Observing Linux Behavior

CSCI411 Lab

Adapted by MA Doman from Linux Kernel Projects by Gary Nutt

Exercise Goal: You will learn several important characteristics of the Linux kernel, processes, memory and other resources. You will write a program to use the **/proc** virtual file system to inspect various kernel values that reflect the machines load average, process resource utilization and so on. After you have obtained the kernel status, you will prepare a report of the cumulative behavior that you observed.

Contents

Introduction

The Linux kernel is a collection of data structure instances (*kernel variables)* and functions. The collective kernel variables define the kernel's perspective of the state of the entire computer system. Each externally invoked function – a system call or an IRQ (interrupt request) – provides a prescribed service and causes the system state to be changed by having the kernel code changed its kernel variables. If you could inspect the kernel variables, then you could infer the state of the entire computer system.

Many variables make up the entire kernel state, including variables to represent each process, each open file, the memory, and the CPU. Kernel variables are no different than ordinary C program variables, other than they are kept in kernel space. They can be allocated on a stack (in kernel space) or in static memory so that they do not disappear when a stack frame is destroyed. Because the kernel is implemented as a monolithic module, many of the kernel variables are declared as global static variables. Some of the kernel variables are instances of built-in C data types, but most are instances of an extensible data type, a C STRUCT.

The /proc filesystem, mounted under the /proc directory, is a means to provide communication between the Linux® kernel and user space. The '/proc' file system doesn't contain 'real' files but contains runtime system information (e.g. system memory, devices mounted, hardware configuration, etc). This novel approach is referred to as a virtual filesystem (see Appendix B). A *virtual file system* approach provides an interface to kernel data structures in a form that looks like files and directory's on a file system. In the /proc filesystem, virtual files can be read from or written to as a means of communicating with entities in the kernel, but unlike regular files, the content of these virtual files is dynamically created. It's sometimes referred to as a process information pseudo-file system.

The /proc file system provides an easy mechanism for viewing and changing various system attributes. For this reason it can be regarded as a control and information center for the kernel. In fact, quite a lot of system utilities are simply calls to files in this directory. For example, 'lsmod' (Linux command to show the kernel modules currently loaded) is the same as 'cat /proc/modules' while 'lspci' (linux command to show all PCI busses in the system and the devices connected to them) is a synonym for 'cat /proc/pci'. By altering files located in this directory you can even read/change kernel parameters (using linux command sysctl) while the system is running.

Problem Statement

This lab exercise is to study some aspects of the organization and behavior of a Linux system by observing values stored in kernel variables.

Part A

Answer the following questions about the Linux machine that you will be using to do these exercises. Include the program/command or other method you used to obtain the information. For each, give a brief explanation the information provided.

1. Host name
2. What version of the Linux kernel is being used
3. Number of processing units on the machine
4. For each processor:

- The CPU Vendor
- The model

5. How much memory is configured into it? How much memory is currently available on it?
6. How long in seconds has the system been up (since last boot)?
7. How much of that time was spent idle, in seconds?
8. How much of the total CPU time has been spent executing in user mode? System mode? Idle?
9. How many context switches has the kernel performed?
10. How many context switches has your current bash process kernel performed? (hint got to the process id folder for this process and look at the status)
11. How many processes have been created since the system was booted across all CPUs?
12. A list of load averages
13. The number of disk read requests made on the system for any one disk block device

Part B

Write a default version of a program to report the behavior of the Linux kernel by inspecting the kernel state. The program should print the following values on the **monitor (or stdout device)**.

A. Host name
B. Number of processing units on the machine
    a. cat /proc/cpuinfo | grep "processor" | wc -l
C. For each processor:
    a. The CPU Vendor
    b. The model
D. Kernel version
E. Amount of time since the system was last booted. How long (in hours minutes and seconds) has it been since the system was last booted?
F. The amount of memory configured into this computer and the memory currently available

The output can look something like this:

Note: Some answers may look different on your output depending on the virtual file referenced.

