Anthony Mejia - SID: **861152139**
Christian Campos - SID: **862080812**

Lab 4: Advanced Memory management (shared memory)

Implemented Shared Memory:

For this assignment, we are given a starter code and we only need to modify the two system calls , shm_open() and shm_close() in shm.c.
We need to understand how shm_table data structure first.

Shm_table struct data structure:
The shared memory table keeps track of up to 64 pages of shared memory. Each page has: an id, a pointer to the physical frame, and a reference count.

shm_open():
We are iterating through shm_table by a given id if it already exists or not.
Case 1:
If this id exists , then we take the physical address of this page and map it to our virtual address space. Also, we increase the reference counter of the given entry.
Case 2:
If id doesn't exist in the shm_table, then we take the first empty entry and initialize it to a given id. Also, we allocate memory for this page and map it to our virtual address space and increase the reference counter.

shm_close():
We are iterating through shm_table for a given id. We decrease the reference counter for that entry. If the counter is already 0 , then we clean the entry.

For both cases, we increase the size of allocated memory for one page and update the pointer to the page.

The following screenshots below are our modification of the code we made in lab 4

shm.c:

```
int shm_open(int id, char **pointer) {
  //you write this
  struct proc *curproc = myproc();

  uint sz = PGROUNDUP(curproc->sz);
  int i;
  int j = 64;
  int k = 0;
  acquire(&(shm_table.lock));
  for(i = 0; i < 64; i++){
      if(j == 64 && shm_table.shm_pages[i].id == 0){
          j = i;
        }
    if(shm_table.shm_pages[i].id == id){
          k = 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 (k == 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(curproc->pgdir, (char *)sz, PGSIZE, V2P(shm_table.shm_pages[j].frame), PTE_W|PTE_U);
    shm_table.shm_pages[j].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
}
```

```
int shm_close(int id) {
  //you write this too!
  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--;
          }
          if(shm_table.shm_pages[i].refcnt == 0){
            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
}
```

To run lab 4 file.
First , use make clean qemu-nox inside the xv6 directory.
Then, enter shm_cnt to run lab 4.

Test Output:

```
$ shm_cnt
Counter in ParentCounter in Child is 2 at address 4000
Counter in Child is 1002 at address 4000
Counter in Child is 2002 at is 1 at address 4000
Counter address 4000
Counter in Child is 4002 at address 4000
Counter 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  in Parent is 3002 at address 4000
Counter in Parent is 9002 at a8ddress 4000
Counter in Parent is 10002 at address 4000
C002 at address 4000
Counter in Child is 12002 at address 4000
Counter in Child is 13002 at address 4000
Counter in Parent is 11002 at address 4000
Counter iounter in child is 14001

n Parent is 15001 at address 4000
Counter in Parent is 16001 at address 4000
Counter in Parent is 17001 at address 4000
Counter in Parent is 18001 at address 4000
Counter in Parent is 19001 at address 4000
Counter in parent is 20000
$
```