Kernel Development Learning Pipeline

📝 ACTC Assignment 2 : Build a "Mini" Container Runtime

This assignment will be formatted and submitted using a simulated Linux kernel mailing list patchset submission and review workflow.

We first present the details of the assignment itself, followed by submission instructions.

Before beginning work for this assignment, please obtain your credentials by entering your student ID on the registration page.

Then, please create our fedora container with the following invocation, setting `$username` and `$password` appropriately beforehand.

```
sh -c 'read -rp "username: " username && curl -u $username
https://winter2025-iit.actc.underground.software/Containerfile
| podman build -t kdlp_container -'
```

Create a podman volume for persistent storage.

```
podman volume create kdlp_volume
```

Get into the container

```
podman run -it --rm --hostname kdlp --name kdlp -v
kdlp_volume:/home kdlp_container
```

Clone the submission repository

```
git clone https://winter2025-
iit.actc.underground.software/cgit/submissions
cd submissions
```

Create a new branch that is a copy of the origin master branch to do your work for the ACTC2 assignment

```
git checkout --no-track -b ACTC2 origin/master
```

Create the `$USER/ACTC2` directory and subdirectory

```
mkdir -p $USER/ACTC2
cd $USER/ACTC2
```

Do the assignment and make your commits in this directory.

This is not mandatory, however this container will provide you with a Linux environment with the necessary packages, git config, and mutt config to easily complete this assignment, submit them to our mailing list, and review the submissions of your peers.

You can configure your own environment if you prefer but if you run into problems setting up your environment please use the container rather than ask the instructors for help setting up your environment.

About this website

The Activity Log records all email you send as a part of this assignment.

The Course Dashboard attempts to match the emails you send to each component of each assignment, that is to say, initial submission, peer review, and final submission.

The automated feedback and score section will provide feedback when it is ready.

The human feedback section will contain the remarks from the grader once your final grade is ready.

Due dates

Component	Date
Initial Submission Deadline	EOD Thursday, 25 December 2025
Peer Review Deadline	EOD Sunday, 28 December 2025
Final Submission Deadline	EOD Tuesday, 30 December 2025

All due dates are 23:59 IST on the day stated in the table above

1. Overview

In this assignment, you will demystify container technology (like **Docker** or **Podman**) by writing a simple container runtime in C from scratch. You will interact directly with the **Linux kernel's isolation primitives—Namespaces, Cgroups**, and **Seccomp**—to create a secure, isolated environment for a process.

By the end of this task, you will have a program that can run a shell inside a custom root filesystem, isolated from the host's process list, hostname, and protected by security policies.

---

2. Prerequisites

- **OS:** Linux (Virtual Machine recommended).
- **Permissions:** Root access (`sudo`) is required for namespace operations.
- **Packages:** You will need the development headers for capabilities and seccomp: `bash sudo dnf install gcc make libcap-ng-devel libseccomp-devel # sudo apt-get install gcc make libcap-ng-dev libseccomp-dev`
- **Knowledge:** Basic C, Linux system calls (`fork`, `exec`, `mount`), and CLI usage.
- **Resources: Root Filesystem (rootfs):** You can export one from Alpine Linux or Fedora Linux using podman: `bash [sudo] mkdir rootfs [sudo] podman export $(podman create alpine) | tar -C rootfs -xvf -`

---

3. Program Requirements

Your program must be named `simple_container` and accept the following arguments:

```
sudo ./simple_container <rootfs_path> <command> [args...]
```

- `<rootfs_path>`: The directory to become the new root (e.g., `./rootfs`).
- `<command>`: The binary to execute inside the container (e.g., `/bin/sh`).

Core Rules

- **Strict Error Handling:** Every system call (e.g., `mount`, `unshare`) must be checked. If it fails, print a descriptive error message and exit.
- **Comments:** Explain your code.

---

4. Implementation Steps

Phase 1: The Skeleton & Namespace Isolation

Create the basic process structure. Container runtimes typically use `clone` or `unshare` to create new namespaces.

- **Goal:** The child process should run in a separate **UTS (Hostname) namespace.**

Tasks:

0x00: Use `fork()` to create a child process.
0x01: The parent waits for the child to exit.

