Pre-requisite For the Lab Session:

- Familiarity with Golang. If you are not familiar with Golang, you can easily understand the Golang code if you know C. As we are not going to make any major changes to the code during this lab session, it is okay if you do not have familiarity with Golang. If you are interested in learning golang, you can follow this tutorial -- https://tour.golang.org/basics/1 though it is not necessary for this lab session.
- Shell commands
    - ps -- http://man7.org/linux/man-pages/man1/ps.1.html or $ man ps
    - ip -- http://man7.org/linux/man-pages/man7/ip.7.html or $ man ip
    - pstree -- http://man7.org/linux/man-pages/man1/pstree.1.html or $ man pstree
    - grep -- http://man7.org/linux/man-pages/man1/grep.1.html or $ man grep
- All required files for the lab session is hosted at: https://github.com/cendhu/container-namespaces
- Clone the above repo:
  $ cd ~/
  $ git clone https://github.com/cendhu/container-namespaces
  $ mv container-namespaces lab

We have 10 tasks in total

Task 1: Find out the first process created by the OS. Is it init or systemd?

- *pstree* command would list down the complete parent-child tree. The root of the tree is the first process created by the OS.

Task 2: Find out all ancestors of your terminal process?

- *pstree* command can be used to find all ancestors of a process. In the *pstree* output, search for the word terminal and then you can easily identify all the ancestors of the terminal process.

Task 3: Print all child of your terminal process

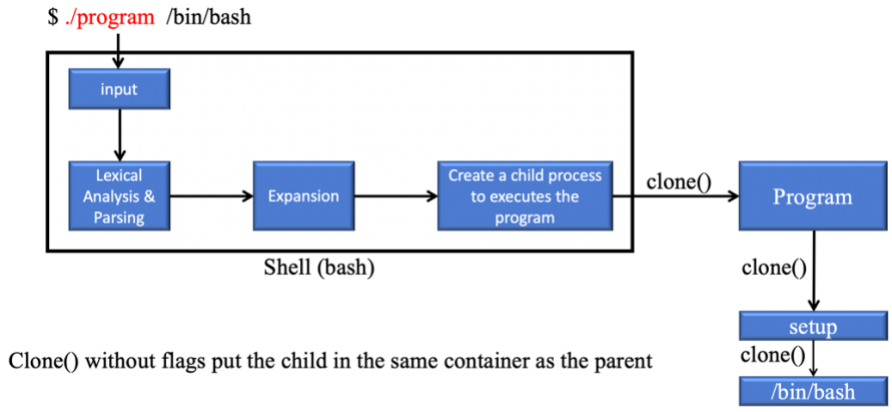
- find the *pid* of your terminal process using *ps* command or *pstree* (you can use either *pstree* --help or $ *man ps* to find out how or use *pidof* command)
- using *pstree* and *pid* of your terminal process, print all child of your terminal process (hint: *pstree --help*)

Task 4: Verify whether your OS have initiated the default global namespaces

- for any two process in the system, find the *pid* (hint: $ *ps aux*)
- using the *pid*, access */proc* filesystem to find out the namespaces of the process. For e.g., if a *pid* of a process is 100, *ls -la /proc/100/ns* command would print the namespaces on which the process is hosted.
- verify that by default all processes on the system is running inside the default global namespaces.

