XV6 Containerization

Just a taste

Yu-Ho Hsieh
yhsieh34

Ver. 2019/04/15

Outlines

- Problem Statement – What do I try to do?
- Demo – Let's start from something interesting
- Implementation – How I approach these?
- Future work – What may happen before 4/26 and 5/4

Problem Statement

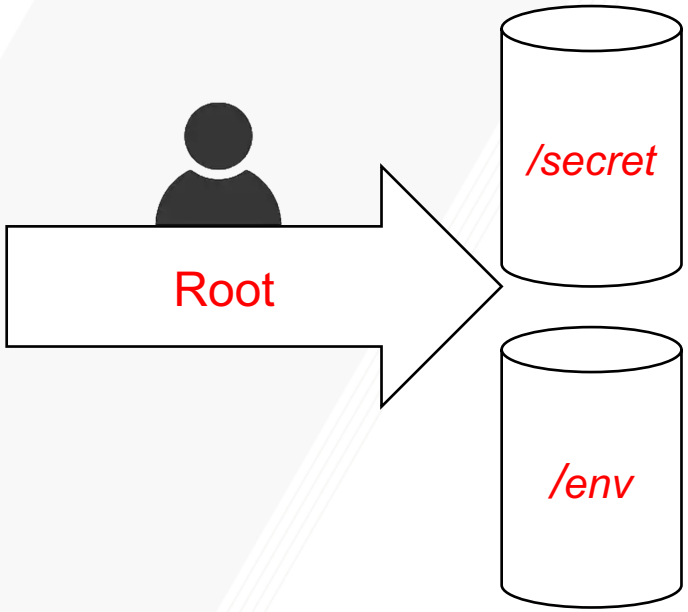
Create a simplified container feature in XV6

Let’s start everything with the demo

Demo

```
$ mkdir  secret
$ mkdir env
$ ls
```

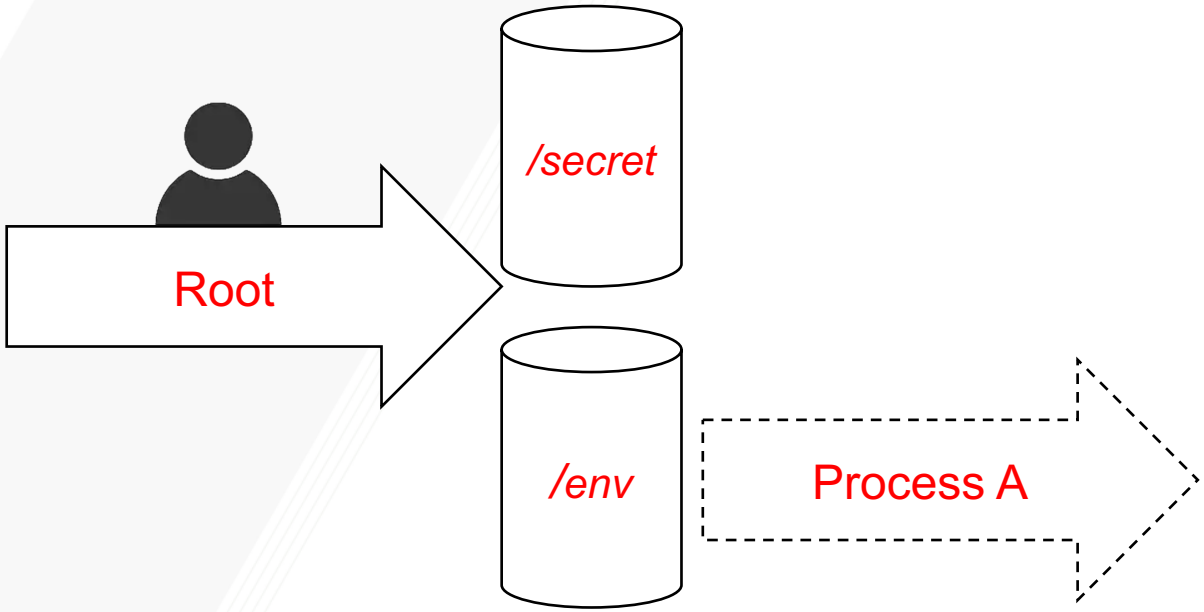
Root creates a *secret* directory and an *env* directory



