2.1 Background

Zones extend the isolation of processes beyond what is traditionally provided by UNIX and UNIX-like systems, including OpenBSD. Traditionally, all processes running on an OpenBSD are visible to all other processes. This can be demonstrated by running commands like top(1), ps(1), and pgrep(1)/pkill(1), which can show all processes running in a system:

```
$ ps -ax
  PID TT  STAT        TIME COMMAND
    1 ??  I        0:01.01 /sbin/init
35862 ??  Ip       0:00.01 /sbin/slaacd
 9544 ??  Ip       0:00.01 slaacd: engine (slaacd)
33073 ??  IpU      0:00.01 slaacd: frontend (slaacd)
96644 ??  IU       0:00.01 /sbin/dhcpleased
82639 ??  Ip       0:00.01 dhcpleased: engine (dhcpleased)
68436 ??  IpU      0:00.01 dhcpleased: frontend (dhcpleased)
 6881 ??  IpU      0:00.01 /sbin/resolvd
69588 ??  IpU      0:00.03 syslogd: [priv] (syslogd)
54598 ??  Spc      0:00.03 /usr/sbin/syslogd
14516 ??  IU       0:00.01 pflogd: [priv] (pflogd)
15079 ??  Spc      0:00.12 pflogd: [running] -s 160 -i pflog0 -f /var/log/
   pflog
94692 ??  S<pc     0:00.12 ntpd: ntp engine (ntpd)
37809 ??  Sp       0:00.26 ntpd: dns engine (ntpd)
 1816 ??  I<pU     0:00.00 /usr/sbin/ntpd
63841 ??  I        0:00.01 sshd: /usr/sbin/sshd [listener] 0 of 10-100
   startups
[illegible] ??  Ip       0:00.02 /usr/sbin/smtpd
[illegible] ??  Ipc      0:00.02 smtpd: crypto (smtpd)
[illegible] ??  Ipc      0:00.02 smtpd: control (smtpd)
 5777 ??  Ip       0:00.02 smtpd: lookup (smtpd)
45996 ??  Ipc      0:00.04 smtpd: dispatcher (smtpd)
37682 ??  Ipc      0:00.02 smtpd: queue (smtpd)
97246 ??  Ipc      0:00.02 smtpd: scheduler (smtpd)
48848 ??  IpU      0:00.00 sndiod: helper (sndiod)
47188 ??  I<pc     0:00.00 /usr/bin/sndiod
96369 ??  Ip       0:00.02 /usr/sbin/cron
45067 ??  I        0:00.07 sshd: dlg [priv] (sshd)
32638 ??  S        0:00.03 sshd: dlg@ttyp0 (sshd)
 1730 p0  Sp       0:00.02 -ksh (ksh)
16990 p0  R+pU/2   0:00.00 ps -ax
33428 00  I+pU     0:00.01 /usr/libexec/getty std.9600 tty00
$
‘‘‘
```

While all processes are visible to each other, they are restricted from interacting with each other based on the user that each process is running as. A non-root user can only signal their own processes. Attempts to signal processes running as another user fails:

```
$ whoami
dlg
$ ps -U _sndio
  PID TT  STAT        TIME COMMAND
47188 ??  I<pc     0:00.00 /usr/bin/sndiod
$ kill 47188
ksh: kill: 47188: Operation not permitted
$
```

However, the root user is allowed to signal any process:

```
$ doas kill 47188
doas (dlg@comp3301.eait.uq.edu.au) password:
$ ps -U _sndio
  PID TT  STAT        TIME COMMAND
$
```

3 Zones Implementation

Zones are implemented for this assignment to add further isolation of processes. Processes running within a zone can only see and interact with processes running within the same zone, regardless of which user within the zone is running the commands. This implementation is loosely modelled on the design of Solaris Zones as described in PSARC/2002/174.

The exception to this enhanced isolation is for processes running in the "global" zone, which is the default zone that is created and exists on boot. Processes running in the global zone can see all other processes in the system, including those running in other (non-global) zones, and the root user in the global zone can signal any of these processes too. However, non-root users in the global zone cannot signal processes in other zones, even if they are running as the same user.

The provided diff implements changes to the kernel and several userland utilities and adds a `zone(8)` command and man page. The `zone(8)` command provides several sub-commands that expose the functionality of the kernel zone subsystem.

3.1 Provided Zone Syscalls

`zone_create()`

```
zoneid_t    zone_create(const char *zonename);
```

`zone_create()` creates a new zone id for use in the system, with a unique name specified by `zonename`.

`zone_destroy()`

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

`zone_destroy()` deletes the specified zone instance. The zone must have no running processes inside it for the request to succeed.

`zone_enter()`

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

`zone_enter()` moves the current process into the specified zone.

