COMP3301 Assignment 1

Syscalls and Scheduling Priority - A *Nice* Assignment?

Due: 12pm Monday in Week 5 (25th of August)
Submission: Git (code) and Blackboard (reflection)
Code submission is marked in your prac session in week 5
Last Updated: August 13, 2025

1 Academic Integrity

All assessments are **individual**. You should feel free to discuss aspects of C programming and assessment specifications with fellow students and discuss the related APIs in general terms. You should not actively help (or seek help from) other students with the actual coding of your assessment. 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 will be subject to automated checks for plagiarism and collusion. If we detect plagiarism or collusion (outside of the base code given to everyone), formal misconduct proceedings will be initiated against you. If you're having trouble, seek help from a teaching staff member. Do not 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 website: https://eecs.uq.edu.au/current-students/guidelines-and-policies-students/student-conduct.

Do not post your code to a public place such as the course discussion forum or a public code repository. (Code in private posts to the discussion forum is permitted.) You must assume that some students in the course may have very long extensions so do not post your code to any public repository. You must follow the following code usage and referencing rules for all code committed to your Git repository (not just the version that you submit), in Table 1.

Table 1: Code Origin and Uses/References

Code Origin	**Usage/Referencing**
Code provided by teaching staff this semester Code provided to you by COMP3301 teaching staff or posted on the discussion forum by teaching staff.	Permitted May be used freely without reference. (You must be able to point to the source if queried about it - so you may find it easier to reference the code.)
Code you wrote this semester for this course Code you have personally written this semester for COMP3301.	Permitted It may be used freely without reference, provided you have not shared or published it.
Unpublished code you wrote earlier Code you have personally written in a previous enrollment in this course or in another UQ course or for other reasons and where that code has not been shared with any other the person or published in any way or submitted for assessment.	Conditions apply; requires references May be used provided you understand the code AND the source of the code is referenced in a comment adjacent to that code. If such code is used without appropriate referencing, then this will be considered misconduct.
C Code from AI tools	Conditions apply; requires references

Whilst students may use AI and/or MT technologies, successful completion of the assessment in this course will require students to critically engage in specific contexts and tasks for which artificial intelligence will provide only limited support and guidance.	**A failure to reference generative AI or MT use may constitute student misconduct under the Student Code of Conduct.** To pass this assessment, students will be required to demonstrate detailed comprehension of their submission, independent of AI and MT tools.
Code copied from other sources Code, in any programming language: • copied from any website or forum (including Stack Overflow and CSDN); • copied from any public or private repositories; • copied from textbooks, publications, videos, apps; • copied from code provided by teaching staff only in a previous offering of this course; • written by or partially written by someone else or written with the assistance of someone else (other than a teaching staff member); • written by an AI tool that you did not personally and solely interact with; • written by you and available to other students; or • from any other source besides those mentioned in earlier table rows above.	Prohibited May not be used. If the source of the code is referenced adjacent to the code then this will be considered code without academic merit (not misconduct) and will be removed from your assignment prior to marking (which may cause compilation to fail and zero marks to be awarded). Copied code without adjacent referencing will be considered misconduct and action will be taken. This prohibition includes code written in other programming languages that has been converted to C.
Code that you have learned from Examples, websites, discussions, videos, code (in any programming language), etc. that you have learned from or that you have taken inspiration from or based any part of your code on but have not copied or just converted from another programming language. This includes learning about the existence of and behaviour of library functions.	Conditions Apply, references required May be used provided you do not directly copy code AND you understand the code AND the source of the code or inspiration or learning is referenced in a comment adjacent to that code. If such code is used without appropriate referencing then this will be considered misconduct.

2 Introduction

In this assignment, you will be extending an existing implementation of *zones*, provided to you in the form of a base code patch, to include additional features.

This assignment serves as an introduction to the kernel and kernel development workflows, and aims to strengthen your understanding of syscalls and priority scheduling covered in lectures 1 and 3. It also assesses your skills in reading, understanding and modifying existing code.

