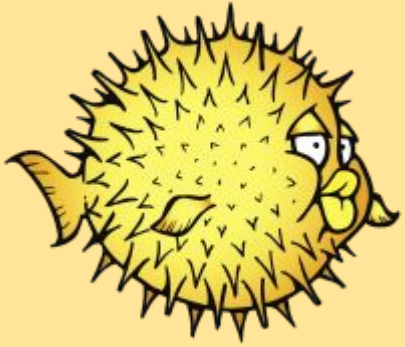


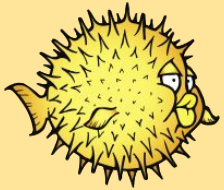
OpenBSD Zones

COMP3301 - Assignment 1

OpenBSD

COMP3301 - 2025

The University Of Queensland
AUSTRALIA

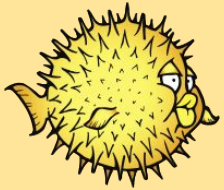


Academic Integrity and Plagiarism

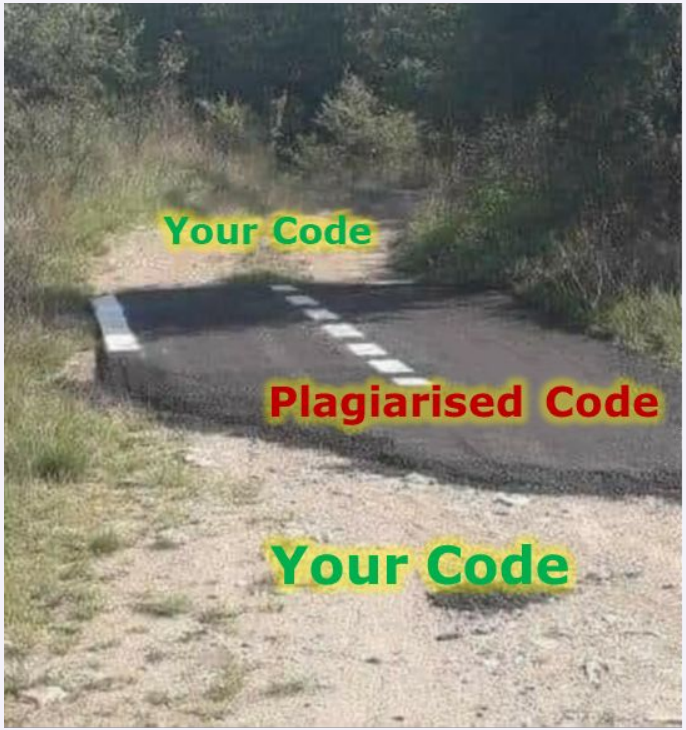
- Assessments are individual.
- Feel free to discuss aspects of C programming and assignment spec generally.
- Looking at another student's work is cheating (electronic or printing)
- All code submitted goes through automated checks for plagiarism and collusion
- If plagiarism or collusion is detected formal misconduct proceedings will be initiated
- If you're having trouble seek help from teaching staff
- Don't be tempted to copy another students code
- Official policies can be found here:

https://eecs.uq.edu.au/current-students/guidelines-and-policies-students/student-conduct

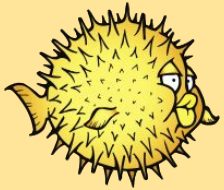
.



Your Code In Our Eyes

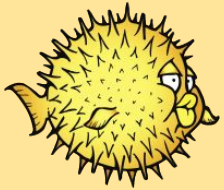


- All of us have years if not decades of C experience
- We will catch you if you cheat, not a joke
- We catch dozens of students every year



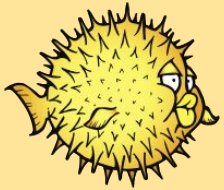
Introduction

- You will be extending an existing implementation of *zones* to include additional features
- Base code provided
- Serves as an introduction to the kernel and kernel development workflows
- Aims to strengthen your understanding of syscalls and priority scheduling
- Assesses your skills in reading, understanding and modifying existing code