Demo

```
$ ctool run env
$ ls
```

Root mounts a process on */env* and runs it

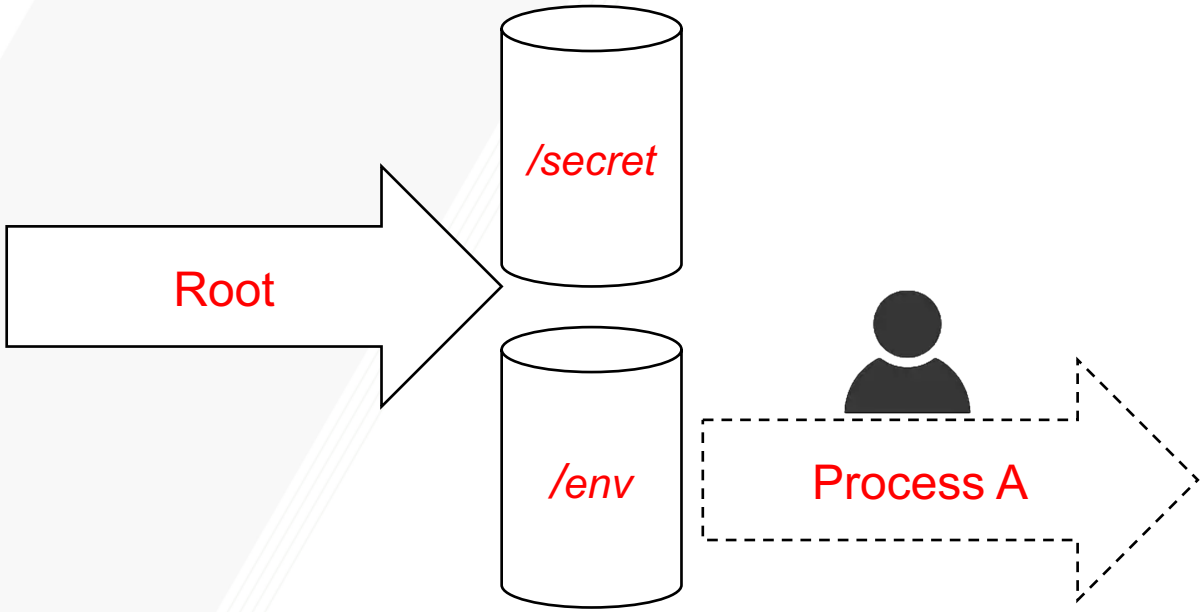


Demo

```
$ ctrl + t
$ ls
```