A: Host hame : ubuntu

B: Number of processing units: 1

C: CPU(s) Type and model :

processor : 0
vendor_id : GenuineIntel
model : 92
model name : Intel(R) Pentium(R) CPU N4200 @ 1.10GHz

D: Linux Kernel Version: Linux version 4.4.0-101-generic (buildd@lcy01-amd64-006) (gcc version 5.4.0 20160609 (Ubuntu 5.4.0-6ubuntu1~16.04.5)) #124-Ubuntu SMP Fri Nov 10 18:29:59 UTC 2017

E: System time:

Time since last re-boot: 22627 seconds which is equivalent to 6 hours 17 minutes 7 seconds.
Time in idle is 682 seconds which is equivalent to 0 hours 11 minutes 22 seconds.

F: **Memory information**

MemTotal: 508192 kB
MemFree: 19532 kB

Attacking the Problem

Linux, Solaris. and other versions of UNIX provide a very useful mechanism for inspecting the kernel state, called the /proc file system. This is the key mechanism that you can use to do this exercise.

The /proc ***File System***

The /proc file system is an OS mechanism whose interface appears as a directory in the conventional UNIX file system (in the root directory). You can change to /proc just as you change to any other directory. For example,

```
bash$ cd /proc
```

makes /proc the current directory. Once you have made /proc the current directory, you can list its contents by using the Is command. The contents appear to be ordinary files and directories. However, a file in /proc or one of its subdirectories is actually a program that reads kernel variables and reports them as ASCII strings. Some of these routines read the kernel tables only when the pseudo file is opened, whereas others read the tables each time that the file is read.

Several directories as well as files are contained in /proc. The subdirectories with numeric names contain more pseudo files to read information about the process whose process ID is the same as the directory name.. The directory **self** contains process-specific information for the process that is using /proc. The exact contents of the /proc directory tree vary among different Linux versions, so you must experiment with the pseudo files to view the information provided. One site you might find helpful is:

Red Hat Enterprise Deployment Guide
https://www.centos.org/docs/5/html/5.1/Deployment_Guide/ch-proc.html

Files in /proc are read just like ordinary ASCII files. For example, when you type to the shell a command such as

cat /proc/version

You will get a message printed to **screen (stdout)** that resembles the following:

```
$ cat /proc/version
CYGWIN_NT-6.1 version 1.7.32(0.274/5/3) (corinna@calimero.vinschen.de) (gcc version
4.8.3 20140522 (Fedora Cygwin 4.8.3-5) (GCC) ) 2014-08-13 23:06
```

To read a /proc pseudo file's contents, you open the file and use the stream library routines such as **getline()** to read the file one line at a time. Then you can look for content matching the information you need. The exact files (and tables) read depend on the specific version of Linux you have. To find out exactly which file interfaces are available to you through **/proc** read the **proc man** page on the system.

You may have to look up the format for some files. For example:

```
The /proc/diskstats file displays the I/O statistics of block devices. Each
line contains the following 14 fields:
                1 - major number
                2 - minor mumber
                3 - device name
                4 - reads completed successfully
                5 - reads merged
                6 - sectors read
                7 - time spent reading (ms)
                8 - writes completed
                9 - writes merged
               10 - sectors written
               11 - time spent writing (ms)
               12 - I/Os currently in progress
               13 - time spent doing I/Os (ms)
               14 - weighted time spent doing I/Os (ms)
```

Organizing a Solution

For Part B. . read the appropriate file in **/proc** and print the data through the standard output (cout) file. The following code segment contains an example of how to read the **/proc/sys/kernel/hostname** file. You can use it as a prototype for solving the exercise by reading other pseudo files. This will require some exploration of the **/proc** and inspection of the various pseudo files as you investigate different directories.

```
#include <iostream>

…
// Open the file for CPU information
    fstream  thisProcFile;
```

```
    thisProcFile.open ("/proc/sys/kernel/hostname", ios::in);
// Read the file. Note this file only has the host name in it
    string line;
    getline(thisProcFile, line);
    cout << line << endl << endl;
    thisProcFile.close();
```

