COMP3301/COMP7308 Assignment 3

- School of Information Technology and Electrical Engineering
- The University of Queensland
- Semester Two, 2020
- Revision: $Revision: 261 $

1 OpenBSD Containers

This assignment asks you to implement a portion of the functionality necessary to provide "containers" on OpenBSD.

Container technologies aim to provide virtualisation at the kernel level instead of at the hardware level. This is implemented by extending a kernel to partition and isolate certain services to prevent processes in a container from interacting with processes in another container. Such isolation may require limiting the visibility of processes and file descriptors in the system, creating independent users and views of the file system, virtualising the network stack, and guaranteeing access to resources.

Several container technologies exist such as Docker, Solaris Zones, FreeBSD jails, and AIX WPARs. The tasks in this assignment will be loosely modelled on the design of Solaris Zones as documented in PSARC/2002/174. The aim will be to implement isolation of processes and several kernel variables in OpenBSD

This is an individual assignment. You should feel free to discuss aspects of C programming and the assignment specification with fellow students. You should not actively help (or seek help from) other students with the actual coding of your assignment solution. It is cheating to look at another student's code and it is cheating to allow your code to be seen or shared in printed or electronic form. You should note that all submitted code may be subject to automated checks for plagiarism and collusion. If we detect plagiarism or collusion, formal misconduct proceedings will be initiated against you. If you're having trouble, seek help from a member of the teaching staff. Don't be tempted to copy another student's code. You should read and understand the statements on student misconduct in the course profile and on the school web-site: https://www.itee.uq.edu.au/itee-student-misconduct-including-plagiarism

2 Specification

You will add basic support for process and some kernel variable isolation to the OpenBSD kernel and user land by implementing a simplified zones infrastructure modelled on the Solaris specification and implementation.

Process isolation is the prevention of a process in one zone being able to see or signal a process running in another zone.

The exception to this isolation will be the "global" zone, which can view all processes in the system, including those running in other zones. The ability to create, destroy, and enter zones, and signal processes in other zones, will be limited to the root user in the global zone. The global zone will exist by default, and by default the initial processes in the system will belong to the global zone.

Support for zones requires the addition of several system calls.

The assignment also requires that several kernel variables accessed via the `sysctl(2)` system call will be changed to have per-zone behaviour or values.

2.1 Code Style

Your code is to be written according to OpenBSD's style guide, as per the `style(9)` man page.

2.2 Interfaces

The following system calls need to be added to the OpenBSD system

2.2.1 `zone_create`

```
int                zone_create(zoneid_t z);
```

`zone_create` should create a new zone id for use in the system.

On success it should return 0.

On failure it should return -1 and set errno accordingly:

EPERM the current program is not in the global zone
EPERM the current user is not root
ERANGE too many zones are currently running
EBUSY a zone already exists with the specified id

2.2.2 `zone_destroy`

```
int                zone_destroy(zoneid_t z);
```

`zone_destroy` should delete the specified zone instance.

On success it should return 0.

On failure it should return -1 and set errno accordingly:

EPERM the current program is not in the global zone
EPERM the current user is not root
ESRCH the specified zone does not exist
EBUSY the specified zone is still in use, ie, a process is still running in the zone

2.2.3 `zone_enter`

```
int                zone_enter(zoneid_t z);
```

`zone_enter` moves the current process into the zone.

On success it should return 0.

On failure it should return -1 and set errno accordingly:

EPERM the current program is not in the global zone
EPERM the current user is not root
ESRCH the specified zone does not exist

2.2.4 `zone_list`

```
int                zone_list(zoneid_t *zs, size_t *nzs);
```

In the global zone `zone_list` will provide the list of zones in the running system as an array of zoneid_ts. If run in a non-global zone, the list will only contain the current zone.

The value at `nzs` will refer to the number of array entries in `zs` on input.

On success it should return 0 and the value at `nzs` will be set to the number of zones listed in `zs`.

On failure it should return -1 and set errno accordingly:

EFAULT `zs` or `nzs` point to a bad address
ERANGE if the number at `nzs` is less than the number of running zones in the system

2.2.5 `zone_lookup`

```
zoneid_t        zone_lookup(zoneid_t z);
```

Test whether a zone with the specified identifier exists. If the identifier argument is -1, return the zone id for the calling process.

Lookups from the non-global zone should fail unless -1 is passed as the argument.

On failure it should return -1 and set errno accordingly:

ESRCH the current program is not in the global zone
ESRCH the specified zone does not exist

2.2.6 fork(2)

When a process forks, the child must inherit the zone it is running in from its parent.

The only way for a process to change zones is via the `zone_enter` syscall, which is limited to root processes in the global zone.

2.2.7 kill(2)

The kernel signalling code should be modified to provide the following semantics:

- If any user in a non-global zone tries to signal any process in another zone, it should fail with ESRCH.
- If a non-root user in the global zone signals a process in another zone, it should fail with EPERM.
- root in the global zone may signal any process in any zone
- Users within a zone should get normal signalling semantics

2.2.8 sysctl(3)

The kernel side of sysctl should modify its handling of CTL_KERN KERN_PROC and KERN_FILE to filter results.

