COL331 Report: Assignment-3

Manav Rao • 2016CS10523
Vinayak Rastogi • 2016CS10345

Overview

Date of Submission

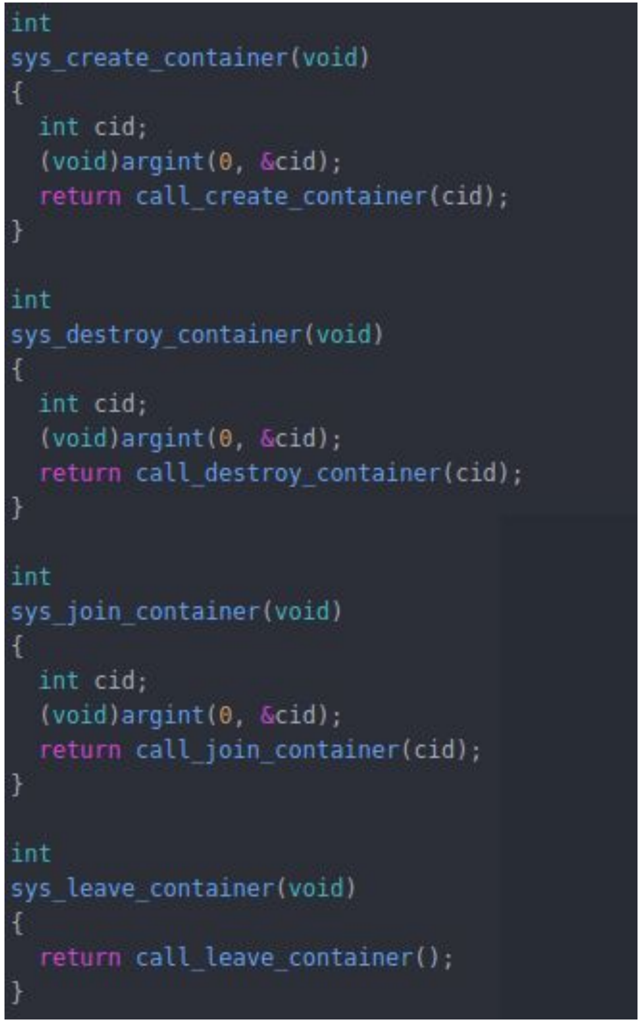
April 25th, 2019

Report Contents

- Standard operations
    - Create Container
    - Join Container
    - Leave Container
    - Destroy Container
- Command Output
    - ls Command
    - ps Command
    - Copy-On-Write Mechanism
    - Scheduling Mechanism

Objective:

Implementing container related services in XV6 toy Operating System to understand Virtualization in operating systems.

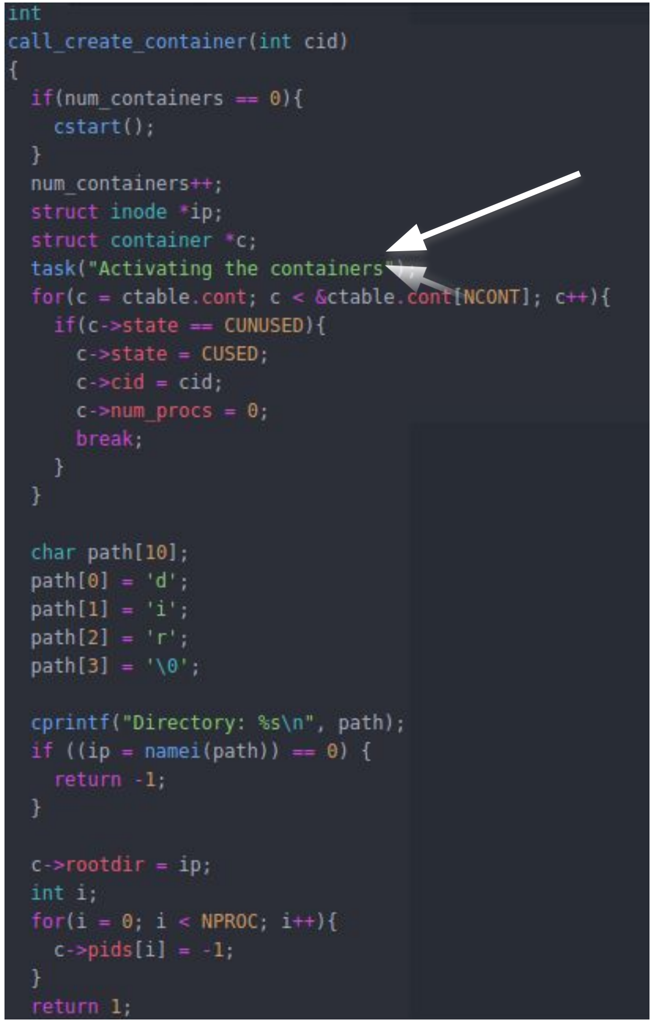


```
int
sys_create_container(void)
{
  int cid;
  (void)argint(0, &cid);
  return call_create_container(cid);
}

int
sys_destroy_container(void)
{
  int cid;
  (void)argint(0, &cid);
  return call_destroy_container(cid);
}

int
sys_join_container(void)
{
  int cid;
  (void)argint(0, &cid);
  return call_join_container(cid);
}

int
sys_leave_container(void)
{
  return call_leave_container();
}
```