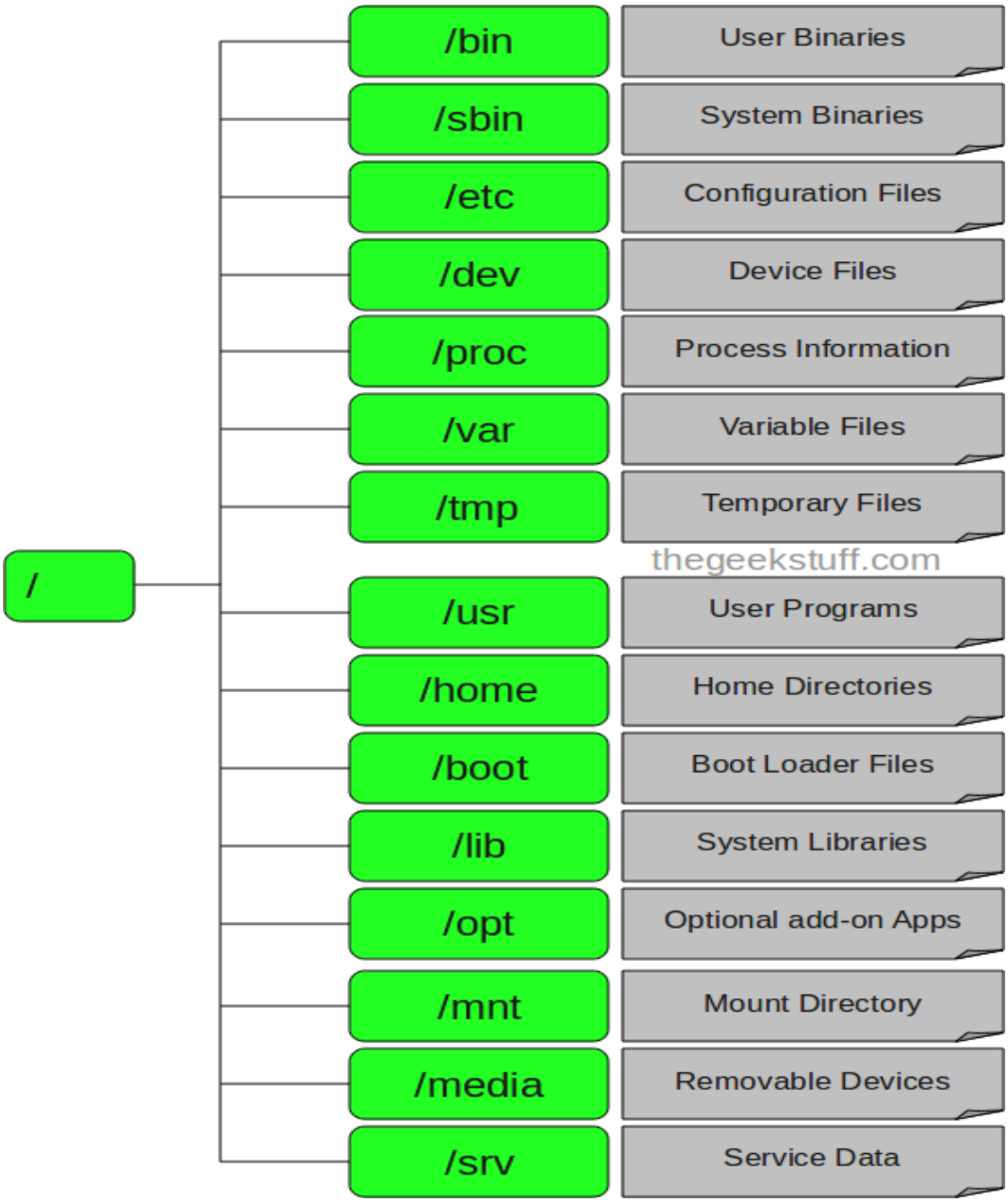
Now you are ready to do the work, that is, to start reading kernel variables by using various /proc files. Hint: The string object has a *find* method that looks for a match. The *find()* method returns the position of the first character of the first match. If no matches were found, the function returns string::npos. You can use that method to read through the file and look for the field you want.

