CS695: Assignment 3
Spring 2023-24
ready, get set, containers!

Statutory note:

- This course is on a **no-plagiarism** diet.
- All parties involved in plagiarism harakiri will be penalized to the maximum extent.
- The Moss Detective Agency has agreed to conduct all investigations. https://theory.stanford.edu/~aiken/moss/ (https://theory.stanford.edu/~aiken/moss/)
- Byomkesh, Sherlock, Phryne, Marple, and Hercule are on standby.
- Hardcoding the runtime output in the code will be heavily penalized.
- Generative AI (ChatGPT, Gemini, etc.) is your friend, but it cannot generate outputs for you to submit.
- **Warning**: Submission Guidelines should be strictly followed; otherwise, your submission will not count. (0 Marks)

This assignment introduces the basic primitives, tools and procedures for setting up containers.

Environment Setup

This assignment has elements depending on the underlying OS. To facilitate efficient grading complete this assigment on the following Virtual Machines *(These are the same Virtual Machines used for Assignment 1)*.

x86_64 (Linux/Windows/Intel Mac) Users

Virtualization Tool: VirtualBox https://www.virtualbox.org/wiki/Downloads (https://www.virtualbox.org/wiki/Downloads)
Click here to download VirtualBox VM Image (https://www.cse.iitb.ac.in/~puru/courses/spring2023-24/assignments/a1/cs695_amd64.ova)

Apple Silicon Users

Virtualization Tool: UTM https://mac.getutm.app/ (https://mac.getutm.app/)
Click here to download UTM VM Image (https://www.cse.iitb.ac.in/~puru/courses/spring2023-24/assignments/a1/cs695_arm64.utm.zip)

Virtual Machine Credentials

Username:	cs695
Password:	1234
Root Password:	1234

Updated assignment code tarball: link (https://drive.google.com/file/d/1_nOP6i-qeOY598PdSz1Mb3BknHGlbtbd/view?usp=sharing)

- Don't change the directory structure in your submission.
- Replace <rollnumber> in the top level directory name with your roll number in small case.
- **Important**: This tarball has an updated git structure required for grading. Copy your solution for Task 1 and Task 2 within this updated tarball.

Task 1: namespaces with system calls

In this task we will explore various available system calls to play with namespaces of a process. The following article will help we get started with the different system calls available: https://lwn.net/Articles/531381/ (https://lwn.net/Articles/531381/)

Following are the logical actions –

1. Create a new child process in a new UTS and PID namespace
2. Create a second child process and attach it to the UTS and PID namespace of the first child process

Note: System calls of interest are clone, setns and unshare.

For the PID namespace, the namespace is determined during creation and cannot be changed. To create a newprocess with a new PID namespace has a flag to be used with the clone system call. To create a new child process and attach to an existing namespace, the parent process has to associate the flag with an existing process namespace. The clone system call does not have a flag to create and attach a new child process to an existing namespace.
Make changes to/via the parent process such that child2 is created in the same PID namespace as child1.

```
man 7 pid_namespaces
man 2 setns
man 2 clone
man 2 pidfd_open
man 2 unshare
```

Write code at marked portions in the file `task1/namespace_prog.c`

Don't make any changes in the file outside of marked portions.

The output of the program should be as follows:
(Parent hostname can be anything depending on your system)

```
---------------------------------------
Parent Process PID: 43156
Parent Hostname: cs695
---------------------------------------
Child1 Process PID: 1
Child1 Hostname: Child1Hostname
---------------------------------------
Parent Process PID: 43156
Parent Hostname: cs695
---------------------------------------
Child2 Process PID: 2
Child2 Hostname: Child1Hostname
---------------------------------------
Parent Process PID: 43156
Parent Hostname: cs695
---------------------------------------
```

Task 2: CLI containers

In this task we will create a simplified version of a container using command line tools. Here are some references to get started –
chroot (https://www.howtogeek.com/441534/how-to-use-the-chroot-command-on-linux/)
unshare (https://man7.org/linux/man-pages/man1/unshare.1.html)
cgroup v2 (https://www.kernel.org/doc/html/v4.18/admin-guide/cgroup-v2.html)

The task is divided in 3 subtasks, each contributing new features to the container.
The provide (single) bash script `task2/simple_container.sh` , needs to be used and completed for functionality of all the 3 subtasks.