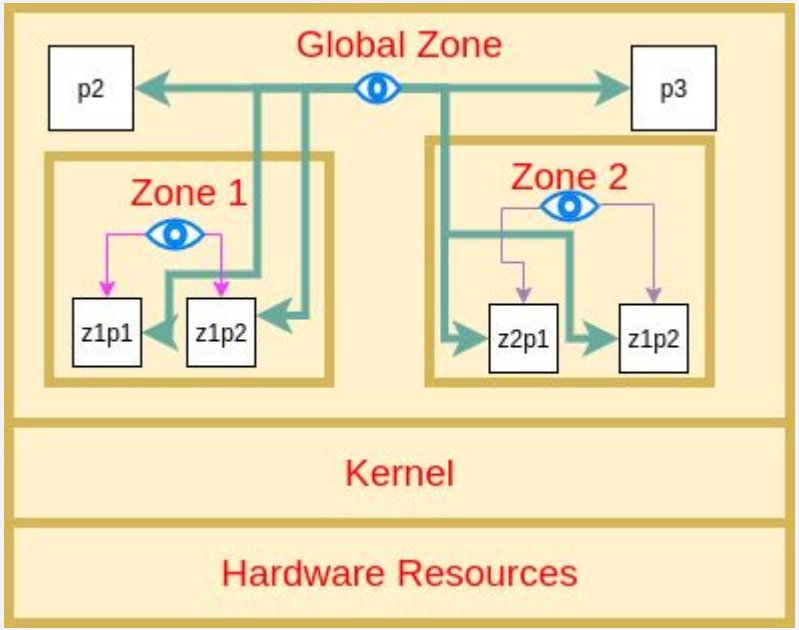


Zones – What Are They?

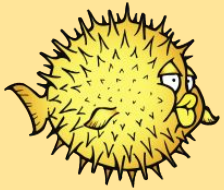
- A “container” for processes
- Essentially, any process that runs in a zone is isolated to that zone and can’t see any other zones
- From the universal “global” zone, you can see every process
- Zones are only aware of their own existence, and the processes that execute within themselves
- Why? – It’s lightweight, and creates an environment where you can execute processes you only want to be visible by other processes in the said zone



COMP3301 Zones



- Zones separate each execution environment
- Zones wrap processes to exist within themselves
- Processes in the global zone can see all other processes
- Processes within a non global zone can only see other processes in that zone
- Zones all have an associated owner and group that they belong to
- The global zone always belongs to root



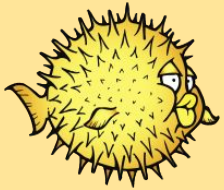
COMP3301 Zones - **zone(8)**

zone(8)

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

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

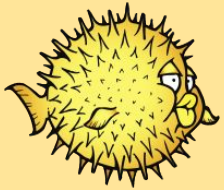
See spec for more details



COMP3301 Zone commands

- zone create zonename
  - Create a zone named zonename
- zone destroy zonename
  - Destroy the zone named zonename
- zone exec zonename command …
  - Run a command in the zone zonename
- zone id [zonename]
  - Get the id of zonename, or the current zone if none specified
- zone chown zonename user
  - Set zonename's owner to user
- zone chgrp zonename group
  - Set zonename's group to group

See spec for more details



COMP3301 Zone Syscalls (Existing)

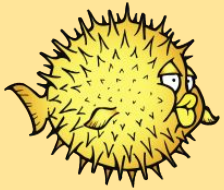
- zoneid_t zone_create(const char *zonename);

@return zoneid_t id of the zone that was created

- int zone_destroy(zoneid_t z)
- int zone_enter(zoneid_t z);
- int zone_list(zoneid *zs, size_t *nzs);
- int zone_info(zoneid_t z, struct zinfo* info)
- int zone_chown(zoneid_t z, uid_t owner);
- int zone_chgrp(zoneid_t z, gid_t group);

@return int returns -1 on fail, 0 otherwise

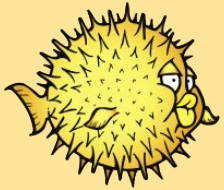
See spec for more details



Struct zinfo

```
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 */
};
```

