Checkpoint/Restart in Linux

Sukadev Bhattiprolu
IBM Linux Technology Center

09/2009

Agenda

- What and Why Checkpoint/Restart
- Prerequisites and Requirements
- Usage Overview
- Kernel API
- Current Status (v18 posted)
- Demo
- Design/API Discussion

What is Checkpoint/Restart ?

- Checkpoint: save state of a running application
- Restart: resume application from saved state
- Migration: checkpoint on one host, restart on another
  - Static migration
  - Live migration

Why Checkpoint/Restart ?

- Reduced application downtime: checkpoint, reboot, restart
- Application mobility
  - User-session mobility
  - Migrate application to another server before system upgrade
- Improve system utilization
- Faster error recovery with periodic checkpoints

Why Checkpoint/Restart ?

- Slow-start applications
- Debug – start from last checkpoint
- General time-travel
- Other use-cases ?

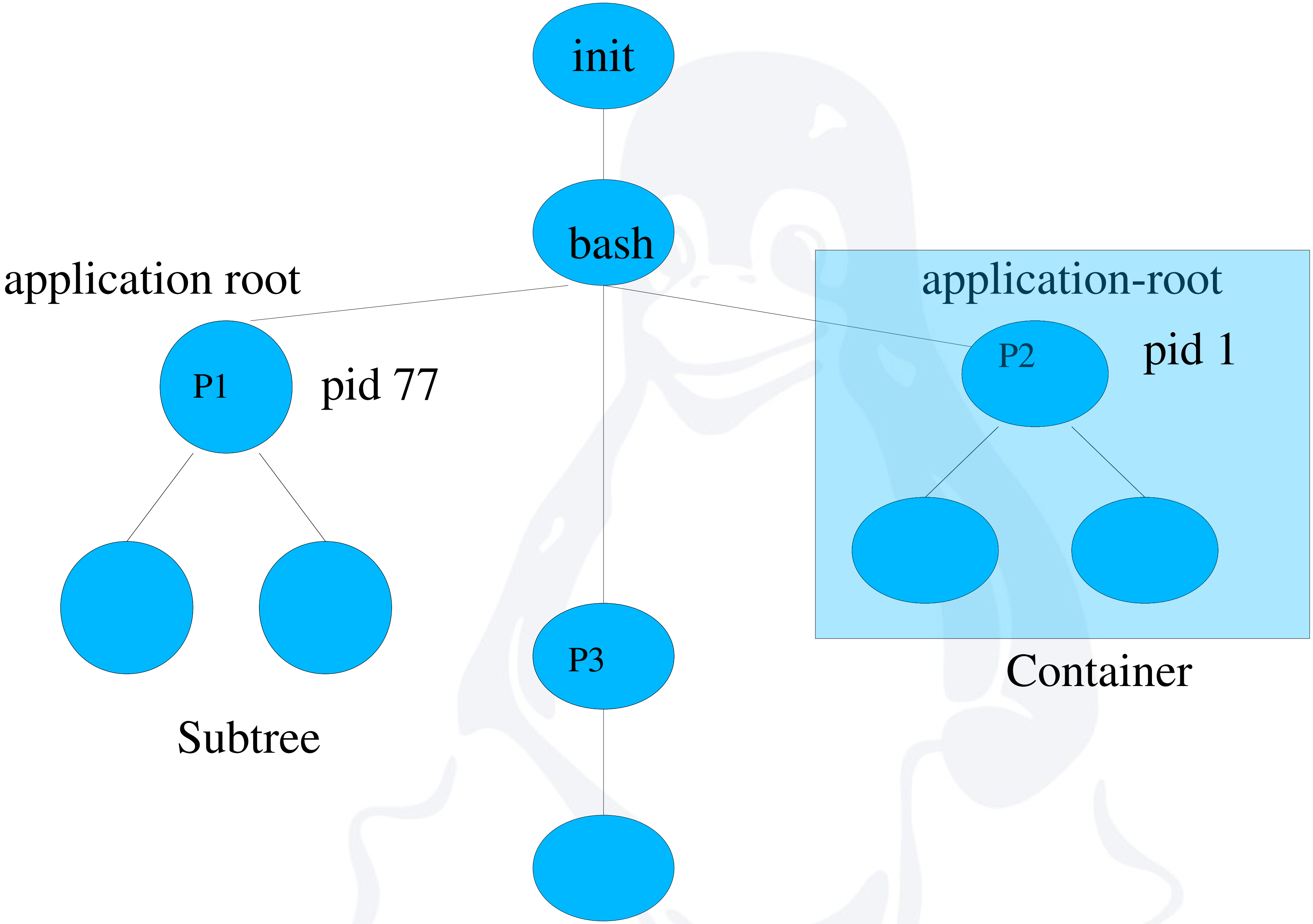
Pre-requisite: Freezer

- Freeze application process-tree for consistent checkpoint
- Freezer implemented as a cgroup
- Status: merged into mainline
- Usage:

```
$ mount -t cgroup -o freezer foo /cgroups
$ echo FROZEN > /cgroups/$pid/freezer.state
$ cat /cgroups/$pid/freezer.state
FROZEN
$ echo THAWED > /cgroups/$pid/freezer.state
```

Pre-requisite: Containers

- Containers – isolated name spaces enable reuse of resource ids
- Needed in C/R to restore original resource ids in the application
- Create containers using clone(2) system call (or a wrapper to it)
- Status: Mount, UTS, IPC, PID, Network name spaces, devpts merged



init

bash

application root

P1

pid 77

Subtree

P3

application-root

P2

pid 1

Container

Basic Requirements

- Transparent - work with existing binaries
- Low impact on other subsystems
  - Performance
  - No duplicate code-paths
  - Maintenance overhead
- Integrated into kernel
- Generalized: Not restricted to specific applications

Basic Requirements

- Allow full-container and subtree C/R
  - Full-container: C/R of complete process tree
  - Subtree: C/R of part of process tree
- Full-container C/R needed to:
  - Restore original resource-ids
  - Prevent 'leaks' in shared resources
- Subtree C/R useful within limits for:
  - Resource-id agnostic applications
  - C/R aware applications
