COL331 Operating Systems

Assignment 3 - Virtualization in XV6

Name: Tarun Kumar Yadav/Minhaj Shakeel **Entry No: 2016CS10359/2016CS10354**

1 High Level Details

1. All of container functionalities have been implemented in kernel.
2. Scheduler follows a round-robin strategy.
3. Copy on write and Copy on read mechanisms have been implemented.

2 Container Manager

Container manager is kernel data sturctures maintained by kernel only. A process can create a container and a process not inside any container can destroy a container. implemented system calls:

```
uint create_container(void)
uint destroy_container(uint container_id)
```

create_container() system call searches for an available container, sets its process count, open file count to zero. destroy_container() system call ensures caller is authorized for the call. If so, kills are the processes inside the container, deletes are files owned by processes in the container and sets the container free.
Processes can use the following two calls to join or leave container.

```
uint join_container(uint container_id)
uint leave_container(void)
```

A process already inside a container can not join any other container. join_container() system call just adds the process to its list and it becomes the responsibility of this container to schedule this process. leave_container() just deletes the entry and the process scheduling is handled by kernel.

3 Scheduler

The scheduler follows a round robin strategy. For one round of non-container process scheduling, one process from each container is scheduled. The processes inside container are also scheduled in round robin fashion.

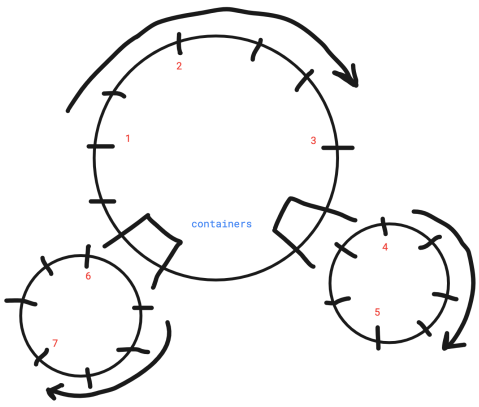

Figure 1: Round robin scheduling

The schedule followed in the figure is:

```
1, 2, 3, 4, 6, 1, 2, 3, 5, 7
1, 2, 3, 4, 6, 1, 2, 3, 5, 7
...
```

The chosen strategy is fair as kernel 'can not see' the processes inside container and only container is scheduled per round.

3.1 Log calls

```
void scheduler_log_on(void);
void scheduler_log_on(void);
```

These two calls turn global flags ON/OFF in kernel, so that any scheduling event is/is_not printed.

```
int scheduler_log_toggle = 0;
```

4 Resource Isolation

4.1 ps

Processes inside container can only 'see' processes inside container. This is achieved by adding the following filed to PCB.

```
uint container_id;
```

4.2 ls

Files made by container processes are appended with special character followed by container_id. Then ls just searches all files in directory, and prints those without the special character(these from non-container processes) and those with special character and container_id attatched.

4.3 Copy on open/write

Two methods to isolate the changes to file system have been implemented. Copy on open makes a copy of file immediately if a container processe opens a non-container file.

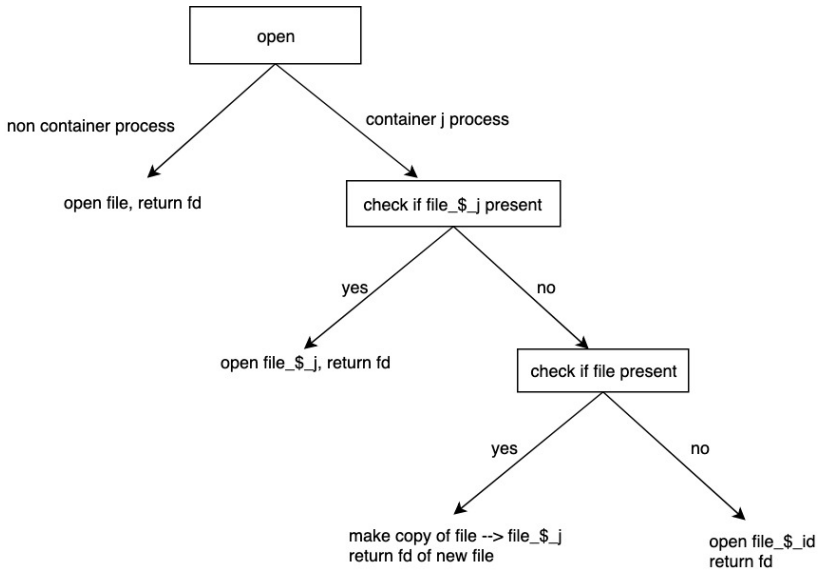

Figure 2: Copy on open mechanism

This way we need not maintain a cfd – fd – pid table.
Copy on write mechanism is a little more complicated as many tables need to be maintained. For example:

1. cfd – fd – pid table, to translate every fd request.
2. fd – name table, to make a copy of file on write.
3. fd – container table, to check if file is owned by container, to ensure double copy of file is not made.

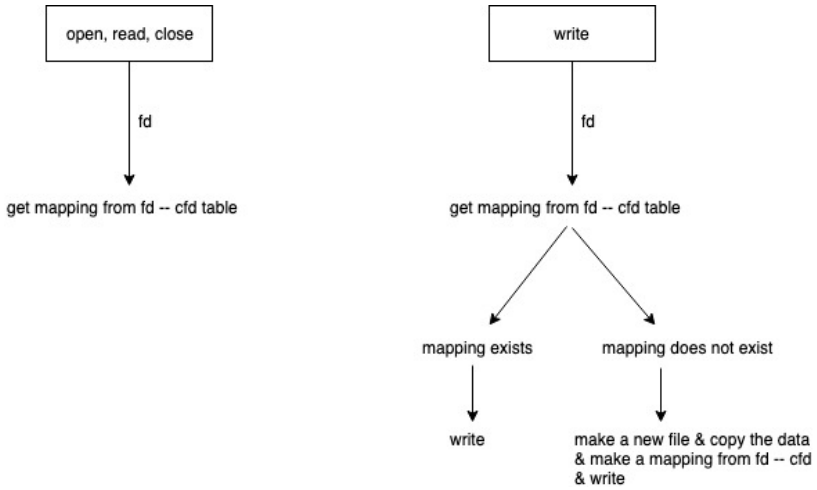

Figure 3: Copy on write mechanism