***LBNL-49659***
***Checkpoint Requirements-V1-7***
***February 26, 2002***

# *Requirements for Linux Checkpoint/Restart*

Jason Duell
Paul Hargrove
Eric Roman

***LBNL-49659***
***Checkpoint Requirements-V1-7***
***February 26, 2002***

# *Requirements for Linux Checkpoint/Restart*

Jason Duell
Paul Hargrove
Eric Roman

***Approved for public release***

***Lawrence Berkeley National Laboratory***
***Berkeley, CA 94720***

This work was supported by the Director, Office of Science, Office of Advanced Scientific Computing Research, U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

DISCLAIMER

## Revision History

**Revision 01 (February 20, 2002)**
First externally released version.

**Revision 02 (February 26, 2002)**
Added corrections from LBNL reviewers. Added features table to S2.

## *Contents*

# Requirements for Linux Checkpoint/Restart

## S1. Introduction

### S1.1. Purpose

This document has 4 main objectives:

- Describe data to be saved and restored during checkpoint/restart
- Describe how checkpoint/restart is used within the context of the Scalable Systems environment, and MPI applications
- Identify issues for a checkpoint/restart implementation
- Sketch the architecture of a checkpoint/restart implementation

#### S1.1.1. Audience

This document is written for:

- Developers within the *Checkpoint/Restart Team*, who need to understand the scope of a checkpoint/restart implementation
- *Scalable Systems participants*, who use checkpoint/restart with other Scalable Systems software
- *Developers* of checkpoint/restart for other operating systems, looking for baseline requirements from which they can develop their own checkpoint/ restart, and
- *Users* of the checkpoint/restart interfaces.

### S1.2. Scope

#### S1.2.1. Name

This document describes requirements pertaining to *Checkpoint/Restart for Linux* (abbr. CR). CR is a component SciDAC Scalable Systems Software Checkpoint/Restart Project.

#### S1.2.2. Boundary

The Scalable Systems Checkpoint/Restart Project seeks to provide checkpoint/restart for MPI applications running on Linux clusters under the Scalable Systems Software. This work is broken down into 4 main areas:

- Checkpoint/Restart for Linux (CR)
- Checkpointable MPI Libraries
- Resource Management Interface to Checkpoint/Restart
- Development of Process Management Interfaces

This document describes Checkpoint/Restart for Linux (CR). The 3 remaining areas are described elsewhere (see section S1.4. for references).

CR is an operating system component that provides checkpoint and restart services to processes. A checkpoint service takes a running application and saves it to a file. A restart service reads the data saved during a checkpoint, restores the state of the application, and continues to execute the application. The application, in most cases, remains unaware that there was any interruption to its execution.

CR is designed for scientific computing applications. These applications may be divided into two types: serial applications and parallel applications. For the purposes of this document, a serial application is an application that executes under a single instance of a Linux kernel. Creating a valid context fileset for a serial application require no coordination with processes running on remote nodes.

Most parallel applications are written using an MPI (Message Passing Interface) library. Checkpointing these applications requires modifications to the MPI library. The MPI library must be modified to coordinate the

execution of an application so that a consistent distributed checkpoint may be taken. The necessary modifications will be described in a separate document. (See S1.4. for references)

Full integration of checkpoint/restart into a resource management system requires interfaces to a scheduler, queue manager and process manager. These interfaces will be described in the Scalable Systems Software Specification (See S1.4. for references)

### S1.2.3. Goals

#### S1.2.3.1. Develop CR for parallel MPI jobs under Linux

The main goal of the Checkpoint/Restart project is to implement checkpoint/restart for Linux. This project's focus is on clusters used for high-end scientific computing.

#### S1.2.3.2. Integrate into Scalable Systems framework

Another goal is to add full support for checkpoint/restart to the interfaces and subsystems defined by the Scalable Systems Software ISIC.

#### S1.2.3.3. Document Checkpoint/Restart Design

A major goal of this project is to produce a fully documented design and implementation. Although production implementations of checkpoint/restart exist, there is little or no public documentation describing their internals. We will equip those seeking to implement checkpoint/restart for other operating systems with the knowledge they require to be successful.

#### S1.2.3.4. Higher utilization

Checkpoint/restart seeks to improve the utilization of cluster systems. We plan to integrate checkpoint/restart with scheduling systems optimized for checkpoint/restart. We also plan to write a set of job management utilities for fast system startup and shutdown.

#### S1.2.3.5. Transparent

We want checkpoint/restart to be completely transparent to applications programmers and end users. Most users should not need to change any of their application code to make their applications checkpointable.

Our kernel checkpoint/restart does not interact with application-implemented checkpoint/restart (see section S2.4.4. for more information.)

### S1.2.4. Scenarios for Checkpoint/Restart

#### S1.2.4.1. Scenarios from System Managers

**Large Systems Management:** Checkpoint/Restart is often listed as a requirement for supercomputer procurements. By implementing checkpoint/restart for Linux clusters, we hope to bring Linux clusters one step closer to full adoption in this demanding environment.

**Job Preemption:** Checkpoint/Restart is a mechanism to preempt running jobs. UNIX kernels suspend running jobs by simply removing them from the list of runnable processes. Virtual memory, and many other system resources, is not freed until the job exits. UNIX provides no way to free the virtual memory consumed by a process. Sites seeking to timeshare large jobs must provide large amounts of virtual memory. Checkpoint/restart provides a way around this limitation. Checkpoint/Restart allows the virtual memory allocated to a job to be completely freed to another job.

**Multiuser scheduling:** Multiuser, or gang, scheduling is the main scenario for job preemption on parallel systems. By preempting large jobs running in parallel, the scheduler may interleave the execution of shorter jobs with large running jobs.

**Multiplexed Batch Queues:** Some sites use checkpoint/restart as a means to multiplex their queue configurations. At these sites, queues are not always runnable. A large job queue may only be open at night, or on the weekends; jobs from smaller or shorter queues may only be favored during the day. During transitions between queue schedules, checkpoint/restart is used to save the work accomplished in the old queue, and restore

the work done in the new queue. This allows for a level of flexibility in queue scheduling not present at sites without checkpoint/restart.

