CERES Software Bulletin 95-11

Link Editing Basics, September 5, 1995

1.0 Purpose
To provide programmers with a simplified, quick overview and reference of how libraries are searched during compile, linking, and run times.

2.0 Originator
Marsha Sherland (m.m.sherland@larc.nasa.gov)

3.0 Description
Linking programs which rely on symbol resolution from libraries other than those supplied with the primary programming language in which it was written can be confusing and frustrating. The rules that the compilers and linkers follow for such symbol resolution are outlined here in a simplified form for reference. For a more thorough discussion of this subject, see the man page for 'ld' and the AnswerBook discussion in the "Utilities and Libraries" section.

When you compile and link a program, the output can be either object code or executable code. The executable code will be complete or incomplete depending on how symbols are resolved, and runnable or non-runnable depending on the existance of an initial entry point. The code is complete if it does not need symbols during run-time; otherwise, it is incomplete. The code is runnable if it contains an initial entry point; otherwise, it is non-runnable. The actual code plus the options on the command line for the compiler or the 'ld' command will determine which type of output is produced.

The command line for compiling and/or linking specifies a list of code in source or object form and can specify libraries and directories to search for unresolved symbol resolution. Source or object code is processed in the order specified on the command line. Each piece processed is concatenated to the last to create an executable file, a.out by default. As each piece of code is processed, any unresolved symbols are resolved in some way if possible.. If static linking is in effect, then the resolution produces code concatenated directly into the executable file. If dynamic linking is in effect, then the resolution is handled by marking the executable file with information so that the symbol can be resolved or shared during run-time. With dynamic linking, the symbol resolution information actually contained in the executable file varies depending on how the appropriate library for that symbol was found. If the appropriate library was found via the -L options or found in one of the standard libraries, then directory and library information is noted. If the appropriate library was found due to the LD_LIBRARY_PATH variable, then only library information is noted. At run-time, if only library information has been noted, then the *current* LD_LIBRARY_PATH directories are searched. This creates a very flexible, controllable environment for the programmer but adds a layer of complexity that can easily be misunderstood. The two flowcharts on the next 2 pages gives a *simplified* overview of this process.

Note that the command line can specify some modules to be linked statically and other modules

to be linked dynamically by using several -B options on the line, thus toggling the -B option from static to dynamic as desired.

Some definitions are included here for clarity.

.a file:

Archive format library file. Modules are often in object code. Library names are of the form lib<x>.a where <x> is the name of the library as referred to by the -l option. These libraries must be used when code is being statically linked.

a.out file:

File in executable format. May or may not be complete or runnable. It is complete if all symbols are resolved statically; otherwise it is incomplete. It is runnable if it has an initial entry point; otherwise it is nonrunnable.

dynamically linked file:

An executable in which some symbols remain unresolved until run-time. Specified on the command line with the -B (-Bdynamic) option. See ld.so.

ld:

The static link editor. 'ld' combines a variety of object files to produce an exeutable a.out formatted file.

ld.so:

The dynamic link editor. 'ld.so' is used when running an executable that is not complete and therefore needs symbol resolution or code sharing. Performs link editing operations that were not done by 'ld'. Maps in and binds the required shared object file.

.o file:

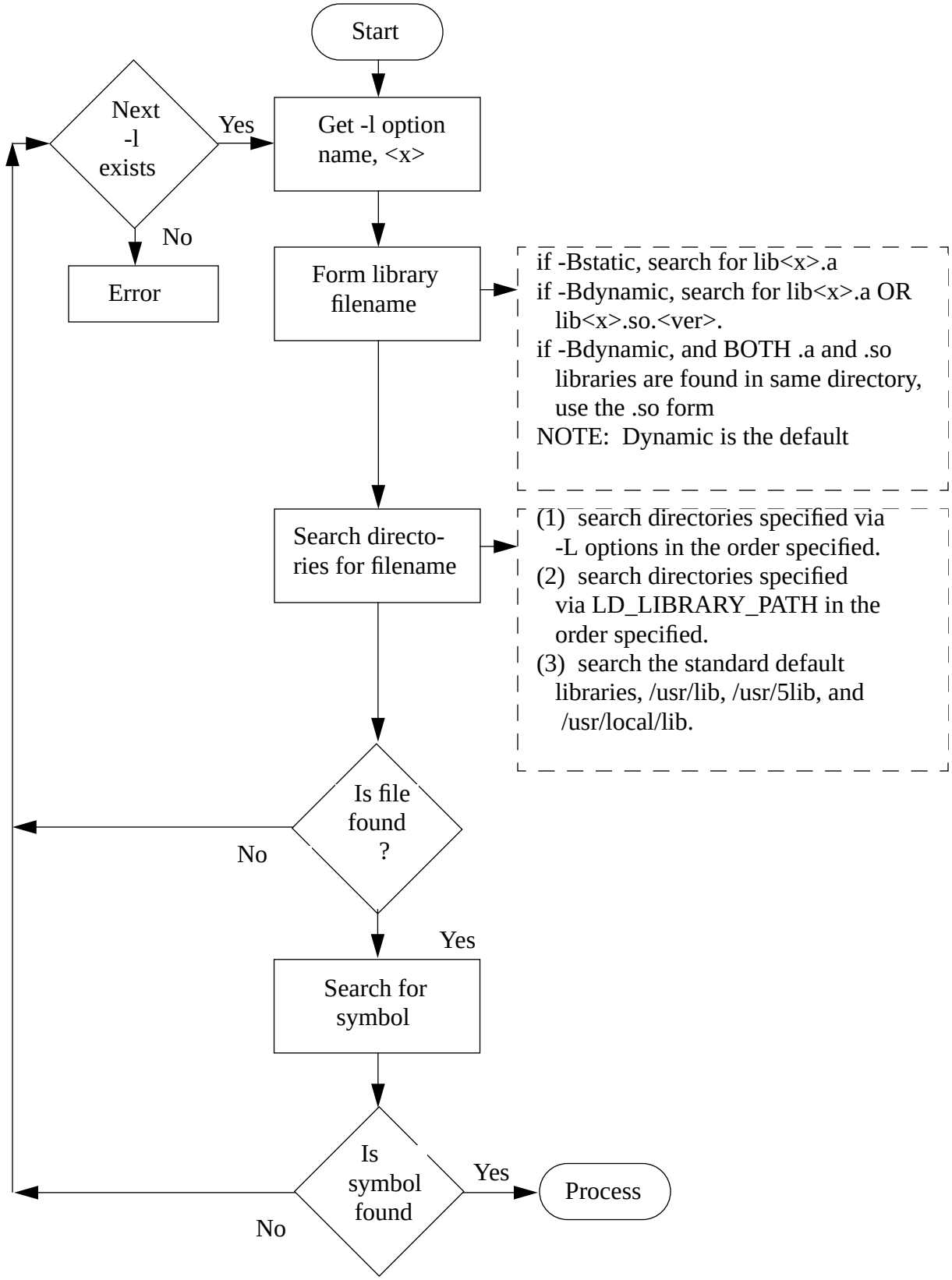
Object code. Produced by a compiler command (f90, acc, etc.) with the -C option. Also produced by a link edit command (ld) with a -r option.