Don't make any changes in `simple_container.sh` outside of marked portions

Subtask 2a

The first basic requirement of a container is that an user within a container should not be able to view/modify files outside the container's scope. To achieve this filesystem isolation, generally a directory within a host system is assigned as the root directory of the container. The root is the starting location for all paths in a filesystem so a container cannot explore filesystem beyond its assigned root.

Run the program `container_prog` by assigning the provided directory `container_root` as its root directory.

Note that many programs are dependent on dynamic libraries for its execution. These programs when loaded, open their dependency library files at some predefined paths. For correct exection all dependency libraries for `container_prog` should be available at correct paths within the chrooted environment. **All required dependencies should be copied/linked from within the script. The script should execute correctly given an empty container_root directory.**

Read comments and fill in commands at marked portions in `simple-container.sh` script.

Subtask 2b

Based on the output of *subtask 2a*, the container has the same PID namespace as the host. This is an isolation violation as a user from within a container can access/kill process in the host. In this subtask attach a new PID and UTS namespace to the container. Within the new PID namespace, the pid of processes launched within the container should start with 1.

Note that this is an incremental change and the container running within the same chrooted environment as in *subtask 2a*

Read comments and fill in commands at marked portions in `simple-container.sh` script.

Subtask 2c

The third requirement of the container setup is resource-provisioning. A container should not be able to hog unrestricted resources of the system. cgroup allows fine granularity control of different resources for each process group. In this subtask, create a new cgroup which restricts the maximum CPU utilization to 50%. This cgroup should be assigned to all processes running within the container.

Note:

- CPU utilzation within a cgroup is specified in terms of quota and period. Quota is the maximum total time on CPU cores the cgroup gets in the specified period of time. The quota can be greater than period on a multicore system. You can refer to this article (https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/resource_management_guide/sec-cpu) for more details. In this subtask, you are required to set the utilization limit to **50% of single core** and **not** 50% utilization limit of all cores.
- There are significant changes between cgroup v1 and cgroup v2. You will be working with cgroup v2 in this assignment.

Read comments and fill in commands at marked portions in `simple-container.sh` script.

Sample output on running `simple-container.sh` :

Output Subtask 2a

```
Process PID: 1693
Child Process PID: 1694
– – – – – – – – – – – – – –

Files/Directories in root directory:
lib64
container_prog
.
..
lib
– – – – – – – – – – – – – –
```

Output Subtask 2b

```
Process PID: 1
Child Process PID: 2
– – – – – – – – – – – – – –

Files/Directories in root directory:
lib64
container_prog
.
..
lib
– – – – – – – – – – – – – –

Hostname within container: new_hostname

Computation Benchmark:
Time Taken: 1440778 ms
Value (Ignore): 107375219085276240

Hostname in the host: cs695
```

Output Subtask 2c

```
Process PID: 1
Child Process PID: 2
– – – – – – – – – – – – – –

Files/Directories in root directory:
lib64
container_prog
.
..
lib
– – – – – – – – – – – – – –

Computation Benchmark:
```

```
Time Taken: 2866732 ms
Value (Ignore): 107375219085276240
```

Task 3: containers in the wild

In this task, we will develop a more complete container management tool like Docker. The tool will provide the following features:

- Ability to create full fledged debian container images
- Instantiating new containers from an image
- Enabling following network capabilities for the containers
  - Network path between container and host
  - Network path between container and public
  - TCP Port forwarding between host and container
  - Network path between different containers on the host

There are two script files for this task `config.sh` and `conductor.sh`

- `config.sh` is used to set configuration parameters for our tool
- `conductor.sh` script is the actual tool that will perform all operaions

> **Important**: Within the `config.sh` script set the `DEFAULT_IFC` variable to the name of network interface connecting your VM to the external network. Without this step you will not be able to access external network from your containers. You can use `ip a` command to list all network interfaces.