3 Zones Overview

3.1 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 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        0:00.16 /sbin/init
47423 ??  IpU      0:00.01 syslogd: [priv] (syslogd)
 4805 ??  Spc      0:00.12 /usr/sbin/syslogd
17662 ??  IU       0:00.00 pflogd: [priv] (pflogd)
18233 ??  I<pc     0:00.24 ntpd: ntp engine (ntpd)
82862 ??  Ip       0:00.02 ntpd: dns engine (ntpd)
62773 ??  I<pU     0:00.08 /usr/sbin/ntpd
69091 ??  Sp       0:00.17 /usr/sbin/cron
59488 ??  I        0:00.03 sshd-session: uqygao13 [priv] (sshd-session)
91575 ??  S        0:00.18 sshd-session: uqygao13@ttyp0 (sshd-session)
92672 p0  Sp       0:00.02 -ksh (ksh)
18318 p0  R+pU/0   0:00.00 ps -ax
36658 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
uqygao13
$ ps -U _sndio
  PID TT  STAT        TIME COMMAND
86619 ??  I<pc     0:00.00 /usr/bin/sndiod
$ kill 86619
ksh: kill: 86619: Operation not permitted
```

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

```
$ doas kill 86619
$ ps -U _sndio
  PID TT  STAT        TIME COMMAND
```

3.2 Zones

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 modeled 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.

Zones, much like files, have owner and group information. Only the owner of a zone, or members of the group associated with the zone, can make changes to it (i.e., change ownership/group, enter or destroy).

The provided base code 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.

3.3 Existing Syscalls

`zone_create`

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

Creates a new zone with a unique name specified by `zonename`, and returns its ID on success or `-1` on error. The newly created zone will have its owner and group set to the user and group of the process which created the zone.

`zone_destroy`

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

Destroys the specified zone instance. The zone must have no running processes inside it for the request to succeed. The caller must also either be root, the owner of the zone or a member of the group associated with the zone for this syscall to succeed. Returns `0` on success and `-1` on error.

`zone_enter`

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

Moves the current process into the specified zone. The caller must either be root, the owner of the zone or a member of the group associated with the zone for this syscall to succeed.

`zone_list`

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

Provides the list of zones in the running system as an array of `zoneid_t`s. If called from a process running in the global zone, the IDs of all zones are returned, otherwise only the ID of the current zone is returned. Returns `0` on success and `-1` on error.

`zone_info`

```
int         zone_info(zoneid_t z, struct zinfo* info);
```

Provides the information regarding the zone identified by the ID z in a `zinfo` structure:

```
struct zinfo {
    zoneid_t    zi_id;                      /* identifier */
    char        zi_name[MAXZONENAMELEN];    /* name */
    uid_t       zi_owner;                   /* owner */
    gid_t       zi_group;                   /* group */
    time_t      zi_ctime;                   /* creation time */
    int         zi_priority;                /* scheduling priority */
};
```

If called from the global zone, z can be any zone ID, however it must be the ID of the current zone if called from a non-global zone. Returns `0` on success and `-1` on error.

`zone_id`

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

Returns the ID associated with the zone named name. Any zone name can be specified if called from the global zone, otherwise only the name of the current zone can be specified. If name is `NULL`, the ID of the current zone is returned. Returns `-1` on error.

`zone_chown`

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

Changes the owner of the zone identified by z to user. Any zone ID can be specified if called from the global zone, otherwise only the current zone may be specified. The global zone cannot have its ownership changed. The caller must also either be root, the owner of the zone or a member of the group associated with the zone for this syscall to succeed. Returns `0` on success and `-1` on error.

`zone_chgrp`

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

Changes the group of the zone identified by z to group. Any zone ID can be specified if called from the global zone, otherwise only the current zone may be specified. The global zone cannot have its group changed. The caller must also either be root, the owner of the zone or a member of the group associated with the zone for this syscall to succeed. Returns

0 on success and -1 on error.

3.4 zone(8)

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

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

zone create

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

zone destroy

Uses the `zone_destroy()` syscall to destroy the specified zone. The zonename argument is first interpreted as a name, and if not found, is then interpreted as a numeric zone identifier.

zone exec

Uses the `zone_enter()` syscall to move itself into the specified zone, and then executes the program. The zonename argument is first interpreted as a name, and if not found, is then interpreted as a numeric zone identifier.

zone id

Uses the `zone_id()` syscall to return the name of a zone given its ID. If the zonename argument is not given, the current zone ID is returned.

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_info()` syscall to print out the ID and name of each zone.

zone chown / zone chgrp

Uses the `zone_chown()` and `zone_chgrp()` syscalls to change the ownership or group of the specified zone. The zonename argument is first interpreted as a name, and if not found, is then interpreted as a numeric zone identifier. The owner/group argument can either be a string or a numerical identifier.

4 Instructions

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

1. Download the base code patch

   ```
   $ cd ~
   $ ftp https://stluc.manta.uqcloud.net/comp3301/public/a1-base.patch
   ```

2. Create the `a1` branch and apply base code patch

   ```
   $ cd /usr/src
   $ git checkout -b a1 base
   $ git am < ~/a1-base.patch
   ```

3. Make object directories

   ```
   $ make obj
   ```

4. Build and install the updated kernel

   ```
   $ cd /usr/src/sys/arch/amd64/compile/GENERIC.MP
   $ make config
   $ make -j4
   $ doas make install
   $ doas reboot
   ```

5. Build and install the updated includes and `libc`

   ```
   $ cd /usr/src/include
   $ doas make includes
   $ cd /usr/src/libexec/ld.so
   $ make
   $ doas make install
   $ cd /usr/src/lib/libc
   $ make -j4
   $ doas make install
   ```

6. Build and install the updated `ps(1)` and `pkill(1)`/`pgrep(1)`

   ```
   $ cd /usr/src/bin/ps
   $ make
   $ doas make install
   $ cd /usr/src/usr.bin/pkill
   $ make
   $ doas make install
   ```

7. Build and install `zone(8)`

   ```
   $ cd /usr/src/usr.bin/zone
   $ make
   $ doas make install
   ```

5 Assignment Tasks

5.1 Zone List Extension

In its current state, `zone list` lists the ID and name of all zones in the system, which is both inflexible and inconvenient. There exists more properties of zones than just name and ID, such as ownership, group and creation timestamp, and users may only wish to query about a particular zone rather than all zones in the system.

Extend the `zone list` sub-command of `zone(8)` to allow flexible printing of zone details. The `zone list` subcommand shall now take the following parameters (you must update the usage displayed for `zone list`):