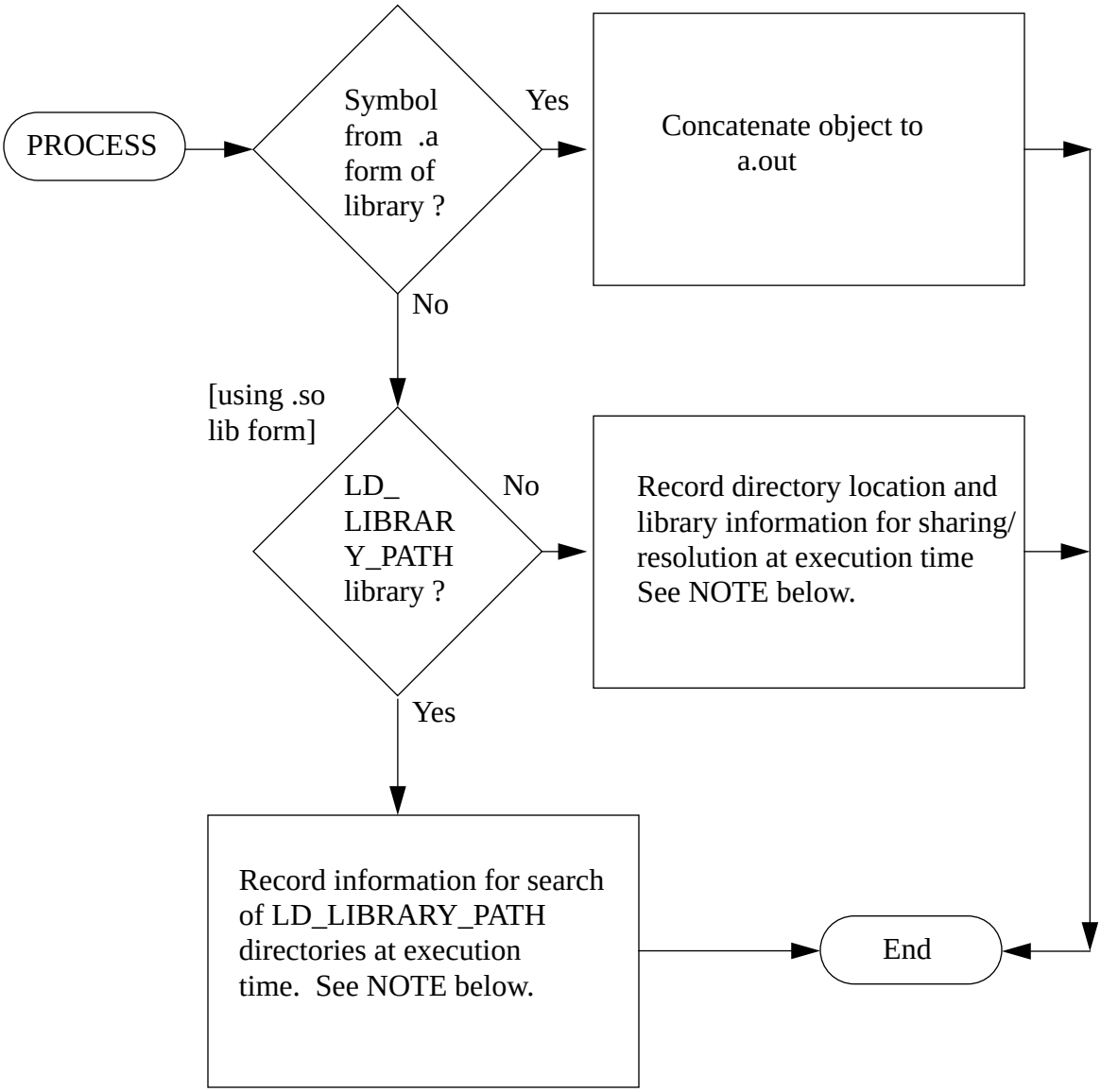
.so files:

Shared object library. Modules are in a.out format but are not runnable since they lack an initial entry point. Library names are of the form lib<x>.so.<ver> where <x> is the name of the library as referred to in the -l option, and <ver> is a major/minor version number combination.

statically linked file:

An executable which has all processed object modules concatenated directly into the executable file. Specified on the command line with the -B (-Bstatic) option.


Start
Next -l exists
Yes
No
Error
Get -l option name, <x>
Form library filename
if -Bstatic, search for lib<x>.a
if -Bdynamic, search for lib<x>.a OR lib<x>.so.<ver>.
if -Bdynamic, and BOTH .a and .so libraries are found in same directory, use the .so form
NOTE: Dynamic is the default
Search directo-ries for filename
(1) search directories specified via -L options in the order specified.
(2) search directories specified via LD_LIBRARY_PATH in the order specified.
(3) search the standard default libraries, /usr/lib, /usr/5lib, and /usr/local/lib.
Is file found ?
No
Yes
Search for symbol
Is symbol found
Yes
No
Process



NOTE: An incomplete but runnable a.out file will use ld.so to resolve and bind shared code. For any unresolved symbols that do NOT have location and library information, ld.so will FIRST check the directories in LD_LIBRARY_ PATH, then the directories listed with the -L option, and then the standard default libraries, /usr/lib, /usr/5lib, and /usr/local/lib. NOTE: this is a DIFFERENT order of search from the order used during the linking process.