1 Examining a Running Linux System Using the /proc Filesystem

The nucleus (kernel) of the Linux operating system consists of data structures and functions\(^1\). The overall (kernel) state of a system running Linux is defined by the values stored in kernel data structures. Functions that can be called from outside the kernel (“system calls”) inspect or alter the state of kernel data structures.

There are kernel variables to represent the information the kernel holds about each open file, every process, the system’s memory, the CPU etc. Just like a normal C program, the kernel can create local variables on the stack, and it can use global (static) variables to share information between different functions (and separately compiled program files or modules). The Linux kernel is in many ways one monolithic “program” which makes extensive use of global data structures to share information between its different parts.

1.1 The /proc Filesystem

Linux, as well as some other instances of Unix, have a very useful mechanism for inspecting some kernel data structures: the /proc filesystem. You can access /proc just like any other directory in the Linux filesystem; you will find various files and subdirectories there. The difference is that the files in /proc are not stored on any disk — instead they are “virtual” files which display the contents of the current state of some Linux kernel data structures. You can think of this as the kernel creating these files on-the-fly, as you read them, using its internal state to supply the contents. In the following exercise, you will be asked to find out some information about a running Linux system by looking at these files. You may find the following command useful:

- `man 5 proc`

  (`man` inspects online manual pages and 5 means section 5 of the manual, which is the section describing system files. Use `q` to exit the manual. You can use `/[text]` to search for text in the manual page.)

1.2 Examining the State of a Linux System

Log on to any lab machine running Linux. Use the /proc filesystem to answer the following questions about the system you have logged on to.

1. What is the type and model of the CPU?

2. How many CPUs are there (potential trick question — ask one of the lab helpers if you are confused!)?

3. What version of the Linux kernel is running?

4. How much memory does the system have?

5. How much memory is currently available?

---

\(^1\)This part of the exercise with acknowledgements to Gary Nutt.
6. How much time, in milliseconds, has the system spent reading and writing to disk?²

7. How many processes have been created since the system booted?

8. How many context switches has the kernel performed?

9. How long has it been since the system booted?

## 2 Processes and Priorities in Linux

The aim of this exercise is to give you some practical experience of processes and priorities under Linux. I do not expect you to understand everything that you can observe here — for that, come to the next 18 Lectures!

### 2.1 The “heartbeat” program

Download the following C++ program from


```cpp
#include <sys/time.h>
#include <iostream>

using namespace std;

double time_in_seconds() {
    double result;
    struct timeval tv;
    struct timezone tz;
    gettimeofday( &tv, &tz );
    result = (double) tv.tv_sec;
    result += (double) tv.tv_usec / 1000000.0;
    return result;
}

int main( int argc, char *argv[] ) {
    int r = 0;
    double current_time = time_in_seconds();
    double time_at_start = current_time;
    for( int i = 0; i < 100; i++ ) {
        // Waste some time!
        for( int j = 0; j < 10000000; j++ ) {
            r += j;
        }
        cerr << "Iteration " << i << " took "
             << time_in_seconds() - current_time << " seconds.\n";
        current_time = time_in_seconds();
    }
    double time_at_finish = time_in_seconds();
    cout << "Overall time taken was "
         << (time_at_finish - time_at_start) << "\n";
    return 0;
}
```

Compile this program with `g++ -o heart heart.cc`. Run the program once to make sure it works.

²Hint: Once you have located the right file in `/proc`, the columns you are need are columns 4 (milliseconds spent reading) and 8 (milliseconds spent writing).
2.2 top

top is a very useful program that gives information on what processes are currently running on a Linux system, and how much resources those processes are using. Find out more by looking at the manual pages (man top).

2.3 Observing Processes and Priorities

Open three new terminal windows, and arrange them in such a way that you can see all three. In one of the windows, start top. Make sure that you are in the directory where the heart program is located in the other two terminals.

1. Run the heart program in one terminal and observe.

2. Run the heart program in both terminals and observe.
   You will notice that we are now running two instances of the same program, resulting in two distinct processes in the OS.

3. Run the heart program in both terminals again, but this time start the program with the command nice ./heart in the second terminal. What do you observe?
   This illustrates process priorities; the effect of nice is to start a process with lower priority.

Olav Beckmann, Imperial College London, October 10, 2005