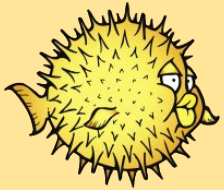
See spec for more details



Assignment 1 – Task Overview

- **Programming Component**
  - **Zone list extension (pure userland)**
  - **Zone priority (kernel and userland)**
    - Zone priority getting
    - Zone priority setting
    - Zone command extension
    - getpriority(2) and setpriority(2) extension
  - **Robustness, Portability, Modularity**
  - **Code Style**
- **Reflection**

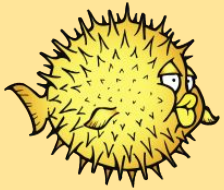
See spec for more details



Functionality - Zone list extension

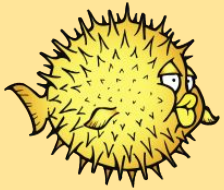
- Expand the zone list *subcommand* to show different elements contained in the zinfo struct given particular parameters.
  - You don't have to write your own syscall
- What we are looking for:
  - you can read and understand userland code,
  - Expand on already existing functions
- Essentially you will need to:
  - Parse arguments and options
  - Print them in a nice table using string formatting with printf

See spec for more details



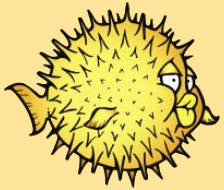
Nice Values and Priority - Scheduling and Priority

- Essentially a “priority” we give the process to execute as
- However a static priority can be problematic – more detail in lectures
  - Starvation, Priority Inversion etc. can occur
  - Scheduler changes priorities dynamically
  - E.g to counter starvation: older processes have a higher priority
- This is the ***actual*** or real priority of the process
- However we want a value for a relative priority that is static and easy to work with
  - We allocate each process a “nice” value
  - Static priority that the priority bases priorities off to work out the *actual* priority



getpriority(2) and setpriority(2)

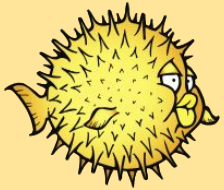
- int getpriority(int which, id_t who);
- int setpriority(int which, id_t who, int prio);
  - int which: PRIO_PROCESS, PRIO_PGRP, PRIO_USER
  - id_t who: The id of the object to change the priority of with respect to which
  - int prio: The new priority value (nice value) for setpriority()



Zone Priority Modifications

- Zone priority subcommands and syscalls
  - Zone Info Update for Getting Priority
  - Zone Priority Set System Call
  - Zone command extension
- Setpriority/getpriority expansion

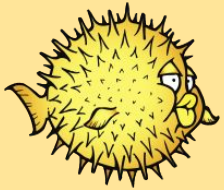
See spec for more details



Functionality - Priority Getting

- Modify the zones system tall to populate .zi_priority field of the zinfo structure
  - You'll have to modify a system calls
- What we are looking for
  - You can read kernel code and find where a system call is defined
  - You can update and modify a system call in the kernel
- See prac 2

See spec for more details



Functionality - Priority Setting

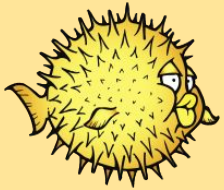
Implement a new syscall to set the priority in the kernel:
int zone_setpri(zoneid_t z, int prio);

- Set the priority of the zone in the zones subsystem
- Update the zone's subsystem so when a process gets ran in a zone, it's priority will be set to the priority of that zone

- What we are looking for
  - You can expand a kernel subsystem to add a new syscall
  - You can add a new syscall into the kernel
  - You are able to go through kernel code and understand how it works
  - You can work out how to appropriately update a kernel subsystem to set the priority of a task

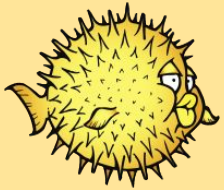
See prac 2 for syscall setup

See spec for more details



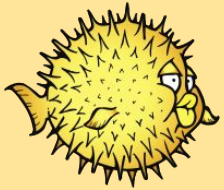
Functionality - priority subcommands and syscalls

- You will need to update the following syscalls in the kernel:
  - int getpriority(int which, id_t who);
  - int setpriority(int which, id_t who, int prio);
    - int which: PRIO_PROCESS, PRIO_PGRP, PRIO_USER, PRIO_ZONE
    - id_t who: The id of the object to change the priority of with respect to which
    - int prio: The new priority for setpriority()
  - If PRIO_ZONE is set for which, then it should deal with zone priorities.
  - This behaviour must be consistent with zone_setpri()
- What we are looking for
  - You can find where system calls exist in the kernel
  - You can make changes to existing kernel code to update functionality