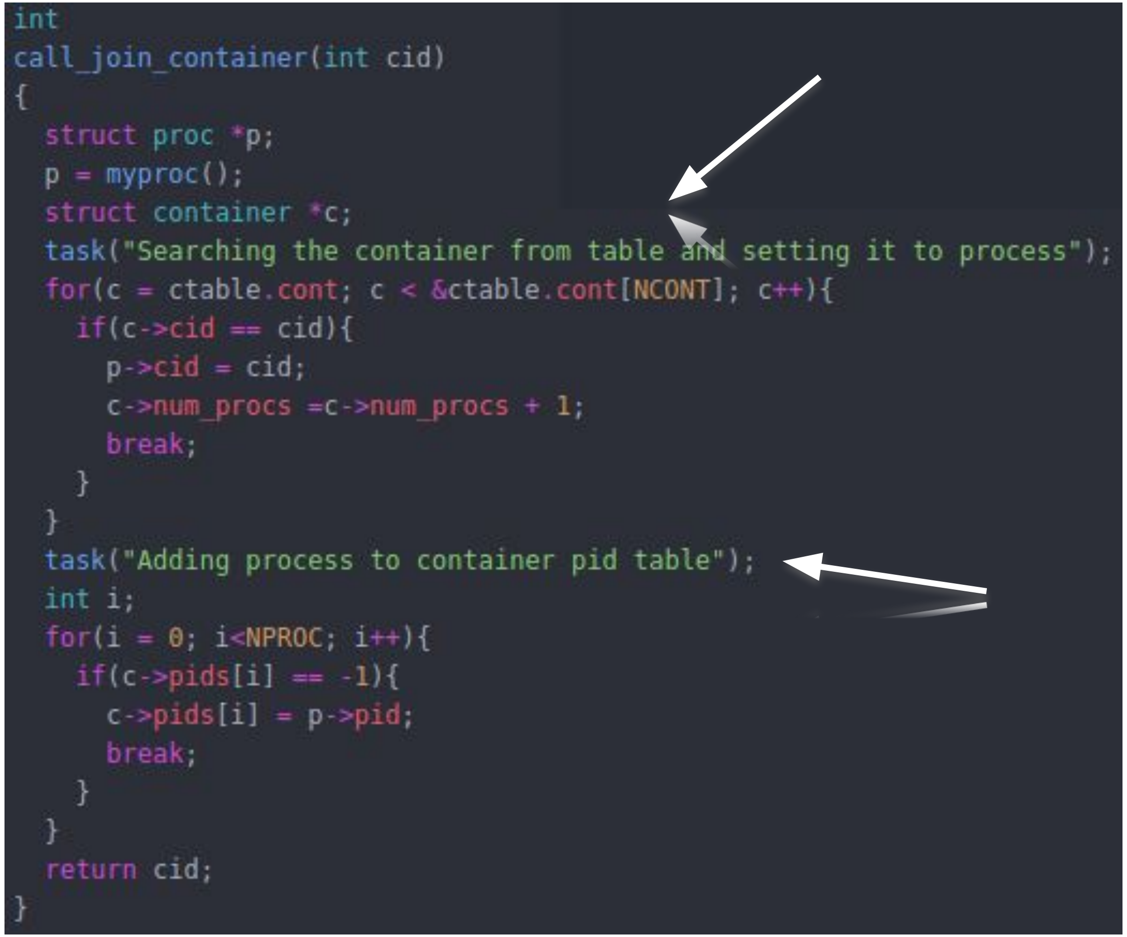
STANDARD OPERATIONS AND BASIC FUNCTIONALITIES:

1) **Create Container** :-
   a) First inactive/unused slot chosen for initialization
   b) Changes status to active and return cid of cont.
2) **Join Container** :-
   a) Process is assigned the first empty slot in the cont. w/c it requests to join
   b) If required a new cont. Can also be created with 0 initial processes
3) **Leave Container** :-
   a) Find cid of the container to w/c process belongs
   b) Number of Process decreased by 1 after step (a)
   c) If it were last process, set state of cont. to 'READY'
4) **Destroy Container** :-
   a) Kill all processes of the container
   b) Container slot state set to UNUSED

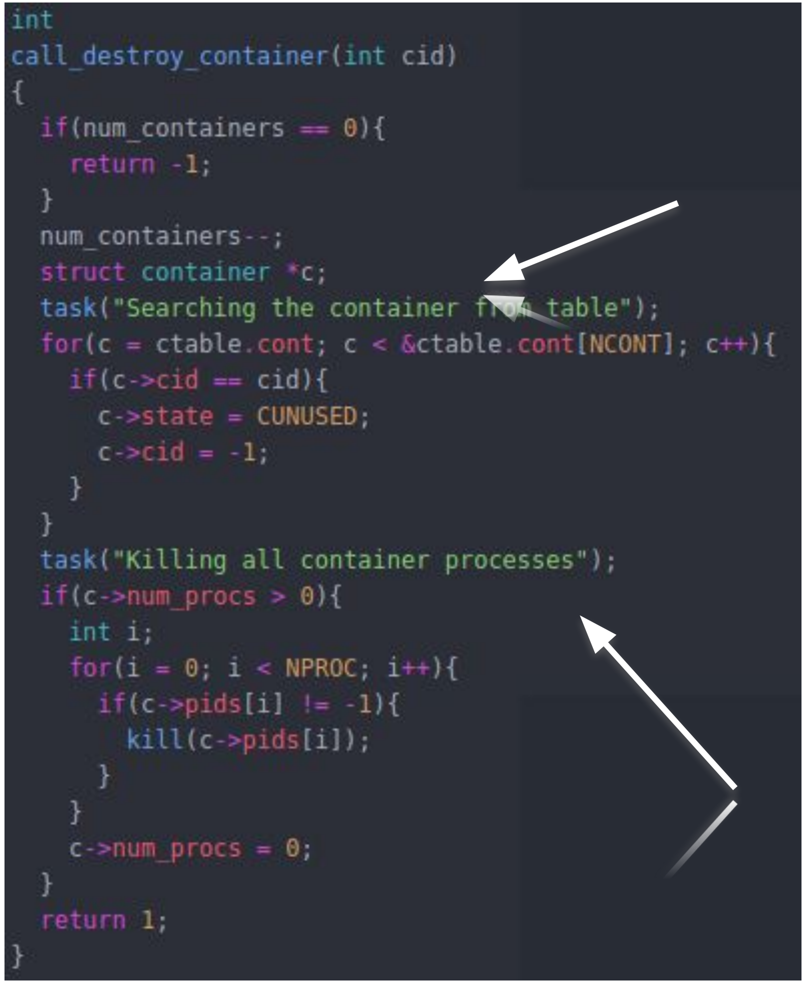
*Container Table -> A seperate Table like Data-Structure to keep track of all the containers mimicking how OS keeps track of individual processes.