**Process Migration:** Checkpoint/restart is a simple mechanism for process migration across nodes. Process migration is the act of transferring a UNIX process from one node to another. Process migration is sometimes used by existing systems to move jobs from slower to faster nodes. On other systems, the scheduler uses migration to pack running jobs into a more efficient topology.

#### S1.2.4.2. Scenarios from Applications Programmers

**Coverage:** Checkpoint/restart should be applicable to a broad range of applications. Ideally, checkpoint/restart would be available for every application running on a system.

**Transparent:** Checkpoint/restart should have little visibility to users or applications. Application execution should produce the same results whether or not the application was checkpointed.

**Effort required to make checkpointable applications:** Although conceptually very simple, checkpoint/restart is very difficult to implement. Applications may have special requirements that a kernel-level checkpoint/restart cannot address. Some effort may be required on the part of applications programmers to exploit checkpoint/restart. It is desirable to keep the involvement of applications programmers with checkpoint/restart to a minimum.

**Fault tolerance:** Although fault tolerant applications are not the main goal of this project, we recognize that fault tolerance roles do exist for checkpoint/restart. This implementation attempts to address simple fault tolerance requirements, but makes no claim to completeness in this area.

### S1.2.5. Impact

**Lower wait time**: The scheduler may use checkpoint/restart to preempt long running jobs, and allow shorter jobs to run in their place.

**System utilization**: Utilization is the fraction of available CPU time consumed by user applications. Checkpoint/restart improves system utilization by allowing greater flexibility in the ordering of jobs. Running jobs may be suspended and replaced with jobs currently in a wait queue.

**Increase available CPU time**: Checkpoint/restart increases the amount of CPU time deliverable to applications. Systems administrators may use checkpoint/restart to drain a system of running jobs. With checkpoint/restart, systems administrators don't have to wait for running jobs to terminate before performing a shutdown. The administrator simply checkpoints the jobs, reboots the system, and restarts the jobs. The jobs continue as though the system reboot had never occurred. This eliminates idle time spent in a long "queue draining" period.

## S1.3. Definitions, Acronyms, and Abbreviations

| | |
|---|---|
| **Application:** | Applications are what users run or the code that they develop. |
| **CLI:** | Abbreviation for command line interface. |
| **Context file:** | Output of a checkpoint call. This is the file where information necessary to restart a process is stored. |
| **CR:** | Abbreviation for Checkpoint/Restart for Linux. |
| **Node:** | A computer with memory accessible only to itself, running a single instance of the operating system kernel. |
| **Process:** | A POSIX process, as implemented by Linux. |
| **Resource:** | A data structure or other object within the kernel that must be obtained to restart a process or session. |
| **Restore:** | Read information from context fileset and act on the kernel so it is consistent with this information. Restore operations take place only while a process or session is restarted. Ideally, a restore operation does not expose changes in data, the checkpoint, or original process |

| | |
|---|---|
| | termination to the application programmer. |
| **Save:** | Write information to context fileset. Save operations take place only while a process is being checkpointed. |
| **Session:** | A POSIX session, as implemented by Linux. |
| **Slow system call:** | Defined by POSIX. Slow system calls are calls that can block for a very large amount of time, such as wait(), sleep(), select(), and pause(). |
| **Utilization:** | The fraction of available CPU time consumed by user applications |

### S1.4. References

1. CVS Repository. These files are available to Checkpoint/Restart developers with access to the CR CVS repository. Available as checkpoint-docs from FTG CVS server.
   - CR.TEST. Testing plan for CR
   - CR.PLAN. Project plan
   - CR.SURVEY. Evaluation survey
   - CR.SPEC. Detailed specification for CR
   - CR.MPI. Description of MPI
2. Checkpoint/Restart Web Page. http://www.nersc.gov/research/ftg/checkpoint/
3. Gropp, W., Snir, M., Nitzber, B., and Lusk, E. MPI: The Complete Reference. (MIT Press, 1998)
4. Scalable Systems Software Interface Specification. Not currently available. See http://www.csm.ornl.gov/scidac/ScalableSystems for more information.

### S1.5. Document Overview

The remainder of this document is divided into two sections: Section 2 provides a summary description of the kernel checkpoint/restart implementation. Section 3 delves into detailed requirements for the implementation.

Section S2.1. describes the main interfaces to CR. S2.2. describes the main functions that CR is expected to perform. S2.3. lists characteristics of the users of CR. S2.4. lists constraints imposed on the implementation.

Section S3.1. describes performance requirements on checkpoint/restart. S3.2. describes our supported platforms. S3.3. lists checkpointable applications. S3.4. describes standards conformance. S3.5. lists requirements for transparency. S3.6. describes the outside interfaces to CR. S3.7. and S3.8. describe the environment necessary for checkpoint/restart to function. S3.9. describes data structures and kernel semantics that must be dealt with.

## S2. Overall Description

### S2.1. Perspective

CR exists inside some overall framework for resource management. Typical applications are supported by checkpoint/restart without any intervention from the user or the application programmer. Applications with unique needs may use a special API to handle checkpoint signals. Systems programs, such as schedulers, manage checkpoint/restart operation through system calls. Operators invoke checkpoint/restart through command line interfaces. User libraries, such as MPI, interact with checkpoint/restart through APIs.

#### S2.1.1. System Call Interface

A system call interface provides the triggers for checkpoint/restart operations. The system call libraries must identify the processes or sessions that must be checkpointed, as well as the type of checkpointing to be performed. Through system-initiated checkpoints, a shell, or other utility, may invoke the checkpoint or restart system calls to request that a checkpoint of a given process or session be taken. Through application-initiated checkpoints, applications request that the kernel take a checkpoint of them.

