Chris Hung Huynh, SID: 862024281

Edwin Leon, SID: 862132870

Lab 4 Report

In this lab we were asked to implement shared memory by creating two system calls called shm_open and shm_closed. The system call shm_open would see if the segment id already exists. If the id does not exist then it would allocate a page and map it. However, if the id does exist then the reference counter would be increased and it would map the segment in between the virtual and physical address. In order to implementation of shm_open is as follows:

1. First we defined a "struct proc curproc", a "uint sz" (sz of the curproc), an "int i", and an "int match". The "int i" would be used for the for-loops, while the "int match" would be used to determine whether there was a match of id's.
2. Next, we grabbed the lock for the shm_table so that no other process can access the table and prevent losing updates.
3. Then, we created a for-loop that went from 0 to 63 because the struct defines that there are 64 pages of shared memory.
    a. Inside the for-loop we checked if "shm_table.shm_pages[i].id == id". If it is true then "match" would be set to 1 and then we map the page using mappages. After mapping the page we increment the reference counter "shm_table.shm_pages[i].refcnt++" and we "break" from the for-loop. If "shm_table.shm_pages[i].id" does not equal the id then we do nothing and keep looping until a match is found or "i" not less than 64.
4. After the for-loop, we check if "match == 0".
    a. If it is false then we move on.
    b. If it is true then we create a for-loop that goes from 0 to 63. Inside the for-loop it checks if "shm_table.shm_pages[i].id == 0" which means it is an empty entry in the table. If this is true then "shm_table.shm_pages[i].id = id" and break from the for-loop. If it is not true we keep traversing through the table until the statement is true or "i" is no longer less than 64.

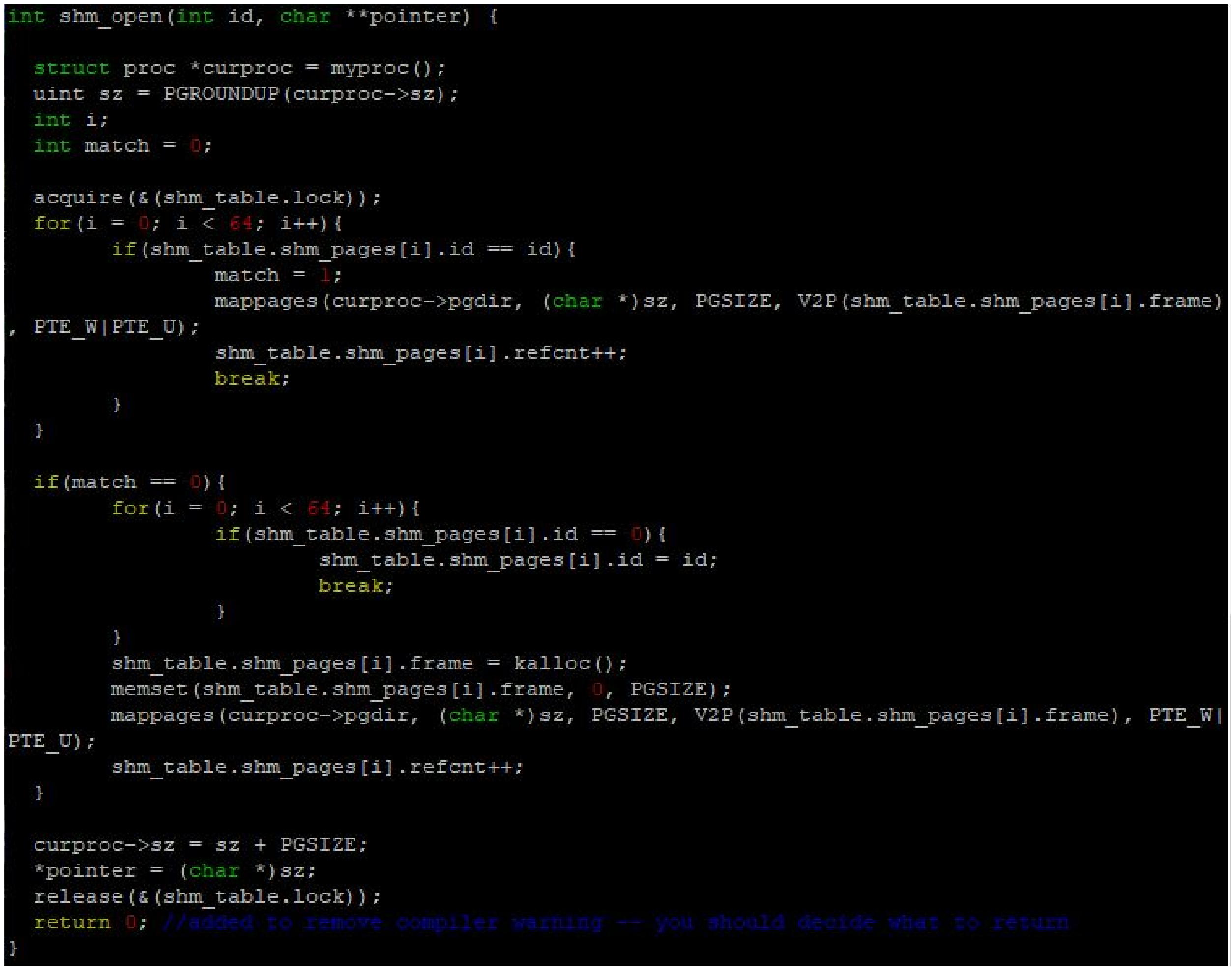
c. After the for-loop we use “kalloc()” to allocate a page and we store the address in “shm_table.shm_pages[i].frame”.

