Berkeley Lab Checkpoint/Restart (BLCR) for Linux Clusters

http://ftg.lbl.gov/checkpoint

checkpoint@lbl.gov

Paul H. Hargrove and Jason C. Duell

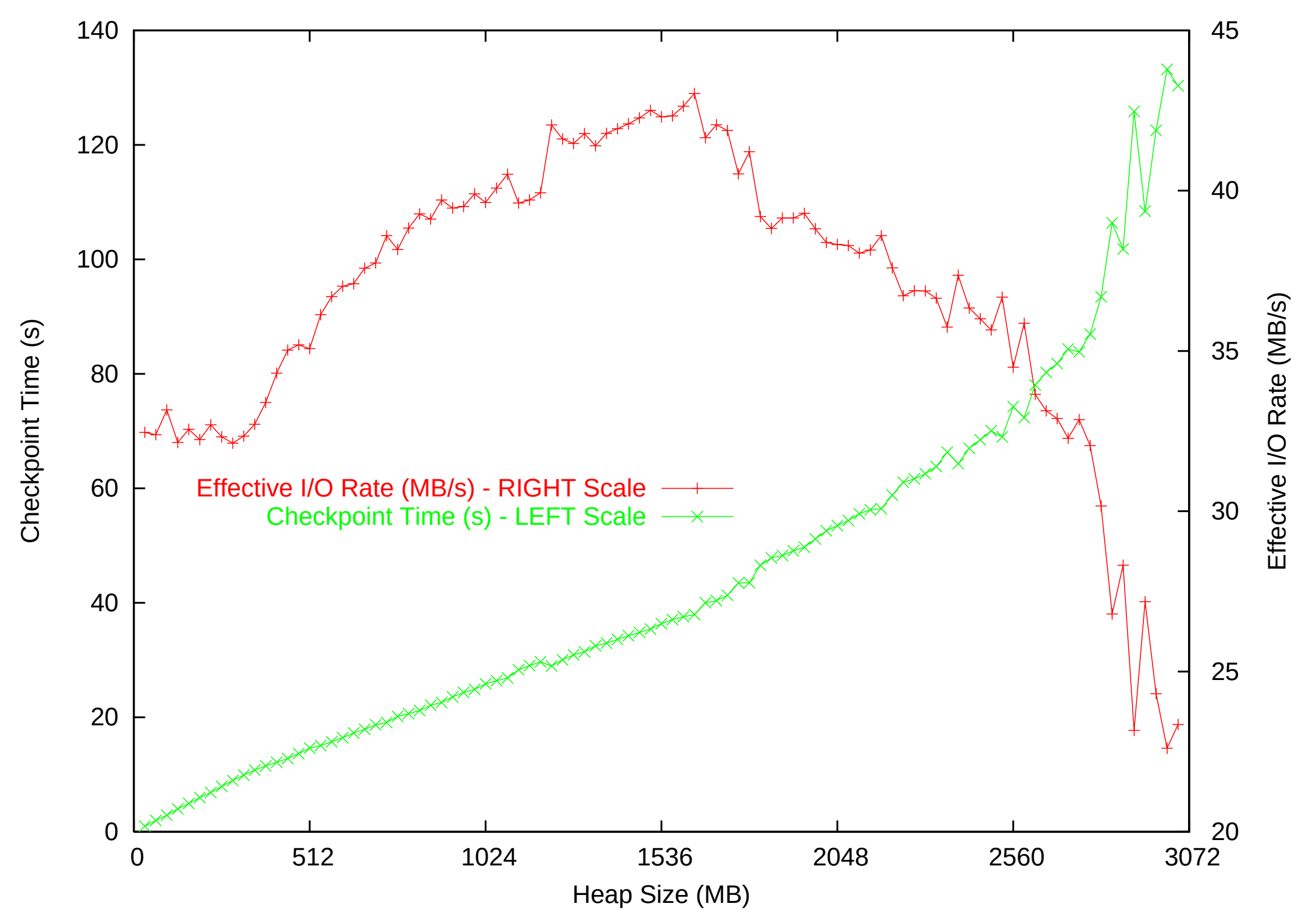
Lawrence Berkeley National Laboratory

Berkeley Lab Checkpoint/Restart (BLCR for short) is a part of the SciDAC Scalable Systems Software ISIC, with the goal of providing a reusable, production-quality, system-level implementation of Checkpoint/Restart for High Performance Computing on Linux clusters.