#### S2.1.2. Application Programmers Interface

An API provides applications programmers with hooks to assist checkpoint/restart. These hooks allow the application to block off code sections where checkpoints are not permitted. The hooks also give applications a chance to respond to checkpoint/requests, and take appropriate action.

### S2.1.3. Command Line Interface

A command line interface to the checkpoint/restart system calls provides systems administrators and other users a means to request checkpoints of applications be taken.

### S2.1.4. MPI Library Interface

The MPI libraries require notification that checkpoints must take place. These notifications are necessary so that the MPI library may coordinate processes across nodes, so that consistent checkpoints are possible.

### S2.1.5. Scalable Systems Interface

The Scalable Systems components require ways to send checkpoint requests to checkpointable components. Scalable Systems components must also be made aware that checkpoint/restart is possible on a system. The scheduler requires a way to initiate a checkpoint or restart, and it requires a way to decide when to use checkpoint/restart. The queue manager provides users with interfaces to request that a checkpoint be taken, or request that a checkpointed job resume execution. A process manager provides interfaces to the checkpoint/restart functions in the underlying operating system.

NOTE: This document attempts to address the needs of the Scalable Systems Software. Since Scalable Systems Software is incomplete at this time, full interfaces to checkpoint/restart are described in other documents. See S1.4. for references.

## S2.2. Functions

### S2.2.1. Functional Requirements

#### S2.2.1.1. Checkpoint

The checkpoint function must take a running job and store all state information about the job into a context fileset. Checkpoint first notifies the job that a checkpoint is to occur. Checkpoint then stops the job, and creates the context file set. When finished checkpoint sends the job a signal to exit, or notifies the job that a checkpoint has just occurred.

#### S2.2.1.2. Restart

A restart operation restores a running job to the kernel by reading a context fileset and recreating the jobs state. After reacquiring any resources necessary for the job (PID, virtual memory, etc.), restart notifies the job that it has been restarted.

#### S2.2.1.3. Suspend

The suspend function stops a job's execution, without writing most of the operating system context to disk. Suspend is the main mechanism for preemption in situations where the overhead with a full checkpoint is too high, and job persistence across reboots is unnecessary. This function exists as a compromise between a simple SIGSTOP approach, and a full checkpoint.

#### S2.2.1.4. Resume

The resume function restores a suspended job to the system, and makes it runnable. Suspend is the logical counterpart to the resume function.

### S2.2.2. Application classes

Checkpoint/restart must consider four classes of application: serial applications, MPI processes, shell scripts, and interactive shells. These application classes differ by the amount of information and coordination necessary to checkpoint them.

#### S2.2.2.1. Simple serial applications

A simple serial application is one that has very little interaction with the operating system. A simple application is one that:

- does not use interprocess communication (sockets, pipes, named FIFOs, IPC)
- does not open files
- performs all I/O to the standard input, output and error streams
- runs as a single process

#### S2.2.2.2. MPI Processes

MPI processes are processes taking part in an MPI application. These processes may use sockets for communication with processes on other nodes. In other cases, they may communicate locally using shared memory. MPI processes may also use a high-speed network interface for communications. These high-speed interfaces typically use proprietary programming interfaces, and cannot easily be handled in a generic fashion. These processes may be treated as simple processes, with the caveat that their communications libraries must be reinitialized.

#### S2.2.2.3. Shell scripts

Shell scripts are shells running a script written to disk. These shells typically interact with the signal, terminal and session management functions of the kernel. Shell scripts are composed of multiple processes. Shell scripts often use pipes for communication between processes.

#### S2.2.2.4. Interactive shells

An interactive shell is a shell running under a controlling terminal.

### S2.2.3. Security

Checkpoint/restart must address concerns for system security. In no case should CR allow a user to gain privileges on a system. CR should allow processes to access only the resources accessible to the invoking process. CR should not grant additional privileges to restarted processes. See section S3.6.2, S3.9.3., and S3.9.8. for full requirements.

### S2.2.4. Altered context filesets

CR must assume that context filesets may be modified at any time. In some cases, these modifications may be benign, compromising no aspect of system security or process execution. In cases where process execution may be altered, CR should assume that modifications are correct, and make no attempt to compensate or correct program execution, except as directed in the context fileset.

### S2.2.4. Supported Applications

| Function | Full support | Partial support | No support |
|---|---|---|---|
| C and Fortran | X | | |
| Shell scripts | X | | |
| Interactive shells | X | | |
| Signal handling | X | | |
| Sockets | | | X |
| System V IPC | | | X |
| Interval timers | X | | |
| Statically linked | | X[1] | |
| Multithreaded applications | X | | |
| Processes and sessions | X | | |
| Pipes | X | | |
| Named pipes | X | | |
| System accounting | X | | |
| Open files | X | | |
| Mapped files | X | | |
| Block device files | | X[2] | |
| Asynchronous file I/O | | | X |
| /proc files | | X[3] | |
| Open-unlinked files | X | | |
| Locks | X | | |
| Pinned memory | X | | |
| Mprotect()'d memory | X | | |
| Ptrace'd applications | | | X |

1. Only supported if the libraries that the original binary was linked against are checkpointable.
2. Only support for /dev/null, /dev/zero is provided.
3. References to /proc/self/* are restored, as well as references to /proc/PID, where PID is the PID of another process within the session being checkpointed.

## S2.3. Users

This section describes characteristics of the users of checkpoint/restart. We identify five types of user: Applications programmers, end users, systems operators, MPI library developers and systems programmers.

### S2.3.1. Applications Programmers

Applications programmers use the checkpoint/restart facilities to make their applications checkpointable. Applications programmers are often unfamiliar with the particulars of the Linux operating system, and may not be familiar with UNIX systems programming in general.

### S2.3.2. End Users

End users of checkpoint/restart may be involved with some application development, but are often scientists with no programming experience. These users will use checkpoint/restart to manage the jobs they have running on a system.

### S2.3.3. Systems Operators

Systems operators are concerned with the overall operations of the queue system. They use checkpoint/restart inside shell scripts and are often responsible for integrating checkpoint/restart into new environments. They are concerned with the maintenance of the checkpoint/restart systems.