Submission Requirements

- Submit your source code.
- Submit the completed lab report questions provided.

Appendix A: Linux Directory Structure Explained with Diagram

The kernel maintains a single hierarchical directory structure to organize all files in the system.(This contrasts with operating systems such as MS Windows where each disk device has its own directory hierarchy) . At the base of Linux's hierarchy is the *root directory*, named / (slash). All files and directors are children or further removed descendants of the root directory



1. / – Root

- Every single file and directory starts from the root directory.
- Only root user has write privilege under this directory.
- Please note that /root is root user's home directory, which is not same as /.

2. /bin – User Binaries

- Contains binary executables.
- Common linux commands you need to use in single-user modes are located under this directory.
- Commands used by all the users of the system are located here.
- For example: ps, ls, ping, grep, cp.

3. /sbin – System Binaries

- Just like /bin, /sbin also contains binary executables.
- But, the linux commands located under this directory are used typically by system aministrator, for system maintenance purpose.
- For example: iptables, reboot, fdisk, ifconfig, swapon

4. /etc – Configuration Files

- Contains configuration files required by all programs.
- This also contains startup and shutdown shell scripts used to start/stop individual programs.
- For example: /etc/resolv.conf, /etc/logrotate.conf

5. /dev – Device Files

- Contains device files.
- These include terminal devices, usb, or any device attached to the system.
- For example: /dev/tty1, /dev/usbmon0

6. /proc – Process Information

- Contains information about system process.
- This is a pseudo filesystem contains information about running process. For example: /proc/{pid} directory contains information about the process with that particular pid.
- This is a virtual filesystem with text information about system resources. For example: /proc/uptime

7. /var – Variable Files

- var stands for variable files.
- Content of the files that are expected to grow can be found under this directory.
- This includes — system log files (/var/log); packages and database files (/var/lib); emails (/var/mail); print queues (/var/spool); lock files (/var/lock); temp files needed across reboots (/var/tmp);

8. /tmp – Temporary Files

- Directory that contains temporary files created by system and users.
- Files under this directory are deleted when system is rebooted.

9. /usr – User Programs

- Contains binaries, libraries, documentation, and source-code for second level programs.
- /usr/bin contains binary files for user programs. If you can't find a user binary under /bin, look under /usr/bin. For example: at, awk, cc, less, scp
- /usr/sbin contains binary files for system administrators. If you can't find a system binary under /sbin, look under /usr/sbin. For example: atd, cron, sshd, useradd, userdel
- /usr/lib contains libraries for /usr/bin and /usr/sbin

- /usr/local contains users programs that you install from source. For example, when you install apache from source, it goes under /usr/local/apache2

10. /home – Home Directories

- Home directories for all users to store their personal files.
- For example: /home/john, /home/nikita

11. /boot – Boot Loader Files

- Contains boot loader related files.
- Kernel initrd, vmlinux, grub files are located under /boot
- For example: initrd.img-2.6.32-24-generic, vmlinuz-2.6.32-24-generic

12. /lib – System Libraries

- Contains library files that supports the binaries located under /bin and /sbin
- Library filenames are either ld* or lib*.so.*
- For example: ld-2.11.1.so, libncurses.so.5.7

13. /opt – Optional add-on Applications

- opt stands for optional.
- Contains add-on applications from individual vendors.
- add-on applications should be installed under either /opt/ or /opt/ sub-directory.

14. /mnt – Mount Directory

- Temporary mount directory where sysadmins can mount filesystems.

