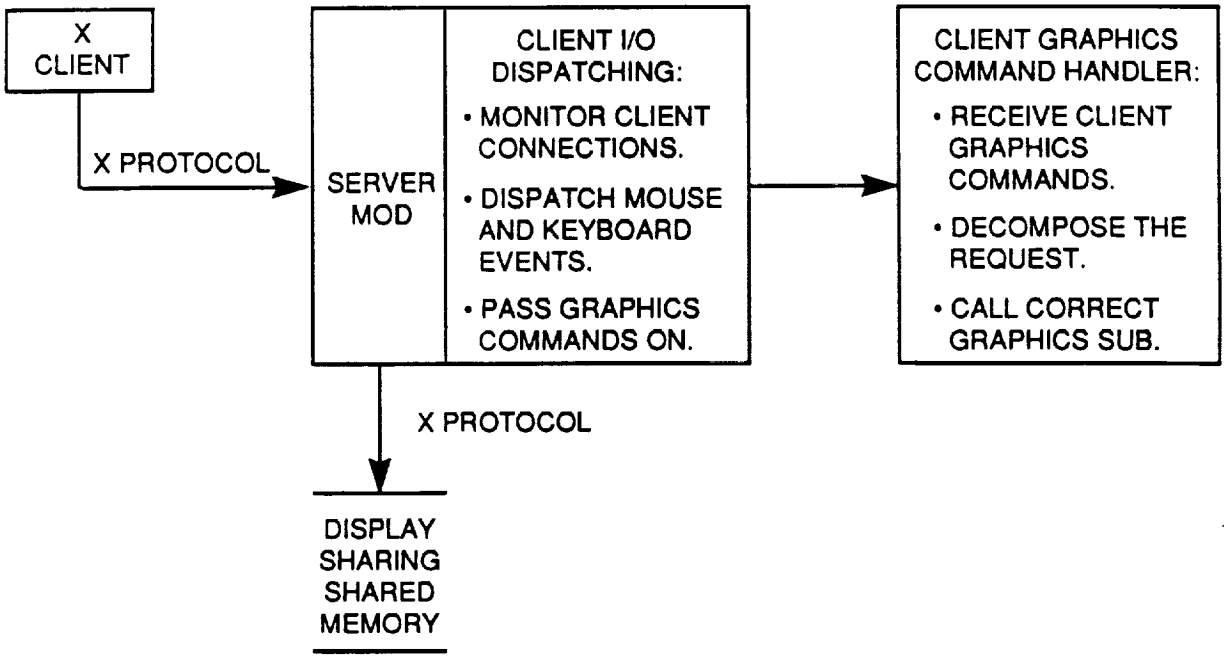

FIGURE 3.2 THE X WINDOW SERVER WITH MODIFICATION

len; - 1, if the source of the protocol was local.
- Actual number of X protocol request bytes pointed to by ptr, if the source of the protocol was from the network. (len is not used by the Display Sharing prototype).

SwRI defined the requirement, and Concurrent added the single line of code to their X server, X Graphics Control Module (**Xgcm**), and performed an incremental link, leaving one unresolved reference, namely **multicast**. SwRI provided the multicast routine. The object module **multicast.o** was then linked, by SwRI, with the load module **Xgcm** (with 1 unresolved reference). The link results in a fully linked and executable X server, **XGCM**.

The **multicast** subroutine, as shown in Figure 3.3, is called each time there is X protocol read by the server from any client. The first time the routine is called, it creates a shared memory area for use in communication with the Protocol Distributor (PD), Protocol Receiver (PR), and Local Distribution Manager (LDM). Appendix C contains a listing and explanation of the data members of this shared memory area. **Multicast** also initializes this shared memory area, creates some semaphores to control dual access to shared memory, and then continues with normal **multicast** processing.

The next action taken by **multicast** determines if the X protocol packet is an X request concerning a graphics context (see Xlib Programming Manual's 1 and 2 for detailed information on Graphics Contexts). Such a request results when the client makes an X graphics call such as **X_ChangeGC**, which is used to change various values in a particular graphics context. The graphics context contains information such as foreground and background color, or line style and width. The X implementation requires that **multicast** store the state information for future use by the Display Sharing system.

To understand this constraint consider that all X resources (i.e., colormaps, fonts, windows, graphics contexts) are referenced by a unique id number, called an XID. This XID is used by the X server to identify the resource and index into the server's local data space for state information concerning the resource. The graphics context, however, is handled slightly differently. The graphics context, or GC, contains the type of information which may change quite rapidly, depending on the application. Line style, line width, and colors are examples of these types of data. In order to reduce network traffic and improve efficiency, the graphics context state information is contained in the X client's data space, as shown in Figure 3.4, as well as the server's. This allows the X client (actually Xlib) to send only the members of the GC which have changed, not the entire GC, thereby reducing the number of bytes sent (and thus network traffic).

Currently, there is no standard way for one X client to retrieve state information for another X client's GC, even if the XID of that GC is known.

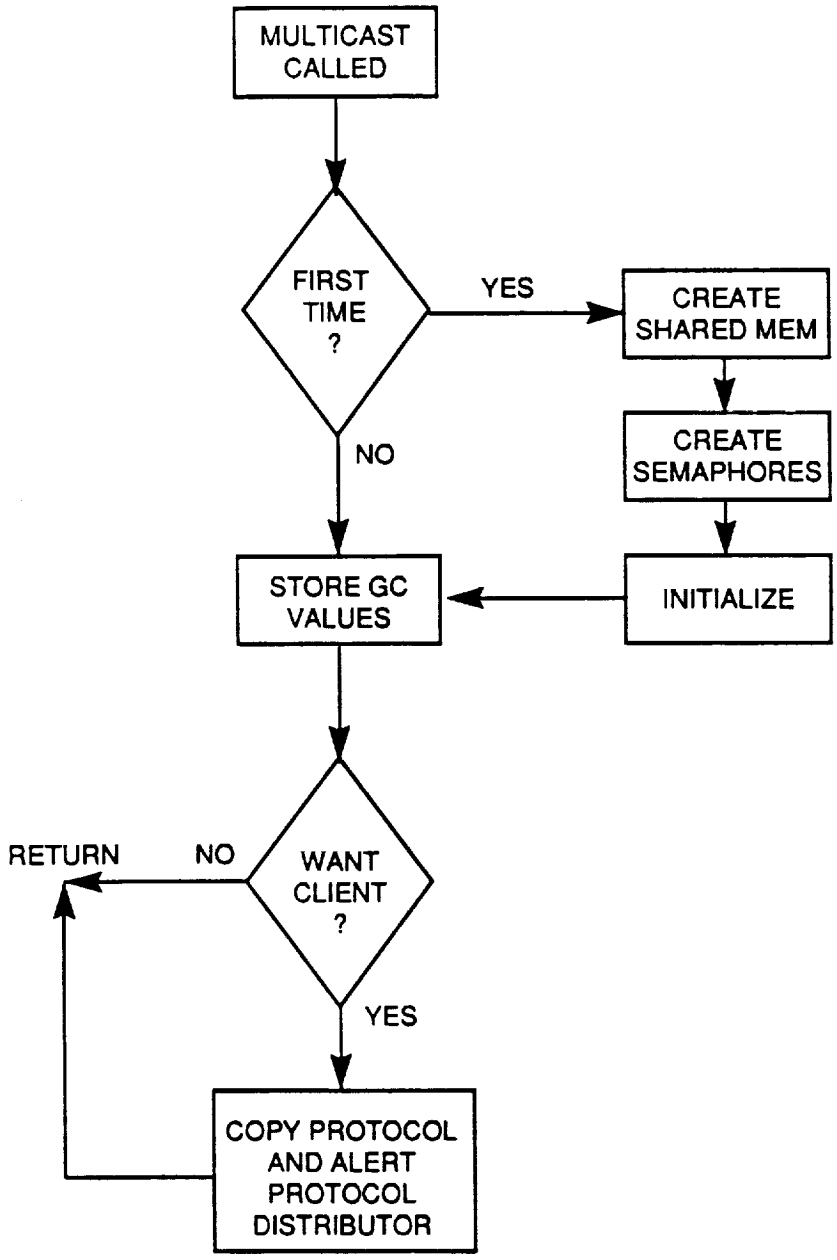

FIGURE 3.3 MULTICAST DATA FLOW

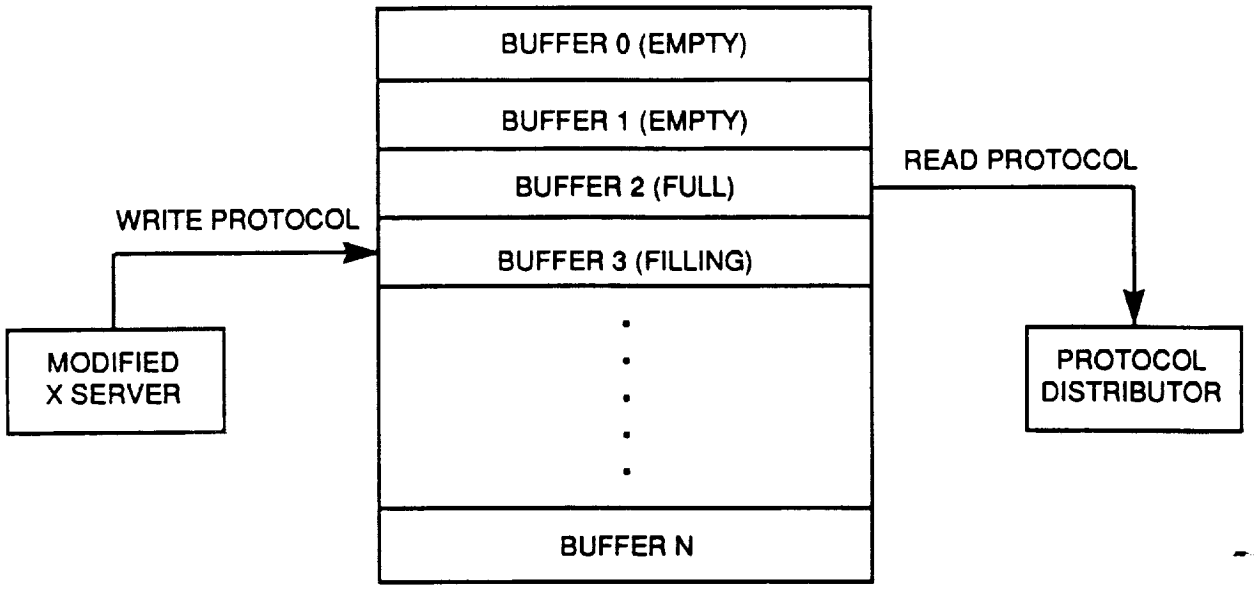

FIGURE 3.4 X PROTOCOL BUFFERS

This presents a problem when a Receiving Station's PR begins to receive X protocol from a particular Source Station. The X protocol which it receives will contain XIDs, which refer to X resources on the Source Station only and are not directly translatable into resources on the Receiving Station. This means that the PR must create a resource locally which is identical to the resource on the Source Station. For a GC, this means that state information contained in the GC must be retrieved from the Source Station and transmitted to the Receiving Station. Since there is no standard programmatic method for accessing this state information external to the client which created the GC, the state information must be stored by **multicast** on the first X protocol transfer from the client to the server.

The storage of this state information must take place even for clients which are not currently being distributed. This is necessary to maintain the currency of the state information in case that client is distributed in the future.

Determination of whether a client is distributed is based on a set of flags in the shared memory area. These flags are implemented as an array (wanted) of integers which are used to indicate when a particular client's protocol is 'wanted' for distribution. The LDM sets and clears the values. If the X protocol received is for a client which is not currently 'wanted', **multicast** simply returns and takes no further action. If the X protocol received is from a client which is 'wanted' for distribution, **multicast** copies the client buffer into shared memory and notifies the PD that there is protocol available for distribution by unlocking the semaphore associated with that buffer. **Multicast** then returns and takes no further action.

3.1.2 Protocol Distributor

The Protocol Distributor (PD) is a separate task, responsible for routing X protocol from a distributed client (X application) to the Protocol Multiplexer (PM). When **multicast** places X protocol into the shared memory buffer, PD copies the protocol out of shared memory, forms a standard Display Sharing data package, and then writes the data package to the PM. The PD also responds to requests from the local Protocol Receiver (PR), indicated through shared memory area flags, to send state information concerning window attributes or graphics contexts to the PM.

3.1.2.1 PD Initialization

Upon initialization, PD takes the following sequence of actions:

- Set up signal catching/handling routines for SIGALRM (used for timeout purposes) and SIGUSR1 (used by the PR to signal a state information request).
- Attach to the shared memory segment created by **multicast**.
- Make contact with the Central Distribution Manager (CDM) and the PM. This is done by issuing a remote procedure call (RPC)

broadcast to any existing CDM. On the basis of this broadcast, the CDM assigns the PD a unique ID number, indicating that all the necessary data structures have been set up by the CDM and the PM.

- The PD then connects to the PM by first creating a socket with the **socket()** system call, and then connecting and establishing a logical circuit to the PM by using the **connect()** system call. At this point the PD has established a logical connection path to the PM. This is the path used to transmit X protocol from the source workstation to the PM.

3.1.2.2 PD Processing

The following paragraphs outline the processing performed by PD.

3.1.2.2.1 X Protocol Transmission

In the shared memory area created by **multicast** there are multiple buffers used (rotationally) to transmit X protocol between **multicast** and PD. Access to each buffer is controlled using a separate semaphore. The PD checks each buffer sequentially, attempting to lock the semaphore associated with each buffer, as shown in Figure 3.4, and then process the X protocol contained in the buffer. PD does not proceed to the next buffer until it has successfully locked the current buffer and processed X protocol from that buffer.

The buffer which **multicast** fills for PD may contain more than one X protocol request. Once PD has locked the buffer, it must parse the buffer and determine each type of X protocol request. Not all protocol requests are passed on to the PM. Some are not valid for distribution, such as **X_ListFonts** or **X_GetImage**. These types of requests require a reply from the server and do not directly affect what is seen on the screen. Only the protocol which affects the information presented on the screen, and does not require a reply from the server, is distributed. PD decides which X protocol is valid for distribution. With each distributable protocol request, PD builds a standard data package to send to the PM. This data package contains information such as the unique client number associated with the protocol, and the total number of bytes (see Section 3.1.2.3). Once the data package is built, the semaphore associated with that buffer is unlocked, and the data package is written to the file descriptor associated with the network connection to the PD.

3.1.2.2.2 State Information Transmission and Shared Memory Requests

The PD may be requested, by the PR, to send state information (such as window attributes and graphics contexts) to the PM. The PR may do this by setting a request flag in shared memory and then sending a SIGUSR1 signal to the PD. Upon reception of the SIGUSR1 signal, the PD may perform one of the following (based on shared memory flags):

- Get Window Attributes: This request is made by the PR when state information concerning a window on another workstation is needed.

The PM gets the request, routes it to the proper workstation, and the PR/PD on that workstation sends out the state information. Eventually, that state information is received by the local PR and is used to create a local window. Figure 3.5 shows the path of a Get Window Attributes request and the actual state information through the Display Sharing system.

- Get Graphics Context: This request is made by the PR when it needs state information concerning a graphics context on another workstation. The PM gets the request, routes it to the proper workstation, and the PR/PD on that workstation sends out the state information. Eventually, that state information is received by the local PR and is used to create a local graphics context. Figure 3.6 shows the path of a Get Graphics Context request and the actual state information through the Display Sharing system.

- Send Window Attributes: This request is made by the PR when it has received a request from the PM to send a particular window's current window attributes. The PD copies the window attributes out of shared memory (having been placed there by the PR) and sends these attributes to the PM, who in turn routes it to the requesting PD/PR.

- Send Graphics Context: This request is made by the PR when it has received a request from the PM to send a particular graphics context's current state information. The PD copies the graphics context out of shared memory (having been constantly updated and replaced by multicast) and sends this information to the PM, who in turn routes it to the requesting PD/PR.

- Send Expose Event: This request is made by the PR when it has created a window (based on in-coming X protocol) and needs to receive an update on all the windows from the distributor of that window. The request is sent on to the PM, who determines the source station for the particular window, and the PM sends the request to the PR on that workstation. The PR there causes an expose event to occur for the window being distributed. Assuming that expose events are handled properly by the client displaying the window, the result is a stream of X protocol visually describing those parts of the window that do not normally get updated (labels, borders, static data).

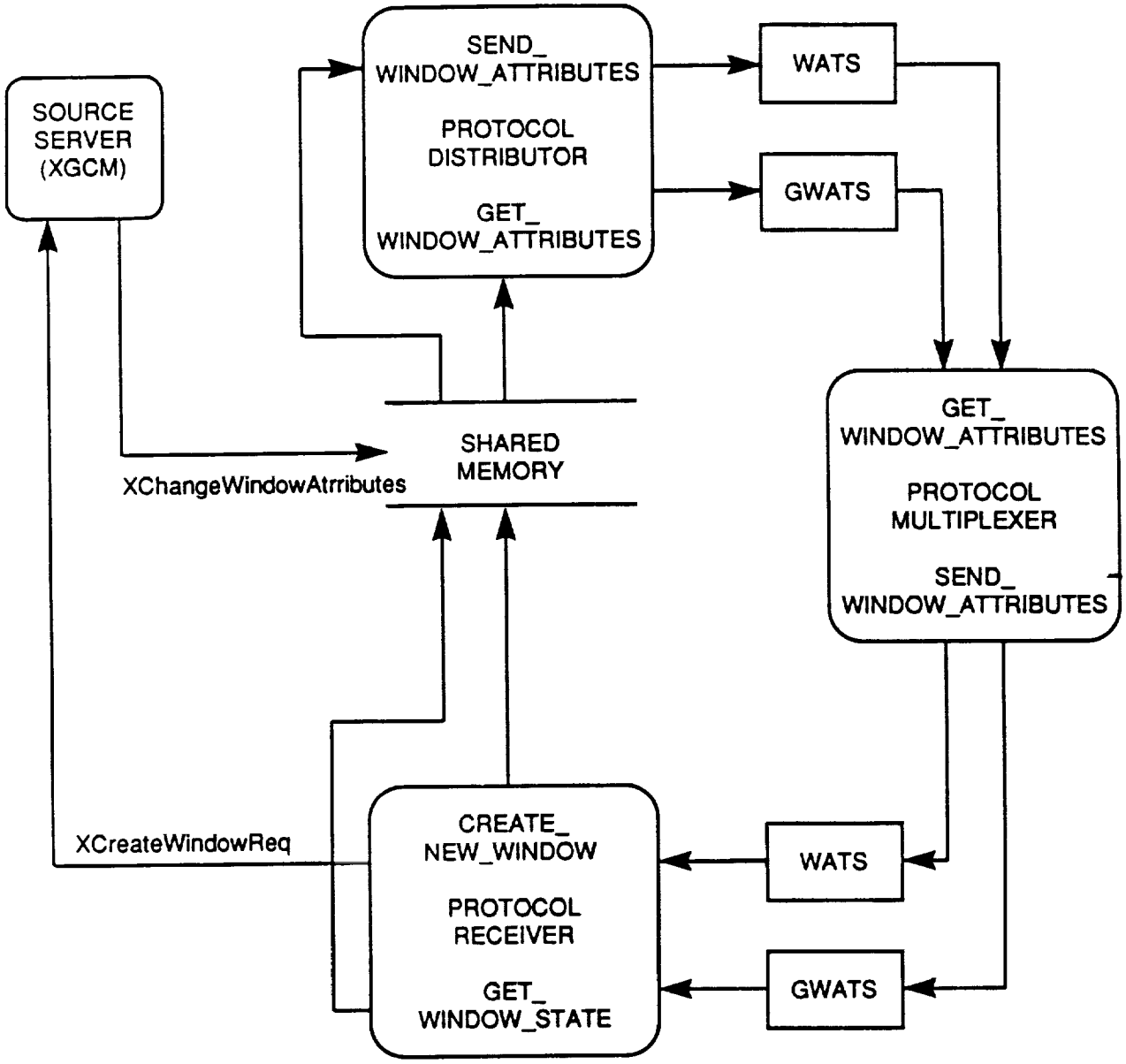

FIGURE 3.5 GET WINDOW ATTRIBUTES STATE INFORMATION DATA FLOW

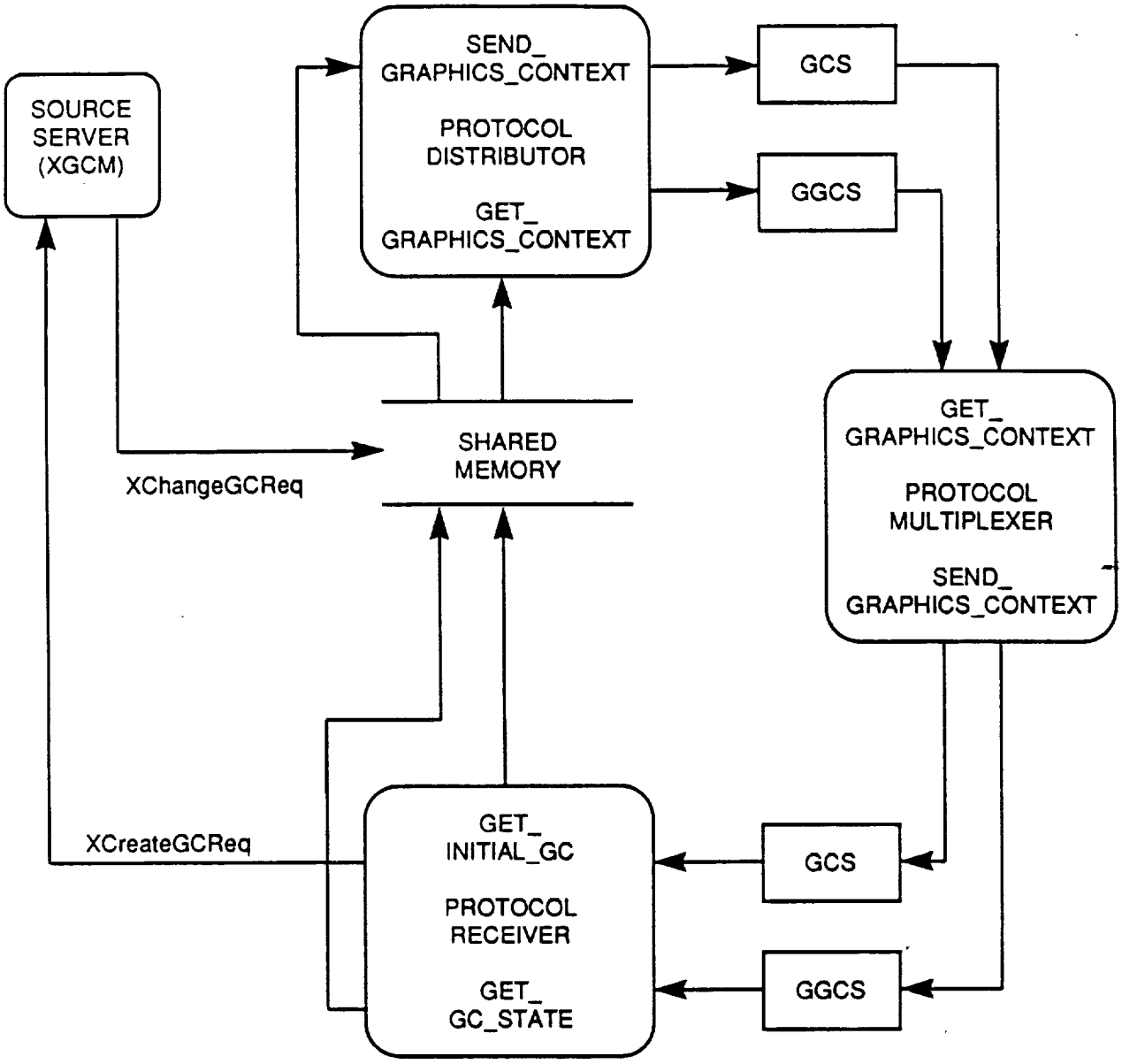

FIGURE 3.6 GET GRAPHICS CONTEXT STATE INFORMATION DATA FLOW

3.1.2.3 Protocol Distributor I/O Packet Structure

The Protocol Distributor (PD) I/O packet is a structure with the following fields:

signal : Array of unsigned characters describing the I/O request type

HDR : A structure consisting of the following fields:

- length : Integer containing the total length of the I/O packet
- client : Integer containing the source client id

buffer : Array of unsigned characters containing the X protocol packet

The above structure can also be referenced as an array of unsigned characters of the total combined length of the above listed fields.

3.1.3 Protocol Receiver

The Protocol Receiver (PR) is a separate task, responsible for routing X protocol received from the Protocol Multiplexer (PM) to the local X server. It does this by opening a standard connection to the local server with the **X_OpenDisplay** routine. This routine returns a pointer to a **Display** structure used to store information concerning this particular client and server relationship (Figure 3.7). Included in this information is a buffer which is used to hold X protocol requests from the client (in this case PR) to the server. Periodically this buffer is 'flushed' or written to the server. The server receives the request and acts upon them.

Another function of PR is to receive requests for state information from other PD/PR pairs on remote workstations. For this function, PR maintains a separate server connection. This connection is used to query the server for such state information, or in the case of an expose event, to cause a local expose event from the **X_SendEvent** call (see Section 3.1.3.2, PR Processing).

3.1.3.1 PR Initialization

Upon initialization, PR takes the following sequence of actions:

- Set up signal handling/catching routine for the SIGALRM signal. This signal is used to implement I/O timeouts.
- Open a connection, with the **X_OpenDisplay** call, to the local X server. This connection is used to retrieve state information concerning various X resources when requested by a PR/PD pair on a remote workstation.
- Attach to the shared memory segment which **multicast** has created.

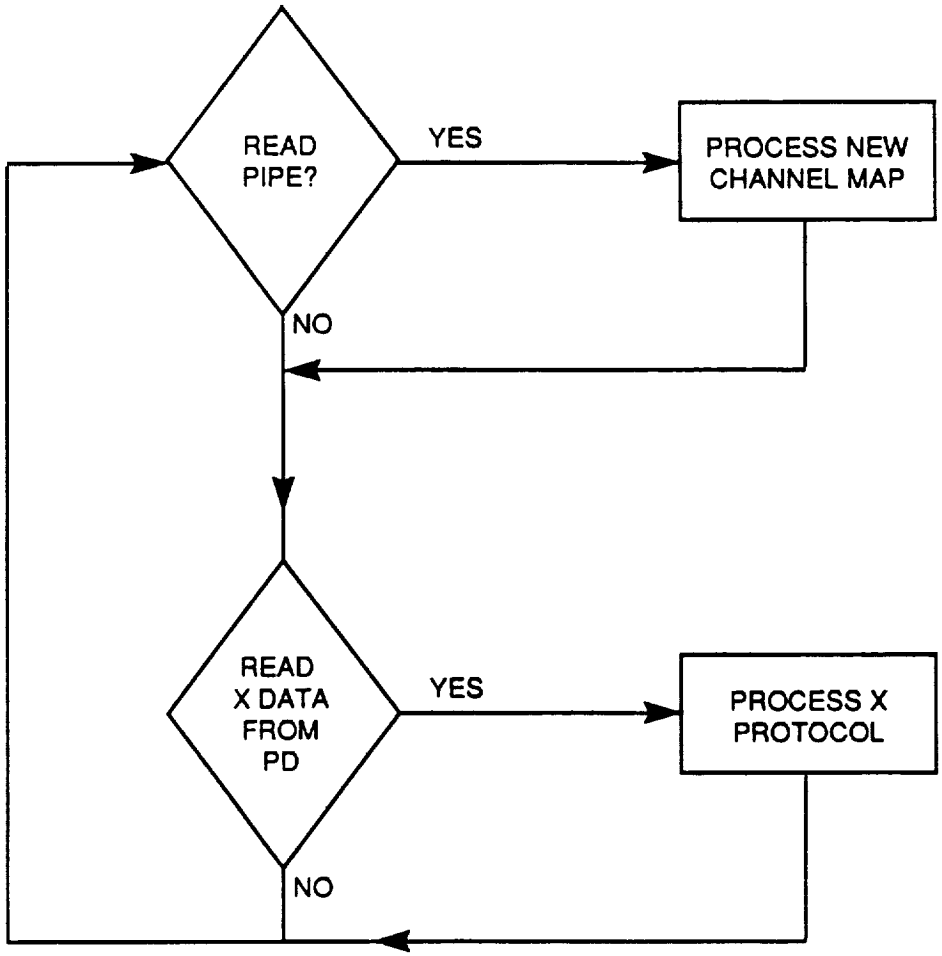

FIGURE 3.7 PROTOCOL MULTIPLEXER DATA FLOW

- Make contact with the Central Distribution Manager (CDM), and the PM. This is done by issuing an RPC broadcast to any existing CDM. On the basis of this broadcast, the CDM assigns the PR a unique ID number (the same as the local PD, see Section 3.1.2.1, PD Initialization).

- The PR then connects to the PM by first creating a socket with the **socket** system call, and then using the **listen** and **accept** system calls to accept a connection by the PM. At this point the PM has established a logical connection path to the PR. This path is used to transmit X protocol from the PM to the target or destination workstation.

3.1.3.2 PR Processing

The following paragraphs outline the processing performed by PR.

3.1.3.2.1 X Protocol Reception

Once initialization has taken place and the PM has established a network connection to PR, PR will wait until it receives a communications packet from PM. Appendix C contains a list of the types of communications packets which are sent and received by PM, and also denotes the packets received by PR.

When an **X_DATA** packet is received, the following processing occurs:

1) The protocol is first checked to determine if it is the first X protocol packet from a client (indicated by a client number in the communications packet, see Section 3.1.2.3, Protocol Distributor I/O Packet Structure). When an X protocol packet is received from a 'new' client, PR creates a new connection to the local X server for this client with the **X_OpenDisplay** system call. This connection will later be used to transmit the X protocol to the server.

2) The X protocol packet is then decomposed to determine the type of X request being made, such as **X_ChangeGC** or **X_ClearArea**. Each protocol packet will be a request type indicating some action to be directed regarding a resource (window, gc, pixmap, font etc. - see the X Protocol Reference Manual 0 for details of resources). For example, the **X_ChangeGC** request is used to change certain characteristics of a particular GC, such as foreground or background color for drawing. As well as containing the information for the foreground or background color, the packet also contains the XID of the GC to be affected.

3) All XIDs contained in the protocol packet are exchanged, or mapped, into XIDs which refer to local X resources. The XID contained in a received **X_ChangeGC**, for example, refers to a resource which exists only on the source workstation and has no meaning at the receiving

workstation. In the case of the **X_ChangeGC** request, PR performs the following:

- o If the XID is a 'new' XID, meaning this is the first X protocol packet for the XID received by PR, then PR must create a resource on the local workstation which is identical to that of the resource on the source workstation. In the case of the **X_ChangeGC** request, PR must create a GC with the same characteristics as the GC on the source station to which the XID refers. PR does this through a request for state information concerning the XID. When the state information is received, PR creates an identical GC with the **X_CreateGC** call. The source XID and the newly created destination XID are then added to a linked list for future reference.

- o If the XID is not 'new,' then the XID is used to access the linked list of source and destination XIDs to retrieve the corresponding destination XID. This XID is then substituted into the X protocol packet.

The XID substitution is performed for all XIDs contained in the X protocol packet. At this point the X protocol packet contains only XIDs which refer to locally created resources.

4) The X protocol packet is then copied into the request buffer of the associated client's **Display** structure. When this buffer is full, it is sent (written to the **Display** structure's file descriptor) to the server. This is the same method used in standard Xlib functions.

3.1.3.2.2 State Information Transmission and Shared Memory Requests

The PR may receive requests, from PD/PR pairs on remote Stations, for state information concerning local resources.

- o Get Window Attributes: The PR receives this request as a result of a remote Station's PD/PR pair requesting state information concerning a particular window. The request contains the XID of the window which refers to a resource on the receiving Station. PR receives the request and retrieves the current window attributes using the **X_GetWindowAttributes** call. PR then copies the window attributes into shared memory and makes a Send Window Attributes request of the local PD.

- o Get Graphics Context: The PR receives this request as a result of a remote Station's PD/PR pair requesting state information concerning a particular graphics context, or gc. The request contains the XID of the gc which refers to a resource on the receiving Station. PR receives the request and makes a Send Graphics Context request of the local PD. The current graphics context state information has been maintained by the **multicast** routine (see Section 3.1.1, X Server Modification). After PR makes this request of the PD, the PD retrieves the GC state

information from shared memory and sends it to the PM for ultimate routing to the requestor station.

- Expose Event: A PR receives this request from a remote station's PR, who has received an Expose event from the local server. This will result from initial creation of a Display Shared window, but could also result from the user on the remote Station un-occluding the window, or de-iconifying the window. In all cases, the PR causes a local expose event by using the **X_SendEvent** call, specifying to the client to redraw the entire window.

3.1.3.2.3 Local Expose Event Solicitation

The PR solicits only one type of XEvent from the local server: Expose events. Periodically, PR checks for reception of this type of event. When an Expose event is received, the PR makes a Send Expose Event request of the local PD. The PD, in turn, sends this request to the PM for ultimate routing back to the source Station.

3.1.3.3 Protocol Receiver I/O Packet Structure

See Section 3.1.2.3, Protocol Distributor I/O Structure.

3.1.4 Local Distribution Manager

The Local Distribution Manager (LDM) provides a graphical user interface into the Display Sharing system. The LDM may be called up by a selection from the local window manager's menu. When selected, it presents the user with the following options:

- Retrieve TV Guide
- Distribution Authorization Request
- Reception Authorization Request
- Cancel Distribution on Channel
- Cancel Reception on Channel
- Stop Central Distribution Manager
- Quit

3.1.4.1 LDM Initialization

Upon initialization, LDM takes the following sequence of actions:

- Contact the Central Distribution Manager (CDM) by issuing an RPC broadcast and waiting for the response. The response includes the host name where the CDM resides.

- Attach to the shared memory area created by **multicast**. This allows access to the global variables and permits LDM to set and clear the 'wanted' flags.

- Create and display the user menu.

3.1.4.2 LDM Processing

The following paragraphs outline the processing performed by LDM.

3.1.4.2.1 Retrieve TV Guide

This option allows the user to request, from the CDM, the most current list of channel's and what is being distributed on each one. The LDM makes an RPC request, CDM_GET_LIST, as provided in Appendix D and Appendix E, of the CDM using the **callrpc** system call. The CDM responds to this request by issuing a reply containing the current TV Guide. The current TV Guide is a list of each channel, with the associated alpha-numeric identification string.

3.1.4.2.2 Distribution Authorization Request

With this option, the user may make a local display available for reception throughout the Display Sharing system. When this option is selected, the following actions take place:

- The user is prompted to 'pick' the window for distribution by moving the mouse cursor over the window to be distributed and clicking the left mouse button. LDM will then determine the window of the pointer and retrieve the XID for that window from the local server.

- LDM places the XID of the window to be distributed into the shared memory area.

- LDM prompts the user to enter an alpha-numeric identification for the distributed window. This identification will be the entry in the TV Guide for the channel on which this display is distributed.

- LDM makes an RPC request of the CDM, CDM_DIST_REQ, as provided in Appendix D and Appendix F. This requests authorization to distribute a particular display on a channel. The CDM's reply returns the channel assigned for distribution.

- LDM sets the client and window id in shared memory, and then sets a flag to indicate to the PD that a new client/window is ready for distribution.

3.1.4.2.3 Reception Authorization Request

With this option, the user may elect to receive a display currently being distributed on a channel. When this option is selected, LDM performs the following actions:

- LDM requests from the CDM (using the CDM_GET_LIST RPC request) the most recent copy of the TV Guide and displays the guide to the user.

- The user is prompted to select, using the mouse, one of the active channels to be received.

- LDM makes an RPC request of the CDM, CDM_RECV_REQ, as provided in Appendix D and Appendix G. This request is for authorization to receive a particular channel.

LDM takes no further action. The local PR will immediately begin receiving X protocol for a 'new' channel.

3.1.4.2.4 Cancel Distribution on Channel

When a user no longer wishes to make a display available to the Display Sharing system, this option is selected. When this option is chosen, the LDM takes the following actions:

- Set the 'wanted' flag for the particular client in shared memory to FALSE, indicating to the local PD that this client's protocol should not be distributed.

- The LDM makes an RPC request of the CDM using CDM_REMV_CHAN, as provided in Appendix D and Appendix H. This request contains the channel number on which to halt distribution.

3.1.4.2.5 Cancel Reception on Channel

When a user no longer wishes to receive a channel, this option is selected. When this option is selected, the LDM takes the following actions:

- The LDM makes an RPC request of CDM using CDM_REMV_RECV, as provided in Appendix D and Appendix I. This causes the CDM to notify the PM to no longer distribute protocol to the local PR.

- The LDM notifies the local PR, through flags in shared memory, that no further protocol will be received for a particular channel. The PR then closes the display connection and removes the channel.

3.1.4.2.6 Stop Central Distribution Manager

When this option is selected, LDM requests CDM to stop and remove itself. The user should do this before selecting the 'quit' option on LDM.

3.1.4.2.7 Quit

When this option is selected, LDM detaches from the shared memory area and removes itself.

3.2 Dedicated Display Sharing Host

To reduce the Display Sharing workstation burden from sending and receiving shared displays, the concept of a dedicated Display Sharing host is utilized. The dedicated host's primary responsibility is to receive and retransmit X protocol between Display Sharing stations. Secondarily, the host acts as a clearing house for transmit and receive authorizations.

Hosted on the dedicated host are two Display Sharing processes, as follows:

- The Central Distribution Manager (CDM) and the Protocol Multiplexer (PM). The CDM is responsible for transmit and receive authorizations, and the PM is responsible for receiving X protocol from distributing stations and routing that protocol to receiving stations.

The CDM and PM both access the same shared memory area, as provided in Appendix B. The CDM uses a pipe between itself and PM to coordinate changes to the shared memory area. For example, when CDM needs to change the Channel Map (which is an expanded version of the TV Guide) it writes a byte to the shared pipe. PM periodically checks this pipe for input. When there is input, it pauses and lets CDM update the shared memory area and then continues.

3.2.1 Central Distribution Manager

The CDM may be thought of as the Display Sharing system's coordinator. Each PD/PR pair on each station must register with the CDM and must go through the CDM to send and receive displays. The CDM's functions are as follows:

- Distribution authorization
- Reception authorization
- TV Guide maintenance

The CDM achieves these objectives by becoming the RPC service server (see Appendix D for a description of the RPC requests that CDM processes).

3.2.1.1 CDM Initialization

Upon start-up, the CDM performs the following sequence of actions:

- Create and initialize a shared memory area for use in communication between the CDM and PM.
- Create a pipe for communication between CDM and PM.
- Spawn the Protocol Multiplexer (PM) task.

- Create a User Datagram Protocol (UDP) socket with the **socket** system call.

- Register self as an RPC service (server) with the **svc_register** system call.

- Enter RPC handling loop with the **svc_run** system call.

At this point, PM is awaiting connections from PD/PR pairs which register themselves as they come 'on-line.' The CDM is awaiting RPC requests from each station.

3.2.1.2 CDM Processing

The CDM awaits RPC requests from outside the dedicated host (PD/PR pairs). Below is a list of each type of RPC request:

CDM_GET_LIST:

This is a request from an LDM to send back the most recent TV Guide listing. The CDM uses the **svc_sendreply** call to send back an alpha-numeric string representing the current TV Guide listing.

CDM_REG_DIST:

This is a request from a PD to register itself as it comes 'on-line.' Registering a Distributor (PD) consists of initializing data structures in shared memory and assigning a Station id (which is used as an index into the data structures). This information is passed onto the PM as well.

CDM_REG_RECV:

This is a request from a PR to register itself 'on-line.' Registering a Receiver (PR) consists of initializing data structures in shared memory and assigning a Station id (the same one assigned to the PR's local PD). This information is passed onto the PM as well.

CDM_DIST_REQ:

This request is sent by an LDM which requests approval to begin distributing a display on a channel selected by CDM. The prototype implementation approves every distribution request, assuming that there is an empty channel to assign. The CDM then determines the first empty channel to assign, sets appropriate values in its data structures, and returns the approval and channel number to the requesting LDM. The CDM also notifies the PM of the change to the Channel Map.

CDM_RECV_REQ:

This request is sent by an LDM which requests approval to begin receiving a display associated with the requested channel. The prototype implementation approves every distribution request, assuming that the

maximum number of receivers per channel has not been reached. The CDM adds the station index (the number returned to the PD/PR pair when they register with the CDM) to the list of stations which are currently receiving a particular display on the requested channel. The CDM also notifies the PM of the change to the Channel Map.

CDM_REMV_CHAN:

This request is sent by an LDM which no longer wishes to distribute a display on the indicated channel. The CDM marks the shared memory data structures appropriately and notifies the PM that distribution on the indicated channel is no longer valid.

CDM_REMV_RECV:

This request is sent by an LDM which no longer wishes to receive a display from the indicated channel. The CDM removes the station from the list of receivers for the channel and notifies the PM of the change to the Channel Map.

CDM_PRESENT:

This request is used by both the PD and the PR as a broadcast request to 'find' the CDM/PM pair. The request is made by the PD or PR using the **clnt_broadcast** system call. The CDM responds by returning the hostname of the machine on which it resides.

CDM_GO_AWAY:

This request is sent by LDM to CDM to tell it to stop.

3.2.2 Protocol Multiplexer

The Protocol Multiplexer (PM) may be thought of as the I/O concentrator of the Display Sharing system. It is responsible for reading all X protocol from distributing Stations and then writing that protocol to all receiving Stations. The PM also processes all state information requests and transmittals.

The PM receives processing directions from the CDM from a shared memory area and two Unix pipes. The shared memory area (see Appendix B) is used to store the Channel Map and Station data structures (see Appendix J). These data structures contain information concerning each distributor, receiver, and Station in the Display Sharing system. These structures provide the CDM and PM the means to process X protocol. The Unix pipes are used to synchronize changes in the shared memory area between CDM and PM in leu of a semaphore.

The PM is spawned by the CDM with the following parameters:

cdm_read_fd; an integer (file descriptor) used to read information written to the pipe which the CDM uses to send information to the PM.

cdm_write_fd; an integer (file descriptor) used to write information to the pipe which the CDM monitors.

shmid; an integer containing the identification number of the shared memory area used between the CDM and PM.

3.2.2.1 PM Initialization

Upon initialization, PM performs the following sequence of tasks:

- Set up signal catching/handling routine for SIGALRM to implement and handle timeouts.
- Attach to the shared memory segment which CDM has created.
- Creates a socket with the **socket()** system call to use in accepting the first connection by the first PD/PR pair.
- Use the **bind()** system call to associate the socket just created with the PM's network address.
- Register intent to list and accept connections by using the **listen()** system call.
- Write port number to the CDM for later use.
- Use the **accept()** system call to await the first connection by a PD/PR pair who are 'registering.' Subsequent connections by PD/PR pairs are performed upon notification by the CDM using one of the Unix pipes.

3.2.2.2 PM Processing

The PM has two primary responsibilities during processing, as follows:

- Connecting to or accepting connections from PD/PR pairs, and
- Dispatching X protocol from distributors to receivers, as shown in Figure 3.7.

3.2.2.2.1 New Connections

The CDM may signal to the PM that a PD or PR requests a connection, for the purposes of transmitting or receiving X protocol. The signal occurs by setting a flag in shared memory. This flag indicates that the PM should pause from normal processing and read the pipe for information. While the

PM is paused and awaiting notification on the pipe, the CDM updates shared memory with new values and then notifies the PM that shared memory has been updated.

The PM then determines the purpose of the interruption by examining several flags in the newly updated shared memory area. If the request is to connect to a PD, then the PM makes an **accept()** system call to complete the connection (passive connection). If it is a request from a PR, then the PM makes a **socket()** and **connect()** system call to complete the connection (active connection).

Regardless of the type of connection (active or passive), identifying data concerning each connected Station is placed into shared memory for use by both the CDM and PM. This information is used, for instance, to determine the source Station when a receiving Station has requested an Expose event.

3.2.2.2.2 Protocol Dispatching

The PM performs processing in three distinct areas while dispatching X protocol:

- o Distributors Processing
- o Receivers Processing
- o State Information Processing

Distributors Processing: The PM is notified of a new distributor by a change in the Channel Map (made by the CDM after a **CDM_DIST_AUTH** request). In its processing the PM goes through the Channel Map. For each channel that is active, the Channel Map will contain a valid file descriptor (which results from a connection to the PD and is a virtual circuit between the distributing PD and the PM). If the channel is active (the Channel Map contains a valid file descriptor), the PM will attempt to read data from the file descriptor (see Appendix B). If data is available, it is handled according to the type of data it is.

Receivers Processing: When **X_DATA** is received on a channel, the Channel Map is checked to see if there are any receivers (PRs) for that channel. For every valid PR receiving the channel, the Channel Map will contain a valid file descriptor. This results from a connection to the PR and is a virtual circuit between the PM and the receiving PR. As X protocol is received, the PM goes through the list of receiving PRs and writes the data to each file descriptor.

State Information Processing: The PM is also a clearing house for state information requests and data (see Section 3.1.2.2, PD Processing). For example, when a Station requires state information concerning a graphics context (GC), the Station's PD sends the get_graphics_context_structure request (GGCS) to the PM. The PM receives the request and consults the Channel Map to determine to which Station (PR) to propagate the request. The PM then passes the request to the PR on that station. That Station

will then send out the requested state information (in this case the contents of the graphics context). The PM receives this information and routes it back to the requesting Station.

3.2.2.3 Performance and Redundancy

The current implementation of the Display Sharing prototype contains only one Protocol Multiplexer. One is an adequate number for prototype purposes, but there will certainly be an upper limit on the number of channels which a PM may handle before performance degradation occurs. Factors such as the frequency of update for each channel, and the average size of each X protocol packet for each channel will determine a practical maximum number of channels per PM. If that number is less than the maximum number of channels desired, modifications to the current prototype will allow multiple Protocol Multiplexers to reside on separate computers.

4.0 USING THE DISPLAY SHARING PROTOTYPE IN MOSL

Setting up and running the Display Sharing prototype in the Mission Operations Support Lab (MOSL) requires that the steps specified below be followed in their presented order. The user of the Display Sharing prototype is assumed to have basic knowledge of the Unix operating system.

Some Unix script files and aliases have been defined on the systems mentioned below. If other systems are used where these definitions do not exist, the equivalent commands have been included here, usually in parenthesis after the script file or alias name. For a complete listing of aliases and script files, see Appendix Q.

In the paragraphs below, "graphics head" refers to the physical display and keyboard of the specified server (indicated by the number after the colon in the name). "Window" refers to a window created by a login (rlogin) to a remote server from the local server.

Commands to be typed in by the user are in **bold** typeface.

4.1 Equipment

To run the Display Sharing prototype, the following hardware is needed:

- Source workstation
- Receiver workstation
- Dedicated host workstation

In the MOSL, Newton:1 was used as the source workstation, Triton:0 (or Triton:1) as the receiver workstation and Stegy as the dedicated host. All workstations are Masscomp 6600 systems. In the remainder of this chapter, the server number will be omitted from the workstation names, unless it is required for clarity.

Note that the receiving station's display is also used to access the other two servers through remote login windows.

4.2 Setup

To start a Display Sharing session, begin by logging in on the Triton graphics head as user:root and type:

DS (or cd /user/DS)
dsproto

Wait for the following three windows to be placed on the screen:

Left window : source window (Newton)
Upper right window : receiver window (Triton)
Lower right window : dedicated host window (Stegy)

At the prompt, enter root password in the left window (Newton/source) and in the lower right window (Stegy/dedicated host).

In the absence of the dsproto script file, three xterm windows can be created and placed on the Triton screen manually. In one of the three windows rlogin to Newton, in another to Stegy. Leave the last one as an xterm window to Triton. The Figure 4.1 shows the approximate window layout as created by the dsproto script file.

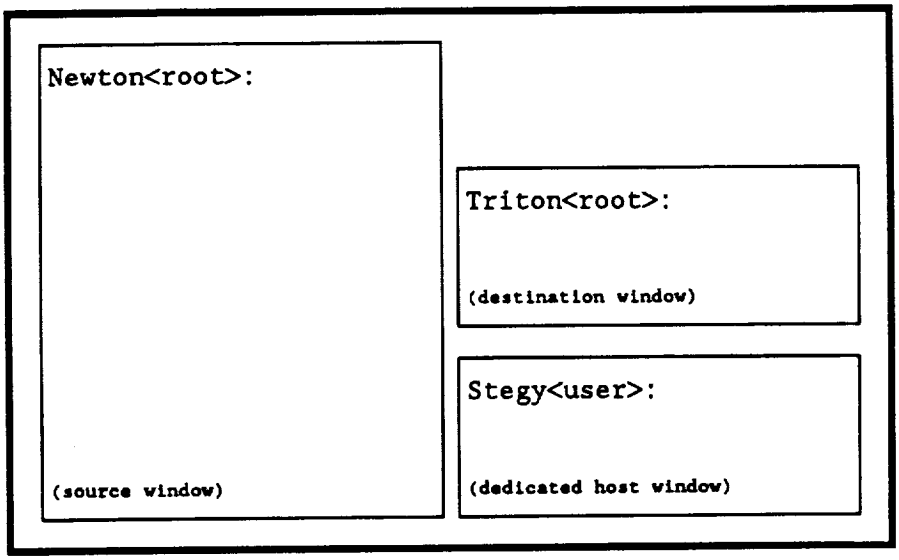

FIGURE 4.1 Display Sharing Window Layout

While working on the Triton graphics head, type the following commands

-in the left window (Newton/source):

```
mc    (or cd /user/DS/mc)
setenv DISPLAY local:1.0
```

- in upper right window (Triton/receiver):

```
mc    (or cd /user/DS/mc)
setenv DISPLAY local:1.0        (if using Triton:1 graphics head)
     OR
setenv DISPLAY local:0.0        (if using Triton:0 graphics head)
```

4.2.1 Removal of Ford Variant Server

NOTE: Do this ONLY if a Ford Variant server is currently running on the Newton (source) graphics head. If this step is needed, it must be done once a session, even if the Display Sharing prototype is stopped and restarted.

To replace the Ford Variant X file with a version of X that is compatible with Display Sharing, on the Triton graphics head, in the left window (Newton/source) type:

```
dsserv
     OR
cd /etc/mcgraphics/X11
cp gcm12.ds gcm12
mc    (or cd /user/DS/mc)
```

If this step is done, be sure to restore the Ford Variant server (see Section 4.10) before ending the Display Sharing session.

When the Display Sharing modified server is compatible with the Ford Variant Server, this step will not have to be done.

4.3 Process and Shared Memory Cleanup

In order for the Display Sharing prototype to run correctly, all shared memory, semaphores, and processes created by a previous Display Sharing session must be removed first.

On the Triton graphics head, in all three windows, check that no shared memory or semaphores are left over from a previous Display Sharing run by typing in the command:

```
ipcs
```

This will give a list of shared memory and semaphores currently being used.

If any shared memory with 0xfb or 0xfa in the key field remains, remove it with:

```
ipcrm -m <id#>      (replace <id#> with the actual number in the
                     id field)
```

If any semaphores with a zero key field remain, remove them with:

```
ipcrm -s <id#>      (replace <id#> with the actual number in the
                     id field)
```

To list processes running on each server, type

```
ps -ed
```

The actual option letters may vary on different systems. To remove a process, use

```
kill <pid#>   (or  kill -9 <pid#>)
```

where <pid#> is the process id of the process to be removed.

On the Triton graphics head, in the left window (Newton/source) remove any of the following processes still running on Newton:

- Xgcm :1 or XGCM :1 (the number after the colon is the same as the server number)
- ../sim/cmap_newton
- ../sim/sc
- pd
- pr
- ldmg

On the Triton graphics head, in the upper right window (Triton/receiver) remove any of the following processes still running on Triton:

- pd
- pr
- ldmg
- dummy

If more than one copy of each of the following processes is running, leave one running and remove the rest of them:

- ../sim/cmap_triton
- ../sim/sc

When shutting down the Display Sharing session, remove all copies of the above processes.

Verify that all shared memory, semaphores and above mentioned processes are removed after each Display Sharing prototype run.

4.4 Startup

To start a Display Sharing prototype run begin by starting the dedicated host. On the Triton graphics head, in the lower right window (Stegy/dedicated host) type:

```
DShost     (or cd /user/DS/host)
cdm
```

Next start the source server. On the Triton graphics head, in the left window (Newton/source) type:

```
start_source
```

Go to the Newton graphics head and do the following:

- Log in
- Wait for LDM (Local Distribution Manager) menu-corner to appear and place it anywhere on the screen by clicking the left mouse button
- Start an application for Display Sharing (see Section 4.9)

Finally, start the receiver. On the Triton graphics head, in the upper right window type:

start_dest

Wait for the LDM menu-corner to appear and place it anywhere on the screen by clicking the left mouse button.

NOTE: All menu items and windows are selected by moving the mouse to place the cursor over the selected key or area, and then clicking the left mouse button. The sub-menus and received windows are placed by positioning the flashing corner with the mouse and clicking the left mouse button to drop the menu or window in place.

4.5 Distributing a Window

To make a window on the source station available to the receiver(s), do the following on the Newton graphics head:

- Select "Distribution Authorization Request" from the LDM menu
- Select window to be distributed
- Select "ok" in the LDM menu, if the selected window was correct
- Type an ASCII character string in the small sub-menu window to identify the display to be distributed

Multiple clients (applications) can be distributed by repeating the above process for each client.

4.6 Receiving a Window

To receive a window currently being distributed by a source station, do the following on the Triton graphics head:

- Select "Reception Authorization Request" from the LDM menu
- Place the new sub-menu window if the menu-corner appears
- Select the channel to receive from the sub-menu
- Select "Finished" in the sub-menu to make the sub-menu disappear

To receive more than one window, continue to select other available channels from the sub-menu before removing the sub-menu.

4.7 Shutdown of Display Sharing

To perform an orderly shutdown of the Display Sharing prototype at the end of a run or session, do the following in the order indicated below:

On the Triton graphics head, in the LDM menu do the following:

- Select "Cancel Reception on Channel"
- Select each channel being received in the sub-menu
- Select "Finished" in the sub-menu to make the sub-menu disappear

Next on the Newton graphics head, in the LDM menu do the following:

- Select "Cancel Distribution on Channel"
- Select all channels being distributed in the sub-menu
- Select "Finished" in the sub-menu to make the sub-menu disappear
- Select "Stop Central Distribution Manager"
- Select "Quit"

And on the Triton graphics head, in the LDM menu do the following:

- Select "Stop Central Distribution Manager"
- Select "Quit"

Finally, perform all steps in section 4.3 (process and shared memory cleanup) to remove any remaining shared memory, semaphores or processes left after the previous run. This step is especially important if the shutdown could not be performed in the orderly manner described above.

4.8 Restarting Display Sharing

To start another run of the Display Sharing prototype, continue from section 4.4 (Startup).

If any of the three windows on the Triton graphics head have been removed, remove all remaining rlogin windows, and continue from section 4.2 (Setup). However, skip section 4.2.1 (Removal of Ford Variant Server).

4.9 Starting an Application

Most applications can be used for Display Sharing. Some sample applications exist in the following directories:

/user/DS/sim:

bar_c	(color bar chart)
pie_c	(color pie chart)
bar	(black and white bar chart)
alpha	(display with alphabetic character parameters. Number and length of parameters and update rate of display can be selected from command line. For help type: **alpha -h**)

/user/DS/nasadisp:

```
displ     (standard text display generated by NASA)
disp2     (standard graphics display generated by NASA)
```

To run one of the above applications, on the Newton (source) graphics head, do the following:

For applications in the /user/DS/sim directory type:

```
sim       (or cd /user/DS/sim)
<name of application program>
```

For applications in the /user/DS/nasadisp directory type:

```
nasa      (or cd /user/DS/nasadisp)
<name of application program>
```

When cleaning up processes from a Display Sharing run, make sure that any applications that have been started are also terminated. Remove any processes not terminated at the end of a run.

4.10 Finishing Display Sharing Session

After finishing a Display Sharing session in the MOSL, the Ford Variant server must be restored if it had been removed when staring the session. The server is restored on Newton by typing the following commands at the Triton graphics head, in the left window (Newton/source):

```
fvserv <server #>          (replace <server #> with the actual
                           server number)

     OR
cd /etc/mcgraphics/X11
cp gcm12.fv gcm12
Xgcm :<server #> &         (for Newton:1 this is Xgcm :1 &)
```

This should cause a server to start up on the Newton graphics head. If not, verify that any server running on Newton had first been removed. Now all the windows can be closed by logging out of each window.

5.0 DISPLAY SHARING SOFTWARE DESCRIPTION

The Display Sharing prototype software was designed to be modular. The communication routines and some of the common utility functions are shared by several of the functional modules.

5.1 Source and Destination Station Software

The source and destination station software is the same, the only exception being the server module in a receive-only station. In the example Display Sharing session in Chapter 4, one server (the destination) is left unmodified and therefore configured as a receive-only server. The modified server can act both as a source and a destination.

5.1.1 Server

The two versions of the server module use essentially the same software files. Options in the make-file determine which type of server is being compiled. For listings on the server files, see Appendix K.

5.1.1.1 Modified Server Version

In the modified server version, the following files are compiled and linked in with the partially linked X server object to form the executable modified server:

- o multicast.c : server modification routine called by X server for every protocol packet received
- o multix.c : X window related utility routines for the server modification

5.1.1.2 Pseudo Modified Server Version

On a receive-only station, the X server can be left unmodified. A pseudo modified server is started as a process running on the unmodified server. The pseudo modified server is used only at startup to create and initialize shared memory for communication between the PR, PD and LDM.

The Pseudo Modified Server consists of the following files:

- o multicast.c : routine called by pseudo modified server to create shared memory for PR/PD/LDM communication
- o multix.c : X window related utility routines for the pseudo modified server
- o dummy.c : pseudo modified server routine that calls multicast once

An option when compiling the pseudo modified server causes multicast.c to return right after creating the shared memory. For compatibility with the

modified server, the routines multicast.c and multix.c are exactly the same in both versions of the server.

5.1.2 Protocol Distributor (PD)

The Protocol Distributor consists of the following files:

- o pd.c : main PD routine
  - initialization
  - main loop:
    - check protocol buffer
    - if protocol to send then
      - distribute protocol
      - rotate buffer
    - endif
- o pdio.c : PD I/O related routines
- o pdutil.c : PD utility routines
- o alarm.c : set and clear alarm
- o mutil.c : utility subroutines used by local and central management functions
- o netwrite.c : network write routine

For listings on these files, see Appendix L. Files alarm.c, mutil.c and netwrite.c are shared by other modules.

5.1.3 Protocol Receiver (PR)

The Protocol Receiver consists of the following files:

- o pr.c : main PR routine
  - initialization
  - main loop:
    - wait for protocol
    - translate protocol
    - send protocol to server
- o prinit.c : PR initialization routines
- o prio.c : PR I/O related routines
- o prproto.c : PR Protocol handling routines
- o prutil.c : PR general utility routines
- o alarm.c : set and clear alarm
- o mutil.c : utility subroutines used by local and central management functions
- o netread.c : network read routine

For listings on these files, see Appendix M. Files alarm.c, mutil.c and netread.c are shared by other modules.

5.1.4 Local Distribution Manager (LDM)

The Local Distribution Manager consists of the following file:

- ldmg.c : LDM (graphics) routines

For a listing of this file, see Appendix N.

5.2 Dedicated Host Software

The dedicated host consists of a Central Distribution Manager (CDM) and a Protocol Multiplexer (PM). After creating the PM, CDM loops waiting for RPC requests. PM receives all protocol to be distributed and sends it to each station receiving on the channel that the protocol is being distributed on.

5.2.1 Central Distribution Manager (CDM)

The Central Distribution Manager consists of the following files:

- cdm.c : main CDM routine
  - initialization
  - create PM
  - main loop:
    handle incoming RPC calls
- cdm_rpc.c : CDM RPC request handling code
- mutil.c : utility subroutines used by local and central management functions

For listings on these files, see Appendix O. File mutil.c is shared by other modules.

5.2.2 Protocol Multiplexer (PM)

The Protocol Multiplexer consists of the following files:

- pm.c : main PM routine
  - initialization
  - main loop:
    channel map update
    check for protocol to distribute
- pmio.c : PM I/O related routines
- pmutil.c : PM utility routines
- alarm.c : set and clear alarm
- netwrite.c : network write routine
- netread.c : network read routine

For listings on these files, see Appendix P. Files alarm.c, netread.c and netwrite.c are shared by other modules.

6.0 DISPLAY SHARING RESEARCH TOOLS

6.1 Protocol Profiler

A program called profile was developed to enable monitoring of X protocol requests generated by X-clients. An array in shared memory is used for each client to store the following data:

- The number of times each request has occurred, and
- The number of request bytes transmitted from that client.

The byte count used to calculate throughput includes Display Sharing overhead bytes but does not account for any other overhead added by the network.

Throughput, and a breakdown of requests for each client, are displayed once a minute or by user demand. The byte counts are accumulated in two ways:

- Over the time since the last printout (one minute or less), and
- Over the total time elapsed since the array was last cleared by the user.

The user can display the contents of the array at any time. Displaying - whether by user demand or initiated by the program - will not change any of the request counts. It will clear the contents of the one minute timer and the byte count accumulated since the previous display. The running total byte count is not affected. The user can request to clear the whole array, which in addition to clearing the request counts and byte counts, also clears both the one minute timer and the running timer.

There are two versions of the profile program enabling it to be used in two different ways:

- **Stand Alone Version**: to profile requests generated by a client in a Non-Display Sharing environment

- **Display Sharing Version**: to profile requests generated by a client while it is running in a Display Sharing environment

6.2 Stand Alone Version

This version runs on a server that has been modified specifically for this purpose. Any X client(s) running on this server will be monitored, and the requests they generate are displayed by the profiler which is also running on this server. Display Sharing overhead bytes are included in the throughput byte count.

6.3 Display Sharing Version

To monitor requests and actual throughput generated by clients during Display Sharing, the profiler monitors the shared memory area updated by

the regular modified server used for Display Sharing. In addition to being able to see X requests generated during normal operation, as in the stand-alone version, requests generated by receiver's expose events are also monitored.

Throughput measured with this version will reflect a possible slowdown compared with the throughput measured with the stand alone version, if the Display Sharing prototype slows down the server.

7.0 PROTOTYPE EVALUATION

7.1 Workstation Performance

Two different approaches to Display Sharing were evaluated, the Modified Server Approach, and the Display Sharing Wedge Approach (see below).

To compare the performance of the two approaches, a test routine was developed. This evaluation program runs as a client on the workstation. The program sends repeated X requests, which require a response, to the server. It measures the time beginning when a request is sent by the client, and ending when the response is received from the server (round-trip-time).

The **X_Sync** request was selected as the X request to be sent, because it only causes the server to return a response. The client waits until a response is received before sending the next request.

Typically the client buffers several X requests, and sends them all at one time to the server. The **X_Sync** request is never buffered waiting for additional requests. It causes the request buffer to be flushed (sent to the server) immediately. This enables the timing of individual requests. In the evaluation program 50,000 such **X_Sync** requests were sent to the X window server and the elapsed time was measured and the average round-trip time computed.

The evaluation test was run on an RTU/Unix 4 system. The workstation was a Concurrent 6600 system with two 33 MHz CPU's. All tasks were operating at normal priority and the standard mix of Unix processes were running.