0x02: Child uses `unshare(CLONE_NEWUTS)` to detach its hostname from the host. Unshare other namespaces such as `CLONE_NEWNS`, `CLONE_NEWIPC` and `CLONE_NEWPID`

0x03: Child changes its hostname to `mycontainer`.

- **Verify:** Running `hostname` inside the container shows the new name, while the host remains unchanged.

Manual pages:

0x00: `man 2 fork`

0x01: `man 2 unshare`

0x02: `man 7 uts_namespaces`

0x03: `man 2 clone`

Phase 2: Filesystem Isolation (The Jail)

Trap the process inside the provided `rootfs` directory. **Do not use chroot.** You must use **pivot_root** for better security.

- **Goal:** The process sees `<rootfs_path>` as `/`. The old host filesystem is inaccessible.

Tasks:

0x00: **Private Mounts:** Mark the root mount (`/`) as `MS_PRIVATE` so mount events don't leak to the host.

0x01: **Bind Mount:** `pivot_root` requires the new root to be a mount point. Bind mount the `rootfs` path to itself.

0x02: **Pivot:** Use `syscall(SYS_pivot_root, ...)` to swap the root.

0x03: **Cleanup:** Unmount the old host root (detach it) and remove the temporary directory.

- **Verify:** Running `ls /` inside the container shows only the container filesystem.

Manual pages:

0x00: `man 2 pivot_root`

Phase 3: PID Isolation (Process Identity)

Isolate the process IDs so the container cannot see host processes.

- **Goal:** The command running inside the container should be **PID 1.** Running `ps` should show only container processes.
- **Concept:** `unshare(CLONE_NEWPID)` affects the children of the calling process, not the process itself.

Food for thought

There are two ways to make the child PID 1 * Unshare the namespace **before forking.** * The child forks for a second time.

Tasks (follow the first way):

0x00: **Refactor Unshare:** Add `unshare(CLONE_NEWPID)` to the **Parent** process (before the `fork`) and remove it from the Child's unshare.

0x01: **Mount /proc:** The `ps` command relies on the `/proc` filesystem. You must mount a fresh `proc` filesystem at `/proc` **after** pivoting root.

- **Verify:** Run `ps aux` inside the container. You should see very few processes, and your shell should be PID 1.

Manual pages:

0x00: `man 7 pid_namespaces`

Phase 4: Resource Isolation (Cgroups)

Prevent the container from consuming all system memory using **Cgroup v2.**

- **Goal:** Limit the container to **100MB** of RAM**.**

Tasks (in Parent Process):

0x00: **Create Group:** Create a directory `/sys/fs/cgroup/simple_container`**.**
0x01: **Set Limit:** Write `"100000000"` (100MB) to `memory.max` in that directory**.**
0x02: **Add Process:** Write the Child's PID to `cgroup.procs`**.**
0x03: **Cleanup:** After the child exits**,** remove the cgroup directory using `rmdir`**.**

Man pages

0x00: `man 7 cgroups`

Phase 5: Security – Capabilities

The root user inside a container shouldn't be as powerful as the real root**.**

- **Goal:** Drop dangerous capabilities
- **Library:** Use **libcap-ng.**

Tasks (in Child Process):

0x00: Create a **whitelist** of exactly these allowed capabilities: `CAP_KILL`, `CAP_SETGID`, `CAP_SETUID`, `CAP_NET_BIND_SERVICE`, `CAP_SYS_CHROOT`
0x01: Use `capng_clear` to wipe the slate clean**.**
0x02: Use `capng_update` to add your whitelist to **Effective, Permitted** and **Bounding** sets**.**
0x03: Apply the changes with `capng_apply`**.**

Man pages

0x00: `man 7 capabilities`
0x01: `man 3 capng_clear`
0x02: `man 3 capng_update`
0x03: `man 3 capng_apply`

Phase 6: Security – Seccomp (Syscall Filtering)

Restrict which system calls the container can make to the kernel**.**

- **Goal:** Prevent the container from calling `reboot()`, `swapon()`, or kernel module functions**.**
- **Library:** Use **libseccomp.**

Tasks (in Child Process):

