System Administration Decal
Intermediate Lab #2
Due: February 22, 2010

Introduction

This week we covered the layout of a Unix filesystem and learned about the various types of files that are commonly seen on Unix: regular, device, directory, and FIFO (named pipe.) We're going to build on these ideas and learn a little more about all of these things.

Things to remember:

- Send your responses to mgasidlo+decal@ocf.berkeley.edu; text format is preferred.
- Remember to include your cs198-XX username with your submission.
- If you don't have a Linux environment at home, SSH to *icom1.eecs.berkeley.edu* using your cs198-XX username.
- Finally, remember that the slides, man pages, and Google are your friends.

I. Filesystems upon Filesystems

One thing that is special about Unix is that the filesystem is *device agnostic.* This means that you can browse through the filesystem, and /home could be on one hard drive while /boot is on another. Contrast this to Windows where different drives have completely isolated filesystems, differentiated by drive letters. Here, we'll take a look at this concept of *mounting* other filesystems on top of your *root filesystem* in Unix.

First, log in to icom1, as a normal user.
1. Run *df –h* at the command line. What does this command do? What can you say about where your home directory is stored? Is it on the machine you're logged into? If not, where is it?
2. How can you tell whether a particular mount is a network share or available from a hard drive connected to the computer? Feel free to guess.
3. Use the output of *df –h* to figure out on what physical (or network) filesystem each file resides, on icom1. Bonus points (as if these labs are graded for quality, right?) if you can explain what the files in a-c represent. (They all actually exist on icom1.)
    a. /boot/vmlinuz-2.6.18-164.11.1.e15
    b. /var/lib/rpm
    c. /usr/bin/Xorg
    d. Your home directory

II. The /proc Filesystem

A notable filesystem amongst those that are mounted on top of your root filesystem is the /proc filesystem. This is a special filesystem that is provided by the

kernel that doesn't actually use any disk space – it is all in memory. Its purpose is to provide information about processes and allow you to twiddle some internal knobs within the kernel.

The proc(5) manpage on a Linux box can assist you throughout this problem.
1. Use the *ps* command to find a process that you own. Try to look it up in /proc and describe what it tells you about that process.
2. Use the *ps* command with an argument that lets you see other people's processes (what is it?) Choose a process that is not yours and try to examine it in /proc. What happens?
3. Use the output of *ls –l* in /proc to explain your answer for #2 more clearly.
   (Hint: The two username-looking entries in the output of *ls –l* represent who *owns* the file, and which *group* co-owns the file, respectively. To the left of that is the permission listing for the file, representing whether users, groups, or anyone else can **r**ead, **w**rite or e**x**ecute the file. If this isn't clear, don't worry about it. We will go over it in detail next week.)

4. What is */proc/kcore*? What happens when you read it? Do you think it's a good idea for the behavior to be this way? Why or why not?
5. Check out */proc/net/dev*. Try to understand the table it provides – you may need to widen your terminal window to view it correctly. Find a way to use the *watch* command (look it up!) to provide a running update of its output.
6. */proc/self* is really special. What does it do? What feature of the filesystem does it use to accomplish its task? Again, use *ls –l* to examine it.

III. Device Files – The /dev Directory

Now we'll take a brief look at device files, which are all located in /dev. Have http://www.lanana.org/docs/device-list/devices-2.6+.txt open in a browser window for this exercise.
1. Look through /dev and find a way to redirect the output of a process to standard error using a file in /dev.
2. Check out the /dev/zero and /dev/full files. How are these similar to /dev/null? (It's *not* because "full" is only one letter away from "null".)
3. Use *df –h* to figure out the root partition for the machine you're on. Try *cat*ing it. Why can't you do it, and why is it good that you (as a normal user) can't?
4. Use *ls –l* to look through /dev. Why aren't there any filesizes present? What replaces them in the output of *ls*? (Hint: Research the *mknod* command.)