Functions provided by the tool are documented as follows:

> `sudo ./conductor.sh -h` can also be used to list out the functions of the tool

build

Usage: `./conductor.sh build <image-name>`
Operation: This function will download and create a debian system – the root directory structure of a debian system and all its dub-directories in a local directory, using the debootstrap command. This local directory is similar to the setup using a container image (and is referred as an image in this assignment). The command will store the container image within the configured images directory as `image-name`

images

Usage: `./conductor.sh images`
Operation: This function will list all container images available in the configured images directory.

rmi

Usage: `./conductor.sh rmi <image-name>`
Operation: This function will delete the given image from the configured images directory.

run

Usage: `./conductor.sh <image-name> <container-name> -- [command <args>]`
Operation: This function will start a new container with name as `container-name` from the specified image. `command <args>` is the program and it's arguments (if any) that will be executed within the container as the first process. If no command is given, it will execute `/bin/bash` by default. The container will have isolated UTS, PID, NET, MOUNT and IPC namespaces. It will also mount the following filesystems within the container:

- `procfs` for tools dependent on it like ps, top etc. to work correctly
- `sysfs` to be able to setup ip forwarding using iproute2 tool
- `/dev` (in host) bind mount to `container-rootfs/dev` to enable the container to see network devices

The first process will have `pid = 1` in the container. It will also set correct file permission on the `container-rootfs/` so that tools like apt work correctly. A directory named `container-name` within the configured `CONTAINERDIR` directory will be created. Furthermore `container-name` will have a subdirectory named `rootfs` which will be mounted as root directory for the container.

Note: If you run a container and exit from it, you cannot run the same container again. First you will need to delete the container using `stop` command, then run it.

ps

Usage: `./conductor.sh ps`
Operation: This function will show all running containers by querying entries within the configured `CONTAINERDIR` directory.

stop

Usage: `./conductor.sh stop <container-name>`
Operation: This function will stop a running container with given name. Stopping a container involves:

- Killing the unshare process that started the container
- Killing all processes running within the container
- Unmounting any remaining mount point within the contaier rootfs.
- Deleting `container-name` with configured `CONTAINERDIR` directory.

Note: Stopping a container will delete all state of the container

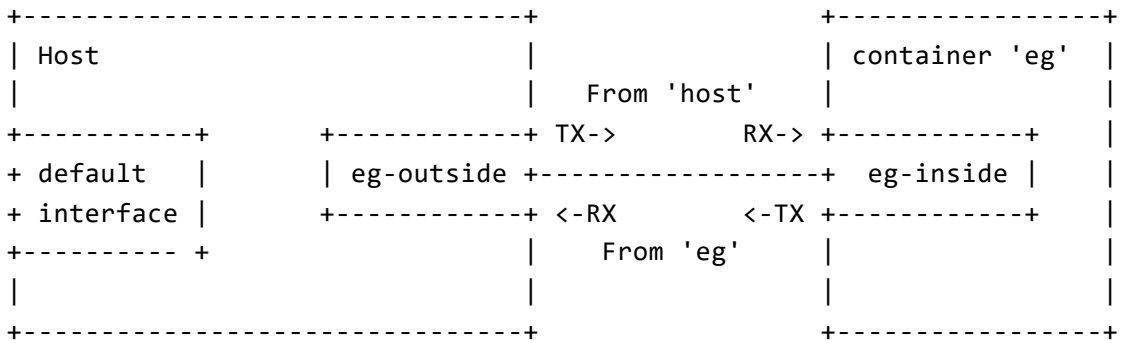
exec

Usage: `./conductor.sh exec <container-name> -- [command <args>]`
Operation: This function executes the given program along with it arguments within the specified running container. If no command is provided it will execute `/bin/bash` by default. The executed program will be in the same UTS, PID, NET, MOUNT and IPC namespace as the specified container. Furthermore it will see the root directory as `conatiner-name/rootfs` . It will also see the same `procfs` , `sysfs` and `/dev` filesystem as configured within the container and tools like ps, top etc. should work correctly.