  - Development
- Enable self-checkpoint

Checkpoint Usage

- Create application in a container
- Freeze container (from parent)
- Checkpoint:

```
$ checkpoint -p 1234 > checkpoint-img.1
```

- Snapshot file system – leverage fs capability (btrfs, nilfs etc)
- Thaw application
- Terminate application (if necessary)

Restart Usage

- Restore file system state to snapshot (leverage fs capabilities)
- Restart application in new container

```
$ restart –container --wait < checkpoint-img.1
```

C/R Kernel API (proposed)

- sys_checkpoint(pid_t pid, int fd, ulong flags);
- sys_restart(pid_t pid, int fd, ulong flags);

  - pid: root of application process-tree to checkpoint/restart
  - fd: file descriptor or socket to/from which to write/read checkpoint image

C/R Kernel API (proposed)

```
struct clone_arg {
    u64 clone_flags;
    u32 parent_tid,
    u32 child_tid;
    u32 nr_pids;
    /* plus some reserved space */
}
```

- sys_clone2() (struct clone_struct *cs, pid_t *pids)
  - Allow additional clone-flags
  - Allow ability to choose pids for child process

C/R Kernel API: Image format

- Checkpoint image format:
  - Blob that may change over time
  - Has a version number
  - Stream-able
- User space tools convert image between kernel versions
- General layout:
  - Image header
  - Task hierarchy
  - Task state of each task
  - Image trailer
- Shared objects saved only once

Status: Done

- Currently restored (in ckpt-v18)
  - Process-trees, pthreads, signals, handlers
  - SYS-V IPCs, FIFOs, itimers
  - Devices: null, random, zero, pts
  - Self-checkpoint
- File systems:
  - Regular files and directories in normal fs
  - Some special fs (devpts)
- Architectures: Working on i386, ppc64, s390

C/R Network state

- AF_UNIX Sockets restored
- AF_INET Sockets:
  - Restored if both ends were checkpointed
  - If one end was checkpointed, connection restored if restarted within tcp delay

C/R: Devices

- PTY devices restored. Other virtual devices can be restored
  - Use Client/Server model and C/R server
  - Display: Use VNC
  - Audio: Pulse Audio
- If application tied to specific hardware, C/R in kernel is complex
  - Device like /dev/rtc maybe in use
  - Device may not be available
  - Restore such devices in user-space ?