`zone_list()`

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

In the global zone `zone_list()` provides 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.

zone_name()

```
int     zone_name(zoneid_t z, char *name, size_t namelen);
```

The `zone_name()` syscall provides the name of the zone identified by the `z` argument. If run in a non-global zone the `z` id must be the identifier for the current zone. In the global zone it can be any zone identifier.

zone_id()

```
zoneid_t    zone_id(const char *name);
```

`zone_id()` provides the id associated with the `name` zone. If run in a non-global zone, only the current zone name may be specified. If `name` is a `NULL` pointer the zone id calling process is running in is returned.

zone_stats()

```
int     zone_stats(zoneid_t z, struct zstats *zstats);
```

`zone_stats()` provides an assortment of operating system statistics resulting from processes in the zone associated with the id `z`.

zone_rename()

```
int     zone_rename(zoneid_t z, char *newname);
```

`zone_rename()` alters the name of the zone identified by the `z` argument. The new name will be the name provided in the `newname` argument. `zone_rename()` handles the necessary tree updates on the zone names tree.

This syscall will be necessary for you to implement the `zone rename` subcommand.

3.2 zone(8)

```
usage:  zone create zonename
        zone destroy zonename
        zone exec zonename command ...
        zone list
        zone id [zonename]
        zone name [zid]
        zone stats [-H] [-o property[,...] zone [...]
```

The `zone(8)` program uses the zone syscalls to allow systems administrators or operators to use the zone subsystem in the kernel.

zone create

`zone create` uses the `zone_create()` syscall to create a zone with the specified name.

zone destroy

`zone destroy` uses the `zone_destroy()` syscall to create a zone with the specified name. If a zone with the specified name does not exist, `zone(8)` will attempt to interpret the argument as a numeric zone identifier.

`zone exec`

`zone exec` uses the `zone_enter()` syscall to move itself into the specified zone, and then executes the program. If a zone with the specified name does not exist, `zone(8)` will attempt to interpret the argument as a numeric zone identifier.

`zone list`

`zone list` uses the `zone_list()` syscall to fetch a list of ids for the currently running zones, and iterates over it calling the `zone_name()` syscall to print out the list of zone ids and names.

`zone name` / `zone id`

`zone name` and `zone id` use their associated syscalls `zone_name()` and `zone_id()` to return the name of a zone given its id, or the id of a zone given its name.

`zone stats`

`zone stats` uses the `zone_stat()` syscall to obtain and print out to the user a series of statistics from processes running in the current zone. See the manual page in `zone(8)` for more information.

3.3 Your Tasks

You will be adding additional functionality to a series of `zone(8)` sub-commands, adding three new `zone(8)` sub-commands, and implementing any necessary changes to the kernel zones system to support them.

Your additional functionality centers around zone permissions. Files have an associated "user" and "group", and this user or group may have permission to operate on the file. Your task is to associate zones with a particular owner and group, and allow the owner of the zone and users who are in that group to perform operations on the zone (regardless of whether they are the owner of the zone).

In short, where zones are now only controllable by root, your changes will allow the owner of a zone and a different group of users to control a zone.

The additional sub-commands you will be implementing are: `zone rename`, which will change the name of a zone; `zone chown`, which will change the owner of a zone in a manner similar to the existing `chown(8)`; and `zone chgrp`, which will change the group of a zone in a manner similar to the existing `chgrp(8)`.

4 Instructions

To complete the assignment, you will need to do the following.

4.1 Apply the diff

```
- Fetch https://stluc.manta.uqcloud.net/comp3301/public/2024/a1-zones-base.
    patch
- Create an a1 branch
  - ‘git checkout -b a1 openbsd-7.5‘
- Apply the base patch to the a1 branch
```

```
  - `git am /path/to/a1-zones-base.patch` in /usr/src
- Build the kernel
  - `cd /usr/src/sys/arch/amd64/compile/GENERIC.MP`
  - `make obj`
  - `make config`
  - `make -j 5`
  - `doas make install`
- Reboot into the kernel
  - `doas reboot`
- `make obj` in /usr/src
- `doas make includes` in /usr/src/include
  - Verify the zones syscalls are in /usr/include/sys/syscall.h
  - Verify /usr/include/sys/zones.h exists
- Make and install libc
  - `cd /usr/src/lib/libc`
  - `make -j 5`
  - `doas make install`
- Optional: make ps, and pkill/pgrep
- make zone(8)
  - `cd /usr/src/usr.sbin/zone`
  - `make`
  - `doas make install`
- Verify `zone(8)` and the zones subsystem works:
$ zone list
      ID NAME
       0 global
$ zone create
usage: zone create zonename
$ zone create test
zone: create: Operation not permitted
$ doas zone create test
doas (dlg@comp3301.eait.uq.edu.au) password:
$ zone list
      ID NAME
       0 global
   42101 test
$ zone id
0
$ zone id test
42101
$ zone exec test ps -aux
zone: enter: Operation not permitted
$ doas zone exec test ps -aux
USER       PID %CPU %MEM   VSZ   RSS TT  STAT   STARTED       TIME COMMAND
root     41705  0.0  0.1   628   580 p0  R+pU/0  3:37PM    0:00.14 ps -aux
$ doas zone exec test zone id
42101
$ doas zone exec test zone id global
zone: id: No such process
$
```