- the `kinfo_proc` structure has been modified to include a ps_zoneid field which identifies the zone the process is running in
- the global zone does not get a filtered list of processes
- non-global zones will get a list of processes that exist in their current zone

The following CTL_KERN variables will be modified to have per-zone settings:

KERN_HOSTNAME The global zone will default to an empty hostname value. Non-global zones will default the host name to the zone id it is created with. The host name value can only be changed by the root user within a zone.
KERN_DOMAINNAME The domain name value will default to an empty string in both the global and non-global zones. The domain name value can only be changed by the root user within a zone.
KERN_HOSTID The host identifier will default to 0. The host identifier can only be changed by the root user within a zone.
KERN_BOOTTIME The boot-time value for non-global zones will be set to the time at which a zone was created. It is read-only.

The following CTL_KERN variables will be modified to be read-only in non-global zones:

- KERN_MAXCLUSTERS
- KERN_CACHEPCT
- KERN_POOL_DEBUG

2.2.9 struct process

`struct process` represents the kernels state relating to a running program. This type should be extended to record which zone the process is running in.

2.3 Boilerplate

A diff will be provided that adds header files, programs, and modifications to the system to use and test the kernel zone functionality. The diff can be applied by running the following:

```
$ cd /usr/src
$ mkdir usr.sbin/zone
$ patch < /path/to/assignment3-boilerplate.diff
Hmm...  Looks like a unified diff to me...
```

The following types and limits will be made available to the system (both kernel and userland) via a `sys/zone.h` header file. Data structures and programs have been extended to use these values, and will be provided as part of a boilerplate diff:

2.3.1 zoneid_t

Zones will be uniquely identified in the running system by a numerical identifier using the following type:

```
typedef int zoneid_t;
```

2.3.2 MAXZONES

The system should be limited to only providing up to `MAXZONES` at a time.

2.3.3 MAXZONEIDS

While the system is limited to running a `MAXZONES` number of zones (including the global zone, the limit on identifiers is higher to avoid rapid reuse.

2.3.4 struct kinfo_proc

`struct kinfo_proc` is a serialisation of the kernels process and proc structures for userland to consume. It has been extended to include the id of the zone that the process is running in.

2.3.5 libc

`libc` has been updated to provide stubs for the system calls described above. The system provides a `zones.h` file that prototypes the system call stubs.

Programs needing to interact with the kernels zone infrastructure can `#include <zones.h>` and link to `libc`.

2.3.6 Userland Programs

To test the zones subsystem, some userland utilities have been modified. When appropriate, programs were modified to accept the following options:

-z zone Limit the scope of the command to the specified zone. The zone must be specified by numeric id.
-Z The numeric id of the zone should be added to the programs output.

The following changes were implemented:

2.3.7 ps(1)

- the -z option

When a zone is specified, the list of processes displayed by ps will be limited to those processes running in the specified zone.

- the -Z option

-Z causes the zones numeric id to be prepended to the columns that are output by ps(1).

Additionally, "ZONES" may be specified as a column in custom column format specifiers.

2.3.8 pgrep, pkill

- the -z flag

pgrep and pkill will only match on processes that are running in the specified zone.

2.3.9 zone(8)

zone(8) is a new program and can be installed under `/usr/sbin`.

The usage output is shown below:

```
usage:  zone create zoneid
        zone destroy zoneid
        zone list
        zone exec zoneid command ...
```

The sub-commands map to the system calls described above.

2.4 Recommendations

2.4.1 APIs

The APIs may be useful in the implementation of the required functionality.

- `rwlock(9)` - interface to read/write locks
- `copy(9)` - kernel copy functions
- `malloc(9)` - kernel memory allocator
- `pool(9)` - resource-pool manager
- `atomic_inc_int(9)` - atomic increment operations
- `atomic_dec_int(9)` - atomic decrement operations
- `atomic_cas_uint(9)` - atomic compare-and-swap operations
- `refcnt_take(9)` - reference count API
- `sysctl(3)` - get or set system information
- `queue(3)` - implementations of singly-linked lists, doubly-linked lists, simple queues, and tail queues
- `tree(3)` - implementations of splay and red-black trees

2.5 Constraints

Only the sysctl interfaces used by ps, pgrep, fstat, and kill will be tested. Other mechanisms for interacting with processes such as ptrace, ktrace, systrace, process groups, sessions, libkvm using /dev/mem or /dev/kmem will not be tested and do not need to implement isolation between processes in zones.

3 Submission

You are required to implement the changes to the OpenBSD source tree, and submit a diff of them against an OpenBSD CVS server to your subversion repository in an `a3` directory. You are welcome to store changes to

files in another portion of the tree.

Note that files must be added with CVS before diffs of them can be generated. Diffs against CVS should be in the unified diff format. Therefore use `cvs diff -uNp` when generating the diff to commit to subversion.

Submission must be made electronically by committing to your Subversion repository on `source.eait.uq.edu.au`. In order to mark your assignment the markers will check out `a3` from your repository. Code checked in to any other part of your repository will not be marked.

As per the `source.eait.uq.edu.au` usage guidelines, you should only commit a source diff.

3.1 Demo on Zoom

You are required to demo your assignment via a Zoom desktop screen sharing session, during your assigned Zoom Practical Session in week 12. If you do not demo via the Zoom session, then your assignment will not be marked. You are only permitted to demo the code that was submitted by the due date.