Status: Pending

- Time, POSIX Timers, Timezones
- File systems
  - Pseudo FS (eg: /proc)
  - NFS ?
  - Unlinked files, directories
- Devices
- Event-poll (WIP)
- Others:
  - Inotify, mount-points, mount-ns, etc

Discussion

- We have some design/API choices that we would like some feedback on

Q&A: Time

- C/R of time presents interesting challenges
  - Restart may happen after long time
- Choose a policy on restart ?
  - Use current or original time ?
  - Timer-expirations relative/absolute ?
- Is policy per-process or per-restart ?
  - Which policy for new children ?

Q&A: Process tree

- Restore process-tree in user-space ?
  - Leverage existing calls like fork(), clone()
  - Allows subtree C/R
  - Needs clone2()
  - Needs kernel synchronization of processes in tree during restart
- Or, in kernel ?
  - Avoid clone2() and synchronization in-kernel
  - Reduced flexibility ?

Q&A: Restore fs, network

```
$ ns_exec –container – /bin/application –arg
$ echo FROZEN > /cgroups/$pid/freezer.state
$ checkpoint -p 1234 > checkpoint-img.1
$ snapshot-filesystem
# Thaw and terminate app

$ restart –container < checkpoint-img.1
```

Q: Restart program creates container – but how can we have it restore file system and network configuration state ?

Q&A: Kernel API

- cradvise()
  - Eg: skip C/R of some portion of memory or some device and let user-space handle it
- Notify C/R-aware applications of:
  - Pending or completed checkpoint
  - Completed restart
- Restart parts of an application
  - Eg: restore regular file fds, ignore IPC
- Others ?

