

# **Project 1 Specification**

## **/proc FS Navigation**

### **&**

## **Introduction to System Call Tracing**

Assigned: September 2, 2009

Due: September 14, 2009

Language Restrictions: C only

Additional Restrictions: `system()` and `exec*()` system calls may not be used

### **Purpose**

There are two phases to this project, each with its own purpose. The first phase is designed to familiarize you with the Linux `/proc` file system. `/proc` contains a great deal of information regarding the hardware that the kernel is running on. The second phase asks you to analyze the use of system calls with the assistance of the `strace` command.

Both phases of the project aim to help you become more comfortable with C programming, particularly in the use of the C standard library, performing I/O, and string handling.

### **Problem Statement**

Design and implement a system that provides the user with information about their system. The program should be capable of either informing the user about the system hardware or the current running processes.

Implement a program that uses five system calls to perform minor tasks. These tasks can be anything and any system calls may be used. Trace the execution of that program and answer the questions provided in the following section.

### **Questions**

1. The operating system provides an interface to the user (you) to use the services it provides. This is the system call interface. The command line tool, `strace`, allows you to see behind the interface and gives you insight as to how the operating system implements given system calls.
  - a) What system calls did you use to implement part 2?
  - b) What system calls did `strace` report? (All functions printed by `strace` are system calls.)
  - c) What's different between `log1` and `log2`?
2. In lieu of extracting information from the `/proc` file system to complete this project, you may have gotten a few ideas on how to take advantage of this information in future projects. List three pieces of information available in the `/proc` file system that you found particularly interesting.

# Project Task

---

## Part 1: Scanning the /proc FS

---

You are tasked with writing a program that prints to the terminal information regarding the state of the machine. The executable must be named `procinfo`. It should be capable of running in three modes. For extra credit, you may implement an additional, specified mode. The following is a demonstration of the command-line usage:

1. `$> procinfo`
2. `$> procinfo -p`
3. `* $> procinfo -t`
4. `$> procinfo (anything else...)`

### Mode 1: System Information Mode

The following must be printed if the executable is running in system information mode:

- Current time – provide the user with the date.
- System uptime – how long has the machine been powered on?
- System idle time – how long has the idle process been executing?
- Kernel version – what version of the kernel is being executed?
- CPU information:
  - Cache size
  - Clock frequency
  - Model name
  - **Note:** You may not assume there is only one CPU. If there is more than one CPU, you should print out this information for each CPU installed.
- Memory information:
  - Total system memory installed
  - Total system memory in use
- Maximum number of kernel threads that can be created

### Mode 2: Process Information Mode

If the executable is passed the flag “-p”, it must print out information regarding the processes associated with the current terminal. More precisely, the executable should print out all processes that have as a parent process ID the process ID of the shell you are running. This process ID should be the same as the parent process ID of the executable running.

### Mode 3: Process History Mode (Extra Credit, +10)

If the executable is passed the flag “-t”, it should print a trace leading back to the *init* process. All processes have a parent that can be traced, except for the *init* process. Therefore, it should be possible to follow the parent of a process to the *init* process. Therefore, you must print out the following information:

- If the process has a parent, print out the process and scan the parent.
- If the parent process has no valid parent, print out the parent and return.

## Mode 4: Error Mode

Sometimes, users of your executable will make a mistake and type something on the command line that they did not intend to. More likely still, users of your executable will be not be sure how to correctly run your program. In order to assist users, if a user passes invalid arguments to your executable, it should print an error message.

### Formatting Details:

To maximize the effectiveness of the information printed, it should be easy to read. The following demonstrates a suggested output format:

**\$> procinfo**

Printing system information...

-----

Time

-----

Current Time: Wed Sep 2 10:15:21am 2009

System Up Time: 3 Day(s) 2 Hour(s) 10 Minute(s) 23 Second(s)

Idle Time: 0 Day(s) 1 Hour(s) 4 Minute(s) 23 Second(s)

-----

Machine Specs

-----

Number of CPUs: 2

CPU 0

- L1 Cache Size: 256 kB

- Clock Frequency: 800 MHz

- Model: AMD Turion X2

CPU 1

...

Total Installed Memory: 200000 kB

Total Memory In Use: 100000 kB

-----

Kernel Information

-----

Kernel Version: 2.6.15

Maximum Kernel Threads/Processes Creatable: 123000

```
$> procinfo -p
```

Printing processes running on this terminal:

```
PID  COMMAND
4125  bash
4170  nano
4172  procinfo -p
```

**[EXTRA CREDIT]**

```
$> procinfo -t
```

Printing process trace from init to this process (procinfo):

```
> init(1)
-> gdm(2606)
--> gdm(2609)
---> X-session-manag(3029)
----> compiz(3338)
-----> compiz.real(3403)
-----> sh(5128)
-----> gnome-terminal(5129)
-----> bash(5367)
-----> procinfo(7540) [your process]
```

```
$> procinfo (anything else...)
```

```
Usage: procinfo [-p | -t]
```

---

## Part 2: System Call Tracing

---

Write a program that uses five system calls. This program may use any system calls, provided that you only use five. You'll find the system calls available to your machine in the file `/usr/include/unistd.h`. If you are going to be using file-related system calls, it may also help you to `#include fcntl.h` for the `open()` system call flags (`O_RDONLY`, etc, ...).

Once you've written this program, execute the following commands, where `<program_name>` should be substituted for the name of the executable for your program:

```
$> make
$> strace -o log1 <program_name>
$> make clean
$> make
$> strace -o log2 <program_name>
$> more log1
$> more log2
$> diff log1 log2 | more
```

Study the output from the last three commands while considering the questions given at the beginning of this document. To reduce the length of the output from `strace` into the log files, try to minimize the use of other function calls in your program beyond the five system calls.