addnetwork

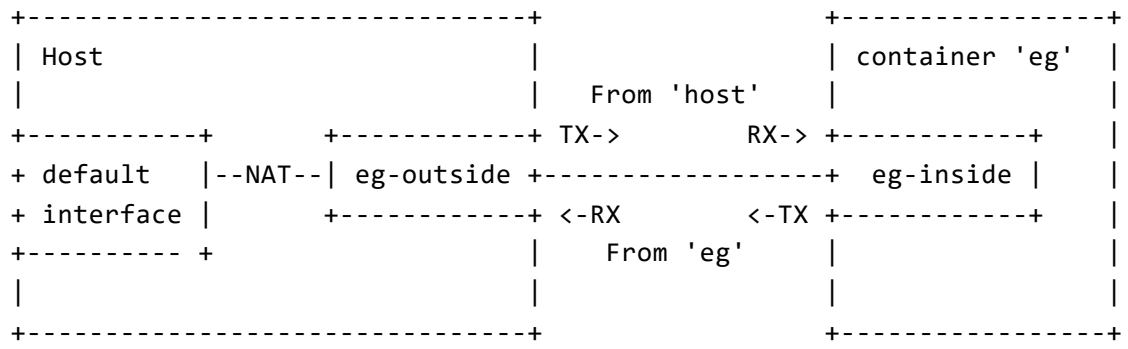
Usage: `./conductor.sh addnetwork <container-name> [options]`
Operation: This function will add a network interface to a container and setup its configurations so that the container can communicate using the network.

By default if no options is given it will setup the container network as shown in the diagram below:



It should be noted that only communication between `eg-outside` and `eg-inside` is possible. That implies that application running inside container `eg` will not be able to access the Internet (end points beyonde-outside.)

If the option `-i` or `--internet` is specified the script should allow for the applications running inside the container to access the Internet. The schematic diagram for the same is shown below:



Although the `-i` options allows Internet usage, exposing services deployed inside containers is still not possible (Basically, if you deploy a server or anything inside the container it will not be accessible outside the host).

You can use the `--expose` or `-e` option to make a port available to services outside of the container. This creates a rule in the host, mapping a container port to a port on the host to the outside world. Here are some examples:

`./sudo conductor.sh addnetwork -e 8080-80` : Map port 80 on the host to TCP port 8080 in the container.

peer

Usage: `./conductor.sh peer <container1-name> <container2-name>`
Operation: By default our conductor isolates the container so that no inter-container communication is possible. This function allows two container to communicate with each other.

The skeleton code implements most of the functionalities of the tool. For Task 3 you need to do the following:

- Understand the working of the tool. To get started the skeleton code has portions marked as **Lesson** along with some comment.
- Implement some parts of the tool which are missing. The required implementations are marked as **Subtask** in the skeleton code.

Initial Setup

You will be required to install debootstrap and iptables tool for the script to work. You can install them with

```
sudo apt install debootstrap iptables
```

Subtask 3.a: Implement run

Implement run to use unshare and chroot to run a container from given image. You also need to mount appropriate filesystems to the rootfs within the container to enable tools tools that utilize those filesystems e.g. ps, top, ifconfig etc. to be confined within the container isolation.