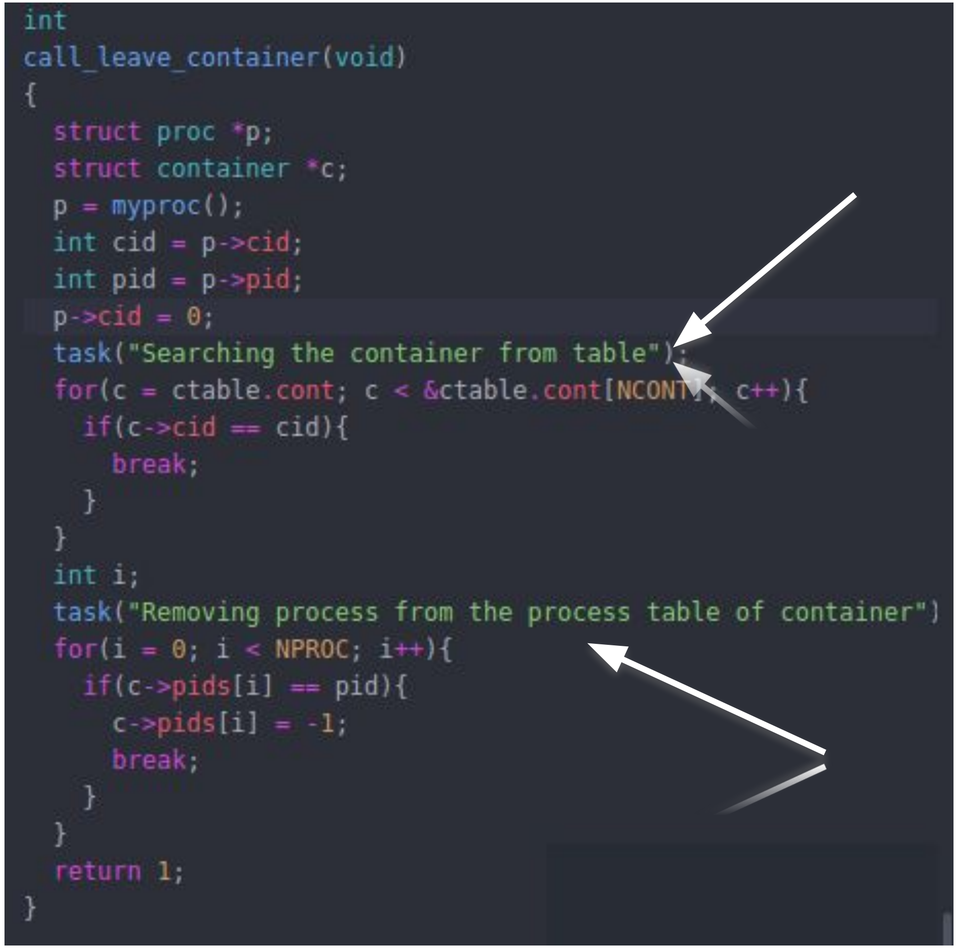
```
int
call_create_container(int cid)
{
  if(num_containers == 0){
    cstart();
  }
  num_containers++;
  struct inode *ip;
  struct container *c;
  task("Activating the containers");
  for(c = ctable.cont; c < &ctable.cont[NCONT]; c++){
    if(c->state == CUNUSED){
      c->state = CUSED;
      c->cid = cid;
      c->num_procs = 0;
      break;
    }
  }

  char path[10];
  path[0] = 'd';
  path[1] = 'i';
  path[2] = 'r';
  path[3] = '\0';

  cprintf("Directory: %s\n", path);
  if ((ip = namei(path)) == 0) {
    return -1;
  }

  c->rootdir = ip;
  int i;
  for(i = 0; i < NPROC; i++){
    c->pids[i] = -1;
  }
  return 1;
```



```
int
call_join_container(int cid)
{
  struct proc *p;
  p = myproc();
  struct container *c;
  task("Searching the container from table and setting it to process");
  for(c = ctable.cont; c < &ctable.cont[NCONT]; c++){
    if(c->cid == cid){
      p->cid = cid;
      c->num_procs =c->num_procs + 1;
      break;
    }
  }
  task("Adding process to container pid table");
  int i;
  for(i = 0; i<NPROC; i++){
    if(c->pids[i] == -1){
      c->pids[i] = p->pid;
      break;
    }
  }
  return cid;
}
```



```
int
call_destroy_container(int cid)
{
  if(num_containers == 0){
    return -1;
  }
  num_containers--;
  struct container *c;
  task("Searching the container from table");
  for(c = ctable.cont; c < &ctable.cont[NCONT]; c++){
    if(c->cid == cid){
      c->state = CUNUSED;
      c->cid = -1;
    }
  }
  task("Killing all container processes");
  if(c->num_procs > 0){
    int i;
    for(i = 0; i < NPROC; i++){
      if(c->pids[i] != -1){
        kill(c->pids[i]);
      }
    }
    c->num_procs = 0;
  }
  return 1;
}
```



```
int
call_leave_container(void)
{
  struct proc *p;
  struct container *c;
  p = myproc();
  int cid = p->cid;
  int pid = p->pid;
  p->cid = 0;
  task("Searching the container from table");
  for(c = ctable.cont; c < &ctable.cont[NCONT]; c++){
    if(c->cid == cid){
      break;
    }
  }
  int i;
  task("Removing process from the process table of container")
  for(i = 0; i < NPROC; i++){
    if(c->pids[i] == pid){
      c->pids[i] = -1;
      break;
    }
  }
  return 1;
}
```