As you add the functionality specified in the next sections, some of these steps will be repeated. eg, changing the kernel means rebuilding and installing the kernel. Adding a syscall means making the syscall stub as a function visible in the headers (make includes), and callable through libc.

A note on errors

We have over-specified the errors you should return from your syscalls - if you do not require an error code (for example, never returning `ENOMEM` on memory failures because you never allocate any memory) then you do not have to use it. The reverse is also true - if you find an error case that is not listed, choose an appropriate error from `errno(2)`. We will not explicitly test all errors, but during your code interview, we will expect you to be able to explain the suitability of the error codes you use.

4.2 Zone Rename

The `zone(8)` commands should be extended to enable renaming of zones. Zones should only be able to be renamed by the owner, root, or members of the zone's group. Additionally, the global zone cannot be renamed, and zone names must be unique.

```
$ zone
usage:  zone create zonename
        zone destroy zonename
        zone exec zonename command ...
        zone list
        zone name [id]
        zone id [zonename]
        zone rename zone newname
$ doas zone create foo
$ zone list
      ID NAME
       0 global
     [illegible]9 foo
$ doas zone rename 298 bar
$ zone list
      ID NAME
       0 global
     289 bar
$ doas zone rename 0 something
zone: rename:  Invalid Argument
$ doas zone rename 289 global
zone: rename: File exists
```

4.3 Modifications to Existing Syscalls

`zone_create()` syscall

The `zone_create()` syscall should now ensure that the created zone is associated with the group of the user that created it, as well as the user themself. Additionally, this will mean ensuring that non-root users can create zones. The definition of `zone_create()` should not change - it should still take a single `char *zonename` as its argument.

All other syscalls

The full suite of `zone_*` syscalls should permit users with matching credentials (owner or group) to perform zone operations on them, not only the root user. The credentials may be changed so appropriate synchronisation should be used. Namely, we expect that, unless credentials are being changed by another thread, authorisation should be non-blocking.

4.4 Zone name and zone list

`zone_name()` syscall

The `zone_name()` syscall should be renamed to `zone_info()`. Subsequently, it should return not only the name and namelen, but also the zone, user and group id, preferably all bundled in a struct format. However you may pass back one or more of these as individual parameters if that is easier. The `zone(8)` userland sub-command for `zone name` should also be modified in line with these changes - the name should be changed to `zone info` and the additional information should be provided to the user. Alternatively, you may also create zone info as an independent command.

`zone list`

The `zone list` subcommand should now take flags: `-o` and `-g`. If the `-o` flag is provided, the owner of the zone should be printed, and if the `-g` flag is provided, the zone's group should be printed. If both flags are provided, print both. The extra fields should be printed as extra columns in the current table format. zone id and name must be displayed first. However, the order of the additional fields does not matter.

4.5 Zone chown and chgrp

The `zone(8)` commands and the kernel zones system should be extended to enable changing the owner and group of a zone. Zone owners and groups should only be able to be changed by the owner, root, or members of the zone's group. Additionally, the owner of the global zone cannot be changed.

```
$ zone
usage: zone create zonename
       zone destroy zonename
       zone exec zonename command ...
       zone list
       zone name [zoneid]
       zone id [zonename]
       zone chown zone user
       zone chgrp zone group
```

The two subcommands you are adding are `zone chown` and `zone chgrp`. `zone chown` takes the name of a zone and uses the `zone_chown()` syscall to change its owner to the user with the specified name. If a zone with the name `zonename` does not exist, `zone(8)` will attempt to interpret the argument as a numeric zone identifier.

`zone chgrp` behaves similarly, but instead, it uses the `zone_chgrp()` syscall to change the zone's group to the specified group name.

To support these subcommands, you will need to implement the following system calls:

`zone_chown()` syscall

```
int     zone_chown(zoneid_t z, uid_t user);
```

The `zone_chown()` syscall alters the owner of the zone identified by the `z` argument. The new owner should be the owner identified by the `user` argument. If called from a non-global zone, then the `z` id must be the identifier for the current zone, but in the global zone, it can be any zone identifier. This means that to the user, a non-global zone should only be able to see itself.

