

Roberto Myftaraga

Senior project report 3

10/05/2025

Docker Based Container Engine: Process, Resource, and Filesystem Isolation

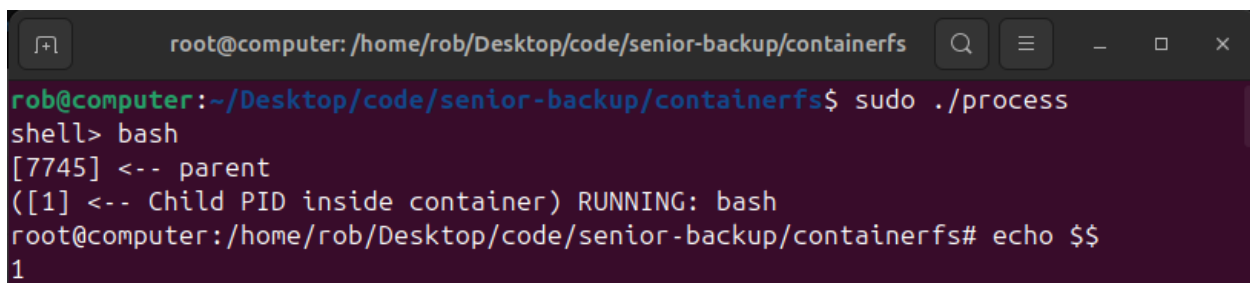
This week, I wanted to get hands-on with coding what I have learned so far. I felt that too much theory without practice would only confuse me, so I began developing a prototype to connect all the concepts I've been studying. My goal was to create a simple command-line shell that could simulate how a container runs isolated processes.

I used the `fork()` and `execvp()` system calls to start new processes, and combined them with `unshare(CLONE_NEWPID)` to create a new PID namespace. This allowed me to run commands such as `ls` or even start an interactive **bash** session inside an isolated process tree. Each time a command runs, the containerized process gets its own PID space and behaves as if it's running independently.

- From my understanding so far, the **unshare()** system call is the key to making this system of containers possible. It provides the ability to isolate specific parts of the system — and we can add as many namespaces as needed to increase isolation (e.g., UTS, mount, or network namespaces).

Through this, I began understanding how a container runtime like Docker manages its process isolation at the kernel level. Although this first version still shares the host filesystem and network, it lays the foundation for further isolation. My next goal is to add filesystem separation using **chroot()** and a UTS namespace for unique hostnames. Overall, this week's progress felt like a major step from pure learning to building something tangible.

From the prototype we can see that we really are running an isolated process :

A terminal window with a dark background. The title bar shows 'root@computer: /home/rob/Desktop/code/senior-backup/containerfs'. The prompt is 'rob@computer:~/Desktop/code/senior-backup/containerfs\$'. The user enters 'sudo ./process'. The prompt changes to 'shell>'. The user enters 'bash'. The prompt changes to '[7745] <-- parent'. The user enters '([1] <-- Child PID inside container) RUNNING: bash'. The prompt changes to 'root@computer:~/Desktop/code/senior-backup/containerfs#'. The user enters 'echo \$\$'. The output is '1'.

```
root@computer: /home/rob/Desktop/code/senior-backup/containerfs
rob@computer:~/Desktop/code/senior-backup/containerfs$ sudo ./process
shell> bash
[7745] <-- parent
([1] <-- Child PID inside container) RUNNING: bash
root@computer:~/Desktop/code/senior-backup/containerfs# echo $$
1
```

The 1 inside bash signifies the PID.

Backup files :

<https://github.com/InspectRM/senior-backup>