7.1.1 Modified X Server Approach

In the Modified X Server Approach the Source Station uses a custom enhanced X window server (see Section 3.1.1), to which the client connects in the normal manner. A modified X window server is only needed on a Source Station, while a receive-only station uses either an unmodified server or the modified server.

The Modified X Server Approach incurs a minimal performance penalty compared to a non-modified server. The additional time is spent in calling the multicast routine (see Section 3.1.1) once for every protocol request. This routine determines whether the request is to be distributed, and if it is, copies the request into a buffer in the shared memory area before returning. No additional processing is performed to non-distributable requests by the Display Sharing software.

Using the above mentioned program to evaluate the workstation performance in the Modified X Server Approach, the average round-trip time per request between the client and the X server was found to be 11 milliseconds (0.011 seconds).

7.1.2 Display Sharing Wedge Approach

The Display Sharing Wedge Approach uses a non-modified X window server. Instead, a program 'layer,' here called Wedge, is inserted between the client and the server. The client connects to Wedge the same way it would to a regular X window server. All of the Display Sharing software used with Wedge is identical to that in the Modified X Server Approach.

In the Display Sharing Wedge Approach, the Wedge program reads every request sent by the source client and then calls the multicast routine. Then Wedge writes the request out to the server. The responses from the server are read by Wedge, and then written to the client.

The Wedge is transparent to both the source client and the server because it appears to the client as a server, and to the server as a regular client.

Compared to the Modified Server Approach, Wedge must perform an extra network write for every request that the client sends to the server, as well as for every response that the server sends back to the client. In addition to the extra processing done by Wedge, it itself is a process running under Unix and taking up system time.

Using the above mentioned program to evaluate the workstation performance in the Display Sharing Wedge Approach, the average round-trip time per request between the client and the server was found to be 60 milliseconds (0.06 seconds).

Based on the above results, it can be determined that using the Display Sharing Wedge Approach is approximately 5.5 times slower than the Modified X Server Approach.

7.2 Network Performance

The following section describes the calculated amount of data sent on the local area network (LAN) for the configuration as follows:

- 50 workstations,
- 250 parameters per display,
- 10 characters per parameter,
- Entire display updated every 2 seconds.

The parameters can best be displayed with the **ImageText8** X request (see X Protocol Reference Manual 0). For this particular request, X Windows protocol adds 16 bytes of overhead to each parameter text string. An additional 0-3 bytes of padding is added to make the parameter byte length evenly divisible by 4. In the case of a 10 character parameter (one character per byte), 2 bytes of pad will be added.

The Display Sharing protocol adds 12 bytes of overhead to any X request. The total length of each packet containing one parameter is 40 bytes (16+10+2+12). Note that these and all byte counts calculated below are Transmission Control Protocol/Internet Protocol (TCP/IP) data byte counts, and do not account for bytes that will be added by Open Systems Interconnection (OSI) overhead when using Transport Protocol, class 4 (TP4).

With 250 parameters per display, the total number of bytes needed to update one screen is 250*40=10,000 bytes. If the screen is updated every 2 seconds, the network traffic between the source station and the dedicated host will be 5,000 bytes per second (Bps) or 40,000 bits per second (bps) at 8 bits/byte.

The same amount of traffic (5,000 bytes per second) will be added for every destination station receiving this display. So, in the case of 1 source station and 49 receiver stations, the total traffic is 50*5,000 = 250,000 Bps. The traffic would be the same if out of the 50 stations some were distributing and the rest were each receiving any one display.

Note that the above estimates require that the workstations, the LAN and the software can keep up with the necessary speed. In the above example the multicast routine in the Display Sharing software must be able to load the shared memory buffers at the rate of at least 5,000 Bps, and PD must be able to unload the buffers and send the data to the PM at the same rate of speed.

For the sample X window graphics display provided by NASA, the above calculations give the following results.

Each of the six sine-waves was drawn with

2 **PolySeg(8)** requests, and
1 **PolySeg(4)** request.

The number in the parenthesis indicates how many segments are drawn by each request. The number of segments that the **PolySeg** request draws determines the byte count of the request. Each segment adds 8 bytes to the basic 12 bytes of X overhead for this request. Display Sharing overhead adds 12 bytes per request for a total of 232 bytes for one pass of a sine-wave. Six sine-waves per display updated every 2 seconds, gives a LAN traffic of (232*6)/2 = 696 Bps. For 50 workstations the total traffic amounts to 34,800 Bps (assuming every station is distributing or receiving only one display).

The above method can be used to calculate the traffic generated by any size or type of display, by profiling the X requests that were used to generate them. Since the network traffic is based on the types and numbers of X requests generated by the clients, the applications can be coded in a way that most effectively reduces the LAN traffic.

Note that the above numbers are based on a stabilized system. The traffic varies from the above rates during receiver station start-up, when state information is being sent to the receiver(s) before the data can be displayed.

The actual traffic measured on the LAN depends on the ability of the software to keep up with the required speed. One area where speed and task scheduling is critical, is at the source station when the Protocol Distributor (PD) empties the protocol buffers in shared memory and sends them to PM (see Section 3.1.2.2). If PD is emptying the buffers slower than the multicast routine is filling them, the server may get delayed while waiting for an empty buffer to become available and this would cause the whole system to slow down.

Another critical area is at the receiver station where the Protocol Receiver (PR) sends the packets it has received to the server. Sending requests individually to the server without buffering will cause the system to slow down.

8.0 OUTSTANDING ISSUES

With the completion of the Display Sharing prototype, there are still some outstanding issues that must be resolved before a production system is implemented. Most of these issues are technical in nature; however, some require policy decisions at NASA.

8.1 Colormaps

In the Display Sharing prototype colors are passed from distributor to receiver by referencing an integer index into the colormap array at X Protocol level. This method can only reproduce original colors if the colormap is identical on all systems, both with respect to the color as well as to its index in the colormap. To insure this, one solution is to establish a standard colormap on all systems.

Another possible solution is to send the RGB value for each new color used. The RGB values will be requested from the source client, much in the same way as window attributes and graphics contexts are now being requested. The RGB value will then be parsed at the receiver to create a new entry in the local colormap. Parsing the color will create the closest matching color available on the receiving system. It will not necessarily be the identical color if the number of bit planes on the receiver were fewer than on the source. Additional processing will be required at the receiver every time a color is referenced to translate the remote index to the local one.

8.2 Fonts

The handling of different fonts is not implemented in the Display Sharing prototype. The issue of fonts is similar to the issue of colormaps, except that a font cannot be parsed by the receiver if it does not already exist. All systems need to agree on a standard set of fonts to be used by Display Sharing applications.

A default font of the correct size could be assigned if an exact matching font is not required.

8.3 Expose Events

Expose events are sent to applications to inform them that a portion of the window or all of the window needs to be regenerated. The application itself is responsible for redrawing the information in the exposed area. To simplify the problem of redrawing only the exact part that was exposed, many applications redraw the entire window when any part is exposed. Expose events are especially important to redraw static data, such as labels, which do not get updated after they are drawn, or data that is only updated very seldom.

In the Display Sharing prototype the receiver causes an expose event on the source station when a new window is created. The source then sends all the

protocol necessary to draw the initial state of the window. The application must handle expose events correctly in order for the received window to be identical to the source window.

8.4 Delay Until Appearance

Although the window is being distributed and that channel is requested for reception, a window will not appear at the receiver station until protocol has been detected from that window. In a situation where a client is only updated once every ten minutes, it may take up to ten minutes before the window appears at the receiver. A window that is never updated, will not appear at the receiver unless an expose event is caused at the source station forcing it to redraw the window.

A solution would be to make the receiver cause an expose event at the source as soon as a channel is requested for reception. This would prevent long delays before a rarely updated window appears.

8.5 Multiple Windows Per Client

To keep the amount of data to be distributed at reasonable levels, only windows that are currently being mapped should be sent. Child windows could include pop-up windows and pull-down menus, which probably should not be distributed.

8.6 Using the Display Sharing Wedge Approach

Use of the Display Sharing Wedge between the client and the server, instead of the modified server has some limits. The Wedge has a definite performance limit (see Section 7.1.2).

Other limiting factors of the Wedge approach include that clients must be started differently in order to be distributed. A client must be specifically attached to the Wedge when it is started. If it is not, it cannot be distributed without stopping and restarting it. Conversely, if a client is attached to Wedge when it is started, it cannot be unattached without being stopped.

The current version of Wedge is simply a stripped version of Xscope from the R4 MIT release. Some improvement in performance may be realized by writing a version of Wedge specifically optimized for Display Sharing.

8.7 Shared Memory and Semaphores

The Display Sharing prototype uses 24 semaphores and less than 32K bytes of shared memory. This must be accounted for when running the prototype to allow enough shared memory and semaphores for Display Sharing. The number of semaphores and the amount of shared memory can be set in the system configuration file.

8.8 Optimized X Code

The Display Sharing prototype's performance is directly affected by the efficiency of the clients that are being shared. To reduce network traffic, the number of X-Events, protocol packets and the number of round-trip requests should be kept to a minimum. This can be affected by the way the clients are designed.

Widgets enable creating clients simpler and faster, but they also reduce the efficiency of the client. Especially in cases where the standard Xlib libraries would be sufficient to create the client, using widgets would negatively affect the performance.

Xgks is a version of gks (Graphics Kernel System) which operates under the X environment. Xgks primitives are converted into X requests that are handled by the X server. Depending on the gks call, a word may be sent as separate characters instead of as a single word. Using Xgks instead of Xlib calls decreases the efficiency of the client.

8.9 Discontinued Distribution of Window

When the source removes a client from distribution, the receiver gets no indication of this, other than that the window is no longer being updated. Other ways of handling this could be to inform the receiver that the window is now static by changing its border.

8.10 Redundant Dedicated Hosts

Currently the Display Sharing system assumes only one dedicated host. The prototype can be modified to accept multiple dedicated hosts. If one host fails the Display Sharing System will try to compensate for it by using one of the other hosts. The LDM would show the combined channel guide from all hosts, and it would be transparent to the user which machine hosts any one channel. The number of redundant dedicated hosts can be set after the confidence level for the system is established.

8.11 Multiple Window Id's Over the Network

Currently the Display Sharing prototype does not handle the fact that window id's are not unique over several workstations. If two windows with identical window id's are being distributed from different workstations, and both are being received by the a third workstation, the receiver cannot be able to correctly attribute the incoming X-protocol to the correct window.

To solve this, additional information such as unique station id's, needs to be stored along with the window id's. Since all window id's on a single workstation are unique, the combination of a window id and a station id would insure that duplicate id's over a network do not exist.

8.12 gcm12 File

The /etc/mcgraphics/X11/gcm12 file must be compatible with the version of the modified server being used. The current modified server is compatible with the latest official release from MassComp, which is incompatible with the Ford Variant version. If the gcm12 file is incompatible with the modified server being used, it must be replaced by a compatible gcm12 file (see Section 4.2.1).

APPENDIX A

WORKSTATION LOCAL SHARED MEMORY

The Local Shared Memory structure shared between the server, PD, PR and LDM has the following fields:

semaphore:

An integer used to control access to shared memory area where protocol buffers used to pass data from the server to PD are stored. The semaphore is set and read by the server and by PD.

buf_stat[]:

An array of integers, indicating the state of each buffer which has one of the following values:

- MULTICAST : Set by multicast indicating that the buffer contains protocol to be distributed
- SM_EMPTY : Set by PD indicating that the buffer is currently empty

sm_status:

Integer set by LDM and used by PR, PD and multicast to indicate the status of the shared memory.

- SM_EMPTY : initialization value
- DIE : LDM has been requested to quit

start:

An integer used by LDM to indicate to multicast that it has authorized distribution on a new channel.

pd_alive:

Integer set by PD indicating when PD is alive and running.

pd_pid:

Integer set by PD containing the process id of the PD.

pr_init:

Integer set by PR indicating when PR is initialized.

pd_init:

Integer set by PD indicating when PD is initialized.

pm_port:

Unsigned short set by PR and LDM containing PM port number of the host where CDM was found.

pr_port:

Unsigned short set by PR containing PR port number.

distributor_id:

Integer set by PD containing the id number for the distributor.

pd_propagate_expose:

Integer set by PR indicating that PR has requested PD to send expose event to client.

expose_client:

XID set by PR indicating the window for PD to send the expose event back to

default_gc:

XID set by PR indicating the id of the source display's default graphics context.

root:

XID set by PR indicating the id of the source display's root.

wanted[]:

Array of integers indexed by client, set and used by multicast indicating if that client is to be distributed.

get_wat:

Integer set by PR indicating that PR wants PD to retrieve window attributes from the source by sending a request to PM.

send_wat:

Integer set by PR indicating that PR is requesting PD to send window attributes requested by PM.

wat_channel:

Integer set by PR indicating the channel for which PR is requesting PD to send the window attributes.

wat_bg_pixel:

Unsigned long set by PR containing the background pixel color for the window attributes that PR is requesting PD to send.

wat_parent:

XID set by PR containing the parent id of the window for which PR is requesting PD to send the window attributes.

wat_port:

Unsigned short set by PR and used by PD containing the receiver port that the window attributes are to be sent to

wat_id:

XID set by PR containing the graphics context id of the window attributes that PR is requesting PD to send.

wats:

An XWindowAttributes structure set by PR containing the window attributes that PR is requesting PD to send.

get_gc:

Integer set by PR indicating that PR wants PD to request graphics context info from PM.

send_gc:

Integer set by PR indicating that PR wants PD to send graphics context info to PM.

gc_channel:

Integer set by PR containing the source channel for which PD is requested to retrieve the graphics context info.

gc_id:

XID set by PR containing the id of the graphics context.

gc_port:

Unsigned short set by PR containing the receiver's port where PD sends the graphics context info once obtained.

pr_close_channel:

Integer set by LDM indicating to PR that there is a channel to be closed.

pr_close_client:

Integer set by LDM indicating to PR which channel to close.

clients[]:

An array of integers set by LDM indicating which client is being distributed on each channel.

source_default_gc[]:

An array of unsigned longs set by LDM and used by PR containing the default graphics context for each channel.

source_root[]:

An array of unsigned longs set by LDM and used by PR containing the source root id for each channel.

gcwin[]:

Array of GCWIN structures set by multicast containing information about the each local graphics context.

The GCWIN structure contains the following fields:

gid	:	XID containing the graphics id
window	:	XID containing the window id
mask	:	Unsigned long with bits set to indicate which GC values are used
GCValues	:	XGCValues structure of graphics context values

win[]:

Array of WIN structures set by multicast containing information about each local window.

The WIN structure contains the following fields:

window	:	XID containing the window id
background	:	Unsigned long containing the background pixel color for the window attributes
parent	:	XID containing the window id of the parent window

client[]:

Array of integers set by multicast and used by PD indicating the client id of the corresponding X Protocol buffer.

len[]:

Array of integers set by multicast and used by PD indicating the length of the corresponding X Protocol buffer.

window:

XID set by LDM and used by multicast containing the id of the window to be distributed.

xbuffer[][]:

Array of buffers of unsigned characters written by multicast and read by PD, containing X packet data to be distributed.

management_host[]:

Character string set by PR and LDM and used by PD containing the host name for the Central Distribution Manager.

APPENDIX B

DEDICATED HOST SHARED MEMORY

The Dedicated Host Shared Memory structure has the following fields:

sm_status:

Integer indicating the status of the shared memory. The following values are possible:

- AOK : initialization value
- PM_DIE : CDM is requested to quit

read_pipe:

Integer set by CDM and used to indicate to PM that there is a new Channel Map.

pm_port:

Unsigned short set by PM and used by CDM containing the PM port number.

new_source_channel:

Integer set by CDM to indicate to PM that the new source channel has been added to the downloaded channel map.

remv_source_channel:

Integer set by CDM to indicate to PM that the source channel has been removed in the downloaded channel map.

new_receiver_channel:

Integer set by CDM to indicate to PM that the new receiver channel has been added to the downloaded channel map.

remv_receiver_channel:

Integer set by CDM to indicate to PM that the receiver channel has been removed in the downloaded channel map.

station_index:

Integer set by CDM and used by PM containing the index of the distributing station.

source_default_gc[]:

An array of unsigned longs set and used by CDM containing the default graphics context for each channel.

source_root[]:

An array of unsigned longs set and used by CDM containing the source root id for each channel.

dest_fd[][]:

Array of integers set and used by PM containing by channel and by receiver all receiver file descriptors actively receiving protocol on each channel.

source_fd[]:

Array of integers set and used by PM containing by channel the file descriptors that are active sources for each channel.

Stations[]:

Array of Stations structures to describe a workstation configured with a Protocol Distributor and a Protocol Receiver. The structure values are set and used in CDM and PM. For a description of the individual structure members, see Appendix J.

ChanMap[]:

Array of ChanMap structures to hold channel map information. The structure values are set and used in CDM and PM. For a description of the individual structure members, see Appendix J.

source_name[][]:

Array of character strings set by CDM containing the source name for each channel.

channel_bytes[]:

Array of unsigned longs to store accumulated byte counts passing through PM for each channel. Used for LAN traffic measurement.

receiver_bytes[]:

Array of unsigned longs to store accumulated byte counts passing through PM for each receiver. Used for LAN traffic measurement.

distributor_bytes[]:

Array of unsigned longs to store accumulated byte counts passing through PM for each distributor. Used for LAN traffic measurement.

no_of_packets[]:

Unsigned long containing the accumulated count of packets passing through PM. Used for LAN traffic measurement.

APPENDIX C

I/O REQUEST TYPES

The I/O Request Types

NOOP:

No operation request.

X_DATA:

Indicates that an X Data packet will follow.

EXPOSE:

Used to request an expose event from the source.

GWATS:

Used to request Window Attributes from the source.

GGCS:

Used to request Graphics Context State from the source.

WATS:

Used to send Window Attributes to the receiver.

GCS:

Used to send Graphics Context State to the receiver.

SHUTCOMP:

Used to indicate that shutdown of receiver channel is complete.

APPENDIX D

RPC REQUEST CODES

CDM_GET_LIST

Request to retrieve the channel map - id list.

CDM_DIST_REQ

Distribution Authorization Request.

CDM_RECV_REQ

Reception Authorization Request.

CDM_REMV_CHAN

Request to remove a channel.

CDM_REMV_RECV

Request to remove a receiver from the linked list.

CDM_PRESENT

Broadcast request to locate dedicated host.

CDM_REG_DIST

Register a Distributor Request.

CDM_REG_RECV

Register a Receiver Request.

CDM_GO_AWAY

Request to go away, exit.

APPENDIX E

RPC DATA STRUCTURE FOR RETRIEVE TV GUIDE

Input structure:

Void

Output structure:

tv_guide[] : Array of strings of characters, one string per channel.

APPENDIX F

RPC DATA STRUCTURE FOR DISTRIBUTION AUTHORIZATION

Input structure:

ChanID

- chanid[] : Array of characters containing the source name
- hostname[] : Array of characters containing the host name
- distributor_id : Integer containing the id number for the distributor
- pr_port : Unsigned short containing PR port number
- default_gc : Unsigned long containing the XID of the default graphics context
- root : Unsigned long containing the XID of the source root
- xid : Unsigned long containing the client xid

Output structure:

DstAuth :

- authorization : Integer indicating whether distribution is authorized
- channel : Integer indicating channel to distribute on
- pm_port : Unsigned short indicating port number

APPENDIX G

RPC DATA STRUCTURE FOR RECEPTION AUTHORIZATION

Input structure:

ChanReq :

- channel	:	Integer indicating channel to receive on
- hostname[]	:	Array of characters containing the host name
- distributor_id	:	Integer containing the id number for the distributor
- pr_port	:	Unsigned short containing PR port number

Output structure:

RecvAuth :

- authorization	:	Integer indicating whether distribution is authorized
- pm_port	:	Unsigned short indicating port number
- default_gc	:	Unsigned long containing the XID of the default graphics context
- root	:	Unsigned long containing the XID of the source root

APPENDIX H

RPC DATA STRUCTURE FOR CANCEL DISTRIBUTION

Input structure:

channel : Integer indicating channel to remove.

Output structure:

retval : Integer indicating whether request was successful.

APPENDIX I

RPC DATA STRUCTURE FOR CANCEL RECEPTION

Input structure:

RemvRecv :

- channel : Integer indicating channel to remove
- portnum : Unsigned short indicating port number

Output structure:

- retval : Integer indicating whether request was successful

APPENDIX J

CHANNEL MAP AND STATION STRUCTURES

ChanMap:

Array of ChanMap structures to hold channel map information. The structure has the following members:

- num_receivers : Integer set and used by PM and CDM, indicating number of receivers for this channel
- client_id : Integer set by CDM and used by PM indicating client number for this channel
- recv_ports[] : Array of unsigned shorts set and used by PM and CDM, containing the port numbers of each receiver
- recv_hostname[][] : Array of character strings set by CDM containing the port name for each receiver
- source_hostname[] : Character string set by PM and CDM and used by PM, containing the source name for this channel

Stations:

Array of Stations structures to describe a workstation configured with a Protocol Distributor and a Protocol Receiver. The structure has the following members:

- pd_fd : Integer set by PM and used by PM and CDM, indicting file descriptor for PM to read from
- pr_fd : Integer set and used by PM, indicating file descriptor for PM to write to
- num_channels : Integer set by PM and used by PM and CDM, indicating number of channels this station is distributing on
- dist_channel[] : Array of integers set and used by PM, containing channel numbers of active channels
- dist_client[] : Array of integers set by CDM and used by PM, containing client being distributed on each associated dist_channel
- hostname[] : Character string set by CDM and used by PM, indicating name of host for distributor

APPENDIX K
SERVER LISTINGS

```
/*************************************************************************
The included program listings are prototypes, no warranty is expressed or
implied for their use in any other fashion.  They should not be considered
or used as production software.  The information in the listings is
supplied on an "as is" basis.  No responsibility is assumed for damages
resulting from the use of any information contained in the listings.

The software in these listings has been compiled on Masscomp 6350's and
6600's and on Sun 3's and 4's.  Modifications may be necessary for use on
other systems.
*************************************************************************/

/*
 * This routine is linked into the X-Server to allow multicasting
 * of client X-protocol to the distribution server.
 */
#include <sys/types.h>
#include <sys/param.h>
#include <sys/lock.h>
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <sys/wait.h>
#include <sys/ioctl.h>
#include <fcntl.h>
#include <signal.h>
#include <setjmp.h>
#include <math.h>
#include <X11/X.h>
#define NEED_REPLIES
#define NEED_EVENTS
#include <X11/Xproto.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include "../includes/ds_manage.h"
#include "../includes/smtypes.h"
#include "../includes/smdef.h"
#include "../includes/xdefs.h"

#define TRUE        1
#define FALSE       0

#ifdef PROFILE
#include "../includes/dist.h"
#define dispShareOverhead PAKLEN
#endif
```

```
/*
 * GLOBAL VARIABLES
 */
struct MC_SHMEMORY              *shmem;
static int                      locked[XBUFFERNUM];
static int                      current_buffer  = 0;
int                             shmid;

/*
 * GLOBAL FUNCTIONS
 */
void        go_away();
int         lock();
void        unlock();
void        lock_others();
static int  semcall();

/*
 * EXTERNAL FUNCTIONS
 */
extern          check_gcwin();
extern          check_new_window();
extern char XFunctName[][80];

/*****************************************************/
/*
 * This routine is called from the X server itself.
 */
multicast(client_index,bufptr,len)
    register int            client_index;
    register unsigned char  *bufptr;
    register int            len;
{
    register struct MC_SHMEMORY *memptr;
    register int                i;
    static int                  first_time      = TRUE;
static int last_len = 0;
static int prev_buffer = XBUFFERNUM - 1;
int lock_status;

    /* Initialize */
    memptr  = shmem;

/*
 * If this is the first time through here for a wanted
 * client, then we need to do some things first before
 * we can continue:
 *      o attach to shared memory
 */
    if (first_time) {

        first_time  = FALSE;
```

```
#ifdef PROFILE
fprintf( stderr,"Display Sharing overhead: %d\n",dispShareOverhead );
#endif

/*
 * Set up to clean up before quitting
 */
        signal(SIGQUIT, go_away);

        for (i=0;i<XBUFFERNUM;i++)
            locked[i]   = FALSE;

        /* create the shared memory area */
        create_shared_memory();

#ifdef TRACE
fprintf(stderr,"MC::shared memory created shmem:0x%x\n",shmem);
#endif

        memptr       = shmem;

        /* create shared memory semaphore */
        create_semaphore();

#ifdef TRACE
fprintf(stderr,"MC::semaphore created\n");
#endif

/*
 * See if we need to note the particular gc-window combination
 */
        check_gcwin(bufptr);


        /* now check to see if the client is wanted */
        if (!memptr->wanted[client_index])
            return;

    }  /* end if first time */

/*
 * Keep checking to see if we are supposed to go away
 */
    if (memptr->sm_status==DIE) {
        fprintf(stderr,"MC:: Requested to die...\n");
        sleep(5);
        fprintf(stderr,"MC:: BYE BYE.\n");
        go_away();
    }

/*
```

```
 * See if we need to note the particular gc-window combination
 */
    check_gcwin(bufptr);

/*
 * If the protocol distributor is not alive, return
 */
    if (!memptr->pd_alive)
        return;

#ifdef DUMMY
return;
#endif

#ifdef PROFILE
    /* increment profile array */
    if ( (*bufptr > lastRequest) || (client_index > MAX_CLIENTS) )
        fprintf(    stderr,    "MC-PROFILE::    Unknown    request:[%d],
client:[%d]\n", *bufptr, client_index );
    else
        memptr->pArray[ *bufptr ][ client_index ]++;

    /* accumulate length for total byte count to calculate throughput */
    len = get_length( bufptr );
    memptr->accumLen[ client_index ] += len + dispShareOverhead;
    memptr->currLen[ client_index ]  += len + dispShareOverhead;

    /* check if profiler wants to exit */
    if ( memptr->wantToExit == TRUE ) {
        memptr->wantToExit = FALSE;
        fprintf( stderr, "MC:: profile wants to exit.\n" );
        sleep(5);
        fprintf( stderr, "MC:: profile done.\n" );
        go_away();
    }
#endif

/*
 * Check to see if there is a new window to be sent.
 */
    check_new_window();

    if ( !(memptr->wanted[client_index]) )
        return;
/*
 * Now write the correct data out to the distribution server.
 */

    len = get_length( bufptr );

    if ( len >= XBUFFERSIZE ) {
        fprintf( stderr, "MC:: *** Request larger than buffer (%ld) !!!
```

```
***\n", len );
        go_away();
    }

    while ( 1 ) {

        lock_status = lock( memptr->semaphore, prev_buffer, IPC_NOWAIT );

/* See if the next buffer is empty so we can put stuff into it */

        if ( memptr->buf_stat[current_buffer] == SM_EMPTY ) {

            unlock( memptr->semaphore, prev_buffer );
            lock( memptr->semaphore, current_buffer, 0 );

            memptr->client[current_buffer]      = client_index;
            memptr->len[current_buffer]         = len;
            memcpy(memptr->xbuffer[current_buffer],bufptr,len);
            memptr->buf_stat[current_buffer]    = MULTICAST;

            unlock( memptr->semaphore, current_buffer );
            lock_others(current_buffer);

            last_len = len;
            prev_buffer = current_buffer;
            current_buffer++;
            if ( current_buffer >= XBUFFERNUM )
                current_buffer = 0;
            return;
        }
/* Buffer is being used, can we add to the end of the previous one ? */

        else if ( (lock_status == 0) &&
                  (memptr->buf_stat[prev_buffer] == MULTICAST) &&
                    (memptr->client[prev_buffer] == client_index) ) {

            if ( (len + last_len) < XBUFFERSIZE ) {
                memptr->len[prev_buffer] += len;
                memcpy( memptr->xbuffer[prev_buffer] + last_len, bufptr,
len );
                unlock( memptr->semaphore, prev_buffer );

                last_len += len;
                return;
            }
            else {
                /* can't fit any more requests into this buffer */

                unlock( memptr->semaphore, prev_buffer );
#ifdef TRACE
fprintf(stderr,"*%d*",current_buffer);
#endif
```

```
                sleep(1);
            }
        }
        else {
/*
 * If we get here:
 *    the current buffer was busy  AND
 * we either
 *    could not lock the previous one  OR
 *    the buffer was empty OR
 *    the new request was for a different client.
 */
            unlock( memptr->semaphore, prev_buffer );

#ifdef TRACE
if ( lock_status != 0 )
 fprintf( stderr,"l:%d.",lock_status);
if ( memptr->buf_stat[prev_buffer] != MULTICAST )
 fprintf( stderr,"s:%d.",memptr->buf_stat[prev_buffer] );
if ( memptr->client[prev_buffer] != client_index )
 fprintf( stderr,"c:%d.",memptr->client[prev_buffer] );
fprintf(stderr,"!%d!",current_buffer);
#endif
            sleep(1);
        }   /* end else */

/*
 * Keep checking to see if we are supposed to go away
 */
        if (memptr->sm_status==DIE) {
            fprintf(stderr,"MC:: Requested to die...\n");
            sleep(5);
            fprintf(stderr,"MC:: BYE BYE.\n");
            go_away();
        }
    }   /* end while */
} /* end multicast */
/*****************************************************/


/*****************************************************/
/*
 * This routine creates the shared memory area used by
 * both the multicast routine and the Protocol Distributor.
 */
int
create_shared_memory()
{
    register int            i;

    /* kill any existing memory segments */
    if ((shmid = shmget(SM_KEY,0,0)) >= 0)  {
```

```
fprintf(stderr,"Shared memory exists (%d), removing it\n",shmid);
        shmctl(shmid,IPC_RMID,(struct shmid_ds *)0);
    }

    /* create a new one */
    shmid = shmget(SM_KEY, sizeof(struct MC_SHMEMORY),
                IPC_CREAT | 0777);
    if (shmid < 0 ) {
        perror("MC::shmget (create_shared_memory:");
        fprintf(stderr,"MC:: shmid < 0 !!!!!!\n");
        go_away();
    }

    /* attach to it */
    shmem = (struct MC_SHMEMORY *)shmat(shmid,0,0);
    if (shmem==(struct MC_SHMEMORY *)-1) {
        perror("MC::shmat (create_shared_memory):");
        fprintf(stderr,"MC:: shmem == *(-1)\n");
        go_away();
    }

#ifdef LOCKIT
    /* lock it into memory */
    if (plockin(shmem,sizeof(struct MC_SHMEMORY))<0)
        perror("MC:: plockin(Shared Memory):");
#endif

    shmem->sm_status            = SM_EMPTY;
    shmem->pd_alive             = FALSE;
    shmem->pd_pid               = -1;
    shmem->pm_port              = 0;
    shmem->pr_port              = 0;
    shmem->window               = 0;
    shmem->pd_propagate_expose  = FALSE;
    for (i=0;i<MAX_CLIENTS;i++)
        shmem->wanted[i]        = FALSE;
    for (i=0;i<MAX_GCS;i++) {
        shmem->gcwin[i].gid     = 0;
        shmem->gcwin[i].window  = 0;
    }
    for (i=0;i<MAX_CHANNELS;i++) {
        shmem->clients[i]           = -1;
        shmem->source_default_gc[i] = 0;
        shmem->source_root[i]       = 0;
    }
    for (i=0;i<XBUFFERNUM;i++)
        shmem->buf_stat[i]      = SM_EMPTY;
    shmem->get_wat              = FALSE;
    shmem->send_wat             = FALSE;
    shmem->wat_channel          = -1;
    shmem->wat_port             = 0;
    shmem->get_gc               = FALSE;
```

```
    shmem->send_gc              = FALSE;
    shmem->gc_channel           = -1;
    shmem->gc_port              = 0;
    shmem->pr_init              = FALSE;
    shmem->pd_init              = FALSE;
    shmem->pr_close_channel     = FALSE;
    shmem->pr_close_client      = -1;

    return(TRUE);

} /* end create_shared_memory */
/*****************************************************/

/*****************************************************/
/*
 * This routine replaces the exit call. May be used to
 * clean up before exiting.
 */
void
go_away()
{
    int         i;
/*
 * Get rid of the semaphore identifier.
 */
    for (i=0;i<XBUFFERNUM;i++)
        semctl(shmem->semaphore,i,IPC_RMID,0);

/*
 * Get rid of the shared memory identifier.
 */
    shmctl(shmid,IPC_RMID,0);

/*
 * Tell someone we are going away.
 */
    fprintf(stderr,"MC:: EXITING.....\n");
    sleep(5);
    exit(0);
} /* end go_away */
/*****************************************************/

/*****************************************************/
/*
 *  This routine returns the length of the particular
 * protocol package. Note that the length is the second
 * two bytes of the packet, in terms of 32 bit
 * quantities. We left shift by two to get the byte
 * count.
 */
int
get_length(ptr)
```

```
    register unsigned char  *ptr;
{
    register unsigned short *shortptr;
    register unsigned short length;

    shortptr = (unsigned short *)(ptr+2);
    length   = (*shortptr<<2);

    return(length);

} /* end get length */
/*****************************************************/

/*****************************************************/
/*
 * This routine creates the shared memory semaphore
 */
create_semaphore()
{
    int                 i;
    union semun {
        int             val;
        struct semid_ds *buf;
        ushort          *array;
    } arg;
    struct sembuff  sb;
    sb.sem_op   = -1;
    sb.sem_flg  = 0;

/*
 * Create all the semaphores.
 */
    shmem->semaphore = semget(IPC_PRIVATE,XBUFFERNUM,0666|IPC_CREAT);
    if (shmem->semaphore<0) {
        perror("MC::semget (create_semaphore):");
        fprintf(stderr,"MC:: shmem->semap < 0 !! \n");
        go_away();
    }

    for (i=0;i<XBUFFERNUM;i++) {
        arg.val = 1;
        if (semctl(shmem->semaphore,i,SETVAL,arg)<0) {
            perror("MC::semctl (create_semaphore):");
            fprintf(stderr,"MC:: semctl < 0) \n");
            go_away();
        }

        sb.sem_num  = i;
        if (semop(shmem->semaphore,&sb,1)==-1) {
            perror("semop (multicast):");
            fprintf(stderr,"MC:: semop == -1) \n");
            go_away();
```

```
        }

        locked[i]       = TRUE;
#ifdef TRACE
fprintf(stderr,"MC:: create_semaphore locking:%d\n",i);
#endif
    } /* end for i */

} /* end create semaphore */
/*****************************************************/

/*****************************************************/
/*
 * This routine locks on a semaphore.
 */
int
lock(id, buffer, flag)
    register int id;
    register int buffer;
    short flag;
{

#ifdef SEMAPHORE
fprintf(stderr,"MC:: attempting to lock buffer:%d\n",buffer);
#endif

/*
 * If it is already locked, do not
 * bother with system call.
 */
    if (locked[buffer])
{
#ifdef SEMAPHORE
fprintf(stderr,"MC:: buffer %d already locked, returning early\n",
buffer);
#endif
        return(0);
}

/*
 * Otherwise, lock the semaphore and begin processing
 */
    if ( semcall(id,-1,buffer,flag) == 0 ) {
        locked[buffer] = TRUE;
#ifdef SEMAPHORE
fprintf(stderr,"MC:: just locked that semaphore\n");
#endif
        return( 0 );
    }

#ifdef SEMAPHORE
fprintf(stderr,"MC:: could not lock that semaphore\n");
```

```
#endif
    return( -1 );

} /* end lock */
/*******************************************************/

/*******************************************************/
/*
 * This routine unlocks a semaphore.
 */
void
unlock(id, buffer)
    register int id;
    register int buffer;
{
#ifdef SEMAPHORE
fprintf(stderr,"MC:: attempting to unlock buffer:%d\n",buffer);
#endif
/*
 * If the semaphore is not locked, then don't unlock it
 */
    if (!locked[buffer]) {

#ifdef SEMAPHORE
fprintf(stderr,"MC:: it is not locked, returning early\n");
#endif
        return;
    }

    semcall(id,1,buffer,0);
    locked[buffer] = FALSE;

#ifdef SEMAPHORE
fprintf(stderr,"MC:: just unlocked that semaphore\n");
#endif

} /* end unlock */
/*******************************************************/

/*******************************************************/
/*
 * This routine performs the semaphore operations.
 */
static int
semcall(sid,op,buffer,flag)
    register int sid;
    register int op;
    register int buffer;
    short flag;
{
    struct sembuff  sb;
```

```
    sb.sem_num  = buffer;
    sb.sem_op   = op;
    sb.sem_flg  = flag;
    if (semop(sid,&sb,1)==-1) {
        if ( sb.sem_flg & IPC_NOWAIT )
            return( -1 );
        perror("semop (multicast):");
        fprintf(stderr,"MC:: semop == -1) \n");
        go_away();
    }

} /* end semcall */
/*****************************************************/

/*****************************************************/
/*
 * This routine attempts to lock the semaphore associated
 * with empty x buffers so that the Protocol Distributor
 * does not do so much busy work.
 */
void
lock_others(current)
    register int            current;
{
    register int            i;
    register struct sembuff *sbptr;
    struct sembuff          sb;

#ifdef SEMAPHORE
fprintf(stderr,"MC:: lock others called in buffer:%d\n",current);
#endif

    sbptr = &sb;
    for (i=0;i<XBUFFERNUM;i++) {
        if (    !locked[i] &&
                i!=current &&
                shmem->buf_stat[i]==SM_EMPTY ) {
#ifdef TRACE
fprintf(stderr,"MC:: locking additionally buffer:%d\n",i);
#endif
            sbptr->sem_num  = i;
            sbptr->sem_op   = -1;
            sbptr->sem_flg  = 0;
            locked[i]   = TRUE;
            if (semop(shmem->semaphore,sbptr,1)==-1) {
                perror("semop (lock_others):");
        fprintf(stderr,"MC:: semop == -1) \n");
                go_away();
            } /* end if */

        } /* end if */
#ifdef SEMAPHORE
```

```
else {
fprintf(stderr,"MC:: not locking %d locked:%d current:%d stat:%d\n",
i,locked[i],current,shmem->buf_stat[i]);
}
#endif

    } /* end for i */

} /* end lock others */
/*****************************************************/
```

```
/*
 * File          : multix.c
 * Date          : 10/26/89
 * Author        : P. Fitzgerald SwRI
 * Description   : This file contains all the X Window related code for the
 *                 multicast routine.
 */
#include <sys/types.h>
#include <sys/param.h>
#include <stdio.h>
#include <X11/X.h>
#define NEED_REPLIES
#define NEED_EVENTS
#include <X11/Xproto.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include "../includes/ds_manage.h"
#include "../includes/smtypes.h"
#include "../includes/smdef.h"
#include "../includes/dist.h"

#define     TRUE    1
#define     FALSE   0
#define     ASYNC   TRUE
#define     NOASYNC FALSE

/*
 * GLOBAL ROUTINES
 */
int             mapped();
int             get_index();

/*
 * EXTERNAL VARIABLES
 */
struct MC_SHMEMORY              *shmem;

/*
 * GLOBAL VARIABLES
 */
static  int                     numwins = 0;

/*****************************************************/
/*
 * This routine checks shared memory to see if the window
 * distribution manager has requested that we start sending
 * a new window over to the target.
 */
void
check_new_window()
{
    register int                client;
```


C-2

```
    register struct MC_SHMEMORY *memptr;

    memptr  = shmem;
/*
 * Note that the upper so many bits of the window id will
 * yield the client index in the MIT implementation.
 */
    if (memptr->start) {
        memptr->start                   = FALSE;
client = memptr->window;
        memptr->wanted[client]          = TRUE;
    }  /* end if */

}  /* end check_new_window */
/*****************************************************/

/***********************************************************/
/*
 * This routine checks the X buffer to determine if there is
 * any code related to a Graphics Context. If there is, the
 * information is stored in shared memory even if the client
 * is not yet 'wanted' for distribution.
 */
void
check_gcwin(bufptr)
    register unsigned char  *bufptr;
{
    register unsigned char      xtype;
    register xCreateGCReq       *CreateGC;
    register xChangeGCReq       *ChangeGC;
    register xCopyGCReq         *CopyGC;
    register xResourceReq       *ResourceReq;

    register xCreateWindowReq           *CreateWindow;
    register xChangeWindowAttributesReq *Cwats;
    register struct MC_SHMEMORY *memptr;

    int                     gc_index;
    int                     dest_index;

    memptr = shmem;

/*
 * Determine what type of X request it is.
 */
    xtype   = (int)(*bufptr);


/*
 * Now handle different types of Graphics Context calls.
 */
        switch (xtype) {
```

```
/*****************************************************************/
/*
 * Handle creation of Window
 */
            case X_CreateWindow:
#ifdef TRACE
fprintf(stderr,"MC:: X_CreateWindow protocol noticed.\n");
#endif
                CreateWindow = (xCreateWindowReq *)bufptr;

store_window_background(CreateWindow->wid,CreateWindow->mask,bufptr,num
wins);
                memptr->win[numwins].parent = CreateWindow->parent;
#ifdef TRACE
fprintf(stderr,"MC:: store parent 0x%x, wid 0x%x, index: %d\n",
CreateWindow->parent,CreateWindow->wid,numwins);
#endif
                numwins++;
                if (numwins>=MAX_WINS) {
                    fprintf(stderr,"MC:: window/background overflow.\n");
                }
                break;
/*****************************************************************/


/*****************************************************************/
/*
 * Handle change of window attributes
 */
            case X_ChangeWindowAttributes:
#ifdef TRACE
fprintf(stderr,"MC:: X_ChangeWindowAttributes protocol noticed.\n");
#endif
                Cwats = (xChangeWindowAttributesReq *)bufptr;

change_window_background(Cwats->window,Cwats->valueMask,bufptr);
                break;
/*****************************************************************/


/*****************************************************************/
/*
 * Handle creation of Graphics Contexts
 */
            case X_CreateGC:

                /* get first new index */
                gc_index = get_index( 0 );
                /* Check to see if we reached our limit */
                if (gc_index < 0) {
fprintf(stderr,"MC:: GC_WINDOW map overflow in shared memory.");
```

```
                    perror("MC::  GC_WINDOW  map  overflow  in  shared
memory.");
                    return;
                }

                /* Set pointer to protocol packet */
                CreateGC = (xCreateGCReq *)bufptr;

                /* Store the initial id values and which drawable the
                   gc is associated with. */
                memptr->gcwin[gc_index].gid = CreateGC->gc;
                memptr->gcwin[gc_index].window  = CreateGC->drawable;
/*
 *  Set foreground and background to default values, server will not set
 *  mask bits for defaults !
 */
                memptr->gcwin[gc_index].GCValues.foreground = 0;
                memptr->gcwin[gc_index].GCValues.background = 1;

#ifdef TRACE
fprintf(stderr,"MC:: check_gcwin (createGC) > gid: 0x%lx, window: 0x%lx,
mask: 0x%x (num: %d)\n",
memptr->gcwin[gc_index].gid,memptr->gcwin[gc_index].window,CreateGC->ma
sk,gc_index);
#endif

                /* Now that we have mapped that one, lets store
                   its current values into shared memory.  */

store_gc_values(gc_index,CreateGC->mask,(bufptr+sz_xCreateGCReq) );
                break;
/*****************************************************************/


/*****************************************************************/
/*
 * Handle Changing of specific graphics context fields.
 */
            case X_ChangeGC:

                /* Pointer to request */
                ChangeGC = (xChangeGCReq *)bufptr;

                /* Is this GC in our list? */
                gc_index    = get_index(ChangeGC->gc);
                if ( gc_index < 0 ) {
fprintf( stderr,"...changeGC trying to access non-existent id: 0x%x\n",
ChangeGC->gc);
                    return;
                }

#ifdef TRACE
```

```
fprintf(stderr,"check_gcwin (changeGC) > gid: 0x%lx, window: 0x%lx (num:
%d)\n",
memptr->gcwin[gc_index].gid,memptr->gcwin[gc_index].window,gc_index);
#endif

                /* Yes so copy new values into memory */
                store_gc_values(gc_index,ChangeGC->mask,
                    (bufptr+sz_xChangeGCReq) );

                break;
/*****************************************************************/

/*****************************************************************/
/*
 * Handle Copying from one GC to another.
 */
            case X_CopyGC:

                /* Pointer to request */
                CopyGC = (xCopyGCReq *)bufptr;

                /* Get source and destination indexes */
                gc_index    = get_index(CopyGC->srcGC);
                dest_index  = get_index(CopyGC->dstGC);

                if ( gc_index < 0 ) {
fprintf( stderr,"...copyGC trying to access non-existent source id:
0x%x\n", CopyGC->srcGC);
                    return;
                }
                if ( dest_index < 0 ) {
fprintf( stderr,"...copyGC trying to access non-existent dest id: 0x%x\n",
CopyGC->dstGC);
                    return;
                }

#ifdef TRACE
fprintf(stderr,"check_gcwin (copyGC) > gid: 0x%lx, window: 0x%lx (srcIx:
%d, destIx: %d)\n",
memptr->gcwin[gc_index].gid,memptr->gcwin[gc_index].window,gc_index,des
t_index);
#endif

                /* Copy GC Values from memory to memory */
                copy_gc_values(gc_index,dest_index,CopyGC->mask);

                break;
/*****************************************************************/


/*****************************************************************/
/*
```

```
 * Handle Freeing a GC.
 */
            case X_FreeGC:

                /* Pointer to request */
                ResourceReq = (xResourceReq *)bufptr;
                gc_index    = get_index(ResourceReq->id);
                if (gc_index<0) {
fprintf( stderr,"...freeGC trying to free non-existent id: 0x%x\n",
ResourceReq->id);
                    return;
                }

#ifdef TRACE
fprintf(stderr,"check_gcwin (freeGC) > gid: 0x%lx, window: 0x%lx (num:
%d)\n",
memptr->gcwin[gc_index].gid,memptr->gcwin[gc_index].window,gc_index);
#endif

                memptr->gcwin[gc_index].gid     = 0;
                memptr->gcwin[gc_index].window  = 0;

                break;
/*****************************************************************/


/*****************************************************************/
/*
 * The default is to just ignore the protocol.
 */
            default:
                break;

        } /* end switch */

} /* end check_gcwin */
/*****************************************************/


/*****************************************************/
/*
 * This routine parses the X data buffer for Graphics
 * Context state data based on the input mask and stores
 * those values in  shared memory.
 */
store_gc_values(index,mask,bufptr)
    register    int             index;
    register    unsigned long   mask;
    register    unsigned char   *bufptr;
{
    register struct MC_SHMEMORY *memptr;
    register        XGCValues   *valptr;
```

```
    memptr  = shmem;
    valptr  = &(memptr->gcwin[index].GCValues);

/*
 * Go through all the possible mask values and if true,
 * store the value into shared memory.
 */

    if (mask&GCFunction) {
#ifdef ALL_MASK
        memptr->gcwin[index].GCValues.function  =
#else
        valptr->function =
#endif
                (int)*((int *)bufptr);
        bufptr+=sizeof(int);
    }

    if (mask&GCPlaneMask) {
#ifdef ALL_MASK
        memptr->gcwin[index].GCValues.plane_mask    =
#else
        valptr->plane_mask  =
#endif
                (unsigned long)*((unsigned long*)bufptr);
        bufptr+=sizeof(unsigned long);
    }

    if (mask&GCForeground) {
#ifdef ALL_MASK
        memptr->gcwin[index].GCValues.foreground    =
#else
        valptr->foreground  =
#endif
                (unsigned long)*((unsigned long*)bufptr);
        bufptr+=sizeof(unsigned long);

#ifdef TRACE
fprintf( stderr,"---store_gc_values (ix: %d): foreground is %d\n",
index, memptr->gcwin[index].GCValues.foreground );
#endif

    }

    if (mask&GCBackground) {
#ifdef ALL_MASK
        memptr->gcwin[index].GCValues.background    =
#else
        valptr->background  =
#endif
                (unsigned long)*((unsigned long*)bufptr);
```

```
        bufptr+=sizeof(unsigned long);

#ifdef TRACE
fprintf( stderr,"---store_gc_values (ix: %d): background is %d\n",
index, memptr->gcwin[index].GCValues.background );
#endif

    }

#ifdef ALL_MASK
    if (mask&GCLineWidth) {
        memptr->gcwin[index].GCValues.line_width    =
                (int)*((int *)bufptr);
        bufptr+=sizeof(int);
    }

    if (mask&GCLineStyle) {
        memptr->gcwin[index].GCValues.line_style =
                (int)*((int *)bufptr);
    }

    if (mask&GCCapStyle) {
        memptr->gcwin[index].GCValues.cap_style =
                (int)*((int *)bufptr);
        bufptr+=sizeof(int);
    }

    if (mask&GCJoinStyle) {
        memptr->gcwin[index].GCValues.join_style    =
                (int)*((int *)bufptr);
        bufptr+=sizeof(int);
    }

    if (mask&GCFillStyle) {
        memptr->gcwin[index].GCValues.fill_style    =
                (int)*((int *)bufptr);
        bufptr+=sizeof(int);
    }

    if (mask&GCFillRule) {
        memptr->gcwin[index].GCValues.fill_rule =
                (int)*((int *)bufptr);
        bufptr+=sizeof(int);
    }

    if (mask&GCTile) {
        memptr->gcwin[index].GCValues.tile =
                (XID)*((XID *)bufptr);
        bufptr+=sizeof(XID);
    }

    if (mask&GCStipple) {
```

```
    memptr->gcwin[index].GCValues.stipple =
            (XID)*((XID *)bufptr);
    bufptr+=sizeof(XID);
}

if (mask&GCTileStipXOrigin) {
    memptr->gcwin[index].GCValues.ts_x_origin   =
            (int)*((int *)bufptr);
    bufptr+=sizeof(int);
}

if (mask&GCTileStipYOrigin) {
    memptr->gcwin[index].GCValues.ts_y_origin   =
            (int)*((int *)bufptr);
    bufptr+=sizeof(int);
}

if (mask&GCFont) {
    memptr->gcwin[index].GCValues.font =
            (XID)*((XID *)bufptr);
    bufptr+=sizeof(XID);
}

if (mask&GCSubwindowMode) {
    memptr->gcwin[index].GCValues.subwindow_mode    =
            (int)*((int *)bufptr);
    bufptr+=sizeof(int);
}

if (mask&GCGraphicsExposures) {
    memptr->gcwin[index].GCValues.graphics_exposures    =
            (Bool)*((Bool *)bufptr);
    bufptr+=sizeof(Bool);
}

if (mask&GCClipXOrigin) {
    memptr->gcwin[index].GCValues.clip_x_origin =
            (int)*((int *)bufptr);
    bufptr+=sizeof(int);
}

if (mask&GCClipYOrigin) {
    memptr->gcwin[index].GCValues.clip_y_origin =
            (int)*((int *)bufptr);
    bufptr+=sizeof(int);
}

if (mask&GCClipMask) {
    memptr->gcwin[index].GCValues.clip_mask =
            (XID)*((XID *)bufptr);
    bufptr+=sizeof(XID);
}
```

```
    if (mask&GCDashOffset) {
        memptr->gcwin[index].GCValues.dash_offset   =
             (int)*((int *)bufptr);
        bufptr+=sizeof(int);
    }

    if (mask&GCDashList) {
        memptr->gcwin[index].GCValues.dashes    =
                (char)*((char *)bufptr);
        bufptr+=sizeof(char);
    }
#endif

    memptr->gcwin[index].mask   = mask;

} /* end store_gc_values */
/*****************************************************/

/*****************************************************/
int
get_index(gid)
    register XID        gid;
{
    register int                    i;
    register struct MC_SHMEMORY *memptr;

    /* Initialize */
    memptr  = shmem;

    for (i=0;i<MAX_GCS;i++) {
        /* already in there */
        if (memptr->gcwin[i].gid==gid)
            return(i);
    }  /* end for */

    return(-1);
} /* end get_index */
/*****************************************************/

/*****************************************************/
/*
 * This routine copies the contents of one graphics context
 * to another.
 */
copy_gc_values(source,dest,mask)
    register    int             source;
    register    int             dest;
    register    unsigned long   mask;
{
    register struct MC_SHMEMORY *memptr;
    register        XGCValues   *valptr;
```

```
    memptr  = shmem;
    valptr  = &(memptr->gcwin[dest].GCValues);

/*
 * Go through all the possible mask values and if true,
 * store the value into shared memory.
 */

    if (mask&GCFunction)
#ifdef ALL_MASK
        memptr->gcwin[dest].GCValues.function   =
#else
        valptr->function    =
#endif
            memptr->gcwin[source].GCValues.function;

    if (mask&GCPlaneMask)
#ifdef ALL_MASK
        memptr->gcwin[dest].GCValues.plane_mask =
#else
        valptr->plane_mask  =
#endif
            memptr->gcwin[source].GCValues.plane_mask;

    if (mask&GCForeground)
#ifdef ALL_MASK
        memptr->gcwin[dest].GCValues.foreground =
#else
        valptr->foreground  =
#endif
            memptr->gcwin[source].GCValues.foreground;

    if (mask&GCBackground)
#ifdef ALL_MASK
        memptr->gcwin[dest].GCValues.background =
#else
        valptr->background  =
#endif
            memptr->gcwin[source].GCValues.background;

#ifdef ALL_MASK

    if (mask&GCLineWidth)
        memptr->gcwin[dest].GCValues.line_width =
            memptr->gcwin[source].GCValues.line_width;

    if (mask&GCLineStyle)
        memptr->gcwin[dest].GCValues.line_style =
            memptr->gcwin[source].GCValues.line_style;

    if (mask&GCCapStyle)
```

```
    memptr->gcwin[dest].GCValues.cap_style  =
        memptr->gcwin[source].GCValues.cap_style;

if (mask&GCJoinStyle)
    memptr->gcwin[dest].GCValues.join_style =
        memptr->gcwin[source].GCValues.join_style;

if (mask&GCFillStyle)
    memptr->gcwin[dest].GCValues.fill_style =
        memptr->gcwin[source].GCValues.fill_style;

if (mask&GCFillRule)
    memptr->gcwin[dest].GCValues.fill_rule  =
        memptr->gcwin[source].GCValues.fill_rule;

if (mask&GCTile)
    memptr->gcwin[dest].GCValues.tile       =
        memptr->gcwin[source].GCValues.tile;

if (mask&GCStipple)
    memptr->gcwin[dest].GCValues.stipple        =
        memptr->gcwin[source].GCValues.stipple;

if (mask&GCTileStipXOrigin)
    memptr->gcwin[dest].GCValues.ts_x_origin    =
        memptr->gcwin[source].GCValues.ts_x_origin;

if (mask&GCTileStipYOrigin)
    memptr->gcwin[dest].GCValues.ts_y_origin    =
        memptr->gcwin[source].GCValues.ts_y_origin;

if (mask&GCFont)
    memptr->gcwin[dest].GCValues.font       =
        memptr->gcwin[source].GCValues.font;

if (mask&GCSubwindowMode)
    memptr->gcwin[dest].GCValues.subwindow_mode =
        memptr->gcwin[source].GCValues.subwindow_mode;

if (mask&GCGraphicsExposures)
    memptr->gcwin[dest].GCValues.graphics_exposures =
        memptr->gcwin[source].GCValues.graphics_exposures;

if (mask&GCClipXOrigin)
    memptr->gcwin[dest].GCValues.clip_x_origin  =
        memptr->gcwin[source].GCValues.clip_x_origin;

if (mask&GCClipYOrigin)
    memptr->gcwin[dest].GCValues.clip_y_origin  =
        memptr->gcwin[source].GCValues.clip_y_origin;

if (mask&GCClipMask)
```

```
        memptr->gcwin[dest].GCValues.clip_mask  =
            memptr->gcwin[source].GCValues.clip_mask;

    if (mask&GCDashOffset)
        memptr->gcwin[dest].GCValues.dash_offset    =
            memptr->gcwin[source].GCValues.dash_offset;

    if (mask&GCDashList)
        memptr->gcwin[dest].GCValues.dashes     =
            memptr->gcwin[source].GCValues.dashes;
#endif

    memptr->gcwin[dest].mask    = mask;

} /* end copy_gc_values */
/*****************************************************/


/*****************************************************/
int
store_window_background(wid,mask,bufptr,number)
    XID             wid;
    unsigned long   mask;
    unsigned char   *bufptr;
    int             number;
{
    unsigned char   *data;
    register struct MC_SHMEMORY *memptr;

    /* Initialize */
    memptr  = shmem;

/*
 * Store this window id into the slot in the array
 */
    memptr->win[number].window = wid;

/*
 * If Background Pixmap specified, then background pixel
 * data follows it (if it is there)
 */
    data = NULL;
    if (mask & CWBackPixmap)
        data = bufptr + sz_xCreateWindowReq + 4;
    else if (mask & CWBackPixel)
        data = bufptr + sz_xCreateWindowReq;

    if (data!=NULL)
        memptr->win[number].background = *(unsigned long *)data;

#ifdef TRACE
fprintf(stderr,"MC::    store_window_background    number:%d    win:0x%x
```

```
background:0x%x\n",
number,memptr->win[number].window,memptr->win[number].background);
#endif

} /* end store_window_background */
/*************************************************/

/*************************************************/
int
change_window_background(wid,mask,bufptr)
    XID             wid;
    unsigned long   mask;
    unsigned char   *bufptr;
{
    int             i;
    int             number;
    unsigned char   *data;
    register struct MC_SHMEMORY *memptr;

    /* Initialize */
    memptr  = shmem;

/*
 * Find the correct entry
 */
    number  = -1;
    for (i=0;i<MAX_WINS;i++) {
        if (memptr->win[i].window==wid) {
            number = i;
            break;
        }
    } /* end for */

    if (number<0) {
#ifdef TRACE
        fprintf(stderr,"MC::     Unable   to   find   window   to   change
background.\n");
#endif
    }
    else {
        data = NULL;
        if (mask & CWBackPixmap)
            data = bufptr + sz_xChangeWindowAttributesReq + 4;
        else if (mask & CWBackPixel)
            data = bufptr + sz_xChangeWindowAttributesReq;
        if (data!=NULL)
            memptr->win[number].background = *(unsigned long *)data;

#ifdef TRACE
fprintf(stderr,"MC::    change_window_background    number:%d    win:0x%x
background:0x%x\n",
number,memptr->win[number].window,memptr->win[number].background);
```

```
#endif

    } /* end else */

} /* end change_window_background */
/*************************************************/
```

```
#include <sys/types.h>
#include <sys/param.h>
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <X11/X.h>
#define NEED_REPLIES
#define NEED_EVENTS
#include <X11/Xproto.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include "../includes/ds_manage.h"
#include "../includes/smtypes.h"
#include "../includes/smdef.h"
#include "../includes/dist.h"

extern  struct MC_SHMEMORY              *shmem;
extern  int     shmid;
#define TRUE    1
#define FALSE   0
main(argc,argv)
    int         argc;
    char        **argv;
{
    int             i;
    int             client;
    xCreateGCReq    CreateGC;
    int             first_time;

    first_time          = TRUE;
    client  = 10;

    CreateGC.reqType    = X_CreateGC;
    CreateGC.pad        = 0;
    CreateGC.length     = sz_xCreateGCReq;
    CreateGC.gc         = 0x100;
    CreateGC.drawable   = 0x200;
    CreateGC.mask       = 0;

    fprintf(stderr,"SwRI FAKE SERVER for client:%d\n",client);

#ifdef TRACE
fprintf(stderr,"Size of XGCValues struct:%d\n", sizeof(XGCValues));
fprintf(stderr,"Size    of    XSetWindowAttributes    struct:%d\n",
sizeof(XSetWindowAttributes));
#endif

#ifdef SLOW
fprintf(stderr,"Compiled with SLOW characteristics.\n");
#endif
#ifdef VSLOW
```

```
fprintf(stderr,"Compiled with VERY SLOW characteristics.\n");
#endif
    fprintf(stderr,"Initialized...\n");

    fprintf(stderr,"Multicast (Server Mod) Running...");
    while (1) {
        if (first_time) {
            fprintf(stderr,"FAKE SERVER creating shared memory.\n");
            first_time  = FALSE;
            multicast(client,&CreateGC,(int)sz_xCreateGCReq);
            fprintf(stderr,"FAKE SERVER going to sleep now.\n");
        }
        sleep(1);
        if (shmem->sm_status==DIE) {
            fprintf(stderr,"DUMMY: Told to go home...bye.\n");
            break;
        }
    }
    fprintf(stderr,"Done.\n");
/*
 * Get rid of the semaphore identifier.
 */
    for (i=0;i<XBUFFERNUM;i++)
        semctl(shmem->semaphore,i,IPC_RMID,0);

/*
 * Get rid of the shared memory identifier.
 */
    shmctl(shmid,IPC_RMID,0);

} /* end main */
```