Hit Ctrl + T to change to container console device

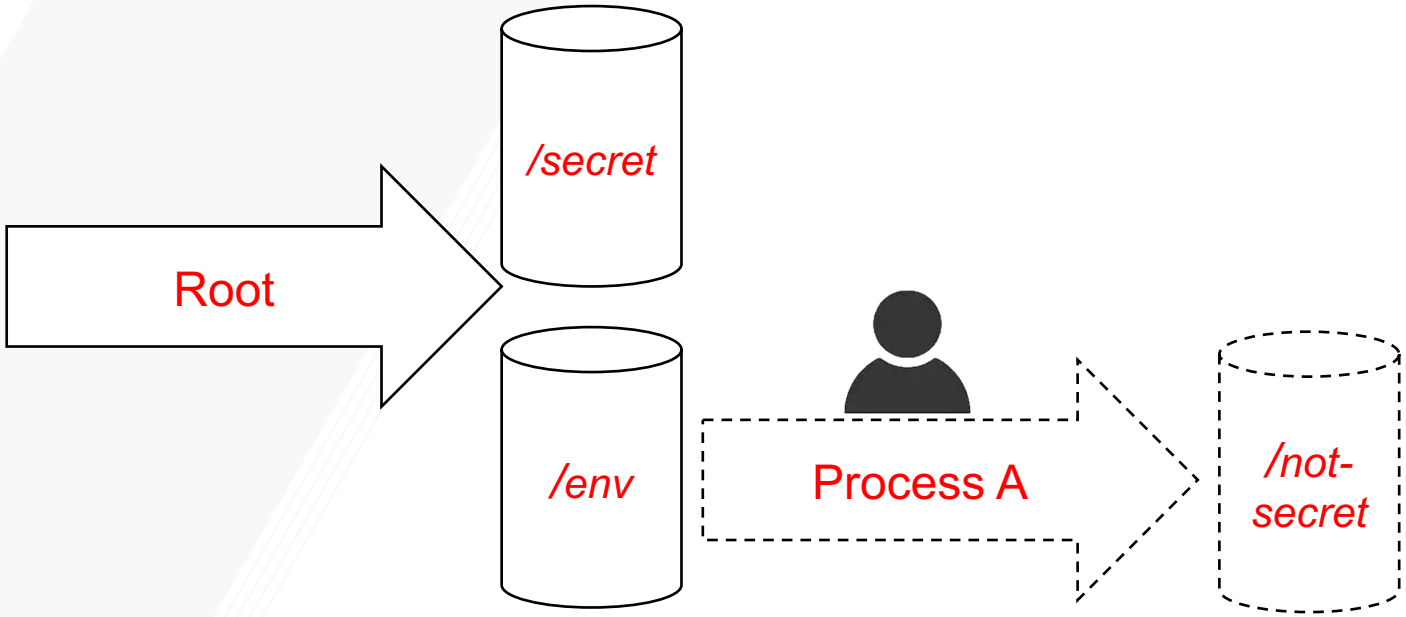
And hit *ls*



Demo

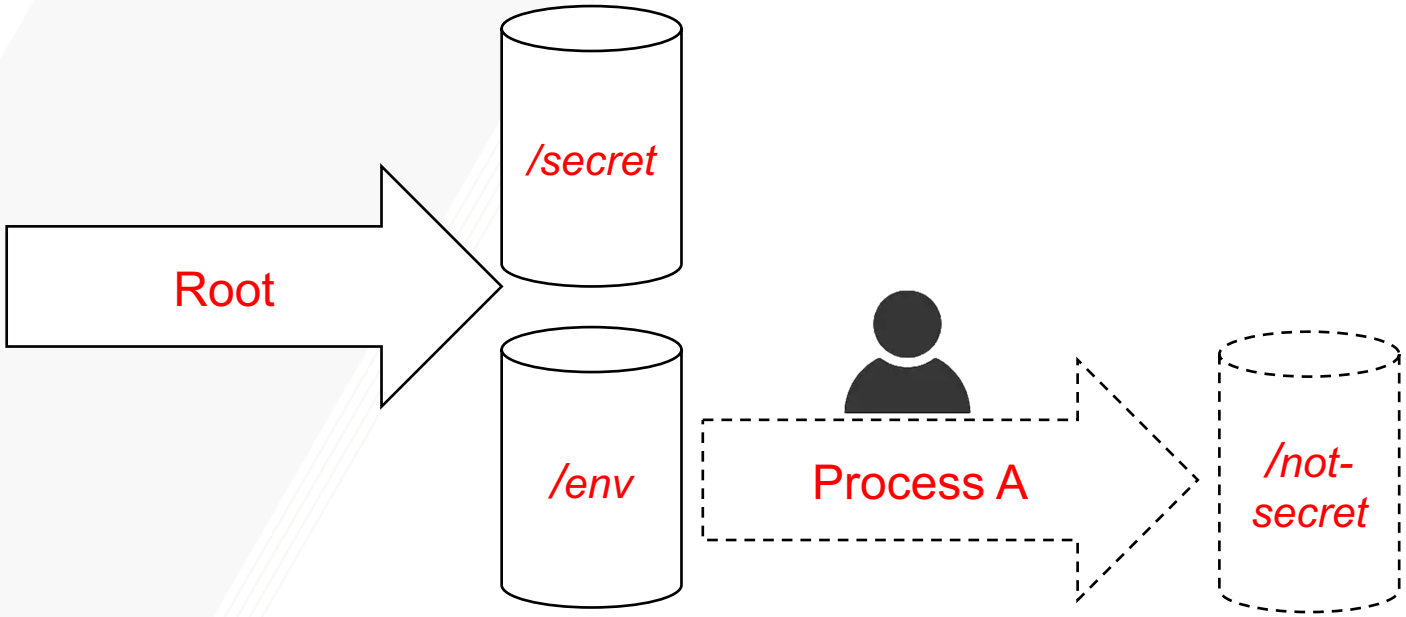