Q&A: Kernel API

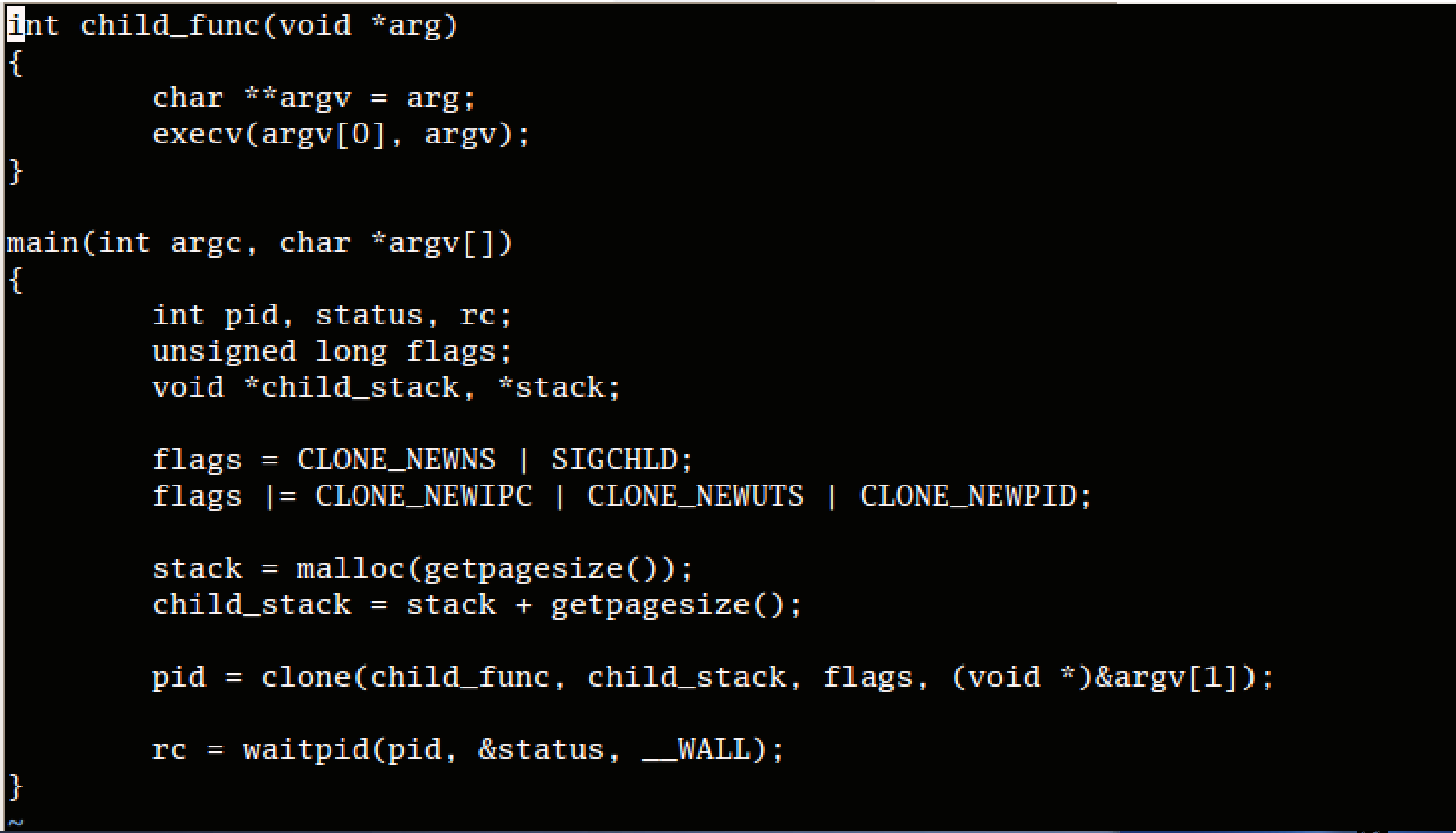
- Ability to control/optimize C/R. Eg:
  - Skip restore of part of memory
  - Skip restart of some device (let user space deal with it)
  - Use parent's UTS namespace
  - Skip C/R of IPC
- Use single system call: cradvise() with flags ?
- Or separate calls: cradvise_fd(), cradvise_mem() etc

Project Info

- Maintainer: Oren Laadan <oren@librato.com>
- Mailing list: Containers@lists.linux-foundation.org
- Wiki: http://ckpt.wiki.kernel.org
- Code:
  - git://git.ncl.cs.columbia.edu/pub/git/linux-cr.git
  - git://git.ncl.cs.columbia.edu/pub/git/user-cr.git
  - git://git.sr71.net/~hallyn/cr_tests.git