APPENDIX L
PROTOCOL DISTRIBUTOR LISTINGS

```
/*************************************************************************
The included program listings are prototypes, no warranty is expressed or
implied for their use in any other fashion.  They should not be considered
or used as production software.  The information in the listings is
supplied on an "as is" basis.  No responsibility is assumed for damages
resulting from the use of any information contained in the listings.

The software in these listings has been compiled on Masscomp 6350's and
6600's and on Sun 3's and 4's.  Modifications may be necessary for use on
other systems.
*************************************************************************/

#define NUMBER_TIMER    100
/*
 * File          : pd.c
 * Author        : P. Fitzgerald - SwRI
 * Date          : 10/3/89
 * Description   :  This  file  contains  the  code  for  the  Protocol
Distributor.
 */
#include <stdio.h>
#include <rpc/rpc.h>
#include <sys/lock.h>
#include <utmp.h>
#include <sys/types.h>
#include <rpcsvc/rusers.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <fcntl.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <sys/wait.h>
#include <signal.h>
#include <errno.h>
#include <X11/X.h>
#define NEED_REPLIES
#define NEED_EVENTS
#include <X11/Xproto.h>
#include <X11/Xlib.h>
#include "../includes/ds_manage.h"
#include "../includes/smtypes.h"
#include "../includes/smdef.h"
#include "../includes/dist.h"
#include "../includes/xdefs.h"

/* EXTERNAL ROUTINES */
```

```
/* GLOBAL FUNCTIONS */
void            lock();
void            unlock();
void            go_away();
void            attach_shared_memory();
void            distribute_protocol();
int             timeout();
int             memory_check();

/* GLOBAL VARIABLES */
static int          last_time = 500;
struct MC_SHMEMORY  *shmem;
char                management_host[HOSTNAMLEN];
char                hostname[HOSTNAMLEN];
int                 semaphore;
int                 semaphore_locked[XBUFFERNUM];
int                 current_buffer  = 0;
int                 shmid;
unsigned short      pm_port = 0;
int                 distributor_id;
int                 pm_fd   = -1;

/*****************************************************************************/
/*
 * Main body
 */
main ()
{
    register struct MC_SHMEMORY *memptr;
    register int                semaphore_reg;
    register int                i;

/*
 * Tell everyone we are here
 */
    sleep(5);
    fprintf(stderr,"SwRI Protocol Distributor starting...\n");

/*
 * Set up to catch timer-timeout signals
 */
    signal(SIGALRM, timeout);
    signal(SIGUSR1, memory_check);
/*
 * Set up to catch kill signals
 */
    signal(SIGQUIT, go_away);

/*
 * Acquire the host name where we reside.
 */
```

```
    if ( gethostname(hostname,sizeof(hostname)) <0) {
        perror("PD::gethostname:");
        go_away();
    }

/*
 * Attach to the shared memory area.
 */
    attach_shared_memory();
    shmem->pd_alive = TRUE; /* say we are here */
    shmem->pd_pid   = getpid();
#ifdef TRACE
fprintf(stderr,"PD:: just put:%d in memory as my pid\n",getpid());
#endif
    for (i=0;i<XBUFFERNUM;i++)
        semaphore_locked[i] = FALSE;

#ifdef LOCKIT
/*
 * Lock self into memory
 */
    if (plock((int)PROCLOCK)<0)
        perror("PD:: plock(PROCLOCK):");
#endif

/*
 * Now wait until the Protocol Receiver is completed.
 */

#ifdef TRACE
fprintf(stderr,"PD:: Waiting on PR to initialize....\n");
#endif

    last_time   = 500;
    set_alarm(500);
    while (!shmem->pr_init) {
        if (shmem->sm_status==DIE)
            go_away();
        sleep(1);
    }
    clear_alarm();

#ifdef TRACE
fprintf(stderr,"PD:: pr just initialized\n");
#endif

/*
 * Grab the port number of the protocol Distributor so that we
 * may make a connection to him.
 */
    pm_port         = shmem->pm_port;
```

```
/*
 * Register self as a distributor
 */
#ifdef TRACE
fprintf(stderr,"registering self %d <%s>\n",pm_port,hostname);
#endif
    register_self(pm_port,hostname);
    shmem->distributor_id   = distributor_id;
    shmem->pd_init          = TRUE;

/*
 * Now loop on grabbing protocol out  shared memory and pumping
 * it to the Protocol Distributor.
 */
    semaphore_reg           = shmem->semaphore;
    memptr                  = shmem;

#ifdef TRACE
fprintf(stderr,"PD:: going into main processing loop.\n");
#endif

    while (1) {

/*
 * See if system is closing down.
 */
        if (memptr->sm_status==DIE) {
            fprintf(stderr,"PD::Requested to shut down.\n");
            go_away();
        }

/*
 * Make sure there is protocol in the buffer
 */
        if ( memptr->buf_stat[current_buffer] == MULTICAST ) {

/*
 * Lock the semaphore controlling access to shared memory area
 */
            lock(semaphore_reg);

/*
 * There is some X protocol to be distributed.
 */

/*
 * Call a routine to distribute the protocol.
 */
            distribute_protocol(pm_fd);

/*
 * Rotate the current buffer.
```

```
*/
            current_buffer++;
            if (current_buffer>=XBUFFERNUM)
              current_buffer  = 0;

        } /* end if MULTICAST */
        else {
            sleep(1);
        }
    } /* end while */

} /* end main */
/*****************************************************************************/


/*****************************************************/
/*
 * This routine replaces a call to exit() to do cleanup.
 */
void
go_away()
{
    int             i;

/*
 * Tell someone we are going away.
 */
    fprintf(stderr,"PD:: EXITING.....\n");
    for (i=0;i<XBUFFERNUM;i++) {
        current_buffer  = i;

        if (semaphore_locked[i])
            unlock(shmem->semaphore);
    }
    sleep(2);
    exit();
} /* end go_away */
/*****************************************************/

/*****************************************************************************/
/*
 * This is a timeout routine to say that we got hung up waiting on
 * someting to happen.
 */
int
timeout()
{

    fprintf(stderr,"PD:: Timeout performing a read or write.\n");

/*
 * Set up to catch timer-timeout signals
```

```
 */
    signal(SIGALRM, timeout);

    if (shmem->sm_status==DIE)
        go_away();
#ifdef TRACE
fprintf(stderr,"PD:: setting alarm back to:%d\n",last_time);
#endif
    set_alarm(last_time);

} /* end timeout */
/*************************************************************************/

/*************************************************************************/
/*
 * This routine is called when it receives a SIGUSR1 signal to
 * check shared memory for requests from the Protocol Receiver.
 */
int
memory_check()
{

/*
 * Set us to ignore further signals for now
 */
    signal(SIGUSR1,SIG_IGN);

#ifdef TRACE
fprintf(stderr,"PD:: memory check called as a result of signal\n");
#endif

/*
 * Call the routine to check what is happening in shared memory
 * to see if Receiver is requesting anything.
 */
    check_shared_memory();

/*
 * Reset the signal handler for this type of signal
 */
    signal(SIGUSR1,memory_check);

#ifdef TRACE
fprintf(stderr,"PD:: memory check exiting\n");
#endif

} /* end memory_check */
```

```
/*************************************************************************/
/*
 * File          : pdio.c
 * Author        : P. Fitzgerald - SwRI
 * Date          : 10/3/89
 * Description   :  This  file  contains  the  code  for  the  Protocol
Distributor.
 * I/O related routines.
 */
#include <stdio.h>
#include <rpc/rpc.h>
#include <utmp.h>
#include <sys/types.h>
#include <rpcsvc/rusers.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <fcntl.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <sys/wait.h>
#include <signal.h>
#include <errno.h>
#include <X11/X.h>
#define NEED_REPLIES
#define NEED_EVENTS
#include <X11/Xproto.h>
#include <X11/Xlib.h>
#include "../includes/ds_manage.h"
#include "../includes/smtypes.h"
#include "../includes/smdef.h"
#include "../includes/dist.h"

/* EXTERNAL ROUTINES */

/* GLOBAL FUNCTIONS */
void            lock();
void            unlock();
void            get_window_attributes();
void            get_graphics_context();
int             connect_to();
int             distributable();
void            attach_shared_memory();
void            distribute_protocol();

/* GLOBAL VARIABLES */
extern  struct MC_SHMEMORY  *shmem;
extern  char                management_host[HOSTNAMLEN];
extern  char                hostname[HOSTNAMLEN];
extern  int                 current_buffer;
extern  int                 shmid;
```

```
extern  int                 pm_fd;
extern  char                XFunctName[][80];


/*****************************************************/
/*
 * This function is called to connect to a particular
 * port number on a remote machine.
 * It returns a file descriptor.
 */
int
connect_to(port)
    register unsigned short     port;
{
    register struct hostent     *hp;
    register int                fd;
    static struct sockaddr_in   sinhim = { AF_INET };

#ifdef TRACE
fprintf(stderr,"PD:: Connect_to called\n");
#endif

/*
 * If the port number is not set yet, get it
 */
    if (port==0)
        port    = shmem->pm_port;

#ifdef TRACE
fprintf(stderr,"PD:: Connecting to port:%d\n",port);
#endif

/*
 * Get management host name out of shared memory.
 */
    strncpy(management_host,shmem->management_host,HOSTNAMLEN);

#ifdef TRACE
fprintf(stderr,"PD::managment_host:<%s>\n",management_host);
#endif

/*
 * Now get hostname, address, and connect to the Multiplexer.
 */
    hp  = gethostbyname(management_host);
    if (!hp) {
        fprintf(stderr,"PD::Host '%s' not found\n",management_host);
        go_away();
    }

#ifdef TRACE
fprintf(stderr,"PD::port number->%d\n",port);
```

```
#endif
    bcopy(hp->h_addr, &sinhim.sin_addr, sizeof(sinhim.sin_addr));
    sinhim.sin_port = htons(port);
    fd  = socket(AF_INET,SOCK_STREAM,0);
    if (fd<0) {
        perror("PD::socket (connect_to):");
        return(-1);
    }
#ifdef TRACE
fprintf(stderr,"PD::socket created.\n");
#endif
    if (connect(fd,&sinhim,sizeof(sinhim))<0) {
        perror("PD::connect (connect_to):");
        return(-1);
    }

#ifdef TRACE
fprintf(stderr,"PD::Socket and connect to PM ok.\n");
#endif

    return(fd);

} /* end connect_to */
/*****************************************************/


/*****************************************************/
/*
 * This is the main routine which takes protocol out of
 * shared memory and sends it to the Protocol Multiplexer.
 */
void
distribute_protocol(fd)
    register    int         fd;
{
    register    int         bytes_written;
    register    union       COMPAK  *cp_reg;
    register    int         length;
    union       COMPAK      cp;
    register int bytes_to_do;
    register unsigned char *bufptr;
    int count;

    count = 0;

    bytes_to_do = shmem->len[current_buffer];
    bufptr = shmem->xbuffer[current_buffer];

/*
 * Set up the structure containing all the information.
 */
    cp_reg                      = &cp;
```

```
    cp_reg->pdtopm.signal[0]    = X_DATA;
    cp_reg->pdtopm.signal[1]    = 0x0;
    cp_reg->pdtopm.signal[2]    = 0x0;
    cp_reg->pdtopm.signal[3]    = 0x0;
    cp_reg->pdtopm.header.client    = shmem->client[current_buffer];

    while ( bytes_to_do > 0 ) {

        count++;

        length = get_length(bufptr);

/*
 * Now determine whether or not the protocol is to be
 * distributed.
 */
        if (distributable(bufptr)) {

            cp_reg->pdtopm.header.length    = length;
            bcopy(bufptr,cp_reg->pdtopm.buffer,length);

#ifdef DATA
fprintf(stderr,"PD::    just    copied    out    some    protocol    from
buffer:%d\n",current_buffer);
fprintf(stderr,".... getting ready to write packet to PM\n");
#endif

/*
 * Now write the buffer to the multiplexer.
 */
            bytes_written    =    netwrite(fd,(unsigned    char
*)cp_reg->compak,(int)PAKLEN+length);

#ifdef DATA
fprintf(stderr,"PD::just    wrote    an    X    packet    of    %d
bytes,client:%d.\n",bytes_written,cp_reg->pdtopm.header.client);
#endif
#ifdef SLOW
sleep(1);
#endif

            if (bytes_written!=(int)(PAKLEN+length)) {
#ifdef GO_AWAY
                go_away();
#endif
            }
        }  /* end if distributable */

        bytes_to_do -= length;
        bufptr += length;

    } /* end while bytes_to_do > 0 */
```

```
/*
 * Unlock the semaphore now.
 */
#ifdef INTENSE
fprintf(stderr,"PD:: setting buffer %d to SM_EMPTY\n",current_buffer);
#endif

    shmem->buf_stat[current_buffer] = SM_EMPTY;
    unlock(shmem->semaphore);

#ifdef TRACE
if ( count != 1 )
 fprintf(stderr,"(%d)",count);
#endif
} /* end distribute_protocol */
/*****************************************************/


/*****************************************************/
/*
 *  This routine returns the length of the particular
 * protocol package. Note that the length is the second
 * two bytes of the packet, in terms of 32 bit
 * quantities. We left shift by two to get the byte
 * count.
 */
int
get_length(ptr)
    register unsigned char *ptr;
{
    register unsigned short *shortptr;
    register unsigned short length;

    shortptr = (unsigned short *)(ptr+2);
    length   = (*shortptr<<2);

    return(length);

} /* end get length */
/*****************************************************/

/*****************************************************/
/*
 * This routine sends an expose event back to the
 * multiplexer, along with the source port number.
 */
send_expose_event(fd,window)
    register int            fd;
    register XID            window;
{
    register int            bytes_written;
```

```
    unsigned char           signal_bytes[SIGLEN+sizeof(XID)];
    XID                     *ptr;

#ifdef TRACE
fprintf(stderr,"PD:: send_expose_event back to pm.\n");
#endif

/*
 * Set up the parameters.
 */
    signal_bytes[0]         = EXPOSE;
    signal_bytes[1]         = 0;
    signal_bytes[2]         = 0;
    signal_bytes[3]         = 0;
    ptr                     = (XID *)&signal_bytes[4];
    *ptr                    = window;
    bytes_written   = netwrite(fd,signal_bytes,SIGLEN+sizeof(XID));
    if (bytes_written!=SIGLEN+sizeof(XID) ) {
        perror("PD:: write (send_expose_event):");
        fprintf(stderr,"PD:: bytes_written:%d length:%d fd:%d\n",
                bytes_written,SIGLEN+sizeof(XID),fd);
#ifdef GO_AWAY
        go_away();
#endif
    }
    shmem->pd_propagate_expose  = FALSE;


#ifdef TRACE
fprintf(stderr,"PD:: send expose to remote, window: 0x%x\n",window);
fprintf(stderr,"PD:: send_expose_event complete.\n");
#endif

} /* end send_expose_event */
/*****************************************************/


/*****************************************************/
/*
 * This routine makes a request of the Protocol Multiplexer
 * to send back information concerning the window attributes
 * for a particular channel.
 */
void
get_window_attributes(fd)
    register    int     fd;
{
    register short      *shortptr;
    union   COMPAK      cp;
    int                 bytes_written;

#ifdef TRACE
```

```
fprintf(stderr,"PD:: GET_WINDOW_ATTRIBUTES noticed in shmem\n");
#endif

/*
 * Set up the parameters to request Window Attributes state info
 */
    cp.pdtopm.signal[0] = GWATS;
    cp.pdtopm.signal[1] = 0;

    /* Send the Receiver's port number with request */
    shortptr            = (short *)&cp.pdtopm.signal[2];
    *shortptr           = shmem->wat_port;

    /* Use the length for the channel number */
    cp.pdtopm.header.length = shmem->wat_channel;

    /* Use the client for the graphics context id */
    cp.pdtopm.header.client = shmem->wat_id;

#ifdef DATA
fprintf(stderr,"PD:: get_window_attributes for wid:0x%x channel:%d\n",
shmem->wat_id,shmem->wat_channel);
#endif

/*
 * Now write that to the Protocol Multiplexer
 */
    bytes_written = netwrite(fd,&cp,PAKLEN);
    if (bytes_written != PAKLEN) {
        perror("read (get_window_attributes):");
        fprintf(stderr,"PD:: bytes_written:%d length:%d fd:%d.\n",
                bytes_written,PAKLEN,fd);
#ifdef GO_AWAY
        go_away();
#endif
    }

#ifdef TRACE
fprintf(stderr,"PD:: wrote get window attributes request to PM.\n");
#endif

/*
 * Notify the Protocol Receiver that we have made the request of the
 * Multiplexer. It is now time for the Receiver to wait on the results.
 */
    shmem->get_wat  = FALSE;

} /* end get_window_attributes */
/*****************************************************/

/*****************************************************/
/*
```

```
 * This routine makes a request of the Protocol Multiplexer
 * to send back information concerning the state information
 * for a particular gc.
 */
void
get_graphics_context(fd)
    register    int     fd;
{
    register short      *shortptr;
    union   COMPAK      cp;
    int                 bytes_written;

#ifdef TRACE
fprintf(stderr,"PD:: GET_GRAPHICS_CONTEXT noticed in shmem\n");
#endif

/*
 * Set up the parameters to request GC state information
 */
    cp.pdtopm.signal[0] = GGCS;
    cp.pdtopm.signal[1] = 0;

    /* Send the Receiver's port number with request */
    shortptr            = (short *)&cp.pdtopm.signal[2];
    *shortptr           = shmem->gc_port;

    /* Use the length for the channel number */
    cp.pdtopm.header.length = shmem->gc_channel;

    /* Use the client for the graphics context id */
    cp.pdtopm.header.client = shmem->gc_id;

#ifdef TRACE
fprintf(stderr,"PD:: get_graphics_context for gc:0x%x channel:%d\n",
shmem->gc_id,shmem->gc_channel);
fprintf(stderr,"PD:: fd:%d\n",fd);
#endif

/*
 * Now write that to the Protocol Multiplexer
 */
    bytes_written = netwrite(fd,&cp,PAKLEN);
    if (bytes_written != PAKLEN) {
        perror("PD:: write (get_graphics_context):");
        fprintf(stderr,"PD:: bytes_written:%d length:%d fd:%d.\n",
                bytes_written,PAKLEN,fd);
#ifdef GO_AWAY
        go_away();
#endif
    }

#ifdef TRACE
```

```
fprintf(stderr,"PD:: wrote get graphics context request to PM.\n");
fprintf(stderr,"PD:: bytes_written was :%d\n",bytes_written);
fprintf(stderr,"PD:: written to:%d  pm_fd:%d\n",fd,pm_fd);
#endif

/*
 * Notify the Protocol Receiver that we have made the request of the
 * Multiplexer. It is now time for the Receiver to wait on the results.
 */
    shmem->get_gc    = FALSE;

} /* end get_graphics_context */
/*****************************************************/

/*****************************************************/
/*
 * This routine is called whenever a request is received
 * from the Protocol Receiver to send out a particular window
 * attributes state.
 */
void
send_window_attributes(fd)
    register    int     fd;
{
    register        int     bytes_written;
    register        short   *shortptr;
    register        unsigned long   *longptr;
    unsigned char   signal_bytes[SIGLEN+8];

/*
 * Write out the signal bytes indicating what we are sending.
 */
#ifdef TRACE
fprintf(stderr,"PD:: sending out a WATS buddy!!!!!!\n");
#endif
    signal_bytes[0]     = WATS;
    signal_bytes[1]     = 0;

    /* Place the Receiver's port number there */
    shortptr            = (short *)&signal_bytes[2];
    *shortptr           = (short)shmem->wat_port;

    /* Now place the background pixel value */
    longptr             = (unsigned long *)&signal_bytes[4];
    *longptr            = shmem->wat_bg_pixel;

#ifdef TRACE
fprintf(stderr,"PD::    JUST    SENT    BACKGROUND    wat_bg_pixel
of:0x%x\n",*longptr);
#endif

    /* Now place the parent XID value */
```

```
    longptr             = (unsigned long *)&signal_bytes[8];
    *longptr            = shmem->wat_parent;

#ifdef TRACE
fprintf(stderr,"PD:: JUST SENT PARENT wat_parent:0x%x\n",*longptr);
#endif

    bytes_written   = netwrite(fd,signal_bytes,SIGLEN+8);
    if (bytes_written != SIGLEN+8) {
        perror("PD:: write (send_window_attributes/signal):");
        fprintf(stderr,"PD:: bytes_written:%d length:%d fd:%d.\n",
            bytes_written,SIGLEN+8,fd);
#ifdef GO_AWAY
        go_away();
#endif
    }

#ifdef TRACE
fprintf(stderr,"PD:: just wrote signal bytes for WATS\n");
fprintf(stderr,"PD::........0x%x 0x%x 0x%x 0x%x\n",
signal_bytes[0],
signal_bytes[1],
signal_bytes[2],
signal_bytes[3]);
#endif

/*
 * Write those values out to the file descriptor
 */
    bytes_written = netwrite(fd,&shmem->wats,sizeof(XWindowAttributes));
    if (bytes_written != sizeof(XWindowAttributes) ) {
        perror("PD:: write (send_window_attributes/XWindowAttributes):");
        fprintf(stderr,"PD:: bytes_written:%d length:%d fd:%d.\n",
            bytes_written,sizeof(XWindowAttributes),fd);
#ifdef GO_AWAY
        go_away();
#endif
    }
#ifdef TRACE
fprintf(stderr,"PD::   just   wrote   window   attributes
bytes:%d\n",bytes_written);
#endif

/*
 * Clear the flag and let the PR know we did it
 */
    shmem->send_wat     = FALSE;

} /* end send_window_attributes */
/*****************************************************/

/*****************************************************/
```

```
/*
 * This routine is called whenever a request is received
 * from the Protocol Receiver to send out a particular
 * graphics context state.
 */
void
send_graphics_context(fd)
    register        int     fd;
{
    register        int             i;
    register        int             index;
    register        int             bytes_written;
    register        unsigned short  *shortptr;
    XID                             window;
    unsigned char                   signal_bytes[SIGLEN+4];

#ifdef TRACE
fprintf(stderr,"PD::  SEND_GRAPHICS_REQUEST  for  STATE  for  id:0x%x
port:%d\n",
shmem->gc_id,shmem->gc_port);
#endif

/*
 * Search the shared memory area for the particular
 * xid entry.
 */
    index   = -1;
    for (i=0;i<MAX_GCS;i++) {
        if (shmem->gcwin[i].gid==shmem->gc_id) {
            index   = i;
            break;
        }
    } /* end for */

/*
 * If we cannot find the proper one
 */
    if (index<0) {
        fprintf(stderr,"PD:: Invalid XID for GGCS:0x%x\n",shmem->gc_id);
for ( i = 0; i < MAX_GCS; i++ )
 fprintf(    stderr,    "...shmem->gcwin[%d].gid    :    0x%x\n",    i,
shmem->gcwin[i].gid );
        shmem->gc_id        = 0;
        shmem->gc_channel   = -1;
#ifdef GO_AWAY
        go_away();
#endif
    }
    else {
/*
 * Write out the signal bytes indicating what we are sending.
 */
```

```
#ifdef TRACE
fprintf(stderr,"PD:: sending out a GCS buddy!!!!!!\n");
#endif
        signal_bytes[0]     = GCS;
        signal_bytes[1]     = 0;
        /* Place the Receiver's port number there */
        shortptr            = (unsigned short *)&signal_bytes[2];
        *shortptr           = (unsigned short)shmem->gc_port;
#ifdef TRACE
fprintf(stderr,"PD:: shmem->gc_port sending is:%d\n",shmem->gc_port);
#endif
        bytes_written   = netwrite(fd,signal_bytes,SIGLEN);
        if (bytes_written != SIGLEN) {
            perror("PD:: write (send_graphics_context/signal):");
            fprintf(stderr,"PD:: bytes_written:%d length:%d fd:%d.\n",
                bytes_written,SIGLEN,fd);
#ifdef GO_AWAY
            go_away();
#endif
        }
#ifdef TRACE
fprintf(stderr,"PD:: just wrote signal bytes for GCS\n");
fprintf(stderr,"PD::........0x%x 0x%x 0x%x 0x%x\n",
signal_bytes[0],
signal_bytes[1],
signal_bytes[2],
signal_bytes[3]);
#endif

/*
 * Write those values out to the file descriptor
 */
        bytes_written = netwrite(fd,&shmem->gcwin[index].GCValues,
                                 sizeof(XGCValues) );
        if (bytes_written != sizeof(XGCValues) ) {
            perror("PD:: write (send_graphics_context/GCValues):");
            fprintf(stderr,"PD:: bytes_written:%d length:%d fd:%d.\n",
                bytes_written,sizeof(XGCValues),fd);
#ifdef GO_AWAY
            go_away();
#endif
        }

#ifdef TRACE
fprintf(stderr,"PD:: just wrote graphics context bytes:%d (index:%d)\n",
bytes_written,index);
fprintf(stderr,"PD::    ...fg:    %d,    bg:    %d\n",
shmem->gcwin[index].GCValues.foreground,

shmem->gcwin[index].GCValues.background );
#endif
```

```
/*
 * Now retrieve the XID of the window associated with the gc
 */
        window = shmem->gcwin[index].window;

#ifdef TRACE
fprintf(stderr,"PD::    original    gc    is
0x%x\n",shmem->gcwin[index].gid);
fprintf(stderr,"PD:: associated window is 0x%x\n",window);
#endif

        bytes_written = netwrite(fd,&window,sizeof(window));
        if (bytes_written != sizeof(window) ) {
            perror("PD:: write (send_graphics_context/window):");
            fprintf(stderr,"PD:: bytes_written:%d length:%d fd:%d.\n",
                bytes_written,sizeof(window),fd);
#ifdef GO_AWAY
            go_away();
#endif
        }

#ifdef TRACE
fprintf(stderr,"PD:: just wrote window bytes of:%d\n",bytes_written);
#endif

    } /* end else */

/*
 * Let the Protocol Receiver know we fulfilled his request
 */
    shmem->send_gc      = FALSE;

} /* end send_graphics_context */
/*****************************************************/
```

```
#define NUMBER_TIMER    100
/*
 * File         : pdutil.c
 * Author       : P. Fitzgerald - SwRI
 * Date         : 10/3/89
 * Description  :  This  file  contains  the  code  for  the  Protocol
Distributor.
 * Utility routines and X routines
 */
#include <stdio.h>
#include <rpc/rpc.h>
#include <utmp.h>
#include <sys/types.h>
#include <rpcsvc/rusers.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <fcntl.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <sys/wait.h>
#include <signal.h>
#include <errno.h>
#include <X11/X.h>
#define NEED_REPLIES
#define NEED_EVENTS
#include <X11/Xproto.h>
#include <X11/Xlib.h>
#include "../includes/ds_manage.h"
#include "../includes/smtypes.h"
#include "../includes/smdef.h"
#include "../includes/dist.h"

/* EXTERNAL ROUTINES */
extern          xdr_RegDist();
extern int      connect_to();

/* GLOBAL FUNCTIONS */
static int      semcall();
void            lock();
void            unlock();
int             distributable();
int             callme();
void            go_away();
void            attach_shared_memory();
void            check_shared_memory();

/* GLOBAL VARIABLES */
extern  struct MC_SHMEMORY  *shmem;
extern  int                 semaphore_locked[XBUFFERNUM];
extern  int                 current_buffer;
```

```
extern  int             shmid;
extern  int             pm_fd;
extern  int             distributor_id;
extern  char            XFunctName[][80];


/*********************************************************************/
/*
 * This routine attaches to the shared memory area used between
 * the multicast (server mod) function, ldm, and protocol distributor
(me).
 * We will keep trying to attach for an awfully long time before giving
up.
 */
void
attach_shared_memory()
{
    alarm(500);

    shmid   = -1;
    while (shmid<0) {
        /* attach to shared memory */
        shmid   = shmget( (int)SM_KEY,
                sizeof(struct MC_SHMEMORY),0777);
        sleep(1);
    }
    shmem   = (struct MC_SHMEMORY *)shmat(shmid,0,0);
    if (shmem == (struct MC_SHMEMORY *)-1 ) {
        perror("PD::Unable to attach to shared memory.");
        go_away();
    }

    clear_alarm();

#ifdef TRACE
printf("PD::attached to shared memory 0x%x\n",shmem);
#endif

} /* end attach_shared_memory */
/*********************************************************************/

/*****************************************************/
/*
 * This routine locks on a semaphore.
 */
void
lock(id)
    register int        id;
{

    if (semaphore_locked[current_buffer]) {
        fprintf(stderr,"PD:: OOPS, forgot to unlock a semaphore:%d\n",
```

```
            current_buffer);
        return;
    }

/*
 * We will loop here until we are able to lock the semaphore.
 * This will allow us to process other events and requests
 * besides just protocol to distribute.
 */
#ifdef INTENSE
fprintf(stderr,"PD:: going to lock buffer:%d\n",current_buffer);
#endif
    semcall(id,-1);
    semaphore_locked[current_buffer]    = TRUE;

#ifdef INTENSE
fprintf(stderr,"PD:: just locked buffer:%d\n",current_buffer);
#endif

} /* end lock */
/*******************************************************/
/*******************************************************/
/*
 * This routine unlocks a semaphore.
 */
void
unlock(id)
    register int        id;
{
    if (!semaphore_locked[current_buffer]) {
        fprintf(stderr,"PD::   OOPS,   called   unlock   when   already
unlocked:%d\n",
            current_buffer);
        return;
    }

    semcall(id,1);
    semaphore_locked[current_buffer]    = FALSE;

#ifdef INTENSE
fprintf(stderr,"PD:: just unlocked buffer:%d\n",current_buffer);
#endif

} /* end unlock */
/*******************************************************/
/*******************************************************/
/*
 * This routine performs the semaphore operations.
 */
static int
semcall(sid,op)
    register int        sid;
```

```
    register int        op;
{
    struct sembuff  sb;

    sb.sem_num  = current_buffer;
    sb.sem_op   = op;
    sb.sem_flg  = 0;

    return(semop(sid,&sb,1));

} /* end semcall */
/*****************************************************/

/*****************************************************/
/*
 * This routine determines if the protocol passed in
 * is for distribution or ignoring.
 */
int
distributable(xptr)
    register    unsigned char   *xptr;
{
    register    unsigned char   xtype;

    xtype   = *xptr;

/*
 * Select X protocol for distribution or ignoring.
 */
    switch(xtype) {

/* IGNORE - RETURN FALSE */
        case X_GetWindowAttributes:
        case X_ChangeSaveSet:
        case X_ReparentWindow:
        case X_CirculateWindow:
        case X_GetGeometry:
        case X_QueryTree:
        case X_InternAtom:
        case X_GetAtomName:
        case X_ChangeProperty:
        case X_GetProperty:
        case X_DeleteProperty:
        case X_ListProperties:
        case X_SetSelectionOwner:
        case X_GetSelectionOwner:
        case X_ConvertSelection:
        case X_SendEvent:
        case X_GrabPointer:
        case X_UngrabPointer:
        case X_GrabButton:
        case X_UngrabButton:
```

```
case X_ChangeActivePointerGrab:
case X_GrabKeyboard:
case X_UngrabKeyboard:
case X_GrabKey:
case X_UngrabKey:
case X_AllowEvents:
case X_GrabServer:
case X_UngrabServer:
case X_QueryPointer:
case X_GetMotionEvents:
case X_TranslateCoords:
case X_WarpPointer:
case X_SetInputFocus:
case X_GetInputFocus:
case X_QueryKeymap:
case X_QueryTextExtents:
case X_ListFonts:
case X_ListFontsWithInfo:
case X_GetFontPath:
case X_SetFontPath:
case X_SetDashes:                /* ? */
case X_GetImage:
case X_InstallColormap:
case X_UninstallColormap:
case X_ListInstalledColormaps:
case X_QueryColors:
case X_LookupColor:
case X_CreateCursor:             /* ? */
case X_CreateGlyphCursor:        /* ? */
case X_FreeCursor:               /* ? */
case X_RecolorCursor:            /* ? */
case X_QueryBestSize:
case X_QueryExtension:
case X_ListExtensions:
case X_ChangeKeyboardMapping:
case X_GetKeyboardMapping:
case X_ChangeKeyboardControl:
case X_GetKeyboardControl:
case X_Bell:
case X_ChangePointerControl:
case X_GetPointerControl:
case X_SetScreenSaver:
case X_GetScreenSaver:
case X_ChangeHosts:
case X_ListHosts:
case X_SetAccessControl:
case X_SetCloseDownMode:
case X_RotateProperties:
case X_ForceScreenSaver:
case X_SetPointerMapping:
case X_GetPointerMapping:
case X_SetModifierMapping:
```

```
        case X_GetModifierMapping:
        case X_NoOperation:
        case X_AllocColor:                      /* ? */
        case X_AllocNamedColor:                 /* ? */
        case X_AllocColorCells:                 /* ? */
        case X_AllocColorPlanes:                /* ? */
/* Currently we are IGNORING:   78 operations */
/* ifdef TRACE */
fprintf(stderr,"PD:: IGNORING...<%s>.\n",XFunctName[xtype]);
/* endif */
            return(FALSE);

/* DISTRIBUTE - RETURN TRUE */
        case X_CreateWindow:
        case X_ChangeWindowAttributes:
        case X_DestroyWindow:
        case X_DestroySubwindows:
        case X_MapWindow:
        case X_MapSubwindows:
        case X_UnmapWindow:
        case X_UnmapSubwindows:
        case X_ConfigureWindow:
        case X_OpenFont:
        case X_CloseFont:
        case X_CreatePixmap:
        case X_FreePixmap:
        case X_CreateGC:
        case X_ChangeGC:
        case X_CopyGC:
        case X_SetClipRectangles:
        case X_FreeGC:
        case X_ClearArea:
        case X_CopyArea:
        case X_CopyPlane:
        case X_PolyPoint:
        case X_PolyLine:
        case X_PolySegment:
        case X_PolyRectangle:
        case X_PolyArc:
        case X_FillPoly:
        case X_PolyFillRectangle:
        case X_PolyFillArc:
        case X_PutImage:
        case X_PolyText8:
        case X_PolyText16:
        case X_ImageText8:
        case X_ImageText16:
        case X_CreateColormap:
        case X_FreeColormap:
        case X_CopyColormapAndFree:
        case X_FreeColors:                      /* ? */
        case X_StoreColors:                     /* ? */
```

```
        case X_StoreNamedColor:              /* ? */
        case X_KillClient:
/* Currently we are Processing: 51 operations. */
#ifdef DATA
fprintf(stderr,"PD:: PROCESSING...<%s>.\n",XFunctName[xtype]);
#endif
            return(TRUE);
        default:
            fprintf(stderr,"PD::   Unknown   type   of   X
protocol:0x%x\n",xtype);
#ifdef GO_AWAY
            go_away();
#endif
            break;
    } /* end switch */
    return(FALSE);

} /* end distributable */
/*****************************************************/

/*****************************************************/
/*
 * This routine registers this distributor with the
 * Central Distribution Manager, who in turn, notifies
 * the Protocol Multiplexer of this request.
 */
void
register_self(port,hostname)
    int     port;
    char    *hostname;
{
    int                 retval;
    struct DistRegister DistRegister;

#ifdef TRACE
fprintf(stderr,"PD:: register self port:%d  hostname:<%s>\n",
port,hostname);
#endif

/*
 * First register self with the Central Distributotion
 * Manager
 */
    sprintf(DistRegister.distname,"%s",hostname);
    retval  = clnt_broadcast(CDM_PROG,CDM_VERS,CDM_REG_DIST,
                             xdr_RegDist,&DistRegister,
                             xdr_RegDist,&DistRegister,callme);
    distributor_id  = DistRegister.distributor_id;

#ifdef TRACE
fprintf(stderr,"PD:: register self <%s>: returned an id of %d\n",
hostname,
```

```
distributor_id);
#endif
/*
 * Now connect with the Protocol Multiplexer
 */
    pm_fd   = connect_to( (unsigned short)port);
    set_no_block(pm_fd);

#ifdef TRACE
fprintf(stderr,"PD:: connected to:%d\n",pm_fd);
#endif

} /* end register_self */
/*****************************************************/


/*****************************************************/
/*
 * Short and simple routine to perform callback on the RPC
 * return call.
 */
int
callme (out,addr)
    char                *out;
    struct sockaddr_in  *addr;
{
    return(1);
} /* end callme */
/*****************************************************/

/*****************************************************/
/*
 * This routine performs checks on various shared
 * memory variables and flags to determine if the
 * Protocol Receiver is requesting any work to be
 * performed, or if the system is dying.
 */
void
check_shared_memory()
{
    register struct MC_SHMEMORY *memptr;

    memptr  = shmem;

/*
 * Check to see if we are to exit
 */
        if (memptr->sm_status==DIE)
            go_away();

/*
 * Now check to see if the Protocol Receiver needs us to request some
```

```
 * gc or window values to/from the Protocol Multiplexer.
 */
        if (memptr->get_wat) {
#ifdef TRACE
fprintf(stderr,"PD:: pr has asked for a get_wat\n");
#endif
            get_window_attributes(pm_fd);
        }
        if (memptr->get_gc) {
#ifdef TRACE
fprintf(stderr,"PD:: pr has asked for a get_gc\n");
#endif
            get_graphics_context(pm_fd);
        }
        if (memptr->send_wat) {
#ifdef TRACE
fprintf(stderr,"PD:: pr has asked for a send_wat\n");
#endif
            send_window_attributes(pm_fd);
        }
        if (memptr->send_gc) {
#ifdef TRACE
fprintf(stderr,"PD:: pr has asked for a send_gc\n");
#endif
            send_graphics_context(pm_fd);
        }

/*
 * If propagate expose event is TRUE, then send that on to the
multiplexer.
 */
        if (memptr->pd_propagate_expose) {
#ifdef TRACE
fprintf(stderr,"PD:: The pr has requested I send an expose event
window:0x%x\n",
memptr->expose_window);
#endif
            send_expose_event(pm_fd,memptr->expose_window);
        }

} /* end check_shared_memory */
/*************************************************/

/*****************************************************************/
/*
 * This routine simply sets O_NDELAY attribute on file descriptor.
 */
set_no_block(fd)
    register int    fd;
{
    register int    flags;
```

```
    if ((flags=fcntl(fd,F_GETFL,0))==-1) {
        perror("PD::fcntl (set_no_block-F_GETFL):");
#ifdef GO_AWAY
        go_away();
#endif
    }
    flags |= (FNDELAY);
    if (fcntl(fd,F_SETFL,flags)<0) {
        perror("PD::fcntl (set_no_block-F_SETFL):");
#ifdef GO_AWAY
        go_away();
#endif
    }

} /* end set_no_block */
/*****************************************************************************/
```

```
/*
 * File          : alarm.c
 * Author        : P. Fitzgerald - SwRI
 * Date          : 11/30/89
 * This file contains a set alarm and a clear alarm routine.
 */
#include <stdio.h>
#include <sys/types.h>
#include <fcntl.h>
#include <signal.h>
#include <errno.h>

/*************************************************************************/
/*
 * This routine sets an alarm.
 */
void
set_alarm(secs)
    int         secs;
{
    if (alarm(secs)<0) {
        perror("PD:: alarm (set_alarm):");
#ifdef GO_AWAY
        go_away();
#endif
    }

} /* end set_alarm */
/*************************************************************************/


/*************************************************************************/
/*
 * This routine clears an alarm.
 */
void
clear_alarm()
{
    if (alarm(0)<0) {
        perror("PD:: alarm (clear_alarm):");
#ifdef GO_AWAY
        go_away();
#endif
    }

} /* end clear_alarm */
/*************************************************************************/
```

```
/*
 * Name          : mutil.c
 * Author        : P. Fitzgerald - SwRI
 * Date          : 10/3/89
 * Description   : This file contains utility subroutines used by various
 * local and central management functions.
 *
 *  Routines:
 *              xdr_PortID()
 *              xdr_tvguide();
 *              xdr_DistAuth();
 *              xdr_ChanID();
 *              xdr_RecvAuth();
 *              xdr_ChanReq();
 *              xdr_RemvRecv();
 *              xdr_RegDist();
 *              xdr_RegRecv();
 */

/*
#include <rpc/rpc.h>
*/
#include "../includes/rpc.h"
#include <X11/X.h>
#define NEED_REPLIES
#define NEED_EVENTS
#include <X11/Xproto.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include "../includes/ds_manage.h"

/*****************************************************************/
/*
 * This routine handles xdr translations of a particular type of
 * data structure which holds host names and port numbers. Each Protocol
 * Receiver, when it comes on line, registers its hostname and port
 * number with the CDM RPC server.
 */
xdr_PortID(xdrsp, pid)
    register XDR                *xdrsp;
    register struct PortID  *pid;
{
    char            *ptr;

/*
 * We need to just verify the various components of
 * the structure.
 */
    ptr = pid->hostname;
    if (!xdr_string(xdrsp,&ptr,PORTNAMLEN))
        return(0);
```

```
    if (!xdr_u_short(xdrsp,&pid->portnum))
        return(0);

    return(1);
} /* end xdr_PortID */
/*****************************************************************/


/*****************************************************************/
xdr_tvguide(xdrsp,tv_guide)
    register XDR                *xdrsp;
    register char             **tv_guide;
{
    u_int           sizep,maxsize;
    unsigned char   *ptr;

    sizep   = CHANNAMLEN * MAX_CHANNELS;
    maxsize = CHANNAMLEN * MAX_CHANNELS;

    ptr = (unsigned char *)tv_guide;
    if (!xdr_bytes(xdrsp,&ptr,&sizep,maxsize))
        return(0);

    return(1);

} /* end xdr_tvguide */
/*****************************************************************/


/*****************************************************************/
/*
 * This routine translates a Distribution Authorization return from
 * a DA request.
 */
xdr_DistAuth(xdrsp,ar)
    register XDR                *xdrsp;
    register struct DistAuth    *ar;
{
/*
 * We need to just verify the various components of
 * the structure.
 */
    if (!xdr_int(xdrsp,&ar->authorization))
        return(0);

    if (!xdr_int(xdrsp,&ar->channel))
        return(0);

    if (!xdr_u_short(xdrsp,&ar->pm_port))
        return(0);

    return(1);
```

```
} /* end xdr_DistAuth */
/*************************************************************************/

/*************************************************************************/
/*
 * This routine handles conversions of the ChanID structure which
 * is sent for a distribution request.
 */
xdr_ChanID(xdrsp,cid)
    register XDR                *xdrsp;
    register struct ChanID  *cid;
{
    char            *ptr;

/*
 * We need to just verify the various components of
 * the structure.
 */
    ptr = cid->chanid;
    if (!xdr_string(xdrsp,&ptr,SOURCENAMLEN))
        return(0);

    ptr = cid->hostname;
    if (!xdr_string(xdrsp,&ptr,HOSTNAMLEN))
        return(0);

    if (!xdr_int(xdrsp,&cid->distributor_id))
        return(0);

    if (!xdr_u_short(xdrsp,&cid->pr_port))
        return(0);

    if (!xdr_u_long(xdrsp,&cid->default_gc))
        return(0);

    if (!xdr_u_long(xdrsp,&cid->root))
        return(0);

    if (!xdr_u_long(xdrsp,&cid->xid))
        return(0);

    return(1);

} /* end xdr_ChanID */
/*************************************************************************/


/*************************************************************************/
/*
 * This routine translates a reception authorization structure.
 */
```

```
xdr_RecvAuth(xdrsp,ra)
    register XDR                *xdrsp;
    register struct RecvAuth    *ra;
{
/*
 * We need to just verify the various components of
 * the structure.
 */
    if (!xdr_int(xdrsp,&ra->authorization))
        return(0);

    if (!xdr_u_short(xdrsp,&ra->pm_port))
        return(0);

    if (!xdr_u_long(xdrsp,&ra->default_gc))
        return(0);

    if (!xdr_u_long(xdrsp,&ra->root))
        return(0);

    return(1);

} /* end xdr_RecvAuth */
/*********************************************************************/


/*********************************************************************/
/*
 * This routine translates a channel request structure.
 */
xdr_ChanReq(xdrsp,cr)
    register XDR                *xdrsp;
    register struct ChanReq *cr;
{
    register struct PortID  *pid;

/*
 * We need to just verify the various components of
 * the structure.
 */
    if (!xdr_int(xdrsp,&cr->channel))
        return(0);

    if (!xdr_int(xdrsp,&cr->distributor_id))
        return(0);

    pid     = &cr->PortID;
    if (!xdr_PortID(xdrsp,pid))
        return(0);

    return(1);
```

```
} /* end xdr_ChanReq */
/*********************************************************************/

/*********************************************************************/
/*
 * This routine translates a remove receiver request structure.
 */
xdr_RemvRecv(xdrsp,rr)
    register XDR            *xdrsp;
    register struct RemvRecv    *rr;
{

/*
 * We need to just verify the various components of
 * the structure.
 */
    if (!xdr_int(xdrsp,&rr->channel))
        return(0);

    if (!xdr_u_short(xdrsp,&rr->portnum))
        return(0);

    return(1);

} /* end RemvRecv */
/*********************************************************************/

/*********************************************************************/
/*
 * This routine handles xdr translations of a particular type of
 * data structure to return values associated with registering
 * a Distributor.
 */
xdr_RegDist(xdrsp, dr)
    register XDR                    *xdrsp;
    register struct DistRegister    *dr;
{
    char                            *ptr;

/*
 * We need to just verify the various components of
 * the structure.
 */
    ptr = dr->distname;
    if (!xdr_string(xdrsp,&ptr,PORTNAMLEN))
        return(0);

    if (!xdr_int(xdrsp,&dr->distributor_id))
        return(0);

    return(1);
```

```
} /* end xdr_RegDist */
/*****************************************************************/

/*****************************************************************/
/*
 * This routine handles xdr translations of a particular type of
 * data structure to return values associated with registering
 * a Receiver.
 */
xdr_RegRecv(xdrsp, rr)
    register XDR                    *xdrsp;
    register struct RecvRegister    *rr;
{
    char                            *ptr;

/*
 * We need to just verify the various components of
 * the structure.
 */
    ptr = rr->recvname;
    if (!xdr_string(xdrsp,&ptr,PORTNAMLEN))
        return(0);

    if (!xdr_int(xdrsp,&rr->distributor_id))
        return(0);

    if (!xdr_u_short(xdrsp,&rr->portnum))
        return(0);

    return(1);
} /* end xdr_RegRecv */
/*****************************************************************/
```

```
/*
 * File          : netwrite.c
 * Author        : P. Fitzgerald - SwRI
 * Date          : 11/30/89
 */
#include <stdio.h>
#include <sys/types.h>
#include <fcntl.h>
#include <signal.h>
#include <errno.h>

/* EXTERNAL ROUTINES */
extern      set_alarm();
extern      clear_alarm();

/*****************************************************************/
/*
 */
int
netwrite(fd,ptr,len)
    register int            fd;
    register unsigned char  *ptr;
    register int            len;
{
    register int            bytes_written;

#ifdef DATA
fprintf(stderr,">>> netwrite called fd:%d ptr:0x%x len:%d\n",
fd,ptr,len);
#endif

/*
 * Turn on an alarm in case we get stuck
 */
    set_alarm(500);

/*
 * Now enter a loop to read until all the bytes are read
 */
    errno = 0;
    bytes_written = write(fd,ptr,len);
    while (errno==EINTR || errno==EINVAL) {
        if (errno==EINTR)
            fprintf(stderr,"Network write interrupted by signal. Wrote %d
bytes. Re-issued.\n",
                bytes_written);
        else {
            fprintf(stderr,"Network write has invalid arguments. Wrote %d
bytes. Re-issued.\n",
                bytes_written);
            fprintf(stderr,"......fd:%d ptr:0x%x len:%d.\n",fd,ptr,len);
        }
```

```
        errno = 0;
        bytes_written = write(fd,ptr,len);
    }
    if (errno!=0) {
        fprintf(stderr,"Network    write    error>    fd:%d    ptr:0x%x
len:%d\n",fd,ptr,len);
        perror("Network write error>");
    }
    clear_alarm();
#ifdef DATA
fprintf(stderr,">>>>>>>> wrote:%d \n",bytes_written);
#endif
    return(bytes_written);

} /* end netwrite */
/*****************************************************************/
```