Task 5: Create a child process in the default namespaces

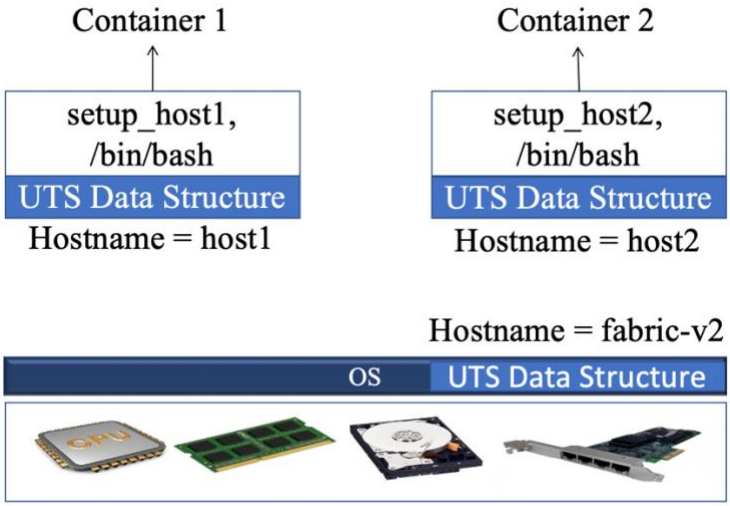
- In this task, we are the executing the following scenario:



- In the folder ~/*lab/program*, we have two golang files
  - program.go
  - setup.go.
- As it is a simple program, you can understand them without adequate familiarity with the golang.
- *program.go* is creating a child process to execute *setup.go*. Here, the parent process name is *program* and the child process name is *setup*.
- *setup.go* executes the command passed by you.
- To compile the code, run the following two commands (under the folder ~/lab/program)
  - $ go build program.go
  - $ go build setup.go
- To run the program, run the following command (under lab/program)
  - $ sudo ./program /bin/bash
- Note that the */bin/bash* is the command passed by you and the child process would execute the command. You can pass any command instead of */bin/bash*.
- As a result of the above execution, you would be inside a child process (if you pass command other than */bin/bash*, the child would execute the command and exit immediately. Hence, it is better to pass */bin/bash* to have an interactive child). Note that we haven't created any new namespaces for the child process.
- Use *pstree* command to ensure that the newly created child process called *setup* does exist.
- Find the *pid* of the newly created child process (called *setup*) and find out whether it is started inside the default global namespaces (need to use */proc/* filesystem as done in Task 4).
- To exit from the child, run $ *exit* or press control + d. Use *pstree* to ensure both the parent (*program*) and the child (*setup*) are not running anymore.
- Instead of */bin/bash* in *sudo ./program /bin/bash*, pass some other commands such as *sleep* or *ls* to see what happens.

Task 6: Create a child process in a new UTS namespace. For other 6 namespaces, it would use the default one.

- In this task, we are going to create two containers as shown below:

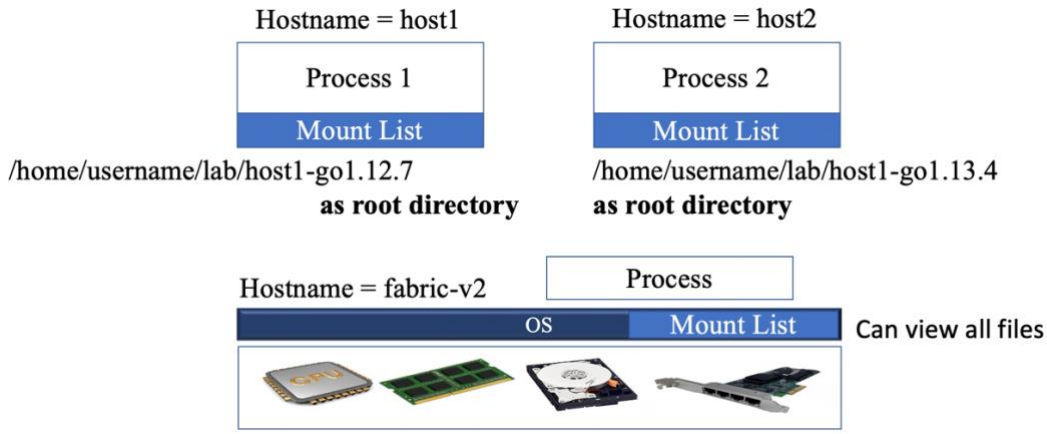


- In the folder ~/*lab/new_uts*, we have four golang files
  - new_uts_host1.go
  - setup_host1.go
  - new_uts_host2.go
  - setup_host2.go.
- Refer to slide 14 to understand what *new_uts_host1.go* and *setup_host1.go* do. The *new_uts_host1.go* creates a new child process called *setup_host1* and put it under a new UTS namespace. Then, the child process changes the hostname (refer to *setup_host1.go*). Similarly, we do the same for host2 as shown in slide 15.
- *new_uts_host1.go* is creating a child process to execute *setup_host1.go*. Here, the parent process name is *new_uts_host1* and the child process name is *setup_host1*.
- *new_uts_host2.go* is creating a child process to execute *setup_host2.go*. Here, the parent process name is *new_uts_host2* and the child process name is *setup_host2*.
- *setup_host1.go* and *setup_host2.go* executes the command passed by you.
- To compile the code, run the following four commands (under the folder lab/new_uts)
  - $ go build new_uts_host1.go
  - $ go build setup_host1.go
  - $ go build new_uts_host2.go
  - $ go build setup_host2.go
- After the compilation, to execute the program, start three terminals.
- On terminal 1, to run the new_uts_host1, run the following command (under the folder ~/lab/new_uts)

    - $ sudo ./new_uts_host1 /bin/bash
- As a result of the above executions, you would be inside a child process. Note that the child process called *setup_host1* is running inside a container now with a new UTS namespace. Check the hostname seen by the child process by executing the command
    - $ hostname
- On terminal 3, run $ hostname to find out the name on the host machine. This would be different than the one printed on terminal 1.
- Move back to terminal 1.
- Use *pstree* command to ensure that the newly created child process called *setup_host1* does exist.
- Find the *pid* of the newly created child process (called *setup_host1*) and find out whether it is started inside the new UTS namespaces (need to use */proc/* filesystem as done in Task 4).
- On terminal 2, to run the new_uts_host2, run the following command (under the folder lab/new_uts)
    - $ sudo ./new_uts_host2 /bin/bash
- As a result of the above executions, you would be inside a child process. Note that the child process called *setup_host2* is running inside a container now with a new UTS namespace. Check the hostname seen the child process by executing the command
    - $ hostname
- On terminal 3, run $ hostname to find out the name on the host machine. This would be different than the one printed in terminal 2.
- Move back to terminal 1.
- Find the *pid* of the newly created child process (called *setup_host2*) and find out whether it is started inside the new UTS namespaces (need to use */proc/* filesystem as done in Task 4).
- Open a third terminal and run $ hostname. Overall, you would see different hostname in all three terminals as each one is running inside a different UTS namespace.
- To exit from both childs (*setup_host1, setup_host2*), run $ *exit* or press Control + d. Use *pstree* to ensure that all processes are dead.

Task 7: Create a child process in a new UTS and MNT namespace. For PID, NET, etc…, it would use the default namespaces.

- In this task, we are going to create two containers as shown below:

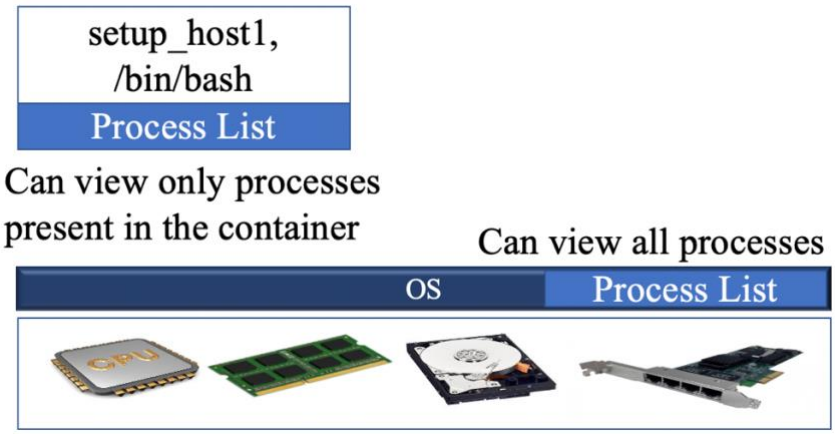


- Ensure that you have copied the following two folders *host1-go1.12.7* and *host2-go1.13.4* to the lab folder. If it is in compressed format such as *host1-go1.12.7.tar.gz*, uncompress it using *tar -xvf host1-go1.12.7.tar.g*. If you are missing these files, please download from https://ibm.box.com/s/kq7pqvbqmq0nvzgydaajangfmlpxykhf (*host1-go1.12.7*) and https://ibm.box.com/s/mwe4hxo5w0tg4rvacehjoslp00uf9q2f (*host2-go1.13.4*)
- In the folder *~/lab/new_uts_mnt*, we have four golang files
  - new_uts_mnt_host1.go
  - setup_host1.go
  - new_uts_mnt_host2.go
  - setup_host2.go.
- Refer to slide 16 to understand what *new_uts_mnt_host1.go* and *setup_host1.go* do. The *new_uts_mnt_host1.go* creates a new child process called *setup_host1* and put it under a new UTS and MNT namespace. Then, the child process changes the hostname (refer to *setup_host1.go*) as well the mount points including the root directory. Similarly, we do the same for host2 as shown in slide 17.
- In *setup_host1.go* and *setup_host2.go*, please provide the full path of *host1-go1.12.7* and *host2-go1.13.4* (as located in your system).
- *new_uts_mnt_host1.go* is creating a child process to execute *setup_host1.go*. Here, the parent process name is *new_uts_mnt_host1* and the child process name is *setup_host1*.
- *new_uts_mnt_host2.go* is creating a child process to execute *setup_host2.go*. Here, the parent process name is *new_uts_mnt_host2* and the child process name is *setup_host2*.
- *setup_host1.go* and *setup_host2.go* executes the command passed by you.
- To compile the code, run the following four commands (under the folder *~/lab/new_uts_mnt*)
  - $ go build new_uts_mnt_host1.go
  - $ go build setup_host1.go
  - $ go build new_uts_mnt_host2.go
  - $ go build setup_host2.go
- After the compilation, to execute the program, start three terminals.

- On terminal 1, to run the new_uts_mnt_host1, run the following command (under the folder ~/lab/new_uts_mnt)
    - $ sudo ./new_uts_mnt_host1 /bin/bash
- As a result of the above executions, you would be inside a child process. Note that the child process called *setup_host1* is running inside a container now with a new UTS and MNT namespace. Check the hostname seen by the child process by executing the command
    - $ hostname
- Also, run $ ls and $ go version to verify whether the mount points have changed.
- On terminal 3, run $ hostname to find out the name on the host machine. This would be different than the one printed in terminal 1. Similarly, run $ go version. It would show the go version installed on the host machine (which might be different from the one in the container).
- Move back to terminal 1.
- Use *pstree* command to ensure that the newly created child process called *setup_host1* does exist.
- Find the *pid* of the newly created child process (called *setup_host1*) and find out whether it is started inside the new UTS and MNT namespaces (need to use */proc/* filesystem as done in Task 4).
- On terminal 2, to run the new_uts_mnt_host2, run the following command (under *~/lab/new_uts*)
    - $ sudo ./new_uts_mnt_host2 /bin/bash
- As a result of the above executions, you would be inside a child process. Note that the child process called *setup_host2* is running inside a container now with a new UTS and MNT namespace. Check the hostname seen by the child process by executing the command
    - $ hostname
- Also, run $ ls and $ go version to verify whether the mount points have changed.
- On terminal 3, run $ hostname to find out the name on the host machine. This would be different than the one printed in terminal 2. Similarly, run $ go version. It would show the go version installed on the host machine (which might be different from the one in the container).
- Move back to terminal 1.
- Find the *pid* of the newly created child process (called *setup_host2*) and find out whether it is started inside the new UTS and MNT namespaces (need to use */proc/* filesystem as done in Task 4).
- Open a third terminal and run $ hostname and $ go version. Overall, you would see different hostname and different go version in all three terminals as each one is running inside a different UTS namespace.
- On terminal 1, 2, and 3, run $ *ps aux* command. In containers as well as the host, we would see all processes as we haven't created a PID namespace. In the next task, we would create a new PID namespace along with UTS and MNT.
- To exit from both childs (*setup_host1*, *setup_host2*), run $ *exit* or press Control + d. Use *pstree* to ensure that all processes are dead.