```
$ mkdir notsecret
```

Process A creates a */notsecret* directory



Demo

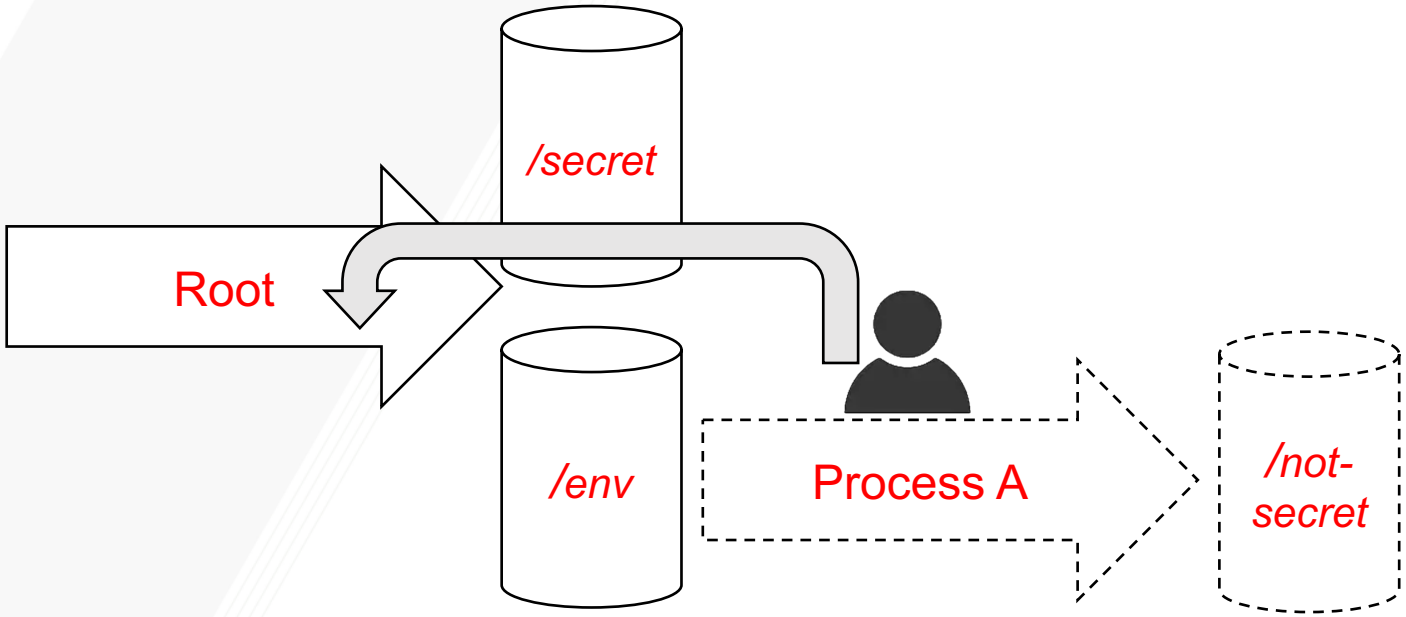
Is it possible to see */secret*?