```
$ zone
        ...
        zone list [-cgHilo] [zonename ...]
        ...
```

If no parameters are specified, the `zone list` command shall output the names of the zones, one on each line (with no list heading), for example:

```
$ zone list
global
secret
database
remote
ass1
```

If no zone name is specified, all zones are listed, otherwise an optional list of zone names can be specified, such that only the specified zones are listed. If a zone with the specified name cannot be found, it is then treated as a numerical identifier, and failing that, the string `Specified zone "<zone-name>" does not exist.` is printed to `stderr` and printing continues for the next specified zone, but the command exits with code `1` in the end to indicate the error.

```
$ zone list secret ass1 database
secret
ass1
database
$ zone list secret mobile remote
secret
Specified zone "mobile" does not exist.
remote
```

If any of the following options are specified, then additional information shall be printed:

- `-i`: Include the IDs of the zones in the list.
- `-o`: Include the names of the users which own the zones in the list. If a user does not exist on the system, its numerical ID is printed.

- `-g`: Include the names of the groups which own the zones in the list. If a group does not exist on the system, its numerical ID is printed.
- `-c`: Include the creation time of the zones in the form of `yyyy/mm/dd HH:MM:SS` in the list. Time printed should be local time.
- `-l`: Use long list format, which means to include all properties of zones (name, ID, owner, group and creation time). This option is the equivalent of `-iogc`.

Options can be specified together or as separate arguments (e.g., `-og` is equivalent to `-o -g`). The properties regarding each zone must be printed in the following order: ID, name, owner, group, creation time. If any of the properties (other than name, which is always printed) is to be printed, an additional heading row must also be outputted, unless disabled by specifying `-H` as an option:

```
   ID NAME       OWNER    GROUP    CTIME
```

The `ID` column is right-aligned and space-padded on the left, and all other columns are left-aligned and space-padded on the right. The last column to be displayed is not space-padded on the right. There is a space separating each column, and the columns have the following widths in characters (excluding the separating space):

Column	Width
`ID`	5
`NAME`	10
`OWNER`	8
`GROUP`	8
`CTIME`	19

It is up to you what happens if something (e.g., name, owner or group) exceeds the column width - we will not test these cases.