APPENDIX M
PROTOCOL RECEIVER LISTINGS

```
/*************************************************************************
The included program listings are prototypes, no warranty is expressed or
implied for their use in any other fashion.  They should not be considered
or used as production software.  The information in the listings is
supplied on an "as is" basis.  No responsibility is assumed for damages
resulting from the use of any information contained in the listings.

The software in these listings has been compiled on Masscomp 6350's and
6600's and on Sun 3's and 4's.  Modifications may be necessary for use on
other systems.
*************************************************************************/

/*
 * File         : pr.c
 * Author       : P. Fitzgerald - SwRI
 * Date         : 10/17/89
 * Description  : This file contains the code for the Protocol Receiver.
 */
#include <stdio.h>
#include <rpc/rpc.h>
#include <utmp.h>
#include <sys/types.h>
#include <rpcsvc/rusers.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <fcntl.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <sys/wait.h>
#include <signal.h>
#include <errno.h>
#include <X11/X.h>
#define NEED_REPLIES
#define NEED_EVENTS
#include <X11/Xproto.h>
#include <X11/Xlib.h>
#include "../includes/ds_manage.h"
#include "../includes/smtypes.h"
#include "../includes/smdef.h"
#include "../includes/dist.h"

/* LOCAL DEFINES */

/* EXTERNAL VARIABLES */

/* EXTERNAL ROUTINES */
extern          xdr_PortID();
```

```
extern          proproto();
extern          send_protocol();
extern          wait_on_protocol();
extern          check_for_new_client();

/* GLOBAL FUNCTIONS */
void            go_away();
void            acquire_port_number();
void            accept_connection();
int             callme();
int             timeout();
void            attach_shared_memory();

/* GLOBAL VARIABLES */
Display             *local_dpy;
Screen              *local_screens[MAX_SCREENS];
Display             *dest[MAX_CLIENTS];
XID                 source_default_gc[MAX_CLIENTS];
XID                 source_root[MAX_CLIENTS];
struct XIDmap       *idmap;
char                management_host[HOSTNAMLEN];
char                hostname[HOSTNAMLEN];
unsigned short      pm_port             = 0;
struct MC_SHMEMORY  *shmem;
int                 shmid;
int                 pm_fd               = -1;
static int          ignore              = TRUE;
int                 protofd;
unsigned short      my_port             = 0;
int                 number_of_clients   = 0;


/*************************************************************************/
/*
 * Main body
 */
main ()
{
    register struct MC_SHMEMORY *memptr;
    register struct Xpacket     *xp;

    struct Xpacket              Xpacket;


    sleep(5);
    fprintf(stderr,"SwRI Protocol Receiver starting...\n");

/*
 * Set up to catch timer-timeout signals
 */
    signal(SIGALRM, timeout);
/*
```

```
 * Set up to catch kill signals
 */
    signal(SIGQUIT, go_away);

/*
 * Call initialization routine
 */
    prinit();

/*
 * Now wait until the Distributor has received
 * an index from the Protocol Multiplexer. Then
 * we will register and connect to that index.
 */

#ifdef TRACE
fprintf(stderr,"PR:: waiting now on the distributor to say he is done\n");
#endif
    set_alarm(500);
    while (!shmem->pd_init) {
        if (shmem->sm_status==DIE)
            go_away();
        sleep(1);
    }
    clear_alarm();

#ifdef TRACE
fprintf(stderr,"PR:: pd init complete, registering self also index:%d\n",
shmem->distributor_id);
#endif
    register_self(shmem->distributor_id);
#ifdef TRACE
fprintf(stderr,"PR:: after register self....\n");
#endif

/*
 * Set register variables to enhance speed.
 */
    xp          = &Xpacket;
    memptr      = shmem;

/*
 * Look for input and wait_on_protocol protocol.
 */
    while (1) {

/*
 * Wait on X protocol to process
 */
        wait_on_protocol(xp);

/*
```

```
 * Now check to see if this is a brand new client which we must open up
 * a client connection for.
 */
        ignore  = check_for_new_client(xp->header.client);

/*
 * Now call the routine to handle the X protocol which was sent us.
 */
        if (!ignore)
            ignore      = prproto(xp);

/*
 * Now check, if not ignore this protocol, then send it on over to the
 * target display.
 */
        if (!ignore)
            send_protocol(xp);

/*
 * See if system is closing down.
 */
        if (memptr->sm_status==DIE) {
            fprintf(stderr,"PR::Requested to shut down.\n");
            go_away();
        }

    } /* end while */

} /* end main */
/*****************************************************************************/



/*****************************************************/
/*
 * This routine replaces a call to exit() to do cleanup.
 */
void
go_away()
{
/*
 * Close the backward display
 */
    XCloseDisplay(local_dpy);

/*
 * Tell someone we are going away.
 */
    fprintf(stderr,"PR:: EXITING.....\n");
    sleep(2);
    exit(0);
} /* end go_away */
```

```
/*****************************************************/

/*************************************************************************/
/*
 * This is a timeout routine to say that we got hung up waiting on
 * someting to happen.
 */
int
timeout()
{

#ifdef TRACE
    fprintf(stderr,"PR:: Timeout performing a read, wait, or write.\n");
#endif
    if (shmem->sm_status==DIE)
        go_away();
    set_alarm(30);

/*
 * Set up to catch timer-timeout signals
 */
    signal(SIGALRM, timeout);

#ifdef TRACE
fprintf(stderr,"PR:: Timeout Exiting\n");
#endif

} /* end timeout */
/*************************************************************************/
```

```
/*
 * File         : prinit.c
 * Author       : P. Fitzgerald - SwRI
 * Date         : 10/17/89
 * Description  : This file contains the code for the Protocol Receiver.
 * Initialization code and utilities.
 */
#include <stdio.h>
#include <rpc/rpc.h>
#include <sys/lock.h>
#include <utmp.h>
#include <sys/types.h>
#include <rpcsvc/rusers.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <fcntl.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <sys/wait.h>
#include <signal.h>
#include <errno.h>
#include <X11/X.h>
#define NEED_REPLIES
#define NEED_EVENTS
#include <X11/Xproto.h>
#include <X11/Xlib.h>
#include "../includes/ds_manage.h"
#include "../includes/smtypes.h"
#include "../includes/smdef.h"
#include "../includes/dist.h"

/* LOCAL DEFINES */

/* EXTERNAL VARIABLES */
extern char                 management_host[HOSTNAMLEN];
extern char                 hostname[HOSTNAMLEN];
extern struct MC_SHMEMORY   *shmem;
extern int                  pm_fd;
extern int                  protofd;
extern int                  shmid;
extern unsigned short       my_port;
extern Display              *local_dpy;
extern Screen               *local_screens[MAX_SCREENS];
extern struct XIDmap        *idmap;

/* EXTERNAL ROUTINES */
extern          xdr_PortID();
extern          go_away();

/* GLOBAL FUNCTIONS */
void            acquire_port_number();
```

```
void            accept_connection();
int             callme();
void            attach_shared_memory();
void            clear_alarm();
void            open_X_connections();
void            init_idmap();

/* GLOBAL VARIABLES */


/*************************************************************************/
/*
 * prinit
 */
void
prinit()
{
    register    int     i;

/*
 * Open a backward connection to the local X server(s) (up to 3)
 */
#ifdef TRACE
fprintf(stderr,"PR:: Open_X_connections called\n");
#endif
    open_X_connections();

/*
 * Attach to shared memory area
 */
    attach_shared_memory();

#ifdef TRACE
fprintf(stderr,"PR:: attached to shared memory\n");
#endif


/*
 * Acquire the host name where we reside.
 */
    if ( gethostname(hostname,sizeof(hostname)) <0) {
        perror("PR::gethostname:");
        go_away();
    }

#ifdef TRACE
fprintf(stderr,"PR:: get host name ok\n");
#endif

/*
 * Allocate initial memory for XID mapping.
 */
```

```
#ifdef TRACE
fprintf(stderr,"PR: init_idmap\n");
#endif
    init_idmap();

/*
 * Now try to contact a Central Distribution Manager, somewhere on
 * the network.
 */
#ifdef TRACE
fprintf(stderr,"PR:: contacting CDM management\n");
#endif
    if (!contact_cdm(management_host)) {
        fprintf(stderr,"PR::Unable  to  contact  Central  Distribution
Manager.\n");
        go_away();
    }
    sprintf(shmem->management_host,"%s",management_host);
    shmem->pr_init  = TRUE;

#ifdef TRACE
fprintf(stderr,"PR:: letting pd know we are running\n");
fprintf(stderr,"PR:: management host was:<%s>\n",management_host);
#endif

/*
 * Acquire a port number and place it in shared memory.
 */
    acquire_port_number();

#ifdef LOCKIT
/*
 * Finally of all, lock self into memory
 */
    if (plock((int)PROCLOCK)<0)
        perror("PR:: plock(PROCLOCK):");
#endif

#ifdef TRACE
fprintf(stderr,"PR:: enough of this init business!!!!\n");
#endif

} /* end prinit */
/*************************************************************************/


/*************************************************************************/
/*
 * This routine acquires a socket and port number.
 */
void
```

```
acquire_port_number()
{
    static struct sockaddr_in   sinhim = { AF_INET };
    static struct sockaddr_in   sinme  = { AF_INET };
    int                         sinlen;

    set_alarm(30);
/*
 * Create a socket (Internet-stream)
 */
    if ((protofd = socket(AF_INET, SOCK_STREAM,0))<0) {
        perror("PR::acquire_port_number (socket):");
        go_away();
    }

/*
 * Bind to my address.
 */
    if (bind(protofd,&sinme,sizeof(sinme))<0) {
        perror("PR::acquire_port_number (bind):");
        go_away();
    }
    sinlen  = sizeof(sinme);
    if (getsockname(protofd, &sinme, &sinlen)<0) {
        perror("PR::acquire_port_number (getsockname):");
        go_away();
    }
    my_port = ntohs(sinme.sin_port);

/*
 * Place port in shared memory.
 */
#ifdef TRACE
fprintf(stderr,"PR:: placing pr port number in shmem:%d\n",my_port);
#endif
    shmem->pr_port  = my_port;

/*
 * Listen and accept to acquire a file descriptor.
 */
    sinlen  = sizeof(sinhim);
    if (listen(protofd,0)<0) {
        perror("PR::acquire_port_number (listen):");
        go_away();
    }
    clear_alarm();

#ifdef TRACE
fprintf(stderr,"PR:: after listen and acquire of port number\n");
#endif

} /* end acquire_port_number */
```

```
/*************************************************************************/

/*************************************************************************/
/*
 * This routine attempts to read the /etc/host table and poll each
 * host until a CDM is found.
 */
#include <sys/socket.h>
#include <netdb.h>
int
contact_cdm(name)
    char        *name;
{
    struct PortID   PortID;
    int             retval;

    sprintf(name,"");
    retval = clnt_broadcast(CDM_PROG,CDM_VERS,CDM_PRESENT,xdr_void,0,
                    xdr_PortID,&PortID,callme);

/*
 * Retrieve some values from the RPC call
 */
    strncpy(name,PortID.hostname,PORTNAMLEN);
    shmem->pm_port  = PortID.portnum;

#ifdef TRACE
fprintf(stderr,"PR:: contact cdm returned hostname:<%s>\n",name);
fprintf(stderr,"PR::    ... and pm_port          :%d\n",PortID.portnum);
#endif

    if (retval==0)
        return(TRUE);
    else
        return(FALSE);

} /* end contact_cdm */
/*************************************************************************/

/*************************************************************************/
/*
 * This is a simple call-back routine from the client broadcast
 * search for a CDM host.
 */
int
callme(out,addr)
    char    *out;
    struct sockaddr_in  *addr;
{
    return(1);
} /* end call_me */
/*************************************************************************/
```

```
/*************************************************************************/
/*
 * This routine accepts a connection and returns with a file
 * descriptor to the socket.
 */
void
accept_connection()
{

    static struct sockaddr_in   sinhim = { AF_INET };
    int                         sinlen;

    pm_fd = accept(protofd, &sinhim, &sinlen);
    if (pm_fd<0) {
        perror("PR::acquire_port_number (accept):");
#ifdef GO_AWAY
        go_away();
#endif
    }

#ifdef TRACE
fprintf(stderr,"PR:: accept_connection complete pm_fd:%d\n",pm_fd);
#endif

} /* end accept_connection */
/*************************************************************************/

/*************************************************************************/
/*
 * This routine attaches to the shared memory area used between
 * the multicast (server mod) function, ldm, and protocol distributor
(me).
 * We will keep trying to attach for a long time before giving up.
 */
void
attach_shared_memory()
{
    shmid   = -1;
    set_alarm(500);

    while (shmid<0) {
        /* attach to shared memory */
        shmid   = shmget( (int)SM_KEY,
                sizeof(struct MC_SHMEMORY),0777);
        sleep(1);
    }
    shmem   = (struct MC_SHMEMORY *)shmat(shmid,0,0);
    if (shmem == (struct MC_SHMEMORY *)-1 ) {
        perror("PR::Unable to attach to Server shared memory.");
        go_away();
    }
```

```
    clear_alarm();
/*
 * Place some values in shared memory
 */
    shmem->default_gc       = local_screens[0]->default_gc->gid;
    shmem->root             = local_screens[0]->root;

#ifdef TRACE
fprintf(stderr,"PR:: default_gc:0x%x\n",shmem->default_gc);
fprintf(stderr,"PR:: root      :0x%x\n",shmem->root);
#endif


} /* end attach_shared_memory */
/*************************************************************************/


/*************************************************************************/
/*
 * This routine opens all the necessary backward connections to the
 * local X server for each display.
 */
void
open_X_connections()
{
    int             i;

#ifdef TRACE
fprintf(stderr,"PR:: looking to open server:<%s>\n",getenv("DISPLAY"));
#endif

/*
 * Open a connection to the source display and get all the screen
pointers.
 */
    if (!(local_dpy = XOpenDisplay(getenv("DISPLAY")))) {
        fprintf(stderr,"PR:: Cannot open backward connection to <%s>.\n",
                getenv("DISPLAY"));
        go_away();
    }
#ifdef TRACE
fprintf(stderr,"PR:: Connection opened to:<%s>.\n",getenv("DISPLAY"));
fprintf(stderr,"PR:: local_dpy:0x%x\n",local_dpy);
#endif
    for (i=0;i<MAX_SCREENS;i++)  {
        if (i<local_dpy->nscreens) {
#ifdef TRACE
fprintf(stderr,"PR:: target screen:%d = %d\n",
i,&local_dpy->screens[i]);
#endif
            local_screens[i]    = &local_dpy->screens[i];
        }
```

```
        else
            local_screens[i]    = NULL;
    } /* end for */

} /* end open_X_connections */
/*****************************************************************************/

/*****************************************************************************/
/*
 * This routine allocates storage for the XID mapping structure.
 */
void
init_idmap()
{

/* allocate first element and set to NULLs */
    idmap   = (struct XIDmap *)malloc(sizeof(struct XIDmap));
    if (idmap==NULL) {
        perror("PR:: malloc (XID map):");
        go_away();
    }
    idmap->source       = NULL;
    idmap->dest         = NULL;
    idmap->next         = NULL;

} /* end init_idmap */
/*****************************************************************************/
```

```
/*
 * File         : prio.c
 * Author       : P. Fitzgerald - SwRI
 * Date         : 10/17/89
 * Description  : This file contains the code for the Protocol Receiver.
 * I/O Routines
 */
#include <stdio.h>
#include <rpc/rpc.h>
#include <utmp.h>
#include <sys/types.h>
#include <rpcsvc/rusers.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <fcntl.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <sys/wait.h>
#include <signal.h>
#include <errno.h>
#include </home/overflow/X11/include/X.h>
#define NEED_REPLIES
#define NEED_EVENTS
#include </home/overflow/X11/include/Xproto.h>
#include </home/overflow/X11/include/Xlib.h>
#include </home/overflow/X11/include/Xlibint.h>
#include </home/overflow/X11/include/Xresource.h>
#include "../includes/ds_manage.h"
#include "../includes/smtypes.h"
#include "../includes/smdef.h"
#include "../includes/dist.h"

/* LOCAL DEFINES */

/* EXTERNAL VARIABLES */
extern Display              *dest[MAX_CLIENTS];
extern struct MC_SHMEMORY   *shmem;
extern int                  pm_fd;
extern char                 XFunctName[][80];

/* EXTERNAL ROUTINES */
extern              set_alarm();
extern              clear_alarm();
extern void         final_shutdown();

/* GLOBAL FUNCTIONS */
void                free_id();
void                wait_on_protocol();
void                send_protocol();
void                send_expose_event();
void                cause_expose_event();
```

```
void                get_window_state();
XID                 read_xid();
XID                 unsub_id();
void                waste_data();
void                wait_for_GCS();
void                wait_for_WATS();

/* EXTERNAL VARIABLES */
extern  int                 channel;

/*************************************************************************/
/*
 * This routine waits on X protocol to process.
 */
void
wait_on_protocol (Xptr)
    register    struct Xpacket  *Xptr;
{
    register    int             numread;
    register    short           *shortptr;
    register union COMPAK       *cp_reg;
    register unsigned char      *sb;

    unsigned char               signal_bytes[SIGLEN];
    union COMPAK                cp;

    cp_reg  = &cp;
    sb      = signal_bytes;

/*
 * Loop while we wait for requests to propagate expose events or
 * X protocol from the Multiplexer.
 */
    while (1) {

/*
 * Check for any expose type events on the local server
 */
        check_for_local_events();


#ifdef TRACE
fprintf(stderr,"\n\nPR:: Wait On Protocol\n");
#endif

/*
 * Read the signal byte that tells us what type of communication
 * that we are receiving.
 */
        set_alarm(1500);
        numread = netread(pm_fd,sb,SIGLEN);
        clear_alarm();
```

```
        if (numread!=SIGLEN) {
            perror("PR:: read (reading signal):");
#ifdef GO_AWAY
            go_away();
#endif
        }

#ifdef DATA
fprintf(stderr,"PR:: SIGNAL read 0x%x 0x%x 0x%x 0x%x\n",
signal_bytes[0],signal_bytes[1],signal_bytes[2],signal_bytes[3]);
#endif

/*
 * Now determine what type of message this is by looking at the signal
 * byte.
 */
            switch ((int)signal_bytes[0]) {

/*****************************************************************/
                /* No Operation requested */
                case    NOOP:
#ifdef TRACE
fprintf(stderr,"PR:: NOOP...received\n");
#endif
                    break;
/*****************************************************************/


/*****************************************************************/
                /* Shutdown of receiver channel complete */
                case SHUTCOMP:
#ifdef TRACE
fprintf(stderr,"PR:: SHUTCOMP complete received...\n");
#endif
                    shortptr = (short *)(sb+2);
                    final_shutdown( (int) *shortptr );
#ifdef TRACE
fprintf(stderr,"PR:: client shutdown complete for %d\n",*shortptr);
#endif
                    break;
/*****************************************************************/


/*****************************************************************/
                /* X PROTOCOL FOR TRANSLATION */
                case    X_DATA:
#ifdef TRACE
fprintf(stderr,"PR:: X_DATA....received\n");
#endif
                    /* Channel number for X_DATA only */
                    channel = (int)signal_bytes[1];
                    n      u      m      r      e      a      d    -
```

```
netread(pm_fd,&cp_reg->pdtopm.header,HDRLEN);
                    if (numread!=HDRLEN) {
                        perror("PR:: read (reading header):");
#ifdef GO_AWAY
                        go_away();
#endif
                    }
                    Xptr->header.client = cp_reg->pdtopm.header.client;
                    Xptr->header.len    = cp_reg->pdtopm.header.length;

#ifdef TRACE
fprintf(stderr,"PR:: length of this packet:%d\n",Xptr->header.len);
#endif
                    /* Now read the protocol into memory */
                    n      u      m      r      e      a      d    -
netread(pm_fd,Xptr->buffer,Xptr->header.len);
                    if (numread!=Xptr->header.len) {
                        perror("PR:: read (reading buffer):");
#ifdef GO_AWAY
                        go_away();
#endif
                    }

#ifdef DATA
fprintf( stderr, "PR:: in <%s>\n", XFunctName[ Xptr->buffer[0] ] );
fprintf(stderr,"PR:: just read %d bytes of protocol\n",numread);
fprintf(stderr,"PR:: 0...0x%x 0x%x 0x%x 0x%x \n",
Xptr->buffer[0],
Xptr->buffer[1],
Xptr->buffer[2],
Xptr->buffer[3]);
fprintf(stderr,"PR:: 1...0x%x 0x%x 0x%x 0x%x \n",
Xptr->buffer[4],
Xptr->buffer[5],
Xptr->buffer[6],
Xptr->buffer[7]);
fprintf(stderr,"PR:: 2...0x%x 0x%x 0x%x 0x%x \n",
Xptr->buffer[8],
Xptr->buffer[9],
Xptr->buffer[10],
Xptr->buffer[11]);
fprintf(stderr,"PR:: 3...0x%x 0x%x 0x%x 0x%x \n",
Xptr->buffer[12],
Xptr->buffer[13],
Xptr->buffer[14],
Xptr->buffer[15]);
#endif

                    clear_alarm();
                    return; /* return and handle protocol */
                /* END Handling X Protocol Read In */
/*****************************************************************/
```

```
/*****************************************************************/
                /* EXPOSE EVENT FOR PROPAGATION */
                case    EXPOSE:
#ifdef TRACE
fprintf(stderr,"PR:: EXPOSE....received\n");
#endif
                    cause_expose_event();
                    break;
/*****************************************************************/


/*****************************************************************/
                /* RETRIEVE GRAPHICS CONTEXT INFORMATION */
                case    GGCS:
#ifdef TRACE
fprintf(stderr,"PR:: GGCS....received\n");
#endif
#ifdef DATA
fprintf(stderr,"PR:: 0x%x 0x%x\n",
signal_bytes[2],signal_bytes[3]);
#endif
                    get_gc_state((short *)(sb+2));
                    break;
/*****************************************************************/


/*****************************************************************/
                /* RETRIEVE WINDOW ATTRIBUTES */
                case    GWATS:
#ifdef TRACE
fprintf(stderr,"PR:: GWATS....received\n");
#endif
                    get_window_state((short *)(sb+2));
                    break;
/*****************************************************************/

/*****************************************************************/
                default:
                    fprintf(stderr,"PR:: Unknown signal byte:0x%x\n",
                            (int)signal_bytes[0]);
#ifdef GO_AWAY
                    go_away();
#endif
                    break;
/*****************************************************************/
            } /* end switch */

        } /* end while */

} /* end wait_on_protocol */
/*************************************************/
```

```
/*****************************************************/
/*
 * This routine is responsible for sending the translated
 * protocol to the local x server.
 */
void
send_protocol(Xptr)
    register struct Xpacket *Xptr;
{
    register int            clnt;
    register int            leftovers;

    xGetInputFocusReply rep;
    register xReq *req;
    Display *dpy;
int i;
#ifdef DATA
fprintf(stderr,"PR:: SEND_PROTOCOL to server\n");
#endif

    clnt    = Xptr->header.client;

#ifdef MAXcheck
fprintf(stderr,"PR::    send_protocol>    bufmax:0x%x,
bufptr:0x%x\n",dest[clnt]->bufmax,
dest[clnt]->bufptr);
fprintf(stderr,"PR:: QLength(dpy):0x%x\n",QLength(dest[clnt]));
if (clnt!=6) {
fprintf(stderr,">>>>>> WARNING! clnt!=6, it is:0x%x\n",clnt);
}
#endif

/*
 * First, check to see if there are any events to be handled at the
 * Server level before we go working on the buffers.
 */
    while ( QLength(dest[clnt]) )
        check_for_local_events();

    LockDisplay(dest[clnt]);

#ifdef DATA
fprintf(stderr,"PR:: lock server\n");
fprintf(stderr,"PR::   going   to   write   to
dest[Xptr->header.client]->fd:%d\n",
dest[Xptr->header.client]->fd);
fprintf(stderr,"PR:: clnt:%d\n",clnt);
fprintf(stderr,"PR:: Xptr->header.len   :%d\n",Xptr->header.len);
fprintf(stderr,"\nPR:: DATA WRITTEN TO SERVER FD BELOW:\n");
for (i=0;i<Xptr->header.len;i++) {
```

```
fprintf(stderr,"0x%x ",Xptr->buffer[i]);
}
fprintf(stderr,"\nDATA WRITTEN TO SERVER FD ABOVE\n");
#endif

#ifdef BUFKLUDGE
if (dest[clnt]->bufptr==0) {
fprintf(stderr,"PR:: BUFFER KLUDGE0 INVOKED\n");
fprintf(stderr,".....bufmax:0x%x buffer:0x%x\n",
dest[clnt]->bufmax,dest[clnt]->buffer);
dest[clnt]->bufptr = dest[clnt]->buffer;
dest[clnt]->bufmax = dest[clnt]->buffer + 2048;
dest[clnt]->request = 0;
}
#endif
/*
 * Allocate buffer space in the server if there
 * is enough room, if not, flush the buffer first.
 */
    if ((dest[clnt]->bufptr+Xptr->header.len) > dest[clnt]->bufmax)
{
fprintf(stderr,"PR:: flushing buffer BEFORE placing protocol into it\n");
        _XFlush(dest[clnt]);
}
    dest[clnt]->last_req    = dest[clnt]->bufptr;


#ifdef MAXcheck
fprintf(stderr,".... send_protocol>  bufmax:0x%x,
bufptr:0x%x\n",dest[clnt]->bufmax,
dest[clnt]->bufptr);
#endif

#ifdef DATA
fprintf(stderr,"PR:: placed into buffer %d bytes\n",Xptr->header.len);
#endif

/*
 * Copy the protocol into the buffer
 */

#ifdef MAXcheck
if ( (dest[clnt]->bufptr+Xptr->header.len) >= dest[clnt]->bufmax ) {
fprintf(stderr,"PR:: WHOAH! we are overwriting end of buffer!!!\n");
sleep(2);
}
if (clnt!=6) {
fprintf(stderr,"JUST BEFORE BCOPY, CLNT IS:0x%x\n",clnt);
}
if (Xptr->header.len>400) {
fprintf(stderr,"LEN IS ASTRONOMICAL!!!0x%x\n",Xptr->header.len);
}
```

```
if (Xptr->header.client!=6) {
fprintf(stderr,"CLIENT IS WRONG!!!0x%x\n",Xptr->header.client);
}
#endif

#ifdef BUFKLUDGE
if (dest[clnt]->bufptr==0) {
fprintf(stderr,"PR:: BUFFER KLUDGE1 INVOKED\n");
fprintf(stderr,".....bufmax:0x%x buffer:0x%x\n",
dest[clnt]->bufmax,dest[clnt]->buffer);
dest[clnt]->bufptr = dest[clnt]->buffer;
dest[clnt]->bufmax = dest[clnt]->buffer + 2048;
dest[clnt]->request = 0;
}
#endif

    bcopy((char *)Xptr->buffer,(char
*)dest[clnt]->bufptr,(int)Xptr->header.len);
    dest[clnt]->request++;
    dest[clnt]->bufptr += Xptr->header.len;

    if ( (dest[clnt]->bufptr - dest[clnt]->buffer) & 3) {
fprintf(stderr,"PR:: flushing buffer because address is not on 4 byte
boundary.\n");
        _XFlush(dest[clnt]);
    }

#ifdef DATA
fprintf( stderr, "PR:: protocol <%s> copied into server buffer.\n",
XFunctName[*(Xptr->buffer)] );
#endif

#ifdef MAXcheck
fprintf(stderr,"                    send_protocol>   bufmax:0x%x,
bufptr:0x%x\n",dest[clnt]->bufmax,
dest[clnt]->bufptr);
#endif

#ifdef XFLUSHit
#ifdef DATA
fprintf(stderr,"PR:: flushing protocol to server.\n");
#endif
        _XFlush(dest[clnt]);
#ifdef DATA
fprintf(stderr,"PR:: protocol flushed.\n");
#endif
#endif

#ifdef XSYNC
    dpy = dest[clnt];
    GetEmptyReq(GetInputFocus, req);
    (void) _XReply (dest[clnt], (xReply *)&rep, 0, xTrue);
```

```
#endif
    UnlockDisplay(dest[clnt]);

#ifdef DATA
fprintf(stderr,"PR:: after unlocking display\n");
#endif

/*
 * Call the synchandler to possibly send protocol
 * to server.
 */
#ifdef SYNCHANDLER
    if (dest[clnt]->synchandler)
        (*dest[clnt]->synchandler)(dest[clnt]);
#endif

#ifdef DATA
fprintf(stderr,"PR:: after calling synchandler\n");
#endif

} /* end send_protocol */
/*****************************************************/

/*****************************************************/
/*
 * This routine sends an expose event back to the X server
 * for a given client. Note that the X server is NOT the
 * local one, but actually the source server for a given
 * window. It does this by setting flags in shared memory.
 */
void
send_expose_event(window)
    register    XID window;
{

#ifdef TRACE
fprintf(stderr,"PR:: send_expose_event for window:0x%x\n",window);
fprintf(stderr,"PR:: via signal to PD....\n");
#endif
/*
 * First set the window in shared memory.
 */
    shmem->expose_window        = window;

/*
 * Now set the flag
 */
    shmem->pd_propagate_expose  = TRUE;

/*
 * Now send a signal to the Distributor
 */
```

```
    kill(shmem->pd_pid,SIGUSR1);

/*
 * Now set an alarm and hang around until we
 * see that the PD has taken care of business.
 */
#ifdef TRACE
fprintf(stderr,"PR:: waiting on the pd to complete expose handling.\n");
#endif
    set_alarm(30);
    while (shmem->pd_propagate_expose)
        sleep(1);
    clear_alarm();

#ifdef TRACE
fprintf(stderr,"PR:: pd took care of the expose event\n");
#endif


} /* end send_expose_event */
/*****************************************************/


/*****************************************************************/
/*
 * This routine reads from the pm file descriptor to obtain the
 * XID information passed just after a request signal byte is sent.
 * Note that the first byte read is padding.
 */
XID
read_xid()
{
    XID             xid;
    register int    bytes_read;

#ifdef TRACE
fprintf(stderr,"PR:: read_xid called\n");
#endif

/*
 * Now read the xid for the given request.
 */
    bytes_read  = netread(pm_fd,&xid,sizeof(xid));
    if (bytes_read != sizeof(xid) ) {
        perror("PR:: read (read_xid/xid):");
#ifdef GO_AWAY
        go_away();
#endif
    }

#ifdef TRACE
fprintf(stderr,"PR:: read the xid:0x%x\n",xid);
```

```
#endif
    return(xid);

} /* end read_xid */
/*****************************************************************/


/*****************************************************************/
/*
 * This routine is called after a request for graphics context
 * state information was given to the Protocol Distributor. The
 * Distributor will then pass that request on to the Protocol
 * Multiplexer, who will, in turn, query the appropriate source
 * Protocol Receiver. The Receiver then retrieves the graphics
 * context information, passes it to its own Protocol Distributor,
 * who then passes it back to the Protocol Multiplexer. The PM
 * finally writes it back to us here.
 * Current implementation means that all transmissions to the Receiver
 * during this time, except GCS will be ignored.
 */
void
wait_for_GCS(ptr,window)
    register    XGCValues   *ptr;
    register    XID         *window;
{
    register int            bytes;
    unsigned char           signal_bytes[SIGLEN];

#ifdef TRACE
fprintf(stderr,"PR:: wait_for_GCS called.\n");
#endif

/*
 * Continuous loop until we get what we want
 */
    while (1) {

/*
 * While we are looping, check to see if we need
 * to go away
 */
        if (shmem->sm_status==DIE)
            go_away();

/*
 * Read the signal byte that tells us what type of communication
 * that we are receiving.
 */
        bytes = netread(pm_fd,signal_bytes,SIGLEN);
        if (bytes!=SIGLEN) {
            perror("PR:: read (reading signal/wait_for_GCS):");
#ifdef GO_AWAY
```

```
            go_away();
#endif
        }

/*
 * Now determine what type of signal is read and what to
 * do with it.
 */
        switch (signal_bytes[0]) {

/*****************************************************************/
/*
 * Handle the case where we get basic graphics state information
 */
            case    GCS:
#ifdef TRACE
fprintf(stderr,"PR:: GCS state information is coming my way\n");
#endif
                bytes   = netread(pm_fd,(unsigned char *)ptr,
                            sizeof(XGCValues) );
                if (bytes != sizeof(XGCValues) ) {
                    perror("PR:: read (wait_for_GCS/XGCValues):");
#ifdef GO_AWAY
                    go_away();
#endif
                }
#ifdef TRACE
fprintf(stderr,"PR:: Just read %d bytes of GC state info.\n",bytes);
#endif
                bytes   = netread(pm_fd,window,sizeof(*window));
                if (bytes != sizeof(*window) ) {
                    perror("PR:: read (wait_for_GCS/window):");
#ifdef GO_AWAY
                    go_away();
#endif
                }

#ifdef TRACE
fprintf(stderr,"PR:: Just read %d bytes of associated window:0x%x\n",
bytes,*window);
#endif

                return;
/*****************************************************************/


/*****************************************************************/
            case X_DATA:
#ifdef TRACE
fprintf(stderr,"PR:: wait_for_GCS IGNORING AN X_DATA request\n");
#endif
                waste_data();
```

```
                break;
/*****************************************************************/

/*****************************************************************/
            case EXPOSE:
fprintf(stderr,"PR:: wait_for_GCS IGNORING AN EXPOSE request\n");
                break;
/*****************************************************************/


/*****************************************************************/
            case GGCS:
fprintf(stderr,"PR:: wait_for_GCS handling A  GGCS request\n");
                get_gc_state((short *)(&signal_bytes[2]));
                break;
/*****************************************************************/

/*****************************************************************/
            case GWATS:
fprintf(stderr,"PR:: wait_for_GCS STORING A  GWATS request\n");
                break;
/*****************************************************************/


/*****************************************************************/
            case WATS:
fprintf(stderr,"PR:: wait_for_GCS STORING A  WATS request\n");
                break;
/*****************************************************************/

            default:
                fprintf(stderr,"PR::    Unknown    signal    sent    in
wait_for_GCS:0x%x\n",
                    signal_bytes[0]);
                break;
        } /* end switch */

    } /* end while */

} /* end wait_for_GCS */
/*****************************************************************/


/*****************************************************************/
/*
 * This routine simply reads an X_DATA request from the Protocol
 * Multiplexer file descriptor and stores it in a buffer.
 */
void
waste_data()
{
```

```
    register    int     bytes;
    register    int     wasted;
    register    int     *intptr;
    unsigned    char    length[LENLEN];
    unsigned    char    buff[XBUFFERSIZE+PAKLEN];

/*
 * Read the length of the package
 */
    bytes   = netread(pm_fd,length,LENLEN);
    if (bytes != LENLEN) {
        perror("PR:: read (waste_data/length):");
#ifdef GO_AWAY
        go_away();
#endif
    }
    intptr  = (int *)length;
    wasted  = *intptr + CLIENTLEN;

#ifdef TRACE
fprintf(stderr,"PR:: attempting to store %d bytes of data\n",wasted);
fprintf(stderr,"PR::.....length 0x%x 0x%x 0x%x 0x%x\n",
length[0],length[1],length[2],length[3]);
#endif

if (wasted > XBUFFERSIZE+PAKLEN) {
fprintf(stderr,"PR:: trying to waste more bytes than have space...\n");
go_away();
}
/*
 * Now attempt to read and throw away that many bytes
 */
    bytes   = netread(pm_fd,buff,wasted);
    if (bytes != wasted) {
        perror("PR:: read (waste_data/data):");
#ifdef GO_AWAY
        go_away();
#endif
    }

#ifdef TRACE
fprintf(stderr,"PR:: waste_data called, wasted %d bytes\n",wasted);
fprintf(stderr,"PR::.....data0 0x%x 0x%x 0x%x 0x%x\n",
buff[0],buff[1],buff[2],buff[3]);
fprintf(stderr,"PR::.....data1 0x%x 0x%x 0x%x 0x%x\n",
buff[4],buff[5],buff[6],buff[7]);
fprintf(stderr,"PR::.....data2 0x%x 0x%x 0x%x 0x%x\n",
buff[8],buff[9],buff[10],buff[11]);
#endif

} /* end waste_data */
/*****************************************************************/
```

```
/*****************************************************************/
/*
 * This routine is called after a request for window attributes
 * state information was given to the Protocol Distributor. The
 * Distributor will then pass that request on to the Protocol
 * Multiplexer, who will, in turn, query the appropriate source
 * Protocol Receiver. The Receiver then retrieves the window
 * attributes information, passes it to its own Protocol Distributor,
 * who then passes it back to the Protocol Multiplexer. The PM
 * finally writes it back to us here.
 */
void
wait_for_WATS(ptr,background,parent)
    register    XWindowAttributes   *ptr;
    register    unsigned long       *background;
    register    XID                 *parent;
{
    register int                    bytes;
    unsigned char                   signal_bytes[SIGLEN];

#ifdef TRACE
fprintf(stderr,"PR:: wait_for_WATS called.\n");
#endif

/*
 * Continuous loop until we get what we want
 */
    while (1) {

/*
 * While we are looping, check to see if we need
 * to go away
 */
        if (shmem->sm_status==DIE)
            go_away();

/*
 * Read the signal byte that tells us what type of communication
 * that we are receiving.
 */
        bytes = netread(pm_fd,signal_bytes,SIGLEN);
        if (bytes!=SIGLEN) {
            perror("PR:: read (reading signal/wait_for_WATS):");
#ifdef GO_AWAY
            go_away();
#endif
        }

/*
 * Now determine what type of signal is read and what to
 * do with it.
```

```
*/
        switch (signal_bytes[0]) {

/*****************************************************************/
/*
 * Handle the case where we get basic window attributes information
 */
            case    WATS:
#ifdef TRACE
fprintf(stderr,"PR:: WATS state information is coming my way\n");
#endif
                bytes   = netread(pm_fd,background,sizeof(*background));
                if (bytes != sizeof(*background) ) {
                    perror("PR:: read (wait_for_WATS/background");
#ifdef GO_AWAY
                    go_away();
#endif
                }

#ifdef TRACE
fprintf(stderr,"PR:: WATS background:0x%x\n",*background);
#endif

                bytes   = netread(pm_fd,parent,sizeof(*parent));
                if (bytes != sizeof(*parent) ) {
                    perror("PR:: read (wait_for_WATS/parent");
#ifdef GO_AWAY
                    go_away();
#endif
                }

#ifdef TRACE
fprintf(stderr,"PR:: WATS parent:0x%x\n",*parent);
#endif

                bytes   = netread(pm_fd,(unsigned char *)ptr,
                            sizeof(XWindowAttributes) );
                if (bytes != sizeof(XWindowAttributes) ) {
                    perror("PR:: read (wait_for_WATS):");
#ifdef GO_AWAY
                    go_away();
#endif
                }
#ifdef TRACE
fprintf(stderr,"PR:: Just read %d bytes of WATS state info.\n",bytes);
#endif
                return;
/*****************************************************************/


/*****************************************************************/
            case X_DATA:
```

```
#ifdef TRACE
fprintf(stderr,"PR:: wait_for_WATS IGNORING AN X_DATA request\n");
#endif
                waste_data();
                break;
/*****************************************************************/


/*****************************************************************/
            case EXPOSE:
fprintf(stderr,"PR:: wait_for_WATS IGNORING AN EXPOSE request\n");
                break;
/*****************************************************************/


/*****************************************************************/
            case GGCS:
fprintf(stderr,"PR:: wait_for_WATS handling A  GGCS request\n");
                break;
/*****************************************************************/

/*****************************************************************/
            case GWATS:
#ifdef TRACE
fprintf(stderr,"PR:: wait_forWATS STORING A  GWATS request\n");
#endif
                get_window_state((short *)(&signal_bytes[2]));
                break;
/*****************************************************************/


/*****************************************************************/
            case GCS:
fprintf(stderr,"PR:: wait_for_WATS STORING A  GC request\n");
                break;
/*****************************************************************/

            default:
                fprintf(stderr,"PR::    Unknown    signal    sent    in
wait_for_WATS:0x%x\n",
                    signal_bytes[0]);
                break;
        } /* end switch */

    } /* end while */

} /* end wait_for_WATS */
/*****************************************************************/

/*****************************************************************/
/*
 * This routine cycles through the current XID list and checks
```

```
 * each MAPPED window for incoming events. These could be either
 * EXPOSE or X_ERROR events.
 */
check_for_local_events()
{
    register    Window      window;
    XID                     client;
    XEvent                  event;
    XEvent                  report;
    XAnyEvent               *any;

/*
 * Go through the list of clients and windows
 * to see which connections are valid and which
 * ones to look for events on.
 */
    window  = next_window(&client);

#ifdef PRINT_EVENT
if (window==NULL)
fprintf(stderr,"NULL window found, not checking for events...\n");
#endif

    while (window!=NULL) {

#ifdef PRINT_EVENT
fprintf(stderr,"PR:: found a window to look for events on:0x%x\n",window);
f  p  r  i  n  t  f  (  s  t  d  e  r  r  ,  "  P  R  :  :
....QLength(dest[client]):0x%x\n",QLength(dest[client]));
#endif

#ifdef NEXTevent
/*
 * Now handle all the events on this display connection
 */
        while ( XEventsQueued(dest[client],QueuedAfterFlush) ) {
#endif

#ifdef PRINT_EVENT
fprintf(stderr,"PR:: just before XNextEvent bufptr:0x%x bufmax:0x%x\n",
dest[client]->bufptr,dest[client]->bufmax);
#endif

#ifdef WINDOWevent
        if ( (XCheckWindowEvent(dest[client],window,
                ButtonPressMask         |
                ButtonReleaseMask       |
                EnterWindowMask         |
                LeaveWindowMask         |
                PointerMotionMask       |
                PointerMotionHintMask   |
                Button1MotionMask       |
```

```
                Button2MotionMask       |
                Button3MotionMask       |
                Button4MotionMask       |
                Button5MotionMask       |
                ButtonMotionMask        |
                KeymapStateMask         |
                NoEventMask             |
                KeyPressMask            |
                KeyReleaseMask          |
                ExposureMask            |
                VisibilityChangeMask    |
                StructureNotifyMask     |
                ResizeRedirectMask      |
                SubstructureNotifyMask  |
                FocusChangeMask         |
                PropertyChangeMask      |
                ColormapChangeMask      |
                OwnerGrabButtonMask     , &event)) ) {
#endif
#ifdef MASKevent
        if ( (XCheckMaskEvent(dest[client],
                ButtonPressMask         |
                ButtonReleaseMask       |
                EnterWindowMask         |
                LeaveWindowMask         |
                PointerMotionMask       |
                PointerMotionHintMask   |
                Button1MotionMask       |
                Button2MotionMask       |
                Button3MotionMask       |
                Button4MotionMask       |
                Button5MotionMask       |
                ButtonMotionMask        |
                KeymapStateMask         |
                NoEventMask             |
                KeyPressMask            |
                KeyReleaseMask          |
                ExposureMask            |
                VisibilityChangeMask    |
                StructureNotifyMask     |
                ResizeRedirectMask      |
                SubstructureNotifyMask  |
                FocusChangeMask         |
                PropertyChangeMask      |
                ColormapChangeMask      |
                OwnerGrabButtonMask     , &event)) ) {
#endif
#ifdef NEXTevent

                XNextEvent(dest[client],&event);
#endif
                any = (XAnyEvent *)&event;
```

```
#ifdef PRINT_EVENT
fprintf(stderr,"PR:: just after XNextEvent, bufmax:0x%x bufptr:0x%x\n",
dest[client]->bufmax,dest[client]->bufptr);
#endif

            switch (event.type) {
                case Expose:
#ifdef PRINT_EVENT
fprintf(stderr,"PR::  XCheckMaskEvent  found  an  EXPOSE  event  on
client:0x%x\n",client);
#endif

                    while(XCheckTypedEvent(dest[client],Expose,&report))
{
#ifdef TRACE
fprintf(stderr,"local expose ignored win: 0x%x\n",window);
#endif
                    }
                    send_expose_event(unsub_id(window));
#ifdef TRACE
fprintf(stderr,"local expose sent win:
(0x%x/0x%x)\n",unsub_id(window),window);
#endif
                    break;
                case ButtonPress:
#ifdef PRINT_EVENT
fprintf(stderr,"PR:: XCheckMaskEvent: received a ButtonPress\n");
#endif
                    break;
                case ButtonRelease:
#ifdef PRINT_EVENT
fprintf(stderr,"PR:: XCheckMaskEvent: received a ButtonRelease\n");
#endif
                    break;
                case CirculateNotify:
#ifdef PRINT_EVENT
fprintf(stderr,"PR:: XCheckMaskEvent: received a CirculateNotify\n");
#endif
                    break;
                case ClientMessage:
#ifdef PRINT_EVENT
fprintf(stderr,"PR:: XCheckMaskEvent: received a ClientMessage\n");
#endif
                    break;
                case ColormapNotify:
fprintf(stderr,"PR:: Got ColormapNotify\n");
#ifdef PRINT_EVENT
fprintf(stderr,"PR:: XCheckMaskEvent: received a ColormapNotify\n");
#endif
                    break;
                case ConfigureNotify:
```

```
#ifdef PRINT_EVENT
fprintf(stderr,"PR:: XCheckMaskEvent: received a ConfigureNotify\n");
#endif
                break;
            case ConfigureRequest:
#ifdef PRINT_EVENT
fprintf(stderr,"PR:: XCheckMaskEvent: received a ConfigureRequest\n");
#endif
                break;
            case CreateNotify:
#ifdef PRINT_EVENT
fprintf(stderr,"PR:: XCheckMaskEvent: received a CreateNotify\n");
#endif
                break;
            case DestroyNotify:
#ifdef PRINT_EVENT
fprintf(stderr,"PR:: XCheckMaskEvent: received a DestroyNotify\n");
#endif
                break;
            case EnterNotify:
#ifdef PRINT_EVENT
fprintf(stderr,"PR:: XCheckMaskEvent: received a EnterNotify\n");
#endif
                break;
            case LeaveNotify:
#ifdef PRINT_EVENT
fprintf(stderr,"PR:: XCheckMaskEvent: received a LeaveNotify\n");
#endif
                break;
            case FocusIn:
#ifdef PRINT_EVENT
fprintf(stderr,"PR:: XCheckMaskEvent: received a FocusIn\n");
#endif
                break;
            case FocusOut:
#ifdef PRINT_EVENT
fprintf(stderr,"PR:: XCheckMaskEvent: received a FocusOut\n");
#endif
                break;
            case GraphicsExpose:
#ifdef PRINT_EVENT
fprintf(stderr,"PR:: XCheckMaskEvent: received a GraphicsExpose\n");
#endif
                break;
            case NoExpose:
#ifdef PRINT_EVENT
fprintf(stderr,"PR:: XCheckMaskEvent: received a NoExpose\n");
#endif
                break;
            case GravityNotify:
#ifdef PRINT_EVENT
fprintf(stderr,"PR:: XCheckMaskEvent: received a GravityNotify\n");
```

```
#endif
                    break;
                case KeymapNotify:
#ifdef PRINT_EVENT
fprintf(stderr,"PR:: XCheckMaskEvent: received a KeymapNotify\n");
#endif
                    break;
                case KeyPress:
#ifdef PRINT_EVENT
fprintf(stderr,"PR:: XCheckMaskEvent: received a KeyPress\n");
#endif
                    break;
                case KeyRelease:
#ifdef PRINT_EVENT
fprintf(stderr,"PR:: XCheckMaskEvent: received a KeyRelease\n");
#endif
                    break;
                case MapNotify:
#ifdef PRINT_EVENT
fprintf(stderr,"PR:: XCheckMaskEvent: received a MapNotify\n");
#endif
                    break;
                case UnmapNotify:
#ifdef PRINT_EVENT
fprintf(stderr,"PR:: XCheckMaskEvent: received a UnmapNotify\n");
#endif
                    break;
                case MappingNotify:
#ifdef PRINT_EVENT
fprintf(stderr,"PR:: XCheckMaskEvent: received a MappingNotify\n");
#endif
                    break;
                case MapRequest:
#ifdef PRINT_EVENT
fprintf(stderr,"PR:: XCheckMaskEvent: received a MapRequest\n");
#endif
                    break;
                case MotionNotify:
#ifdef PRINT_EVENT
fprintf(stderr,"PR:: XCheckMaskEvent: received a MotionNotify\n");
#endif
                    break;
                case PropertyNotify:
#ifdef PRINT_EVENT
fprintf(stderr,"PR:: XCheckMaskEvent: received a PropertyNotify\n");
#endif
                    break;
                case ReparentNotify:
#ifdef PRINT_EVENT
fprintf(stderr,"PR:: XCheckMaskEvent: received a ReparentNotify\n");
#endif
                    break;
```

```
                case ResizeRequest:
#ifdef PRINT_EVENT
fprintf(stderr,"PR:: XCheckMaskEvent: received a ResizeRequest\n");
#endif
                    break;
                case SelectionClear:
#ifdef PRINT_EVENT
fprintf(stderr,"PR:: XCheckMaskEvent: received a SelectionClear\n");
#endif
                    break;
                case SelectionRequest:
#ifdef PRINT_EVENT
fprintf(stderr,"PR:: XCheckMaskEvent: received a SelectionRequest\n");
#endif
                    break;
                case VisibilityNotify:
#ifdef PRINT_EVENT
fprintf(stderr,"PR:: XCheckMaskEvent: received a VisibilityNotify\n");
#endif
                    break;
                default:
#ifdef PRINT_EVENT
fprintf(stderr,"PR::  XCheckMaskEvent:  Unable  to  determine  event
type:0x%x\n",event.type);
#endif
                    break;
            } /* end switch */

#ifdef PRINT_EVENT
fprintf(stderr,"PR:: just after XNextEvent, bufmax:0x%x bufptr:0x%x\n",
dest[client]->bufmax,dest[client]->bufptr);
if (any->send_event)
fprintf(stderr,".....serial:0x%x        send_event:TRUE         dpy:0x%x
window:0x%x\n",
any->serial,any->display,any->window);
else
fprintf(stderr,".....serial:0x%x        send_event:FALSE    dpy:0x%x
window:0x%x\n",
any->serial,any->display,any->window);
#endif

        } /* end while */

/*
 * Get next window and client
 */
        window  = next_window(&client);
    } /* end while */

} /* end check_for_local_events */
/*****************************************************************/
```

```
/*
 * File          : prproto.c
 * Author        : P. Fitzgerald - SwRI
 * Date          : 10/17/89
 * Description   : This file contains the code for the Protocol Receiver.
 * Protocol Handling routines
 */
#include <stdio.h>
#include <rpc/rpc.h>
#include <utmp.h>
#include <sys/types.h>
#include <rpcsvc/rusers.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <fcntl.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <sys/wait.h>
#include <signal.h>
#include <errno.h>
#include <X11/X.h>
#define NEED_REPLIES
#define NEED_EVENTS
#include <X11/Xproto.h>
#include <X11/Xlib.h>
#include </home/overflow/X11/include/Xlibint.h>
#include "../includes/ds_manage.h"
#include "../includes/smtypes.h"
#include "../includes/smdef.h"
#include "../includes/dist.h"
#include "../includes/xdefs.h"

/* LOCAL DEFINES */

/* EXTERNAL VARIABLES */
extern Display              *dest[MAX_CLIENTS];

/* EXTERNAL ROUTINES */
extern  XID                 add_id();
extern  XID                 sub_id();
extern  GC                  get_gc();
extern  void                update_gc();

/* GLOBAL FUNCTIONS */

/* GLOBAL VARIABLES */

/*****************************************************************************/
/*
 * This routine handles all the X protocol.
 */
```

```
prproto (Xptr)
    register struct Xpacket *Xptr;
{
    register    void            *Vptr;
    register    int             request_type;
    register    int             client;
    register    XID             old_id;
    register    GC              gc;


/*
 * Determine what type of X request this is.
 */
    request_type    = (int)Xptr->buffer[0];
    client          = (int)Xptr->header.client;

#ifdef TRACE
f p r i n t f ( s t d e r r , " \ n P R : :    X    R E Q U E S T
for:<%s>,client:%d\n",XFunctName[request_type],client);
#endif

/*********************************************************************/
/*
 * Coerce a void  pointer on top of buffer. This will allow us to use
 * only one pointer ( a register resident one) to access all the various
 * structure types. We will be performing either an add_id() - creating
 *  a new id mapping from source to destination, or a sub_id() -
substituting
 * a local id for a remote one. Then the protocol is sent on to the local
 * X server.
 */
    Vptr    = (void *)(Xptr->buffer);
    switch (request_type) {

/*********************************************************************/
/*
 * X Create Window
 */
        case X_CreateWindow:
            ((xCreateWindowReq *)(Vptr))->parent    =
                    sub_id( ((xCreateWindowReq *)(Vptr))->parent,
                            client,
                            WINDOW_TYPE );
            ((xCreateWindowReq *)(Vptr))->class     = CopyFromParent;
            ((xCreateWindowReq *)(Vptr))->visual    = CopyFromParent;
            ((xCreateWindowReq *)(Vptr))->wid       =
                    add_id( dest[client],
                            ((xCreateWindowReq *)(Vptr))->wid,
                            WINDOW_TYPE,
                            CREATE_ID,
                            client);
/*
```

```
 * WARNING! There could be data that follows this protocol. That means
 * that at some point, code needs to be added to check what type of data
 *  and if there are any resource IDs which need to be created or
translated.
 */
            break;
/*********************************************************************/


/*********************************************************************/
/*
 * The following routines use a standard ResourceReq structure, which
 * contains a generic XID value: id.
 *
 * X Close Font
 * X Destroy Window
 * X Destroy Subwindows
 * X Free Colormap
 * X Free GC
 * X Free Pixmap
 * X Kill Client
 * X Unmap Subwindows
 * X Unmap Window
 */
        case X_CloseFont:
        case X_DestroyWindow:
        case X_DestroySubwindows:
        case X_FreeColormap:
        case X_FreeGC:
        case X_FreePixmap:
        case X_KillClient:
        case X_UnmapSubwindows:
        case X_UnmapWindow:
            old_id                              =    ((xResourceReq
*)(Vptr))->id;
            ((xResourceReq *)(Vptr))->id    =
                sub_id( ((xResourceReq *)(Vptr))->id,
                        client,
                        WINDOW_TYPE);
            free_id(old_id,client);
            break;
/*********************************************************************/

/*********************************************************************/
/*
 * X Change Window Attributes
 */
        case X_ChangeWindowAttributes:
            ((xChangeWindowAttributesReq *)(Vptr))->window  =
                    sub_id(    ((xChangeWindowAttributesReq
*)(Vptr))->window,
```

```
                        client,
                        WINDOW_TYPE);
/*
 * WARNING! There could be data that follows this protocol. That means
 * that at some point, code needs to be added to check what type of data
 * and if there are any resource IDs which need to be created or
translated.
 */
            break;
/*****************************************************************/


/*****************************************************************/
/*
 * These functions also use the standard ResourceReq structure, but
 * we must also solicit Expose type events when windows are mapped.
 *
 * X Map Window
 * X Map Subwindows
 */
        case X_MapWindow:
        case X_MapSubwindows:
            ((xResourceReq *)(Vptr))->id        =
                    sub_id( ((xResourceReq *)(Vptr))->id,
                        client,
                        WINDOW_TYPE);
            XSelectInput(    dest[client],    ((xResourceReq
*)(Vptr))->id,ExposureMask);
            break;
/*****************************************************************/


/*****************************************************************/
/*
 * X Configure Window
 */
        case X_ConfigureWindow:
            ((xConfigureWindowReq *)(Vptr))->window =
                sub_id( ((xConfigureWindowReq *)(Vptr))->window,
                        client,
                        WINDOW_TYPE);
/*
 * WARNING! There could be data that follows this protocol. That means
 * that at some point, code needs to be added to check what type of data
 * and if there are any resource IDs which need to be created or
translated.
 */
            break;
/*****************************************************************/
```

```
/*****************************************************************/
/*
 * X Open Font
 */
        case X_OpenFont:
            ((xOpenFontReq *)(Vptr))->fid       =
                    add_id( dest[client],
                            ((xOpenFontReq *)(Vptr))->fid,
                            FONT_TYPE,
                            CREATE_ID,
                            client);
            break;
/*****************************************************************/


/*****************************************************************/
/*
 * X Create Pixmap
 */
        case X_CreatePixmap:
            ((xCreatePixmapReq *)(Vptr))->pid       =
                    add_id( dest[client],
                            ((xCreatePixmapReq *)(Vptr))->pid,
                            PIXMAP_TYPE,
                            CREATE_ID,
                            client);
            ((xCreatePixmapReq *)(Vptr))->drawable  =
                    sub_id( dest[client],
                            ((xCreatePixmapReq *)(Vptr))->drawable,
                            client,
                            WINDOW_TYPE);
            break;
/*****************************************************************/


/*****************************************************************/
/*
 * Create A Graphics Context
 */
        case X_CreateGC:
            /* Substitute drawable ids */
            ((xCreateGCReq *)(Vptr))->drawable  =
                    sub_id( ((xCreateGCReq *)(Vptr))->drawable,
                            client,
                            WINDOW_TYPE);

            old_id  = ((xCreateGCReq *)(Vptr))->gc;
            /* Add a GC id */
            ((xCreateGCReq *)(Vptr))->gc        =
                    add_id( dest[client],
                            ((xCreateGCReq *)(Vptr))->gc,
```

```
                                GC_TYPE,
                                CREATE_ID,
                                client);
/*
 * Now determine the GC pointer for the original structure and
 * then call a routine to change that structure and flush it.
 */
            update_gc( (xCreateGCReq *)Vptr, get_gc(old_id),client );
#ifdef TRACE
fprintf(stderr,"PR::: Just got a CREATEGC request...gid:0x%x\n",
((xCreateGCReq *)(Vptr))->gc);
fprintf(stderr,".....old gid:0x%x\n",old_id);
fprintf(stderr,".....new gid:0x%x\n",((xCreateGCReq *)(Vptr))->gc);
#endif

            break;
/*********************************************************************/

/*********************************************************************/
/*
 * Change a Graphics Context.
 */
        case X_ChangeGC:
            old_id  = ((xChangeGCReq *)(Vptr))->gc;
            ((xChangeGCReq *)(Vptr))->gc        =
                    sub_id( ((xChangeGCReq *)(Vptr))->gc,
                                client,
                                GC_TYPE);
/*
 * Now determine the GC pointer for the original structure and
 * then call a routine to change that structure and flush it.
 */
            update_gc( (xChangeGCReq *)Vptr, get_gc(old_id),client );
#ifdef TRACE
fprintf(stderr,"PR::: Just got a CHANGEGC request...gid:0x%x\n",
((xChangeGCReq *)(Vptr))->gc);
fprintf(stderr,".....gc pointer for old_id:0x%x\n",old_id);
fprintf(stderr,".....gc pointer for new   :0x%x\n",
((xChangeGCReq *)(Vptr))->gc);
#endif
            break;
/*********************************************************************/


/*********************************************************************/
/*
 * X Copy GC
 */
        case X_CopyGC:
            ((xCopyGCReq *)(Vptr))->srcGC   =
                sub_id( ((xCopyGCReq *)(Vptr))->srcGC,
                        client,
```

```
                        GC_TYPE);
                ((xCopyGCReq *)(Vptr))->dstGC    =
                    sub_id( ((xCopyGCReq *)(Vptr))->dstGC,
                        client,
                        GC_TYPE);
                break;
/*********************************************************************/

/*********************************************************************/
/*
 * X Set Clip Rectangles
 */
        case X_SetClipRectangles:
            ((xSetClipRectanglesReq *)(Vptr))->gc    =
                sub_id( ((xSetClipRectanglesReq *)(Vptr))->gc,
                        client,
                        GC_TYPE);
                break;
/*********************************************************************/



/*********************************************************************/
/*
 * X Clear Area
 */
        case X_ClearArea:
            ((xClearAreaReq *)(Vptr))->window    =
                sub_id( ((xClearAreaReq *)(Vptr))->window,
                        client,
                        WINDOW_TYPE);
                break;
/*********************************************************************/


/*********************************************************************/
/*
 * X Copy Area
 */
        case X_CopyArea:
            ((xCopyAreaReq *)(Vptr))->gc             =
                sub_id( ((xCopyAreaReq *)(Vptr))->gc,
                        client,
                        GC_TYPE);
            ((xCopyAreaReq *)(Vptr))->srcDrawable    =
                sub_id( ((xCopyAreaReq *)(Vptr))->srcDrawable,
                        client,
                        WINDOW_TYPE);
            ((xCopyAreaReq *)(Vptr))->dstDrawable    =
                sub_id( ((xCopyAreaReq *)(Vptr))->dstDrawable,
                        client,
                        WINDOW_TYPE);
```

```
            break;
/*********************************************************************/

/*********************************************************************/
/*
 * X Copy Plane
 */
        case X_CopyPlane:
            ((xCopyPlaneReq *)(Vptr))->gc           =
                sub_id( ((xCopyPlaneReq *)(Vptr))->gc,
                        client,
                        GC_TYPE);
            ((xCopyPlaneReq *)(Vptr))->srcDrawable  =
                sub_id( ((xCopyPlaneReq *)(Vptr))->srcDrawable,
                        client,
                        WINDOW_TYPE);
            ((xCopyPlaneReq *)(Vptr))->dstDrawable  =
                sub_id( ((xCopyPlaneReq *)(Vptr))->dstDrawable,
                        client,
                        WINDOW_TYPE);
            break;
/*********************************************************************/


/*********************************************************************/
/*
 * X Poly Point
 */
        case X_PolyPoint:
            ((xPolyPointReq *)(Vptr))->gc       =
                sub_id( ((xPolyPointReq *)(Vptr))->gc,
                        client,
                        GC_TYPE);
            ((xPolyPointReq *)(Vptr))->drawable =
                sub_id( ((xPolyPointReq *)(Vptr))->drawable,
                        client,
                        WINDOW_TYPE);
            break;
/*********************************************************************/


/*********************************************************************/
/*
 * X Poly Line
 */
        case X_PolyLine:
            ((xPolyLineReq *)(Vptr))->gc        =
                sub_id( ((xPolyLineReq *)(Vptr))->gc,
                        client,
                        GC_TYPE);
            ((xPolyLineReq *)(Vptr))->drawable  =
```

```
                sub_id( ((xPolyLineReq *)(Vptr))->drawable,
                        client,
                        WINDOW_TYPE);
            break;
/*****************************************************************/

/*****************************************************************/
/*
 * X Poly Segment
 */
        case X_PolySegment:
            ((xPolySegmentReq *)(Vptr))->gc         =
                sub_id( ((xPolySegmentReq *)(Vptr))->gc,
                        client,
                        GC_TYPE);
            ((xPolySegmentReq *)(Vptr))->drawable   =
                sub_id( ((xPolySegmentReq *)(Vptr))->drawable,
                        client,
                        WINDOW_TYPE);
            break;
/*****************************************************************/


/*****************************************************************/
/*
 * X Poly Rectangle
 */
        case X_PolyRectangle:
            ((xPolyRectangleReq *)(Vptr))->gc       =
                sub_id( ((xPolyRectangleReq *)(Vptr))->gc,
                        client,
                        GC_TYPE);
            ((xPolyRectangleReq *)(Vptr))->drawable =
                sub_id( ((xPolyRectangleReq *)(Vptr))->drawable,
                        client,
                        WINDOW_TYPE);
            break;
/*****************************************************************/


/*****************************************************************/
/*
 * X Poly Arc
 */
        case X_PolyArc:
            ((xPolyArcReq *)(Vptr))->gc     =
                sub_id( ((xPolyArcReq *)(Vptr))->gc,
                        client,
                        GC_TYPE);
            ((xPolyArcReq *)(Vptr))->drawable   =
                sub_id( ((xPolyArcReq *)(Vptr))->drawable,
                        client,
```

```
                        WINDOW_TYPE);
            break;
/*********************************************************************/


/*********************************************************************/
/*
 * X Fill Poly
 */
        case X_FillPoly:
            ((xFillPolyReq *)(Vptr))->gc        =
                sub_id( ((xFillPolyReq *)(Vptr))->gc,
                        client,
                        GC_TYPE);
            ((xFillPolyReq *)(Vptr))->drawable  =
                sub_id( ((xFillPolyReq *)(Vptr))->drawable,
                        client,
                        WINDOW_TYPE);
#ifdef TRACE
fprintf(stderr,"PR:: Just got a FILLPOLY request...gc:0x%x\n",
((xFillPolyReq *)(Vptr))->gc);
#endif
            break;
/*********************************************************************/


/*********************************************************************/
/*
 * X Poly Fill Rectangle
 */
        case X_PolyFillRectangle:
            ((xPolyFillRectangleReq *)(Vptr))->gc       =
                sub_id( ((xPolyFillRectangleReq *)(Vptr))->gc,
                        client,
                        GC_TYPE);
            ((xPolyFillRectangleReq *)(Vptr))->drawable =
                sub_id( ((xPolyFillRectangleReq *)(Vptr))->drawable,
                        client,
                        WINDOW_TYPE);
#ifdef TRACE
fprintf(stderr,"PR:: Just got a FILLRECTANGLE request...gc:0x%x\n",
((xPolyFillRectangleReq *)(Vptr))->gc);
#endif

            break;
/*********************************************************************/


/*********************************************************************/
/*
 * X Poly Fill Arc
 */
```

```
        case X_PolyFillArc:
            ((xPolyFillArcReq *)(Vptr))->gc      =
                sub_id( ((xPolyFillArcReq *)(Vptr))->gc,
                        client,
                        GC_TYPE);
            ((xPolyFillArcReq *)(Vptr))->drawable   =
                sub_id( ((xPolyFillArcReq *)(Vptr))->drawable,
                        client,
                        WINDOW_TYPE);
            break;
/*********************************************************************/


/*********************************************************************/
/*
 * X Put Image
 */
        case X_PutImage:
            ((xPutImageReq *)(Vptr))->gc        =
                sub_id( ((xPutImageReq *)(Vptr))->gc,
                        client,
                        GC_TYPE);
            ((xPutImageReq *)(Vptr))->drawable  =
                sub_id( ((xPutImageReq *)(Vptr))->drawable,
                        client,
                        WINDOW_TYPE);
            break;
/*********************************************************************/


/*********************************************************************/
/*
 * X Poly Text 8
 */
        case X_PolyText8:
            ((xPolyText8Req *)(Vptr))->gc       =
                sub_id( ((xPolyText8Req *)(Vptr))->gc,
                        client,
                        GC_TYPE);
            ((xPolyText8Req *)(Vptr))->drawable =
                sub_id( ((xPolyText8Req *)(Vptr))->drawable,
                        client,
                        WINDOW_TYPE);
            break;
/*********************************************************************/


/*********************************************************************/
/*
 * X Poly Text 16
 */
        case X_PolyText16:
```

```
            ((xPolyText16Req *)(Vptr))->gc      =
                sub_id( ((xPolyText16Req *)(Vptr))->gc,
                        client,
                        GC_TYPE);
            ((xPolyText16Req *)(Vptr))->drawable    =
                sub_id( ((xPolyText16Req *)(Vptr))->drawable,
                        client,
                        WINDOW_TYPE);
            break;
/*********************************************************************/


/*********************************************************************/
/*
 * X Image Text 8
 */
        case X_ImageText8:
            ((xImageText8Req *)(Vptr))->gc      =
                sub_id( ((xImageText8Req *)(Vptr))->gc,
                        client,
                        GC_TYPE);
            ((xImageText8Req *)(Vptr))->drawable    =
                sub_id( ((xImageText8Req *)(Vptr))->drawable,
                        client,
                        WINDOW_TYPE);
            break;
/*********************************************************************/


/*********************************************************************/
/*
 * X Image Text 16
 */
        case X_ImageText16:
            ((xImageText16Req *)(Vptr))->gc     =
                sub_id( ((xImageText16Req *)(Vptr))->gc,
                        client,
                        GC_TYPE);
            ((xImageText16Req *)(Vptr))->drawable   =
                sub_id( ((xImageText16Req *)(Vptr))->drawable,
                        client,
                        WINDOW_TYPE);
            break;
/*********************************************************************/


/*********************************************************************/
/*
 * X Create Colormap
 */
        case X_CreateColormap:
            ((xCreateColormapReq *)(Vptr))->mid     =
```

```
                    add_id( dest[client],
                            ((xCreateColormapReq *)(Vptr))->mid,
                            COLORMAP_TYPE,
                            CREATE_ID,
                            client);
                ((xCreateColormapReq *)(Vptr))->window  =
                    sub_id( ((xCreateColormapReq *)(Vptr))->window,
                            client,
                            WINDOW_TYPE);
                ((xCreateColormapReq *)(Vptr))->visual  =
                    sub_id( ((xCreateColormapReq *)(Vptr))->visual,
                            client,
                            WINDOW_TYPE);
                break;
/*****************************************************************/


/*****************************************************************/
/*
 * X Copy Colormap And Free
 */
        case X_CopyColormapAndFree:
            ((xCopyColormapAndFreeReq *)(Vptr))->mid        =
                    add_id( dest[client],
                            ((xCopyColormapAndFreeReq *)(Vptr))->mid,
                            COLORMAP_TYPE,
                            CREATE_ID,
                            client);
                old_id          =    ((xCopyColormapAndFreeReq
*)(Vptr))->srcCmap;
                ((xCopyColormapAndFreeReq *)(Vptr))->srcCmap    =
                    sub_id( ((xCopyColormapAndFreeReq *)(Vptr))->srcCmap,
                            client,
                            WINDOW_TYPE);
                free_id(old_id,client);
                break;
/*****************************************************************/


/*****************************************************************/
/*
 * X Alloc Color
 */
        case X_AllocColor:                      /* ? */
            ((xAllocColorReq *)(Vptr))->cmap    =
                sub_id( ((xAllocColorReq *)(Vptr))->cmap,
                        client,
                        COLORMAP_TYPE);
            break;
/*****************************************************************/
```

```
/*****************************************************************/
/*
 * X Alloc Named Color
 */
        case X_AllocNamedColor:                 /* ? */
            ((xAllocNamedColorReq *)(Vptr))->cmap   =
                sub_id( ((xAllocNamedColorReq *)(Vptr))->cmap,
                        client,
                        COLORMAP_TYPE);
            break;
/*****************************************************************/


/*****************************************************************/
/*
 * X Alloc Color Cells
 */
        case X_AllocColorCells:                 /* ? */
            ((xAllocColorCellsReq *)(Vptr))->cmap   =
                sub_id( ((xAllocColorCellsReq *)(Vptr))->cmap,
                        client,
                        COLORMAP_TYPE);
            break;
/*****************************************************************/


/*****************************************************************/
/*
 * X Alloc Color Planes
 */
        case X_AllocColorPlanes:                /* ? */
            ((xAllocColorPlanesReq *)(Vptr))->cmap  =
                sub_id( ((xAllocColorPlanesReq *)(Vptr))->cmap,
                        client,
                        COLORMAP_TYPE);
            break;
/*****************************************************************/


/*****************************************************************/
/*
 * X Free Colors
 */
        case X_FreeColors:                      /* ? */
            ((xFreeColorsReq *)(Vptr))->cmap    =
                sub_id( ((xFreeColorsReq *)(Vptr))->cmap,
                        client,
                        COLORMAP_TYPE);
            break;
/*****************************************************************/
```

```
/*********************************************************************/
/*
 * X Store Colors
 */
        case X_StoreColors:                     /* ? */
            ((xStoreColorsReq *)(Vptr))->cmap   =
                sub_id( ((xStoreColorsReq *)(Vptr))->cmap,
                        client,
                        COLORMAP_TYPE);
            break;
/*********************************************************************/


/*********************************************************************/
/*
 * X Store Named Color
 */
        case X_StoreNamedColor:                 /* ? */
            ((xStoreNamedColorReq *)(Vptr))->cmap   =
                sub_id( ((xStoreNamedColorReq *)(Vptr))->cmap,
                        client,
                        COLORMAP_TYPE);
            break;
/*********************************************************************/


/*********************************************************************/
/*
 * If we get here, we received some X protocol which we don't want
 * to handle. Notify the operator about this and exit.
 */
        default:
            fprintf(stderr,"PR:: Received unknown X protocol packet.\n");
#ifdef GO_AWAY
            go_away();
#endif
/*********************************************************************/


    } /* end switch */

    return(FALSE);

} /* end prproto */
/*****************************************************/
```

```
/*
 * File         : prutil.c
 * Author       : P. Fitzgerald - SwRI
 * Date         : 10/17/89
 * Description  : This file contains the code for the Protocol Receiver.
 * Utility routines.
 */
#include <stdio.h>
#include <rpc/rpc.h>
#include <utmp.h>
#include <sys/types.h>
#include <rpcsvc/rusers.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <fcntl.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <sys/wait.h>
#include <signal.h>
#include <errno.h>
#include <X11/X.h>
#define NEED_REPLIES
#define NEED_EVENTS
#include <X11/Xproto.h>
#include <X11/Xlib.h>
#include </home/overflow/X11/include/Xlibint.h>
#include "../includes/ds_manage.h"
#include "../includes/smtypes.h"
#include "../includes/smdef.h"
#include "../includes/dist.h"
#include "../includes/ecodes.h"
#include "../includes/dsb.bm"
#include "../includes/pr.bm"

/* LOCAL DEFINES */

/* EXTERNAL VARIABLES */
extern Display              *dest[MAX_CLIENTS];
extern XID                  source_default_gc[MAX_CLIENTS];
extern XID                  source_root[MAX_CLIENTS];
extern Display              *local_dpy;
extern Screen               *local_screens[MAX_SCREENS];
extern struct MC_SHMEMORY   *shmem;
extern int                  pm_fd;
extern int                  number_of_clients;
extern unsigned short       my_port;
extern struct XIDmap        *idmap;
extern  char                hostname[HOSTNAMLEN];

/* EXTERNAL ROUTINES */
extern              set_alarm();
```

```
extern              clear_alarm();
extern              callme();
extern              xdr_PortID();
extern              xdr_RegRecv();

/* GLOBAL FUNCTIONS */
void                free_id();
void                cause_expose_event();
void                append_id();
int                 check_for_new_client();
XID                 sub_id(),add_id(),unsub_id();
GC                  get_gc();
XID                 unknown_id(),create_new_window();
XID                 create_new_gc(),gc_to_win();
XID                 get_initial_gc();
XID                 create_new_colormap();
void                get_window_attributes();
void                get_window_state();
void                get_gc_state();
XID                 read_xid();
void                final_shutdown();

/* GLOBAL VARIABLES */
int                 channel;

/*****************************************************/
/*
 * This routine determines if the protocol is from a client
 * which has not been seen before. If it is, then depending
 * on what type of request it is, a GC or a window is
 * created for it.
 */
int
check_for_new_client(client)
    register int            client;
{
    register Display        *dpy;

#ifdef TRACE
fprintf(stderr,"PR:: check_for_new_client\n");
#endif

/*
 * If the display pointer for this client is NULL, then this
 * is a new client. We must therefore, open up a new connection
 * assuming that we haven't reached the maximum number of clients
 * which we are allowed.
 */
#ifdef TRACE
fprintf(stderr,"PR:: checking if %d is new.\n",client);
#endif
    if (number_of_clients>=MAX_CLIENTS) {
```

```
        fprintf(stderr,"PR:: Client MAX overflow.\n");
        return(TRUE); /* ignore */
    }
    else if (dest[client]!=NULL)
        return(FALSE);

#ifdef TRACE
fprintf(stderr,"PR:: FOUND a NEW CLIENT.\n");
fprintf(stderr,"PR:: Opening a connection to :<%s>\n",getenv("DISPLAY"));
#endif

    /* open up the display to multi-cast to */
    dpy = XOpenDisplay(getenv("DISPLAY"));
#ifdef TRACE
fprintf(stderr,"PR:: Opened new connection, bufmax:0x%x, buffer: 0x%x,
bufptr: 0x%x\n",
dpy->bufmax, dpy->buffer, dpy->bufptr);
fprintf(stderr,".....max_request_size: 0x%x\n", dpy->max_request_size );
#endif
    if (dpy==NULL) {
        perror("Cannot open display.");
        go_away();
    }
    number_of_clients++;
    dest[client]        = dpy;

#ifdef TRACE
fprintf(stderr,"PR::   A   new   client   brings   us   up   to   %d
clients.\n",number_of_clients);
fprintf(stderr,"PR:: destination display:0x%x\n",dpy);
fprintf(stderr,"PR:: destination display fd:%d\n",dpy->fd);
#endif


/*
 * Now add this connection's default ids to the linked list
 */

add_default(shmem->source_default_gc[channel],shmem->default_gc,client);
    add_default(shmem->source_root[channel],shmem->root,client);

#ifdef TRACE
fprintf(stderr,"..adding defaults (clnt: %d):\n",client );
fprintf(stderr,"      gc:          src:   0x%x,   dest:
0x%x\n",shmem->source_default_gc[channel],shmem->default_gc);
fprintf(stderr,"      root:   src:   0x%x,   dest:
0x%x\n",shmem->source_root[channel],shmem->root);

fprintf(stderr,"src   root:   0x%x,   dest   root:
0x%x\n",shmem->source_root[channel],shmem->root);
#endif
```

```
/*
 * Fill in some data structures
 */
    shmem->clients[channel] = client;

#ifdef TRACE
fprintf(stderr,"PR:: channel:%d client:%d\n",channel,client);
fprintf(stderr,"PR:: clients[%d]=%d\n",channel,shmem->clients[channel]);
#endif

    /* return the DO NOT IGNORE flag */
    return(FALSE);

} /* end check_for_new_client */
/*****************************************************/

/*****************************************************/
/*
 * This routine causes an expose event to be sent to
 * the local X server for the given window. It is called
 * after receiving notification from the Protocol Multiplexer.
 */
void
cause_expose_event()
{
    register XID        window;
    XExposeEvent        event;
    XID                 client;
    XWindowAttributes   wats;
    Status              stat;

/*
 * Call routine to read the xid
 */
    window  = read_xid();

/*
 * Retrieve the client from the upper bytes of XID
 */
    client  = CLIENT_ID(window);

#ifdef TRACE
fprintf(stderr,"PR:: cause_expose_event for dest window:0x%x\n",window);
fprintf(stderr,"PR:: client for expose event is        :0x%x\n",client);
fprintf(stderr,"PR::  local_dpy
:0x%x\n",local_dpy);
#endif

/*
 * Retrieve the window attributes so we can get the size
 */
#ifdef TRACE
```

```
fprintf(stderr,"PR:: just before getwindowattributes\n");
#endif
    stat    = XGetWindowAttributes(local_dpy,window,&wats);
#ifdef TRACE
fprintf(stderr,"PR:: getwindowattributes finished\n");
fprintf(stderr,"PR:: width:%d height:%d x:%d y:%d\n",
wats.width,wats.height,wats.x,wats.y);
#endif

/*
 * Note: the return values for some Xlib calls are not consistent, so in
 * this case we can't check against Success.
 */
    if (stat != 1) {
        fprintf(stderr,"PR::  cause_expose_event  (XGetWindowAttributes)
failed.\n");
#ifdef GO_AWAY
        go_away();
#endif
    }


/*
 * Now build an expose event and send it to the local
 * server.
 */
    event.type      = Expose;
    event.send_event= TRUE;
    event.display   = local_dpy;
    event.window    = window;
    event.x         = wats.x;
    event.y         = wats.y;
    event.height    = wats.height;
    event.width     = wats.width;
    event.count     = 0;
    XSendEvent(local_dpy,window,TRUE,ExposureMask,&event);
    XFlush(local_dpy);

#ifdef TRACE
fprintf(stderr,"PR:: cause local expose, win: 0x%x\n",window);
fprintf(stderr,"PR:: event sent and flushed.\n");
#endif

} /* end cause_expose_event */
/*****************************************************/


/*****************************************************/
/*
 * This routine adds an XID to the linked list.
 */
XID
```

```
add_id(dest_dpy,source,type,operation,client)
    register Display            *dest_dpy;
    register XID                source;
    register int                type;
    register int                operation;
    register int                client;
{
    register struct XIDmap      *p;
    int                         ignore;

/*
 * Find the end of the linked list
 */
    p   = idmap;
    while (p->next!=NULL)
        p = p->next;

/*
 * Allocate another cell.
 */
    p->next = (struct XIDmap *)malloc(sizeof(struct XIDmap));
    if (p->next==NULL) {
        perror("Can't malloc memory. No multicasting.\n");
#ifdef GO_AWAY
        go_away();
#endif
    }

/*
 * Set up the cell's values
 */
    p           = p->next;
    p->source   = source;
    p->type     = type;

/*
 * Handle default ids, default gc, default window etc.
 */
    if ( (p->source - (p->source&0xff0000))==0) {
        p->dest = local_screens[DefaultScreen(dest_dpy)]->default_gc->gid;
        ignore  = TRUE;
    }
    else
        p->dest     = XAllocID(dest_dpy);

    p->next     = NULL;

/*
 * Handle creation and mapping, flag both
 */
    if (!ignore) {
        if (operation==MAP_ID)
```

```
            p->mapped   = TRUE;
        else
            p->mapped   = FALSE;
        if (operation==CREATE_ID)
            p->created  = TRUE;
        else
            p->created  = FALSE;
        p->client       = client;
    }  /* end if not ignore */

    return(p->dest);

}  /* end add_id */
/*****************************************************/

/*****************************************************/
/*
 * This routine frees all occurrances of a client
 * from the XIDmap.
 */
void
free_client(client)
    register int            client;
{
    register struct XIDmap  *ptr;
    register struct XIDmap  *last;
    register struct XIDmap  *next;

#ifdef TRACE
fprintf(stderr,"PR:: free_client called for client %d\n",client);
#endif
    last    = idmap;
    ptr     = last->next;
    if (ptr==NULL)
        return;

    while (ptr != NULL) {
        if (ptr->client==client) {
            next        = ptr->next;
            last->next  = next;
            free(ptr);
            ptr         = next;
#ifdef TRACE
fprintf(stderr,"PR:: just freed an entry\n");
#endif
        } /* end if */
        else {
#ifdef TRACE
fprintf(stderr,"PR:: skipped one entry\n");
#endif
            last        = ptr;
            ptr         = ptr->next;
```

```
        } /* end else */

    } /* end while */

#ifdef TRACE
fprintf(stderr,"PR:: at end of free_client routine\n");
ptr = idmap;
fprintf(stderr,"PR:: IDMAP BELOW\n");
while (ptr!=NULL) {
fprintf(stderr,"PR:: client:%d\n",ptr->client);
ptr = ptr->next;
}
#endif

} /* end free_client */
/*****************************************************/

/*****************************************************/
/*
 * This routine frees an id from the linked list.
 */
void
free_id(id,client)
    register XID        id;
    register int    client;
{
    register struct XIDmap  *ptr;
    register struct XIDmap  *last;

/*
 * Find the correct entry in linked list
 */
    ptr     = idmap;
    last    = NULL;
    while (ptr->source != id &&
           ptr->client != client)  {
        last    = ptr;
        ptr     = ptr->next;
    }

    /* if the first id is to be removed */
    if (last==NULL)
        idmap   = ptr->next;
    else
        last->next  = ptr->next;

    if (ptr->type==GC_TYPE)
        XFreeGC(dest[client],ptr->gc);
    free(ptr);

} /* end free_id */
/*****************************************************/
```

```
/*****************************************************/
/*
 * This routine takes the source id and returns the
 * destination gc structure pointer.
 */
GC get_gc(source_id)
    register    XID             source_id;
{
    register    struct  XIDmap  *ptr;

/*
 * Find the correct entry in the linked list
 */
    ptr = idmap;
    while (ptr->source != source_id &&
           ptr->next   != NULL)
        ptr = ptr->next;

/*
 * If we get here and take first 'if', then
 * we can't find the associated gc.
 */
    if (ptr->source != source_id)  {
        fprintf(stderr,"PR:: Unable to find gc for existing gid.\n");
        go_away();
    }
    else
        return(ptr->gc);

} /* end get_gc */
/*****************************************************/


/*****************************************************/
/*
 * This routine takes the source id and returns the
 * destination id.
 */
XID sub_id(source_id,client,type)
    register    XID             source_id;
    register    int             client;
    register    int             type;
{
    register    struct  XIDmap  *ptr;

/*
 * Find the correct entry in the linked list
 */
    ptr = idmap;
    while (ptr->source != source_id &&
           ptr->next   != NULL)
```

```
        ptr = ptr->next;

/*
 * If we get here and take this 'if' that means
 * that we have some x protocol for an id which we
 * have never seen before. Lets attempt to create
 * the XID 'thing' if we can.
 */
    if (ptr->source != source_id)
        return(unknown_id(source_id,type,client));
    else
        return(ptr->dest);

} /* end sub_id */
/*****************************************************/

/*****************************************************/
/*
 * This routine takes the destination id and
 * returns the source id.
 */
XID
unsub_id(dest_id)
    register    XID     dest_id;
{
    register    struct  XIDmap      *ptr;

/*
 * Find the correct entry
 */
    ptr = idmap;
    while (ptr->dest != dest_id &&
           ptr->next   != NULL)
        ptr = ptr->next;

/*
 * Return the substitution id
 */
    if (ptr->dest != dest_id)
        return(dest_id);
    else
        return(ptr->source);

} /* end unsub_id */
/*****************************************************/

/*****************************************************/
/*
 * This routine adds the basic XID default IDs for mapping
 * into the linked list - without allocating any XIDs.
 */
add_default(source_id,dest_id,client)
```

```
    register    XID     source_id;
    register    XID     dest_id;
    register    int client;
{
    register    struct XIDmap   *ptr;

/*
 * Go down the linked list to the end
 */
    ptr     = idmap;
    if (ptr->source != NULL) {
        while (ptr->next != NULL)
            ptr = ptr->next;

/*
 * Allocate some memory for this node
 */
        ptr->next   = (struct XIDmap *)malloc(sizeof(struct XIDmap));
        if (ptr->next==NULL) {
            perror("Can't malloc memory. No multicasting.\n");
            go_away();
        }
        ptr = ptr->next;
    } /* end if first slot not null */

    ptr->source     = source_id;
    ptr->dest       = dest_id;
    ptr->next       = NULL;
    ptr->client     = client;
    ptr->mapped     = TRUE;     /* indeed it already does exist */
    ptr->created    = TRUE;     /* ditto                        */

} /* end add_default */
/*******************************************************/

/*******************************************************/
/*
 * This routine travels down the linked list of XIDs
 * and searches for a window id that has been mapped.
 * When it does, it returns the next one or -1 if there
 * is not one at all.
 */
Window
next_window(client)
    register XID    *client;
{
    static struct   XIDmap      *head = NULL;
    static struct   XIDmap      *ptr  = NULL;

#ifdef TRACE
fprintf(stderr,"PR::next_window\n");
#endif
```

```
    if (idmap==NULL)
        return(NULL);

    if (head==NULL) {
        head    = idmap;
        ptr     = idmap;
    }
    if (ptr==NULL)
        ptr     = idmap;

    while (1) {
        ptr = ptr->next;
        if (ptr==NULL)
            return(NULL);
        else
            if (ptr->type==WINDOW_TYPE &&
                ptr->mapped ) {
                *client = ptr->client;
#ifdef TRACE
fprintf(stderr,"PR::   found   a   mapped   window:0x%x
client:0x%x\n",ptr->dest,ptr->client);
#endif
                return(ptr->dest);
            }
    } /* end while */

} /* end next window */
/*****************************************************/

/*********************************************************************/
/*
 * This routine takes an XID from an x protocol packet which has
 * not been seen before on a create etc. and tries to determine what
 * type of id it is and how can we re-create it.
 */
XID
unknown_id(id,type,client)
    register XID        id;
    register int        type;
    register int        client;
{

/*
 * What type is it?
 */
    switch (type) {
        case        GC_TYPE:
#ifdef TRACE
fprintf( stderr, "PR:: !!!!(unknown_id) we got a GC_TYPE!!!!\n" );
#endif
            return(create_new_gc(id,client));
```

```
        case        WINDOW_TYPE:
            return(create_new_window(id,client));
        case        COLORMAP_TYPE:
#ifdef C_TRACE
fprintf( stderr,"PR:: (unknown_id) COLORMAP_TYPE id: %d, client: %d\n",
id, cient );
#endif
            return(create_new_colormap(id,client));
        default:
            fprintf(stderr,"PR:: Unknown XID type (unknown_id).\n");
#ifdef GO_AWAY
            go_away();
#endif
            break;
    } /* end switch */
    return(-1);

}  /* end unknown_id */
/*********************************************************************/

/*********************************************************************/
/*
 * This routine creates a window on the target display, based on
 * its associated source window's attributes.
 */
XID
create_new_window(source_win,client)
    register Window         source_win;
    register int            client;
{
    XWindowAttributes       wat;
    XSetWindowAttributes    swat;
    Window                  newwin;
    int                     screen;
    Colormap                cmap;
    Status                  stat;
    unsigned long           background;
    XID                     parent;
    Pixmap                  border_pm;
    Pixmap                  icon_pm;

#ifdef TRACE
fprintf(stderr,"PR:: create_new_window for:0x%x\n",source_win);
fprintf(stderr,"PR:: client:%d\n",client);
fprintf(stderr,"PR:: getting window attributes...\n");
fprintf(stderr,"PR:: Channel:%d\n",channel);
#endif

/*
 * First get the window attributes from the source window
 */
#ifdef TRACE
```

```
fprintf(stderr,"PR>>>   GET   WINDOW   ATTRIBUTES   FOR
SOURCE:0x%x\n",source_win);
#endif
    get_window_attributes(source_win,channel,&wat,&background,&parent);

#ifdef TRACE
fprintf(stderr,"parent window before trans: 0x%x\n",parent);
#endif

    /* translate the parent to the destination display. */

    parent = sub_id(parent,client,WINDOW_TYPE);

#ifdef TRACE
fprintf(stderr,"parent window after trans:  0x%x\n",parent);
#endif

    /* put Display Sharing border around top parent window */

    if (parent == shmem->root)
        wat.border_width = 5;

#ifdef TRACE
fprintf(stderr,"Border width is %d\n",wat.border_width);
#endif

/*
 * Next create a window on the target that looks the same as the source
 */
#ifdef TRACE
fprintf(stderr,"PR:: X_CreateSimpleWindow called.\n");
#endif
    newwin  = XCreateSimpleWindow(dest[client],parent,
                wat.x,wat.y,
                wat.width,wat.height,
                wat.border_width,
                background,background);

    if (newwin==BadAlloc ||
        newwin==BadMatch ||
        newwin==BadValue ||
        newwin==BadWindow ) {
        perror("Unable to create window on target.");
        XUngrabServer(local_dpy);
        return(source_win);
    }

/*
 * Retrieve some default information and then set
 * the colormap to be the default colormap.
 */
        screen  = DefaultScreen(dest[client]);
```

```
        cmap    = DefaultColormap(dest[client],screen);
        stat    = XSetWindowColormap(dest[client],newwin,cmap);
        if (stat==BadColor)
            fprintf(stderr,"PR:: XSetWindowColormap> BadColor.\n");
        else if (stat==BadMatch)
            fprintf(stderr,"PR:: XSetWindowColormap> BadMatch.\n");
        else if (stat==BadWindow)
            fprintf(stderr,"PR:: XSetWindowColormap> BadWindow.\n");

#ifdef TRACE
fprintf(stderr,"PR:: background (and border): 0x%x\n", background);
#endif

/*
 * Create a border pixmap for display sharing
 */
    border_pm   =
XCreatePixmapFromBitmapData(dest[client],newwin,dsb_bits,
                        dsb_width,dsb_height,
                        BlackPixel(dest[client],screen),
                        WhitePixel(dest[client],screen),
                        DefaultDepth(dest[client],screen));

    icon_pm = XCreatePixmapFromBitmapData(dest[client],newwin,pr_bits,
                        pr_width,pr_height,
                        BlackPixel(dest[client],screen),
                        WhitePixel(dest[client],screen),
                        DefaultDepth(dest[client],screen));

    XSetStandardProperties(dest[client],newwin,"Shared
Window","SharedWin",
            icon_pm,NULL,0,NULL);

/*
 * Now change the border pixmap to the correct one.
 * Also, propagate any events back to parent,
 * old -> CWBorderPixmap | CWBackPixel | CWDontPropagate,
 */
    swat.do_not_propagate_mask  = FALSE;
    if (parent == shmem->root) {
        swat.border_pixmap      = border_pm;
        stat = XChangeWindowAttributes(dest[client],newwin,CWBorderPixmap
| CWDontPropagate,&swat);
    }
    else {
        s                   t                   a                   t    =
XChangeWindowAttributes(dest[client],newwin,CWDontPropagate, &swat);
    }
    if (stat==BadAccess ||
        stat==BadColor  ||
        stat==BadCursor ||
        stat==BadMatch  ||
```

```
        stat==BadPixmap ||
        stat==BadValue  ||
        stat==BadWindow ) {
        perror("PR:: XChangeWindowAttributes (%d):");
    }

    /* Set window background to the correct color */
    XSetWindowBackground(dest[client],newwin,background);

#ifdef TRACE
fprintf(stderr,"PR:: default window border: 0x%x, bg: 0x%x\n", background,
background );
#endif

/*
 * Ask to receive exposure events on this window
 */
#ifdef TRACE
fprintf(stderr,"PR:: Soliciting expose events for new window:0x%x
client:%d\n",newwin,client);
#endif
    stat = XSelectInput(dest[client],newwin,
                ButtonPressMask         |
                ButtonReleaseMask       |
                EnterWindowMask         |
                LeaveWindowMask         |
                PointerMotionMask       |
                PointerMotionHintMask   |
                Button1MotionMask       |
                Button2MotionMask       |
                Button3MotionMask       |
                Button4MotionMask       |
                Button5MotionMask       |
                ButtonMotionMask        |
                KeymapStateMask         |
                NoEventMask             |
                KeyPressMask            |
                KeyReleaseMask          |
                ExposureMask            |
                VisibilityChangeMask    |
                OwnerGrabButtonMask     |
                StructureNotifyMask     |
                ResizeRedirectMask      |
                SubstructureNotifyMask  |
                FocusChangeMask         |
                PropertyChangeMask      |
                ColormapChangeMask      |
                ExposureMask
                );

#ifdef TRACE
fprintf(stderr,"PR:: status from xselectinput is:<%s>\n",ecodes[stat]);
```

```
#endif

/*
 * Map this window
 */
#ifdef TRACE
fprintf(stderr,"PR:: XMapWindow.\n");
#endif
    XMapWindow(dest[client],newwin);

/*
 * Send all this to server NOW!!
 */
#ifdef TRACE
fprintf(stderr,"PR:: Flush it.\n");
#endif
    XFlush(dest[client]);
#ifdef TRACE
fprintf(stderr,"PR:: ...flushed.\n");
#endif

/*
 * Now add this id into the linked list
 */
    append_id(source_win,newwin,client,WINDOW_TYPE,NULL);

    return(newwin);

} /* end create_new_window */
/*****************************************************************/

/*****************************************************************/
/*
 * This routine creates a Default colormap entry and stores it in
 * the linked list for XIDs.
 */
XID
create_new_colormap(source_cmap,client_index)
    register    XID     source_cmap;
    register    int     client_index;
{
    Colormap            cmap;
    static char         colors[4][15] = {"red","yellow","green","blue"};
    XColor              ids[4];
    int                 i;


/*
 * First create (at least get the id of) the default colormap
 */
    cmap    =
DefaultColormap(dest[client_index],DefaultScreen(dest[client_index]));
```

```
/*
 * Allocate some colors in that map
 */
    for( i = 0 ; i < 4 ; i++) {
        XParseColor(dest[client_index],cmap,colors[i],&ids[i]);
        XAllocColor(dest[client_index],cmap,&ids[i]);
    }

/*
 * Now store this in the linked list so it can be substituted for
 */
    append_id(source_cmap,client_index,COLORMAP_TYPE,NULL);

    return(cmap);

} /* end create_new_colormap  */
/*****************************************************************/

/*****************************************************************/
/*
 * This routine creates a graphics context on the target display,
 * based on its associated source graphics context.
 */
XID
create_new_gc(source_gid,client_index)
    register XID        source_gid;
    register int        client_index;
{

    XGCValues   source_values;
    GC          dest_gc;
    XID         window;

#ifdef TRACE
fprintf(stderr,"PR::create_new_gc\n");
#endif

#ifdef TRACE
fprintf(stderr,"PR:: get_initial_gc values for:0x%x\n",source_gid);
fprintf(stderr,"PR:: &source_values:0x%x\n",&source_values);
#endif

    window = get_initial_gc(source_gid,&source_values);

#ifdef TRACE
fprintf(stderr,"PR:: get_initial_gc -> [%x](win)\n",window);
fprintf(stderr,"PR::    XCreateGC    for    local    gc,    source_gid:
0x%x\n",source_gid);
fprintf(stderr,"PR:: background:0x%x foreground:0x%x\n",
source_values.background,source_values.foreground);
#endif
```

```
    dest_gc = XCreateGC(dest[client_index],
                sub_id(window,client_index,WINDOW_TYPE),
                    GCForeground | GCBackground,
                    &source_values);

#ifdef TRACE
fprintf(stderr,"PR::    XSetForeground    dest_gc.foreground:%d\n",
source_values.foreground);
fprintf(stderr,"PR::    XSetBackground    dest_gc.background:%d\n",
source_values.background);
fprintf(stderr,"PR:: the GC we created is :0x%x\n",dest_gc);
#endif

    append_id(source_gid,dest_gc->gid,client_index,GC_TYPE,dest_gc);
    return(dest_gc->gid);

} /* end create_new_gc */
/*****************************************************************/

/*****************************************************************/
void
append_id(source_id,dest_id,client_index,type,gc)
    register XID        source_id;
    register XID        dest_id;
    register int        client_index;
    register int        type;
    register GC         gc;
{
    register struct XIDmap      *p;

#ifdef TRACE
if (type == WINDOW_TYPE)
fprintf(stderr,"appending    0x%x    ->    0x%x,clnt:    %d
(window)\n",source_id,dest_id,client_index);
if (type == GC_TYPE)
fprintf(stderr,"appending    0x%x    ->    0x%x,clnt:    %d
(gc:0x%x)\n",source_id,dest_id,client_index,gc);
#endif

    p   = idmap;
    while (p->next!=NULL)
        p = p->next;
    p->next = (struct XIDmap *)malloc(sizeof(struct XIDmap));
    if (p->next==NULL) {
        perror("Can't malloc memory. No multicasting.\n");
        go_away();
    }
    p           = p->next;
    p->source   = source_id;
    p->type     = type;
    p->dest     = dest_id;
```

```
    p->next     = NULL;
    p->created  = TRUE;
    p->client   = client_index;
    p->gc       = gc;
    if (p->type==WINDOW_TYPE)
        p->mapped   = TRUE;

}  /* end append_id */
/*********************************************************************/


/*********************************************************************/
/*
 * This routine makes a request of the Protocol Distributor. The
 * request is to retrieve the window attributes state information
 * for a given source channel and xid. This receiver's port number
 * is also passed to indicate where to send the window attributes
 * information once it is obtained.
 */
void
get_window_attributes(gid,chan,ptr,background,parent)
    register XID                    gid;
    register int                    chan;
    register XWindowAttributes      *ptr;
    register unsigned long          *background;
    register XID                    *parent;
{

#ifdef TRACE
fprintf(stderr,"PR:: GET_WINDOW_ATTRIBUTES - need state information.\n");
#endif

/*
 * Set the get window attributes parameters and flags and wait.
 */
    shmem->wat_channel  = chan;
    shmem->wat_id       = gid;
    shmem->wat_port     = shmem->pr_port;
    shmem->get_wat      = TRUE;

/*
 * Send a signal to the Distributor to get him to check
 * shared memory.
 */

#ifdef TRACE
fprintf(stderr,"PR:: sending signal to pd\n");
#endif
    kill(shmem->pd_pid,SIGUSR1);

#ifdef TRACE
fprintf(stderr,"PR::  made  request  for  window  attributes  id:0x%x
```

```
channel:%d\n",
gid,chan);
fprintf(stderr,"PR:: waiting for a response now...\n");
#endif

/*
 * Now wait on that  flag to turn FALSE
 */
    set_alarm(30);      /* 30 seconds for alarm */
    while (shmem->get_wat) {
        if (shmem->sm_status==DIE)
            go_away();
        sleep(1);
    } /* end while */
    clear_alarm();

/*
 * Now we need to read the file descriptor from the
 * Protocol Multiplexer, dispatching requests, until
 * the WATS information is sent to us.
 */
    wait_for_WATS(ptr,background,parent);

#ifdef TRACE
fprintf(stderr,"PR:: window attribute request fulfilled...\n");
fprintf(stderr,"PR::   background:0x%x,   parent:0x%x\n",   *background,
*parent);
#endif

} /* end get_window_attributes */
/*****************************************************************/


/*****************************************************************/
/*
 * This routine makes a request of the Protocol Distributor. The
 * request is to retrieve the graphics context state information
 * for a given source channel and xid. This receiver's port number
 * is also passed to indicate where to send the graphics state
 * information once it is obtained.
 */
XID
get_initial_gc(gid,ptr)
    register XID        gid;
    register XGCValues  *ptr;
{
    XID                 window;

#ifdef TRACE
fprintf(stderr,"PR:: GET_INITIAL GC - retrieve state information.\n");
#endif
```

```
/*
 * Set the get gc state information and wait
 */
    shmem->gc_channel   = channel;
    shmem->gc_id        = gid;
    shmem->gc_port      = shmem->pr_port;
    shmem->get_gc       = TRUE;

#ifdef TRACE
fprintf(stderr,"PR:: made request for gc state id:0x%x channel:%d\n",
gid,channel);
#endif

/*
 * Now send a signal to the Distributor to get him to
 * check shared memory.
 */
    kill(shmem->pd_pid,SIGUSR1);

#ifdef TRACE
fprintf(stderr,"PR:: sent signal to that task\n");
#endif

/*
 * Now wait on that  flag to turn FALSE
 */
    set_alarm(100);     /* 30 second alarm */
    while (shmem->get_gc) {
        if (shmem->sm_status==DIE)
            go_away();
    } /* end while */
    clear_alarm();

#ifdef TRACE
fprintf(stderr,"PR:: gc state request fulfilled...waiting for actual
data...\n");
#endif

/*
 * Now we need to read the file descriptor from the
 * Protocol Multiplexer, dispatching requests, until
 * the GCS information is sent to us.
 */
    wait_for_GCS(ptr,&window);

#ifdef TRACE
fprintf(stderr,"PR:: actual gc state data retrieved.\n");
fprintf(stderr,"PR:: window returned is 0x%x\n",window);
#endif
    return(window);

} /* end get_initial_gc */
```

```
/*********************************************************************/

/*********************************************************************/
/*
 * This routine retrieves the state information for a given XID,
 * to fulfill a request received from the Protocol Multiplexer.
 * This routine sets a flag in shared memory to indicate to the
 * Protocol Distributor to send a particular GC state to the
 * Protocol Multiplexer.
 */
void
get_gc_state(port)
    register short      *port;
{
    register XID            xid;

#ifdef TRACE
fprintf(stderr,"PR:: get_gc_state called for port:%d\n",*port);
#endif

/*
 * Read the XID for a given request
 */
    xid     = read_xid();

#ifdef TRACE
fprintf(stderr,"PR:: get_gc_state, xid:0x%x\n",xid);
#endif

/*
 * Signal the Protocol Distributor which graphics
 * context state information to send to the Protocol
 * Multiplexer.
 */
    if (!shmem->get_gc) {
        shmem->gc_channel   = 0;
        shmem->gc_id        = xid;
        shmem->gc_port      = *port;
#ifdef TRACE
fprintf(stderr,"PR:: Requesting send gc from somebody\n");
#endif
        shmem->send_gc      = TRUE;
/*
 * Send signal to PD to handle this
 */
        kill(shmem->pd_pid,SIGUSR1);
    }
else {
fprintf(stderr,"PR:: get_gc and send_gc CONFLICT!!!!\n");
go_away();
}
```

```
/*
 * Now hang around until the request is fulfilled. This
 * ensures that the Protocol Receiver will not receive
 * another request of this type and overwrite the shared
 * memory variables.
 */
    set_alarm(30);
    while (shmem->send_gc) {
        if (shmem->sm_status==DIE)
            go_away();
    }
    clear_alarm();

#ifdef TRACE
fprintf(stderr,"PR:: continuing on after PD fulfilled send request\n");
#endif

} /* end get_gc_state */
/*********************************************************************/

/*********************************************************************/
/*
 * This routine retrieves the state information (window) for a given
 * XID to fulfill a request received from the Protocol Multiplexer.
 */
void
get_window_state(port)
    register short      *port;
{
    register XID        window;
    register int        client;
    register int        chan;
    register int        i;
    Status              stat;

#ifdef TRACE
fprintf(stderr,"PR:: get_window_state called\n");
#endif

/*
 * Read the XID for a given request
 */
    window      = read_xid();
    client  = CLIENT_ID(window);

/*
 * We are going to read another 4 bytes here, these will be the channel
 * number. We will use the read_xid routine since it also reads 4 bytes.
 */
    chan        = (int)read_xid();
```

```
#ifdef TRACE
fprintf(stderr,"PR:: for window:0x%x  client:%d\n",window,client);
fprintf(stderr,"PR::   calling   XGetWindowAttributes   with
local_dpy:0x%x\n",local_dpy);
fprintf(stderr,"PR::    ....window:0x%x
&shmem->wats:0x%x\n",window,&shmem->wats);
fprintf(stderr,"PR:: ....port  :%d\n",*port);
fprintf(stderr,"PR:: ....chan  :%d\n",chan);
fprintf(stderr,"PR:: ....l_dpy :0x%x\n",local_dpy);
#endif


/*
 * Now retrieve, from the X server, information about this particular
 * window.
 */
#ifdef TRACE
fprintf(stderr,"PR:: just before getwindowattributes\n");
#endif
    stat = XGetWindowAttributes( local_dpy, window, &shmem->wats);
#ifdef TRACE
fprintf(stderr,"PR:: getwindowattributres is finished\n");
fprintf(stderr,"PR:: width:%d height:%d x:%d y:%d\n",
shmem->wats.width,shmem->wats.height,shmem->wats.x,shmem->wats.y);
#endif

/*
 * Note: the return values for some Xlib calls are not consistent, so in
 * this case we can't check against Success.
 */
    if (stat != 1) {
        fprintf(stderr,"PR::   get_window_state   (XGetWindowAttributes)
failed.\n");
#ifdef GO_AWAY
        go_away();
#endif
    }

#ifdef TRACE
fprintf(stderr,"PR:: just did an XGetWindowAttributes...\n");
fprintf( stderr,"win:0x%x\n",window );
#endif

/*
 * Search through shared memory for this window and place the background
 * pixel and parent values into shared memory to send.
 */
    for (i=0;i<MAX_WINS;i++) {
        if (shmem->win[i].window==window) {
            shmem->wat_bg_pixel = shmem->win[i].background;
            shmem->wat_parent   = shmem->win[i].parent;
```

```
#ifdef TRACE
fprintf(stderr,"get   parent:   0x%x,   win:   0x%x,   index:
%d\n",shmem->win[i].parent, window,i);
fprintf(stderr,"PR:: get_window_state> win: 0x%lx, index: %d, bg: 0x%lx
parent: 0x%lx\n",
shmem->win[i].window,i,shmem->wat_bg_pixel,shmem->wat_parent);
#endif
            break;
        }
    } /* end for */

/*
 * Signal the Protocol Distributor which window
 * attributes state information to send to the Protocol
 * Multiplexer.
 */
    if (!shmem->get_wat) {
#ifdef TRACE
fprintf(stderr,"PR:: just asked the PD to send the attributes on to
PM\n");
#endif
        shmem->wat_channel  = chan;
        shmem->wat_id       = window;
        shmem->wat_port     = *port;
        shmem->send_wat     = TRUE;
/*
 * Send signal to PD to handle this
 */
        kill(shmem->pd_pid,SIGUSR1);
    } /* end if !shmem->get_wat */

/*
 * Now hang around until the request is fulfilled. This
 * ensures that the Protocol Receiver will not receive
 * another request of this type and overwrite the shared
 * memory variables.
 */
#ifdef TRACE
fprintf(stderr,"PR:: requested the PD to send over the attributes and sent
signal\n");
fprintf(stderr,"PR:: entering a loop to wait on pd to finish\n");
#endif
    set_alarm(30);
    while (shmem->send_wat) {
        if (shmem->sm_status==DIE)
            go_away();
    }
    clear_alarm();

#ifdef TRACE
fprintf(stderr,"PR:: continuing on after PD fulfilled send request\n");
#endif
```

```
} /* end get_window_state */
/*****************************************************************/

/*****************************************************************/
/*
 * This routine makes an rpc call to register itself onto a
 * particular Distributor's index.
 */
void
register_self(index)
    int         index;
{
    struct RecvRegister RecvReg;
    struct PortID       PortID;
    int                 retval;

#ifdef TRACE
fprintf(stderr,"PR:: register_self> my_port is %d\n",my_port);
#endif

/*
 * Set up the values for the RPC call
 */
    strncpy(RecvReg.recvname,hostname,HOSTNAMLEN);
    RecvReg.distributor_id  = shmem->distributor_id;
    RecvReg.portnum         = (int)my_port;

#ifdef TRACE
fprintf(stderr,"PR:: Register Self host:<%s> index:%d\n",
RecvReg.recvname,RecvReg.distributor_id);
fprintf(stderr,"PR:: my port is :%d\n",my_port);
#endif

/*
 * Make the RPC call and get return values
 */
    retval = clnt_broadcast(CDM_PROG,CDM_VERS,CDM_REG_RECV,
                    xdr_RegRecv,&RecvReg,
                    xdr_PortID,&PortID,callme);

/*
 * Accept a connection on the port (from the Protocol Multiplexer)
 */
    accept_connection();

} /* end register self */
/*****************************************************************/


/*****************************************************************/
```

```
/*
 * This routine is called when the final shutdown of a client has
 * occurred.
 */
void
final_shutdown(client)
    int         client;
{

#ifdef TRACE
fprintf(stderr,"PR:: requested to close down a channel!!!!!\n");
#endif
    number_of_clients--;
    channel                 = shmem->pr_close_client;
    shmem->clients[channel] = 0;
#ifdef TRACE
fprintf(stderr,"PR::   that   brings   us   down   to   %d
clients\n",number_of_clients);
fprintf(stderr,"PR:: channel:%d\n",channel);
fprintf(stderr,"PR:: client:%d\n",client);
fprintf(stderr,"PR:: dest[client]:0x%x\n",dest[client]);
#endif
    XCloseDisplay(dest[client]);
#ifdef TRACE
fprintf(stderr,"PR:: XCloseDisplay complete\n");
#endif
    dest[client]            = NULL;
    free_client(client);
#ifdef TRACE
fprintf(stderr,"PR:: free_client complete\n");
#endif
    shmem->pr_close_channel = FALSE;
#ifdef TRACE
fprintf(stderr,"PR::   closed   down   that   sucker.
Channel:%d\n",shmem->pr_close_client);
#endif

}  /* end final_shutdown_client */
/*****************************************************************/


/*****************************************************************/
/*
 * This routine goes through the 'dirty' mask and updates the local
 * gc with all the changed values.
 */
void
update_gc(req,gc,client)
    register xChangeGCReq   *req;
    register GC             gc;
    register int            client;
{
```

```
    unsigned char           *bufptr;
    unsigned long           mask;

#ifdef TRACE
fprintf(stderr,"PR:: UPDATE_GC called for gid:0x%x client:0x%x\n",
gc->gid,client);
#endif

/*
 * Has this gc been changed any?
 */
    gc->dirty   = req->mask;
    if (gc->dirty==0)
{
fprintf(stderr,"....NOT DIRTY.\n");
        return;
}
    mask        = req->mask;
    bufptr      = (unsigned char *)req + sz_xChangeGCReq;

/*
 * Go through all the possible mask values and if true,
 * store the value into the XGCValues structure
 */

    if (mask&GCFunction) {
        gc->values.function =
                (int)*((int *)bufptr);
        bufptr+=sizeof(int);
    }

    if (mask&GCPlaneMask) {
        gc->values.plane_mask =
                (unsigned long)*((unsigned long *)bufptr);
        bufptr+=sizeof(unsigned long);
    }

    if (mask&GCForeground) {
        gc->values.foreground   =
                (unsigned long)*((unsigned long *)bufptr);
        bufptr+=sizeof(unsigned long);
    }

    if (mask&GCBackground) {
        gc->values.background   =
                (unsigned long)*((unsigned long *)bufptr);
        bufptr+=sizeof(unsigned long);
    }

    if (mask&GCLineWidth) {
        gc->values.line_width   =
                (int)*((int *)bufptr);
```

```
    bufptr+=sizeof(int);
}

if (mask&GCLineStyle) {
    gc->values.line_style   =
            (int)*((int *)bufptr);
    bufptr+=sizeof(int);
}

if (mask&GCCapStyle) {
    gc->values.cap_style    =
            (int)*((int *)bufptr);
    bufptr+=sizeof(int);
}

if (mask&GCJoinStyle) {
    gc->values.join_style   =
            (int)*((int *)bufptr);
    bufptr+=sizeof(int);
}

if (mask&GCFillStyle) {
    gc->values.fill_style   =
            (int)*((int *)bufptr);
    bufptr+=sizeof(int);
}

if (mask&GCFillRule) {
    gc->values.fill_rule    =
            (int)*((int *)bufptr);
    bufptr+=sizeof(int);
}

if (mask&GCTile) {
    gc->values.tile = 0;
    bufptr+=sizeof(XID);
}

if (mask&GCStipple) {
    gc->values.stipple  = 0;
    bufptr+=sizeof(XID);
}

if (mask&GCTileStipXOrigin) {
    gc->values.ts_x_origin  =
            (int)*((int *)bufptr);
    bufptr+=sizeof(int);
}

if (mask&GCTileStipYOrigin) {
    gc->values.ts_y_origin  =
            (int)*((int *)bufptr);
```

```
    bufptr+=sizeof(int);
}

if (mask&GCFont) {
    bufptr+=sizeof(XID);
}

if (mask&GCSubwindowMode) {
    gc->values.subwindow_mode   =
            (int)*((int *)bufptr);
    bufptr+=sizeof(int);
}

if (mask&GCGraphicsExposures) {
    gc->values.graphics_exposures   =
            (Bool)*((Bool *)bufptr);
    bufptr+=sizeof(Bool);
}

if (mask&GCClipXOrigin) {
    gc->values.clip_x_origin    =
            (int)*((int *)bufptr);
    bufptr+=sizeof(int);
}

if (mask&GCClipYOrigin) {
    gc->values.clip_y_origin    =
            (int)*((int *)bufptr);
    bufptr+=sizeof(int);
}

if (mask&GCClipMask) {
    gc->values.clip_mask    =
            (XID)*((XID *)bufptr);
    bufptr+=sizeof(XID);
}

if (mask&GCDashOffset) {
    gc->values.dash_offset  =
            (int)*((int *)bufptr);
    bufptr+=sizeof(int);
}

if (mask&GCDashList) {
    gc->values.dashes   =
            (char)*((char *)bufptr);
    bufptr+=sizeof(char);
}

FlushGC(dest[client],gc);
XFlush(dest[client]);
```

```
} /* end update_gc */
/*****************************************************************/
```

```
/*********************************************************
 *                                                       *
 *      For a listing of alarm.c, see Appendix L         *
 *                                                       *
 *********************************************************/
```

```
/*********************************************************
 *                                                       *
 *      For a listing of mutil.c, see Appendix L         *
 *                                                       *
 *********************************************************/
```

```
/*
 * File          : netread.c
 * Author        : P. Fitzgerald - SwRI
 * Date          : 11/30/89
 * This file performs a read on the network. The MASSCOMP version of read,
 * when applying to a network device, does not guarantee that the call
 * will complete with the number of bytes requested, even if BLOCKING is
set.
 * This routine can be used interchangeably with read() and will not
return
 * without 1 of following conditions:
 *              1 - number of bytes requested was read
 *              2 - timeout (30 seconds currently)
 *              3 - error return from OS
 */
#include <stdio.h>
#include <sys/types.h>
#include <fcntl.h>
#include <signal.h>
#include <errno.h>

/* EXTERNAL ROUTINES */
extern      set_alarm();
extern      clear_alarm();

/*****************************************************************/
/*
 * This routine reads from the network a certain number of bytes. The
 * Masscomp implementation of a socket means that if all the bytes are
 * not ready, then the read() call will return with an EWOULDBLOCK error,
 * EVEN though the file descriptor has BLOCKING set. This routine should
 * appear to the caller identical to that of the standard read() routine.
 */
int
netread(fd,ptr,len)
    register int            fd;
    register unsigned char  *ptr;
    register int            len;
{
    register int            bytes_read;
    register int            total;

/*
 * Turn on an alarm in case we get stuck
 */
    set_alarm(500);

/*
 * Now enter a loop to read until all the bytes are read
 */
    bytes_read = 0;
    total      = 0;
```

```
    while (total < len) {
        errno = 0;
        bytes_read = read(fd,ptr+total,len-total);
/*
 * If we get interrupted from a system call, re-issue the read.
 */
        while (errno==EINTR) {
            fprintf(stderr,"Network read interrupted by signal. Read %d
bytes. Re-issued.\n",
                bytes_read);
            errno = 0;
            bytes_read = read(fd,ptr+total,len-total);
        }
        if (bytes_read<0)  {
            clear_alarm();
            return(-1);
        }
        total += bytes_read;
    } /* end while */
    return(total);

} /* end netread */
/*****************************************************************/
```