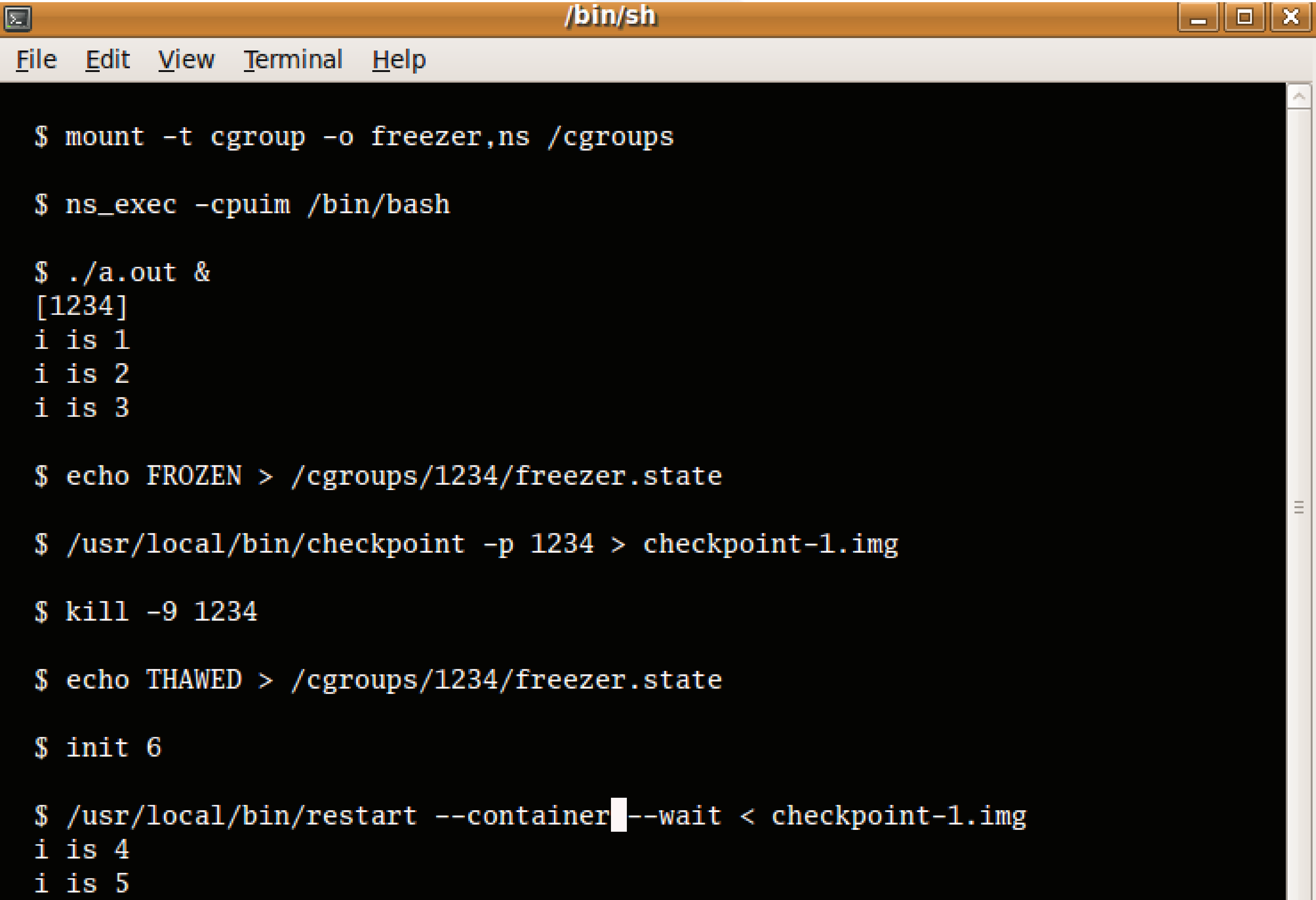
Backup slides

Simple ns_exec.c



```
int child_func(void *arg)
{
        char **argv = arg;
        execv(argv[0], argv);
}

main(int argc, char *argv[])
{
        int pid, status, rc;
        unsigned long flags;
        void *child_stack, *stack;

        flags = CLONE_NEWNS | SIGCHLD;
        flags |= CLONE_NEWIPC | CLONE_NEWUTS | CLONE_NEWPID;

        stack = malloc(getpagesize());
        child_stack = stack + getpagesize();

        pid = clone(child_func, child_stack, flags, (void *)&argv[1]);

        rc = waitpid(pid, &status, __WALL);
}
~
```

C/R Example



```
$ mount -t cgroup -o freezer,ns /cgroups

$ ns_exec -cpuim /bin/bash

$ ./a.out &
[1234]
i is 1
i is 2
i is 3

$ echo FROZEN > /cgroups/1234/freezer.state

$ /usr/local/bin/checkpoint -p 1234 > checkpoint-1.img

$ kill -9 1234

$ echo THAWED > /cgroups/1234/freezer.state

$ init 6

$ /usr/local/bin/restart --container --wait < checkpoint-1.img
i is 4
i is 5
```

Resources to Checkpoint

- Process trees,
- UTS (host name)
- Memory (shared-mem, mmaps etc)
- Open-file state
- SysV IPC, FIFOs
- AF_UNIX and AF_INET Sockets
- Restart-timers
- Signal state
- Devices
- Time
- Others ?
- TBD: Drop and use following two slides ?

Existing implementations

- OpenVZ (Parallels)
- Zap (Columbia University)
- MCR (IBM)
- BLCR
- Others (www.checkpointing.org)