15. /media – Removable Media Devices

- Temporary mount directory for removable devices.
- For examples, /media/cdrom for CD-ROM; /media/floppy for floppy drives; /media/cdrecorder for CD writer

16. /srv – Service Data

- srv stands for service.
- Contains server specific services related data.
- For example, /srv/cvs contains CVS related data.
-

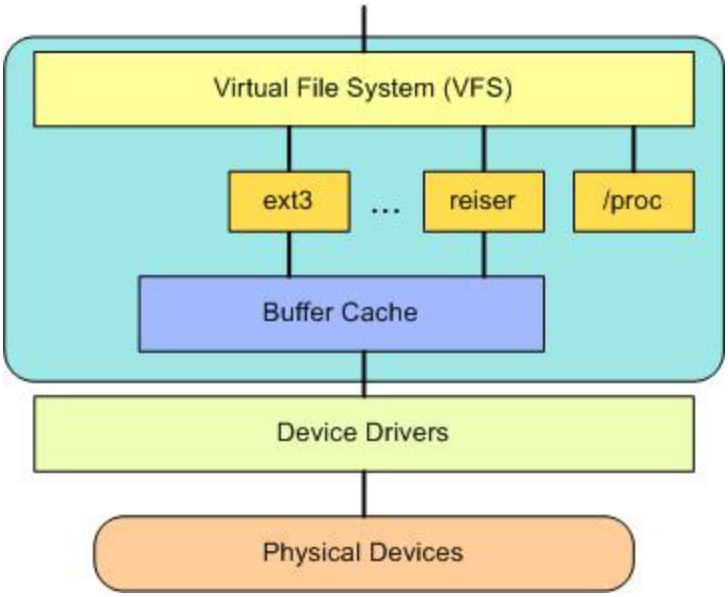
Appendix B: Virtual File System

From https://www.centos.org/docs/5/html/5.1/Deployment_Guide/s1-proc-virtual.html:

> Under Linux, all data are stored as files. Most users are familiar with the two primary types of files: text and binary. But the `/proc/` directory contains another type of file called a *virtual file*. It is for this reason that `/proc/` is often referred to as a *virtual file system*.
>
> These virtual files have unique qualities. Most of them are listed as zero bytes in size and yet when one is viewed, it can contain a large amount of information. In addition, most of the time and date settings on virtual files reflect the current time and date, indicative of the fact they are constantly updated.
>
> Virtual files such as `/proc/interrupts`, `/proc/meminfo`, `/proc/mounts`, and `/proc/partitions` provide an up-to-the-moment glimpse of the system's hardware. Others, like the `/proc/filesystems` file and the `/proc/sys/` directory provide system configuration information and interfaces.
>
> For organizational purposes, files containing information on a similar topic are grouped into virtual directories and sub-directories. For instance, `/proc/ide/` contains information for all physical IDE devices. Likewise, process directories contain information about each running process on the system.

The virtual file system (VFS) is an interesting aspect of the Linux kernel because it provides a common interface abstraction for file systems. The VFS provides a switching layer between the system call interface (SCI) and the file systems supported by the kernel (see Figure 4).

Figure 4. The VFS provides a switching fabric between users and file systems



At the top of the VFS is a common API abstraction of functions such as open, close, read, and write. At the bottom of the VFS are the file system abstractions that define how the upper-layer functions are implemented. These are plug-ins for the given file system (of which over 50 exist). You can find the file system sources in ./linux/fs.

Below the file system layer is the buffer cache, which provides a common set of functions to the file system layer (independent of any particular file system). This caching layer optimizes access to the physical devices by keeping data around for a short time (or speculatively read ahead so that the data is available when needed). Below the buffer cache are the device drivers, which implement the interface for the particular physical device.

Appendix C: Windows Operating System Internals

Microsoft Windows has a set of commands to explore their system internals. They are not downloaded on our campus computers. So, they will not be assigned in this course. You can explore them on your own computer. Start with the System Internals site:

https://docs.microsoft.com/en-us/sysinternals/

REFERENCES:

- http://www.thegeekstuff.com/2010/09/linux-file-system-structure/ Posted by: Abhijit Sandhan March 1, 2011 in Linux 3 Comments
- http://www.ibm.com/developerworks/library/l-proc/l-proc-pdf.pdf
- http://geeksterminal.com/linux-directory-structure-file-system-with-diagram/68/
- http://www.ibm.com/developerworks/library/l-linux-kernel/index.html
- Nutt, Gary, "Kernel Projects for Linux", Addison Wesley, Inc, 2001.