APPENDIX N
LOCAL DISTRIBUTION MANAGER LISTINGS

```
/*********************************************************************
The included program listings are prototypes, no warranty is expressed or
implied for their use in any other fashion.  They should not be considered
or used as production software.  The information in the listings is
supplied on an "as is" basis.  No responsibility is assumed for damages
resulting from the use of any information contained in the listings.

The software in these listings has been compiled on Masscomp 6350's and
6600's and on Sun 3's and 4's.  Modifications may be necessary for use on
other systems.
*********************************************************************/

/*
 * File         : ldm.c
 * Author       : P. Fitzgerald - SwRI
 * Date         : 10/3/89
 * Description  : This file contatins all the Local Distribution Manager
 *                code.
 */

/*
 * GUI using motif/XtIntrinsics added by
 * Author           : Stephen Johns
 * Date             : 1-18-90
 * Last Modified    : 5-10-90
 *
 * use tab stops of size 4
 *
 */

#include <ctype.h>
#include <stdio.h>
#include <utmp.h>
#include <sys/types.h>
#include <signal.h>
#include <rpcsvc/rusers.h>
#include <X11/X.h>

#define NEED_REPLIES
#define NEED_EVENTS

#include <X11/Xproto.h>
#include <X11/Xlib.h>
#include "../includes/ds_manage.h"
#include "../includes/smtypes.h"
#include "../includes/smdef.h"
#include "../includes/dist.h"
#include "../includes/rpc.h"
```

```
#include <X11/Intrinsic.h>
#include <X11/Shell.h>
#include <Xm/BulletinB.h>
#include <Xm/CascadeB.h>
#include <Xm/LabelG.h>
#include <Xm/MainW.h>
#include <Xm/MessageB.h>
#include <Xm/PushB.h>
#include <Xm/PushBG.h>
#include <Xm/RowColumn.h>
#include <Xm/SelectioB.h>
#include <Xm/SeparatoG.h>
#include <Xm/Text.h>
#include <Xm/Xm.h>

/* WIDGET VARIABLES */

Widget  bboard;
Widget  button[7];
Widget  channel_widget[10];
Widget  finigadget;
Widget  gadgetshell;
Widget  gadget_row_col;
Widget  gadget_row_col_2;
Widget  gadget_window;
Widget  help_btn;
Widget  info_label;
Widget  label;
Widget  labelshell;
Widget  label_row_col;
Widget  label_row_col_2;
Widget  label_widget[10];
Widget  label_window;
Widget  main_cc[3];
Widget  main_pd;
Widget  main_window;
Widget  menu_bar;
Widget  menu_row_col;
Widget  menushell;
Widget  title_label;

Window  *child;

Arg     args[10];
int     n;
char    tmp_text[40];
char    current_recv[10];
int     recv_flag;
int     crecv_flag;
int     cdist_flag;
```

```
/* GLOBAL MESSGAES */

static char gadgstring[4][35]  = {
                                 "Please Make Selection",
                                 "Pick the Window to Distribute",
                                 "LDM Cannot Be Distributed",
                                 "Root Cannot Be Distributed"};
static char button_name[7][35] = {
                                 "Retrieve TV Guide",
                                 "Distribution Authorization Request",
                                 "Reception Authorization Request",
                                 "Cancel Distribution on Channel",
                                 "Cancel Reception on Channel",
                                 "Stop Central Distribution Manager",
                                 "Quit"};
static char client_data[3][7]  = {
                                 "recvCB",
                                 "cdisCB",
                                 "crcvCB"};

/* FORWARD REFERENCES */

void    make_gadget_widgets();
void    make_label_widgets();
void    make_all_main_widgets();

/* CALLBACKS */

void    cdisCB();
void    crcvCB();
void    distCB();
void    gadgCB();
void    helpCB();
void    infoCB();
void    nameCB();
void    okayCB();
void    qgdtCB();
void    qlblCB();
void    quitCB();
void    recvCB();
void    scdmCB();
void    sendCB();
void    tv_gCB();

/* GLOBAL ROUTINES */

int             callme();
int             contact_cdm();
void            attach_shared_memory();
void            dist_auth_request();
void            go_away();
void            recv_auth_request();
```

```
void            remove_channel();
void            retrieve_tvguide();
static void     semcall();

/* EXTERNAL ROUTINES */

extern  xdr_ChanID();
extern  xdr_ChanReq();
extern  xdr_DistAuth();
extern  xdr_PortID();
extern  xdr_RecvAuth();
extern  xdr_RemvRecv();
extern  xdr_tvguide();

/* GLOBAL VARIABLES */

char                    *btn_text;
char                    hostname[HOSTNAMLEN];
char                    management_host[HOSTNAMLEN];
char                    tv_guide[MAX_CHANNELS][CHANNAMLEN];
Display                 *display;
struct DistAuth         DistAuth;
struct RecvAuth         RecvAuth;
struct MC_SHMEMORY      *shmem;
unsigned short          port_number = 0;

/*****************************************************************/
/*
 * Main program body
 */
main ()
{

/*
 * Set up to catch kill signals
 */
    signal(SIGQUIT, go_away);

/*
 * Retrieve the hostname (Internet).
 */
    if ( gethostname(hostname,sizeof(hostname)) <0) {
        perror("LDM::gethostname:");
        go_away(-1);
    }

/*
 * Now try to contact a Central Distribution Manager, somewhere on
 * the network.
 */
    if (!contact_cdm(management_host)) {
        fprintf(stderr,"LDM::Cannot    contact    Central    Distribution
```

```
Manager.\n");
        go_away(-1);
    }

#ifdef TRACE
    printf("SwRI Local Distribution Manager <%s> starting...\n",hostname);
    printf("...Found   Central   Distribution   Manager   on
<%s>.\n",management_host);
#endif

/*
 * Attach self to shared memory area.
 */

    attach_shared_memory();

/*
 * Setup XtIntrinsics
 */

    XtToolkitInitialize();
    display=XtOpenDisplay(NULL,NULL,"SwRI LDM","Menu",NULL,0,NULL,NULL);

/*
 * Create an application shell for the main menu widgets
 */

    n = 0;
    XtSetArg (args[n],XmNwidth,225); n++;
    XtSetArg (args[n],XmNheight,200); n++;
    XtSetArg(args[n],XmCAllowShellResize,True); n++;

    menushell = XtAppCreateShell(
                        NULL,           /* parent                */
                        "Menu",         /* widget name           */
                        applicationShellWidgetClass, /* widget class*/
                        display,        /* display name          */
                        NULL,           /* argument list         */
                        0);             /* number of arguments   */

/*
 * Create an application shell for the tv guide widgets
 */

    n = 0;
    XtSetArg (args[n],XmNwidth,225); n++;
    XtSetArg (args[n],XmNheight,200); n++;
    XtSetArg(args[n],XmCAllowShellResize,True); n++;

    labelshell  = XtAppCreateShell(
                        NULL,           /* parent                */
                        "TVGuide",      /* widget name           */
```

```
                        applicationShellWidgetClass, /* widget class*/
                        display,        /* display name         */
                        NULL,           /* argument list        */
                        0);             /* number of arguments  */

/*
 * Create an application shell for the selection widgets
 */

    n = 0;
    XtSetArg (args[n],XmNwidth,225); n++;
    XtSetArg (args[n],XmNheight,200); n++;
    XtSetArg(args[n],XmCAllowShellResize,True); n++;

    gadgetshell = XtAppCreateShell(
                        NULL,           /* parent               */
                        "Select",       /* widget name          */
                        applicationShellWidgetClass, /* widget class*/
                        display,        /* display name         */
                        NULL,           /* argument list        */
                        0);             /* number of arguments  */

/*
 * clear out array used to mark channels that are currently receiving
protocol
 */

    for(n = 0 ; n < 10 ; current_recv[n++] = ' ');

/*
 * setup flags to track state that pushbuttons should work under
 */
    recv_flag  = 0;
    crecv_flag = 0;
    cdist_flag = 0;

/*
 * Create the main menu widgets
 */

    make_all_main_widgets();

    XtRealizeWidget(menushell);

    XtMainLoop();

} /* End of Program */


/*****************************************************************
 * Set up menu widgets
 */
```

```
void
make_all_main_widgets()
{
int     i;
char    *ptr;

/*
 * Set up the Main Menu Widget
 */

    n = 0;
    XtSetArg (args[n],XmNwidth,225); n++;
    XtSetArg (args[n],XmNheight,200); n++;

    main_window = XmCreateMainWindow(
                            menushell,      /* parent               */
                            "Menu",         /* widget name          */
                            args,           /* argument list        */
                            n);             /* number of arguments  */

    XtManageChild(main_window);

/*
 * Create the menu bar with main pulldown and help
 */

    n = 0;
    XtSetArg(args[n],XmNlabelString,
                XmStringCreate("Tester",XmSTRING_DEFAULT_CHARSET)); n++;

    menu_bar = XmCreateMenuBar(
                            main_window,    /* parent
*/
                            "menu bar",     /* widget name
*/
                            NULL,           /* argument list        */
                            0);              /* number of arguments
*/

    XtManageChild(menu_bar);

    /* MAIN PULLDOWN */

    main_pd = XmCreatePulldownMenu(
                            menu_bar,       /* parent
*/
                            "main_pd",      /* widget name
*/
                            NULL,           /* argument list        */
                            0);              /* number of arguments
*/
```

```
    n = 0;
    XtSetArg(args[n], XmNsubMenuId, main_pd); n++;

    main_cc[0] = XmCreateCascadeButton(
                            menu_bar,       /* parent
*/
                            "Main",         /* widget name
*/
                            args,           /* argument list
*/
                            n);             /* number of arguments
*/

    XtManageChild(main_cc[0]);

    n = 0;
    XtSetArg(args[n],XmNlabelString,
                XmStringCreate("Info",XmSTRING_DEFAULT_CHARSET)); n++;

    main_cc[1] = XmCreatePushButtonGadget(
                            main_pd,        /* parent
*/
                            "Info",         /* widget name
*/
                            args,           /* argument list
*/
                            n);             /* number of arguments
*/

    XtManageChild(main_cc[1]);

    n = 0;
    XtSetArg(args[n],XmNlabelString,
                XmStringCreate("Quit",XmSTRING_DEFAULT_CHARSET)); n++;

    main_cc[2] = XmCreatePushButtonGadget(
                            main_pd,        /* parent
*/
                            "Quit",         /* widget name
*/
                            args,           /* argument list
*/
                            n);             /* number of arguments
*/

    XtManageChild(main_cc[2]);

/*
 * set up the help button
 */

    n = 0;
```

```
    XtSetArg(args[n],XmNlabelString,
                XmStringCreate("Help",XmSTRING_DEFAULT_CHARSET)); n++;

    help_btn = XmCreateCascadeButton(
                            menu_bar,       /* parent
*/
                            "Help",         /* widget name
*/
                            args,           /* argument list
*/
                            n);             /* number of arguments
*/

/*
 * change the help_button into a 'menuHelpWidget'
 */

    n = 0;
    XtSetArg(args[n], XmNmenuHelpWidget, (XtArgVal)help_btn); n++;
    XtSetValues(
                menu_bar,                   /* parent
*/
                args,                       /* argument list
*/
                n);                         /* number of arguments
*/

    XtManageChild(help_btn);

/*
 * add callbacks for help, info and quit button in the menu bar
 */

    XtAddCallback(main_cc[1],XmNactivateCallback,infoCB,NULL);
    XtAddCallback(help_btn,XmNactivateCallback,helpCB,NULL);
    XtAddCallback(main_cc[2],XmNactivateCallback,quitCB,NULL);

/*
 * create the main menu widget
 */

    n = 0;
    XtSetArg (args[n],XmNwidth,225); n++;
    XtSetArg (args[n],XmNheight,200); n++;
    XtSetArg(args[n],XmNpacking,XmPACK_COLUMN); n++;
    XtSetArg(args[n],XmNnumColumns,1); n++;

    menu_row_col = XmCreateRowColumn(
                            main_window,    /* parent               */
                            "rc",           /* widget name          */
                            args,           /* argument list        */
                            n);             /* number of arguments  */
```

```
    XtManageChild(menu_row_col);

    btn_text    =    XmStringCreateLtoR(gadgstring[0],
XmSTRING_DEFAULT_CHARSET);

    n = 0;
    XtSetArg (args[n],XmNlabelString,btn_text); n++;

    info_label = XmCreateLabelGadget(
                          menu_row_col,   /* parent               */
                          "info",         /* widget name          */
                          args,           /* argument list        */
                          n);             /* number of arguments  */

    XtManageChild(info_label);

/*
 * Create main push buttons on the main menu widget
 */

    for(i = 0 ; i < 7 ; i++){
        b     t     n          _       t     e     x     t                -
XmStringCreateLtoR(button_name[i],XmSTRING_DEFAULT_CHARSET);

        n = 0;
        XtSetArg (args[n],XmNlabelType,XmSTRING); n++;
        XtSetArg (args[n],XmNlabelString,btn_text); n++;
        XtSetArg (args[n],XmNwidth,250); n++;
        XtSetArg (args[n],XmNheight,150); n++;

        button[i] = XtCreateManagedWidget(
                          "button",       /* widget name          */
                          xmPushButtonWidgetClass, /* widget class*/
                          menu_row_col,   /* parent               */
                          args,           /* argument list        */
                          n);             /* number of arguments  */
    }

/*
 * add main callbacks for main menu widgets
 */

    XtAddCallback(button[0],XmNactivateCallback,tv_gCB,NULL);
    XtAddCallback(button[1],XmNactivateCallback,sendCB,NULL);
    XtAddCallback(button[2],XmNactivateCallback,recvCB,NULL);
    XtAddCallback(button[3],XmNactivateCallback,cdisCB,NULL);
    XtAddCallback(button[4],XmNactivateCallback,crcvCB,NULL);
    XtAddCallback(button[5],XmNactivateCallback,scdmCB,NULL);
    XtAddCallback(button[6],XmNactivateCallback,quitCB,NULL);

/*
```

```
* Set up the Gadget Widget for selections
*/

    n = 0;
    gadget_window = XmCreateMainWindow(
                            gadgetshell,    /* parent                */
                            "gadget",       /* widget name           */
                            args,           /* argument list         */
                            n);             /* number of arguments   */

    XtManageChild(gadget_window);

    n = 0;
    bboard = XmCreateBulletinBoard(
                            gadget_window,  /* parent                */
                            "gadget",       /* widget name           */
                            args,           /* argument list         */
                            n);             /* number of arguments   */

    XtManageChild (bboard);

    n = 0;
    XtSetArg(args[n],XmNnumColumns,1); n++;
    XtSetArg(args[n],XmNadjustLast,False); n++;
    XtSetArg(args[n],XmNresizeWidth,True); n++;

    gadget_row_col = XmCreateRowColumn(
                            bboard,         /* parent                */
                            "lrc",          /* widget name           */
                            args,           /* argument list         */
                            n);             /* number of arguments   */

    XtManageChild(gadget_row_col);

    sprintf(tmp_text,"Select a Channel.");
    btn_text = XmStringCreateLtoR(tmp_text,XmSTRING_DEFAULT_CHARSET);

    n = 0;
    XtSetArg (args[n],XmNlabelString,btn_text); n++;

    title_label = XmCreateLabelGadget(
                            gadget_row_col, /* parent                */
                            "info",         /* widget name           */
                            args,           /* argument list         */
                            n);             /* number of arguments   */

    XtManageChild(title_label);

    sprintf(tmp_text,"(R) Currently Receving Channel.");
    btn_text = XmStringCreateLtoR(tmp_text,XmSTRING_DEFAULT_CHARSET);

    n = 0;
```

```
    XtSetArg (args[n],XmNlabelString,btn_text); n++;

    label = XmCreateLabelGadget(
                            gadget_row_col, /* parent                */
                            "info",         /* widget name           */
                            args,           /* argument list         */
                            n);             /* number of arguments   */

    XtManageChild(label);

    sprintf(tmp_text,"(D) Currently Distributing Channel.");
    btn_text = XmStringCreateLtoR(tmp_text,XmSTRING_DEFAULT_CHARSET);

    n = 0;
    XtSetArg (args[n],XmNlabelString,btn_text); n++;

    label = XmCreateLabelGadget(
                            gadget_row_col, /* parent                */
                            "info",         /* widget name           */
                            args,           /* argument list         */
                            n);             /* number of arguments   */

    XtManageChild(label);

    n = 0;
    label = XmCreateSeparatorGadget(gadget_row_col,"line",args,n);
    XtManageChild(label);

    n = 0;
    XtSetArg(args[n],XmNadjustLast,False); n++;
    XtSetArg(args[n],XmNorientation,XmVERTICAL); n++;
    XtSetArg(args[n],XmNresizeWidth,True); n++;
    XtSetArg(args[n],XmNnumColumns,2); n++;
    XtSetArg(args[n],XmNpacking,XmPACK_COLUMN); n++;

    gadget_row_col_2 = XmCreateRowColumn(
                            gadget_row_col, /* parent                */
                            "grc",          /* widget name           */
                            args,           /* argument list         */
                            n);             /* number of arguments   */

    XtManageChild(gadget_row_col_2);

/*
 * Set up the Label Widget for the tv guide
 */

    n = 0;
    label_window = XmCreateMainWindow(
                            labelshell,     /* parent                */
                            "label",        /* widget name           */
                            args,           /* argument list         */
```

```
                        n);             /* number of arguments  */

XtManageChild(label_window);

n = 0;
bboard = XmCreateBulletinBoard(
                        label_window,   /* parent               */
                        "label",        /* widget name          */
                        args,           /* argument list        */
                        n);             /* number of arguments  */

XtManageChild (bboard);

n = 0;
XtSetArg(args[n],XmNnumColumns,1); n++;
XtSetArg(args[n],XmNadjustLast,False); n++;
XtSetArg(args[n],XmNresizeWidth,True); n++;

label_row_col = XmCreateRowColumn(
                        bboard,         /* parent               */
                        "lrc",          /* widget name          */
                        args,           /* argument list        */
                        n);             /* number of arguments  */

XtManageChild(label_row_col);

sprintf(tmp_text,"Channel Listing.");
btn_text = XmStringCreateLtoR(tmp_text,XmSTRING_DEFAULT_CHARSET);

n = 0;
XtSetArg (args[n],XmNlabelString,btn_text); n++;

label = XmCreateLabelGadget(
                        label_row_col,  /* parent               */
                        "info",         /* widget name          */
                        args,           /* argument list        */
                        n);             /* number of arguments  */

XtManageChild(label);

sprintf(tmp_text,"(R) Currently Receiving Channel.");
btn_text = XmStringCreateLtoR(tmp_text,XmSTRING_DEFAULT_CHARSET);

n = 0;
XtSetArg (args[n],XmNlabelString,btn_text); n++;

label = XmCreateLabelGadget(
                        label_row_col,  /* parent               */
                        "info",         /* widget name          */
                        args,           /* argument list        */
                        n);             /* number of arguments  */
```

```
    XtManageChild(label);

    sprintf(tmp_text,"(D) Currently Distributing Channel.");
    btn_text = XmStringCreateLtoR(tmp_text,XmSTRING_DEFAULT_CHARSET);

    n = 0;
    XtSetArg (args[n],XmNlabelString,btn_text); n++;

    label = XmCreateLabelGadget(
                          label_row_col,  /* parent                */
                          "info",         /* widget name           */
                          args,           /* argument list         */
                          n);             /* number of arguments   */

    XtManageChild(label);

    n = 0;
    label = XmCreateSeparatorGadget(
                          label_row_col,  /* parent                */
                          "line",         /* widget name           */
                          args,           /* argument list         */
                          n);             /* number of arguments   */

    XtManageChild(label);

    n = 0;
    XtSetArg(args[n],XmNadjustLast,False); n++;
    XtSetArg(args[n],XmNorientation,XmVERTICAL); n++;
    XtSetArg(args[n],XmNresizeWidth,True); n++;
    XtSetArg(args[n],XmNnumColumns,2); n++;
    XtSetArg(args[n],XmNpacking,XmPACK_COLUMN); n++;

    label_row_col_2 = XmCreateRowColumn(
                          label_row_col,  /* parent                */
                          "lrc",          /* widget name           */
                          args,           /* argument list         */
                          n);             /* number of arguments   */

    XtManageChild(label_row_col_2);

/* set up labels and PUSHBUTTONS for channels */

    ptr = &tv_guide[0][0];
    for (i=0;i<10;i++) {
        if (current_recv[i] == 'R')
            sprintf(tmp_text,"(R) CH %d : %s",i,ptr);
        else if (current_recv[i] == 'D')
            sprintf(tmp_text,"(D) CH %d : %s",i,ptr);
        else
            sprintf(tmp_text,"    CH %d : %s",i,ptr);

        btn_text = XmStringCreateLtoR(tmp_text,XmSTRING_DEFAULT_CHARSET);
```

```
    n = 0;
    XtSetArg (args[n],XmNlabelString,btn_text); n++;

    label_widget[i] = XmCreateLabelGadget(
                        label_row_col_2,/* parent                */
                        "info",         /* widget name           */
                        args,           /* argument list         */
                        n);             /* number of arguments   */

    XtManageChild(label_widget[i]);

    btn_text = XmStringCreateLtoR(tmp_text,XmSTRING_DEFAULT_CHARSET);

    n = 0;
    XtSetArg (args[n],XmNlabelString,btn_text); n++;
    XtSetArg (args[n],XmNrecomputeSize,True); n++;

    channel_widget[i] = XtCreateManagedWidget(
                        "button",       /* widget name           */
                        xmPushButtonWidgetClass, /* widget class*/
                        gadget_row_col_2, /* parent              */
                        args,           /* argument list         */
                        n);             /* number of arguments   */

    XtAddCallback(channel_widget[i],XmNactivateCallback,gadgCB,NULL);
    XtManageChild(channel_widget[i]);
    ptr+=CHANNAMLEN;
}

btn_text = XmStringCreateLtoR("Finished", XmSTRING_DEFAULT_CHARSET);

n = 0;
XtSetArg (args[n],XmNlabelType,XmSTRING); n++;
XtSetArg (args[n],XmNlabelString,btn_text); n++;

finigadget = XtCreateManagedWidget(
                        "fini",         /* widget name           */
                        xmPushButtonWidgetClass, /* widget class*/
                        gadget_row_col, /* parent                */
                        args,           /* argument list         */
                        n);             /* number of arguments   */

XtAddCallback(finigadget,XmNactivateCallback,qgdtCB,NULL);

btn_text = XmStringCreateLtoR("Finished", XmSTRING_DEFAULT_CHARSET);

n = 0;
XtSetArg (args[n],XmNlabelType,XmSTRING); n++;
XtSetArg (args[n],XmNlabelString,btn_text); n++;

finigadget = XtCreateManagedWidget(
```

```
                                "fini",         /* widget name          */
                                xmPushButtonWidgetClass, /* widget class*/
                                label_row_col,  /* parent               */
                                args,           /* argument list        */
                                n);             /* number of arguments  */

    XtAddCallback(finigadget,XmNactivateCallback,qlblCB,NULL);

}  /* End make_all_main_widgets */

/*****************************************************************/
/* Set up label widgets
 */
void
make_label_widgets()
{

    retrieve_tvguide(management_host,tv_guide,"tv_gCB");

}  /* End make_label_widgets */

/*****************************************************************/
/* Set up gadget widgets
 */
void
make_gadget_widgets(callback)
char    *callback;
{
        retrieve_tvguide(management_host,tv_guide,callback);

}  /* End make_gadget_widgets */

/*****************************************************************/
void qlblCB()
{
    XtUnmapWidget(labelshell);
    XtMapWidget(menushell);
}

/*****************************************************************/
void qgdtCB()
{
    XtUnmapWidget(gadgetshell);
    XtMapWidget(menushell);
}

/*****************************************************************/
void okayCB()
{
    Widget  btn;
    Widget  prompt;
    btn_text    =   XmStringCreateLtoR("Enter    an    ID
```

```
Name",XmSTRING_DEFAULT_CHARSET);

    n = 0;
    XtSetArg(args[n],XmNselectionLabelString,btn_text); n++;

    prompt = XmCreatePromptDialog(
                            menushell,      /* parent                */
                            "name request", /* widget name           */
                            args,           /* argument list         */
                            n);             /* number of arguments   */

    btn = XmSelectionBoxGetChild(prompt,XmDIALOG_HELP_BUTTON);
    XtUnmanageChild(btn);

    btn = XmSelectionBoxGetChild(prompt,XmDIALOG_CANCEL_BUTTON);
    XtUnmanageChild(btn);

    XtAddCallback(prompt,XmNokCallback,nameCB,NULL);

    XtManageChild(prompt);
}

/*********************************************************************/
void nameCB(w,client_data,call_data)
Widget  w;
caddr_t client_data;
caddr_t call_data;
{
    XmSelectionBoxCallbackStruct *boxdata;

    boxdata = (XmSelectionBoxCallbackStruct *) call_data;
    XmStringGetLtoR(boxdata->value,XmSTRING_DEFAULT_CHARSET,&btn_text);
    dist_auth_request(management_host,&DistAuth,child,btn_text);
}

/*********************************************************************/
void tv_gCB()
{
    make_label_widgets();
    if (!XtIsRealized(labelshell))
        XtRealizeWidget(labelshell);
    else
        XtMapWidget(labelshell);
}

/*********************************************************************/
void recvCB()
{
    recv_flag  = 1;
    crecv_flag = 0;
    cdist_flag = 0;
```

```
    n = 0;
    btn_text = XmStringCreateLtoR("Select Channel to Receive",
                                  XmSTRING_DEFAULT_CHARSET);
    XtSetArg (args[n],XmNlabelString,btn_text); n++;
    XtSetValues(title_label,args,n);

    make_gadget_widgets("recvCB");
    if (!XtIsRealized(gadgetshell))
        XtRealizeWidget(gadgetshell);
    else
        XtMapWidget(gadgetshell);

}

/*****************************************************************/
void cdisCB()
{
    recv_flag  = 0;
    crecv_flag = 0;
    cdist_flag = 1;

    n = 0;
    btn_text = XmStringCreateLtoR("Select Channel to Cancel",
                                  XmSTRING_DEFAULT_CHARSET);
    XtSetArg (args[n],XmNlabelString,btn_text); n++;
    XtSetValues(title_label,args,n);

    make_gadget_widgets("cdisCB");
    if (!XtIsRealized(gadgetshell))
        XtRealizeWidget(gadgetshell);
    else
        XtMapWidget(gadgetshell);
}

/*****************************************************************/
void crcvCB()
{
    recv_flag  = 0;
    crecv_flag = 1;
    cdist_flag = 0;

    n = 0;
    btn_text = XmStringCreateLtoR("Select Channel to Cancel",
                                  XmSTRING_DEFAULT_CHARSET);
    XtSetArg (args[n],XmNlabelString,btn_text); n++;
    XtSetValues(title_label,args,n);

    make_gadget_widgets("crcvCB");
    if (!XtIsRealized(gadgetshell))
        XtRealizeWidget(gadgetshell);
    else
        XtMapWidget(gadgetshell);
```

```
}

/*********************************************************************/
void scdmCB() .
{
    int     i;

    stop_cdm(management_host);
    for (i=0;i<5;i++) {
        shmem->sm_status = DIE;
        sleep(1);
    }
}

/*********************************************************************/
void sendCB()
{

    char            dia_text[20];
    int             n;
    int             *root_x,*root_y;
    int             screen;
    int             *win_x,*win_y;
    unsigned int    *keys_buttons;
    Widget          box;
    Widget          diabox;
    Window          *root;
    XEvent          Xtreport;

    /* FORWARD REFERENCES */

    void            findname();

    screen = DefaultScreen(display);
    XGrabPointer(
                display,                    /* display              */
                RootWindow(display,screen), /* grab window          */
                False,                      /* do not pass events   */
                ButtonPressMask,            /* event mask for grab  */
                GrabModeSync,               /* pointer mode is SYNC */
                GrabModeAsync,              /* keyboard mode is ASYNC*/
                None,                       /* do not confine pointer*/
                None,                       /* no special cursor    */
                CurrentTime);               /* time the grab took place*

    XAllowEvents(display, SyncPointer, CurrentTime);

    n = 0;
    btn_text = XmStringCreateLtoR(gadgstring[1],XmSTRING_DEFAULT_CHARSET);
    XtSetArg (args[n],XmNlabelString,btn_text); n++;
    XtSetValues(info_label,args,n);
```

```
XFlush(display);

while(Xtreport.type != ButtonPress){
    XtNextEvent(&Xtreport);
    XtDispatchEvent(&Xtreport);
}

XQueryPointer(
            display,                        /* display              */
            RootWindow(display,screen),     /* root window          */
            &root,                          /* return root window id*/
            &child,                         /* return child window id*/
            &root_x,                        /* x coord based on root*/
            &root_y,                        /* y coord based on root*/
            &win_x,                         /* x coord based on child*/
            &win_y,                         /* y coord based on child*/
            &keys_buttons);                 /* state of modifier keys*/

if (child != 0) findname(child,tmp_text);

if ((child == 0)  | (strncmp(tmp_text,"SwRI LDM",8)) == 0) {
    n = 0;
    if (child == 0)
       btn_text = XmStringCreateLtoR(gadgstring[3],
                                     XmSTRING_DEFAULT_CHARSET);
    else
       btn_text = XmStringCreateLtoR(gadgstring[2],
                                     XmSTRING_DEFAULT_CHARSET);

    XtSetArg(args[n],XmNmessageString,btn_text); n++;

    box = XmCreateErrorDialog(
                        menushell,      /* parent               */
                        "SwRI LDM",     /* widget name          */
                        args,           /* argument list        */
                        n);             /* number of arguments  */

    XtManageChild(box);

    diabox = XmMessageBoxGetChild(box, XmDIALOG_HELP_BUTTON);
    XtUnmanageChild(diabox);

    diabox = XmMessageBoxGetChild(box, XmDIALOG_CANCEL_BUTTON);
    XtUnmanageChild(diabox);
}
else {
    sprintf(dia_text,"Distribute %s?",tmp_text);

    n = 0;
    btn_text = XmStringCreateLtoR(dia_text,XmSTRING_DEFAULT_CHARSET);
    XtSetArg(args[n],XmNmessageString,btn_text); n++;
```

```
        box = XmCreateMessageDialog(
                              menushell,      /* parent                */
                              "SwRI LDM",     /* widget name           */
                              args,           /* argument list         */
                              n);             /* number of arguments   */

        XtManageChild(box);

        diabox = XmMessageBoxGetChild(box, XmDIALOG_HELP_BUTTON);
        XtUnmanageChild(diabox);

        diabox = XmMessageBoxGetChild(box, XmDIALOG_OK_BUTTON);
        XtAddCallback(diabox,XmNactivateCallback,okayCB,NULL);

    }
    XtRealizeWidget(menushell);
    XtRealizeWidget(box);

    XUngrabPointer(
                              display,        /* display               */
                              CurrentTime);   /* time the grab took place*

    n = 0;
    btn_text = XmStringCreateLtoR(gadgstring[0],XmSTRING_DEFAULT_CHARSET);
    XtSetArg (args[n],XmNlabelString,btn_text); n++;
    XtSetValues(info_label,args,n);
}

/*****************************************************************/
void gadgCB(w,client,call)
Widget  w;
caddr_t client;
caddr_t call;
{

    char                *name;
    char                trash[3];
    char                moretrash[20];
    int                 channel;
    unsigned short      port;
    XmString            name_string;


    /*** determine name ***/

    n = 0;
    XtSetArg (args[n], XmNlabelString, &name_string); n++;
    XtGetValues (w, args, n);
    XmStringGetLtoR (name_string, XmSTRING_DEFAULT_CHARSET,&name);

    if (name != NULL) {
```

```
        sscanf(name,"%c%c%c%s  %d",&trash[0],&trash[1],&trash[2],moretrash,
                &channel);

/*
 * handle a receive
 */

        if (recv_flag) {

            current_recv[channel] = 'R';
            name[0] = '(';
            name[1] = 'R';
            name[2] = ')';

            btn_text = XmStringCreateLtoR(name,XmSTRING_DEFAULT_CHARSET);

            n = 0;
            XtSetArg (args[n],XmNlabelString,btn_text); n++;
            XtSetValues(channel_widget[channel],args,n);

            XFlush(display);

            recv_auth_request(management_host,&RecvAuth,channel);
        }

/*
 * handle a cancel distribution
 */

        if (cdist_flag) {

            current_recv[channel] = ' ';
            sprintf(name,"    CH %d : ",channel);

            btn_text = XmStringCreateLtoR(name,XmSTRING_DEFAULT_CHARSET);

            n = 0;
            XtSetArg (args[n],XmNlabelString,btn_text); n++;
            XtSetValues(channel_widget[channel],args,n);

            XFlush(display);

            remove_channel(management_host,channel);
        }

/*
 * handle a cancel reception
 */

        if (crecv_flag) {

            current_recv[channel] = ' ';
```

```
            name[0] = ' ';
            name[1] = ' ';
            name[2] = ' ';

            btn_text = XmStringCreateLtoR(name,XmSTRING_DEFAULT_CHARSET);

            n = 0;
            XtSetArg (args[n],XmNlabelString,btn_text); n++;
            XtSetValues(channel_widget[channel],args,n);

            XFlush(display);
            port    = shmem->pr_port;

            remove_receiver(management_host,channel,port);
        }
        XtMapWidget(gadgetshell);
    }
}

/*********************************************************************/
void quitCB()
{
        XtFree(btn_text);
        go_away(0);
}

/*********************************************************************/
void findname(win,savename)
Window  win;
char    savename[];
{

    char            *name = NULL;
    int             n;
    unsigned int    nchildren;
    Window          start,parent;
    Window          *children = NULL;

    XFetchName(display,win,&name);
    strcpy(savename,name);

    if (name != NULL)
        return;
    else {
        if(!XQueryTree(display,win,&start,&parent,&children,&nchildren))
            return;

        for (n = 0; n < nchildren; n++) findname(children[n],savename);
    }
}

/*********************************************************************/
```

```
/*
 * This routine makes an RPC call to the Central Distribution
 * Manager to retrieve the current TV guide listing.
 */
void
retrieve_tvguide(remote_host,guide,type)
char            *remote_host;
char            **guide;
char            *type;
{
    int     i;
    char    *ptr;
    char    tmp_text[40];
    Widget  tempwidget;


/*
 * Make an RPC call to retrieve tv guide.
 */
#ifdef TRACE
fprintf(stderr,"LDM:: MAKING RPC CALL->CDM_GET_LIST.\n");
#endif
    if (callrpc(remote_host,CDM_PROG,CDM_VERS,CDM_GET_LIST,xdr_void,0,
            xdr_tvguide,guide) != 0) {
        perror("LDM::callrpc (LDM_GET_LIST):");
        go_away(1);
    }

    ptr = (char *)&guide[0];

    for (i=0;i<10;i++) {
        if (current_recv[i] == 'R')
            sprintf(tmp_text,"(R) CH %d : %s",i,ptr);
        else if (current_recv[i] == 'D')
            sprintf(tmp_text,"(D) CH %d : %s",i,ptr);
        else
            sprintf(tmp_text,"    CH %d : %s",i,ptr);

        if (!strcmp("tv_gCB",type)) {
            b       t       n               t       e       x       t   =
XmStringCreateLtoR(tmp_text,XmSTRING_DEFAULT_CHARSET);
            n = 0;
            XtSetArg (args[n],XmNlabelString,btn_text); n++;
            XtSetValues(label_widget[i],args,n);
        }
        else {
            b       t       n               t       e       x       t   =
XmStringCreateLtoR(tmp_text,XmSTRING_DEFAULT_CHARSET);
            n = 0;
            XtSetArg (args[n],XmNlabelString,btn_text); n++;
            XtSetValues(channel_widget[i],args,n);
        }
```

```
        ptr+=CHANNAMLEN;
    }

} /* end retrieve_tvguide */

/*********************************************************************/
/*
 * This routine requests, through RPC, authorization to distribute
 * a window on a channel.
 */
void
dist_auth_request(remote_host,DistAuth,xid,buff)
char            *remote_host;
struct DistAuth *DistAuth;
int             xid;
char            *buff;
{
    struct ChanID   ChanID;
    XmString        name_string;
    char            *name;

/*
 * Request the operator to enter some sort of channel
 * identification to be used to mark the channel.
 */
    ChanID.pr_port          = shmem->pr_port;
    ChanID.default_gc       = (unsigned long)shmem->default_gc;
    ChanID.root             = (unsigned long)shmem->root;
    ChanID.distributor_id   = shmem->distributor_id;

    strncpy(ChanID.chanid,buff,SOURCENAMLEN);
    strncpy(ChanID.hostname,hostname,HOSTNAMLEN);

    /* below !!!! should be replace with actual xid value */
    /* of the window to be distributed                    */
    ChanID.xid  = CLIENT_ID(xid);
    shmem->window                   = ChanID.xid;
    shmem->start                    = TRUE;
/*
 * WARNING! below will have to be shifted by CLIENT_ID macro
 * before accessing array once the user interface is in and
 * the xid is a real live xid instead of a number entered by hand.
 */
    shmem->wanted[ChanID.xid]       = TRUE;

#ifdef TRACE
fprintf(stderr,"LDM::just set wanted flag for client:%d\n",ChanID.xid);
fprintf(stderr,"LDM::making rpc call to host      :<%s>\n",remote_host);
#endif

/*
 * Make an RPC call to send the Distribution request.
```

```
 * Pass in the requested channel id and source xid.
 */
#ifdef TRACE
fprintf(stderr,"LDM:: MAKING RPC CALL->CDM_DIST_REQ.\n");
#endif
    i                                                                    f
(callrpc(remote_host,CDM_PROG,CDM_VERS,CDM_DIST_REQ,xdr_ChanID,&ChanID,
                xdr_DistAuth,DistAuth)
                != 0) {
        perror("LDM::callrpc (LDM_DIST_AUTH):");
        go_away(1);
    }

#ifdef TRACE
if (DistAuth->authorization==AUTHORIZED)
fprintf(stderr,"LDM::Authorization : AUTHORIZED\n");
else
fprintf(stderr,"LDM::Authorization : NOT AUTHORIZED\n");
fprintf(stderr,"LDM::Channel       :%d\n",DistAuth->channel);
fprintf(stderr,"LDM::Port          :%d\n",DistAuth->pm_port);
#endif

/*
 * Place the port number of the Protocol Distributor in
 * shared memory.
 */
    shmem->pm_port  = DistAuth->pm_port;
        shmem->clients[DistAuth->channel]       = ChanID.xid;

#ifdef TRACE
fprintf(stderr,"LDM:: setting channel:%d to client:0x%x\n",
DistAuth->channel,ChanID.xid);
#endif

    /*** determine name ***/

    n = 0;
    XtSetArg (args[n], XmNlabelString, &name_string); n++;
    XtGetValues (channel_widget[DistAuth->channel], args, n);
    XmStringGetLtoR (name_string, XmSTRING_DEFAULT_CHARSET,&name);

    current_recv[DistAuth->channel] = 'D';
    name[0] = '(';
    name[1] = 'D';
    name[2] = ')';

    btn_text = XmStringCreateLtoR(name,XmSTRING_DEFAULT_CHARSET);

    n = 0;
    XtSetArg (args[n],XmNlabelString,btn_text); n++;
    XtSetValues(channel_widget[DistAuth->channel],args,n);
```

```
    XFlush(display);

} /* end dist_auth_request */
/*****************************************************************/


/*****************************************************************/
void
remove_channel(remote_host,channel)
char    *remote_host;
int     channel;
{
    int retval;

/*
 * Set the client to not wanted.
 */
    shmem->wanted[ shmem->clients[channel] ] = FALSE;
    sleep(2);

#ifdef TRACE
fprintf(stderr,"LDM:: set client:%d channel:%d OFF\n",
shmem->clients[channel],channel);
#endif

/*
 * Make an RPC call to remove a channel from service.
 */
#ifdef TRACE
fprintf(stderr,"LDM:: MAKING RPC CALL->CDM_REMV_CHAN\n");
#endif
    i                                                              f
(callrpc(remote_host,CDM_PROG,CDM_VERS,CDM_REMV_CHAN,xdr_int,&channel,
           xdr_int,&retval) != 0) {
        perror("LDM::callrpc (LDM_REMV_CHAN):");
        go_away(1);
    }

/*
 * Let operator know what is going on
 */
    if (retval==REQUEST_OK)
        fprintf(stderr,"LDM::Channel %d removed from service.\n",channel);
    else
        fprintf(stderr,"LDM::Channel    %d    NOT    removed    from
service.\n",channel);

} /* end remove_channel */
/*****************************************************************/


/*****************************************************************/
```

```
/*
 * This routine requests, through RPC, to receive a particular
 * channel.
 */
void
recv_auth_request(remote_host,RecvAuth,channel)
char            *remote_host;
struct RecvAuth *RecvAuth;
int             channel;
{
    struct ChanReq      ChanReq;

/*
 * Get the channel identification from the operator.
 */
    port_number = shmem->pr_port;
    ChanReq.channel = channel;
    sprintf(ChanReq.PortID.hostname,"%s",hostname);
    ChanReq.PortID.portnum  = port_number;
    ChanReq.distributor_id  = shmem->distributor_id;

#ifdef TRACE
fprintf(stderr,"LDM::   Requesting   channel:%d   for
port:%d\n",channel,port_number);
#endif

/*
 * Make an RPC call to receive reception authorization.
 */
#ifdef TRACE
fprintf(stderr,"LDM:: MAKING RPC CALL->CDM_RECV_REQ.\n");
#endif
    if (callrpc(remote_host,CDM_PROG,CDM_VERS,CDM_RECV_REQ,xdr_ChanReq,
                &ChanReq,xdr_RecvAuth,RecvAuth)
        != 0) {
        perror("LDM::callrpc (LDM_RECV_REQ):");
        go_away(1);
    }

    /* Set some values in shared memory */
    shmem->source_default_gc[channel]   = RecvAuth->default_gc;
    shmem->source_root[channel]         = RecvAuth->root;

#ifdef TRACE
if ( RecvAuth->authorization == AUTHORIZED)
fprintf(stderr,"LDM::Reception authorization   : AUTHORIZED\n");
else
fprintf(stderr,"LDM::Reception authorization   : NOT AUTHORIZED\n");
fprintf(stderr,"LDM::PC Port number           :%d\n",RecvAuth->pm_port);
fprintf(stderr,"LDM::Source    default    gc
:0x%x\n",RecvAuth->default_gc);
fprintf(stderr,"LDM::Source root               :0x%x\n",RecvAuth->root);
```

```
#endif

/*
 * Place the port number of the Protocol Distributor in
 * shared memory.
 */
    shmem->pm_port  = RecvAuth->pm_port;


} /* end recv_auth_request */
/*********************************************************************/


/*********************************************************************/
/*
 * This routine requests that a particular port (receiver) on a
 * particular channel, be removed from distribution.
 */
remove_receiver(remote_host,channel,port)
char            *remote_host;
int             channel;
unsigned short  port;
{
    struct RemvRecv RemvRecv;
    int             retval;

#ifdef TRACE
fprintf(stderr,"LDM::remove port:%d\n",port);
fprintf(stderr,"LDM::channel    :%d\n",channel);
#endif

/*
 * Make an RPC call to remove a receiver.
 */
    RemvRecv.channel        = channel;
    RemvRecv.portnum        = port;

#ifdef TRACE
fprintf(stderr,"LDM:: MAKING RPC CALL->CDM_REMV_RECV.\n");
#endif
    if(callrpc(remote_host,CDM_PROG,CDM_VERS,CDM_REMV_RECV,xdr_RemvRecv,
                &RemvRecv,xdr_int,&retval)
                != 0) {
        perror("LDM::callrpc (LDM_REMV_RECV):");
        go_away(1);
    }

    if (retval==REQUEST_OK)
        fprintf(stderr,"LDM::Request to remove receiver OK\n");
    else
        fprintf(stderr,"LDM::Request to remove receiver FAILED\n");
```

```
/*
 * Notify the Protocol Receiver to remove one
 */
    sleep(5);
    shmem->pr_close_client  = channel;
    shmem->pr_close_channel = TRUE;

} /* end remove_receiver */
/*********************************************************************/


/*********************************************************************/
/*
 * This routine attempts to read the /etc/host table and poll each
 * host until a CDM is found.
 */
#include <sys/socket.h>
#include <netdb.h>
int
contact_cdm(name)
char        *name;
{
    struct PortID   PortID;
    int             retval;

    sprintf(name,"");
#ifdef TRACE
fprintf(stderr,"LDM:: making clnt_broadcast...CDM_PRESENT?\n");
#endif
    retval = clnt_broadcast(CDM_PROG,CDM_VERS,CDM_PRESENT,xdr_void,0,
                    xdr_PortID,&PortID,callme);
    strncpy(name,PortID.hostname,PORTNAMLEN);

    if (retval==0)
        return(TRUE);
    else
        return(FALSE);

} /* end contact_cdm */
/*********************************************************************/


/*********************************************************************/
/*
 * This is a simple call-back routine from the client broadcast
 * search for a CDM host.
 */
int
callme(out,addr)
char                *out;
struct sockaddr_in  *addr;
{
```

```
    return(1);
} /* end call_me */
/*************************************************************************/

/*************************************************************************/
/*
 * This routine attaches to the shared memory area used between
 * the multicast (server mod) function, ldm, and protocol distributor
(me).
 */
void
attach_shared_memory()
{
    int             shmid;

    /* attach to shared memory */
    shmid   = shmget( (int)SM_KEY,sizeof(struct MC_SHMEMORY),0777);

    while (shmid<0) {
        shmid   = shmget( (int)SM_KEY,sizeof(struct MC_SHMEMORY),0777);
        perror("LDM::Unable to shmget Server shared memory.");
        sleep(2);
    }
    shmem   = (struct MC_SHMEMORY *)shmat(shmid,0,0);

    if (shmem==NULL) {
        perror("LDM::Unable to attach to Server shared memory.");
        go_away(1);
    }

/*
 * Set management host name in shared memory.
 */
    strncpy(shmem->management_host,management_host,HOSTNAMLEN);


#ifdef TRACE
printf("PD::attached to shared memory\n");
#endif

} /* end attach_shared_memory */
/*************************************************************************/

/*****************************************************/
/*
 * This routine sends a request to the CDM to tell it
 * to stop.
 */
stop_cdm(remote_host)
char        *remote_host;
{
    int retval;
```

```
/*
 * Ask CDM to stop.
 */
#ifdef TRACE
fprintf(stderr,"LDM:: MAKING RPC CALL->CDM_GO_AWAY.\n");
#endif
    i                                                                    f
(callrpc(remote_host,CDM_PROG,CDM_VERS,CDM_GO_AWAY,xdr_void,0,xdr_int,
                    &retval) != 0) {
        perror("LDM::callrpc (LDM_GO_AWAY):");
    }

} /* end stop_cdm */
/*****************************************************/


/*****************************************************/
/*
 * Call this routine instead of exit to clean things up.
 */
void
go_away(code)
int     code;
{
    fprintf(stderr,"LDM:: EXITING......\n");
    exit(code);
} /* end go_away */
/*****************************************************/


void helpCB()
{
    Widget  box;
    Widget  btn;

    n = 0;
    XtSetArg(args[n],XmNmessageString,
            XmStringCreate("No Help Available",XmSTRING_DEFAULT_CHARSET));
n++;

    box = XmCreateMessageDialog(
                            main_window,     /* parent
*/
                            "PopHelp",       /* widget name
*/
                            args,            /* argument list
*/
                            n);              /* number of arguments
*/
    XtManageChild(box);
```

```
    btn = XmMessageBoxGetChild(
                            box,                        /* parent
*/
                            XmDIALOG_HELP_BUTTON);   /* widget class
*/

    XtUnmanageChild(btn);

    btn = XmMessageBoxGetChild(
                            box,                        /* parent
*/
                            XmDIALOG_CANCEL_BUTTON);/* widget class
*/

    XtUnmanageChild(btn);
}

/*********************/
void infoCB()
{
    Widget  box;
    Widget  btn;
    char    *string = "Written by S. Johns";

    n = 0;
    XtSetArg(args[n],XmNmessageString,
            XmStringCreate(string,XmSTRING_DEFAULT_CHARSET)); n++;

    box = XmCreateMessageDialog(
                            main_window,     /* parent
*/
                            "PopHelp",       /* widget name
*/
                            args,            /* argument list
*/
                            n);              /* number of arguments
*/
    XtManageChild(box);

    btn = XmMessageBoxGetChild(
                            box,                        /* parent
*/
                            XmDIALOG_HELP_BUTTON);   /* widget class
*/

    XtUnmanageChild(btn);

    btn = XmMessageBoxGetChild(
                            box,                        /* parent
*/
                            XmDIALOG_CANCEL_BUTTON);/* widget class
*/
```

```
    XtUnmanageChild(btn);
}
```

