We have to fill in shm_open, first we have to look to see if the id we are opening already exists inside the table. Before we check for id, we acquire the lock first

```
acquire(&(shm_table.lock));
for(int i = 0; i < 64; ++i)
{
  if(shm_table.shm_pages[i].id == id)
  {
```

If we find it, it means another process has done this already, so now we use mappages to map the physical address of the page in the table to a page in the virtual address space.

```
if(mappages(myproc()->pgdir, (void *)PGROUNDUP(myproc()->sz), PGSIZE,
V2P(shm_table.shm_pages[i].frame), PTE_W|PTE_U) == 0)
{
```

Pgdir is the page directory pointer
Next is free virtual address that we attach to our page
PGSIZE is the size we are mapping
v2p() line is grabbing the physical address which is the frame pointer in the shm_table
Then we pass in the permissions

Next we increment refcnt and return pointer to virtual address to our passed in pointer variable

```
  shm_table.shm_pages[i].refcnt += 1;
  *pointer = (char *)PGROUNDUP(myproc()->sz);
  myproc()->sz += PGSIZE;
  release(&(shm_table.lock));
  return 0;
`
```

We release the lock and then return since we are finished. We dont need to do anything else in this case. Now we move onto case 2

We now have to look for a free address, so we start looking through the table again, for the first id that equals 0. 0 would mean that it is free.

```
//we look for id first, and then we look for first available
for(int j = 0; j < 64; ++j)
{
  if(shm_table.shm_pages[j].id == 0)
  {
```

We then set this empty entry's id to the id we are passing in. Then we kalloc a page, which allocates a page and then returns a pointer to it, and we store that in the frame pointer. We use memset after this to initialize that space we just allocated to all 0. We then set refcnt to 1.

We then have to map this new page like in case 1. We follow the steps in case 1 after mapping and release the lock and return.

```
shm_table.shm_pages[j].id = id;
shm_table.shm_pages[j].frame = kalloc();
memset(shm_table.shm_pages[j].frame, 0, PGSIZE);
shm_table.shm_pages[j].refcnt = 1;

if(mappages(myproc()->pgdir, (void *)PGROUNDUP(myproc()->sz), PGSIZE,
V2P(shm_table.shm_pages[j].frame), PTE_W|PTE_U) == 0)
{
  *pointer = (char *)PGROUNDUP(myproc()->sz);
  myproc()->sz += PGSIZE;
  release(&(shm_table.lock));
  return 0;
}
else
{
  release(&(shm_table.lock));
  return -1;

}

}
```

We then have to implement shm_close, which is closing the page with the id we are passing in. Once we find it, we decrement the refcnt and now we have two cases.
If the refcnt is still greater than 0, then we do nothing because its still being shared and used. But if it hits zero, we have to clear the data in that entry in the table. So we set all the values to zero, we dont need to free up the page.

```
int shm_close(int id) {
//you write this too!
acquire(&(shm_table.lock));

for(int i = 0; i < 64; ++i)
{
  if(shm_table.shm_pages[i].id == id)
  {
    shm_table.shm_pages[i].refcnt--;
    if(shm_table.shm_pages[i].refcnt > 0)
    {
    }
    else
    {
      shm_table.shm_pages[i].id = 0;
      shm_table.shm_pages[i].frame = 0;
      shm_table.shm_pages[i].refcnt = 0;
    }
    break;
  }
}
release(&(shm_table.lock));
return 0; //added to remove compiler warning -- you should decide what to return
}
```

Example output

```
$ shm_cnt
Counter in Parent is 1 at address 4000
Counter in Parent is 1001 at address 4000
Counter in ParCounter in Child is 2002 at address 4000
Counter in Child is 3002 at address 4000
Counter in Child is 4002 at address 4000
Counter in ent is 2001 at address 4000
Counter in Parent is 6002 at address 4000
Counter in Parent is 7002 at address 4000
Counter in Parent is 8002 at address 4000
Counter in Parent is 9002 at address 4000
Counter in Parent is 10002 at address 4000
Counter in Parent is 11002 at address 4000
Counter in Parent is 12002 at address 4000
Counter in parent is 13001
Child is 5002 at address 4000
Counter in Child is 14001 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
```