d. Once the page is allocated, we use memset to fill the memory starting from the address stored in “shm_table.shm_pages[i].frame”.

e. After, we map the page and increment the reference counter “shm_table.shm_pages[i].refcnt++”.

5. Next, we update sz and we return the pointer to the virtual address *pointer = (char *)sz”

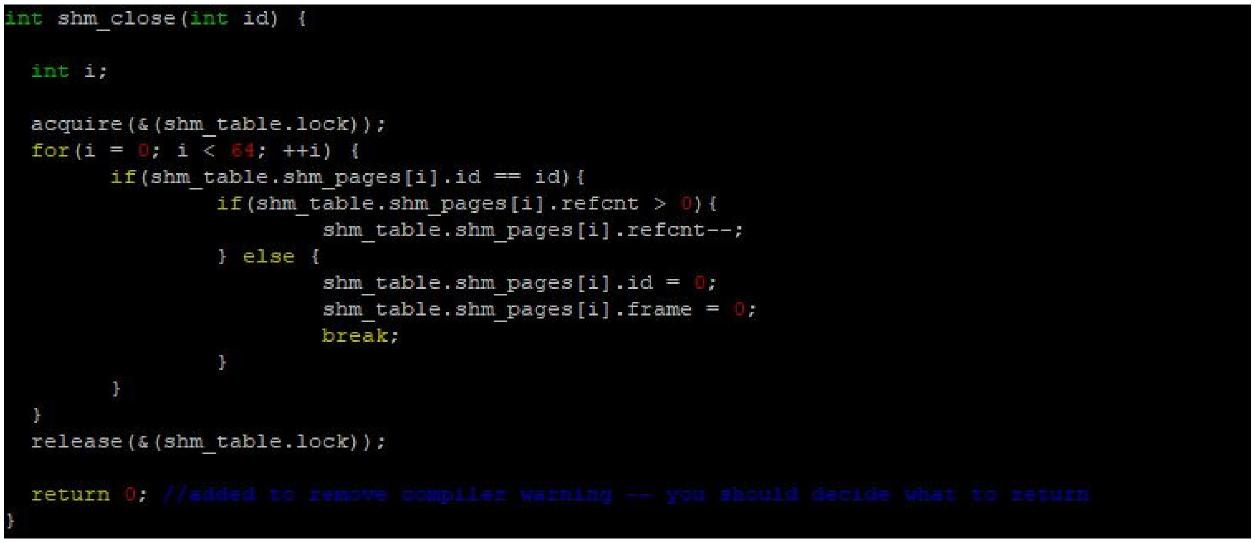
6. Finally we release the lock for the shm_table.



```
int shm_open(int id, char **pointer) {

  struct proc *curproc = myproc();
  uint sz = PGROUNDUP(curproc->sz);
  int i;
  int match = 0;

  acquire(&(shm_table.lock));
  for(i = 0; i < 64; i++){
        if(shm_table.shm_pages[i].id == id){
                match = 1;
                mappages(curproc->pgdir, (char *)sz, PGSIZE, V2P(shm_table.shm_pages[i].frame)
, PTE_W|PTE_U);
                shm_table.shm_pages[i].refcnt++;
                break;
        }
  }

  if(match == 0){
        for(i = 0; i < 64; i++){
                if(shm_table.shm_pages[i].id == 0){
                        shm_table.shm_pages[i].id = id;
                        break;
                }
        }
        shm_table.shm_pages[i].frame = kalloc();
        memset(shm_table.shm_pages[i].frame, 0, PGSIZE);
        mappages(curproc->pgdir, (char *)sz, PGSIZE, V2P(shm_table.shm_pages[i].frame), PTE_W|
PTE_U);
        shm_table.shm_pages[i].refcnt++;
  }

  curproc->sz = sz + PGSIZE;
  *pointer = (char *)sz;
  release(&(shm_table.lock));
  return 0; //added to remove compiler warning -- you should decide what to return
}
```