```
$ zone list -l
   ID NAME       OWNER    GROUP    CTIME
    0 global     root     wheel    2025/07/15 21:55:02
  217 secret     uqmdsouz comp3301 2025/07/16 18:21:14
 6295 database   uqygao13 comp3301 2025/07/16 18:37:20
93538 remote     admin    admin    2025/07/21 13:43:27
 3380 ass1       s4743447 student  2025/07/27 09:16:31
$ zone list -l ass1 remote
   ID NAME       OWNER    GROUP    CTIME
 3380 ass1       s4743447 student  2025/07/27 09:16:31
93538 remote     admin    admin    2025/07/21 13:43:27
$ zone list -l mobile
   ID NAME       OWNER    GROUP    CTIME
Specified zone "mobile" does not exist.
$ zone list -g -i ass1 remote
   ID NAME       GROUP
 3380 ass1       student
93538 remote     admin
$ zone list -iogH secret database ass1
  217 secret     uqmdsouz comp3301
 6295 database   uqygao13 comp3301
 3380 ass1       s4743447 student
```

The order which zones are listed in does not matter, same goes with error messages from listing zones which do not exist. We will not test cases where the same option or zone name is specified multiple times, you may either allow these cases or treat them as errors. Errors relating to invalid command-line arguments however should be handled like other `zone(8)` sub-commands.

5.2 Scheduling Priority Background

OpenBSD, like other UNIX-like operating systems, allows users to change the priority of processes, which in turn affects how they are scheduled relative to other processes. The syscalls `getpriority(2)` and `setpriority(2)` can be used to get and set the priority of process(es).

```
int     getpriority(int which, id_t who);
int     setpriority(int which, id_t who, int prio);
```

The scheduling priority of the process, process group, or user, as indicated by `which` and `who` is obtained with the `getpriority()` call and set with the `setpriority()` call. `which` is one of `PRIO_PROCESS`, `PRIO_PGRP`, or `PRIO_USER`, and `who` is interpreted relative to `which` (a process identifier for `PRIO_PROCESS`, process group identifier for `PRIO_PGRP`, and a user ID for `PRIO_USER`). A zero value of `who` denotes the current process, process group, or user. `prio` is a value in the range -20 to 20. The default priority is 0; lower priority values means higher priority.

The `getpriority()` call returns the highest priority (lowest numerical value) enjoyed by any of the specified processes. The `setpriority()` call sets the priorities of all of the specified processes to the specified value. Priority values outside the range -20 to 20 are capped at the appropriate limit. Only root may lower priority values (increase priority).

The numerical scheduling priority value between -20 and 20 is also known as the "nice" value, the higher the value, the "nicer" you are to other processes. You can execute a program with a specific nice value using `nice(1)`, and the nice values of running processes can be changed by `renice(8)`, which uses the aforementioned syscalls. Nice values of processes can be displayed by running `ps -l`, under the `NI` column.

Note that `ps(1)` has a `PRI` column and an `NI` column, all mentions of "priority" in this specification refers to the nice value (`NI` column). The `PRI` column is something else, which you do not need to worry about for this assignment.

5.3 Zone Priority

Priority Getting

Extend the `zone` structure to include an integer member for storing zone priority, and set the priority value to 0 when a new zone is created. This priority value has the same meaning as the scheduling priority (nice value) of processes.

Modify the `zone_info()` syscall to populate the `.zi_priority` field of the `zinfo` structure with the zone priority upon invocation.

Note: Zone priority refers to the priority of the zone, a value stored in the `zone` structure. Process priority refers to the priority of an individual process. While normally all processes within a zone would have the same priority as the zone, it is possible to change a process' priority manually, at which point its priority no longer matches that of the zone.

Priority Setting

Implement the following syscall:

```
int     zone_setpri(zoneid_t z, int prio);
```

Sets the zone's priority to `prio` and the priority of all processes within the zone to be that of `prio`. `prio` has the same meaning as the `prio` argument of `setpriority(2)`, and is capped at limits.

The global zone cannot have its priority changed, and `errno` shall be set to `EPERM` for any attempt to set the global zone's priority, regardless of the value. The caller must also either be root, the owner of the zone or a member of the group associated with the zone for this syscall to succeed, and only root may set a priority value lower than the current priority value (increase the priority). If a non-root user attempts to lower the priority value of a zone, the syscall fails and `errno` is set to `EPERM`.

If setting the priority of the zone as a non-root user would cause the priority value of a process to decrease, then that process is skipped and `errno` is set to either `EACCES` or `EPERM` the end. If called from the global zone, any zone ID may be specified, otherwise only the ID of the current zone may be specified. Returns `0` on success and `-1` on error.

Extend the zone subsystem such that when a process is moved into a particular zone from the global zone, its priority is changed to be the zone's priority.

Zone Command Extension

Extend the `zone(8)` command to add a new sub-command `zone setpri`, which calls `zone_setpri()`:

```
$ zone
usage:  zone create zonename
        zone destroy zonename
        zone exec zonename command ...
        zone id [zonename]
        zone list [-cgHilo] [zonename ...]
        zone chown zonename user
        zone chgrp zonename group
        zone setpri zonename pri
```

Nothing should be printed upon a successful invocation of this sub-command. If a zone named `zonename` does not exist, then `zonename` is taken to be a numerical zone identifier, and failing that, it is considered as an error. Invalid priority values are also considered as errors. How errors are handled is up to you, but all errors must be handled appropriately and reasonably.

Existing Interface Extension

Extend the `getpriority(2)` and `setpriority(2)` syscalls to allow getting/setting of priorities at zone level. If `PRIO_ZONE` (already defined) is specified as `which`, then priority is get/set at zone level, and `who` is taken to be the zone ID.

The behaviour of `getpriority(PRIO_ZONE, ...)` shall be consistent with the rest of the syscall, that is, return the highest priority (lowest value) of all processes within the zone, regardless of the priority of the zone itself. The behaviour of `setpriority(PRIO_ZONE, ...)` shall match that of `zone_setpri()`.

These syscalls should accept all zone IDs if called from the global zone, otherwise only the ID of the current zone shall be accepted, to align with the concept that processes within non-global zones could only see the zone which they are running in. You must also ensure that only authorised users (root, owner or group member of the zone) could set priorities.

5.4 Reflection

Reflect on your implementation by answering the following questions:

1. How did you locate the files which you have modified in your implementation? List the tools (e.g., `cscope`, `grep`) and steps.
2. What bugs have you encountered? Describe one bug. If you did not encounter any bugs, describe something which you believe was tricky.
3. How did you debug the bug which you have identified earlier? List the steps taken and tools used (e.g., `ddb`, `printf`). If you did not encounter any bugs, describe the steps you would take and tools you would use to debug a kernel panic.
4. How did you fix the bug which you have identified and debugged earlier? If you did not encounter any bugs, describe some differences between debugging userland programs and kernels.

These questions are open-ended, there is no expected answer. Upload your reflection as a PDF file to the Blackboard A1 reflection submission. Page length is a maximum of 2 pages or less. **PDF name must be STUDENT_NUMBER_A1.pdf.** Note this is your `4XXXXXXX` ID number and not your `s4XXXXXX` login.

6 Misc. Requirements

6.1 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/cstyle.pl`.

Code style marks will be calculated based on the number of style violations in the code which you have written yourself or modified - style violations in the OpenBSD source tree or in the base code will not affect code style marks. Some level of functionality is required to score marks for code style (i.e., no submission implies no style violations, however no style marks will be awarded in that case).

6.2 Reliability, Robustness, Portability and Modularity

In order to score higher marks, your code is expected to be reliable and robust, that is it should handle all errors appropriately and should not crash unexpectedly. Your code should also be portable and modular, in the sense that constants should not be hard coded and similar code should not be duplicated in multiple areas. Your code should also be free of race conditions and undefined behaviour.

6.3 Compilation (Pass/Fail)

Your code must be compile-able under the `GENERIC.MP` (generic multiprocessor) configuration for the AMD64 (aka x86-64 and x64) architecture. Code with compile-time errors will be marked manually, and may result in heavy penalties up to the deduction of all functionality marks.

7 Git Submission

Submission must be made electronically by committing and pushing your changes to your course-provided Git repository on `source.eait.uq.edu.au`. Code checked into any other branch in your repository or not pushed to the `a1` branch (i.e., left on your VM) will not be marked.

As per the `source.eait.uq.edu.au` usage guidelines, you should only commit source code and makefiles. Please do not commit `cscope` outputs, core dumps or base code patch files.

Your `a1` branch should consist of:

- The OpenBSD base commit (provided)
- The A1 base patch commit (provided)
- Commit(s) for adding the required functionality (by yourself)

Commit history and commit messages do not contribute to your grade. However, it is strongly recommended that you commit frequently and use appropriate commit messages.

7.1 Marking

Your submission will be marked by course staff during an in-person demo with you at your prac session of the due week. You must attend your session in person; otherwise, your submission will not be marked. Online attendance (e.g., via Zoom) is not permitted.

7.2 Demo Session

You will be asked to demonstrate your assignment as part of your regular practical session. Demonstrations will run on a marking VM (not your VM), which has nothing but the latest code which you have pushed to the `a1` branch before the submission deadline. Manual and automated tests and manual code examinations will be conducted during the demo. Changes to your code during the session will not be accepted.

It is your responsibility to ensure that your code compiles and operates correctly. Code that fails to compile or code that crashes the kernel upon boot results in your submission not being marked for functionality.

You may be asked to explain certain aspects of your submission during the demo session. Most of the questions will be open-ended and have no single correct answer. However, failure to demonstrate your understanding of the code which you have presumably written or a wrong understanding of the concepts yet correct code may result in allegations of academic misconduct.

8 Recommendations

8.1 Backups

It is *highly recommended* that you use Git to regularly backup your assignment code. Beware that corruptions to your file-system can happen in the event of kernel crashes, and will require you to start from scratch if there is no backup. Regularly committing code to Git and pushing to origin ensures that your code will not be lost in the event of a file-system corruption.

It is also recommended that you take a snapshot of your virtual machine (VM) before you attempt the programming aspect of this assignment. This can be done by running `snapshot <name>` in your VM Control.

8.2 Relevant Manual Pages

The following manual pages may be useful for the understanding of existing functionalities or the implementation of the assignment. You are not required or expected to use all of these in the code which you write.

- getopt(3) - Get Option Character
- err(3) - Formatted Error Messages
- getpriority(2) - Priority Getting Syscall

- setpriority(2) - Priority Setting Syscall
- TAILQ_INIT(3) - Doubly-Linked List Macros
- RB_PROTOTYPE(3) - Red-Black Tree Macros
- KASSERT(9) - Kernel Assert Routines
- rwlock(9) - Read/Write Lock

8.3 Testing

Public tests are provided for this assignment. It is suggested that you run the command `sync` before running any of the tests (or any test program which you write), to minimise the chance of file system corruption in the event of an unexpected kernel crash.

The public tests only test the very basic functionalities of your implementation, and therefore your implementation is not guaranteed to be fully correct even if you pass all the public tests. You are encouraged to write your own test cases.

Installation

The public tests can be installed by running the following commands in your VM:

```
$ cd ~
$ ftp https://stluc.manta.uqcloud.net/comp3301/public/a1_public_tests.tar
$ tar -xvf a1_public_tests.tar
$ cd a1test
$ ./install
```

This will install the test program `a1test` and 24 public tests.

`a1test` Test Program

The `a1test` program takes the following command-line arguments:

```
$ a1test <public|custom|all> <run|info|dump> <all|name,name2...>
```

The first argument must be one of `public`, `custom` or `all`. `public` tells the test program to search for public tests, `custom` tells the test program to search for custom tests (which you can write yourself) and `all` tells the test program to search for all tests.

The second argument must be one of `run`, `info` or `dump`. `run` executes the selected test(s), `info` prints out the information about the selected test(s) and `dump` dumps your outputs (`stdout` and `stderr`) and the expected outputs of the selected test(s) to the current directory (in case you want to run `diff(1)` or `sdiff(1)`).

The third argument is a list of tests to run/info/dump, or `all` to run/info/dump all tests. The list is comma separated.

Example Test Commands

To run:

- All test, you run `a1test all run all`.
- A particular test (e.g., 3.2), you run `a1test all run 3.2`.
- A selection of tests (e.g., 1.5, 3.2 and 4.1), you run `a1test all run 1.5,3.2,4.1`.

To display the information about:

- All tests, you run `a1test all info all`.
- A particular test (e.g., 1.6), you run `a1test all info 1.6`.
- A selection of tests (e.g., 1.3, 2.0 and 3.1), you run `a1test all info 1.3,2.0,3.1`.

To dump your outputs to the current directory for:

- All tests, you run `a1test all dump all`.
- A particular test (e.g., 1.4), you run `a1test all dump 1.4`.
- A selection of tests (e.g., 1.1, 3.3, 4.0), you run `a1test all dump 1.1,3.3,4.0`.

Examples of passed and failed tests with screenshots are in the A1 Tests thread on Ed.

9 Specification Clarifications

Specification clarifications and/or updates may be issued, they will be communicated through Blackboard and/or Ed. If needed, you are encouraged to seek for spec clarifications in the A1 Spec Clarification Megathread on Ed rather than by making individual posts or emailing teaching staff.

All clarifications made since the initial release of this specification are coloured red in this document.

10 Appendix