COL331 – Operating Systems

Assignment 3 Report

Akshat Khare (2016CS10315)
Divyanshu Saxena (2016CS10316)

Overview

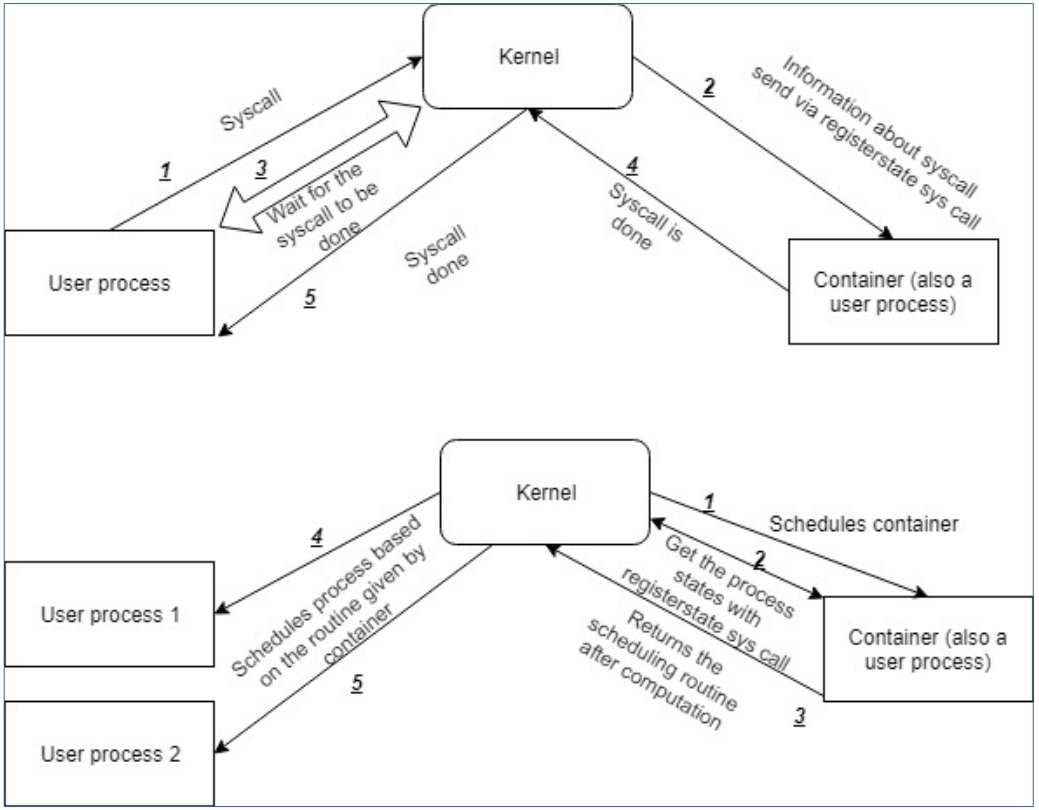
- The assignment gives an implementation for Containers on xv6 toy Operating System.
- The entire implementation of the Container has been done on a **User Process**. Some changes to the system calls have been made so as to allow efficient and effective isolation among containers.
- The implementation has been done so as to resemble the work flow as in actual Containers, as closely as possible.

Structure (I)

- The scheduler for the containers has been implemented in the user process for the container itself – thereby allowing **separate scheduling policies** for separate containers. This has been implemented through a unique **registerState** system call.
- The **registerState** system call can be called only by a container process. It performs the following three tasks:
  - Provides the container process, information about its assigned processes.
  - Takes in the next scheduled process from the container. Then, sleeps the current running process and makes the scheduled process from the container, runnable – thereby enabling its scheduling by the kernel scheduler.
  - Works as a communication between process and the assigned container. (More details have been mentioned later)

Structure (II)

- Below is the depiction of the flow of execution between containers and their assigned processes. The first diagram shows the communication of the system call from the process to the container and the second diagram shows the scheduling from the container process.



Operations (I)

- The basic set of operations have been defined as follows, in the form of system calls:
  - The **create_container** system call makes the calling user process a container and
  - The **join_container** system call joins the current calling process to the given container, and sets the appropriate variables in the process table (required for our implementation).
  - The **leave_container** system call removes the assignment of the current calling process to its respective container and unsets all the variables set in **join_container.**
  - The **destroy_container** system call unsets all assignments of processes to the current calling container process and kills the container. Note that the processes *still remain*, but unassigned.

Operations(II)

- **ls** command has been implemented as a system call, where in only those files are printed that are **either global** to all containers or the files that have been **created within** the container assigned to the calling process.
- Similarly, **ps** system call has been modified to allow for the container processes and global processes to be listed out. A **process-container mapping** in the kernel has been used for this purpose.
- **COW** (Copy on write) has been implemented using the File System calls. The **open** system call, simply opens a file and gives a virtual file id to the process. When a process writes to this file, the **write** system call checks whether the file belongs to the current container or not. If it does not, then a new duplicate file of the format ***#<cid>#<filename>*** is created by the container process and the changes are made to this new file. Also, the earlier virtual file id is internally mapped by the container to this newly created file – so that the process does not know whether there has been a change in the file system.

Extra Work

- Scheduler implemented in **User Process.**

  [Advantage:- Allows **separate scheduling strategies** for separate containers]

- System calls sent by non-container processes are **trapped by the kernel** and sent to the respective container process.

  [Advantage:- **No message passing** between process and container is required. **No change in system calls** required for the user programs. Hence, all programs from non-container implementations can be shipped directly into our container implementation]

- Overloaded operations over malloc have been provided in our container implementation. (**malloc, writemalloc, readmalloc** system calls have been implemented). Malloc has been implemented using modified **page table** (proxy addressses used by user process)

  [Advantage:- Extending th concept of resource isolation of files to variables and page tables, we have succesfully given implementation for **variable management** over containers as well]