Potential Errors:

- EPERM - the user does not have permission to alter the zone `z`
- ESRCH - the zone identified by `z` does not exist
- ENOMEM - the system was not able to allocate memory
- EINVAL - the zone to alter was the global zone

`zone_chgrp()` syscall

```
int     zone_chgrp(zoneid_t z, gid_t group);
```

The `zone_chgrp()` syscall alters the owner of the zone identified by the `z` argument. The new owner should be the group identified by the `group` argument. If called from a non-global zone, then the `z` id must be the identifier for the current zone, but in the global zone, it can be any zone identifier. This means that to the user, a non-global zone should only be able to see itself.

Potential Errors:

- EPERM - the user does not have permission to alter the zone `z`
- ESRCH - the zone identified by `z` does not exist
- ENOMEM - the system was not able to allocate memory
- EINVAL - the zone to alter was the global zone

5 Other Requirements & Suggestions

5.1 Code Style

Your code is to be written according to OpenBSD's style guide, as per the `style(9)` man page. An automatic tool for checking for style violations is available at:
`https://stluc.manta.uqcloud.net/comp3301/public/2022/cstyle.pl`
This tool will be used to calculate your style marks for this assignment.

5.2 Compilation

Your code for this assignment is to be built on an `amd64` OpenBSD 7.5 system identical to your course-provided VM.

The following steps must succeed:

- `make obj; make config; make` in src/sys/arch/amd64/compile/GENERIC.MP
- `make obj; make includes` in src
- `make obj; make; make install` in src/lib/libc
- `make obj; make; make install` in src/usr.sbin/zone

The existing `Makefiles` in the provided code are functional as-is, but may need modification as part of your work for this assignment. Note that the existing Makefile ensures the `-Wall` flag is passed to the compiler, as well as a few other warning and error-related flags.

5.3 Provided code

The provided code, which forms the basis for this assignment, can be downloaded as a single patch file at:
https://stluc.manta.uqcloud.net/comp3301/public/2024/a1-zones-base.patch

You should create a new `a1` branch in your repository based on the `openbsd-7.5` tag using `git checkout`, and then apply this base patch using the `git am` command:

```
$ git checkout -b a1 openbsd-7.5
$ ftp https://stluc.manta.uqcloud.net/comp3301/public/2024/a1-zones-base.
    patch
$ git am < a1-zones-base.patch
$ git push origin a1
```

5.4 Recommendations

The following order will likely be the most reasonable way to complete this assignment:

1. Download, build, and install the zones patch.
2. Add the `zone rename` subcommand to `zone(8)`.
3. Minimally modify `zone_create()` to store credentials.
4. Rewrite `zone_name()` to `zone_info()`.
   This ensures you have a way to view the credentials of a zone.
5. Add the `zone_chown()` and `zone_chgrp()` syscalls.
6. Add the corresponding `zone chown` and `zone chgrp` commands to `zone(8)`.
7. Fix up any tiny bugs and ensure it's all working. But you did that as you were going... right?

Additionally, it is strongly recommended (and in some cases, required) that the following APIs be considered for use as part of your changes:

- `sys/ucred.h` - provides necessary handlers for dealing with user and group credentials
- `copyin(9)/copyout(9)` - provides the ability to copy data across the userspace boundary
- `user_from_uid(3)` - conversions from group/user name to id and back
- `strtonum(3)` - BSD style safe string to int conversions
- Finally, you may wish to look at the header file `sys/proc.h` to see how user and group credentials are currently stored by threads.

6 Reflection

Provide a reflection on your implementation by briefly answering the following questions:

1. Describe the steps you took or draw a flowchart.
2. Describe an error that you encountered.
3. Describe how the error was debugged.
4. Describe how the bug was solved.

Upload your answers as a pdf to the Blackboard a1 reflection submission. Page length is a maximum 2 pages or less. **Pdf name must be your STUDENT_NUMBER_a1.pdf.** Note this is your XXXXXXXX ID number and not sXXXXXXX login.

7 Submission

Submission must be made electronically by committing to your course-provided Git repository on `source.eait.uq.edu.au`. In order to mark your assignment the markers will check out the `a1` branch from your repository. Code checked into any other branch in your repository will not be marked.

As per the `source.eait.uq.edu.au` usage guidelines, you should only commit source code and Makefiles.

Your `a1` branch should consist of:

- The openbsd-7.5 base commit
- The A1 base patch commit
- Your commit(s) for adding the required functionality

7.1 Marking

Your submission will be marked by course tutors and staff, during an in-person demo with you, at your lab session during the due week. You must attend your session, in-person, otherwise your submission will not be marked. Online attendence, e.g. zoom, is not permitted.