COL380 A3 Report

Shashwat Shivam - 2016CS10328

24 April, 2019

1 Points about implementation

Please note the following points before reading further explanations :-

- The container systems have been implemented inside the kernel.
- All required data structures have been defined in proc.h and proc.c.

2 Required Data Structures

2.1 Container Table

To store the states of the containers a struct named container has been defined on similar lines as of the proc table. An associated lock is defined to ensure the accesses to the container table are exclusive. Container struct has the following fields :-

- **Container ID**: This contains the unique container id which is assigned when a container is created. These are assigned in a increasing number fashion.
- **Container States**: This field helps distinguish between containers **IN_USE** and containers **UNUSED**.
- **Pmap**: This is explained in the subsection 2.2.
- **Process id**: This stores the number upto which local pids have been assigned.

- **Num_procs_active**: Tells the number of processes which are active in the current container.

2.2 Pmap

This struct contains the following fields :-

- **local_id**: This array stores the local pids of the processes inside the container.
- **global_id**: This array stores global pids of processes inside container.
- **state**: This array stores the state of the process inside the container. For simplicity purposes this has been limited to two options Active or unused slot.
- **name**: Stores the names of processes inside the container.

2.3 Other Modifications/Additions

- **Container_scheduled_next**: Stores the container_id of the next scheduled container.
- **Process_scheduled_next**: Stores the value of the next process scheduled by each container. (To be used by scheduler).
- **Container id proc**: This field has been added in process struct (**proc.h**).
- **container_id dirent**: This field was also added into the directory entry data structure which is used to create files with list of files in a certain directory (**fs.h**).
- **container_id inode**: This field was added into inode struct which temporarily stores files in memory when modifying them (**file.h**).

3 Standard Operations

3.1 create_container()

The create system call searches for an **UNUSED** slot in the container table (after acquiring a lock). When an empty space is found, it initializes the value of its container id. It returns the unique container id of the created container.

3.2 join_container(uint container_id)

The join container system call modifies the container id of the current process. It the finds an empty slot in the pmap table of the container. Writes the global id to the process and also assigns a local id to the process unique in the container. The number of procs inside the container is increased by 1.

3.3 leave_container()

The leave container command removes the process from its specific container and also marks the slot as empty. The number of procs in the container is decreased by 1.

3.4 destroy_container(uint container_id)

The destroy container command first kills any process that is inside the container. Then it marks the corresponding container slot in the table as free.

4 Command Outputs

4.1 ls command

The ls command was initially provided as a user program. This was converted into a user side library. Then the modified directory entry structure was included to store the container id of the specific file (0 for default). The new ls command also cross-verified the container id of the process with the file to not print any files which were from other containers. The last column in the ls command now corresponds to container id.

4.2 ps command

The ps command was implemented from scratch (initially unavailable). This was modified to use the container table pmap struct(explained in 2.2). The processes were listed from the pmap table so that only the processes from the container to which the process belonged would be listed. (The code can be found in function print_processes() in proc.c)

4.3 COW mechanism

This function is implemented in multiple parts. First when a file from host device (no container) is opened by process of some container then it is opened normally.Here containers are identified by the dirlookup function which has been modified to return files from host device and process's container. The dirlookup checks the directory entry struct for container id of the files.

When the file is loaded into the memory, it is stored as an inode. The container id of the file is also loaded into the inode. When a write is made from a container to the host system (sys_write is called) , we check and detect such a case. We open a new file and write the contents of the old file into the new file and modify the file descriptor of the open file to point to the new file. The new file has a container id specific to the container from which the write is being made. The details of this operation can be found in **sys_write()** in **sysfile.c**.

5 Other Additions

5.1 Virtual Scheduler

A virtual scheduler is implemented as a function call. The function schedule_next_container() which starts searching from previously executed container in a cyclic manner for a container which has a RUNNABLE process. The previously executed process for each container is also saved so that within each container the processes execute in a round robin fashion. The code for this can be found in function schedule_next_container() in proc.c.

5.2 Malloc

Malloc exists as a user level memory manager library in xv6. The pages are allocated to a process at its start. To show the usage of memory instead of catching memory allocations in sbrk (which is called very infrequently), we add a function in malloc itself (make_page_table_entry). This function is just a dummy function which prints a memory address as a function of process id and malloc'ed address.

6 Test Script

The test script can be executed by using command user_test_assig3. The outputs are there along with explanations. Also the test script is commented to parse the output easily and check the contents and behaviour