### S2.3.4. MPI Library Developers

MPI library developers interact with checkpoint/restart by making their MPI libraries functional in the checkpoint/restart environment. MPI developers are familiar with UNIX system programming. One of their goals is to hide checkpoint/restart semantics from applications programmers as much as possible.

### S2.3.5. Systems Programmers

Systems programmers interact with checkpoint/restart to write utilities such as shell functions for checkpoint/restart, batch systems, and other applications. Systems programmers are very familiar with UNIX system programming. They are looking to add support for checkpoint/restart to new batch systems, and to use checkpoint/restart in new ways for system functionality.

## S2.4. Constraints

This subsection describes some of the limitations of checkpoint/restart, and other needs that we were unable to express in terms of formal requirements.

### S2.4.1. No sockets

Due to the difficulty of checkpointing network sockets, CR will make no attempt to checkpoint applications running communicating through sockets. UNIX domain sockets and Internet sockets are outside of the scope of this implementation. Checkpoint for these applications will be provided through application callbacks described in S3.6.7.

### S2.4.2. MPI Applications

CR must checkpoint and restart applications using an MPI interface.

NOTE: Since modifications to the underlying MPI libraries are usually necessary, applications must be linked with a checkpointable MPI library. Full requirements for checkpointable MPI are described in a separate document. This document describes the options available for checkpointing MPI-1 and MPI-2 applications. See S1.4. for references.

### S2.4.3. No POSIX 1003.1m

POSIX started work on a standard checkpoint/restart interface, POSIX 1003.1m. POSIX never finalized 1003.1m. The group assigned to 1003.1m abandoned the work due to manpower constraints. Due to the status of this standard, compliance with POSIX 1003.1m is not a requirement for CR.

### S2.4.4. No application implemented checkpoint

Some applications implement their own checkpoint/restart. CR has no role in checkpointing these applications.

### S2.4.5. ptrace support

ptrace calls allow a process to monitor the activity of another process. CR will not attempt to cover applications using ptrace. Behavior for ptrace is undefined, but may be added in a later release.

### S2.4.6. No System V IPC (Messages & semaphores)

Although System V IPC is sometimes a useful IPC mechanism, CR will not attempt to address applications using System V message queues, semaphores, or shared memory. This feature may be added if deemed necessary.

### S2.4.7. No shared file system assumed

In a process migration role, CR must make certain assumptions about the nature of the underlying file system. CR should assume that the name, path, and contents of files remain constant when moving processes between nodes.
CR should not assume that inode numbers are constant. CR should not assume that file systems are shared between the restarting and checkpointing nodes. CR should not assume that a file system (e.g. NFS or GFS) is shared between nodes participating in a migration.

#### S2.4.8. Limited support for statically linked binaries

CR should allow statically linked binaries to be checkpointed. It may be impossible for CR to restart an arbitrary statically linked binary, since the binary may have been linked against a non-checkpointable library. For example, an MPI application may be linked with a version of MPI that does not have support for CR coordination.

#### S2.4.9. Limited support for files in /proc/pid/

CR should allow processes with files open in /proc/pid to be restartable. The restarted /proc/pid files should exist within the scope of the processes checkpointed in a single call to the checkpoint system call.

#### S2.4.10. Limited support for block device files

It is not necessary for CR to restart processes accessing arbitrary block device files. Such applications are rare (examples are databases and file system integrity checks), and are generally not scientific applications. Support for certain block device files will be added in special cases (e.g. /dev/zero, /dev/null).

#### S2.4.11. Some support for kernel version change

CR should allow processes to migrate through changes in kernel versions. CR is not required to support arbitrary changes in kernel versions (e.g. 2.5.2 to 0.99). CR should permit context files to be valid through changes in the kernel patchlevel (e.g. 2.4.3 to 2.4.4), or through recompiles of the same kernel version.

#### S2.4.12. No asynchronous I/O

CR will not attempt to cover applications using asynchronous I/O.

## S3. Specific Requirements

This section lists requirements for the Checkpoint/Restart System.

In this section, mandatory requirements for checkpoint/restart are indicated by the phrase: “CR shall.” SHOULD is used to denote constraints, negative requirements and nonessential requirements. WILL is used to describe statements of fact.

### S3.1. Performance Requirements

#### S3.1.1. Preemption overhead

Since checkpoint/restart is intended mainly for process preemption, it must take care not to introduce so much overhead that the original utilization or throughput benefits are lost.

**S3.1.1.1. Time to take a checkpoint:** The time to take a checkpoint should be largely dominated by the time to write the context fileset to disk.

**S3.1.1.2. Time to start a checkpoint:** The time to start a checkpoint, i.e. the time between invocation of the checkpoint system call, and the completion of the applications checkpoint handlers (see S3.6.4. ) should be kept small.

**S3.1.1.3. Time to restart:** A restart operation should be dominated by the time to read the context file set from disk.

**S3.1.1.4. Time to suspend:** A suspend operation should be dominated by the time to arrange for the process address space to be written to disk.

**S3.1.1.5. Time to resume:** A resume operation should be dominated by the time to reload the process address space from disk.

#### S3.1.2. Runtime overhead for processes

CR should not introduce significant overhead to processes while they execute.

#### S3.1.3. Size of context fileset

CR should minimize the size of the context fileset. CR should attempt to optimize for shared address spaces and identical mapped regions by saving only a single copy of these regions to disk.

### S3.2. Supported Platforms

**S3.2.1. Linux:** CR shall be implemented for the Linux 2.4 kernel.

#### S3.2.2. Architectures

**S3.2.2.1. Intel IA/32:** CR shall support processes running on the Intel IA/32 processor.
**S3.2.2.2. Intel IA/64:** CR shall support processes running on the Intel IA/64 processor.
**S3.2.2.3. Alpha:** CR shall support processes on the Alpha processor.