APPENDIX O
CENTRAL DISTRIBUTION MANAGER LISTINGS

```
/*************************************************************************
The included program listings are prototypes, no warranty is expressed or
implied for their use in any other fashion.  They should not be considered
or used as production software.  The information in the listings is
supplied on an "as is" basis.  No responsibility is assumed for damages
resulting from the use of any information contained in the listings.

The software in these listings has been compiled on Masscomp 6350's and
6600's and on Sun 3's and 4's.  Modifications may be necessary for use on
other systems.
*************************************************************************/

/*
 * File         : cdm.c
 * Author       : P. Fitzgerald - SwRI
 * Date         : 10/3/89
 * Description  : This file contatins all the Central Distribution Manager
 *                code.
 */
#include <stdio.h>
#include <rpc/rpc.h>
#include <utmp.h>
#include <sys/types.h>
#include <rpcsvc/rusers.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#define NEED_REPLIES
#define NEED_EVENTS
#include <X11/Xproto.h>
#include <X11/Xlib.h>
#include "../includes/ds_manage.h"


/* EXTERNAL ROUTINES */
extern void handle_RPC();
extern void download_channel_map();

/* GLOBAL FUNCTIONS */
void        register_RPC();
void        go_away();

/* GLOBAL VARIABLES */
char                tv_guide[MAX_CHANNELS][CHANNAMLEN];
unsigned char       guide_on[MAX_CHANNELS];
unsigned short      pm_port = 0;
```

```
int                 pm_fd;
struct ChanMap      ChanMap[MAX_CHANNELS];
struct CDM_SHMEMORY *shmem;
int                 shmid;
char                management_host[HOSTNAMLEN];
int                 cdm_to_pm[2];
int                 pm_to_cdm[2];

/*********************************************************************/
/*
 * Main body
 */
main (argc,argv)
    int         argc;
    char        **argv;
{

/*
 * Retrieve the host name (Internet).
 */
    if ( gethostname(management_host,sizeof(management_host)) <0) {
        perror("CDM::gethostname:");
        go_away();
    }
    printf("SwRI Central Distribution Manager <%s> starting....\n",
                management_host);

/*
 * Initialize the world.
 */
    init();

/*
 * Fork off the Protocol Multiplexer
 */
    if (!create_multiplexer()) {
        fprintf(stderr,"CDM::Unable    to    create    the    Protocol
Multiplexer.\n");
        go_away();
    }

/*
 * Create a UDP socket, register RPC server and never return.
 */
    register_RPC();
    /* NO RETURN */

} /* end main */
/*********************************************************************/
```

```
/*************************************************************************/
/*
 * This routine creates a UDP socket and registers the service
 * with the RPC mechanism.
 */
void
register_RPC()
{
    SVCXPRT     *transp;


/*
 * Create a socket (UDP)
 */
    transp  = svcudp_create(RPC_ANYSOCK);
    if (transp == NULL ) {
        perror("CDM::svcudp_create:");
        go_away();
    }
    pmap_unset(CDM_PROG,CDM_VERS);

/*
 * register service with RPC mechanism
 */
    if (svc_register(transp,CDM_PROG,CDM_VERS, handle_RPC,
            IPPROTO_UDP) == 0) {
        perror("CDM::svc_register:");
        go_away();
    }
    svc_run();  /* never returns */
    perror("CDM::svc_run RETURNED:");
    exit(-1);

} /* end register_RPC */
/*************************************************************************/


/*************************************************************************/
/*
 * This routine performs all the initialization for the CDM.
 */
init()
{
    int         i;
    int         j;

/*
 * Now create a shared memory area
 */
    shmid   = create_shared_memory();

/*
```

```
* Initialize the tv guide
* and channel map
*/
    for (i=0;i<MAX_CHANNELS;i++) {
        sprintf(tv_guide[i],"%s","");
        guide_on[i]                          = FALSE;
        shmem->ChanMap[i].num_receivers      = 0;
        shmem->ChanMap[i].client_id          = 0;
        for (j=0;j<MAX_RECEIVERS;j++) {
            ChanMap[i].recv_ports[j]         = 0;
            shmem->ChanMap[i].recv_ports[j] = 0;
            shmem->dest_fd[i][j]             = -1;
        }
        shmem->source_fd[i]                  = -1;
        sprintf(shmem->ChanMap[i].source_hostname,"");
    }
    shmem->sm_status                         = AOK;
    shmem->read_pipe                         = FALSE;
    shmem->new_source_channel                = -1;
    shmem->remv_source_channel               = -1;
    shmem->new_receiver_channel              = -1;
    shmem->remv_receiver_channel             = -1;
    shmem->station_index                     = -1;
    for (i=0;i<MAX_STATIONS;i++) {
        shmem->Stations[i].pd_fd             = -1;
        shmem->Stations[i].pr_fd             = -1;
        shmem->Stations[i].num_channels      = 0;
        for (j=0;j<MAX_CHANNELS;j++) {
            shmem->Stations[i].dist_channel[j]  = -1;
            shmem->Stations[i].dist_client[j]   = -1;
        }
        sprintf(shmem->Stations[i].hostname,"");
    }

} /* end init */
/*********************************************************************/


/*********************************************************************/
/*
 * This routine creates the Protocol Multiplexer and a pipe to the
 * Multiplexer for communications.
 */
int
create_multiplexer()
{
    int             pid;
    char            readfd[2*sizeof(cdm_to_pm[0])];
    char            writefd[2*sizeof(pm_to_cdm[0])];
    char            sid[2*sizeof(shmid)];
    unsigned char   byte;
    int             bytes_read;
```

```
    static struct sockaddr_in   sinhim = { AF_INET };
    register struct hostent *hp;

/*
 * First create a pipe for communications.
 * One for writing to pm and one for reading from pm.
 */
    if ( pipe(cdm_to_pm)!= 0)  {
        perror("CDM::pipe (create_multiplexer):");
        return(FALSE);
    }
    if ( pipe(pm_to_cdm)!= 0)  {
        perror("CDM::pipe (create_multiplexer):");
        return(FALSE);
    }

/*
 * Set up parameters as character strings.
 */
    sprintf(readfd,"%d",cdm_to_pm[RFD]);
    sprintf(writefd,"%d",pm_to_cdm[WFD]);
    sprintf(sid,"%d",shmid);

#ifdef TRACE
fprintf(stderr,"CDM::Pipe file descriptors created.\n");
fprintf(stderr,"CDM::cdm_to_pm[0]:%d  cdm_to_pm[1]:%d\n",
cdm_to_pm[0],cdm_to_pm[1]);
fprintf(stderr,"CDM::pm_to_cdm[0]:%d  pm_to_cdm[1]:%d\n",
pm_to_cdm[0],pm_to_cdm[1]);
fprintf(stderr,"CDM::RFD:%d  WFD:%d\n",RFD,WFD);
#endif

/*
 * Now fork into two processes
 */
    if ( (pid=fork()) == 0) {
        execle("./pm","pm",readfd,writefd,sid,NULL,NULL);
        perror("CDM::execle (create_multiplexer):");
        go_away();
    }

#ifdef TRACE
fprintf(stderr,"CDM::After fork....goind to rendezvous\n");
#endif

/*
 * Now rendezvous with the pm
 */
    bytes_read  = read(pm_to_cdm[RFD],&byte,1);
    if (bytes_read != 1) {
        perror("CDM::read (create_multiplexer):");
        return(FALSE);
```

```
    }

#ifdef TRACE
fprintf(stderr,"CDM::After rendezvous\n");
#endif

/*
 * Now read the pm port number.
 */
    bytes_read = read(pm_to_cdm[RFD],&pm_port,sizeof(pm_port));
    if (bytes_read != sizeof(pm_port) ) {
        perror("CDM::read (create_multiplexer):");
        return(FALSE);
    }

#ifdef TRACE
fprintf(stderr,"CDM::Protocol Multiplexer port number read:%d\n",pm_port);
#endif

/*
 * Now get hostname, address, and connect to the Multiplexer.
 */
    hp  = gethostbyname(management_host);
    if (!hp) {
        fprintf(stderr,"CDM::Host '%s' not found\n",management_host);
        go_away();
    }
    bcopy(hp->h_addr, &sinhim.sin_addr, sizeof(sinhim.sin_addr));
    sinhim.sin_port = htons(pm_port);
    if ((pm_fd = socket(AF_INET,SOCK_STREAM,0))<0) {
        perror("CDM::socket (create_multiplexer):");
        go_away();
    }
    if (connect(pm_fd,&sinhim,sizeof(sinhim))<0) {
        perror("CDM::connect (create_multiplexer):");
        go_away();
    }

#ifdef TRACE
fprintf(stderr,"CDM::Socket and connect to PM ok.\n");
#endif
/*
 * Download the channel map
 */
    download_channel_map(-1,-1,-1,-1);

#ifdef TRACE
fprintf(stderr,"CDM::After download of channel map\n");
#endif

    return(TRUE);
```

```
} /* end create_multiplexer */
/*************************************************************************/


/*************************************************************************/
/*
 * This routine creates a shared memory area.
 */
int
create_shared_memory ()
{
#define RWMODE      0666

#ifdef TRACE
fprintf(stderr,"CDM::cdm_key:%d\n",CDM_KEY);
fprintf(stderr,"CDM::sizeof(struct CDM_SHMEMORY):0x%x\n",
sizeof(struct CDM_SHMEMORY));
#endif
    /* attach to shared memory */
    shmid   = shmget( (int)CDM_KEY,sizeof(struct CDM_SHMEMORY),
                IPC_CREAT|RWMODE);
    if (shmid<0) {
        perror("CDM::shmget (create_shared_memory):");
        go_away();
    }
    shmem = (struct CDM_SHMEMORY *)shmat(shmid,0,0);
    if (shmem==(struct CDM_SHMEMORY *)-1) {
        perror("CDM::shmat (create_shared_memory):");
        go_away();
    }

    return(shmid);

} /* end create shared memory */
/*************************************************************************/


/*************************************************************************/
/*
 * This routine replaces exit, and in fact calls it. It also may be
 * used to perform housekeeping.
 */
void
go_away()
{

/*
 * Get rid of the shared memory identifier.
 */
    shmctl(shmid,IPC_RMID,0);

/*
```

```
 * Tell someone we are going away.
 */
    fprintf(stderr,"CDM:: EXITING.......\n");
    sleep(2);
    exit(0);

} /* end go_away */
/*****************************************************************************/
```

```
/*
 * File          : cdm_rpc.c
 * Author        : P. Fitzgerald - SwRI
 * Date          : 10/4/89
 * Description   : This file contatins all the Central Distribution Manager
 *                 RPC request handling code.
 */
#include <stdio.h>
#include <rpc/rpc.h>
#include <utmp.h>
#include <sys/types.h>
#include <rpcsvc/rusers.h>
#include <X11/X.h>
#define NEED_REPLIES
#define NEED_EVENTS
#include <X11/Xproto.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include "../includes/ds_manage.h"

/* EXTERNAL ROUTINES */
extern                  xdr_PortID();
extern                  xdr_tvguide();
extern                  xdr_DistAuth();
extern                  xdr_RecvAuth();
extern                  xdr_ChanID();
extern                  xdr_RegDist();
extern                  xdr_RegRecv();
extern                  xdr_ChanReq();
extern                  xdr_RemvRecv();
extern                  go_away();

/* EXTERNAL VARIABLES */
extern  char                tv_guide[MAX_CHANNELS][CHANNAMLEN];
extern unsigned char        guide_on[MAX_CHANNELS];
extern  unsigned short      pm_port;
extern struct CDM_SHMEMORY  *shmem;
extern  struct ChanMap      ChanMap[MAX_CHANNELS];
extern char                 management_host[HOSTNAMLEN];
extern  int                 cdm_to_pm[2];
extern  int                 pm_to_cdm[2];

/* GLOBAL ROUTINES */
void                    send_back_int();
void                    send_back_tvguide();
void                    register_distributor();
void                    register_receiver();
void                    send_back_hostname();
void                    distribution_request();
void                    reception_request();
void                    download_channel_map();
int                     add_receiver();
```

```
void                    remove_receiver();
void                    remove_channel();
void                    copy_channel_map();

/*****************************************************************/
/*
 * This routine handles all the rpc calls for distribution management.
 */
void
handle_RPC(rqstp, transp)
    struct  svc_req     *rqstp;
    SVCXPRT             *transp;
{
    register struct CDM_SHMEMORY    *memptr;

    memptr  = shmem;
/*
 * Determine what type of request is being made and act upon it.
 */
#ifdef TRACE
fprintf(stderr,"\nCDM::HEY- WE GOT AN RPC CALL!!!!!.\n");
#endif
    switch (rqstp->rq_proc) {

/*******************************************************/
/*
 * Request to retrieve channel map - id list.
 */
        case CDM_GET_LIST:
#ifdef TRACE
fprintf(stderr,"CDM::CDM_GET_LIST request\n");
#endif
            send_back_tvguide(transp);
#ifdef TRACE
fprintf(stderr,"CDM::END...CDM_GET_LIST\n");
#endif
            return;
/*******************************************************/


/*******************************************************/
/*
 * Register Distributor Request
 */
        case CDM_REG_DIST:
#ifdef TRACE
fprintf(stderr,"CDM:: CDM_REG_DIST request\n");
#endif
            register_distributor(transp);
            return;
/*******************************************************/
```

```
/*********************************************************/
/*
 * Register Receiver Request
 */
        case CDM_REG_RECV:
#ifdef TRACE
fprintf(stderr,"CDM:: CDM_REG_RECV request\n");
#endif
            register_receiver(transp);
            return;
/*********************************************************/

/*********************************************************/
/*
 * Distribution authorization request.
 */
        case CDM_DIST_REQ:
#ifdef TRACE
fprintf(stderr,"CDM::CDM_DIST_REQ request\n");
#endif
            /* retrieve the requested source name (ID) */
            distribution_request(transp);
#ifdef TRACE
fprintf(stderr,"CDM::END...CDM_DIST_REQ\n");
#endif
            return;
/*********************************************************/



/*********************************************************/
/*
 * Reception authorization request.
 */
        case CDM_RECV_REQ:
#ifdef TRACE
fprintf(stderr,"CDM::CDM_RECV_REQ request\n");
#endif
            reception_request(transp);
#ifdef TRACE
fprintf(stderr,"CDM::END...CDM_RECV_REQ\n");
#endif
            return;
/*********************************************************/



/*********************************************************/
/*
 * Request to remove a channel.
 */
        case CDM_REMV_CHAN:
```

```
#ifdef TRACE
fprintf(stderr,"CDM::CDM_REMV_CHAN request\n");
#endif
            remove_channel(transp);
#ifdef TRACE
fprintf(stderr,"CDM::END...CDM_REMV_CHAN\n");
#endif
            return;
/*********************************************************/


/*********************************************************/
/*
 * Handle request to remove a receiver from the linked list.
 */
        case CDM_REMV_RECV:
#ifdef TRACE
fprintf(stderr,"CDM::CDM_REMV_RECV request\n");
#endif
            remove_receiver(transp);
#ifdef TRACE
fprintf(stderr,"CDM::END...CDM_REMV_RECV\n");
#endif
            return;
/*********************************************************/


/*********************************************************/
/*
 * Handle a request to go away.
 */
        case CDM_GO_AWAY:
#ifdef TRACE
fprintf(stderr,"CDM:: someone asked me to go away....\n");
#endif
            memptr->sm_status   = PM_DIE;
            send_back_int( (int)REQUEST_OK, transp);
            sleep(5);
#ifdef TRACE
fprintf(stderr,"CDM::bye bye you guys...\n");
#endif
            go_away();
/*********************************************************/


/*********************************************************/
/*
 * Handle a presence check. NOP.
 */
        case CDM_PRESENT:
```

```
#ifdef TRACE
fprintf(stderr,"CDM::CDM_PRESENT? request\n");
#endif
            send_back_hostname(transp,management_host);
#ifdef TRACE
fprintf(stderr,"CDM::END...CDM_PRESENT?\n");
#endif
            return;
/*********************************************************/

/*********************************************************/
/*
 * Error, invalid request
 */
        default:
#ifdef TRACE
fprintf(stderr,"CDM:: Unknown RPC request. What gives?\n");
#endif
            svcerr_noproc(transp);
            return;
/*********************************************************/
    } /* end switch */

} /* end handle rpc */
/*************************************************************************/

/*************************************************************************/
/*
 * This routine implements or handles requests for distribution.
 */
void
distribution_request(transp)
    SVCXPRT         *transp;
{
    register int                channel;

    struct DistAuth             DistAuth;
    struct ChanID               ChanID;

/*
 * Retrieve the requested channel ID.
 */
    if (!svc_getargs(transp,xdr_ChanID,&ChanID)) {
        perror("CDM::CDM_DIST_REQ: svc_getargs:");
#ifdef GO_AWAY
        go_away();
#endif
    }
    shmem->station_index    = ChanID.distributor_id;

#ifdef TRACE
fprintf(stderr,"MC:: source_index(distributor_id):%d\n",
```

```
ChanID.distributor_id);
#endif

/*
 * Find the first empty channel.
 */
    for (channel=0;channel<MAX_CHANNELS;channel++)
        if (!guide_on[channel])
            break;

/* If no more channels available, do not authorize distribution */
    if (channel>=MAX_CHANNELS) {
        DistAuth.authorization  = NOT_AUTHORIZED;
        DistAuth.channel        = -1;
        DistAuth.pm_port        = 0;
    }
/* Else, set pm port number, enter channel id into channel map */
    else {
        guide_on[channel]                       = TRUE;
        DistAuth.authorization                  = AUTHORIZED;
        DistAuth.pm_port                        = pm_port;
        DistAuth.channel                        = channel;
        shmem->Stations[ChanID.distributor_id].dist_client
                [shmem->Stations[ChanID.distributor_id].num_channels]
                    = ChanID.xid;
        shmem->ChanMap[channel].client_id   = ChanID.xid;

/* PFF - 3 lines below added back after blowing them away - 26JAN90 */
        shmem->source_default_gc[channel]   = ChanID.default_gc;
        shmem->source_root[channel]         = ChanID.root;
        strncpy(shmem->source_name[channel],ChanID.hostname,HOSTNAMLEN);

        strncpy(tv_guide[channel],ChanID.chanid,CHANNAMLEN);

strncpy(shmem->ChanMap[channel].source_hostname,ChanID.hostname,SOURCEN
AMLEN);
#ifdef TRACE
fprintf(stderr,"CDM::Authorizing    :%s\n",ChanID.chanid);
fprintf(stderr,"CDM::Host Name      :%s\n",ChanID.hostname);
fprintf(stderr,"CDM::Pr Port Number :%d\n",ChanID.pr_port);
fprintf(stderr,"CDM::Channel        :%d\n",channel);
fprintf(stderr,"CDM::PM Port Number :%d\n",pm_port);
fprintf(stderr,"CDM:: Source name  :<%s>\n",shmem->source_name[channel]);
fprintf(stderr,"CDM:: ChanID.name   :<%s>\n",ChanID.hostname);
fprintf(stderr,"CDM::Source gc      :0x%x\n",ChanID.default_gc);
fprintf(stderr,"CDM::Source root    :0x%x\n",ChanID.root);
fprintf(stderr,"CDM:: xid (client)  :0x%x\n",ChanID.xid);
#endif
    }

/*
 * Now send the reply back
```

```
 */
    if (!svc_sendreply(transp,xdr_DistAuth,&DistAuth)) {
        perror("CDM::svc_sendreply (DistAuth):");
#ifdef GO_AWAY
        go_away();
#endif
    }

/*
 * If we were able to fulfill the request, download the channel map
 * This is done here because if we don't return a reply to the calling
 * RPC caller, then it will send another RPC request very soon and we
 * don't want to handle two identical requests. Download channel map
 * could take a little bit, since the PM may be busy and cannot respond
 * immediately. The PM only checks for input on a given channel when ALL
 * protocol from a round ( a round is defined as checking all current
 * distributors for input and distributing to all active receivers) is
 * processed.
 */
    if (channel<MAX_CHANNELS)
        download_channel_map(channel,-1,-1,-1);

} /* end distribution_request */
/*************************************************************************/

/*************************************************************************/
/*
 * This routine sends back the host name.
 */
void
send_back_hostname(transp,hostname)
    SVCXPRT *transp;
    register char       *hostname;
{
    struct PortID   PortID;

    strncpy(PortID.hostname,hostname,PORTNAMLEN);
    PortID.portnum = shmem->pm_port;

#ifdef TRACE
fprintf(stderr,"CDM::Sending back hostname:<%s>\n",hostname);
fprintf(stderr,"CDM:: and pm port of      :%d\n",PortID.portnum);
#endif
    if (!svc_sendreply(transp,xdr_PortID,&PortID)) {
        perror("CDM::svc_sendreply (hostname):");
#ifdef GO_AWAY
        go_away();
#endif
    }

} /* end send_back_hostname */
/*************************************************************************/
```

```
/*************************************************************************/
/*
 * This routine sends back the current tv guide to the caller.
 */
void
send_back_tvguide(transp)
    SVCXPRT *transp;
{
    if (!svc_sendreply(transp,xdr_tvguide,tv_guide)) {
        perror("CDM::svc_sendreply (tv_guide):");
#ifdef GO_AWAY
        go_away();
#endif
    }

} /* end send_back_tvguide */
/*************************************************************************/

/*************************************************************************/
/*
 * This routine sends back messages to the caller.
 */
void
send_back_int(value,transp)
    int                 value;
    SVCXPRT *transp;
{
    if (!svc_sendreply(transp,xdr_int,&value)) {
        perror("CDM::svc_sendreply (int):");
#ifdef GO_AWAY
        go_away();
#endif
    }

} /* end send_back */
/*************************************************************************/


/*************************************************************************/
/*
 * This routine removes a channel from distribution.
 */
void
remove_channel(transp)
    SVCXPRT *transp;
{
    register int            j;
    register int            *ch_ptr;
    register int            ch;
    int                     channel;
```

```
/*
 * Retrieve the channel number to remove
 */
    ch_ptr  = &channel;
    if (!svc_getargs(transp,xdr_int,ch_ptr)) {
        perror("CDM::CDM_REMV_CHAN: svc_getargs:");
        send_back_int( (int)REQUEST_FAILED,transp );
#ifdef GO_AWAY
        go_away();
#endif
    }
    send_back_int( (int)REQUEST_OK,transp);
    ch      = channel;

#ifdef TRACE
fprintf(stderr,"CDM:: remove_channel for channel:%d\n",ch);
#endif

/*
 * Close down that channel
 */
    guide_on[ch]            = FALSE;
    sprintf(tv_guide[ch],"");

/*
 * Send new channel map to concentrator.
 */
    download_channel_map(-1,ch,-1,-1);
    ChanMap[ch].num_receivers   = 0;
    for (j=0;j<MAX_RECEIVERS;j++) {
        ChanMap[ch].recv_ports[j]   = 0;
        sprintf(ChanMap[ch].recv_hostname[j],"");
    }

#ifdef TRACE
fprintf(stderr,"CDM::Channel %d removed from service.\n",ch);
#endif

} /* end remove_channel */
/*************************************************************************/


/*************************************************************************/
/*
 * This routine downloads, to the Protocol Concentrator, the current
 * contents of the channel map.
 */
void
download_channel_map(new_source_chan,remv_source_chan,
                     new_receiver_chan,remv_receiver_chan)
        register int    new_source_chan,remv_source_chan;
        register int    new_receiver_chan,remv_receiver_chan;
```

```
{
    register struct CDM_SHMEMORY    *memptr;
    register int                    bytes_written;
    register int                    bytes_read;

    unsigned char   byte;

    memptr      = shmem;

#ifdef TRACE
fprintf(stderr,"CDM:: before setting read_pipe to TRUE\n");
#endif

    memptr->read_pipe   = TRUE;

#ifdef TRACE
fprintf(stderr,"CDM::download_channel_map>  just  set  read_pipe  to
TRUE....\n");
#endif

/*
 * Now wait on confirmation that PM is ready to halt.
 */

#ifdef TRACE
fprintf(stderr,"CDM:: reading IM_WAITING byte from pm\n");
#endif

    bytes_read = read(pm_to_cdm[RFD],&byte,1);
    if (bytes_read!=1) {
        perror("CDM::read (download_channel_map):");
#ifdef GO_AWAY
        go_away();
#endif
    }
    if (byte==IM_WAITING) {
#ifdef TRACE
fprintf(stderr,"CDM:: just read IM_WAITING from PM\n");
#endif
        memptr->new_source_channel  = new_source_chan;
        memptr->remv_source_channel = remv_source_chan;
        memptr->new_receiver_channel    = new_receiver_chan;
        memptr->remv_receiver_channel= remv_receiver_chan;
        copy_channel_map();
    } /* end if */
    else
#ifdef TRACE
{
fprintf(stderr,"CDM:: just read %d\n",byte);
#endif
        fprintf(stderr,"CDM::Protocol Concentrator out of sync. (Channel
Map).\n");
```

```
#ifdef TRACE
)
#endif

/*
 * Now tell the PM that we are through copying.
 * But, only do it if we are in sync with pm.
 */
    if (byte==IM_WAITING) {
        byte    = IM_DONE;
        bytes_written = write(cdm_to_pm[WFD],&byte,1);
#ifdef TRACE
fprintf(stderr,"CDM:: just write IM_DONE\n");
#endif
        if (bytes_written!=1) {
            perror("CDM::write (download_channel_map):");
#ifdef GO_AWAY
            go_away();
#endif
        )
    } /* end if */

#ifdef TRACE
fprintf(stderr,"CDM::download_channel_map> done!\n");
#endif

} /* end download_channel_map */
/*************************************************************************/


/*************************************************************************/
/*
 * This routine handles reception requests.
 */
void
reception_request(transp)
    SVCXPRT         *transp;
{
    struct ChanReq  ChanReq;
    struct RecvAuth RecvAuth;
    int             ok;

/*
 * Retrieve the channel request structure.
 */
    if (!svc_getargs(transp,xdr_ChanReq,&ChanReq)) {
        perror("CDM::CDM_RECV_REQ: svc_getargs:");
#ifdef GO_AWAY
        go_away();
#endif
    }
```


C-4

```
#ifdef TRACE
fprintf(stderr,"CDM::Request received to receive
channel:%d\n",ChanReq.channel);
fprintf(stderr,"CDM::Request was from:%s\n",ChanReq.PortID.hostname);
fprintf(stderr,"CDM::receiver_index(distributor_id):%d\n",ChanReq.distr
ibutor_id);
#endif
#ifdef SLOW
sleep(5);
#endif

/*
 * Determine if anybody is broadcasting on that channel.
 */
    if (    ChanReq.channel>=MAX_CHANNELS ||
            !guide_on[ChanReq.channel]) {
#ifdef TRACE
fprintf(stderr,"CDM:: Nobody is broadcasting on that channel yet.\n");
#endif
            RecvAuth.authorization  = NOT_AUTHORIZED;
            RecvAuth.pm_port        = 0;
    } /* end if */

    else {
/*
 * Now add that reciever to the list on the channel map and
 * then download the new map to the Protocol Concentrator.
 */

#ifdef TRACE
fprintf(stderr,"CDM:: Adding receiver channel:%d port:%d name:<%s>.\n",
ChanReq.channel,ChanReq.PortID.portnum,ChanReq.PortID.hostname);
#endif
            ok  = add_receiver(ChanReq.channel,ChanReq.PortID.portnum,
                               ChanReq.PortID.hostname);
            if (!ok)    {
                RecvAuth.authorization  = NOT_AUTHORIZED;
                RecvAuth.pm_port        = 0;
#ifdef TRACE
fprintf(stderr,"CDM:: add_receiver request NOT AUTHORIZED.\n");
#endif
            }
            else {
#ifdef TRACE
fprintf(stderr,"CDM:: add_receiver request AUTHORIZED.\n");
#endif
                RecvAuth.authorization  = AUTHORIZED;
                RecvAuth.pm_port        = pm_port;
                RecvAuth.default_gc     =
shmem->source_default_gc[ChanReq.channel];
                RecvAuth.root           =
shmem->source_root[ChanReq.channel];
```

```
                shmem->station_index    = ChanReq.distributor_id;
#ifdef TRACE
fprintf(stderr,"CDM:: source_default_gc:0x%x\n",RecvAuth.default_gc);
fprintf(stderr,"CDM:: source_root      :0x%x\n",RecvAuth.root);
#endif
            }
    } /* end else */

#ifdef SLOW
sleep(1);
#endif

/*
 * Now send the reply back
 */
    if (!svc_sendreply(transp,xdr_RecvAuth,&RecvAuth)) {
        perror("CDM::svc_sendreply (RecvAuth):");
#ifdef GO_AWAY
        go_away();
#endif
    }

/*
 * Note: the download of the channel map is done AFTER the svc_sendreply
so that
 * the caller can get a timely response back and does not re-issue the RPC
call.
 */
    if (ok)
        download_channel_map(-1,-1,ChanReq.channel,-1);

} /* end reception_request */
/*************************************************************************/

/*************************************************************************/
/*
 * This routine adds a receiver to the channel map, and then downloads
 * the channel map to the protocol concentrator.
 */
int
add_receiver(channel,port,remote_name)
    int             channel;
    unsigned short  port;
    char            *remote_name;
{
    int             where;

#ifdef TRACE
fprintf(stderr,"CDM::add_receiver routine called.\n");
#endif

    if (ChanMap[channel].num_receivers>=MAX_RECEIVERS)
```

```
        return(FALSE);

/*
 * Search around for an empty place.
 */
    for (where=0;where<MAX_RECEIVERS;where++)
#ifdef TRACE
{
#endif
        if (ChanMap[channel].recv_ports[where]==0)
            break;
#ifdef TRACE
else
fprintf(stderr,"CDM:: slot %d not empty.\n",where);
}
#endif


#ifdef TRACE
fprintf(stderr,"CDM:: Found empty slot in channel map:%d\n",where);
#endif

    ChanMap[channel].recv_ports[where] = port;
#ifdef TRACE
fprintf(stderr,"CDM:: Placed port:%d in recv_port[%d]\n",port,where);
#endif
    s        t        r        n        c        p        y        (
ChanMap[channel].recv_hostname[where],remote_name,PORTNAMLEN);
#ifdef TRACE
fprintf(stderr,"CDM:: Copied port name <%s> into channel map.\n",
remote_name);
fprintf(stderr,"CDM:: channel:%d  receiver:%d\n",channel,where);
#endif
    ChanMap[channel].num_receivers++;

#ifdef TRACE
fprintf(stderr,"CDM::Adding a receiver for channel:%d\n",channel);
fprintf(stderr,"CDM::Port                          :%d\n",port);
fprintf(stderr,"CDM::Port name                     :<%s>\n",remote_name);
fprintf(stderr,"CDM::Number of recievers now       :%d\n",
ChanMap[channel].num_receivers);
#endif

#ifdef TRACE
fprintf(stderr,"CDM:: end add_receiver.\n");
#endif
    return(TRUE);

} /* end add_receiver */
/*****************************************************************************/
```

```
/*************************************************************************/
/*
 * This routine removes a receiver from the distribution list.
 */
void
remove_receiver(transp)
    SVCXPRT *transp;
{
    register    int         i;
    struct      RemvRecv    RemvRecv;

#ifdef TRACE
int         j;
#endif

/*
 * Retrieve the port number and channel to remove.
 */
    if (!svc_getargs(transp,xdr_RemvRecv,&RemvRecv)) {
        perror("CDM::CDM_REMV_RECV: svc_getargs:");
#ifdef GO_AWAY
        go_away();
#endif
    }

#ifdef TRACE
fprintf(stderr,"CDM::looking for port entry #%d\n",RemvRecv.portnum);
fprintf(stderr,"CDM::Channel number       :%d\n",RemvRecv.channel);
fprintf(stderr,"CDM::number of receivers
:%d\n",ChanMap[RemvRecv.channel].num_receivers);
#endif

/*
 * Download new channel map and clear out port number
 */
#ifdef TRACE
fprintf(stderr,"CDM::number of
receivers:%d\n",ChanMap[RemvRecv.channel].num_receivers);
for (j=0;j<ChanMap[RemvRecv.channel].num_receivers;j++) {
fprintf(stderr,"CDM::Channel %d: Port(%d)=%d\n",
RemvRecv.channel,j,ChanMap[RemvRecv.channel].recv_ports[j]);
}
#endif
        ChanMap[RemvRecv.channel].num_receivers--;
        if (ChanMap[RemvRecv.channel].num_receivers<0)
            ChanMap[RemvRecv.channel].num_receivers = 0;
        send_back_int( (int)REQUEST_OK, transp);
/*
 * Determine which port number for which receiver and
 * set it to zero.
 */
        for (i=0;i<MAX_RECEIVERS;i++) {
```

```
            if (ChanMap[RemvRecv.channel].recv_ports[i]==RemvRecv.portnum)
{
                ChanMap[RemvRecv.channel].recv_ports[i] = 0;
#ifdef TRACE
fprintf(stderr,"PM:: just set ChanMap[%d].recv_ports[%d] to zero.\n",
RemvRecv.channel,i);
#endif
                break;
            } /* end if */
        } /* end for */
        download_channel_map(-1,-1,-1,RemvRecv.channel);
#ifdef TRACE
for (j=0;j<ChanMap[RemvRecv.channel].num_receivers;j++)
fprintf(stderr,"CDM::Channel %d: Port(%d)=%d\n",
RemvRecv.channel,ChanMap[RemvRecv.channel].recv_ports[j]);
#endif

} /* end remove_receiver */
/*************************************************************************/


/*************************************************************************/
/*
 * This routine copies the local channel map to the shared memory
 * area for use by the Protocol Concentrator.
 */
void
copy_channel_map()
{
    register int                    i,j;
    register struct CDM_SHMEMORY    *memptr;

    memptr  = shmem;

#ifdef SLOW
fprintf(stderr,"CDM::copying channel map into shared memory.\n");
#endif
    for (i=0;i<MAX_CHANNELS;i++)  {
        memptr->ChanMap[i].num_receivers    = ChanMap[i].num_receivers;
        for (j=0;j<MAX_RECEIVERS;j++) {
            memptr->ChanMap[i].recv_ports[j]    =
ChanMap[i].recv_ports[j];

strncpy(memptr->ChanMap[i].recv_hostname[j],ChanMap[i].recv_hostname[j]
,PORTNAMLEN);
        }
    }
#ifdef SLOW
fprintf(stderr,"CDM:: copy done.\n");
sleep(1);
#endif
```

```
} /* end copy_channel_map */
/*************************************************************************/


/*************************************************************************/
/*
 * This routine acquires a unique distributor id and returns it
 */
void
register_distributor(transp)
    SVCXPRT *transp;
{
    struct  DistRegister    DistReg;

/*
 * Retrieve the hostname for the distributor
 */
    if (!svc_getargs(transp,xdr_RegDist,&DistReg)) {
        perror("CDM::CDM_REG_DIST: svc_getargs:");
#ifdef GO_AWAY
        go_away();
#endif
    }
/*
 * Retrieve an id for this guy
 */
    DistReg.distributor_id  = get_dist_id();

/*
 * Place the hostname and id into shared memory
 */
    sprintf(shmem->Stations[DistReg.distributor_id].hostname,
                "%s",DistReg.distname);

#ifdef TRACE
fprintf(stderr,"CDM:: Distributor name to register:<%s>\n",
                DistReg.distname);
#endif

/*
 * Now download the channel map and indicate that a new
 * distributor is awaiting connection.
 */

download_channel_map(REGISTER_DISTRIBUTOR,-(DistReg.distributor_id+1),-
1,-1);

/*
 * Now send the reply back
 */
    if (!svc_sendreply(transp,xdr_RegDist,&DistReg)) {
        perror("CDM::svc_sendreply (RegDist):");
```

```
#ifdef GO_AWAY
        go_away();
#endif
    }

} /* end register_distributor */
/*************************************************************************/


/*************************************************************************/
/*
 * This routine looks into shared memory and comes up with a unique
 * id (index into array) for this particular distributor.
 */
int
get_dist_id()
{
    register int        i;

    for (i=0;i<MAX_STATIONS;i++) {
        if (shmem->Stations[i].pd_fd<0)
            return(i);
    }
    fprintf(stderr,"MC:: Unable to register distributor, no more
room.\n");
#ifdef GO_AWAY
    go_away();
#endif

} /* end get_dist_id */
/*************************************************************************/

/*************************************************************************/
/*
 * This routine registers a receiver with a particular distributor.
 */
void
register_receiver(transp)
    SVCXPRT *transp;
{
    struct PortID           PortID;
    struct  RecvRegister    RecvReg;

/*
 * Retrieve the hostname for the distributor
 */
    if (!svc_getargs(transp,xdr_RegRecv,&RecvReg)) {
        perror("CDM::CDM_REG_RECV: svc_getargs:");
#ifdef GO_AWAY
        go_away();
#endif
    }
```

```
#ifdef TRACE
fprintf(stderr,"CDM:: Receiver name to register:<%s>\n",
                RecvReg.recvname);
fprintf(stderr,"CDM:: for index:%d\n",RecvReg.distributor_id);
fprintf(stderr,"CDM:: for portnum:%d\n",RecvReg.portnum);
#endif

/*
 * Now download the channel map and indicate that a new
 * distributor is awaiting connection.
 */
    download_channel_map(REGISTER_RECEIVER,-(RecvReg.distributor_id+1),
                    -((int)RecvReg.portnum),-1);

/*
 * Now send a reply back. Note: this avoids repeated sending out of
 * the RPC request by the sender.
 */
    if (!svc_sendreply(transp,xdr_PortID,&PortID)) {
        perror("CDM::svc_sendreply (register_receiver):");
#ifdef GO_AWAY
        go_away();
#endif
    }

} /* end register_receiver */
/*************************************************************************/
```