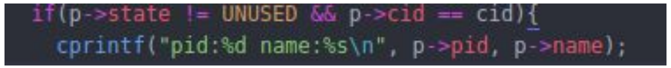
COMMAND OUTPUTS OF BASIC SYSTEM CALLS:

‘ls’ command

- Name of the files decided by creator of the files
- The container name is appended to the file name
- When the ls command is called the files which are displayed include:
    - Files created by main (already present files)
    - Files created by the process itself (which are appended by the cid)

‘ps’ command

- It displays the process count as earlier.
- The only difference is that it now displays the process count of a container.
- A check on cid is placed to maintain the same.

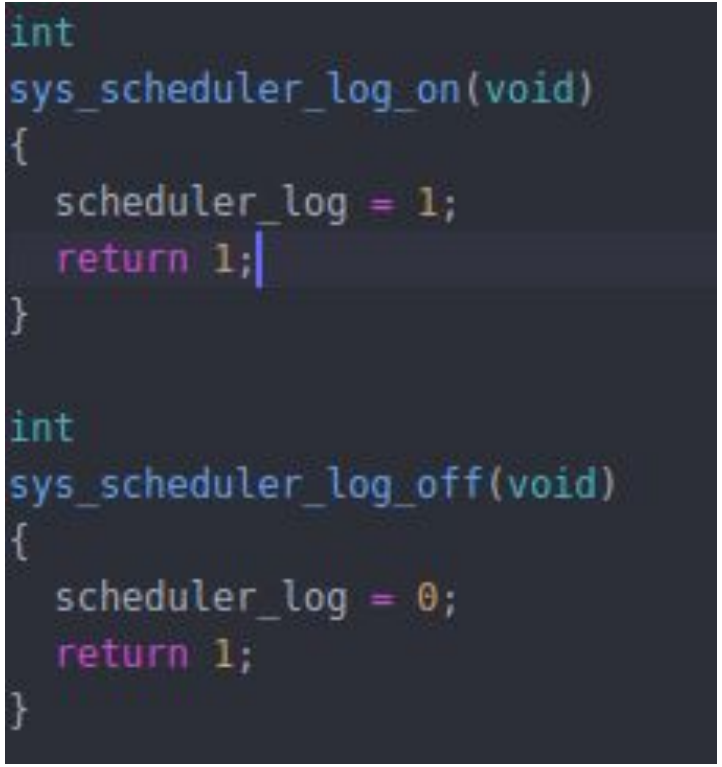


```
if(p->state != UNUSED && p->cid == cid){
  cprintf("pid:%d name:%s\n", p->pid, p->name);
```

COMMAND OUTPUTS OF BASIC SYSTEM CALLS:

Copy on Write Mechanism

- The private path/name of the file is decided for the container based upon the checks to find if the process is created by the container or not
- If the file is already created by another process in the same container then the same file is opened for the current process
- Depending on the Container Id, a separate file with a unique name is derived from the original parent file is created for different containers
- By this mechansim, this file will not be accessible to other containers



```
int
sys_scheduler_log_on(void)
{
  scheduler_log = 1;
  return 1;
}

int
sys_scheduler_log_off(void)
{
  scheduler_log = 0;
  return 1;
}
```

SCHEDULING MECHANISM:

- A simple kernel end scheduler is maintained.
- A toggle variable is kept to maintain the same.
- In the main scheduling task, the cid is cross checked corresponding to the scheduler_toggle.