Demo

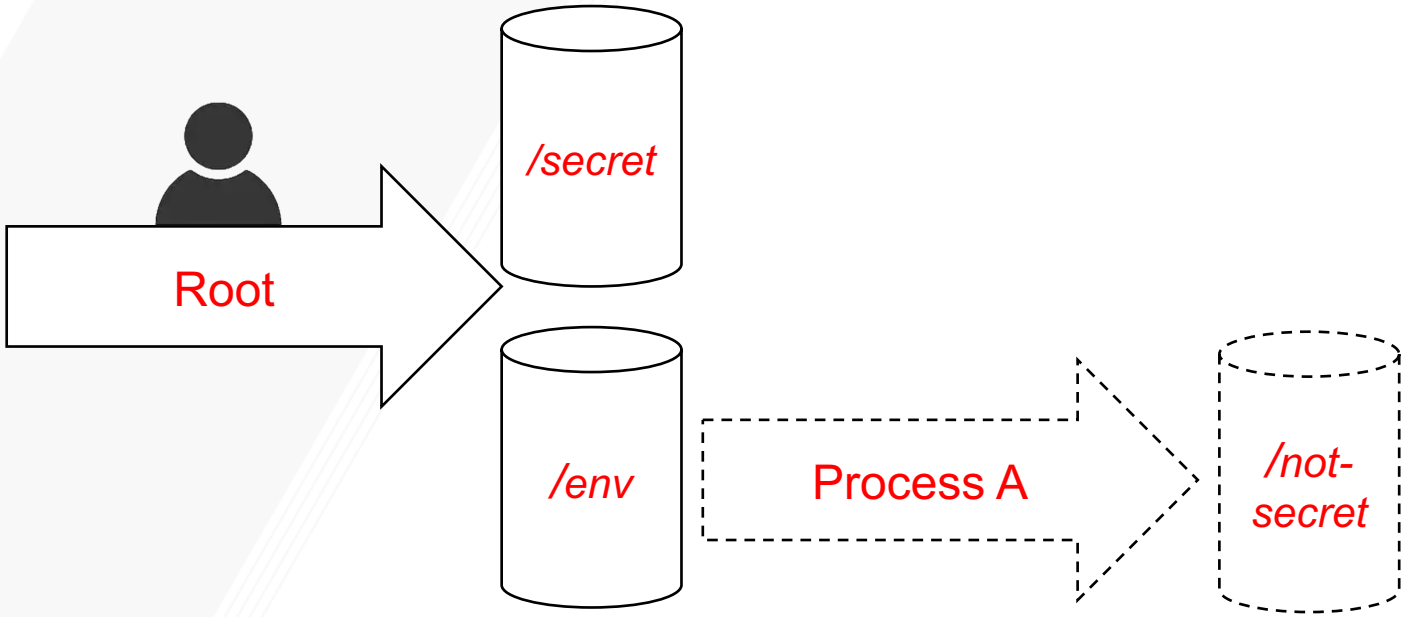
```
$ cd ..
```

Let's move one level up (i.e. *cd ..*)



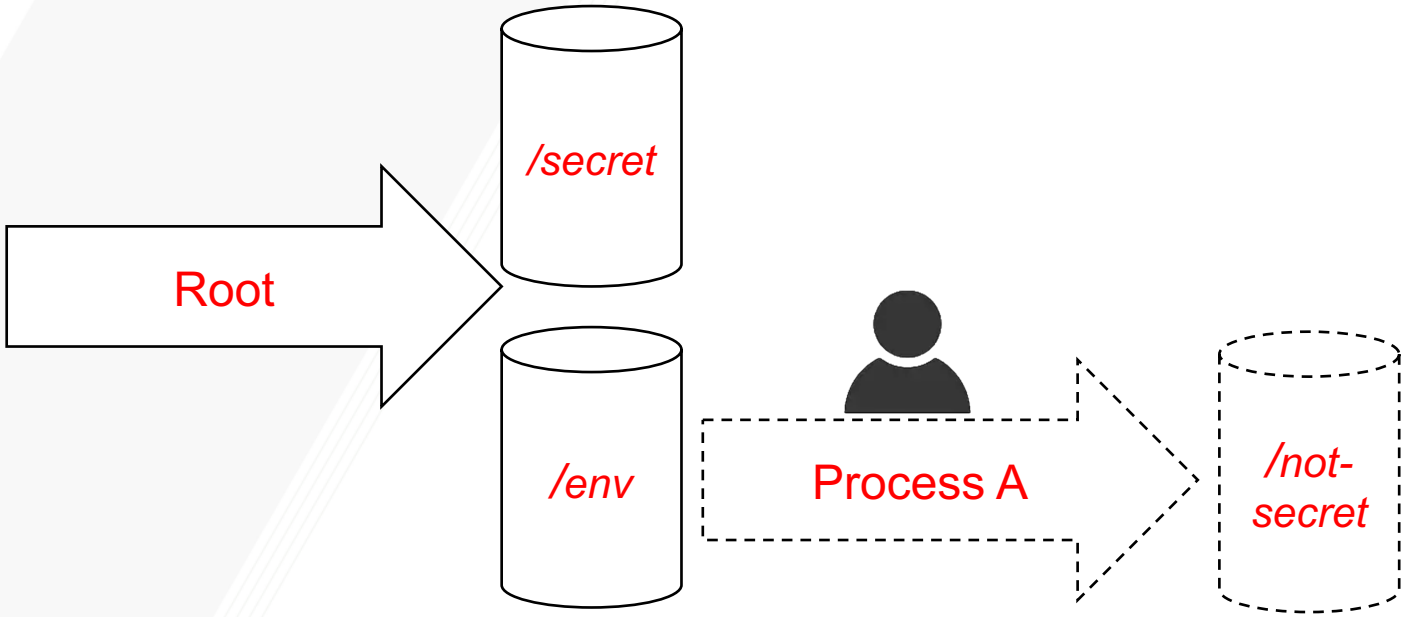
Demo

Shall we see */secret* when we hit */s* now?