#### S3.2.3. SMP Support

**S3.2.3.1. SMP:** CR shall support processes running on symmetric multiprocessor kernels.
**S3.2.3.2. Single:** CR shall support processes running on single CPU kernels.

### S3.3. Applications Coverage

This section describes requirements for application coverage.

#### S3.3.1. C

CR shall checkpoint and restart applications written in C.

#### S3.3.2. Fortran

CR shall checkpoint and restart applications written in Fortran.

#### S3.3.3. Interpreted Scripts

CR shall checkpoint and restart applications written in the common scripting languages.

**S3.3.3.1.** CR should not interfere with the job control facilities of an interpreter or a scripting language.

**S3.3.3.2.** CR shall checkpoint and restart applications written in the following scripting languages:

a) Bourne Shell (/bin/sh)
b) Korn Shell (/bin/ksh)
c) Bourne Again Shell (/bin/bash)
d) C Shell (/bin/csh)
e) TCSH (/bin/tcsh)
f) Perl
g) Python
h) Tcl
i) Java

#### S3.3.4. MPI Applications

CR shall checkpoint and restart applications that make use of an MPI Library. (NOTE: Full requirements for checkpointable MPI are described in a separate document. See the references in S1.4.)

**S3.3.5. Interactive Sessions:** CR shall checkpoint and restart interactive UNIX sessions (i.e. Sessions with a controlling TTY.)

#### S3.3.6. Multi-thread / Multi-process Applications

CR shall checkpoint and restart applications consisting of multiple threads of control, started through the common mechanisms (fork, vfork and pthread_create).

### S3.4. Relevant Standards

#### S3.4.1. Scalable Systems Software

CR shall operate under the Scalable Systems Software suite.

### S3.5. Transparency

CR should not require large changes to application source code to make an application checkpointable. In most cases, no changes to application source code will be necessary.

#### S3.5.1. No redesign of applications

CR shall be compatible with the programming models used by existing applications.

#### S3.5.2. Unmodified source code

CR shall be compatible with existing application source code.

**S3.5.2.1.** CR shall not require significant changes to the source of existing applications.
**S3.5.2.2.** CR shall document changes necessary to make applications checkpointable.

#### S3.5.3. Object file support

CR shall be compatible with existing object files (i.e. built without support for checkpoint/restart).

#### S3.5.4. Same linked libraries

CR shall be compatible with existing dynamically linked binaries (i.e. built without support for checkpoint/restart.

#### S3.5.5. No changes in semantics

CR shall preserve the semantics provided by existing libraries.

### S3.6. External Interfaces

#### S3.6.1. Command line interface

**S3.6.1.1.** CR shall provide a command to checkpoint a process.
**S3.6.1.2.** CR shall provide a command to restart a process.
**S3.6.1.3.** CR shall provide a command to checkpoint a session.
**S3.6.1.4.** CR shall provide a command to restart a session.
**S3.6.1.5**. CR shall provide a command to suspend a process.
**S3.6.1.6.** CR shall provide a command to resume a process.
**S3.6.1.7.** CR shall provide a command to suspend a session.
**S3.6.1.8.** CR shall provide a command to resume a session.
**S3.6.3.9.** CR shall provide a command to determine whether a processes or sessions described in a context fileset are not restartable.
**S3.6.3.10**. CR shall provide a command to list the contents of a context fileset to standard output.
**S3.6.3.11.** If any of the commands provided in this section are unable to checkpoint or restart the appropriate object, CR shall return an error code indicating the reason for failure.
**S3.6.3.11.** If any of the commands provided in this section are unable to checkpoint or restart the appropriate object, CR shall return an error code indicating the reason for failure.

#### S3.6.2. Security

**S3.6.2.1. Privileges for checkpoint**: CR shall ensure that only a user with permission to send a signal to a process may successfully initiate a checkpoint request.
**S3.6.2.2. Privileges for restart:** CR shall ensure that only a user with privileges to read the context fileset and signal a process described in the fileset may successfully restart a process.
**S3.6.2.3. Privileges for suspend**: CR shall ensure that only a user with permission to send a signal to a process may successfully suspend a requested process or session.
**S3.6.2.4. Privileges for resume**: CR shall ensure that only a user with permission to send a signal to a suspended process may successfully resume a requested process or session.
**S3.6.2.5. Permissions on context fileset:** CR shall ensure that context files are readable only by the user of the checkpointed process and writable by no one.
**S3.6.2.6. Default ownership:** CR shall ensure that context files are owned by the user of the checkpointed process, if a process is checkpointed.
**S3.6.2.7.** CR shall ensure that context files shall be owned by the user of the session leader, if a session is checkpointed.

### S3.6.3. Context file interface

**S3.6.3.1.** CR shall provide a C-callable API interface to read the contents of a context fileset.
**S3.6.3.2.** CR shall provide a C-callable API interface to modify the contents of a context fileset.
**S3.6.3.3.** CR shall provide access to all objects described in section S3.9. that are saved in a fileset.
**S3.6.3.4.** CR shall provide access to the following additional information:

a) The time that the checkpoint was taken
b) The user who initiated the checkpoint
c) Output of the uname system call at the time the checkpoint was taken
d) The architecture (Intel, Alpha, etc) that the checkpoint was taken on

### S3.6.4. System call interface

**S3.6.4.1. System initiated checkpoints:** CR shall provide a system call to initiate a checkpoint of a given process or session.
**S3.6.4.2. checkpoint-terminate:** CR shall send the checkpointed process a specified signal after the checkpoint has taken place, if directed to do so by the process invoking the checkpoint system call.
**S3.6.4.2.1. checkpoint-continue:** CR shall allow a process to continue execution after a checkpoint has occurred, unless directed to terminate the process.
**S3.6.4.3.** CR shall allow a process to checkpoint itself.
**S3.6.4.4.** CR shall allow a process to checkpoint its own session.
**S3.6.4.5.** CR shall provide a system call to initiate a restart from a context fileset.
**S3.6.4.5.1.** If a resource required by the process is unavailable, or in use at the time of resource, CR shall abort the restart operation and return an error code indicating the reason for the failure.
**S3.6.4.5.2. Context file validation:** CR shall check context filesets for validity during a restart. If CR is passed an invalid input file, CR shall return an error code indicating the reason for failure.
**S3.6.4.6.** CR shall ensure that a process waiting (through one of the wait system calls) receives notification that the process has exited, if the checkpointed process terminates after checkpoint.