Task 8: Create a child process in a new UTS, MNT, PID namespace. For NET, USER etc…, it would use the default namespaces.

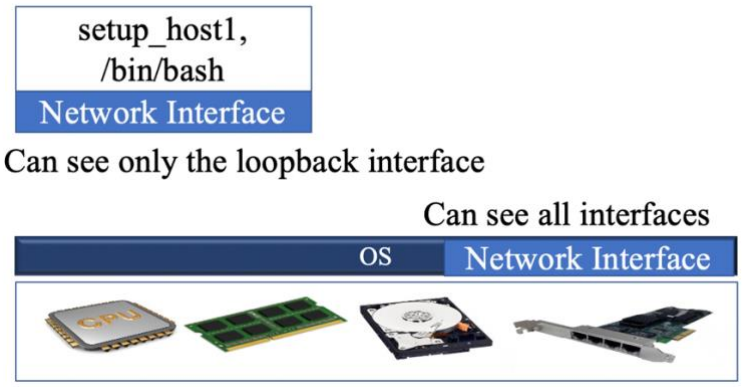
- In this task, we are going to create a container as shown below:



- In the folder ~/*lab/new_uts_mnt_pid*, we have two golang files
    - new_uts_mnt_pid_host1.go
    - setup_host1.go
- By this time, you would be very familiar with how to compile and run the program.
- Assuming that you have created *setup_host1* child process with new UTS, MNT, and PID namespace, if you run $ *ps aux* inside the child process container, you would see only a very few processes. Whereas on the host machine, if you run the same command, you would see all running processes.
- Run the following command $ *sleep 1000* inside the container.
- Run $ *ps aux* inside the container and find the pid of *sleep 1000*
- On the host machine, find the pid of *sleep 1000* using *ps aux* and *grep*. You would find a different pid on the host which is the actual pid of *sleep 1000*
- On the host machine, run $ *kill 9 pid.* Replace pid with the pid of sleep 1000. This would kill the *sleep 1000* process running inside the container. Note that we have killed a process that is running inside the container from host. This could be a security risk which virtual machine can prevent.

Task 9: Create a child process in a new UTS, MNT, PID, NET namespace. For USER, CGROUP etc…, it would use the default namespaces.

- In this task, we are going to create a container as shown below:



- In the folder ~/*lab/new_uts_mnt_pid_net*, we have two golang files
    - new_uts_mnt_pid_net_host1.go
    - setup_host1.go

- Assuming that you have created *setup_host1* child process with new UTS, MNT, PID, and NET namespace, if you run $ *ip a* inside the child process container, you would see only a loopback network interface. Whereas on the host machine, if you run the same command, you would see all network interfaces.

Task 10: Explore unshare and nsenter linux command

- unshare -- $ man unshare
- nsenter -- $ man nsenter

Task 11: Pull a docker image of fedora and run it on top of ubuntu VM.

- Pull the fedora image using the command $ *docker pull*
- Run the fedora on top of ubuntu VM using the command $ *docker run -it fedora /bin/bash*
- Now, we are running fedora on top of ubuntu VM.

Task 12: Start a process in the fedora container and see whether the process is visible at the ubuntu host

- Inside the fedora container, run the following command $ *sleep 1000*
- On the ubuntu host, using the command $ *ps aux | grep sleep*, find the pid of the process as per the ubuntu host. Though we run fedora on top of ubuntu, we can see that the sleep 1000 is a ubuntu process. This is because, we are not really running a fedora OS on top of ubuntu OS as we do in virtual machine. Instead we are running the fedora libraries and binaries so that we run any fedora based application within the container. We still have single OS which is ubuntu.