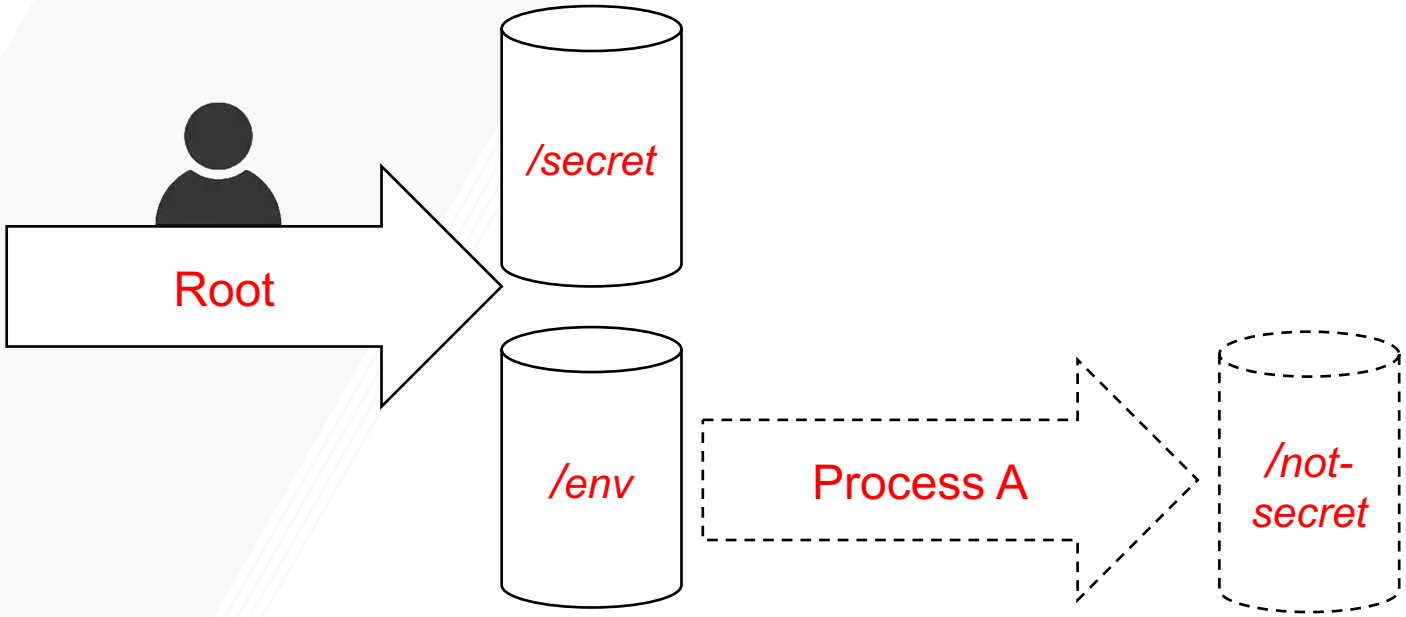
Demo

No! We are still on process A



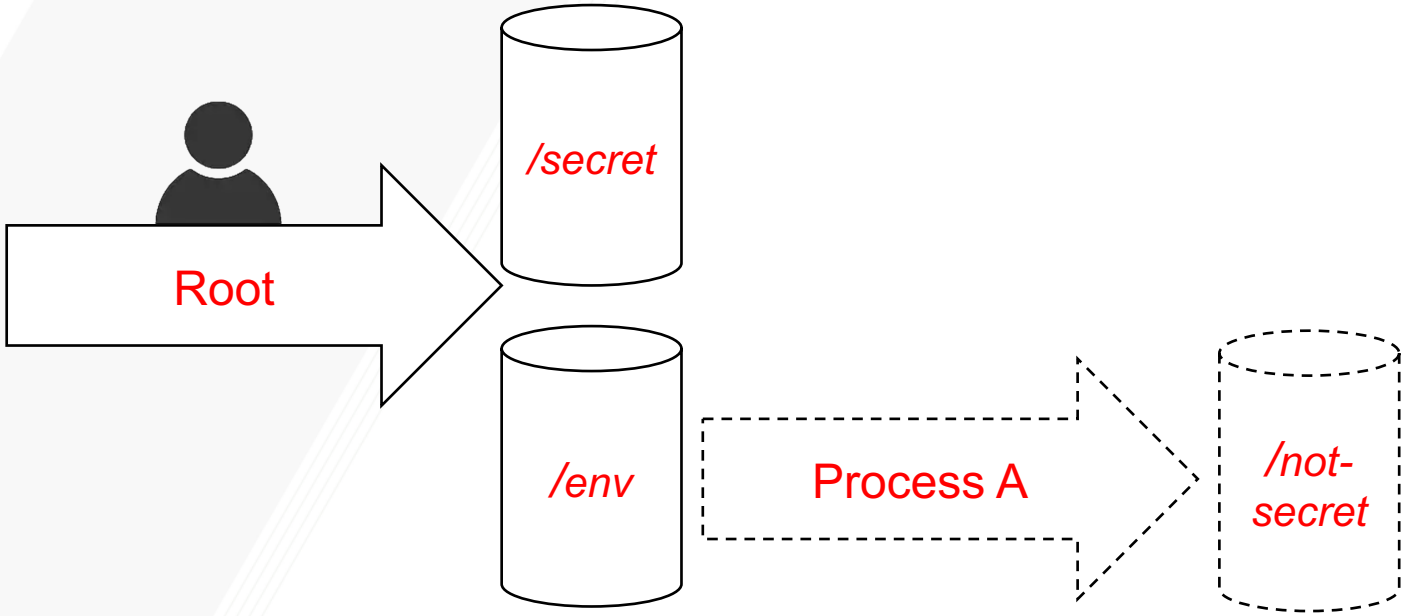
Demo