APPENDIX P
PROTOCOL MULTIPLEXER LISTINGS

```
/*************************************************************************
The included program listings are prototypes, no warranty is expressed or
implied for their use in any other fashion.  They should not be considered
or used as production software.  The information in the listings is
supplied on an "as is" basis.  No responsibility is assumed for damages
resulting from the use of any information contained in the listings.

The software in these listings has been compiled on Masscomp 6350's and
6600's and on Sun 3's and 4's.  Modifications may be necessary for use on
other systems.
*************************************************************************/

#define MAX_TRIES       150
/*
 * File          : pm.c
 * Author        : P. Fitzgerald - SwRI
 * Date          : 10/3/89
 * Description   : This file contains the code for the Protocol  *
       Multiplexer.
 */
#include <stdio.h>
#include <rpc/rpc.h>
#include <utmp.h>
#include <sys/types.h>
#include <rpcsvc/rusers.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <fcntl.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <signal.h>
#include <errno.h>
#include <X11/X.h>
#define NEED_REPLIES
#define NEED_EVENTS
#include <X11/Xproto.h>
#include <X11/Xlib.h>
#include "../includes/ds_manage.h"
#include "../includes/smtypes.h"
#include "../includes/smdef.h"
#include "../includes/dist.h"

/* DEFINES */
#define     LISTEN_BACKLOG  140

/* EXTERNAL ROUTINES */
```

```
/* GLOBAL FUNCTIONS */
void            attach_shared_memory();
void            set_alarm();
void            clear_alarm();
void            check_for_new_connections();
void            check_for_input();
void            close_receiver_fd();
void            accept_source();
void            close_source();
void            connect_receiver();
void            close_receiver();
void            go_away();

/* GLOBAL VARIABLES */
int             cdm_write_fd;
int             cdm_read_fd;
int             protofd;
unsigned short  port_number=0;
int             my_fd;
char            hostname[HOSTNAMLEN];
struct ChanMap  *ChanMap;
struct CDM_SHMEMORY *shmem;
int                 shmid;

/*************************************************************************/
/*
 * Main body
 */
main (argc,argv)
    int         argc;
    char        **argv;
{
    register struct CDM_SHMEMORY    *memptr;

/*
 * Convert input parameters into file descriptors.
 */
    sscanf(argv[1],"%d",&cdm_read_fd);
    sscanf(argv[2],"%d",&cdm_write_fd);
    sscanf(argv[3],"%d",&shmid);

/*
 * Call the initialization routine
 */
    pminit();

/*
 * Loop forever, processing new channel map and redirecting
 * input from sources to all requested destinations.
 */
    memptr          = shmem;
```

```
    while (1) {

/*
 * See if there is a new channel map from the CDM.
 */
        if (memptr->read_pipe)
            read_channel_map();

/*
 * See if there is any protocol to distribute.
 */
        check_for_input();

/*
 * See if we are being requested to exit.
 */
        if (memptr->sm_status==PM_DIE) {
            fprintf(stderr,"PM::Requested to exit.\n");
            go_away();
        }

    } /* end while */

} /* end main */
/*************************************************************************/


/*************************************************************************/
/*
 * This routine goes through the channel map, whenever there is
 * a new connection to be made, it makes it.
 */
void
check_for_new_connections()
{
    register struct CDM_SHMEMORY    *memptr;
    register int                    dist_id;
    register int                    recv_port;

    memptr  = shmem;
#ifdef MANAGE
fprintf(stderr,"PM::check for new connections\n");
fprintf(stderr,"PM::  new_source_channel->%d\n",shmem->new_source_channel);
fprintf(stderr,"PM::  remv_source_channl->%d\n",shmem->remv_source_channel);
fprintf(stderr,"PM::  new_receiver_chann->%d\n",shmem->new_receiver_channel);
fprintf(stderr,"PM::  remv_receiver_chan->%d\n",shmem->remv_receiver_channel);
#endif
/*
 * Set an alarm in case we get hung up.
```

```
 */
    set_alarm(500);

/*
 * Check the flags to see if we are accepting connections,
 * connecting, or closing connections.
 */
    if (memptr->new_source_channel>=0)
        accept_source(memptr->new_source_channel);
    else if (memptr->remv_source_channel>=0)
        close_source(memptr->remv_source_channel);
    else if (memptr->new_receiver_channel>=0)
        connect_receiver(memptr->new_receiver_channel);
    else if (memptr->remv_receiver_channel>=0)
        close_receiver(memptr->remv_receiver_channel);

/*
 * Check to see if a Protocol Distributor wants to
 * register with us.
 */
    if (memptr->new_source_channel==REGISTER_DISTRIBUTOR) {
        dist_id = -(memptr->remv_source_channel)-1;
#ifdef MANAGE
fprintf(stderr,"PM::   Protocol   Distributor   is   registerring
id:%d\n",dist_id);
#endif
        accept_distributor(dist_id);
    } /* end if */

/*
 * Check to see if a Protocol Receiver wants to
 * register with us.
 */
    if (memptr->new_source_channel==REGISTER_RECEIVER) {
        dist_id     = -(memptr->remv_source_channel)-1;
        recv_port   = -(memptr->new_receiver_channel);
#ifdef MANAGE
fprintf(stderr,"PM:: Protocol Receiver is registering id:%d\n",dist_id);
fprintf(stderr,"PM:: Receiver port is                 :%d\n",recv_port);
#endif
        accept_receiver(dist_id,(unsigned short)recv_port);
    } /* end if */

/*
 * Clear those flags
 */
    memptr->new_source_channel      = -1;
    memptr->new_receiver_channel    = -1;
    memptr->remv_source_channel     = -1;
    memptr->remv_receiver_channel   = -1;

/*
```

```
 * Clear that alarm
 */
    clear_alarm();

#ifdef MANAGE
fprintf(stderr,"PM:: END check_for_new_connections\n");
#endif

} /* end check_for_new_connections */
/*****************************************************************************/


/*****************************************************************************/
/*
 * This routine replaces the exit call (and in fact calls it). It may
 * also be used for cleanup before exiting.
 */
void
go_away()
{

/*
 * Notify someone that we are going away.
 */
    fprintf(stderr,"PM:: EXITING.....\n");
    sleep(2);
    exit(1);

} /* end go_away */
/*****************************************************************************/
```

```
#define MAX_TRIES       150
/*
 * File          : pmio.c
 * Author        : P. Fitzgerald - SwRI
 * Date          : 10/3/89
 * This file contains code for all the i/o functions of the Protocol  *
Multiplexer
 */
#include <stdio.h>
#include <rpc/rpc.h>
#include <utmp.h>
#include <sys/types.h>
#include <rpcsvc/rusers.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <fcntl.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/time.h>
#include <signal.h>
#include <errno.h>
#include <X11/X.h>
#define NEED_REPLIES
#define NEED_EVENTS
#include <X11/Xproto.h>
#include <X11/Xlib.h>
#include "../includes/ds_manage.h"
#include "../includes/smtypes.h"
#include "../includes/smdef.h"
#include "../includes/dist.h"
#include "../includes/xdefs.h"

/* EXTERNAL ROUTINES */
extern int      client_to_channel();

/* GLOBAL FUNCTIONS */
int             netread();
void            attach_shared_memory();
int             read_protocol();
void            set_alarm();
void            clear_alarm();
void            route_protocol();
void            check_for_new_connections();
void            get_window_attributes();
void            get_graphics_context();
void            check_for_input();
void            close_receiver_fd();
void            request_receiver();
void            send_graphics_context();
void            send_window_attributes();
```

```
/* GLOBAL VARIABLES */
extern int                  cdm_write_fd;
extern int                  cdm_read_fd;
extern int                  protofd;
extern int                  shmid;
extern  struct CDM_SHMEMORY *shmem;

#ifdef BYTES
int sig_bytes;
#endif

/*************************************************************************/
/*
 * This routine reads the entire channel map (in pieces) from the
 * Central Distribution Manager.
 */
read_channel_map()
{
    register int    bytes_read;
    register int    bytes_written;
    unsigned char   byte;

/*
 * Set an alarm before we read the channel map.
 */
    set_alarm(500);

    shmem->read_pipe    = FALSE;

/*
 * Tell CDM that we will wait until he says to go
 */
    byte    = IM_WAITING;
    bytes_written   = write(cdm_write_fd,&byte,1);
    if (bytes_written != 1) {
        perror("PM::write (read_channel_map):");
#ifdef GO_AWAY
        go_away();
#endif
    }

/*
 * Now read until byte read.
 */
    bytes_read  = read(cdm_read_fd,&byte,1);
    while (bytes_read<=0) {
        bytes_read  = read(cdm_read_fd,&byte,1);
        if (shmem->sm_status==PM_DIE) {
            fprintf(stderr,"PM::Requested to exit.\n");
            go_away();
        }
    }
```

```
    if (bytes_read != 1) {
        perror("PM::read (read_channel_map):");
#ifdef GO_AWAY
        go_away();
#endif
    }
    if (byte!=IM_DONE)
        fprintf(stderr,"PM::CDM out of sync. Read channel map.\n");

#ifdef MANAGE
fprintf(stderr,"PM::Got a new channel map.\n");
#endif

/*
 * Now clear the alarm.
 */
        clear_alarm();

/*
 * Now that we have read the channel map, we need to go
 * through it to determine if there are any new connections
 * to make.
 */
    check_for_new_connections();

} /* end read channel map */
/*************************************************************************/


/*************************************************************************/
/*
 * This routine checks to see if it can read X protocol from the
 * socket.
 */
int
read_protocol(fd,cp,channel,station)
    register int                fd;
    register union COMPAK       *cp;
    register int                *channel;
    register int                station;
{
    register int                bytes_read;
    register int                packet_length;
    int                         wat_id,wat_channel;
    int                         gc_id,gc_channel;
    XID                         client;
    unsigned short              *shortptr;

/*
 * Attempt to read some protocol from the nonblocking file descriptor.
 * If there is data there, then we will handle the type of request
 * that is being made, otherwise, return false.
```

```
 */
    bytes_read  = read(fd,cp->pdtopm.signal,SIGLEN);
    if (bytes_read<=0)
        return(FALSE);
    else if (bytes_read!=SIGLEN) {
        fprintf(stderr,"PM:: Read of partial signal. Performing netread
for remainder.\n");
        b   y   t   e   s   _   r   e   a   d           +   =
netread(fd,&cp->pdtopm.signal[bytes_read],SIGLEN-bytes_read);
        fprintf(stderr,"PM:: Network read of partial signal remainder
complete.\n");
    } /* end else */

#ifdef BYTES
sig_bytes = bytes_read;
shmem->no_of_packets++;
#endif

#ifdef TBYTES
fprintf(stderr,"Sig: 0x%x (%d)\n",cp->pdtopm.signal[0],sig_bytes);
#endif

#ifdef DATA
fprintf(stderr,"PM:: signal_byte read(0):0x%x\n",cp->pdtopm.signal[0]);
fprintf(stderr,"PM:: length of signal:%d\n",bytes_read);
#endif

/*
 * Now determine what type of SIGNAL byte was read.
 */
    switch(cp->pdtopm.signal[0]) {

/*************************************************************/
/*
 * If this is an expose event request, then call
 * the proper routine to handle and then return FALSE.
 */
        case EXPOSE:
#ifdef TRACE
fprintf(stderr,"PM:: that was an expose signal byte.\n");
#endif
/*
 * Now read which window/client the expose event is for
 */
            bytes_read = netread(fd,&client,sizeof(XID));
            if (bytes_read != sizeof(XID) ) {
                perror("PM:: read (reading xid):");
#ifdef GO_AWAY
                go_away();
#endif
            }
```

```
#ifdef TRACE
fprintf(stderr,"PM:: just read client of :0x%x\n",client);
#endif

/*
 * Determine which channel this is on.
 */
            *channel    = fd_to_channel(fd);

#ifdef TRACE
fprintf(stderr,"PM:: expose for client:0x%x\n",client);
fprintf(stderr,"PM:: channel           :%d\n",*channel);
#endif

#ifdef BYTES
shmem->channel_bytes[ *channel ] += sig_bytes + bytes_read;
shmem->distributor_bytes[ station ] += sig_bytes + bytes_read;
#endif

#ifdef TBYTES
fprintf( stderr, "PM:: - %d expose\n", bytes_read );
fprintf( stderr, "PM:: ...channel %d, station %d\n", *channel, station );
#endif

/*
 * Send the request on the the Protocol Receiver
 */
            request_receiver(client,*channel,(unsigned char)EXPOSE,0);
            break;
/*************************************************************/

/*************************************************************/
/*
 * Check to see if this is a request to retrieve window
 * attributes.
 */
        case GWATS:
#ifdef TRACE
fprintf(stderr,"PM:: Retrieve Window Attributes request.\n");
#endif
            /* read the header information */
            bytes_read = netread(fd,&cp->pdtopm.header,(int)HDRLEN);
            if (bytes_read!=HDRLEN) {
                perror("PM:: read length (GWATS:read_protocol):");
#ifdef GO_AWAY
                go_away();
#endif
            }
            shortptr    = (unsigned short *)&cp->pdtopm.signal[2];
            wat_channel = cp->pdtopm.header.length;
            wat_id      = cp->pdtopm.header.client;
```

```
#ifdef TRACE
fprintf(stderr,"PM::    Window:0x%x        channel:0x%x
wat_port:%d\n",wat_id,wat_channel,*shortptr);
#endif

#ifdef BYTES
shmem->channel_bytes[ wat_channel ] += sig_bytes + bytes_read;
shmem->distributor_bytes[ station ] += sig_bytes + bytes_read;
#endif

#ifdef TBYTES
fprintf( stderr, "PM:: - %d gwat\n", bytes_read );
fprintf( stderr, "PM:: ...channel %d, station %d\n", wat_channel, station
);
#endif

            get_window_attributes((XID)wat_id,wat_channel,*shortptr);
            break;
/*************************************************************/

/*************************************************************/
/*
 * Check to see if this is a request to retrieve graphcis
 * context state information.
 */
        case GGCS:
#ifdef TRACE
fprintf(stderr,"PM:: Retrieve Graphics Context request.\n");
#endif
            /* retrieve the port number of the receiver */
            shortptr    = (unsigned short *)&cp->pdtopm.signal[2];

            /* read the header information */
            bytes_read = netread(fd,&cp->pdtopm.header,(int)HDRLEN);
            if (bytes_read!=HDRLEN) {
                perror("PM:: read length (GGCS:read_protocol):");
#ifdef GO_AWAY
                go_away();
#endif
            }
            gc_channel  = cp->pdtopm.header.length;
            gc_id       = cp->pdtopm.header.client;
#ifdef TRACE
fprintf(stderr,"PM:: GC:0x%x    channel:%d\n",gc_id,gc_channel,channel);
fprintf(stderr,"PM:: port number was :%d\n",*shortptr);
#endif

#ifdef BYTES
shmem->channel_bytes[ gc_channel ] += sig_bytes + bytes_read;
shmem->distributor_bytes[ station ] += sig_bytes + bytes_read;
#endif
```

```
#ifdef TBYTES
fprintf( stderr, "PM:: - %d ggcs\n", bytes_read );
fprintf( stderr, "PM:: ...channel %d, station %d\n", gc_channel, station
);
#endif
            get_graphics_context((XID)gc_id,gc_channel,*shortptr);
            break;
/*************************************************************/

/*************************************************************/
/*
 * Handle a pass through request to send graphics context
 * information back from source to destination channel.
 */
        case GCS:
#ifdef TRACE
fprintf(stderr,"PM:: HEY!!!!!!!! Got a GCS Request from a Distributor\n");
f p r i n t f ( s t d e r r , " P M : :      0 x % x
0x%x\n",cp->pdtopm.signal[2],cp->pdtopm.signal[3]);
#endif
            send_graphics_context(fd,(short *)&cp->pdtopm.signal[2]);
            break;
/*************************************************************/

/*************************************************************/
/*
 * Handle a pass through request to send window attributes
 * information back from source to destination channel.
 */
        case WATS:
#ifdef TRACE
fprintf(stderr,"PM:: HEY!!!!!!! Got a WATS Request from a Distributor\n");
#endif
            send_window_attributes(fd,(short *)&cp->pdtopm.signal[2]);
            break;
/*************************************************************/


/*************************************************************/
/*
 * Otherwise it is an X protocol packet so read the header information
which
 * includes length of X packet and client number.
 */
        case X_DATA:

#ifdef DATA
fprintf(stderr,"PM:: HEY!!!!! Got an X_DATA request\n");
#endif

            bytes_read = netread(fd,&cp->pdtopm.header,(int)HDRLEN);
            if (bytes_read!=HDRLEN) {
```

```
                perror("PM:: read length (X_DATA:read_protocol):");
#ifdef GO_AWAY
                go_away();
#endif
            } /* end if bytes != hdrlen */

#ifdef INTENSE
fprintf(stderr,"PM:: just read %d of header.\n",bytes_read);
#endif

#ifdef BYTES
sig_bytes += bytes_read;
#endif
#ifdef TBYTES
fprintf( stderr, "- %d xdata hdr (%d)\n", bytes_read,
cp->pdtopm.header.length );
#endif

/*
 * Now determine which channel based on which client
 */
            *channel    =
client_to_channel(cp->pdtopm.header.client,station);

            /* We've read the header, now retrieve length from it */
            packet_length           = cp->pdtopm.header.length;

#ifdef INTENSE
f p r i n t f ( s t d e r r , " P M : :    c h a n n e l : % d
packet_length:%d\n",*channel,packet_length);
#endif

/*
 * Retrieve the channel number from the second signal byte.
 * WARNING!!!!! Below means that number of channels must be less than
 * 256!.
 */
            cp->pdtopm.signal[1]    = *(unsigned char *)channel;

/*
 * Since we are now reading from a blocking file descriptor, lets check
 * once, while we have the chance, to see if we need to go away.
 */
            if (shmem->sm_status==PM_DIE) {
                fprintf(stderr,"PM::Requested to exit.\n");
                go_away();
            }

#ifdef INTENSE
fprintf(stderr,"PM:: reading data now....\n");
#endif
```

```
/*
 * Read from the network file descriptor to get rest of package
 */
            bytes_read = netread(fd,cp->pdtopm.buffer,packet_length);

#ifdef BYTES
shmem->channel_bytes[ *channel ] += sig_bytes + bytes_read;
shmem->distributor_bytes[ station ] += sig_bytes + bytes_read;
#endif

#ifdef TBYTES
fprintf( stderr, "- %d x_data bytes\n", bytes_read );
fprintf( stderr, "PM:: ...channel %d, station %d\n", *channel, station );
#endif
            return(TRUE);
            break;
/*************************************************************/

        default:
            fprintf(stderr,"PM::    Invalid    signal    byte,    unknown
request:0x%x\n",
                        cp->pdtopm.signal[0]);
            break;

    } /* end switch */

    return(FALSE);

} /* end read_protocol */
/*************************************************************************/


/*************************************************************************/
/*
 * This routine routes protocol from sources to destinations based on
 * the channel map in shared memory.
 */
void
route_protocol(channel,cp)
    register int                channel;
    register union COMPAK       *cp;
{
    register int                fd;
    register int                i;
    register int                number_done;
    register int                bytes_written;
    register int                length;
#ifdef BYTES
int station;
#endif

/*
```

```
 * Travel down the list of receivers for this channel.
 */
    i           = 0;
    number_done = 0;
    length      = cp->pdtopm.header.length;

    while(number_done!=shmem->ChanMap[channel].num_receivers) {
        fd  = shmem->dest_fd[channel][i];
        if (fd>=0) {
#ifdef TRACE
fprintf(stderr,"PM:: .....sending something over to receiver:%d channel:%d
fd:%d len:%d\n",
i,channel,fd,length);
#endif

if ( fd <= 1 )
 fprintf(stderr,"%d,",fd);

            bytes_written = netwrite(fd,cp->compak,PAKLEN+length);

            if (bytes_written!=length+PAKLEN)
                perror("PM:: write (route_protocol):");
            number_done++;

#ifdef BYTES
shmem->no_of_packets++;
station = pr_fd_to_station( fd );
shmem->channel_bytes[ channel ] += bytes_written;
shmem->receiver_bytes[ station ] += bytes_written;
#endif

#ifdef TBYTES
fprintf(  stderr,  "PM::  route_protocol  -  %d  protocol  bytes\n",
bytes_written );
fprintf( stderr, "PM:: ...channel %d, station %d\n", channel, station );
#endif
        }
        i++;
    } /* end while */

} /* end route_protocol */
/*************************************************************************/



/*****************************************************/
/*
 * This routine handles a request from a Protocol Receiver
 * to retrieve the initial window attributes for a given
 * window. The request was passed to the Protocol Distributor,
 * who in  turn passed it onto the Protocol Multiplexer.
```

```
 * The Multiplexer will then pass the request on to the
 * source Protocol Receiver.
 * Parameters:
 *          win             - source window XID which the attributes
 *                      are requested for.
 *          source_channel  - this is the channel number which
 *                      win is being broadcast on.
 *          port            - port number of the source's receiver
 */
void
get_window_attributes(win,source_channel,port)
        XID             win;
        int             source_channel;
        unsigned short  port;
{
#ifdef TRACE
fprintf(stderr,"PM:: GOT A GET_WINDOW_ATTRIBUTES request from a PD\n");
fprintf(stderr,"PM:: Get_Window_Attributes  win:0x%x   source_channel:%d
\n",
win,source_channel);
fprintf(stderr,"PM:: Port for that was        :%d\n",port);
#endif
/*
 * Call the routine to request state information
 */
    request_receiver(win,source_channel,(unsigned char)GWATS,port);

} /* end get_window_attributes */
/*****************************************************/


/*****************************************************/
/*
 * This routine handles a request from a Protocol Receiver
 * to retrieve the initial state information of a graphics
 * context. The request was passed to the Protocol Distributor,
 * who in turn passed it onto the Protocol Multiplexer.
 * The Multiplexer will then pass the request on to the
 * source Protocol Receiver.
 * Parameters:
 *          gc              - source gc XID which the attributes
 *                      are requested for.
 *          source_channel  - this is the channel number which
 *                      win associated with the gc is being broadcast on.
 */
void
get_graphics_context(gc,source_channel,port)
        XID         gc;
        int         source_channel;
        short       port;
{
#ifdef TRACE
```

```
fprintf(stderr,"PM:: GOT A GET_GRAPHICS_CONTEXT from a PD\n");
fprintf(stderr,"PM:: sending a GGCS to a receiver gc:0x%x
source_channel:%d port:%d\n",
gc,source_channel,port);
#endif

/*
 * Call a routine to request the state information
 */
    request_receiver(gc,source_channel,(unsigned char)GGCS,port);

} /* end get_graphics_context */
/*****************************************************/

/*****************************************************/
/*
 * This routine sends a request for
 * source Protocol Receiver.
 */
void
request_receiver(xid,channel,type,port)
    register XID            xid;
    register int            channel;
    unsigned char           type;
    unsigned short          port;
{
    register int                    bytes_written;
    register int                    fd;
    register int                    numwrite;
    unsigned char                   signal_bytes[SIGLEN+8];
    short                           *shortptr;
    int                             *intptr;
    XID                             *xidptr;
#ifdef BYTES
int station;
#endif

int i;
#ifdef TRACE
fprintf(stderr,"PM:: request_state_info for xid:0x%x channel:%d type:%d
port:%d\n",
xid,channel,type,port);
#endif

/*
 * Call a routine to retrieve the source channel's fd (Protocol Receiver's
fd)
 */
    fd = channel_to_fd(channel);

#ifdef TRACE
fprintf(stderr,"PM:: fd for source receiver is:%d.\n",fd);
```

```
#endif

/*
 * Send the expose event back to the source Receiver.
 */
    signal_bytes[0]     = type;
    signal_bytes[1]     = 0;

/*
 * Depending on the type of request, either the channel,
 * or the receiver port number goes next.
 */
    shortptr            = (short *)&signal_bytes[2];

    /* Send over the port involved */
    *shortptr           = (short)port;

    /* Send over the xid involved */
    xidptr              = (XID *)&signal_bytes[4];
    *xidptr             = xid;

#ifdef TRACE
fprintf(stderr,"PM:: xid for request_receiver is:0x%x\n",xid);
fprintf(stderr,"PM:: channel is                  :0x%x\n",channel);
#endif

    /* Send over the channel */
    intptr              = (int *)&signal_bytes[8];
    *intptr             = channel;


    switch(type) {
        case GWATS:
            numwrite        = SIGLEN+8;
            break;
        case NOOP:
            numwrite        = SIGLEN;
            break;
        default:
            numwrite        = SIGLEN+4;
    } /* end select */

#ifdef TRACE
if (type==GWATS)
fprintf(stderr,"PM:: GWATS GOING TO A RECEIVER\n");
else if (type==NOOP)
fprintf(stderr,"PM:: NOOP GOING TO A RECEIVER\n");
else if (type==GGCS)
fprintf(stderr,"PM:: GGCS GOING TO A RECEIVER\n");
else if (type==EXPOSE)
fprintf(stderr,"PM:: EXPOSE GOING TO A RECEIVER\n");
#endif
```

```
#ifdef DATA
fprintf(stderr,"DATA BELOW\n");
for (i=0;i<numwrite;i++)
fprintf(stderr,"0x%x ",signal_bytes[i]);
fprintf(stderr,"\nDATA ABOVE\n");
#endif

    bytes_written   = netwrite(fd,signal_bytes,numwrite);
    if (bytes_written!=numwrite) {
        perror("PM:: write (request_receiver):");
#ifdef GO_AWAY
        go_away();
#endif
    }

#ifdef TRACE
fprintf(stderr,"PM:: just wrote the request for information to the
Receiver,%d bytes.\n",
bytes_written);
#endif

#ifdef BYTES
shmem->no_of_packets++;
station = pr_fd_to_station( fd );
shmem->channel_bytes[ channel ] += bytes_written;
shmem->receiver_bytes[ station ] += bytes_written;
#endif

#ifdef TBYTES
fprintf( stderr, "PM:: request_receiver - %d request bytes\n",
bytes_written );
fprintf( stderr, "PM:: ...channel %d, station %d\n", channel, station );
#endif
} /* end request_receiver */
/*******************************************************/


/*****************************************************************/
/*
 * This routine reads graphics context state information
 * from a Protocol Distributor and routes it back to the
 * appropriate Protocol Receiver.
 */
void
send_graphics_context(source_fd,port)
    register    int         source_fd;
    register    short       *port;
{
    register    int         dest_fd;
    register    int         bytes;
    register    XGCValues   *ptr;
```

```
    XGCValues               values;
    unsigned char           signal_bytes[SIGLEN];
    XID                     window;
#ifdef BYTES
int channel;
int station;
#endif

    /* Initialize */
    ptr             = &values;

#ifdef TRACE
fprintf(stderr,"PM:: send_graphics_context for port:%d\n",*port);
#endif
/*
 * First find the receiver's file descriptor, based on its port
 * number.
 */
    dest_fd = port_to_fd(*port);

#ifdef TRACE
fprintf(stderr,"PM:: just got the fd for that receiver:%d\n",dest_fd);
fprintf(stderr,"PM:: source fd was                    :%d\n",source_fd);
#endif
/*
 * Now read in the state information from the source Distributor
 */
    bytes   = netread(source_fd,(unsigned char *)ptr,sizeof(XGCValues));
    if (bytes != sizeof(XGCValues) ) {
        perror("PM:: read (send_graphics_context/XGCValues):");
#ifdef GO_AWAY
        go_away();
#endif
    }

#ifdef BYTES
sig_bytes += bytes;
#endif

#ifdef TRACE
fprintf(stderr,"PM:: just read XGCValues bytes:%d\n",bytes);
#endif
/*
 * Now read the window associated with the graphics context
 */
    bytes   = netread(source_fd,&window,sizeof(window));
    if (bytes != sizeof(window) ) {
        perror("PM:: read (send_graphics_context/window):");
#ifdef GO_AWAY
        go_away();
#endif
    }
```

```
#ifdef BYTES
station = pd_fd_to_station( source_fd );
channel = pd_fd_to_channel( source_fd );
shmem->channel_bytes[ channel ] += sig_bytes + bytes;
shmem->distributor_bytes[ station ] += sig_bytes + bytes;
#endif

#ifdef TBYTES
fprintf( stderr, "PM:: send_graphics_context - %d read sgc bytes\n",
sig_bytes + bytes );
fprintf( stderr, "PM:: ...channel %d, station %d\n", channel, station );
#endif

#ifdef TRACE
fprintf(stderr,"PM:: just read a window id:0x%x\n",window);
#endif

/*
 * Now write the signal bytes out
 */
    signal_bytes[0]     = GCS;
    signal_bytes[1]     = 0;
    signal_bytes[2]     = 0;
    signal_bytes[3]     = 0;

    bytes   = netwrite(dest_fd,signal_bytes,SIGLEN);
    if (bytes != SIGLEN) {
        perror("PM:: write (send_graphics_context/signal):");
#ifdef GO_AWAY
        go_away();
#endif
    }

#ifdef BYTES
sig_bytes = bytes;
#endif

#ifdef TRACE
fprintf(stderr,"PM:: just wrote the GCS signal to pr\n");
#endif

/*
 * Now write that information back out to the appropriate Receiver
 */

    bytes   = netwrite(dest_fd,ptr,sizeof(XGCValues));
    if (bytes != sizeof(XGCValues) ) {
        perror("PM:: write (send_graphics_context/XGCValues):");
#ifdef GO_AWAY
        go_away();
#endif
```

```
    }

#ifdef BYTES
sig_bytes += bytes;
#endif

#ifdef TRACE
fprintf(stderr,"PM:: just wrote XGCValues bytes:%d\n",bytes);
#endif
/*
 * Now write the window associated with the graphics context
 */
    bytes   = netwrite(dest_fd,&window,sizeof(window));
    if (bytes != sizeof(window)) {
        perror("PM:: write (send_graphics_context/window):");
#ifdef GO_AWAY
        go_away();
#endif
    }

#ifdef BYTES
shmem->no_of_packets++;
station = pr_fd_to_station( dest_fd );
shmem->channel_bytes[ channel ] += sig_bytes + bytes;
shmem->receiver_bytes[ station ] += sig_bytes + bytes;
#endif

#ifdef TBYTES
fprintf( stderr, "PM:: send_graphics_context - %d write sgc bytes\n",
sig_bytes + bytes );
fprintf( stderr, "PM:: ...channel %d, station %d\n", channel, station );
#endif

#ifdef TRACE
fprintf(stderr,"PM:: just wrote window bytes:%d\n",bytes);
#endif

} /* end send_graphics_context */
/*****************************************************************/

/*****************************************************************/
/*
 * This routine reads window attributes state information
 * from a Protocol Distributor and routes it back to the
 * appropriate Protocol Receiver.
 */
void
send_window_attributes(source_fd,port)
    register    int     source_fd;
    register    short   *port;
{
    register    int                 dest_fd;
```

```
    register    int                 bytes;
    unsigned char                   pixels[4];
    register    XWindowAttributes   *ptr;
    XWindowAttributes               values;
    unsigned char                   signal_bytes[SIGLEN];
#ifdef BYTES
int channel;
int station;
#endif

    /* Initialize */
    ptr             = &values;

#ifdef TRACE
fprintf(stderr,"PM:: send_window_attributes for port:%d\n",*port);
#endif
/*
 * First find the receiver's file descriptor, based on its port
 * number.
 */
    dest_fd = port_to_fd(*port);

#ifdef TRACE
fprintf(stderr,"PM:: just got the fd for that receiver:%d\n",dest_fd);
fprintf(stderr,"PM:: source fd was                    :%d\n",source_fd);
#endif
/*
 * Now read in the state information from the source Distributor
 */
    bytes   = netread(source_fd,pixels,8);
    if (bytes != 8) {
        perror("PM:: read (send_window_attributes/pixels");
#ifdef GO_AWAY
        go_away();
#endif
    }

#ifdef BYTES
sig_bytes += bytes;
#endif

#ifdef TRACE
fprintf(stderr,"PM:: just read 8 bytes of pixel info\n");
#endif

    bytes   =   netread(source_fd,(unsigned   char
*)ptr,sizeof(XWindowAttributes));
    if (bytes != sizeof(XWindowAttributes) ) {
        perror("PM:: read (send_window_attributes/XWindowAttributes):");
#ifdef GO_AWAY
        go_away();
#endif
```

```
    }

#ifdef BYTES
station = pd_fd_to_station( source_fd );
channel = pd_fd_to_channel( source_fd );
shmem->channel_bytes[ channel ] += sig_bytes + bytes;
shmem->distributor_bytes[ station ] += sig_bytes + bytes;
#endif

#ifdef TBYTES
fprintf( stderr, "PM:: send_window_attributes - %d read gwat bytes\n",
sig_bytes + bytes );
fprintf( stderr, "PM:: ...channel %d, station %d\n", channel, station );
#endif

#ifdef TRACE
fprintf(stderr,"PM:: just read XWindowAttributes bytes:%d\n",bytes);
#endif

/*
 * Now write the signal bytes out
 */
    signal_bytes[0]     = WATS;
    signal_bytes[1]     = 0;
    signal_bytes[2]     = 0;
    signal_bytes[3]     = 0;

    bytes   = netwrite(dest_fd,signal_bytes,SIGLEN);
    if (bytes != SIGLEN) {
        perror("PM:: write (send_window_attributes/signal):");
#ifdef GO_AWAY
        go_away();
#endif
    }

#ifdef BYTES
sig_bytes = bytes;
#endif

#ifdef TRACE
fprintf(stderr,"PM:: just wrote the WATS signal to pr\n");
#endif
/*
 * Now write the pixel information
 */

    bytes   = netwrite(dest_fd,pixels,8);
    if (bytes != 8) {
        perror("PM:: write (send_window_attributes/pixels):");
#ifdef GO_AWAY
        go_away();
#endif
```

```
    }

#ifdef BYTES
sig_bytes += bytes;
#endif

#ifdef TRACE
fprintf(stderr,"PM:: just wrote 8 bytes of pixel info\n");
#endif

/*
 * Now write that information back out to the appropriate Receiver
 */

    bytes   = netwrite(dest_fd,ptr,sizeof(XWindowAttributes));
    if (bytes != sizeof(XWindowAttributes) ) {
        perror("PM:: write (send_window_attributes/XWindowAttributes):");
#ifdef GO_AWAY
        go_away();
#endif
    }

#ifdef BYTES
shmem->no_of_packets++;
station = pr_fd_to_station( dest_fd );
shmem->channel_bytes[ channel ] += sig_bytes + bytes;
shmem->receiver_bytes[ station ] += sig_bytes + bytes;
#endif

#ifdef TBYTES
fprintf( stderr, "PM:: send_window_attributes - %d write swat bytes\n",
sig_bytes + bytes );
fprintf( stderr, "PM:: ...channel %d, station %d\n", channel, station );
#endif

#ifdef TRACE
fprintf(stderr,"PM:: just wrote XWindowAttributes bytes:%d\n",bytes);
#endif

} /* end send_window_attributes */
/*****************************************************************/


/*****************************************************************/
/*
 * This routine accepts a connection to a Protocol Distributor.
 */
void
accept_distributor(index)
    register int                    index;
{
    register int                    fd;
```

```
    static struct   sockaddr_in     sinhim = {AF_INET};
    int                             sinlen;
    sinlen  = sizeof(sinhim);

#ifdef MANAGE
fprintf(stderr,"PM:: accept_distributor, waiting on an accept...\n");
#endif

/*
 * Accept a connection from the Protocol Distributor
 */
    fd      = accept(protofd,&sinhim,&sinlen);
    set_no_block(fd);
    if (fd<0) {
        perror("PM::accept (accept_distributor):");
#ifdef GO_AWAY
        go_away();
#endif
    }

/*
 * Place the value of the file descriptor into shared memory.
 */
    shmem->Stations[index].pd_fd    = fd;

#ifdef MANAGE
fprintf(stderr,"PM:: Just accepted connection to <%s> at:%d\n",
shmem->Stations[index].hostname,shmem->Stations[index].pd_fd);
#endif

}/* end accept_distributor */
/*********************************************************************/

/*********************************************************************/
/*
 * This routine checks all the file descriptors for every Protoco
Distributor
 * which has registerred with us.
 */
void
check_for_input()
{
    register int                    station;
    register union COMPAK           *cp_reg;
    register struct CDM_SHMEMORY    *memptr;
    register int                    fd;
    register int                    i;
    union COMPAK                    cp;
    int                             channel;
    int                             fdmask;
    int                             nfound;
    int                             numcheck;
```

```
    struct timeval                  timeout;

    /* Initializations */
    cp_reg      = &cp;
    memptr      = shmem;

#ifdef SELECT
    fdmask      = 0;
    numcheck    = 0;
    timeout.tv_sec  = 1;
    timeout.tv_usec = 0;

/*
 * Go through the list of valid fds and build fd select mask
 */
    for (station=0;station<MAX_STATIONS;station++) {

/*
 * If the station is an active distributor, then set
 * the file descriptor mask.
 */
        if (memptr->Stations[station].pd_fd>0) {
            fdmask =  fdmask | (1<<memptr->Stations[station].pd_fd);
            if (memptr->Stations[station].pd_fd>numcheck)
                numcheck = memptr->Stations[station].pd_fd;
        }

    } /* end for */

/*
 * WARNING!!!! Use of the select system call means that
 * we can only check file descriptors which are < 32 in value.
 * Now use the select call to check for input
 */


    nfound  = select(numcheck+1,&fdmask,NULL,NULL,&timeout);

/*
 * If there is protocol on the file descriptor, then
 * determine which channel it is to be broadcast upon.
 */
    if (nfound>0) {
        numcheck    = 0;
        for (i=0;i<MAX_STATIONS&&numcheck<nfound;i++) {
            if (fdmask&(1<<i)) {
                numcheck++;
                if (read_protocol(i,cp_reg,&channel,pd_fd_to_station(i))
) {
                    route_protocol(channel,cp_reg);
                }
            } /* end if */
```

```
        } /* end for */
    } /* end if */
    else if (nfound==0)
        return;
    else
        perror("PM:: select for input:");
#else

/*
 * Go through the list of valid fds and check for input.
 */
    for (station=0;station<MAX_STATIONS;station++) {

/*
 * If the station is an active distributor, then get
 * the file descriptor and check for incoming protocol.
 */
        if (memptr->Stations[station].pd_fd>0) {
            fd  = memptr->Stations[station].pd_fd;

/*
 * If there is protocol on the file descriptor, then
 * determine which channel it is to be broadcast upon.
 */
            if (read_protocol(fd,cp_reg,&channel,station))
{
#ifdef DATA
fprintf(stderr,"PM:: routing protocol station:%d channel:%d\n",
station,channel);
#endif
                route_protocol(channel,cp_reg);
}

        } /* end if there is a distributor fd valid */

    } /* end for station */

#endif

} /* end check_for_input */
/*****************************************************************/

/*****************************************************************/
/*
 * This routine accepts a connection to a receiver.
 */
void
accept_receiver(index,pr_port)
    int             index;
    unsigned short  pr_port;
{
    int             fd;
```

```
#ifdef MANAGE
fprintf(stderr,"PM:: accept_receiver called index:%d port:%d\n",
index,pr_port);
#endif

/*
 * Now actually connect to that receiver
 */
    fd  = connect_to(pr_port,shmem->Stations[index].hostname);

#ifdef MANAGE
fprintf(stderr,"PM:: connected to returned an fd of :%d\n",fd);
#endif
/*
 * Place in shared memory
 */
    shmem->Stations[index].pr_fd    = fd;

} /* end accept_receiver */
/*************************************************************************/


/*************************************************************************/
/*
 * This routine notifies a particular receiver that a given client's
 * protocol will no longer be received and it may safely close the
 * X connection for it.
 */
void
final_shutdown(fd,client)
    int         fd;
    int         client;
{
    short           *shortptr;
    short           shortval;
    unsigned char   signal_bytes[4];
    int             bytes_written;

/*
 * Set up the communications package
 */
    signal_bytes[0] = SHUTCOMP;
    signal_bytes[1] = 0;
    shortval        = client;
    shortptr        = (short *)&signal_bytes[2];
    *shortptr       = shortval;

#ifdef MANAGE
fprintf(stderr,"PM:: final_shutdown fd:%d client:%d\n",fd,client);
#endif

/*
```

```
 * Write it out
 */
    bytes_written   = netwrite(fd,signal_bytes,sizeof(signal_bytes));
    if (bytes_written!=sizeof(signal_bytes)) {
        perror("PM:: write (final_shutdown):");
#ifdef GO_AWAY
        go_away();
#endif
    }

} /* end final_shutdown */
/*************************************************************************/
```

```
#define MAX_TRIES       150
/*
 * File         : pmutil.c
 * Author       : P. Fitzgerald - SwRI
 * Date         : 10/3/89
 * This file contains utility routines for the Protocol Multiplexer.
 */
#include <stdio.h>
#include <rpc/rpc.h>
#include <utmp.h>
#include <sys/types.h>
#include <rpcsvc/rusers.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <fcntl.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <signal.h>
#include <errno.h>
#include <X11/X.h>
#define NEED_REPLIES
#define NEED_EVENTS
#include <X11/Xproto.h>
#include <X11/Xlib.h>
#include "../includes/ds_manage.h"
#include "../includes/smtypes.h"
#include "../includes/smdef.h"
#include "../includes/dist.h"

/* DEFINES */
#define     LISTEN_BACKLOG  140


/* EXTERNAL ROUTINES */

/* GLOBAL FUNCTIONS */
int             get_distributor_index();
void            attach_shared_memory();
int             channel_to_fd();
int             channel_to_client();
int             pr_fd_to_station();
int             pd_fd_to_station();
int             pd_fd_to_channel();
int             recv_fd_to_station();
void            set_alarm();
void            clear_alarm();
void            get_graphics_context();
void            accept_source();
void            close_source();
void            connect_receiver();
void            close_receiver();
```

```
int             timeout();

/* EXTERNAL VARIABLES */
extern  int             cdm_write_fd;
extern  int             cdm_read_fd;
extern  int             protofd;
extern  unsigned short  port_number;
extern  int             my_fd;
extern  char            hostname[HOSTNAMLEN];
extern  struct ChanMap  *ChanMap;
extern  struct CDM_SHMEMORY *shmem;
extern  int                 shmid;
extern  int                 place;

/*****************************************************************/
/*
 * Initialization routine
 */
void
pminit()
{

/*
 * Acquire the host name where we reside.
 */
    if ( gethostname(hostname,sizeof(hostname)) <0) {
        perror("PM::gethostname:");
        go_away();
    }

/*
 * Set up to catch timer-timeout signals
 */
    signal(SIGALRM, timeout);

/*
 * Now attach to the shared memory segment between the CDM and the PM
 */
    attach_shared_memory();

/*
 * Write a byte to let CDM know we are here.
 */
    rendezvous_with_cdm();

/*
 * Now send the port number.
 */
    acquire_port_number();

/*
 * Let everyone know we are here
```

```
 */
    fprintf(stderr,"SwRI Protocol Multiplexer starting...\n");

} /* end pminit */
/*****************************************************************************/


/*****************************************************************************/
/*
 * This routine writes a byte to the pipe to let the CDM know we are
 * alive.
 */
rendezvous_with_cdm()
{
    register int    bytes_written;
    unsigned char   byte;

    bytes_written = write(cdm_write_fd,&byte,1);
    if (bytes_written != 1) {
        perror("PM::write (pm):");
#ifdef GO_AWAY
        go_away();
#endif
    }
} /* end rendezvous_with_cdm */
/*****************************************************************************/


/*****************************************************************************/
/*
 * This routine acquires a socket and port number, and then sends that
 * number back to the CDM.
 */
acquire_port_number()
{
    int                 bytes_written;
    static struct sockaddr_in   sinhim = { AF_INET };
    static struct sockaddr_in   sinme  = { AF_INET };
    int                 sinlen;

/*
 * Create a socket (Internet-stream)
 */
    if ((protofd = socket(AF_INET, SOCK_STREAM,0))<0) {
        perror("PM::acquire_port_number (socket):");
        go_away();
    }

/*
 * Bind to my address.
 */
    if (bind(protofd,&sinme,sizeof(sinme))<0) {
```

```
        perror("PM::acquire_port_number (bind):");
        go_away();
    }
    sinlen  = sizeof(sinme);
    if (getsockname(protofd, &sinme, &sinlen)<0) {
        perror("PM::acquire_port_number (getsockname):");
        go_away();
    }
    port_number = ntohs(sinme.sin_port);

/*
 * Place port number in shared memory.
 */
    shmem->pm_port  = port_number;

/*
 * Listen and accept to acquire a file descriptor.
 */
    sinlen  = sizeof(sinhim);
    if (listen(protofd,(int)LISTEN_BACKLOG)<0) {
        perror("PM::acquire_port_number (listen):");
        go_away();
    }

/*
 * Now write the port number to the CDM.
 */
    bytes_written=write(cdm_write_fd,&port_number,sizeof(port_number));
    if (bytes_written != sizeof(port_number) ) {
        perror("PM::write (pm):");
#ifdef GO_AWAY
        go_away();
#endif
    }

    my_fd = accept(protofd, &sinhim, &sinlen);
    if (my_fd<0) {
        perror("PM::acquire_port_number (accept):");
        go_away();
    }
    set_no_block(my_fd);

} /* end acquire_port_number */
/*****************************************************************************/


/*****************************************************************************/
/*
 * This routine simply sets O_NDELAY attribute on file descriptor.
 */
set_no_block(fd)
    register int    fd;
```

```
{
    register int    flags;

    if ((flags=fcntl(fd,F_GETFL,0))==-1) {
        perror("PM::fcntl (set_no_block-F_GETFL):");
#ifdef GO_AWAY
        go_away();
#endif
    }
    flags |= (FNDELAY);
    if (fcntl(fd,F_SETFL,flags)<0) {
        perror("PM::fcntl (set_no_block-F_SETFL):");
#ifdef GO_AWAY
        go_away();
#endif
    }

} /* end set_no_block */
/*****************************************************************************/

/*****************************************************************************/
/*
 * This routine simply clears O_NDELAY attribute on file descriptor.
 */
set_block(fd)
    register int    fd;
{
    register int    flags;

    if ((flags=fcntl(fd,F_GETFL,0))==-1) {
        perror("PM::fcntl (set_block-F_GETFL):");
#ifdef GO_AWAY
        go_away();
#endif
    }
    flags &= (~FNDELAY);
    if (fcntl(fd,F_SETFL,flags)<0) {
        perror("PM::fcntl (set_block-F_SETFL):");
#ifdef GO_AWAY
        go_away();
#endif
    }

} /* end set_block */
/*****************************************************************************/

/*****************************************************************************/
/*
 * This routine attaches to the shared memory area between the Central
 * Distribution Manager and the Protocol Multiplexer.
 */
void
```

```
attach_shared_memory()
{
#define RWMODE      0666

    /* attach to shared memory */
    shmid   = shmget( (int)CDM_KEY,
                sizeof(struct CDM_SHMEMORY),RWMODE);
    if (shmid<0) {
        perror("PM::shmget (attach_shared_memory):");
        go_away();
    }
    shmem   = (struct CDM_SHMEMORY *)shmat(shmid,0,0);
    if (shmem==NULL) {
        perror("PM::shmat (attach_shared_memory):");
        go_away();
    }

/*
 * Set the pointer to the Channel Map in shared memory.
 */
    ChanMap = shmem->ChanMap;

} /* end attach_shared_memory */
/*****************************************************************************/


/*****************************************************************************/
/*
 * This routine accepts a connection from a source (Protocol Distributor).
 * When the connection is accepted, the file descriptor is set in the
 * appropriate slot in the channel map.
 */
void
accept_source(channel)
    register int        channel;
{
    register struct CDM_SHMEMORY    *memptr;
    register int                    index;

#ifdef MANAGE
fprintf(stderr,"PM:: accept_source channel:%d\n",channel);
#endif

    /* Initialization */
    memptr  = shmem;

#ifdef MANAGE
fprintf(stderr,"PM:: accept_source for channel:%d\n",channel);
#endif

/*
 * Search through the base list until we
```

```
 * find the file descriptor we want.
 */
    i           n           d           e           x           =
get_distributor_index(memptr->ChanMap[channel].source_hostname);

#ifdef MANAGE
fprintf(stderr,"PM:: Distributor:<%s> index was:%d\n",
memptr->source_name[channel],index);
#endif

/* Set the value of that file descriptor */
    memptr->source_fd[channel]                  =
memptr->Stations[index].pd_fd;

memptr->Stations[index].dist_channel[memptr->Stations[index].num_channels]

                                                = channel;
    memptr->Stations[index].num_channels++;

#ifdef MANAGE
fprintf(stderr,"PM:: channel:%d\n",channel);
f  p  r  i  n  t  f  (  s  t  d  e  r  r  ,  "  P  M  :  :
client:0x%x\n",memptr->Stations[index].dist_client[channel]);
fprintf(stderr,"PM:: accept_source completed for channel and marked as
distributing.\n");
fprintf(stderr,"PM::   number   of   channels   for   this
distributor:%d\n",memptr->Stations[index].num_channels);
#endif

} /* end accept_source */
/*************************************************************************/


/*************************************************************************/
/*
 *  This  routine  closes  a  connection  to  a  give  source  (Protocol
Distributor).
 */
void
close_source(channel)
    register int                    channel;
{
    register int                    i;
    register int                    index;
    register struct CDM_SHMEMORY    *memptr;

#ifdef MANAGE
fprintf(stderr,"PM:: close_source channel:%d\n",channel);
#endif

    memptr  = shmem;
#ifdef MANAGE
```

```
fprintf(stderr,"PM::closing source channel:%d\n",channel);
fprintf(stderr,"PM::    source    name    was
:<%s>\n",memptr->source_name[channel]);
#endif
/*
 * Set the source file descriptor to invalid.
 */
    memptr->source_fd[channel]  = -1;

/*
 * Get an index of the entry in the Distributors table and
 * mark the Distributing flag to FALSE
 */
    i             n             d             e             x             -
get_distributor_index(memptr->ChanMap[channel].source_hostname);
    sprintf(memptr->ChanMap[channel].source_hostname,"");

    remove_channel(channel,index);

#ifdef MANAGE
fprintf(stderr,"PM:: distributor_index:%d\n",index);
#endif


/*
 * Close down all the receiving receivers.
 */
    for (i=0;i<memptr->ChanMap[channel].num_receivers;i++) {
#ifdef MANAGE
fprintf(stderr,"PM:: closing down receiver:%d\n",i);
#endif
        memptr->ChanMap[channel].recv_ports[i]  = 0;
    }
    memptr->ChanMap[channel].num_receivers  = 0;

} /* end close_source */
/*****************************************************************/


/*****************************************************************/
/*
 * This routine connects to a receiver (Protocol Receiver).
 * It does this by determining, from the channel number, which
 * port number is set that does not have a corresponding file
 * descriptor open. It then connects to the receiver and set the
 * file descriptor accordingly.
 */
void
connect_receiver(channel)
    register int                channel;
{
    register int                position;
```

```
    register struct CDM_SHMEMORY    *memptr;

    memptr  = shmem;

#ifdef MANAGE
fprintf(stderr,"PM:: connect_receiver channel:%d\n",channel);
fprintf(stderr,"PM:: Connect a new receiver.\n");
fprintf(stderr,"PM::  distributor  index  (memptr->station_index)  from
shmem:%d\n",
memptr->station_index);
fprintf(stderr,"PM:: MAX_CHANNELS:%d\n",MAX_CHANNELS);
#endif

/*
 * Search the list of ports to see which one does not have a valid
 * file descriptor next to it.
 */
    for (position=0;position<MAX_CHANNELS;position++)
#ifdef MANAGE
{
fprintf(stderr,"PM:: searching for empty slot channel:%d port:%d name:<%s>
position:%d.\n",
channel,memptr->ChanMap[channel].recv_ports[position],memptr->ChanMap[c
hannel].source_hostname,position);
#endif
        if (memptr->ChanMap[channel].recv_ports[position]!=0 &&
            memptr->dest_fd[channel][position]<0)
            break;
#ifdef MANAGE
}
#endif

/*
 * If we got here and found no file descriptor, then we have a problem.
 */
    if (position>=MAX_CHANNELS) {
        fprintf(stderr,"PM:: Found no port number (connect_receiver).\n");
        return;
    }

#ifdef MANAGE
fprintf(stderr,"PM::  Going  to  connect  to
port:%d\n",memptr->ChanMap[channel].recv_ports[position]);
fprintf(stderr,"PM::  Port  name
:<%s>\n",memptr->ChanMap[channel].source_hostname);
fprintf(stderr,"PM:: Position                :%d\n",position);
fprintf(stderr,"PM::  Distributor  index
:%d\n",memptr->station_index);
#endif

/*
 * Set the file descriptor for the appropriate receiver
```

```
 */
    memptr->dest_fd[channel][position]  =
memptr->Stations[memptr->station_index].pr_fd;

/* ifdef MANAGE */
fprintf(stderr,"PM::    file    descriptor    received    from
connect:%d\n",memptr->dest_fd[channel][position]);
/* endif */
fprintf( stderr,"PM:: channel: %d, pos: %d\n",channel, position );

} /* end connect_receiver */
/*********************************************************************/


/*********************************************************************/
/*
 * This routine closes a connection to a Protocol Receiver.
 */
void
close_receiver(channel)
    register int                    channel;
{
    register int                    i;
    register int                    number_checked;
    register struct CDM_SHMEMORY    *memptr;

    memptr  = shmem;

#ifdef MANAGE
fprintf(stderr,"PM:: close_receiver channel:%d\n",channel);
#endif

#ifdef MANAGE
fprintf(stderr,"PM:: close_receiver for channel:%d\n",channel);
fprintf(stderr,"PM:: searching through channel map now....\n");
#endif

/*
 * Go through the port/fd list and determine which valid fd
 * does not have a valid port number.
 */
    number_checked  = 0;
    i               = 0;
    while (number_checked<=memptr->ChanMap[channel].num_receivers) {
#ifdef MANAGE
fprintf(stderr,"PM:: fd:%d  port:%d\n",
memptr->dest_fd[channel][i],memptr->ChanMap[channel].recv_ports[i]);
#endif
        if (memptr->dest_fd[channel][i]>0  &&
            memptr->ChanMap[channel].recv_ports[i]==0) {
#ifdef MANAGE
fprintf(stderr,"PM:: found a receiver to close.\n");
```

```
f p r i n t f ( s t d e r r , " P M : :
ChanMap[channel].client_id:%d\n",ChanMap[channel].client_id);
#endif
#ifdef SLOW
sleep(1);
#endif

final_shutdown(memptr->dest_fd[channel][i],ChanMap[channel].client_id);
            memptr->ChanMap[channel].recv_ports[i]  = 0;
            memptr->dest_fd[channel][i]             = -1;
#ifdef MANAGE
fprintf(stderr,"PM::    just    set    memptr->dest_fd[%d][%d]    to
-1.\n",channel,i);
#endif
            break;
        }
        number_checked++;
#ifdef MANAGE
fprintf(stderr,"PM:: just checked receiver:%d\n",number_checked);
#endif
    } /* end while */

} /* end close_receiver */
/*************************************************************************/


/*************************************************************************/
/*
 * This is a timeout routine to say that we got hung up waiting on
 * someting to happen.
 */
int
timeout()
{

    fprintf(stderr,"PM:: Timeout performing a read or write.\n");
    if (shmem->sm_status==DIE)
        go_away();
    set_alarm(500);

} /* end timeout */
/*************************************************************************/

/*****************************************************/
/*
 * This function is called to connect to a particular
 * port number on a remote machine.
 * It returns a file descriptor.
 */
int
connect_to(port,remote_name)
    unsigned short      port;
```

```
    char                *remote_name;
{
    static struct sockaddr_in   sinhim = { AF_INET };
    register struct hostent     *hp;
    int                         fd;

#ifdef MANAGE
fprintf(stderr,"PM:: Connect to port:%d  name:<%s>.\n",port,remote_name);
#endif

/*
 * Now get hostname, address, and connect to the Multiplexer.
 */
    hp  = gethostbyname(remote_name);
    if (!hp) {
        fprintf(stderr,"PM::Host '%s' not found\n",remote_name);
        go_away();
    }

#ifdef MANAGE
fprintf(stderr,"PM::port number->%d\n",port);
#endif
    bcopy(hp->h_addr, &sinhim.sin_addr, sizeof(sinhim.sin_addr));
    sinhim.sin_port = htons(port);
    fd  = socket(AF_INET,SOCK_STREAM,0);
    if (fd<0) {
        perror("PM::socket (connect_to):");
        return(-1);
    }
#ifdef MANAGE
fprintf(stderr,"PM::socket created.\n");
#endif
    if (connect(fd,&sinhim,sizeof(sinhim))<0) {
        perror("PM::connect (connect_to):");
        return(-1);
    }

#ifdef MANAGE
fprintf(stderr,"PM::Socket and connect to PM ok.\n");
#endif

    return(fd);
} /* end connect_to */
/*****************************************************/

/*****************************************************/
/*
 * This routine searches the Distributor list and returns the
 * the file desriptor for the source for the given
 * channel. Note that this is a writeable file descriptor,
 * meaning that it is the one used by the Protocol Receiver
 * which is hosted at the source workstation.
```

```
 */
int
channel_to_fd(channel)
    register    int channel;
{
    register        int             station;
    register struct CDM_SHMEMORY    *memptr;

    /* initialize */
    memptr  = shmem;

#ifdef TRACE
fprintf(stderr,"PM:: channel_to_fd for channel:%d\n",channel);
fprintf(stderr,"PM:: channel_to_fd source name:<%s>\n",
memptr->source_name[channel]);
#endif

/*
 * Go through the list of distributors and look for a match of
 * source names.
 */

#ifdef TRACE
fprintf(stderr,"PM:: going through channel map looking for station\n");
#endif
    for (station=0;station<MAX_STATIONS;station++) {
#ifdef TRACE
fprintf(stderr,"PM:: Checking source_name:<%s> against portname:<%s>\n",
memptr->source_name[channel],memptr->Stations[station].hostname);
#endif
        if ( strcmp(memptr->ChanMap[channel].source_hostname,
                    memptr->Stations[station].hostname)==0)
            break;
    } /* end for */


#ifdef TRACE
fprintf(stderr,"PM:: station %d is <%s>\n",station,
memptr->Stations[station].hostname);
fprintf(stderr,"PM:: returning source fd:%d\n",
memptr->Stations[station].pr_fd);
#endif
    if (memptr->Stations[station].pr_fd<0) {
        fprintf(stderr,"PM:: Unable to find receiver fd in channel_to_fd
for station:%d\n",
                station);
#ifdef GO_AWAY
        go_away();
#endif
    }
    return(memptr->Stations[station].pr_fd);
```

```
} /* end channel_to_fd */
/*****************************************************/

/*****************************************************/
/*
 * This routine searches the channel map to retrieve
 * a file descriptor. This file descriptor is for the
 * Protocol Receiver associated with the port. This
 * will be used to WRITE state information from the
 * source workstation.
 */
int
port_to_fd(port)
    register        short           port;
{
    register struct CDM_SHMEMORY    *memptr;
    register        int             c;
    register        int             r;

    /* initialize */
    memptr  = shmem;

#ifdef TRACE
fprintf(stderr,"PM:: port_to_fd for port:%d\n",port);
#endif

/*
 * Go through all channels, all receiver ports until we get a match. Then
 * retrieve the appropriate file descriptor.
 */
    for (c=0;c<MAX_CHANNELS;c++) {
        for (r=0;r<MAX_RECEIVERS;r++)
            if (memptr->ChanMap[c].recv_ports[r]==port)
#ifdef TRACE
{
fprintf(stderr,"PM:: port_to_fd returning:%d\n",memptr->dest_fd[c][r]);
#endif
                return(memptr->dest_fd[c][r]);
#ifdef TRACE
}
#endif
    }

/*
 * Else we have failed for some unknown reason. Sigh.
 */
    fprintf(stderr,"PM:: Unable to find matching reciever fd for
port:%d\n",
                port);
#ifdef GO_AWAY
    go_away();
#endif
```

```
    return(-1);

} /* end port_to_fd */
/*****************************************************/


/*****************************************************/
/*
 * This routine searches the list of all registerd Protocol
 * Distributors and returns the index of the entry, based on the
 * name.
 */
int
get_distributor_index(name)
    register char       *name;
{
    register int                    i;
    register struct CDM_SHMEMORY    *memptr;

#ifdef TRACE
fprintf(stderr,"PM:: get_distributor_index called for:<%s>\n",name);
#endif

    memptr  = shmem;
    for (i=0;i<MAX_STATIONS;i++)  {
#ifdef TRACE
fprintf(stderr,"PM:: checking Distributor:<%s> with:<%s>\n",
memptr->Stations[i].hostname,name);
#endif
        if (strncmp(name,memptr->Stations[i].hostname,
                    HOSTNAMLEN)==0) {
#ifdef TRACE
fprintf(stderr,"PM:: Previous one was a match, index:%d\n",i);
#endif
            return(i);
        }
    }

/*
 * Otherwise, we could not find the entry
 */
    fprintf(stderr,"PM::  get_distributor_index  unable  to  find  file
descriptor.\n");
#ifdef GO_AWAY
    go_away();
#endif
    return(-1);

} /* end get_distributor_index */
/*****************************************************/
```

```
/*****************************************************/
/*
 * This routine takes the client number and station
 * number for a given distributor and returns the
 * channel number associated with the client.
 */
int
client_to_channel(client,station)
    register    int client;
    register    int station;
{
    register    int i;

    for (i=0;i<MAX_CHANNELS;i++)  {
        if (shmem->Stations[station].dist_client[i]==client)
            return(shmem->Stations[station].dist_channel[i]);
    } /* end for */

    fprintf(stderr,"PM::  Unable  to  get  channel  from  client:0x%x
station:%d.\n",
            client,station);
#ifdef GO_AWAY
    go_away();
#endif
    return(-1);

} /* end client_to_channel */
/*****************************************************/

/*****************************************************/
/*
 * This routine is the reverse of the one above it. It
 * takes the channel and station number and returns the
 * associated client number.
 */
int
channel_to_client(channel,station)
    register    int     channel;
    register    int     station;
{
    register    int     i;

    for (i=0;i<MAX_CHANNELS;i++)  {
        if (shmem->Stations[station].dist_channel[i]==channel)
            return(shmem->Stations[station].dist_client[i]);
    }
    fprintf(stderr,"PM::  Unable  to  get  client  from  channel:%d
station:%d.\n",
            channel,station);
    return(-1);

} /* end channel_to_client */
```

```
/*****************************************************/

/*****************************************************/
/*
 * This routine takes a station number and a client
 * number and returns the index of the entry in the
 * Distributors table.
 */
int
client_to_index(client,station)
    register    int     client;
    register    int     station;
{
    register    int     i;

    for (i=0;i<MAX_CHANNELS;i++)
        if (shmem->Stations[station].dist_client[i]==client)
            return(i);

    fprintf(stderr,"PM:: Unable to find client:%d in station:%d\n",
            client,station);
    return(-1);

} /* end client_to_index */
/*****************************************************/
/*****************************************************/
/*
 * This routine takes a station number and a channel
 * number and returns the index of the entry in the
 * Distributors table.
 */
int
channel_to_index(channel,station)
    register    int     channel;
    register    int     station;
{
    register    int     i;

    for (i=0;i<MAX_CHANNELS;i++)
        if (shmem->Stations[station].dist_channel[i]==channel)
            return(i);

    fprintf(stderr,"PM:: Unable to find channel:%d in station:%d\n",
            channel,station);
    return(-1);

} /* end channel_to_index */
/*****************************************************/

/*****************************************************/
/*
 * This routine takes a receiver fd and searches the
```

```
 * Station list for the source station.
 */
int
pr_fd_to_station( pr_fd )
    register    int     pr_fd;
{
    register    int                 station;
    register struct CDM_SHMEMORY    *memptr;

    /* initialize */
    memptr = shmem;

    for ( station = 0; station < MAX_STATIONS; station++ ) {
        if ( memptr->Stations[ station ].pr_fd == pr_fd )
            return( station );
    } /* end for */

    fprintf( stderr, "PM:: Unable to find station in pr_fd_to_station for
pr_fd: %d\n",
                pr_fd );
#ifdef GO_AWAY
    go_away();
#endif

    return( -1 );

} /* end pr_fd_to_station */
/*****************************************************/
/*****************************************************/
/*
 * This routine takes a distributor fd and searches the
 * Station list for the source station.
 */
int
pd_fd_to_station( pd_fd )
    register    int     pd_fd;
{
    register    int                 station;
    register struct CDM_SHMEMORY    *memptr;

    /* initialize */
    memptr = shmem;

    for ( station = 0; station < MAX_STATIONS; station++ ) {
        if ( memptr->Stations[ station ].pd_fd == pd_fd )
            return( station );
    } /* end for */

    fprintf( stderr, "PM:: Unable to find station in pd_fd_to_station for
pd_fd: %d\n",
                pd_fd );
#ifdef GO_AWAY
```

```
    go_away();
#endif

    return( -1 );

} /* end pd_fd_to_station */
/*****************************************************/

/*****************************************************/
/*
 * This routine takes a distributor fd and searches the
 * source_fd list for the channel.
 */
int
pd_fd_to_channel( pd_fd )
    register    int     pd_fd;
{
    register    int                 channel;
    register struct CDM_SHMEMORY    *memptr;

    /* initialize */
    memptr = shmem;

    for ( channel = 0; channel < MAX_STATIONS; channel++ ) {
        if ( memptr->source_fd[ channel ] == pd_fd )
            return( channel );
    } /* end for */

    fprintf( stderr, "PM:: Unable to find channel in pd_fd_to_channel for
pd_fd: %d\n",
                pd_fd );
#ifdef GO_AWAY
    go_away();
#endif

    return( -1 );

} /* end pd_fd_to_channel */
/*****************************************************/

/*****************************************************/
/*
 * This routine removes a channel from a given distributor
 * and resets all appropriate memory structures, counts, etc.
 */
remove_channel(channel,station)
    register    int     channel;
    register    int     station;
{
    register    int     index;
    register    int     i;
    register    int     oldnum;
```

```
#ifdef MANAGE
fprintf(stderr,"PM::    remove_channel    channel:%d
station:%d\n",channel,station);
#endif

/*
 * Get the index of the entry in the Distributors table
 */
    index   = channel_to_index(channel,station);
    oldnum  = shmem->Stations[station].num_channels;

#ifdef MANAGE
fprintf(stderr,"PM:: remove_channel:%d station:%d oldnum:%d  index:%d\n",
channel,station,oldnum,index);
#endif

/*
 * Decrement the number of channels this distributor is broadcasting.
 */
    shmem->Stations[station].num_channels--;

#ifdef MANAGE
fprintf(stderr,"PM:: number of channels now is:%d\n",
shmem->Stations[station].num_channels);
#endif
    if (shmem->Stations[station].num_channels<0)
        shmem->Stations[station].num_channels   = 0;

#ifdef MANAGE
fprintf(stderr,"PM:: just BEFORE shifted everything\n");
for (i=0;i<shmem->Stations[station].num_channels+1;i++) {
    fprintf(stderr,"PM:: station:%d dist_chan:%d dist_client:%d\n",
station,
shmem->Stations[station].dist_channel[i],
shmem->Stations[station].dist_client[i]);
}
#endif
/*
 * Shift down all the client-channel pairs to fill the gap.
 */
    for (i=index;i<oldnum-1;i++) {
        shmem->Stations[station].dist_channel[i] =
            shmem->Stations[station].dist_channel[i+1];
        shmem->Stations[station].dist_client[i] =
            shmem->Stations[station].dist_client[i+1];
#ifdef MANAGE
fprintf(stderr,"PM::    dist_chan:%d    -
dist_chan:%d,station:%d\n",i,i+1,station);
fprintf(stderr,"PM::    dist_clie:%d    -
dist_clie:%d,station:%d\n",i,i+1,station);
#endif
```

```
    } /* end for */
    shmem->Stations[station].dist_channel[oldnum-1] = -1;
    shmem->Stations[station].dist_client[oldnum-1]  = -1;

#ifdef MANAGE
fprintf(stderr,"PM:: just shifted everything\n");
for (i=0;i<shmem->Stations[station].num_channels;i++) {
    fprintf(stderr,"PM:: station:%d dist_chan:%d dist_client:%d\n",
station,
shmem->Stations[station].dist_channel[i],
shmem->Stations[station].dist_client[i]);
}
fprintf(stderr,"PM:: shift results above\n");
#endif
} /* end remove_source */
/*****************************************************/



/*****************************************************/
/*
 * This routine performs the following:
 * 1) The file descriptor being passed in is the file
 *    descriptor for a particular distributor, whose
 *    sibling receiver has gotten an expose event from
 *    the local server. This distributor fd will be
 *    used to determine the sibling receiver's fd.
 *
 * 2) The sibling reciever's fd is then used to
 *    search the fd array to find a matching fd,
 *    which will then give us the receiving channel.
 */
int
fd_to_channel(pd_fd)
    register int                    pd_fd;
{
    register int                    pr_fd;
    register int                    i,j;
    register struct CDM_SHMEMORY    *memptr;

#ifdef TRACE
fprintf(stderr,"PM:: fd_to_channel for fd:%d\n",pd_fd);
#endif

/*
 * Search for the sibling receiver's fd
 */
    memptr  = shmem;
    pr_fd   = -1;
    for (i=0;i<MAX_STATIONS;i++)
        if (memptr->Stations[i].pd_fd==pd_fd)
            pr_fd   = memptr->Stations[i].pr_fd;
```

```
    if (pr_fd<0) {
        fprintf(stderr,"PM:: Unable to find sibling receiver fd.\n");
#ifdef GO_AWAY
        go_away();
#endif
    }
#ifdef TRACE
fprintf(stderr,"PM:: sibling receiver is:%d\n",pr_fd);
#endif
/*
 * Search the receiver's fd array for this fd.
 */
    for (i=0;i<MAX_CHANNELS;i++)
        for (j=0;j<MAX_RECEIVERS;j++)
            if (memptr->dest_fd[i][j]==pr_fd)
                return(i);
    fprintf(stderr,"PM:: Unable to find source channel for fd.\n");
#ifdef GO_AWAY
    go_away();
#endif

} /* end fd_to_channel */
/*******************************************************/
```

```
/*****************************************************
 *                                                   *
 *      For a listing of alarm.c, see Appendix L     *
 *                                                   *
 *****************************************************/
```

```
/*****************************************************
 *                                                   *
 *      For a listing of netread.c, see Appendix L   *
 *                                                   *
 *****************************************************/
```

```
/*****************************************************
 *                                                   *
 *      For a listing of netwrite.c, see Appendix L *
 *                                                   *
 *****************************************************/
```

