Monitoring Memory and CPU Usage for Processes in Linux

Jatczak, David McNabb, Trent

December 8, 2016

1 Introduction

1.1 Context

Addressing this issue requires knowledge of what information is relevant to the calculation of memory and CPU (Central Processing Unit) resources, and how to get that information. Specifically, knowledge of the Linux '/proc/' file is needed.

1.2 Problem Statement

Computer users who are used to Windows might use Ctrl+Alt+Delete when they wish to see which processes are running and how much of the system's resources those processes are consuming. They may not be comfortable using the terminal to run commands that would display that information, and they might prefer to see the information in a Graphical User Interface (GUI). We would like to write a program that we can make accessible via the Ctrl-Alt-Delete keyboard shortcut that resembles the Windows Task Manager. It should allow for a clear and easy view of which processes are running and what resources each process is using. It should be easy to terminate a running process.

1.3 Result

We have written a program that is called on our system using the Ctrl-Alt-Delete keyboard shortcut. The program uses a GUI to display the running

processes and the percentage of CPU resources and memory used by each process. The program allows users to easily select and terminate processes.

1.4 Outline

The rest of this report is structured as follows: Section 2 presents information about calculating system resource usage by specific processes using the Linux '/proc/' file; Section 3 describes the result of the project, which is the program we have written; Section 4 evaluates the result of the project; Section 5 is the conclusion.

2 Background Information

The Linux '/proc/' file system stores information about processes and also other system information. Each process has a directory in the '/proc/' file system[3, p. 792]. The directory is created at '/proc/[pid]/', where pid is the process identification number. Each of these directories include a 'stat' file, a 'statm' file, and a '/status' file. In addition to the information about each process, there is also system information, i.e. the '/proc/cpuinfo' file and the '/proc/meminfo' file.
The Resident Set Size in the number of pages that the system currently has in real memory. If we multiply that by the page size, that gives the amount of RAM currently used by the process. If we divide that by the total amount of memory used by the system, that will give us the percentage of system memory used by the process. The calculation is:
(Resident Set Size * Page Size)/(Total System Memory.)
Information about the resident set size of a process can be found is '/proc/status/statm' [2]. Information about the total system memory can be found in '/proc/meminfo' [2].

A CPU can either be running or idle. If it is running, it can be running a user space program, or running the kernel [4]. If you add the time it is running the kernel, plus the time it is running a user space program, plus the amount of time it is idle, you get what we will call the Total CPU availability. The percentage of CPU usage by a process is the amount of time that the CPU is running that specific process, divided by the Total CPU availability multiplied by 100. The percentage of total CPU usage is the amount of us-

age, divided the total CPU amount, multiplied by 100, where the amount of usage is the time running user space programs plus the time running kernel space programs.
The amount of time that a process has been running in user mode can be found in '/proc/pid/stat'. [2] The total time spent by the CPU running user space processeses and the amount of time running the kernel can be found in '/proc/stat' [2].

Shortcuts, also known as hotkeys, that give functionality to applications are typically handled by an event handler. Whenever a user presses a key on an operating system, an event is made and sent out for the event handlers to handle. There are many more types of events than key presses and key releases, such as mouse movements, and GUI events. Event handlers are in event loops. Event loops are embedded in applications themselves and/or in the operating system. Simple processes such as shell tools do not typically have event handlers, but many modern applications do. The typical structure of an event loop is that goes through the queue of events it has, does what it needs and then chooses to dispatch the event or not, afterwards waiting for more events.

The Qt framework offers a library that gives GUI functionality and is based in C++, but it also used in other languages such as Python where it has been ported. The Qt framework has an event loop and receives its events from windowing systems such as X11 or Wayland. If you are interested in this topic, it is explained further by Thiago Macieira in his powerpoint on the Qt event loop[1].

For this text, we use X11 as our window system. X11 is the base of the GUI system for Linux. It handles all the communication between the user input and other windows because it is the framework that other applications are working off. X11 has its own event loop, and is basically at the top of the hierarchy of event loops.

To interact with X11's event loop, you require a library such as Xlib or XCB. XLib was made by the same company that created X11. XCB is a faster alternative to Xlib and is asynchronous. Both of these libraries communicate with X11 via binary because X11 uses a client-server design for its system.

3 Result

Result goes here.

4 Evaluation

Evaluation goes here.

5 Conclusion

5.1 Summary

Summary and highlights.

5.2 Relevance

Relevance with respect to the course topic.

5.3 Future Work

Question for TA – future work in the field of this project, or ways that the project application itself could be continued on.

Contributions of Team Members

References

[1] Thiago Macieira - Qt Core Maintainer `http://github.com/boostcon/cppnow_presentations_2013/blob/master/mon/qt_event_loop.pdf?raw=true`

[2] `http://man7.org/linux/man-pages/man5/proc.5.html`

[3] Tanenbaum, A. S. (2015). *Modern operating systems.*

[4] `http://blog.scoutapp.com/articles/2015/02/24/understanding-linuxs-cpu-stats`