Modularity, Robustness Portability and Reusability

- In COMP3301 we expect you to write robust structured, reusable code
- For this criteria essentially we are looking for
  - You are able to write things in the same structure of the already existing code
  - That you can handle errors in a robust way
  - That your code is free of race conditions
  - That your code clear and concise
    - For a kernel developer who hasn't seen the subsystem before

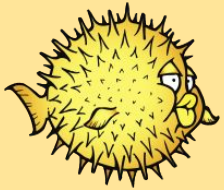


Reflection

Reflect on your implementation by answering the following questions:

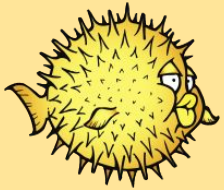
1. How did you locate the files which you have modified in your implementation? List the tools (e.g., cscope, grep) and steps.
2. What bugs have you encountered? Describe one bug. If you did not encounter any bugs, describe something which you believe was tricky.
3. How did you debug the bug which you 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,

See spec for more details



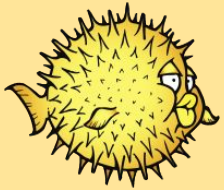
Submission – Git

- You have learned to use git in the pracs (if you didn't already know it)
  - If you haven't, make sure to go back over that prac and know how to branch, commit and push
- You must **push** to the a1 branch
  - We will tag this at submission on the remote
- Commit effectively, commit often
- Remember to push!
- Go through the git section of the pracs if you haven't already
- Check the online remote repository viewer to ensure your code is pushed
  - (And it's the right version)
- We cannot mark any code that was not pushed at submission time



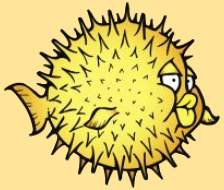
Submission – Late/Extensions

- Make a private post on Ed for late or extension submissions
- For late submissions you lose 10% per day for up to 7 days
- Ensure you add the correct commit hash for your submission and that it is pushed to remote
- There will be a template posted on ed closer to the deadline



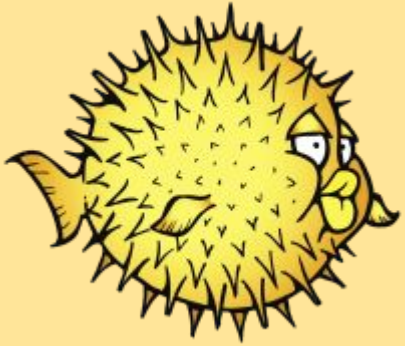
Where to start – tips

- Zone list update
  - low hanging fruit, all userland
- Zone info update for getting priority
  - Small modification of an already existing syscall
- Zone setpriority
  - Need to explore the kernel a bit more to understand
  - See getpriority(2) and setpriority(2)
  - Need to implement a new system call from scratch
- getpriority(2) and setpriority(2)
  - Seperate from zones
  - Should be similar to previous tasks but you need to find the correct location



Kernel debugging tips

- Kernel takes a long time to compile and reboot
  - The more you can get from each run, the better
  - You may use printf()
    - Can view output from the console
- When your kernel crashes it drops into ddb
  - Try and read these messages and work out where your kernel crashes
  - Gives you a function call stack like you would from an assertion in userland
  - Errors:
    - protection fault - you tried to access protected memory you shouldn't
    - Lock fault - you tried to enter a lock twice
    - uvm/page fault - closest thing to a “segfault” – pointer dereferences etc.
    - “Panic” - manual kernel panic caused by a panic() call
    - Assertion failed – kernel assert fail from a KASSERT() call



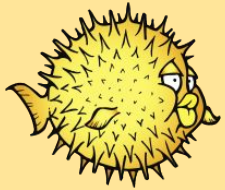
Happy devving!

Thanks for coming.

OpenBSD

COMP3301 - 2025

The University Of Queensland
AUSTRALIA



(Extra) The Original Zones from Solaris!