APPENDIX Q
ALIASES AND SCRIPT FILES

The Display Sharing prototype uses the following aliases:

```
alias DS 'cd /user/DS'
alias mc 'cd /user/DS/mc'
alias DShost 'cd /user/DS/host'
alias sim 'cd /user/DS/sim'
alias nasa 'cd /user/DS/nasadisp'
```

The Display Sharing protype has defined the following script files:

dsproto:

```
xterm -geometry 76x67+8+13 -T SOURCE/XGCM: -n SOURCE/XGCM: -e rlogin
     orion&
xterm -geometry 76x31+653+220 -T DEST/Dummy: -n DEST/Dummy: &
xterm -geometry 76x18+653+650 -T MANG/CDM: -n MANG/CDM: -e rlogin
     polaris&
```

dsserv:

```
cp /etc/mcgraphics/X11/gcm12.ds /etc/mcgraphics/X11/gcm12
```

fvserv <server #>:

```
cp /etc/mcgraphics/X11/gcm12.fv /etc/mcgraphics/X11/gcm12
Xgcm :%1 &
```

start_source:

```
echo 'Start SOURCE Workstation Code'
XGCM :1 &
echo 'X Window Server Started for Display local:1.0'
sleep 20
../sim/cmap_orion &
sleep 10
pd &
echo 'Protocol Distributor Started'
pr &
echo 'Protocol Receiver Started'
ldmg &
../sim/sc &
```

start_dest:

```
echo 'Starting Destination Workstation stuff...'
../sim/cmap_aries &
../sim/sc &
dummy &
nice -20 pd &
nice -20 pr &
ldmg &
```

APPENDIX R
RPC RELATED INCLUDE FILES

```
/* @(#)auth.h  20.1 (MASSCOMP) 3/24/87 compiled 3/16/88 */
/*
 * auth.h, Authentication interface.
 *
 * Copyright (C) 1984, Sun Microsystems, Inc.
 *
 * The data structures are completely opaque to the client.  The client
 * is required to pass a AUTH * to routines that create rpc
 * "sessions".
 */

/*
 * Modified for Masscomp kernel 3/87 by Bob Doolittle
 */

#define MAX_AUTH_BYTES 400


/*
 * Status returned from authentication check
 */
enum auth_stat {
   AUTH_OK=0,
   /*
    * failed at remote end
    */
   AUTH_BADCRED=1,          /* bogus credentials (seal broken) */
   AUTH_REJECTEDCRED=2,        /* client should begin new session */
   AUTH_BADVERF=3,          /* bogus verifier (seal broken) */
   AUTH_REJECTEDVERF=4,        /* verifier expired or was replayed */
   AUTH_TOOWEAK=5,          /* rejected due to security reasons */
   /*
    * failed locally
   */
   AUTH_INVALIDRESP=6,      /* bogus response verifier */
   AUTH_FAILED=7            /* some unknown reason */
};


union des_block {
   struct {
       u_long high;
       u_long low;
   } key;
   char c[8];
};
```

```
/*
 * Authentication info.  Opaque to client.
 */
struct opaque_auth {
   enum_t  oa_flavor;      /* flavor of auth */
   caddr_t oa_base;    /* address of more auth stuff */
   u_int   oa_length;      /* not to exceed MAX_AUTH_BYTES */
};


/*
 * Auth handle, interface to client side authenticators.
 */
typedef struct {
   struct  opaque_auth ah_cred;
   struct  opaque_auth ah_verf;
   union   des_block   ah_key;
   struct auth_ops {
       void    (*ah_nextverf)();
       int (*ah_marshal)();    /* nextverf & serialize */
       int (*ah_validate)();   /* validate varifier */
       int (*ah_refresh)();    /* refresh credentials */
       void    (*ah_destroy)();    /* destroy this structure */
   } *ah_ops;
   caddr_t ah_private;
} AUTH;


/*
 * Authentication ops.
 * The ops and the auth handle provide the interface to the authenticators.
 *
 * AUTH*auth;
 * XDR *xdrs;
 * struct opaque_auth verf;
 */
#define AUTH_NEXTVERF(auth)     \
       ((*((auth)->ah_ops->ah_nextverf))(auth))
#define auth_nextverf(auth)     \
       ((*((auth)->ah_ops->ah_nextverf))(auth))

#define AUTH_MARSHALL(auth, xdrs)  \
       ((*((auth)->ah_ops->ah_marshal))(auth, xdrs))
#define auth_marshall(auth, xdrs)  \
       ((*((auth)->ah_ops->ah_marshal))(auth, xdrs))

#define AUTH_VALIDATE(auth, verfp) \
       ((*((auth)->ah_ops->ah_validate))((auth), verfp))
#define auth_validate(auth, verfp) \
       ((*((auth)->ah_ops->ah_validate))((auth), verfp))

#define AUTH_REFRESH(auth)      \
```

```
        ((*((auth)->ah_ops->ah_refresh))(auth))
#define auth_refresh(auth)      \
        ((*((auth)->ah_ops->ah_refresh))(auth))

#define AUTH_DESTROY(auth)      \
        ((*((auth)->ah_ops->ah_destroy))(auth))
#define auth_destroy(auth)      \
        ((*((auth)->ah_ops->ah_destroy))(auth))


extern struct opaque_auth _null_auth;


/*
 * These are the various implementations of client side authenticators.
 */

/*
 * Unix style authentication
 * AUTH *authunix_create(machname, uid, gid, len, aup_gids)
 * char *machname;
 * int uid;
 * int gid;
 * int len;
 * int *aup_gids;
 */
#ifdef KERNEL
extern AUTH *authkern_create();     /* takes no parameters */
#else
extern AUTH *authunix_create();
extern AUTH *authunix_create_default();/* takes no parameters */
extern AUTH *authnone_create();     /* takes no parameters */
#endif

#defineAUTH_NULL    0
#defineAUTH_UNIX    1       /* unix style (uid, gids) */
#defineAUTH_SHORT   2       /* short hand unix style */
```

```
/* @(#)auth_unix.h 20.1 (MASSCOMP) 3/24/87 compiled 3/16/88 */
/*
 * auth_unix.h, Protocol for UNIX style authentication parameters for RPC
 *
 * Copyright (C) 1984, Sun Microsystems, Inc.
 */

/*
 * Modified for Masscomp kernel 3/87 by Bob Doolittle
 */

/*
 * The system is very weak.  The client uses no encryption for  it
 * credentials and only sends null verifiers.  The server sends backs
 * null verifiers or optionally a verifier that suggests a new short hand
 * for the credentials.
 */

/* The machine name is part of a credential; it may not exceed 255 bytes
*/
#define MAX_MACHINE_NAME 255

/* gids compose part of a credential; there may not be more than 10 of them
*/
#define NGRPS 8

/*
 * Unix style credentials.
 */
struct authunix_parms {
   u_long   aup_time;
   char    *aup_machname;
   int  aup_uid;
   int  aup_gid;
   u_int    aup_len;
   int *aup_gids;
};

extern bool_t xdr_authunix_parms();

/*
 * If a response verifier has flavor AUTH_SHORT,
 * then the body of the response verifier encapsulates the following
structure;
 * again it is serialized in the obvious fashion.
 */
struct short_hand_verf {
   struct opaque_auth new_cred;
};
```

```
/* @(#)clnt.h  20.1 (MASSCOMP) 3/24/87 compiled 3/16/88 */
/*
 * clnt.h - Client side remote procedure call interface.
 *
 * Copyright (C) 1984, Sun Microsystems, Inc.
 */

/*
 * Modified for Masscomp kernel 3/87 by Bob Doolittle
 */

/*
 * Rpc calls return an enum clnt_stat.  This should be looked at more,
 * since each implementation is required to live with this (implementation
 * independent) list of errors.
 */
enum clnt_stat {
   RPC_SUCCESS=0,          /* call succeeded */
   /*
    * local errors
    */
   RPC_CANTENCODEARGS=1,        /* can't encode arguments */
   RPC_CANTDECODERES=2,         /* can't decode results */
   RPC_CANTSEND=3,          /* failure in sending call */
   RPC_CANTRECV=4,          /* failure in receiving result */
   RPC_TIMEDOUT=5,          /* call timed out */
   /*
    * remote errors
    */
   RPC_VERSMISMATCH=6,      /* rpc versions not compatible */
   RPC_AUTHERROR=7,     /* authentication error */
   RPC_PROGUNAVAIL=8,       /* program not available */
   RPC_PROGVERSMISMATCH=9,      /* program version mismatched */
   RPC_PROCUNAVAIL=10,      /* procedure unavailable */
   RPC_CANTDECODEARGS=11,       /* decode arguments error */
   RPC_SYSTEMERROR=12,      /* generic "other problem" */

   /*
    * callrpc errors
    */
   RPC_UNKNOWNHOST=13,      /* unknown host name */

   /*
    * _ create errors
    */
   RPC_PMAPFAILURE=14,      /* the pmapper failed in its call */
   RPC_PROGNOTREGISTERED=15,    /* remote program is not registered */
   /*
    * unspecified error
    */
   RPC_FAILED=16
};
```

```
/*
 * Error info.
 */
struct rpc_err {
   enum clnt_stat re_status;
   union {
       int RE_errno;        /* realated system error */
       enum auth_stat RE_why;  /* why the auth error occurred */
       struct {
           u_long low; /* lowest verion supported */
           u_long high;    /* highest verion supported */
       } RE_vers;
       struct {         /* maybe meaningful if RPC_FAILED */
           long s1;
           long s2;
       } RE_lb;         /* life boot & debugging only */
   } ru;
#define re_errno    ru.RE_errno
#define re_why      ru.RE_why
#define re_vers     ru.RE_vers
#define re_lb       ru.RE_lb
};


/*
 * Client rpc handle.
 * Created by individual implementations, see e.g. rpc_udp.c.
 * Client is responsible for initializing auth, see e.g. auth_none.c.
 */
typedef struct {
   AUTH    *cl_auth;           /* authenticator */
   struct clnt_ops {
       enum clnt_stat  (*cl_call)();   /* call remote procedure */
       void        (*cl_abort)();  /* abort a call */
       void        (*cl_geterr)(); /* get specific error code */
       bool_t      (*cl_freeres)(); /* frees results */
       void        (*cl_destroy)();/* destroy this structure */
   } *cl_ops;
   caddr_t         cl_private; /* private stuff */
} CLIENT;


/*
 * client side rpc interface ops
 *
 * Parameter types are:
 *
 */

/*
```

```
 * enum clnt_stat
 * CLNT_CALL(rh, proc, xargs, argsp, xres, resp, timeout)
 * CLIENT *rh;
 * u_long proc;
 * xdrproc_t xargs;
 * caddr_t argsp;
 * xdrproc_t xres;
 * caddr_t resp;
 * struct timeval timeout;
 */
#defineCLNT_CALL(rh, proc, xargs, argsp, xres, resp, secs) \
   ((*(rh)->cl_ops->cl_call)(rh, proc, xargs, argsp, xres, resp, secs))
#defineclnt_call(rh, proc, xargs, argsp, xres, resp, secs) \
   ((*(rh)->cl_ops->cl_call)(rh, proc, xargs, argsp, xres, resp, secs))

/*
 * void
 * CLNT_ABORT(rh);
 * CLIENT *rh;
 */
#defineCLNT_ABORT(rh)  ((*(rh)->cl_ops->cl_abort)(rh))
#defineclnt_abort(rh)  ((*(rh)->cl_ops->cl_abort)(rh))

/*
 * struct rpc_err
 * CLNT_GETERR(rh);
 * CLIENT *rh;
 */
#defineCLNT_GETERR(rh,errp)    ((*(rh)->cl_ops->cl_geterr)(rh, errp))
#defineclnt_geterr(rh,errp)    ((*(rh)->cl_ops->cl_geterr)(rh, errp))


/*
 * bool_t
 * CLNT_FREERES(rh, xres, resp);
 * CLIENT *rh;
 * xdrproc_t xres;
 * caddr_t resp;
 */
#defineC L N T _ F R E E R E S ( r h , x r e s , r e s p )
((*(rh)->cl_ops->cl_freeres)(rh,xres,resp))
#definec l n t _ f r e e r e s ( r h , x r e s , r e s p )
((*(rh)->cl_ops->cl_freeres)(rh,xres,resp))

/*
 * void
 * CLNT_DESTROY(rh);
 * CLIENT *rh;
 */
#defineCLNT_DESTROY(rh)    ((*(rh)->cl_ops->cl_destroy)(rh))
#defineclnt_destroy(rh)    ((*(rh)->cl_ops->cl_destroy)(rh))
```

```
/*
 * RPCTEST is a test program which is accessable on every rpc
 * transport/port.  It is used for testing, performance evaluation,
 * and network administration.
 */

#define RPCTEST_PROGRAM     ((u_long)1)
#define RPCTEST_VERSION     ((u_long)1)
#define RPCTEST_NULL_PROC   ((u_long)2)
#define RPCTEST_NULL_BATCH_PROC((u_long)3)

/*
 * By convention, procedure 0 takes null arguments and returns them
 */

#define NULLPROC ((u_long)0)

/*
 * Below are the client handle creation routines for the various
 * implementations of client side rpc.  They can return NULL if a
 * creation failure occurs.
 */

#ifndef KERNEL
/*
 * Memory based rpc (for speed check and testing)
 * CLIENT *
 * clntraw_create(prog, vers)
 * u_long prog;
 * u_long vers;
 */
extern CLIENT *clntraw_create();

/*
 * TCP based rpc
 * CLIENT *
 * clnttcp_create(raddr, prog, vers, sockp, sendsz, recvsz)
 * struct sockaddr_in *raddr;
 * u_long prog;
 * u_long version;
 * register int *sockp;
 * u_int sendsz;
 * u_int recvsz;
 */
extern CLIENT *clnttcp_create();

/*
 * UDP based rpc.
 * CLIENT *
 * clntudp_create(raddr, program, version, wait, sockp)
 * struct sockaddr_in *raddr;
```

```
 * u_long program;
 * u_long version;
 * struct timeval wait;
 * int *sockp;
 *
 * Same as above, but you specify max packet sizes.
 * CLIENT *
 * clntudp_bufcreate(raddr, program, version, wait, sockp, sendsz, recvsz)
 * struct sockaddr_in *raddr;
 * u_long program;
 * u_long version;
 * struct timeval wait;
 * int *sockp;
 * u_int sendsz;
 * u_int recvsz;
 */
extern CLIENT *clntudp_create();
extern CLIENT *clntudp_bufcreate();

/*
 * If a creation fails, the following allows the user to figure out why.
 */
struct rpc_createerr {
   enum clnt_stat cf_stat;
   struct rpc_err cf_error; /* useful when cf_stat == RPC_PMAPFAILURE */
};

extern struct rpc_createerr rpc_createerr;
#endif !KERNEL

#ifdef KERNEL
/*
 * Kernel udp based rpc
 * CLIENT *
 * clntkudp_create(addr, pgm, vers)
 * struct sockaddr_in *addr;
 * u_long pgm;
 * u_long vers;
 */
extern CLIENT *clntkudp_create();
#endif

#define UDPMSGSIZE 8800    /* rpc imposed limit on udp msg size */
#define RPCSMALLMSGSIZE400 /* a more reasonable packet size */
```

```
/* @(#)rpc_msg.h   20.1 (MASSCOMP) 3/24/87 compiled 3/16/88 */
/*
 * rpc_msg.h
 * rpc message definition
 *
 * Copyright (C) 1984, Sun Microsystems, Inc.
 */

/*
 * Modified for Masscomp kernel 3/87 by Bob Doolittle
 */

#define RPC_MSG_VERSION    ((u_long) 2)
#define RPC_SERVICE_PORT   ((u_short) 2048)

/*
 * Bottom up definition of an rpc message.
 * NOTE: call and reply use the same overall stuct but
 * different parts of unions within it.
 */

enum msg_type {
   CALL=0,
   REPLY=1
};

enum reply_stat {
   MSG_ACCEPTED=0,
   MSG_DENIED=1
};

enum accept_stat {
   SUCCESS=0,
   PROG_UNAVAIL=1,
   PROG_MISMATCH=2,
   PROC_UNAVAIL=3,
   GARBAGE_ARGS=4,
   SYSTEM_ERR=5
};

enum reject_stat {
   RPC_MISMATCH=0,
   AUTH_ERROR=1
};

/*
 * Reply part of an rpc exchange
 */

/*
 * Reply to an rpc request that was accepted by the server.
 * Note: there could be an error even though the request was
```

```
 * accepted.
 */
struct accepted_reply {
   struct opaque_auth  ar_verf;
   enum accept_statar_stat;
   union {
       struct {
           u_long  low;
           u_long  high;
       } AR_versions;
       struct {
           caddr_t where;
           xdrproc_t proc;
       } AR_results;
       /* and many other null cases */
   } ru;
#definear_results  ru.AR_results
#definear_vers     ru.AR_versions
};

/*
 * Reply to an rpc request that was rejected by the server.
 */
struct rejected_reply {
   enum reject_stat rj_stat;
   union {
       struct {
           u_long low;
           u_long high;
       } RJ_versions;
       enum auth_stat RJ_why;  /* why authentication did not work */
   } ru;
#definerj_vers ru.RJ_versions
#definerj_why  ru.RJ_why
};

/*
 * Body of a reply to an rpc request.
 */
struct reply_body {
   enum reply_stat rp_stat;
   union {
       struct accepted_reply RP_ar;
       struct rejected_reply RP_dr;
   } ru;
#definerp_acpt ru.RP_ar
#definerp_rjct ru.RP_dr
};

/*
 * Body of an rpc request call.
 */
```

```
struct call_body {
   u_long cb_rpcvers;  /* must be equal to two */
   u_long cb_prog;
   u_long cb_vers;
   u_long cb_proc;
   struct opaque_auth cb_cred;
   struct opaque_auth cb_verf; /* protocol specific - provided by client
*/
};

/*
 * The rpc message
 */
struct rpc_msg {
   u_long          rm_xid;
   enum msg_type       rm_direction;
   union {
       struct call_body RM_cmb;
       struct reply_body RM_rmb;
   } ru;
#definerm_call     ru.RM_cmb
#definerm_reply    ru.RM_rmb
};
#defineacpted_rply ru.RM_rmb.ru.RP_ar
#definerjcted_rply ru.RM_rmb.ru.RP_dr


/*
 * XDR routine to handle a rpc message.
 * xdr_callmsg(xdrs, cmsg)
 * XDR *xdrs;
 * struct rpc_msg *cmsg;
 */
extern bool_t  xdr_callmsg();

/*
 * XDR routine to pre-serialize the static part of a rpc message.
 * xdr_callhdr(xdrs, cmsg)
 * XDR *xdrs;
 * struct rpc_msg *cmsg;
 */
extern bool_t  xdr_callhdr();

/*
 * XDR routine to handle a rpc reply.
 * xdr_replymsg(xdrs, rmsg)
 * XDR *xdrs;
 * struct rpc_msg *rmsg;
 */
extern bool_t  xdr_replymsg();

/*
```

```
 * Fills in the error part of a reply message.
 * _seterr_reply(msg, error)
 * struct rpc_msg *msg;
 * struct rpc_err *error;
 */
extern void_seterr_reply();
```

```
/* @(#)svc.h   20.2 (MASSCOMP) 5/15/87 compiled 3/16/88 */
/*
 * svc.h, Server-side remote procedure call interface.
 *
 * Copyright (C) 1984, Sun Microsystems, Inc.
 */

/*
 * Modified for Masscomp kernel 3/87 by Bob Doolittle
 */

/*
 * This interface must manage two items concerning remote procedure calling:
 *
 * 1) An arbitrary number of transport connections upon which rpc requests
 * are received.  The two most notable transports are TCP and UDP;  they
are
 *  created and registered by routines in svc_tcp.c and svc_udp.c,
respectively;
 * they in turn call xprt_register and xprt_unregister.
 *
 * 2) An arbitrary number of locally registered services.  Services are
 * described by the following four data: program number, version number,
 * "service dispatch" function, a transport handle, and a boolean that
 * indicates whether or not the exported program should be registered with
a
 * local binder service;  if true the program's number and version and the
 * port number from the transport handle are registered with the binder.
 * These data are registered with the rpc svc system via svc_register.
 *
 * A service's dispatch function is called whenever an rpc request comes
in
 * on a transport.  The request's program and version numbers must match
 * those of the registered service.  The dispatch function is passed two
 * parameters, struct svc_req * and SVCXPRT *, defined below.
 */

enum xprt_stat {
   XPRT_DIED,
   XPRT_MOREREQS,
   XPRT_IDLE
};

/*
 * Server side transport handle
 */
typedef struct {
#ifdef KERNEL
#ifdef EXOS
   struct soinfo   *xp_sock;
#else EXOS
   struct socket   *xp_sock;
```

```
#endif EXOS
#else KERNEL
   int     xp_sock;
#endif KERNEL
   u_short     xp_port;     /* associated port number */
   struct xp_ops {
       bool_t  (*xp_recv)();    /* receive incomming requests */
       enum xprt_stat (*xp_stat)(); /* get transport status */
       bool_t  (*xp_getargs)(); /* get arguments */
       bool_t  (*xp_reply)();   /* send reply */
       bool_t  (*xp_freeargs)();/* free mem allocated for args */
       void    (*xp_destroy)(); /* destroy this struct */
   } *xp_ops;
   int     xp_addrlen;  /* length of remote address */
   struct sockaddr_in xp_raddr;     /* remote address */
   struct opaque_auth xp_verf;  /* raw response verifier */
   caddr_t     xp_p1;       /* private */
   caddr_t     xp_p2;       /* private */
} SVCXPRT;

/*
 *  Approved way of getting address of caller
 */
#define svc_getcaller(x) (&(x)->xp_raddr)

/*
 * Operations defined on an SVCXPRT handle
 *
 * SVCXPRT     *xprt;
 * struct rpc_msg  *msg;
 * xdrproc_t        xargs;
 * caddr_t      argsp;
 */
#define SVC_RECV(xprt, msg)             \
   (*(xprt)->xp_ops->xp_recv)((xprt), (msg))
#define svc_recv(xprt, msg)             \
   (*(xprt)->xp_ops->xp_recv)((xprt), (msg))

#define SVC_STAT(xprt)                  \
   (*(xprt)->xp_ops->xp_stat)(xprt)
#define svc_stat(xprt)                  \
   (*(xprt)->xp_ops->xp_stat)(xprt)

#define SVC_GETARGS(xprt, xargs, argsp)         \
   (*(xprt)->xp_ops->xp_getargs)((xprt), (xargs), (argsp))
#define svc_getargs(xprt, xargs, argsp)         \
   (*(xprt)->xp_ops->xp_getargs)((xprt), (xargs), (argsp))

#define SVC_REPLY(xprt, msg)                \
   (*(xprt)->xp_ops->xp_reply) ((xprt), (msg))
#define svc_reply(xprt, msg)                \
   (*(xprt)->xp_ops->xp_reply) ((xprt), (msg))
```

```
#define SVC_FREEARGS(xprt, xargs, argsp)        \
   (*(xprt)->xp_ops->xp_freeargs)((xprt), (xargs), (argsp))
#define svc_freeargs(xprt, xargs, argsp)        \
   (*(xprt)->xp_ops->xp_freeargs)((xprt), (xargs), (argsp))

#define SVC_DESTROY(xprt)                \
   (*(xprt)->xp_ops->xp_destroy)(xprt)
#define svc_destroy(xprt)                \
   (*(xprt)->xp_ops->xp_destroy)(xprt)


/*
 * Service request
 */
struct svc_req {
   u_long      rq_prog;    /* service program number */
   u_long      rq_vers;    /* service protocol version */
   u_long      rq_proc;    /* the desired procedure */
   struct opaque_auth rq_cred; /* raw creds from the wire */
   caddr_t     rq_clntcred;    /* read only cooked cred */
   SVCXPRT *rq_xprt;       /* associated transport */
};


/*
 * Service registration
 *
 * svc_register(xprt, prog, vers, dispatch, protocol)
 * SVCXPRT *xprt;
 * u_long prog;
 * u_long vers;
 * void (*dispatch)();
 * int protocol;  /* like TCP or UDP, zero means do not register
 */
extern bool_t  svc_register();

/*
 * Service un-registration
 *
 * svc_unregister(prog, vers)
 * u_long prog;
 * u_long vers;
 */
extern voidsvc_unregister();

/*
 * Transport registration.
 *
 * xprt_register(xprt)
 * SVCXPRT *xprt;
 */
```

```
extern voidxprt_register();

#ifndef KERNEL
/*
 * Transport un-register
 *
 * xprt_unregister(xprt)
 * SVCXPRT *xprt;
 */
extern voidxprt_unregister();
#endif !KERNEL


/*
 * When the service routine is called, it must first check to see if it
 * knows about the procedure;  if not, it should call svcerr_noproc
 * and return.  If so, it should deserialize its arguments via
 * SVC_GETARGS (defined above).  If the deserialization does not work,
 * svcerr_decode should be called followed by a return.  Successful
 * decoding of the arguments should be followed the execution of the
 * procedure's code and a call to svc_sendreply.
 *
 * Also, if the service refuses to execute the procedure due to too-
 * weak authentication parameters, svcerr_weakauth should be called.
 * Note: do not confuse access-control failure with weak authentication!
 *
 * NB: In pure implementations of rpc, the caller always waits for a reply
 * msg.  This message is sent when svc_sendreply is called.
 * Therefore pure service implementations should always call
 * svc_sendreply even if the function logically returns void;  use
 * xdr.h - xdr_void for the xdr routine.  HOWEVER, tcp based rpc allows
 * for the abuse of pure rpc via batched calling or pipelining.  In the
 * case of a batched call, svc_sendreply should NOT be called since
 * this would send a return message, which is what batching tries to avoid.
 * It is the service/protocol writer's responsibility to know which calls
are
 * batched and which are not.  Warning: responding to batch calls may
 * deadlock the caller and server processes!
 */

extern bool_t  svc_sendreply();
extern voidsvcerr_decode();
extern voidsvcerr_weakauth();
extern voidsvcerr_noproc();

/*
 * Lowest level dispatching -OR- who owns this process anyway.
 * Somebody has to wait for incoming requests and then call the correct
 * service routine.  The routine svc_run does infinite waiting; i.e.,
 * svc_run never returns.
 * Since another (co-existant) package may wish to selectively wait for
 * incoming calls or other events outside of the rpc architecture, the
```

```
 * routine svc_getreq is provided.  It must be passed readfds, the
 * "in-place" results of a select system call (see select, section 2).
 */

#ifndef KERNEL
/* dynamic; must be inspected before each call to select */
extern int svc_fds;

/*
 * a small program implemented by the svc_rpc implementation itself;
 * also see clnt.h for protocol numbers.
 */
extern void rpctest_service();
#endif !KERNEL

extern voidsvc_getreq();
extern voidsvc_run();   /* never returns */

/*
 * Socket to use on svcxxx_create call to get default socket
 */
#defineRPC_ANYSOCK -1

/*
 * These are the existing service side transport implementations
 */

#ifndef KERNEL
/*
 * Memory based rpc for testing and timing.
 */
extern SVCXPRT *svcraw_create();

/*
 * Udp based rpc.
 */
extern SVCXPRT *svcudp_create();
extern SVCXPRT *svcudp_bufcreate();

/*
 * Tcp based rpc.
 */
extern SVCXPRT *svctcp_create();

#else

/*
 * Kernel udp based rpc.
 */
extern SVCXPRT *svckudp_create();
#endif !KERNEL
```

```
/* @(#)svc_auth.h  20.1 (MASSCOMP) 3/24/87 compiled 3/16/88 */
/*
 * svc_auth.h, Service side of rpc authentication.
 *
 * Copyright (C) 1984, Sun Microsystems, Inc.
 */

/*
 * Modified for Masscomp kernel 3/87 by Bob Doolittle
 */

/*
 * Server side authenticator
 */
extern enum auth_stat _authenticate();
```

```
/* @(#)types.h 20.1 (MASSCOMP) 3/24/87 compiled 3/16/88 */
/*
 * Rpc additions to <sys/types.h>
 */

/*
 * Modified for Masscomp kernel 3/87 by Bob Doolittle
 */

#definebool_t  int
#defineenum_t  int
#defineFALSE   (0)
#defineTRUE    (1)
#define __dontcare__   -1

#ifndef KERNEL
#define mem_alloc(bsize)   malloc(bsize)
#define mem_free(ptr, bsize)   free(ptr)
#ifndef major      /* ouch! */
#include <sys/types.h>
#endif
#else
#define mem_alloc(bsize)   kmem_alloc((u_int)bsize)
#define mem_free(ptr, bsize)   kmem_free((caddr_t)(ptr), (u_int)(bsize))
#endif
```

```
/* @(#)xdr.h   20.1 (MASSCOMP) 3/24/87 compiled 3/16/88 */
/*
 * xdr.h, External Data Representation Serialization Routines.
 *
 * Copyright (C) 1984, Sun Microsystems, Inc.
 */

/*
 * Modified for Masscomp kernel 3/87 by Bob Doolittle
 */

/*
 * XDR provides a conventional way for converting between C data
 * types and an external bit-string representation.  Library supplied
 * routines provide for the conversion on built-in C data types.  These
 * routines and utility routines defined here are used to help implement
 * a type encode/decode routine for each user-defined type.
 *
 * Each data type provides a single procedure which takes two arguments:
 *
 * bool_t
 * xdrproc(xdrs, argresp)
 *     XDR *xdrs;
 *     <type> *argresp;
 *
 * xdrs is an instance of a XDR handle, to which or from which the data
 * type is to be converted.  argresp is a pointer to the structure to be
 * converted.  The XDR handle contains an operation field which indicates
 * which of the operations (ENCODE, DECODE * or FREE) is to be performed.
 *
 * XDR_DECODE may allocate space if the pointer argresp is null.  This
 * data can be freed with the XDR_FREE operation.
 *
 * We write only one procedure per data type to make it easy
 * to keep the encode and decode procedures for a data type consistent.
 * In many cases the same code performs all operations on a user defined
type,
 * because all the hard work is done in the component type routines.
 * decode as a series of calls on the nested data types.
 */

/*
 * Xdr operations.  XDR_ENCODE causes the type to be encoded into the
 * stream.  XDR_DECODE causes the type to be extracted from the stream.
 * XDR_FREE can be used to release the space allocated by an XDR_DECODE
 * request.
 */
enum xdr_op {
   XDR_ENCODE=0,
   XDR_DECODE=1,
   XDR_FREE=2
};
```

```
/*
 * This is the number of bytes per unit of external data.
 */
#define BYTES_PER_XDR_UNIT (4)

/*
 * A xdrproc_t exists for each data type which is to be encoded or decoded.
 *
 * The second argument to the xdrproc_t is a pointer to an opaque pointer.
 * The opaque pointer generally points to a structure of the data type
 * to be decoded.  If this pointer is 0, then the type routines should
 * allocate dynamic storage of the appropriate size and return it.
 * bool_t  (*xdrproc_t)(XDR *, caddr_t *);
 */
typedefbool_t (*xdrproc_t)();

/*
 * The XDR handle.
 * Contains operation which is being applied to the stream,
 * an operations vector for the paticular implementation (e.g. see
xdr_mem.c),
 * and two private fields for the use of the particular impelementation.
 */
typedef struct {
   enum xdr_op x_op;       /* operation; fast additional param */
   struct xdr_ops {
       bool_t  (*x_getlong)(); /* get a long from underlying stream */
       bool_t  (*x_putlong)(); /* put a long to " */
       bool_t  (*x_getbytes)();/* get some bytes from " */
       bool_t  (*x_putbytes)();/* put some bytes to " */
       u_int   (*x_getpostn)();/* returns bytes off from beginning */
       bool_t  (*x_setpostn)();/* lets you reposition the stream */
       long *  (*x_inline)();  /* buf quick ptr to buffered data */
       void    (*x_destroy)(); /* free privates of this xdr_stream */
   } *x_ops;
   caddr_t     x_public;   /* users' data */
   caddr_t     x_private;  /* pointer to private data */
   caddr_t     x_base;     /* private used for position info */
   int     x_handy;/* extra private word */
} XDR;

/*
 * Operations defined on a XDR handle
 *
 * XDR     *xdrs;
 * long    *longp;
 * caddr_t  addr;
 * u_int    len;
 * u_int    pos;
 */
#define XDR_GETLONG(xdrs, longp)          \
```

```
   (*(xdrs)->x_ops->x_getlong)(xdrs, longp)
#define xdr_getlong(xdrs, longp)                \
   (*(xdrs)->x_ops->x_getlong)(xdrs, longp)

#define XDR_PUTLONG(xdrs, longp)                \
   (*(xdrs)->x_ops->x_putlong)(xdrs, longp)
#define xdr_putlong(xdrs, longp)                \
   (*(xdrs)->x_ops->x_putlong)(xdrs, longp)

#define XDR_GETBYTES(xdrs, addr, len)           \
   (*(xdrs)->x_ops->x_getbytes)(xdrs, addr, len)
#define xdr_getbytes(xdrs, addr, len)           \
   (*(xdrs)->x_ops->x_getbytes)(xdrs, addr, len)

#define XDR_PUTBYTES(xdrs, addr, len)           \
   (*(xdrs)->x_ops->x_putbytes)(xdrs, addr, len)
#define xdr_putbytes(xdrs, addr, len)           \
   (*(xdrs)->x_ops->x_putbytes)(xdrs, addr, len)

#define XDR_GETPOS(xdrs)                \
   (*(xdrs)->x_ops->x_getpostn)(xdrs)
#define xdr_getpos(xdrs)                \
   (*(xdrs)->x_ops->x_getpostn)(xdrs)

#define XDR_SETPOS(xdrs, pos)               \
   (*(xdrs)->x_ops->x_setpostn)(xdrs, pos)
#define xdr_setpos(xdrs, pos)               \
   (*(xdrs)->x_ops->x_setpostn)(xdrs, pos)

#defineXDR_INLINE(xdrs, len)                \
   (*(xdrs)->x_ops->x_inline)(xdrs, len)
#definexdr_inline(xdrs, len)                \
   (*(xdrs)->x_ops->x_inline)(xdrs, len)

#defineXDR_DESTROY(xdrs)                \
   if ((xdrs)->x_ops->x_destroy)            \
       (*(xdrs)->x_ops->x_destroy)(xdrs)
#definexdr_destroy(xdrs)                \
   if ((xdrs)->x_ops->x_destroy)            \
       (*(xdrs)->x_ops->x_destroy)(xdrs)

/*
 * Support struct for discriminated unions.
 * You create an array of xdrdiscrim structures, terminated with
 * a entry with a null procedure pointer.  The xdr_union routine gets
 * the discriminant value and then searches the array of structures
 * for a matching value.  If a match is found the associated xdr routine
 * is called to handle that part of the union.  If there is
 * no match, then a default routine may be called.
 * If there is no match and no default routine it is an error.
 */
#define NULL_xdrproc_t ((xdrproc_t)0)
```

```
struct xdr_discrim {
   int value;
   xdrproc_t proc;
};

/*
 * In-line routines for fast encode/decode of primitve data types.
 * Caveat emptor: these use single memory cycles to get the
 * data from the underlying buffer, and will fail to operate
 * properly if the data is not aligned.  The standard way to use these
 * is to say:
 * if ((buf = XDR_INLINE(xdrs, count)) == NULL)
 *     return (FALSE);
 * <<< macro calls >>>
 * where ''count'' is the number of bytes of data occupied
 * by the primitive data types.
 *
 * N.B. and frozen for all time: each data type here uses 4 bytes
 * of external representation.
 */
#define IXDR_GET_LONG(buf)      ntohl(*buf++)
#define IXDR_PUT_LONG(buf, v)       (*buf++ = htonl(v))

#define IXDR_GET_BOOL(buf)      ((bool_t)IXDR_GET_LONG(buf))
#define IXDR_GET_ENUM(buf, t)       ((t)IXDR_GET_LONG(buf))
#define IXDR_GET_U_LONG(buf)        ((u_long)IXDR_GET_LONG(buf))
#define IXDR_GET_SHORT(buf)     ((short)IXDR_GET_LONG(buf))
#define IXDR_GET_U_SHORT(buf)       ((u_short)IXDR_GET_LONG(buf))

#define IXDR_PUT_BOOL(buf, v)       IXDR_PUT_LONG((buf), ((long)(v)))
#define IXDR_PUT_ENUM(buf, v)       IXDR_PUT_LONG((buf), ((long)(v)))
#define IXDR_PUT_U_LONG(buf, v)     IXDR_PUT_LONG((buf), ((long)(v)))
#define IXDR_PUT_SHORT(buf, v)      IXDR_PUT_LONG((buf), ((long)(v)))
#define IXDR_PUT_U_SHORT(buf, v)    IXDR_PUT_LONG((buf), ((long)(v)))

/*
 * These are the "generic" xdr routines.
 */
extern bool_t  xdr_void();
extern bool_t  xdr_int();
extern bool_t  xdr_u_int();
extern bool_t  xdr_long();
extern bool_t  xdr_u_long();
extern bool_t  xdr_short();
extern bool_t  xdr_u_short();
extern bool_t  xdr_bool();
extern bool_t  xdr_enum();
extern bool_t  xdr_array();
extern bool_t  xdr_bytes();
extern bool_t  xdr_opaque();
extern bool_t  xdr_string();
extern bool_t  xdr_union();
```

```
#ifndef KERNEL
extern bool_t  xdr_float();
extern bool_t  xdr_double();
extern bool_t  xdr_reference();
extern bool_t  xdr_wrapstring();
#endif !KERNEL

/*
 * These are the public routines for the various implementations of
 * xdr streams.
 */
extern void   xdrmem_create();     /* XDR using memory buffers */
#ifndef KERNEL
extern void   xdrstdio_create();   /* XDR using stdio library */
extern void   xdrrec_create();     /* XDR pseudo records for tcp */
extern bool_t xdrrec_endofrecord();/* make end of xdr record */
extern bool_t xdrrec_skiprecord(); /* move to begining of next record */
extern bool_t xdrrec_eof();    /* true iff no more input */
#else
extern void xdrmbuf_init();    /* XDR using kernel mbufs */
#endif !KERNEL
```

APPENDIX S
DISPLAY SHARING INCLUDE FILES

```
/*
 * Name: dist.h
 * Author  : P. Fitzgerald SwRI
 * Date: 10/10/89
 */

/* bits and fields within a resource id */
#define CLIENTOFFSET 20                    /* client field */
#define RESOURCE_ID_MASK   0x7FFFF         /* low 19 bits */
#define CLIENT_BITS(id) ((id) & 0xfff00000)/* hi 12 bits */
#define CLIENT_ID(id) ((int)(CLIENT_BITS(id) >> CLIENTOFFSET)) /*  hi  12
bits */

/*
 * Defines for the signal byte.
 */
#define    NOOP         0x20    /* No operation */
#define    X_DATA       0xf0    /* X Data Follows */
#define    EXPOSE       0xf1    /* Expose Event Request */
#define    SNAME        0xf2    /* Source Name Follows */
#define    GWATS        0xf3    /* Get Window Attributes Request */
#define    GGCS         0xf4    /* Get Graphics Context State Request */
#define    WATS         0xf5    /* Window Attributes Follow */
#define    GCS          0xf6    /* Graphics Context State Follows */
#define    SHUTCOMP     0xf7    /* Shutdown of receiver channel complete */

#define    SIGLEN       4
#define    LENLEN       4
#define    CLIENTLEN    4
#define    HDRLEN       (LENLEN+CLIENTLEN)
#define    PAKLEN       (LENLEN+CLIENTLEN+SIGLEN)
#define    MAX_BATCH    10

typedef struct HDR {
#ifdef notdef
   unsigned char   length[LENLEN];
   unsigned char   client[CLIENTLEN];
#endif
   int             length;
   int             client;
};

typedef struct PD_to_PM {
   unsigned char   signal[SIGLEN];
   struct HDR      header;
   unsigned char   buffer[XBUFFERSIZE];
};

typedef union COMPAK {
```

```
    struct PD_to_PM     pdtopm;
    unsigned char       compak[PAKLEN+XBUFFERSIZE];
};
```

```
/*
 * File         : ds_manage.h
 * Author       : P. Fitzgerald - SwRI
 * Date         : 10/3/89
 * Description : Contains Central Distribution Management RPC-related
 *               defines.
 */

/*
 * Define the Central Distribution Management RPC numbers.
 */
#defineCDM_PROG         0x20000050
#defineCDM_VERS         0x0
#defineCDM_PROC         0x1

/*
 * Sub-functions for Central Distribution Manager functions.
 */
#defineCDM_GET_LIST     0x01
#defineCDM_REG_PORT     0x02
#defineCDM_DIST_REQ     0x03
#defineCDM_RECV_REQ     0x04
#defineCDM_REMV_CHAN    0x05
#defineCDM_REMV_RECV    0x06
#defineCDM_PRESENT      0x07
#defineCDM_REG_DIST     0x08
#defineCDM_REG_RECV     0x09
#defineCDM_GO_AWAY      0xff

/*
 * Authorization returns.
 */
#defineAUTHORIZED       0x00
#define NOT_AUTHORIZED  0x01

/*
 * Miscellaneous defines
 */
#definePORTNAMLEN       20
#defineCHANNAMLEN       PORTNAMLEN
#defineSOURCENAMLEN     PORTNAMLEN
#defineHOSTNAMLEN       PORTNAMLEN
#defineREQUEST_OK       0x00
#defineREQUEST_FAILED   0x01
#defineMAX_STATIONS     20
#defineMAX_CHANNELS     20
#defineMAX_RECEIVERS    3           /* per channel */
#defineRFD              0
#defineWFD              1
#defineREGISTER_DISTRIBUTOR     -2
#defineREGISTER_RECEIVER        -3
```

```
/*
 * Structure to hold an ID (character string 10 digits) and
 * a port number (unsigned short).
 */
struct PortID {
   char        hostname[PORTNAMLEN];
   unsigned    short   portnum;
};

/*
 * Structure to request distributor registration
 * and return values.
 */
struct DistRegister {
   char        distname[HOSTNAMLEN];
   int         distributor_id;
};
struct RecvRegister {
   char        recvname[HOSTNAMLEN];
   int         distributor_id;
   unsigned short  portnum;
};

/*
 * Structure to return with distribution authorization request.
 */
struct DistAuth {
   int             authorization;
   int             channel;
   unsigned short  pm_port;
};

/*
 * Structure to old the hostname, and distributor fd
 * so that input from the distributor may be received
 * even if it is not currently distributing on a channel
 */
struct Station {
   int pd_fd;              /* for PM to read from */
   int pr_fd;              /* for PM to write to  */
   int num_channels;       /* number of channels this station
distributing on */
   int dist_channel[MAX_CHANNELS]; /* each slot contains channel
number,if active */
   int dist_client[MAX_CHANNELS];  /* client being distributed on
associated dist_channel */
   char    hostname[HOSTNAMLEN];   /* name of host for distributor */
};

/*
 * Structure to return with reception authorization request.
 */
```

```
struct RecvAuth {
   int             authorization;
   unsigned short  pm_port;
   unsigned long   default_gc;
   unsigned long   root;
};


/*
 * Structure to hold the channel map, listing sources and destinations.
 */
struct ChanMap {
   int     num_receivers;  /* number of receivers for this channel */
   int     client_id;      /* client number for this channel */
   unsigned short  recv_ports[MAX_RECEIVERS]; /* port numbers of each
                                              receiver */
   char    recv_hostname[MAX_RECEIVERS][HOSTNAMLEN];   /* port name for
                                                 each receiver */
   char    source_hostname[HOSTNAMLEN];    /* source name for this
channel */
};

/*
 * Structure to hold a request for channel reception.
 */
struct ChanReq {
   int             channel;
   int             distributor_id;
   struct  PortID  PortID;
};

/*
 * Structure to hold a channel id request
 */
struct ChanID {
   char            chanid[SOURCENAMLEN];
   char            hostname[HOSTNAMLEN];
   int             distributor_id;
   unsigned short  pr_port;
   unsigned long   default_gc;
   unsigned long   root;
   unsigned long   xid;
};

/*
 * Structure to hold a channel and port number to remove from
 * list.
 */
struct RemvRecv {
   int             channel;
   unsigned short  portnum;
};
```

```
/*
 * Structure to hold channel map in shared memory.
 */
#define CDM_KEY         250                 /* shared memory id          */
#define GOT_NEW_LIST        23
#define IM_WAITING      24
#define IM_DONE         25

/* status defines */
#define    AOK      0
#define    PM_DIE   1
struct CDM_SHMEMORY {
   int          sm_status;      /* shared memory status flag            */
   int          read_pipe;      /* used by CDM to indicate to pm that   */
                                /* there is a new Channel Map           */
   unsigned short  pm_port;     /* Protocol Multiplexer port number     */

   /* These flags indicate to the PM, just what type of change */
   /* took place in the 'new' channel map download by CDM      */
   int          new_source_channel;
   int          remv_source_channel;
   int          new_receiver_channel;
   int          remv_receiver_channel;
   int          station_index;

   unsigned long        source_default_gc[MAX_CHANNELS];
   unsigned long        source_root[MAX_CHANNELS];

   /* dest_fd is a list of all receiver file descriptors actively  receiving
protocol for each channel
   */
   int          dest_fd[MAX_CHANNELS][MAX_RECEIVERS];

   /* source_fd is a list of file descriptors that are active sources
for each channel
      */
   int                  source_fd[MAX_CHANNELS];

   /* A station is a workstation configured with a Protocol Distributor
and */
   /* a Protocol Receiver
 */
   struct Station  Stations[MAX_STATIONS];

   /* For each channel there is an entry in this table which tells us the
 */
   /*  o   number of receivers for that channel                          */
   /*  o   their port numbers                                            */
   /*  o   their hostnames                                               */
   /*  o   the hostname of the source station                            */
   /*  o   the client id of the display being distributed on that
```

```
channel
   */
   struct ChanMap       ChanMap[MAX_CHANNELS];

   char                 source_name[MAX_CHANNELS][SOURCENAMLEN];

#ifdef BYTES
   unsigned long        channel_bytes[MAX_CHANNELS];
   unsigned long        receiver_bytes[MAX_STATIONS];
   unsigned long        distributor_bytes[MAX_STATIONS];
   unsigned long        no_of_packets;
#endif
};
```

```
/*
 * File         : ds_manage.h
 */
static char ecodes[][80]   = {
" Success",
" BadRequest",
" BadValue",
" BadWindow",
" BadPixmap",
" BadAtom",
" BadCursor",
" BadFont",
" BadMatch",
" BadDrawable",
" BadAccess",
" BadAlloc",
" BadColor",
" BadGC",
" BadIDChoice",
" BadName",
" BadLength",
" BadImplementation" };
```

```
/* @(#)rpc.h   20.1 (MASSCOMP) 3/24/87 compiled 3/16/88 */
/*
 * rpc.h, Just includes the billions of rpc header files necessary to
 * do remote procedure calling.
 *
 * Copyright (C) 1984, Sun Microsystems, Inc.
 */

/*
 * Modified for Masscomp kernel 3/87 by Bob Doolittle
 */

#include "rpc/types.h"      /* some typedefs */
#include "netinet/in.h"

/* external data representation interfaces */
#include "rpc/xdr.h"        /* generic (de)serializer */

/* Client side only authentication */
#include "rpc/auth.h"       /* generic authenticator (client side) */

/* Client side (mostly) remote procedure call */
#include "rpc/clnt.h"       /* generic rpc stuff */

/* semi-private protocol headers */
#include "rpc/rpc_msg.h"    /* protocol for rpc messages */
#include "rpc/auth_unix.h" /* protocol for unix style cred */

/* Server side only remote procedure callee */
#include "rpc/svc.h"        /* service manager and multiplexer */
#include "rpc/svc_auth.h"   /* service side authenticator */
```

```
/*
 * File         : smdef.h
 */

/*
 * Definitions for shared memory
 */
#define SM_KEY      251                 /* shared memory id          */
#define SKEY        129                 /* semaphore key             */
#define D_USING     0                   /* status flag               */
#define SM_EMPTY    -1                  /* status flag               */
#define S_USING     1                   /* status flag               */
#define MULTICAST   2                   /* status flag               */
#define DIE         3                   /* die you pig               */
#define MAX_GCS     250
#define MAX_WINS    150

typedef struct GCWIN {
   XID              gid;
   XID              window;
   unsigned long    mask;
   XGCValues        GCValues;
};

typedef struct WIN {
   XID              window;
   XID              parent;
   unsigned long    background;
};

/* structure defining shared memory between two servers */
typedefstruct MC_SHMEMORY {

   /* Semaphore value to sync process upon */
   int          semaphore;
   int          buf_stat[XBUFFERNUM];

   /* Shared memory status flags */
   int          sm_status;          /* Shared Memory Status      */
   int          start;

   /* Protocol Distributor alive flag */
   int          pd_alive;
   int          pd_pid;

   /* Protocol Receiver initialized flag */
   int          pr_init;
   int          pd_init;

   /* Port numbers for various tasks */
   unsigned short  pm_port;             /* PM port number           */
   unsigned short  pr_port;             /* PR port number           */
```

```
/* ID number for the distributor */
int         distributor_id;

/* Propagate expose request information */
int         pd_propagate_expose;        /* expose please flag       */
XID         expose_window;

/* Default values for source server */
XID         default_gc;         /* Default graphics context */
XID         root;                   /* Root XID                 */

/* Flags to indicate that a client is wanted */
int         wanted[MAX_CLIENTS];        /* is client for multicast? */

/* Information for current state of window attributes */
/* and graphics contexts.                                        */
int                 get_wat;
int                 send_wat;
int                 wat_channel;
unsigned long       wat_bg_pixel;
XID                 wat_parent;
unsigned short      wat_port;
XID                 wat_id;
XWindowAttributes   wats;

int                 get_gc;
int                 send_gc;
int                 gc_channel;
XID                 gc_id;
unsigned short      gc_port;

/* Flags to close down a channel */
int             pr_close_channel;
int             pr_close_client;

/* Array to hold client id being distributed on each channel */
int             clients[MAX_CHANNELS];

/* Default gcs and root ids for all channels */
unsigned long       source_default_gc[MAX_CHANNELS];
unsigned long       source_root[MAX_CHANNELS];

/* Storage for current local gc state information */
struct GCWIN        gcwin[MAX_GCS];

/* Storage for current local window state information */
struct WIN          win[MAX_WINS];

/* X Protocol buffer */
int             client[XBUFFERNUM];/* client number         */
int             len[XBUFFERNUM];    /* length of x protocol pakt*/
```

```
    Window          window;             /* window to start sending  */
    unsigned char   xbuffer[XBUFFERNUM][XBUFFERSIZE];

    /* Host name for Central Distribution Manager */
    char                management_host[HOSTNAMLEN];

#ifdef PROFILE
    /* Flag to let multicast know to get rid of shared memory id */
    int wantToExit;

    /* Array to accumulate frequency of requests for each client */
    int pArray[ MAX_REQUESTS ][ MAX_CLIENTS];

    /* Arrays to accumulate packet lengths and -counts */
    unsigned long accumLen[ MAX_CLIENTS ];
    unsigned long currLen[ MAX_CLIENTS ];
#endif
);
```

```
/*
 * File         : smtypes.h
 */

/*
 * Structure to hold mapped source to destination
 * XIDs such as: colormaps, gcs, and windows.
 */
#define XBUFFERSIZE         2048        /* max x protocol packet    */
#define XBUFFERNUM          24
#define MAX_CLIENTS         20
#define MAX_XIDS            100
#define MAX_SCREENS         3
#define WRITEFD             1
#define READFD              0

#ifdef PROFILE
#define lastRequest X_NoOperation
#define MAX_REQUESTS (lastRequest+1)
#endif

/* ID types for protocol handling */
#define    WINDOW_TYPE      1
#define    GC_TYPE          2
#define    COLORMAP_TYPE    3
#define    FONT_TYPE        4
#define    CURSOR_TYPE      5
#define    PIXMAP_TYPE      6

/* OPERATION  for protocol handling */
#define    CREATE_ID        1
#define    MAP_ID           2

typedef struct XIDmap {
   XID              source;
   XID              dest;
   int              client;
   int              type;
   int              created;
   int              mapped;
   GC               gc;
   struct XIDmap    *next;
};

typedef struct _XHeader {
       int          client;
       int          len;
};

typedef struct Xpacket {
   struct _XHeader header;
   unsigned char   buffer[XBUFFERSIZE];
```

```
};

typedef struct sembuff {
   short        sem_num;
   short        sem_op;
   short        sem_flg;
};
```

```
/*
 * File         : xdefs.h
 */

char   XFunctName[][80] = {
" X_UndefinedRequest",
" X_CreateWindow",
" X_ChangeWindowAttributes",
" X_GetWindowAttributes",
" X_DestroyWindow",
" X_DestroySubwindows",
" X_ChangeSaveSet",
" X_ReparentWindow",
" X_MapWindow",
" X_MapSubwindows",
" X_UnmapWindow",
" X_UnmapSubwindows",
" X_ConfigureWindow",
" X_CirculateWindow",
" X_GetGeometry",
" X_QueryTree",
" X_InternAtom",
" X_GetAtomName",
" X_ChangeProperty",
" X_DeleteProperty",
" X_GetProperty",
" X_ListProperties",
" X_SetSelectionOwner",
" X_GetSelectionOwner",
" X_ConvertSelection",
" X_SendEvent",
" X_GrabPointer",
" X_UngrabPointer",
" X_GrabButton",
" X_UngrabButton",
" X_ChangeActivePointerGrab",
" X_GrabKeyboard",
" X_UngrabKeyboard",
" X_GrabKey",
" X_UngrabKey",
" X_AllowEvents",
" X_GrabServer",
" X_UngrabServer",
" X_QueryPointer",
" X_GetMotionEvents",
" X_TranslateCoords",
" X_WarpPointer",
" X_SetInputFocus",
" X_GetInputFocus",
" X_QueryKeymap",
" X_OpenFont",
" X_CloseFont",
```

```
" X_QueryFont",
" X_QueryTextExtents",
" X_ListFonts",
" X_ListFontsWithInfo",
" X_SetFontPath",
" X_GetFontPath",
" X_CreatePixmap",
" X_FreePixmap",
" X_CreateGC",
" X_ChangeGC",
" X_CopyGC",
" X_SetDashes",
" X_SetClipRectangles",
" X_FreeGC",
" X_ClearArea",
" X_CopyArea",
" X_CopyPlane",
" X_PolyPoint",
" X_PolyLine",
" X_PolySegment",
" X_PolyRectangle",
" X_PolyArc",
" X_FillPoly",
" X_PolyFillRectangle",
" X_PolyFillArc",
" X_PutImage",
" X_GetImage",
" X_PolyText8",
" X_PolyText16",
" X_ImageText8",
" X_ImageText16",
" X_CreateColormap",
" X_FreeColormap",
" X_CopyColormapAndFree",
" X_InstallColormap",
" X_UninstallColormap",
" X_ListInstalledColormaps",
" X_AllocColor",
" X_AllocNamedColor",
" X_AllocColorCells",
" X_AllocColorPlanes",
" X_FreeColors",
" X_StoreColors",
" X_StoreNamedColor",
" X_QueryColors",
" X_LookupColor",
" X_CreateCursor",
" X_CreateGlyphCursor",
" X_FreeCursor",
" X_RecolorCursor",
" X_QueryBestSize",
" X_QueryExtension",
```

```
" X_ListExtensions",
" X_ChangeKeyboardMapping",
" X_GetKeyboardMapping",
" X_ChangeKeyboardControl",
" X_GetKeyboardControl",
" X_Bell",
" X_ChangePointerControl",
" X_GetPointerControl",
" X_SetScreenSaver",
" X_GetScreenSaver",
" X_ChangeHosts",
" X_ListHosts",
" X_SetAccessControl",
" X_SetCloseDownMode",
" X_KillClient",
" X_RotateProperties",
" X_ForceScreenSaver",
" X_SetPointerMapping",
" X_GetPointerMapping",
" X_SetModifierMapping",
" X_GetModifierMapping",
" X_NoOperation" };
```

```
/*
 * File         : ds.bm
 */

#define ds_width 32
#define ds_height 32
#define ds_x_hot -1
#define ds_y_hot -1
static char ds_bits[] = {
   0xff, 0xff, 0xff, 0xff, 0x01, 0x00, 0x00, 0x80, 0xfd, 0xff, 0xff, 0xbf,
   0xfd, 0xff, 0xff, 0xbf, 0xfd, 0xff, 0xff, 0xbf, 0xfd, 0xff, 0xff, 0xbf,
   0xfd, 0x7f, 0xf8, 0xbf, 0xfd, 0x3f, 0xf0, 0xbf, 0xfd, 0x1f, 0xe0, 0xbf,
   0xfd, 0x8f, 0xc7, 0xbf, 0x7d, 0xc7, 0x8f, 0xbf, 0x7d, 0xe2, 0x1f, 0xbf,
   0x7d, 0xf0, 0x3f, 0xbe, 0x7d, 0xf8, 0xff, 0xbf, 0x7d, 0xf0, 0x01, 0xb0,
   0x7d, 0xe0, 0xfd, 0xb7, 0xfd, 0xff, 0x85, 0xb7, 0x0d, 0x80, 0xb5, 0xb7,
   0xed, 0xbf, 0xb5, 0xb4, 0x2d, 0xbc, 0xb5, 0xb5, 0xad, 0xbd, 0x85, 0xb5,
   0xad, 0xa5, 0xfd, 0xb5, 0xad, 0xad, 0x0d, 0xb4, 0x2d, 0xac, 0xfd, 0xb7,
   0xed, 0xaf, 0x01, 0xb0, 0x6d, 0xa0, 0xff, 0xbf, 0xed, 0xbf, 0xff, 0xbf,
   0x0d, 0x80, 0xff, 0xbf, 0xfd, 0xff, 0xff, 0xbf, 0xfd, 0xff, 0xff, 0xbf,
   0x01, 0x00, 0x00, 0x80, 0xff, 0xff, 0xff, 0xff};
```

```
/*
 * File         : dsb.bm
 */

#define dsb_width 5
#define dsb_height 5
#define dsb_x_hot -1
#define dsb_y_hot -1
static char dsb_bits[] = {
   0x03, 0x07, 0x0e, 0x1c, 0x18};
```

```
/*
 * File         : pr.bm
 */

#define pr_width 32
#define pr_height 32
static char pr_bits[] = {
   0x00, 0x00, 0x00, 0x00, 0xfc, 0xff, 0xff, 0x3f, 0xfe, 0xff, 0xff, 0x7f,
   0xfe, 0xff, 0xff, 0x7f, 0xfe, 0xff, 0xff, 0x7f, 0x1e, 0x00, 0x00, 0x78,
   0x1e, 0x00, 0x00, 0x78, 0x1e, 0x00, 0x00, 0x78, 0x1e, 0x00, 0x00, 0x78,
   0x1e, 0x00, 0x00, 0x78, 0xde, 0x7f, 0x00, 0x78, 0x5e, 0x40, 0x10, 0x78,
   0x5e, 0x41, 0x18, 0x78, 0x5e, 0x51, 0xfc, 0x7b, 0x5e, 0x55, 0xfe, 0x7b,
   0x5e, 0x55, 0xff, 0x7b, 0x5e, 0x55, 0xfe, 0x7b, 0x5e, 0x55, 0xfc, 0x7b,
   0x5e, 0x40, 0x18, 0x78, 0xde, 0x7f, 0x10, 0x78, 0x1e, 0x00, 0x00, 0x78,
   0x1e, 0x00, 0x00, 0x78, 0x1e, 0x00, 0x00, 0x78, 0x1e, 0x00, 0x00, 0x78,
   0x1e, 0x00, 0x00, 0x78, 0x1e, 0x00, 0x00, 0x78, 0x1e, 0x00, 0x00, 0x78,
   0xfe, 0xff, 0xff, 0x7f, 0xfe, 0xff, 0xff, 0x7f, 0xfe, 0xff, 0xff, 0x7f,
   0xfc, 0xff, 0xff, 0x3f, 0x00, 0x00, 0x00, 0x00};
```

```
/*
 * File         : cs.bm
 */

#define sc_width 32
#define sc_height 32
#define sc_x_hot -1
#define sc_y_hot -1
static char sc_bits[] = {
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
   0x7f, 0x00, 0x00, 0xfe, 0x7f, 0x00, 0x00, 0xfe, 0x7f, 0x00, 0x00, 0xfe,
   0x7f, 0x00, 0x00, 0xfe, 0x7f, 0x00, 0x00, 0xfe, 0x7f, 0x00, 0x00, 0xfe,
   0x7f, 0x00, 0x00, 0xfe, 0x7f, 0xaa, 0xaa, 0xfe, 0x7f, 0x55, 0x55, 0xfe,
   0x7f, 0xaa, 0xaa, 0xfe, 0x7f, 0x55, 0x55, 0xfe, 0x7f, 0xaa, 0xaa, 0xfe,
   0x7f, 0x00, 0x00, 0xfe, 0x7f, 0xdb, 0xb6, 0xfe, 0x7f, 0xb6, 0x6d, 0xfe,
   0x7f, 0x6d, 0xdb, 0xfe, 0x7f, 0xdb, 0xb6, 0xfe, 0x7f, 0xb6, 0x6d, 0xfe,
   0x7f, 0x00, 0x00, 0xfe, 0x7f, 0xff, 0xff, 0xfe, 0x7f, 0xff, 0xff, 0xfe,
   0x7f, 0xff, 0xff, 0xfe, 0x7f, 0xff, 0xff, 0xfe, 0x7f, 0xff, 0xff, 0xfe,
   0x7f, 0x00, 0x00, 0xfe, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff};
```