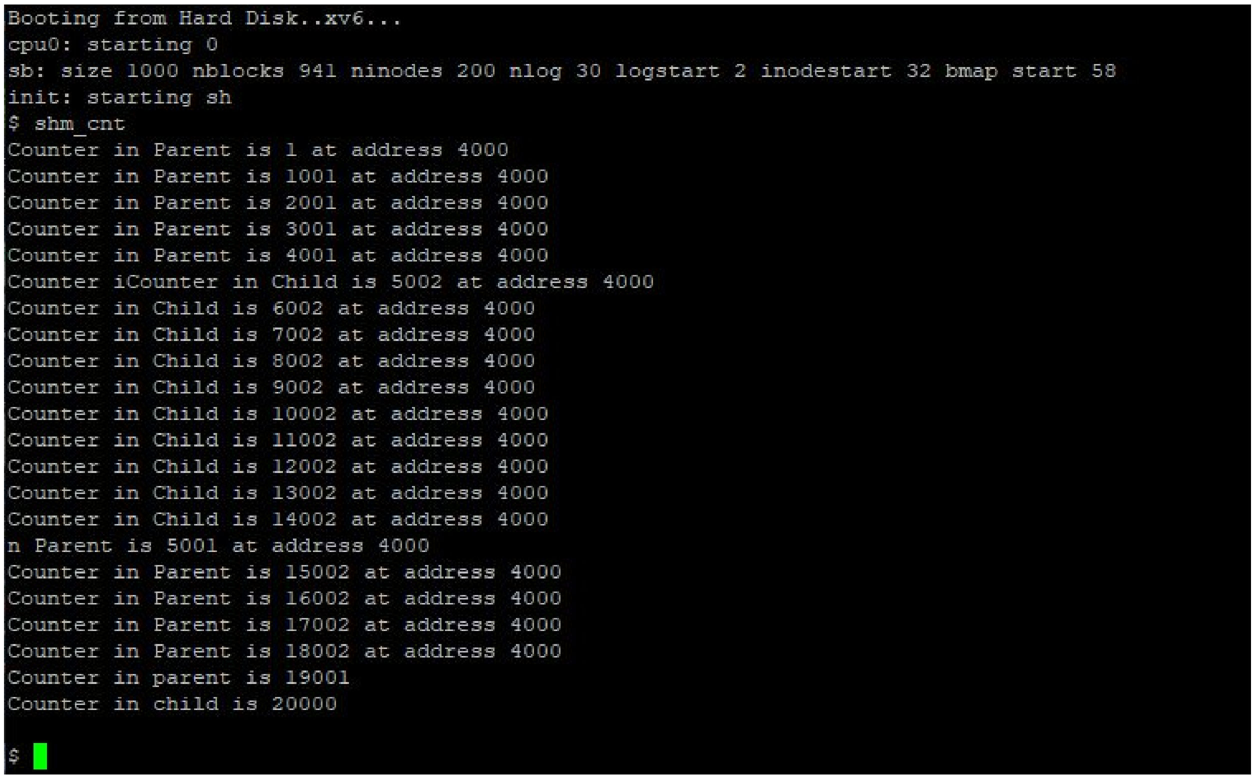
Next, we had to implement shm_close, which function was that it had to look for shared memory segments in the shm_table. If it found a segment then it would decrease the reference counter (if it was possible). However, if the reference counter could not be decreased then the shm_table would be cleared. To implement this we did as follows:

1. First we declare a variable "int i" that would be used to traverse through the for-loop.
2. Next, we grabbed the lock for the shm_table so that no other process can access the table and prevent losing updates.
3. We then created a for-loop that could go from 0 to 63.
    a. Inside the for-loop we check if "shm_table.shm_pages[i].id == id". If this is false then we just keep iterating in the for-loop until the statement is true or "i" is no longer less than 64.
    b. If it is true then
        i. It first checks if the reference counter is greater than 0. If it is greater the zero than we decrease the reference counter by 1 and we keep iterating through the for-loop.
        ii. If it is false then we clear the shm_table by setting the page id and frame to 0 and we break from the for-loop.
4. Finally the lock for the shm_table is released.



```
int shm_close(int id) {

  int i;

  acquire(&(shm_table.lock));
  for(i = 0; i < 64; ++i) {
        if(shm_table.shm_pages[i].id == id){
                if(shm_table.shm_pages[i].refcnt > 0){
                        shm_table.shm_pages[i].refcnt--;
                } else {
                        shm_table.shm_pages[i].id = 0;
                        shm_table.shm_pages[i].frame = 0;
                        break;
                }
        }
  }
  release(&(shm_table.lock));

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

Results:



```
Booting from Hard Disk..xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ shm_cnt
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 4001 at address 4000
Counter iCounter in Child is 5002 at address 4000
Counter in Child is 6002 at address 4000
Counter in Child is 7002 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
Counter in Child is 11002 at address 4000
Counter in Child is 12002 at address 4000
Counter in Child is 13002 at address 4000
Counter in Child is 14002 at address 4000
n Parent is 5001 at address 4000
Counter in Parent is 15002 at address 4000
Counter in Parent is 16002 at address 4000
Counter in Parent is 17002 at address 4000
Counter in Parent is 18002 at address 4000
Counter in parent is 19001
Counter in child is 20000

$
```