Lab 4: Advanced Memory management (shared memory)

Handed out March 2, 2020
Due March 16, 2020

Lab 4:

Objectives

- Implement Shared Memory

Preliminaries

For this assignment we will use some starter code which is needed for Lab 4 (the same base used for Lab 3). You can get the starter code from the lab4 repository on github.

Implement shared memory

In this assignment you are implementing support to enable two processes to share a memory page. This is implemented by having an entry in both process page tables point to the same physical page.

Start by looking at the user program shm_cnt.c. In this program we fork a process, then both processes open a shared memory segment with the same id using:

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

For this system call, the first parameter gives an id for the shared memory segment, and the pointer is used to return a pointer to the shared page. By having this pointer be of type `shm_cnt`, we can access this struct off of this pointer and we would be accessing the shared memory page.

The code then proceeds to have both processes go through a loop repeatedly incrementing the counter in the shared page (acquiring a user level spin lock to make sure we dont lose updates; test your program without the lock and see if it makes a difference). The uspinlock is implemented in the starter code in uspinlock.c and uspinlock.h -- take a look.

At the end, each process prints the value of the counter, closes the shared memory segment and exits using `shm_close(1)`. One of them should have a value of 20000 reflecting updates from both the processes. Check your code without the spinlock to see if you lose updates.

Your task is to implement shm_open and shm_close. They are already added as system calls; you should write your code in shm.c

shm_open looks through the shm_table to see if this segment id already exists. If it doesn't then it needs to allocate a page and map it, and store this information in the shm_table. Dont forget to grab the lock while you are working with the shm_table (why?). If the segment already exists, increase the refence count, and use mappages to add the mapping between the virtual address and the physical address. In either case, return the virtual address through the second parameter of the system call.

shm_close is simpler: it looks for the shared memory segment in shm_table. If it finds it it decrements the reference count. If it reaches zero, then it clears the shm_table. You do not need to free up the page since it is still mapped in the page table. Ok to leave it that way.

Hints

Check out the Lab 4 survival guide for more detailed help.