Lab 4 Report

Alp Can Aloglu, Najmeh Mohammadi Arani

SID : 862190749 / 862216499

E-mail: aalog001@ucr.edu / nmoha034@ucr.edu

- **All code changes you made on xv6 source code** (hint: `diff -r original_xv6 your_xv6`)

  Code diff is submitted through different submissions.

- **Detailed explanation about these changes**

  shm.c

```
35,36d34
<     struct proc *current_process = myproc();
<     uint sz = PGROUNDUP(current_process->sz);
38d35
<     acquire(&(shm_table.lock));
40,41d36
<     int i; int j = 64;
<     int id_existence = 0;
43,71c38
<     for(i = 0 ; i < 64 ; i++){ //there are 64 pages
<        if(j == 64 && shm_table.shm_pages[i].id == 0) {//segment id does not
exist
<          j = i;//store the first empty page number
<        }
<        if(shm_table.shm_pages[i].id == id){//segment id exist
<          id_existence = 1;
<          // add the mapping between the virtual address and the physical
address
<          mappages(current_process->pgdir , (char *)sz, PGSIZE,
V2P(shm_table.shm_pages[i].frame), PTE_W|PTE_U);
<          //increase the refence count
<          shm_table.shm_pages[i].refcnt++;
<          break;
<        }
<     }
<
<     //if we did not find the segment id, allocate a page and map it, and store
```

```
this information in the shm_table
<     if(id_existence == 0 && j < 64){
<         shm_table.shm_pages[j].id = id;
<         shm_table.shm_pages[j].frame = kalloc();
<         memset(shm_table.shm_pages[j].frame, 0, PGSIZE);
<         mappages(current_process->pgdir, (char *)sz, PGSIZE,
V2P(shm_table.shm_pages[j].frame), PTE_W|PTE_U);
<         shm_table.shm_pages[j].refcnt++;
<     }
<
<     current_process->sz = sz + PGSIZE;//increase the number of pages
<     *pointer = (char *)sz;//update the pointer
<
<     release(&(shm_table.lock));
<
<     return id_existence; //added to remove compiler warning -- you should
decide what to return
---
> return 0; //added to remove compiler warning -- you should decide what to
return
77,78d43
<     int resultRefcnt;
<     acquire(&(shm_table.lock));
80d44
<     int i;
82,96d45
<     for(i = 0 ; i < 64 ; i++) { //there are 64 pages
<         if (shm_table.shm_pages[i].id == id) {//segment id exist
<             if(shm_table.shm_pages[i].refcnt == 1){
<                 shm_table.shm_pages[i].id = 0;
<                 shm_table.shm_pages[i].frame = 0;
<                 shm_table.shm_pages[i].refcnt = 0;
<                 break;
<             }
<             if(shm_table.shm_pages[i].refcnt > 1){
<                 shm_table.shm_pages[i].refcnt--;
<             }
<         }
<     }
<     resultRefcnt = shm_table.shm_pages[id].refcnt;
<     release(&(shm_table.lock));
99c48
<     return resultRefcnt; //added to remove compiler warning -- you should
decide what to return
```

```
---
> return 0; //added to remove compiler warning -- you should decide what to
return
```

shm_open function:

We loop through shm_table. In the loop we save the first empty page number, and we check if there is an element with the same id that is given to function. If there is a matching id, we map vp of the current process to the same physical address that is pointed by frame of the element of the shm_pages with the same id, and increase that element's reference counter. When we exit the loop we check the flag for id_existance, if it is 0 that means we did not encounter any physical page with the same id. So, we allocate a physical memory for the first empty shm_pages's element, assign the id given to us to its id, and set its reference counter to 1. Then we map the physical memory address to the current process. After we are done with all we update current_process's sz to sz + PGSIZE, and we update the pointer for sz. And we return id_existance to report if we allocated a new physical mem or we updated an already existing one.

Shm_close function:

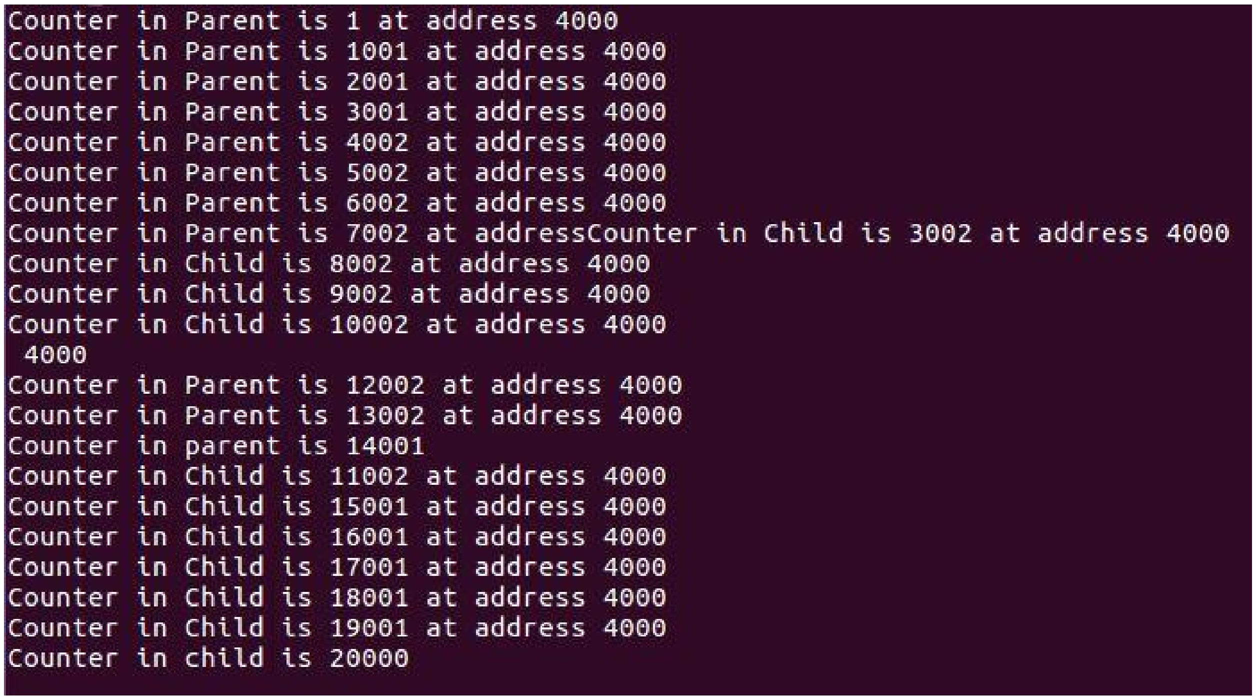
We loop through the shm_table to find matching the element with matching id, if we find the match we check if there is another process using it by checking its reference count. If the reference count is more than 1, we lower the reference count only. If the reference count is equal to 1, that means the process that is calling for shm_close is the only one that is using that page, then we close that page by setting its id, frame and reference to 0.

- **Screenshots about how you run the related program(s) and results**

shm_cnt.c

```c
#include "types.h"
#include "stat.h"
#include "user.h"
#include "uspinlock.h"

struct shm_cnt {
    struct uspinlock lock;
    int cnt;
 };

int main(int argc, char *argv[])
{
int pid;
int i=0;
struct shm_cnt *counter;
 pid=fork();
 sleep(1);

//shm_open: first process will create the page, the second will just attach to the same page
//we get the virtual address of the page returned into counter
//which we can now use but will be shared between the two processes

shm_open(1,(char **)&counter);

//  printf(1,"%s returned successfully from shm_open with counter %x\n", pid? "Child": "Parent", counter);
 for(i = 0; i < 10000; i++)
    {
     uacquire(&(counter->lock));
     counter->cnt++;
     urelease(&(counter->lock));

//print something because we are curious and to give a chance to switch process
     if(i%1000 == 0)
       printf(1,"Counter in %s is %d at address %x\n",pid? "Parent" : "Child", counter->cnt, counter);
}

 if(pid)
     {
       printf(1,"Counter in parent is %d\n",counter->cnt);
    wait();
    } else
    printf(1,"Counter in child is %d\n\n",counter->cnt);

//shm_close: first process will just detach, next one will free up the shm_table entry (but for now not the page)
  shm_close(1);
  exit();
  return 0;
}
```

shm_cnt.c output



```
Counter in Parent is 1 at address 4000
Counter in Parent is 1001 at address 4000
Counter in Parent is 2001 at address 4000
Counter in Parent is 3001 at address 4000
Counter in Parent is 4002 at address 4000
Counter in Parent is 5002 at address 4000
Counter in Parent is 6002 at address 4000
Counter in Parent is 7002 at addressCounter in Child is 3002 at address 4000
Counter in Child is 8002 at address 4000
Counter in Child is 9002 at address 4000
Counter in Child is 10002 at address 4000
 4000
Counter in Parent is 12002 at address 4000
Counter in Parent is 13002 at address 4000
Counter in parent is 14001
Counter in Child is 11002 at address 4000
Counter in Child is 15001 at address 4000
Counter in Child is 16001 at address 4000
Counter in Child is 17001 at address 4000
Counter in Child is 18001 at address 4000
Counter in Child is 19001 at address 4000
Counter in child is 20000
```

Analyze the result for shm_cnt.c

Because we are using spinlock to update the counter in shared memory, we do not encounter a race condition and are able to get 20000 at the end. Which also shows that our parent and child processes were able to share memory with shm_open.