0x00: Initialize a filter with `seccomp_init(SCMP_ACT_ALLOW)` (**Allow-list** by default)**.**
0x01: Add rules to block specific syscalls: `reboot`, `swapon`, `swapoff`, `init_module`, `finit_module` and `delete_module`**.** Use `SCMP_ACT_ERRNO(EPERM)` to return a permission error**.** Using the `SCMP_SYS()` macro is recommended**.**
0x02: Load the filter into the kernel using `seccomp_load`**.**

Man pages

0x00: `man 2 seccomp`

---

5. Helpful Snippets

`pivot_root` wrapper is not in the standard libc headers. You must define a wrapper:

```
static int pivot_root(const char *new_root, const char
*put_old) {
    return syscall(SYS_pivot_root, new_root, put_old);
}
```

Compiling: You must link against the security libraries:

```
gcc -o simple_container simple_container.c -lcap-ng -lseccomp
# pkg-config --libs libcap-ng  libseccomp # to see these flags
```

---

6. Submission

Submit patches-by-mail with a single C file named **simple_container.c**, a **README** and a **Makefile.**

- Patch 1 adds your makefile as to the file `<username>/ACTC2/Makefile`
- Patch 2 implements phase 1 by creating `<username>/ACTC2/simple_container.c`
- Patch 3 implements phase 2 by modifying `<username>/ACTC2/simple_continer.c`
- Patch 4 implements phase 3 by modifying `<username>/ACTC2/simple_continer.c`
- Patch 5 implements phase 4 by modifying `<username>/ACTC2/simple_continer.c`
- Patch 6 implements phase 5 by modifying `<username>/ACTC2/simple_continer.c`
- Patch 7 implements phase 6 by modifying `<username>/ACTC2/simple_continer.c`
- Don't forget a cover letter (Patch 0) containing what you would put in the README
- Submit your patches to `ACTC2@winter2025-iit.actc.underground.software`

Submitting this assignment

The assignment must be submitted in the form of an email patchset generated by `git format-patch` from commits made in your local copy of this repository that includes a cover letter describing your work. You will use `git send-email` to submit the assignment as described above.

As part of the peer review process, this assignment will require you to submit your patchset at least twice.

Start the assignment early. If you run into issues and get stuck it gives you time to ask questions and get help before the deadlines so you can submit something on time and get credit for the assignment. If you finish early, you can resubmit as many times as you'd like.

Any submission that violates these guidelines or fails to compile with no warnings or errors will receive a zero.

Students submit assignments and review peer submissions on this list.

Each assignment involves the following three stages:

Step 1: Initial Submission (due *23:59 IST on Thursday, 25 December 2025*)

- The student makes their submission to the mailing list using `git send-email`
- If the initial submission is late, the student will get a zero on the entire assignment
- The subject line of the initial submission patchset should be tagged as a "Request for Comments", otherwise known as RFC, as explained in the patchset guidelines section below.
- Each re-submission should increment the version number in the subject line, as explained in the resubmission guidelines below.

Step 2: Peer Review (due *23:59 IST on Sunday, 28 December 2025*)

- Each student is assigned two other students' work to review on their submission dashboard
- For each student to whom you are assigned, you will do the following:
- Make a new branch off of master/main and switch over to that branch to apply the patches.
- Apply the student's latest patchset submission to your local tree
- Make sure each patch applies cleanly IN ORDER (a.k.a no corrupt patches, whitespace errors, etc.)
- Make sure the code compiles without warnings or errors
- Sanity check that any programs run without immediately crashing
- Make sure that the output looks reasonable
- Scan through the rubric, as each assignment's peer review requirements will be different
- If there are any problems with the submission, report them in your review
- If you determine that there are no issues with the submission, inform the recipient.
- If it turns out that there were issues with the submission that you missed, points will be deducted from your overall assignment grade

After completing your review, reply to the reviewee's cover letter email for the reviewed patchset as described below.