- Implemented at the system-level
  - No source code modifications for most scientific applications
  - Recompilation or relinking with a supported MPI may be required
  - Permits save/restore of resources not easily accessible from user space
- Saves and restores the basic resources
  - Virtual address space with file mappings (exec and libs)
  - Registers
  - Signal Handlers
  - Open files ("by reference" not full content)
  - PID and TGID
- Available on x86 and x86-64 architectures
  - No significant technical barriers to other ports
- Available on a wide range of kernels
  - Linux 2.4.x and 2.6.x from kernel.org
  - SuSE 7.2 through 10.0
  - Red Hat 7.1 through 9
  - Fedora Core 2 through 4
  - Many others
  - NO support effort for 2.5.x development kernels
- Open Source Software
  - Kernel portions under GPL
  - User space portions under LGPL
- Implemented as a loadable kernel module
  - No kernel patches to void vendor support
  - Can more easily track kernel updates
- Uses GNU autoconf for wide kernel coverage
  - Can check for specific kernel functions and data types
  - Not dependent on kernel version number, making it tolerant of patches applied by the distribution, vendor or site.
  - Key to supporting Red Hat 9 kernel (a mix of Linux 2.4.x and 2.6.x code)
- Integrates GNU autotools with Linux kbuild
  - `configure; make; make install`
- Preemptive
  - Not intended to replace application-level checkpointing when available
  - Can be used to respond to fault precursors, such as:
    - High correctable error rates from ECC memory
    - High checksum failures from interconnect
    - Elevated temperature (loss of a fan or chilled water)
  - Can be used for preeemption in batch scheduling
    - Higher utilization (no queue draining)
    - Lower wait times (time slice the "wide" jobs)
    - Multiple QoS options
- SMP-safe and thread-safe
  - Supports both LinuxThreads and NPTL versions of POSIX threads

- Single-node with an extension interface
  - Handles most resources within a single Linux kernel instance (see chart at left)
  - Allows external support for resources BLCR doesn't save/restore
    - LAM/MPI 7.x uses this interface to provide coordinated checkpointing of MPI jobs and associated socket or gm communications.
    - Support in other MPIs in-progress
- Migration-capable (limited)
  - No node-specific data required in the checkpoint
  - No attempt to hide IP or hostname changes
  - MPI support exists in LAM/MPI
- No resource virtualization or interception of C-library calls
  - BLCR adds absolutely zero code to libc execution paths
- Security-conscious
  - Checkpoint permissions follows that of signals and core dumps
    - Process owner or root required to request a checkpoint
    - Set-id processes may **not** be checkpointed
  - Restart procedure does **not** bypass normal permission checks
  - Modification of a context file will **not** lead to privilege escalation

