CS/COE 1550 LAB 1

PART 1: SETTING UP THE XV6 DEVELOPMENT ENVIRONMENT.

We will use the Linux cluster provided by the CS Department.

1. First, whenever the amount of AFS free space in your account is less than 500MB, you can carry out the following process to increase. your disk quota (thanks to our Tech Support Team!):
   a. Log in to https://my.pitt.edu
   b. Click on "Profile" at the top of the screen
   c. Click on "Manage Your Account"
   d. Click on "Manage Email Quota"
   e. Click on "Increase My UNIX Quota"
2. Log in to linux.cs.pitt.edu using your Pitt account. From a UNIX box, you can type: **ssh ksm73@linux.cs.pitt.edu** assuming ksm73 is your Pitt ID. From Windows, you may use PuTTY.
3. Under Linux, run the following command to download the Xv6 source code: **git clone git://github.com/mit-pdos/xv6-public.git**
4. **cd xv6-public** then type **make qemu-nox** to compile and run XV6. To exit, press CTRL+A then X.

PART 2: ADDING A SYSTEM CALL TO XV6[1]

You'll then add a new system call called getcount to xv6, which, when passed a valid system call number (listed in the file "syscall.h") as an argument, will return the number of times the referenced system call was invoked by the calling process.

For instance, consider the following test program (getcount.c):

```
#include "types.h"
#include "user.h"
#include "syscall.h"
int  main(int argc, char *argv[])
{
   printf(1, "initial fork count %d\n", getcount(SYS_fork));
   if (fork() == 0) {
      printf(1, "child fork count %d\n", getcount(SYS_fork));
      printf(1, "child write count %d\n", getcount(SYS_write));
   } else {
      wait();
      printf(1, "parent fork count %d\n", getcount(SYS_fork));
      printf(1, "parent write count %d\n", getcount(SYS_write));
   }
   printf(1, "wait count %d\n", getcount(SYS_wait));
   exit();
}
```

will produce the following output (note that each character is output with a separate call to write/printf in xv6):

```
initial fork count 0
child fork count 0
child write count 19
wait count 0
parent fork count 1
parent write count 41
wait count 1
```

Hints

You will need to modify several files for this exercise, though the total number of lines of code you'll be adding is quite small. At a minimum, you'll need to alter syscall.h, syscall.c, user.h, and usys.S to implement your new system call. It may be helpful to trace how some other system call is implemented (e.g., uptime) for clues.

---

[1] Adapted from http://moss.cs.iit.edu/cs450/assign01-xv6-syscall.html

You will likely also need to update struct proc, located in proc.h, to add a syscall-count tracking data structure for each process. To re-initialize your data structure when a process terminates, you may want to look into the functions in proc.c.

Chapter 3 of the xv6 book (in pdf) contains details on traps and system calls (though most of the low level details won't be necessary for you to complete this exercise).

Testing

To test your implementation, you'll run the getcount executable (when booted into xv6), which is based on the program above. Because the program depends on your implementation, it isn't compiled into xv6 by default. When you're ready to test, you should add getcount to UPROGS declaration in the Makefile.

Note that while the getcount *example* only prints out counts for three different system calls, your implementation should support all the system calls listed in syscall.h. It is a good exercise to add tests to the test program, located in getcount.c, for other system calls.