- If the student approves of a submission, then the student will reply to the cover letter of the patchset with a single line containing the following:

```
Acked-by: $FIRSTNAME $LASTNAME <$USERNAME@winter2025-
iit.actc.underground.software>
```

- If the student finds issues with a submission, then the student will reply to the cover letter of the patchset with detailed feedback about their concerns and conclude the email with a single line containing the following:

```
Nacked-by: $FIRSTNAME $LASTNAME <$USERNAME@winter2025-
iit.actc.underground.software>
```

- In parallel, other students have been assigned the student's submission and the student should receive feedback from two other students
- These reviews are due 48 hours after the initial submission deadline
- A late or missing submission yields a zero for the review section of the assignment
- If a student does not complete peer review their maximum assignment grade is 80%
- Reviews are graded based on how many issues a student missed. The student receives 20% off for each unique issue not spotted with max penalty of 100%

Step 3: Final submission (due *23:59 IST on Tuesday, 30 December 2025*)

- The student, if canny, will act on the feedback from the received reviews
- Regardless of whether the student made changes to their initial submission, they must make a final submission
- A late or missing final submission will result in a zero grade for the assignment
- This is due 120 hours after the initial submission deadline, and 48 hours after the peer review deadline
- The overall assignment grade is composed of 80% for the final submission and 10% for each peer review
- While the initial submission is not explicitly graded, failure to submit anything or submissions devoid of any effort whatsoever will result in a zero

Patchset Guidelines

The specific assignment instructions above specify which files to edit, and how many commits you should be making which will inform the overall structure of your patchset, but every patchset must follow these general guidelines:

- Each commit gets its own patch with a title and body
- The patch series is introduced with one additional patch, the cover letter.

Fortunately, git format-patch can generate the appropriate files for you:

```
$ git format-patch -3 --cover-letter --rfc -v1
```

This command generates git email patches from a base repository. The arguments mean the following:

`-3` specifies that the three most recent commits should be included, and therefore 3 email patch files will be generated. Change the number as needed.

`--cover-letter` specifies that a cover letter email template file is generated as "patch 0" of the patchset. You must always use this.

Use `-v<n>`, in this case `-v1` to specify the version number of this patchset. Increase this number each time you resubmit an assignment. Use `--rfc` to denote on each patch that the changes are a draft posted for review. Use this when generating initial submission patchsets, but not when generating the final submission patchsets.

All of the patches must follow the patch guidelines below.

Commits, from which the patches are generated, should follow the commit guidelines below, and the cover letter must follow the cover letter guidelines below.

You will receive an automatic zero on the assignment if any of the patches in your patchset are corrupt. A patch is considered corrupt if it does not apply.

Always test to see if your generated patchset applies cleanly before submission. Generating corrupt patches shouldn't be possible if you use `git format-patch`, but if you edit the files manually they might get corrupted. You have been warned! The correct way to edit patches is to edit the underlying commits and then regenerate the patchset.

Note that version numbers are maintained between RFC and non-RFC patchsets--in the best case, your RFC would be v1 and your final submission would be v2. Your cover letter should include a summary of changes since the previous version. To edit previous commits, see `man git-rebase` and the `--amend` option from `man git-commit`.

Patch Guidelines

Assignments must be submitted in the format of git email text patches grouped into patchsets.

Binary patches are forbidden because all work on this assignment is in plaintext.

Every patch in the patch series (including the cover letter) must end with a "Signed-off-by" line, called the DCO (Developer Certificate of Origin). The line must exactly match this format:

```
Signed-off-by: $FIRSTNAME $LASTNAME <$USERNAME@winter2025-
iit.actc.underground.software>
```

The DCO line must be the final line of the email body right before the start of the patch diff, or in the case of the cover letter, before the start of the patchset summary and diffstat output.

Fortunately, you can make git add this line automatically for you when you author a commit:

```
$ git commit -s
```

This will add the DCO line in the commit message automatically from information in the current user's git config, or if present the repository's `.git/config` file.

You will need to remember to add your DCO to the cover letter manually.

Commit Guidelines

Within the submissions repository each student will create a directory matching their username. If any starter files are needed, the assignment descriptions will specify the necessary details. You will do the assignment and turn your work into commits using `git commit -s`.

When you author a commit the first line(s) you type into your editor will become the title, and by hitting enter twice and leaving a blank line the subsequent text will become the full commit message. The `git format-patch` utility will automatically put the title and message of a commit into the respective title and body of the corresponding generated email patch file.