**S3.6.7. Application callbacks:** CR shall provide processes with notifications of checkpoint, resume and restart events.
**S3.6.7.1.** CR shall notify processes when a checkpoint is to occur (before the kernel takes a checkpoint) to allow the processes to prepare itself accordingly.
**S3.6.7.2.** CR shall notify processes that they are continuing execution immediately after a checkpoint has completed.
**S3.6.7.3.** CR shall notify processes when a restart has completed.
**S3.6.7.4.** CR shall allow applications to block notification in certain critical sections of code to prevent a checkpoint from being taken during critical sections of code.
**S3.6.7.5.** CR shall allow applications to re-enable checkpointing so that checkpoints may be taken upon leaving a critical section of code.

**S3.6.8. Control over checkpoint behavior**: CR shall provide an interface to control behavior for checkpointable objects within the operating system.
**S3.6.8.1.** CR shall have controls for behavior of the following objects during checkpoint:
files
shared libraries
**S3.6.8.2.** CR shall accept parameters that specify appropriate behavior for checkpointable objects through the system call interface.
**S3.6.8.3.** CR shall accept parameters to be passed to the CR system call through the CR command line interface.
**S3.6.8.4.** CR shall allow the user invoking checkpoint to specify the following behaviors for any arbitrary set of files (including files that are not open at the time of the checkpoint):

a) check
b) nocheck
c) truncate
d) notruncate
e) backup_restore

f) backup_anonymous.

The meaning of these flags is described in section S3.9.8.

**S3.6.8.5. Control for shared libraries and dynamic libraries:** Saving of shared libraries is described in section S3.9.8.5.

**S3.6.8.5.1.** An option will be provided to make checkpoint save entire shared library or dlopen'd region rather than omitting read-only text sections. This is intended to allow checkpoint/restart across shared library upgrades.

## S3.7. Checkpoint Environment

### S3.7.1. Application in system call

**S3.7.1.1.** CR shall checkpoint processes using a slow system call (see S1.4.) at the time of checkpoint.

**S3.7.1.2.** CR shall restart processes using a slow system call.

**S3.7.1.3.** When possible, CR shall ensure that slow system calls shall be reissued when the processes is restarted.

**S3.7.1.4.** If a slow system call cannot be reissued, CR shall ensure that the system call returns EINTER to the application.

## S3.8. Restart Environment

### S3.8.1. Process migration

CR shall allow sessions or processes described in a context fileset to be restarted on nodes different from the node where the checkpoint was taken if all resources are successfully obtainable.

### S3.8.2. System reboot

CR shall allow sessions or processes described in a context to be restarted regardless of whether the node has been rebooted since the time the context fileset has been created.

### S3.8.3. Shared libraries

CR shall restart processes using runtime loaded libraries under the following conditions:

**S3.8.3.1.** If a flag was passed to the checkpoint system call requesting the shared library to be saved with the process or session.

**S3.8.3.2.** If the name and length of the shared libraries are unchanged since the checkpoint took place.

## S3.9. Operating System Context

This section describes data associated with the operating system kernel.

**S3.9.1.** CR shall save all information necessary to restore a process during a checkpoint.

**S3.9.2.** CR shall restore processes to the state that existed at the time a checkpoint took place.

### S3.9.3. Process credentials

**S3.9.3.1. UID**

**S3.9.3.1.1.** CR shall save all UIDs associated with a checkpointed process.

a) UID
b) effective UID
c) saved UID
d) filesystem UID

**S3.9.3.1.2.** If a root process invokes the restart system call, CR shall set the UIDs of the restarted processes to the respective UIDs stored in the context fileset.

**S3.9.3.1.3.** If a non-root process invokes the restart system call, CR shall set the UIDs of the restarted processes to the UIDs of the process invoking the restart system call. See section S3.6.3. for more information.

**S3.9.3.2. GID**

**S3.9.3.2.1.** CR shall save all GIDs associated with a checkpointed process.

a) GID
b) effective GID
c) saved GID
d) filesystem GID

**S3.9.3.2.2.** If a root process invokes the restart system call, CR shall set the GIDs of the restarted processes to the respective GIDs stored in the context fileset.

**S3.9.3.2.3.** If a non-root process invokes the restart system call, CR shall set the GID of the restarted processes to the GID of the process invoking the restart system call. See section S3.6.3. for more information.

**S3.9.3.3. Supplementary GIDs**

**S3.9.3.3.1.** CR shall save the supplementary group IDs of all checkpointed processes.

**S3.9.3.3.2.** If a root process invokes the restart system call, CR shall set the supplementary group IDs of the restarting processes to the supplementary group IDs of the user stored in the context fileset.

**S3.9.3.3.3.** If a non-root process invokes the restart system call, CR shall set the supplementary GIDs of the restarted processes to the intersection of the supplementary GIDs of the process invoking the restart system call and the supplementary GIDs owned by the restarting process. See section S3.6.3. for more information.

**S3.9.3.4. Capabilities**

**S3.9.3.4.1.** CR shall save a process' capabilities.

**S3.9.3.4.2.** If a non-root process invokes the restart system call, CR shall set the capabilities of the restarting process to the capabilities of the calling process.

**S3.9.3.4.3.** If a root process invokes the restart system call, CR shall set the capabilities of the restarting process to the intersection of the capabilities stored in the context fileset and the capabilities held by the current process.

