The assignment as a whole is about understanding containers and how to create one of them using basic building blocks.

Phase-1 attempts to provide the students with hands on experience in creating containers and experimenting with some kernel features ***(such as namespaces & cgroups)***.

We make use of the shell command **unshare** *(which is a wrapper around the system-call unshare)* to isolated different parts of its execution context from other processes.

The handout for Phase-1 lists a set of steps that experiments this. With an unshared and isolated execution context you can also restrict the resource usage (eg: CPU, memory) of the isolation unit (container). This is experimented at the latter stages of this phase using **cgroups**.

The oral interview will evaluate your understanding based on the steps from this phase. The following is an overview of the TODOs with respect to this phase.

1. Experiment each step mentioned in the handout and get a clear understanding as to what is happening. (**ex**: what exactly is unshare doing?)
2. After mounting the **proc** subsystem (as shown below) into the unshared-container, see the difference between the proc folder inside and outside and see the difference.

```
sudo unshare -fp --mount-proc=/proc /bin/bash
```

3. Why is it necessary to mount **"proc"** to be able to get commands like **top** and **ps** working inside the container?
4. How did you validate that the container was in a new PID namespace? Run two containers in two separate namespaces. Run some commands in the background in one namespace.
    a. **ping 8.8.8.8 </dev/null &>/dev/null &** *(3 times to run three instances of **ping**)*

    Now get the PIDs of these 3 ping processes. Try killing them inside the other container using: **kill -9 PID**

    Do you see the processes from the host?

5. Create a new use in your host

    sudo adduser newt

    Change user to this new_user and run an unshared shell with isolated USER namespace.Now set map the uid of this new-user to some random value inside the namespace.

6. Run two separate containers in different UTS namespace. Set their hostnames to different things and see if changing one affects the other.
7. Without **chroot**, can you traverse anywhere within the host filesystem?
8. Now run two containers and **chroot** into the same root-filesystem. Create/Delete files in container and see if they are visible in the other.
9. Now create two **copies** of root-filesystems. Run two containers with **chroot'ed** to different roots. See if you can traverse to the filesystem of the other container.
10. Create a container, try running a memory hogging program and view its memory usage using **htop**.
11. Now set a memory-control on this container and run a similar program and see what happens.
12. Create two separate containers and make them share one CPU on the ratio of 7:3.
13. Restrict the read bytes per second for a container to 10Mbps using a blkio controller
14. Restrict the write bytes per second for a container to 5Mbps using a blkio controller
15. Set a pid-controller to your container so that it can only create 50 processes at max