The commit title should include the assignment name followed by a colon and a space, then a short summary of the changes in this commit in the imperative tense, for example, `setup: Add $USERNAME/setup.txt`. Any further details should be included in the commit message.

You should make sure that the changes you are including in your commits are tidy. This means that code should follow the kernel code style guidelines, e.g. tabs for indentation, tab width of 8, and no lines exceeding 100 columns.

You must also avoid whitespace errors. These include whitespace at the end of a line, lines with only whitespace on them, extra blank lines at the end of a file, and forgetting the newline on the last line of the file. A good editor will highlight and/or automatically fix these for you, but git will also detect these when formatting and applying patches.

You can check for the other whitespace errors by using `git am <email patch file>` to attempt to apply your patch to the local tree. If `git am` prints a warning like this when you apply the patch:

```
warning: 2 lines add whitespace errors.
```

You should adjust the indicated lines and fix your patch.

Your patches also must apply cleanly to the `HEAD` commit on the `master` branch of the upstream submissions repository. You can verify this by updating and checking out the `master` branch yourself and trying to apply your patches. We should NOT need to apply your previous versions of the patchset in order for the latest version of your patchset to apply. If this is the case, your patchset will be considered corrupt and you will receive a zero.

Sample workflow to check that your patchset applies cleanly:

0x00: Generate your patches and put them in a known location and take note of the filenames

0x01: Make sure your local git worktree is up to date

- You should do this each time you begin work within any git repository
- Use `git remote update` to update all of your local copies of remote trees

0x02: Create and checkout a local branch based on the upstream `origin/master` branch by using:

- `git checkout -b <branch name> origin/master` (branch name can be anything convenient)

0x03: Apply your patchset to this branch using `git am <patch1> <patch2> ... <patchN>`

0x04: If no errors appear: congratulations, your patchset applies cleanly!

- If there are whitespace errors or corrupt patches, revise as needed by rebasing and amending your commits

Cover Letter Guidelines

When creating a patchset, you will generate a cover letter template file.

When you open the cover letter file generated by `git format-patch` in your editor, you will see a summary of all the changes made in the subsequent patches at the bottom and two filler lines indicating where you can add the title (`*** SUBJECT HERE ***`) and message (`*** BLURB HERE ***`) to the email.

You must:

- Replace the subject filler text with the assignment name and your username separated by a dash `'-'`. For example: `Linux kernel - linus_torvalds`
- Replace the blurb filler text with your write-up

Failure to remove these filler lines including the asterisks will result in lost points.

The first line of your cover letter write up should state what you think your degree of success for the assignment was (from 0% to 100%), formatted as follows:

```
Completed: 100%
```

Following this, your cover letter write-up should include a short discussion that includes the following:

- An estimate of how much time you spent working on this assignment
- How you approached the assignment
- A detailed description of any problems that you were not able to resolve before submitting

Failure to cover each of the above will result in a zero grade on your assignment. If your self-assessment of success is not 100%, please document all known issues. Be honest with your self-assessment. If obvious errors are discovered despite high self-assessment, then we reserve the right to give you a zero. We also reserve the right to dock points for grammatical mistakes and spelling errors.

Make sure to add a Signed-off-by line to the cover letter. It goes right at the end before the automatically generated summary of the patchset. If you want to be sure you have it in the right place, you can add it first putting it in place of the `*** BLURB HERE***` filler text and then write the rest of your cover letter above it.

Resubmission Guidelines

You can resubmit as many times as you please before a deadline. If you resubmit, you must regenerate and resend all the patch files and cover letter as a new version of the entire patch series, incrementing the version number in the subject line appropriately as described above in the patchset guidelines.

You must also document what changed since the last submission in your write-up, include a section with a title like "changes since vN" where N is the number of your last submission and you explain what changed.

You can resubmit a peer review by making a new reply to the original cover letter with a new version of your complete review.

For any patchset or peer review, we we only grade the most recent submission.

```
msg = (silence)
whoami = Bardugo_Eleryan
singularity v0.8 https://github.com/underground-software/singularity
```