### S3.9.4. TTY

**S3.9.4.1.** CR shall checkpoint processes using an interactive terminal.

**S3.9.4.2**. CR shall save state associated with the terminal device.

**S3.9.4.3.** CR shall restore the controlling terminal to the saved state.

### S3.9.5. UNIX Process Groups and Sessions

**S3.9.5.1. PPID**

**S3.9.5.1.1.** CR shall save the parent PID of all checkpointed process.

**S3.9.5.1.2.** When restarting sessions, CR shall restore the parent PID of all restarting processes other than a session leader.

**S3.9.5.1.3.** CR shall set the parent PID of a restarting session leader to the process invoking the restart call.

**S3.9.5.1.4.** When asked to restart a single process, CR shall set the parent PID of a restarting process to the PID of the process invoking the restart system call.

**S3.9.5.2. PID**

**S3.9.5.2.1.** CR shall save the PID of all checkpointed processes.

**S3.9.5.2.2.** CR shall restore the PIDs of checkpointed process.

**S3.9.5.2.3.** If the PID of a checkpointed process is in use at the time of a restart, CR shall return an error code through the restart system call.

**S3.9.5.3. Process groups**

**S3.9.5.3.1.** CR shall save all process groups within a session.

**S3.9.5.3.2.** CR shall restore saved process groups.

**S3.9.5.3.3.** CR shall save the process group id (pgid) of all processes.

**S3.9.5.3.4.** CR shall restore the process group id (pgid) of all processes within a process group.

**S3.9.5.3.5.** If a process group id of a restarting session is in at the time of a restart, CR shall return an error code through the restart system call.

**S3.9.5.3.6.** If a single process is restarting, CR shall set the process group id of the process to the process group id of the process invoking the restart call.

**S3.9.5.4. Sessions**

**S3.9.5.4.1.** CR shall save the session ID of a checkpointed process.

**S3.9.5.4.2.** If the session ID of a restarting session is in use at the time of a restart, CR shall return an error code through the restart system call.

**S3.9.5.4.3.** When restarting a single process, CR shall set the session ID of the restarting process to the session ID of the process invoking the restart system call.

**S3.9.5.4.4.** When restarting a session, CR shall send a HUP signal to all members of the session invoking the restart system call.

### S3.9.6. Interprocess Communications

#### S3.9.6.1. Named pipes

**S3.9.6.1.1.** CR shall save data stored in named pipes during a checkpoint.

**S3.9.6.1.2.** CR shall restore data in a named pipe during restart.

**S3.9.6.1.3.** CR shall restore named pipes that exist between two processes if those two processes were checkpointed as part of the same session.

**S3.9.6.1.4.** CR shall recreate the file system entry for a named pipe if the file does not exist during the restart.

#### S3.9.6.2. Pipes

**S3.9.6.2.1.** CR shall save data stored in pipes during a checkpoint.

**S3.9.6.2.2.** CR shall restore data in a pipe during restart.

**S3.9.6.2.3.** CR shall restore pipes that exist between two processes if those two processes were checkpointed as part of the same process group or session.

### S3.9.7. Accounting and Resource Limits

#### S3.9.7.1. CPU time usage

**S3.9.7.1.1.** CR shall save all process accounting information associated with a process.

**S3.9.7.1.2.** CR shall reset CPU time accounting information associated with a process to zero during a restart.

#### S3.9.7.2. Resource limits

**S3.9.7.2.1.** CR shall save a process' resource limits (set by setrlimit).

**S3.9.7.2.2.** CR shall restore a process' resource limits (set by setrlimit) unless these limits are larger than those set for the process invoking the restart system call.

**S3.9.7.2.3.** If the resource limits described by a context fileset are larger than those set for the process invoking the restart system call, CR shall set the resource limits of the restarting process to the limits of the process invoking the restart system call.

### S3.9.8. Files

#### S3.9.8.1. Open files

**S3.9.8.1.1.** CR shall save a process' file descriptors.

**S3.9.8.1.2.** CR shall restore a process' file descriptors, unless these descriptors are overridden (see S3.9.8.1.5. and S3.9.8.1.6. )

**S3.9.8.1.3.** CR shall set the file descriptors of the session leader to those of the process invoking the restart system call.

**S3.9.8.1.4.** CR shall set the file descriptors of a single restarting process to those of the process invoking the restart system call.

**S3.9.8.1.5.** CR shall honor the requirements in section S3.9.8.1. , unless special support for the file overrides these these requirements. (described in S3.9.8.4. )

**S3.9.8.1.6.** CR shall save the close-on-exec flag associated with a file descriptor.

**S3.9.8.1.7.** CR shall restore the close-on-exec flag associated with a file descriptor.

**S3.9.8.1.8. File mode**: CR shall reopen file descriptors with the same mode ('r', 'w', etc.) they had at time of checkpoint.

#### S3.9.8.2. Checking and restoration of file contents:

CR shall allow the following flags to be specified at checkpoint. Specified files do not need to be open by any processes in the checkpoint at the time of the checkpoint for the behaviors to apply.

**S3.9.8.2.1. Unchangeable:** whether or not the file was open at checkpoint, CR shall record a checksum of the file. Restart will fail if the file has been modified (i.e. its checksum is different). If the "Truncate" flag (see below) is also specified, the behavior is slightly different: if the file has only been modified by having new bytes appended to it, it will be restored to its checkpoint state by being truncated to its original length.

In either case, if restart succeeds, and the file was open at the time of the checkpoint, restart will reopen the file and restore the file
pointer to the position it had at checkpoint.

**S3.9.8.2.2. Changeable:** No checks for file modification are performed. If a file was open at checkpoint, at restart CR shall reopen the file and seek to the position the file pointer had at checkpoint. If the "Truncate" flag is also specified, the file will be set to the size it had at checkpoint (if it is shorter than it was at checkpoint, it is extended with 0 bytes).

**S3.9.8.2.3. Append:** No checks for file modification are made. If a file was open at time of checkpoint, restart shall reopen the file and seek to the end of the file. This is often useful for log files.

**S3.9.8.2.4. Backup_restore:** whether or not the file was open at time of checkpoint, a full copy of it shall be stored at checkpoint. At restart, the saved copy shall be reinstated on the filesystem (overwriting any existing version). If the file was open, it is reopened and the file pointer seeked to the same position.