Hit Ctrl + T to change the back to root console



Demo

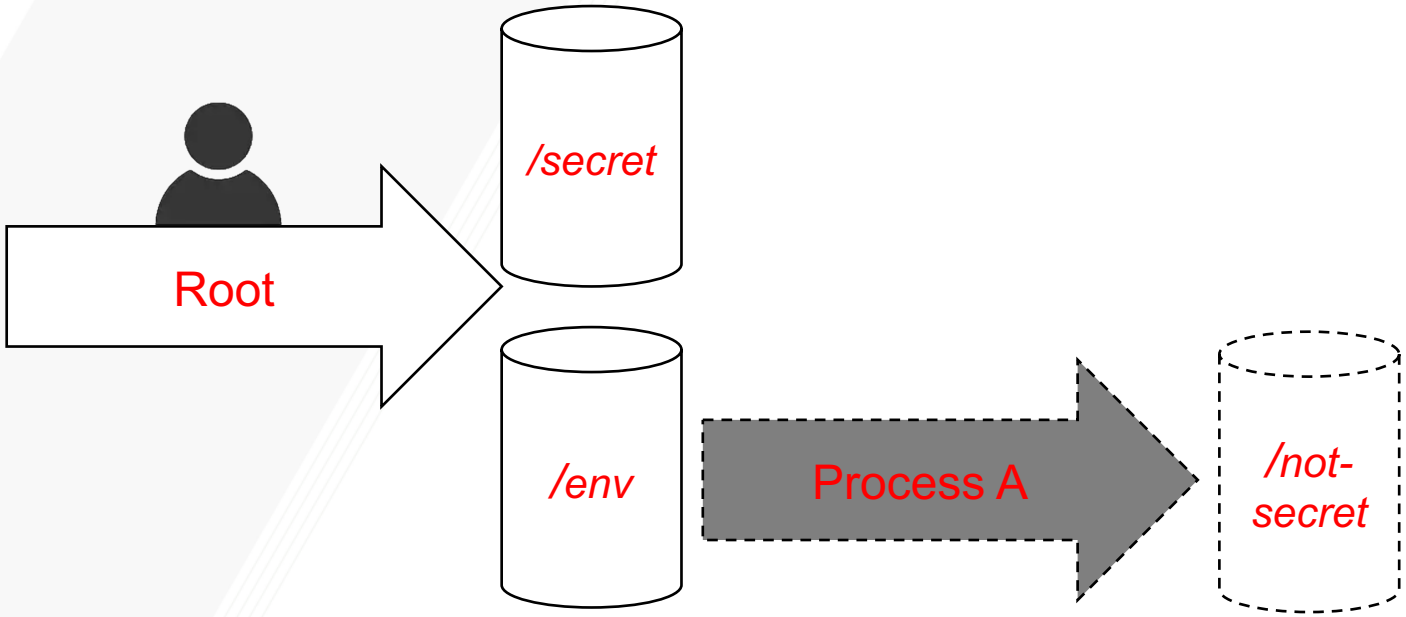
Shall we see */notsecret* under */env*?



