Assignment 2

Pstree

Important Notes

1. This is an **individual** project. Do your own work!
2. The project is based on the Linux operating system.
3. Make sure your program works on linprog.cs.fsu.edu because that is where it will be graded.

Acknowledgment

This project is designed by Prof Zhi Wang in FSU Tallahassee which being derived from 600.318/418: Operating Systems at John Hopkins University.

Overview

In this assignment, you will get to know Xv6 (git://github.com/mit-pdos/xv6-public.git), a simple Unix-like teaching operating system from MIT. You'll do three things in this assignment:

- You'll get xv6 to run on a system emulator, QEMU. This is a pretty quick and painless process.
- You'll trace the **getpid()** system call of xv6 and document what goes on at each step.
- You'll implement the **pstree** program which displays the list of processes currently in the system in a tree structure; for this to work, you'll have to introduce a new system call.

Words of Wisdom: First, please start early! Second, please make minimal changes to xv6; you do not want to make it hard for us to grade!

Part 1: Xv6 on QEMU (20%)

The first part is easy. You need to clone, compile it and make a disk image, and get that disk image to run on QEMU. Cloning the xv6 source as being directed in course module 5. Extract the sources, then take a quick look around.

On linprog.cs.fsu.edu, compiling xv6 should work fine by just typing **make** in the

xv6 directory. You'll see a lot of lines fly by, and eventually you'll have an image file **xv6.img**. Running xv6 on QEMU should work fine by typing **make qemu-nox** in the xv6 directory. And there you have it. Try running *ls* and *mkdir*. You can even run *sh* to get a new shell process (use CTRL-D to leave it as there's no **exit** command). To quit from QEMU to linprog (or your local ubuntu), type **ctrl-a x**. Document this process.

Part 2: The Life of getpid (30%)

The second part is a bit harder, but I will provide additional help trace a system call **close** using the GDB.

Using the same debugging technique, you need to describe how the **getpid()** system call is implemented. Assume that a user program executes the line **getpid()**;. You need to document in the submission exactly **which lines in which source files of xv6 are executed** as a result of this system call. Please don't skip anything. Try to explain each coherent chunk of lines, i.e. tell us why that chunk runs and what purpose it serves. **Your description must start in the user-space with the fictitious line above, go to the kernel-space, and return back to the user-space. Please make your report in no more than 200 words.**

Part 3: Pstree (50%)

Type **man pstree** in linprog.cs.fsu.edu and read the documentation. Read it to understand what *pstree* does. Now relax, there is no need to implement all those command line options: we'll simply print one fixed set of interesting values about each process. But you should become aware of all the stuff a real Unix system keeps track of, xv6 is a lot simpler than that.

If you examine the source code for xv6 you'll find the file **proc.h** somewhere. In it, you find what we are looking for: struct proc. When xv6 is running, all that information is around for each process, **in the kernel**! Since **pstree** is a user space program, it cannot access the process table in the kernel. In modern Unixes, for example in Linux, the **/proc** file system provides all the information to implement **pstree**, but we don't want to add a whole new file system to the kernel. Instead we'll have to add the next best thing, a new system call. But before we get to that, let's agree on the output of **pstree**. For each process, **pstree** prints the process name and the process id. It prints all the processes in a tree structure, using indents (three whitespaces) to show the parent/child relation. Here is an example (this is just an example, your output will be different in details.):

```
init[1]
    sh[4]
      pstree[7]
    sh[5]
      usertests[6]
      cat[8]
```

In this example, **init** is the root of the process tree (i.e., it is the first process ever created by the kernel.) Its process id is 0. It has two child processes, **sh** (pid=4) and **sh** (pid=5). **pstree**, **cat**, and **usertests** are its grandchildren. This essentially is a depth-first traverse of the process tree.

Now, you will add just one system call and the necessary implementation in the kernel. In C, the system call you need to add to xv6 has the following interface:

```
int getprocs(int max, struct uproc table[]);
```

struct uproc is defined as (add it to a new file **uproc.h**):

```
struct uproc {
    int pid;
    int ppid;
    char name[16];
};
```

Your **pstree** program calls **getprocs** with an array of **struct uproc** objects and sets max to the size of that array (in the unit of **struct uproc** objects). For example, your program can **malloc 64 struct uproc** objects (assuming the returned pointer is saved in **pcs**) and call **getprocs** with **getprocs(64, pcs);**. The kernel stores up to **max** entries of the process information into your array, starting at the first slot of the array and filling it consecutively. The kernel returns the *actual number of processes* in existence at that point in time, or -1 if there was an error. After receiving the process information from the kernel, your **pstree** in the user-space formats it into the tree structure as shown in the example.
The problem with this system call is that if there are more processes than you have space for, you will miss some. But then if you use the returned integer to allocate a new **struct uproc** array and call **getprocs** again, some processes may have died or (worse!) new ones may have been created in the meantime. So you'll have to be lucky to get the perfect listing for a busy system. In this assignment, let's assume the system has less than 64 processes.

Delverables

Place two pdf format documents for part1 and part2 in xv6 project folder for pstree. You must ensure that this xv6 project folder includes all files necessary for building **pstree**. Then, submit modified xv6 project folder for pstree project as a gzip compressed tarball. Make sure clean all object files by "make clean" before creating this tarball. The name of your attachment should be

cop4610-project2-yourname.tar.gz

with yourname is replaced by your CS account name in linprog.cs.fsu.edu.

Your submission will be graded by compiling and running it. We will check the presence of the **pstree** command and that it returns the correct results.

- Please make sure your source code can compile. Absolutely no credit for part 3 if it does not compile.
- Please don't include the binary files. Do a **make clean** before submission. You'll make grading harder for us if you do.
- Please don't leave out any files! You'll make grading harder for us if you do.
- Please don't modify any files you don't need to! You'll make grading harder for us if you do.
- Please don't send us the meta-information from your revision control system! You'll make grading harder for us.