Considering you already have an image ready for usage, the sample outputs of the tool if you have set up everything properly is shown below:

```
cs695@cs695:~/task3$ sudo ./conductor.sh images
CS695 Conductor that manages containers
Name                  Size       Date
mydebian              307M       2024-03-20
cs695@cs695:~/task3$ sudo ./conductor.sh ps
CS695 Conductor that manages containers
No containers found
cs695@cs695:~/task3$ sudo ./conductor.sh run mydebian eg
CS695 Conductor that manages containers
root@cs695:/# ps
    PID TTY          TIME CMD
      1 ?        00:00:00 bash
      3 ?        00:00:00 ps
root@cs695:/# exit
exit
cs695@cs695:~/task3$ sudo ./conductor.sh stop eg
CS695 Conductor that manages containers
eg succesfully removed
```

Subtask 3.b: Implement exec

You need to complete the implementation of exec which can execute a command to join the existing namespace {all namespaces: uts, pid, net, mount, ipc} of the running container and execute the given command and args. The executed process should be within correct namespace and root directory as of the container and tools like ps, top should show only processes running within the container.

The following output shows the correct execution of the tool if the implementation is done properly and there exists a container `eg` which is running.

```
cs695@cs695:~/task3$ sudo ./conductor.sh ps
CS695 Conductor that manages containers
Name                 Date
eg                   2024-03-21
cs695@cs695:~/task3$ sudo ./conductor.sh exec eg /bin/bash
CS695 Conductor that manages containers
Executing exec in eg container!
root@cs695:/# ps aef
    PID TTY      STAT   TIME COMMAND
      4 ?        S      0:00 /bin/bash
      5 ?        R+     0:00  \_ ps aef
      1 ?        S      0:00 /bin/bash
      3 ?        S+     0:00  \_ sleep inf
```

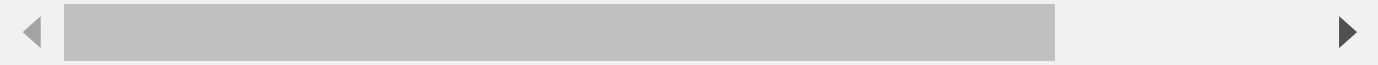
Subtask 3.c: Implement networking

Although most of the networking configuration is already being done in the tool. But one of the most important task i.e adding the interface that does the communication is missing from the implemenation. For the subtask you need to add a veth link (It is a peer link connecting two points) connecting the container's network namespace to the root(host) namespace. The veth link will have two interfaces. Moreover the interfaces should be enable for its proper usage.

If everything is done properly the output of the container and the host should be as follows:

- Container side:

```
cs695@cs695:~/task3$ sudo ./conductor.sh run mydebian eg
[sudo] password for cs695:
CS695 Conductor that manages containers
root@cs695:/# ip a
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
root@cs695:/# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eg-inside@if5: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP gr
    link/ether ca:09:b2:0c:be:ee brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 192.168.1.2/24 scope global eg-inside
       valid_lft forever preferred_lft forever
    inet6 fe80::c809:b2ff:fe0c:beee/64 scope link
       valid_lft forever preferred_lft forever
root@cs695:/# ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=0.151 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=64 time=0.085 ms
^C
--- 192.168.1.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1015ms
rtt min/avg/max/mdev = 0.085/0.118/0.151/0.033 ms
```



- Host side:

```
cs695@cs695:~/task3$ sudo ./conductor.sh addnetwork eg
[sudo] password for cs695:
CS695 Conductor that manages containers
Setting up network 'eg' with peer ip 192.168.1.2. Waiting for interface configuration

cs695@cs695:~/task3$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group de
    link/ether 52:54:00:a3:b7:09 brd ff:ff:ff:ff:ff:ff
    inet <hidden-for-safety> brd <hidden-for-safety> scope global dynamic enp1s0
       valid_lft 8167sec preferred_lft 8167sec
    inet6 <hidden-for-safety> scope link
       valid_lft forever preferred_lft forever
5: eg-outside@if2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP g
    link/ether d2:69:7d:88:da:38 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 192.168.1.1/24 scope global eg-outside
       valid_lft forever preferred_lft forever
    inet6 fe80::d069:7dff:fe88:da38/64 scope link
       valid_lft forever preferred_lft forever
```

Task 4: creating the matrix

The main motivation of the previous task was to understand how tools like Docker, podman, lxc, etc. work internally with the help of Linux tools amd primitives. But why should we bother about containers?