Checkpoint I/O Performance to a Local IDE Disk



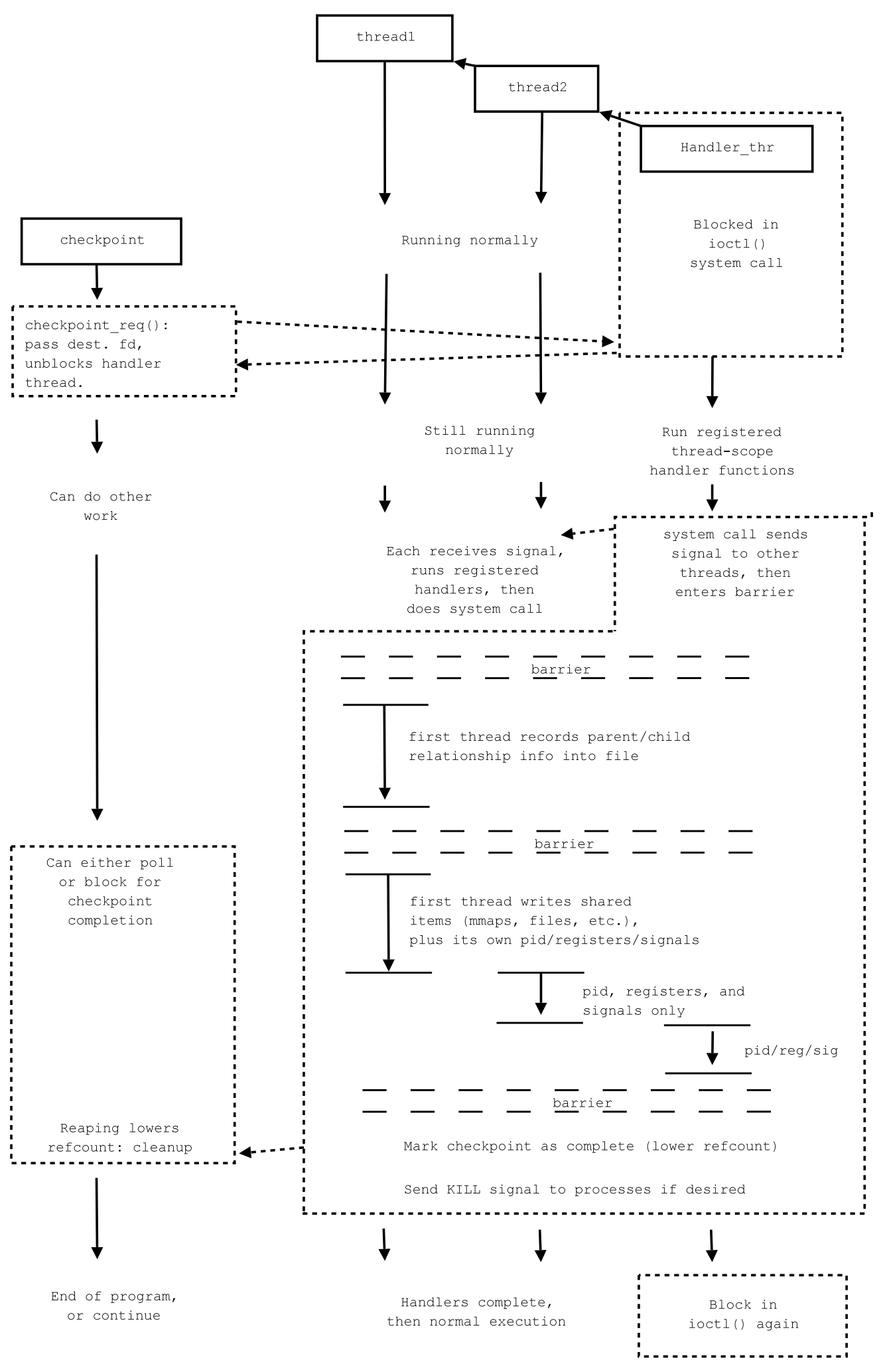
REFERENCES (Available at http://ftg.lbl.gov/checkpoint)

- Sankaran S, Squyres J M, Barrett B, Lumsdaine A, Duell J, Hargrove P and Roman E 2003 The LAM/MPI checkpoint/restart framework: system-initiated checkpointing *Proc. Los Alamos Computer Science Institute (LACSI) Symp. (Santa Fe, New Mexico, USA, October 2003)*
  Also appearing in 2005 *Intl. J. High Performance Computing Applications* **19** 479-93
- Duell J, Hargrove P and Roman E 2002 Requirements for Linux Checkpoint/Restart *Lawrence Berkeley National Laboratory Technical Report LBNL-49659*
- Duell J, Hargrove P and Roman E 2002 The Design and Implementation of Berkeley Lab's Linux Checkpoint/Restart *Lawrence Berkeley National Laboratory Technical Report LBNL-54941*
- Roman E 2003 A Survey of Checkpoint/Restart Implementations *Lawrence Berkeley National Laboratory Technical Report LBNL-54942*

