February 20, 2008
Operating Systems
LIACS
Spring Semester 2008
Assignment #2

Deadline: Tuesday, March 4, 2008
You are encouraged to work in teams of 2 persons.

Goal
Learn several important characteristics of the Linux kernel and processes using the /proc mechanism.

Introduction

The Linux kernel is a collection of data structure instances (kernel variables) and functions. The set of kernel variables define the kernel's view of the state of the entire system. Each externally invoked function - a system call or an IRQ - provides a prescribed service and causes the system state to be changed by having the kernel code change 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 the 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`.

In general, the extensible data structures are defined in C *header files* (also called *include files* or *.h files*) in the Linux source code tree. The Linux source code may be loaded in any subdirectory, though the conventional location is in `/usr/src/linux`. Most of the system's include files are stored in the `.../linux/include` directory. This directory's content depends on the version of the source. In version 2.4.20, it includes the directories `asm-generic, asm-i386, linux, net, scsi`, and `video`. All of the machine independent `include` files are kept in the `linux` subdirectory, and most of the machine-dependent `include` files are kept in `asm-i386`. The other subdirectories contain miscellaneous `include` files.

The `include` file `include/linux/sched.h` defines a data structure used for the process descriptor.

```
struct task_struct {
    /* These are hardcoded -- don't touch*/
    volatile long state; /* -1 unrunnable, 0 runnable, >0
stopped */

    ...
    int pid;
    ...
    uid_t uid, ...;
    gid_t gid, ...;
    ...
};
```

The source code file `kernel/sched.c` contains a declaration of the form

```
struct  task_struct * task[NR_TASKS] = {&init_task, };
```

The task kernel array variable is the pointer to all of the process descriptors. If you could read the task kernel variable and wanted to know the kernel state of a process with a process ID of say 1234, you could find the record `task[i]` such that `task[i]->pid == 1234`. Knowing the value `i`, you could then read the value of the `task[i]->state` field to determine the current state of that process. The `task[i]->uid` field is the user ID for the process, and `task[i]->gid` is the group ID for the process. (A process descriptor contains many more fields, as you will discover in subsequent exercises.) When you know the process ID of a running application (e.g., 1234), you can examine its state by inspecting the `/proc/1234` subtree in the /proc pseudo-file system.

A useful first step to understanding the details of the Linux implementation is to explore the kernel variables and data structure definitions that are used to write the code.

You can begin to examine the values for some of the kernel variables using existing tools. This exercise takes this approach and lets you inspect the kernel state and gives you some intuition about how the kernel behaves.

Assignment

This 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 system you are using:

- What is the CPU type and model?
- What version of the Linux kernel is being used?
- How long (in days, hours, and minutes) has it been since the system was last booted?
- How much total CPU time has been spent executing in user mode? System mode? Idle?
- How much memory is configured into it?
- How much memory is currently available on it?

Part b)

Write a program to report the behavior of the Linux kernel by inspecting the kernel state. The program should print the values on standard output:

- CPU type and model
- Kernel version
- Amount of time since the system was last booted, in the form dd:hh:mm:ss (3 days, 13 hours, 46 minutes, 32 seconds would be formatted as 03:13:46:32)

Part c)

Write a second version of the program that prints the same information as the default version plus the following:

- The amount of time the CPU has spent in user mode, in system mode, and idle
- The number of disk operations (read & write) made on the system
- The number of context switches that the kernel has performed
- The time when the system was last booted
- The number of processes that have been created since the system was booted

Call the first version of the program you created in part b "`info`"; then the second version is distinguished by a parameter: `info -e.`)

Part d)

Write a third version of the program that prints the same information as the second plus the following:

- The amount of memory configured into this computer
- The amount of memory currently available
- A list of load averages (each averaged over the last minute)

For this version you need to provide two additional parameters: one to indicate how often the load average should be read from the kernel and one to indicate the time interval over which the load average should be read.
(Of course, you need to ensure that the computer is doing some work rather than simply running your program.)

Hints:

i) Use the pseudo-file system `/proc` ; you can start learning about it by looking at the man pages and by inspecting the contents of some of the files using the `cat` command.

ii) In order to pass arguments, define a C main function with a header of the following form:
```
int main (int argc, char * argv[])
```

iii) Use system calls such as `fopen, sleep` etc. Another function that might be useful for reading in the desired data is `fscanf()`.

iv) See also pages 112-114 of Operating Systems by G. Nutt.

Deliverables:

- i A README file which contains a list of items you turned in + in what fashion they are to be used. Also mention the platform(s)/kernel version(s) on which you have tested your programs.
- ii The answers to part a.
- iii All (three) C programs - part b, c and d.
- iv A one-page lab report detailing what you have done in the lab, what problems you encountered and more importantly what you have learned from this and what you have learned from your lab partner.

Make sure each deliverable contains the names of the authors, student IDs, assignment number and date turned in!

Due date: as stated before March 4.

Put all files you want to deliver into a separate directory (e.g., *assignment2*), free of object files and/or binaries, and create a gzipped tar file of this directory:

```
      tar -cvzf assignment2.tgz assignment2/
```

Mail your gzipped tar file to Sven van Haastregt (e-mail: svhaastr at liacs dot nl) and *let the subject field of your e-mail contain the string* ``OS Assignment 2".