**S3.9.8.2.5. Backup_anonymous:** if and only if the file was open at checkpoint, a copy of file is made and stored. At restart, the saved version is not substituted for whatever version exists on regular filesystem: instead the saved file is opened as an anonymous file (with pointer seeked to location at checkpoint). The anonymous file will disappear at program exit.

**S3.9.8.2.6. Truncate:** Modifies behavior of Checksum and Nochecksum, as described above.

**S3.9.8.2.7.** CR shall allow these flags to be specified on a per-file basis.

**S3.9.8.3. Standard IO**

**S3.9.8.3.1.** CR shall restore the standard input of processes part of a session that are not session leaders.

**S3.9.8.3.2.** CR shall restore the standard output of processes part of a session that are not session leaders.

**S3.9.8.3.3.** CR shall restore the standard error of processes part of a session that are not session leaders.

**S3.9.8.4. Special files**

**S3.9.8.4.1. /dev/null:** CR shall restore references to /dev/null.

**S3.9.8.4.2. /dev/zero:** CR shall restore references to /dev/zero.

**S3.9.8.4.3. /dev/tty:** CR shall ensure that all references to /dev/tty in the context fileset are restored to /dev/tty of the controlling process.

**S3.9.8.4.3. Aliases to /dev/tty:** CR shall ensure that aliases to the /dev/tty in the context fileset are restored to the corresponding alias of the controlling process.

**S3.9.8.4.4. /dev/random:** CR shall restore references to /dev/random.

**S3.9.8.4.5. /dev/urandom:** CR shall restore references to /dev/urandom.

**S3.9.8.4.6. /proc/self/*:** CR shall restore references to /proc/self.

**S3.9.8.4.7. /proc/_pid_/*:** CR shall restore references to /proc/pid, where pid is the PID of a process described in the context fileset.

**S3.9.8.5. mmap()'d files**

**S3.9.8.5.1. Private mappings:** CR shall restore mappings created with the MAP_PRIVATE flag set.

**S3.9.8.5.2. Shared mappings:** CR shall restore mappings with the MAP_SHARED flag set.

**S3.9.8.6. Unlinked files**

**S3.9.8.6.1.** CR shall save open-unlinked files.

**S3.9.8.6.2.** CR shall restore open-unlinked files.

**S3.9.8.7. umask**

**S3.9.8.7.1.** CR shall save the umask of a process.

**S3.9.8.7.2.** CR shall restore the umask of a process.

**S3.9.8.8. Locks**

**S3.9.8.8.1.** CR shall save the locks that a process holds.

**S3.9.8.8.2.** If the process exits, CR shall release file locks after a checkpoint is taken.

**S3.9.8.8.3.** CR shall reacquire file locks during a restart.

**S3.9.8.8.4.** CR shall abort a restart operation if it is unable to reacquire locks.

**S3.9.8.9. Working directory**

**S3.9.8.9.1.** CR shall save a process' current working directory.

**S3.9.8.9.2.** CR shall restore a process' current working directory.

**S3.9.8.10. Root directory**

**S3.9.8.10.1.** CR shall save a process' root directory.

**S3.9.8.10.2**. CR shall set the root directory of a restarting process to the root directory of the process invoking the restart system call.

**S3.9.8.10.3.** If the root directory of a restarting process does not exist, CR shall abort the restart and return an error indicating the cause of the failure.

**S3.9.8.11. Open directories**

**S3.9.8.11.1.** CR shall save the state of open directories within a process.

**S3.9.8.11.2.** CR shall restore open directories.

### S3.9.9. Signal handling

**S3.9.9.1. Pending signals**

**S3.9.9.1.1.** CR shall save signals pending to a process.

**S3.9.9.1.2.** CR shall restores signals pending to a process.

**S3.9.9.2. Interval timers**

**S3.9.9.2.1.** CR shall save the interval timers registered by a process.

**S3.9.9.2.2.** CR shall restore interval timers registered by a process.

**S3.9.9.3. Signal handlers**

**S3.9.9.3.1.** CR shall save sigaction structures registered by a process.

**S3.9.9.3.2.** CR shall restore sigaction structures registered by a process.

### S3.9.10. Address Space and Execution

**S3.9.10.5. Shared libraries**

**S3.9.10.5.1.** CR shall checkpoint processes using shared libraries.

**S3.9.10.5.2.** CR shall restore processes using shared libraries.

**S3.9.10.5.3.** CR shall save writable regions of shared libraries.

**S3.9.10.1. Dynamically loaded libraries**

**S3.9.10.1.1.** CR shall checkpoint processes using dlopen().

**S3.9.10.1.2.** CR shall restore dynamically loaded libraries open with dlopen().

**S3.9.10.5.3.** CR shall save writable regions of dlopen()'d libraries.

**S3.9.10.2. Hardware context**

**S3.9.10.2.1.** CR shall save the state of the CPU registers.

**S3.9.10.2.2.** CR shall restore the CPU registers.

**S3.9.10.3. Heap**

**S3.9.10.3.1.** CR shall save the process heap.

**S3.9.10.3.2.** CR shall restore tea process heap.

**S3.9.10.4. Stack**

**S3.9.10.4.1.** CR shall save the process stack.

**S3.9.10.4.2.** CR shall restore the process stack.

**S3.9.10.6. Pinned memory**

**S3.9.10.6.1.** CR shall save save memory pinned by mlock().

**S3.9.10.6.2.** CR shall restore pinned memory to the process address space.

**S3.9.10.6.3.** CR shall restore memory that was pinned before the checkpoint to a pinned state.

**S3.9.10.6. Protected memory**

**S3.9.10.6.1.** CR shall save save memory protected by mprotect().

**S3.9.10.6.2.** CR shall restore protected memory to the process address space.

**S3.9.10.6.3.** CR shall restore memory that was pinned before the checkpoint to a protected state.