Current Feature Set

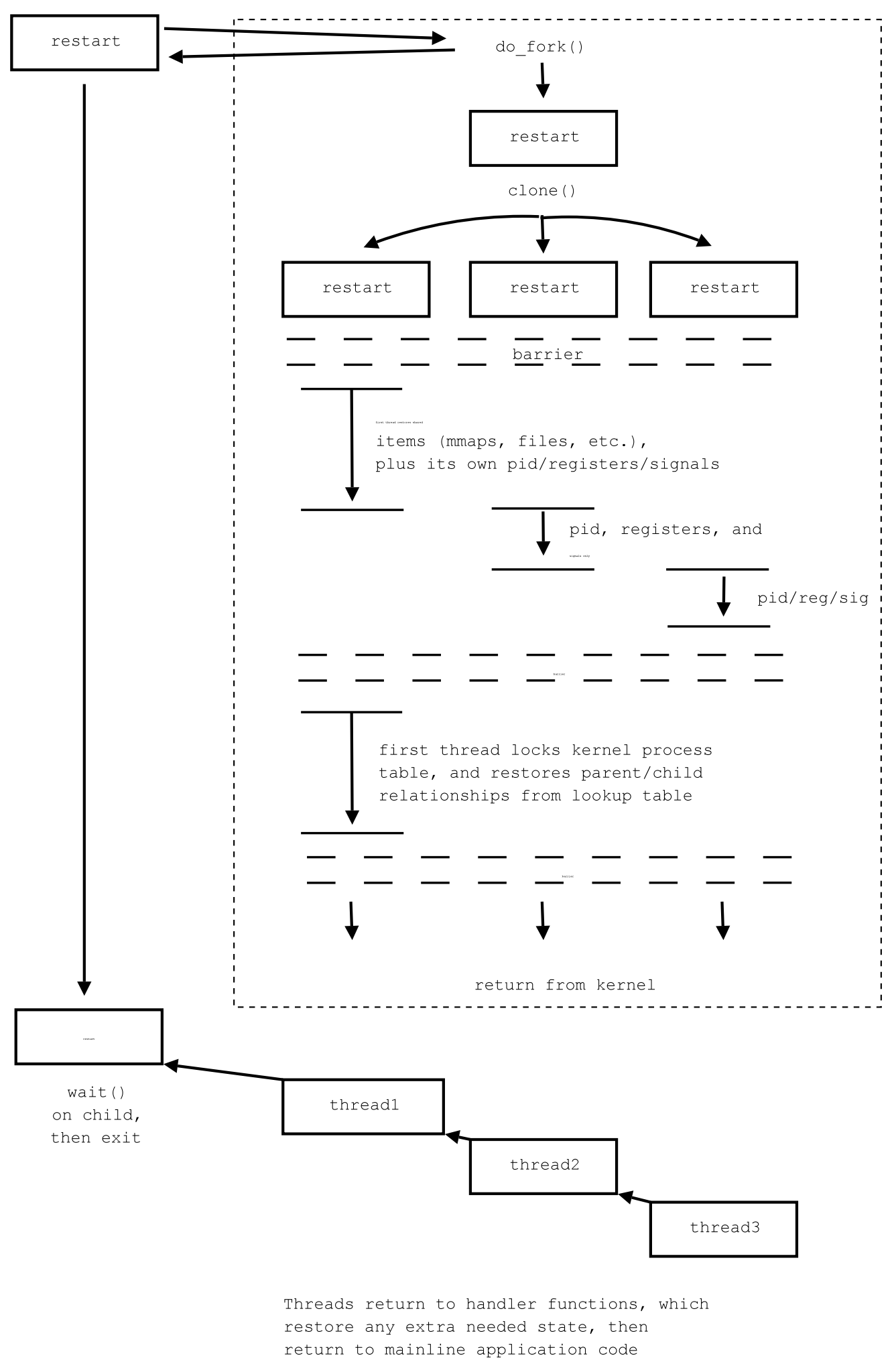
Feature	Implemented	Planned	Won't Support
Single-threaded processes	X		
Threaded processes	X		
Process groups		X	
Sessions		X	
Extension API	X		
Signal state/handlers	X		
Restart system calls	X		
Ptraced processes			X
PID & PPID	X		
UID/GID	IFF root		
Resource limits		X	
rusage info		X	
Regular files	X		
Stdin/stdout/stderr	X		
Unnamed & named pipes	X		
TCP/UDP sockets			X
mmapped files	X		
Shared mmap regions		X	
Block device files (/dev/null, /dev/zero, etc)	X		
/proc/self files	X		
File locks		X	
Asynchronous I/O			?
Mprotect() and madvise()	X		
System V IPC			?
Pthread synchronization	X		

Taking a Checkpoint



thread1
thread2
Handler_thr
checkpoint
Running normally
Blocked in ioctl() system call
checkpoint_req(): pass dest. fd, unblocks handler thread.
Still running normally
Run registered thread-scope handler functions
Can do other work
Each receives signal, runs registered handlers, then does system call
system call sends signal to other threads, then enters barrier
barrier
first thread records parent/child relationship info into file
barrier
Can either poll or block for checkpoint completion
first thread writes shared items (mmaps, files, etc.), plus its own pid/registers/signals
pid, registers, and signals only
pid/reg/sig
barrier
Reaping lowers refcount: cleanup
Mark checkpoint as complete (lower refcount)
Send KILL signal to processes if desired
End of program, or continue
Handlers complete, then normal execution
Block in ioctl() again

Restarting from a Checkpoint



restart
do_fork()
restart
clone()
restart
restart
restart
barrier
items (mmaps, files, etc.), plus its own pid/registers/signals
pid, registers, and
pid/reg/sig
first thread locks kernel process table, and restores parent/child relationships from lookup table
return from kernel
wait() on child, then exit
thread1
thread2
thread3
Threads return to handler functions, which restore any extra needed state, then return to mainline application code

Work supported by the Office of Advanced Scientific Computing Research, Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.