Demo

```
$ ctool pause 0
$ ctrl + p
```

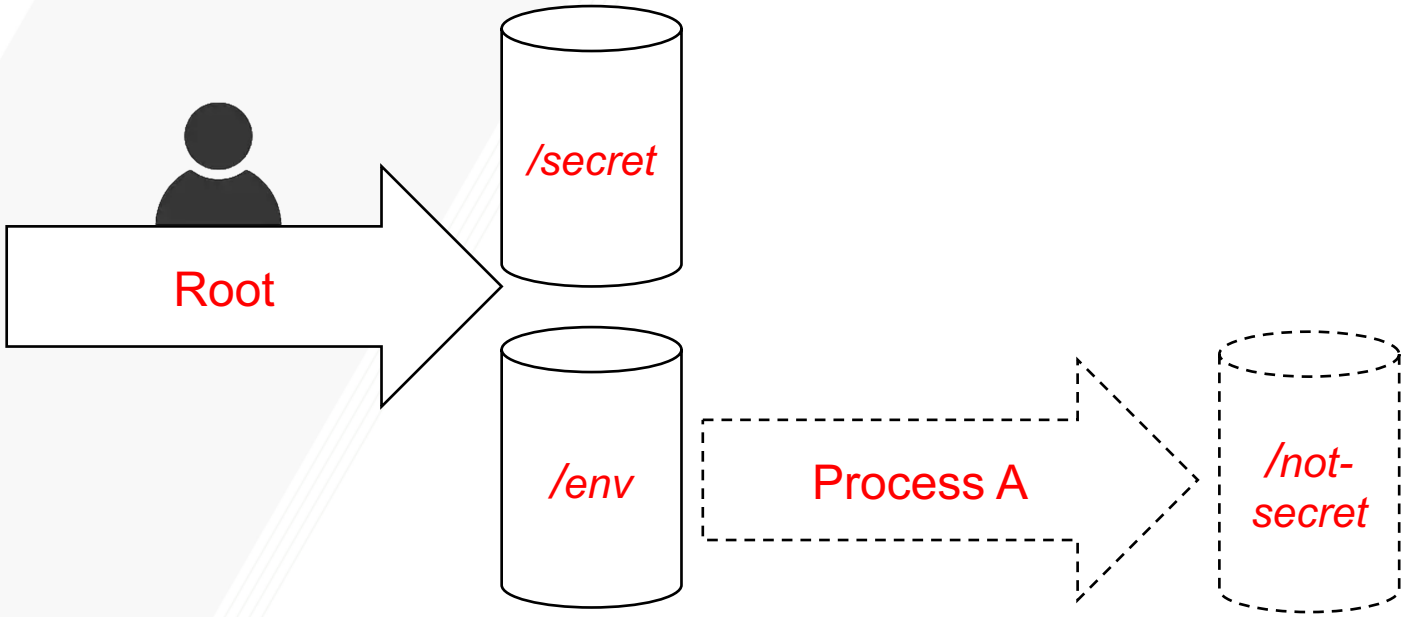
Pause Process A



Demo

```
$ ctool resume 0
```

Resume Process A



Demo

1. Root creates a secret directory
2. Root creates an env directory
3. Root create a process (A) which runs inside the env
4. A creates a directory not-secret in the env
5. A can access not-secret directory but not secret directory
6. Change the console to root process
7. Root can view everything
8. Root can pause the container
9. Root can resume the container

Enough playing, some technical parts…

A Typical Play-around Process

1. Start a container
2. Attach a process to that container
3. Mount the process on specified inode
4. Execute commands in that process
5. Change console between process to see the isolation

Component 1: Console Change

1. Create different device files in using mknod sys call, using minor field in inode to uniquely distinguish the device

2. Create an key interrupt binding in consoleintr function to invoke console change

3. Set up conditions in consoleread and consolewrite function to change the activate device file to stdin the commands and stdout the output

- Key files: console.c, init.c
- Key functions: mknod
- Key attribute: ip→minor

Component 2: Container

1. Create a container table like process table
2. Attach container into process and update the mount inode
3. init → mount → chdir → exec
4. Copy necessary exec files into the container node

- Key struct: container
- Key file: container.c
- Key function: setcont()

Component 3: File Isolation

1. Modify the key concept in namex() function in fs.c

2. Define the behavior if the process tries to access outside the the container

- Key file: fs.c
- Key function: namex()

What remains: Pause and Resume

1. Pause: make the process not runnable

2. Resume: make the process runnable

What remains: Delete

1. Free the container from process

2. Remove the mounted node

Questions ?

I know most of you do not have.

So I prepare some for you ☺

1. During the clean up, should the child process be responsible to clean the folder or the root process?

2. Who should give the command to pause | delete the container? Root or Child?

Challenges / Questions

1. Design the behavior, who has the right to do what
2. Multiple processes in same container

Thank you so much!

Georgia Tech
CREATING THE NEXT