This task is meant to answer the question (partially, if not completely). The task requires us to deploy two services namely the **external-service** and the **counter-service**, in two separate containers. Only the external-service is accessible from the outside world (other hosts on the network), whereas the counter-service should only be accessible from the host. We will be using the conductor tool that implemented in Task 3.

Follow the following steps to implement task 4:

1. Build image for the container.
2. Run two containers say c1 and c2 which should run in background. You can add some sleep in the script so that c1 and c2 are running before the next commands are executed.

> Tip: To keep the container running in background use a first program that will not interact with the terminal and will not exit. e.g. sleep infinity, tail -f /dev/null

3. Copy directory external-service to c1 and counter-service to c2 at appropriate location. We can add these directories in the containers by copying them within `.containers/{c1,c2}/rootfs/` directory
4. Configure network such that:
   4.a. c1 is connected to the internet and c1 has its port 8080 forwarded to port 3000 of the host
   4.b: c2 is connected to the internet and does not have any port exposed
   4.c: peer network is setup between c1 and c2
5. Get IP address of c2. You should use script to get the IP address – can use ip interface configuration within the host to get IP address of c2 or can exec any command within c2 to get it's IP address
6. Within c2 launch the counter service using `exec c2 -- bash [path to counter-service directory within c2]/run.sh`
7. Within c1 launch the external service using `exec c1 -- bash [path to external-service directory within c1]/run.sh "http://<ipaddr_c2>:8080/"`
8. Within the host system open/curl the url: `http://<host-ip>:3000` to verify output of the service. **Note**: `http://localhost:3000` from within the host will not work.
9. On any system which can ping the host system open/curl the url: `http://<host-ip>:3000` to verify output of the service

Frequently Asked Questions

Q. I am unable to do apt update or apt install within the VM. What to do?
A: Make sure you are connected to the Internet. If you are inside IITB campus you can use the following bash script (https://github.com/rickydebojeet/iitb-internet-login) to login to IITB internet using the terminal. This is not required for the containers as they are using NAT and thus using VM's IP to connect to the internet.

Q. Something went wrong. I want to make the setup clean as start.
A: Perform stop operation on all container and remove the image using rmi to make a clean start.

Q. Why I am unable to curl http://<host-ip>:3000?
A: Make sure you firewall is not blocking the port. Update firewall rules or disable the firewall entirely (Not recommended). Also, if your VM network is configured to use NAT, then you will need to forward port 3000 of your VM to the host.

Submission details

To create the submission tar you need to use the following bash script: Link (https://drive.google.com/file/d/1urVP7_17Ui-UCUE2p6NjboxEOG0Rsh4q/view?usp=sharing)
You should execute the script within the same parent directory which contains your solution directory `<rollnumber_assignment3>`

Following is a demo to create the submission tar:

```
cs695@cs695:~/demo$ ls -l
total 12
drwxr-xr-x 7 cs695 cs695 4096 Mar 22 01:16 22m0789_assignment3
-rwxr-xr-x 1 cs695 cs695 4213 Mar 22 01:24 a3_submission_script.sh
cs695@cs695:~/demo$
cs695@cs695:~/demo$ ./a3_submission_script.sh
This script should be executed just outside your submission directory
Enter your roll number in lowercase: 22m0789

Submission directory created
Submissions for the following tasks were found and added
Task 1
Task 2
Task 3
Task 4

Verify the submission directory yourself once
Only required scripts and files are added to the submission directory
that needed to be modified.
Please make sure you are not adding any unnecessary files

Submission tarball created: 22m0789_assignment3.tar.gz
Please verify the contents of the tarball before submission
```

- **You need to submit the generated tar file through moodle**
- **Verify** the tar file before submitting.

Submission Deadline: 1st April 2024, 11.59 pm via Moodle