CMSI 387-01

Operating Systems

Spring 2013

Assignment 0219

This assignment has you walk through different platforms to identify what they have in common, and then lets you "break through" the command line for an initial taste of operating system programming.

Outcomes

This assignment will affect your proficiency measures for outcome *3a* ("CSI: OS"), *2d* and *4a–4c* (System Call Practice), and *4d–4f* (both).

Not for Submission

- If you have it, read Chapters 1 and 2 in SGG.
- You will likely want to do System Call Practice in a virtual machine such as VirtualBox or VMWare Player. For uniformity, use the latest Ubuntu Linux distribution (*http://www.ubuntu.com*).

For Submission

"CSI: OS"

Get yourself in front of a Linux, Mac OS X, and Windows computer and locate/identify/invoke the following parts of *each* operating system. Provide the requested "evidence" that you have actually seen these with your own eyes on each computer—it is *not* enough to just look them up and say what/where they are.

Find…	Document with…
Pre-OS software (firmware, I/O subsystem, boot loader, etc.)	Screenshots with name and description of what component is/does
Kernel file(s)	Screenshot of long-form file listing
Startup scripts and/or configuration files	Screenshot of file listing (just file names are OK)
The "first process"	Screenshot that identifies it on a process listing or activity/task monitor
Network settings	Screenshot of control panel or dialog

Commit your "evidence," along with a README that lists and describes the committed files, under *homework/csi-os*.

System Call Practice

Pick at least three (3) Linux system calls *except for the ones demonstrated in the* `syscall-samples` *and* `kernelwrite` *bazaar code* and write "wrapper" programs for each of them, for a total of three (3) new commands written by you.

- Each program must invoke the system call *using* `syscall` *directly* (i.e., no C convenience functions). If the system call result requires custom data structures use definitions from kernel-level header files and not the standard C library.
- Each program must produce some demonstrable, comprehensible result (e.g., output, changes to the file system, etc.).
- This one time, yes, to expose the low-level underpinnings of what you are doing, you may hardcode operating system constants and literals.
- You may use additional system calls to help do your work, but each program should center around one particular system call's functionality.
- You may resort to "normal" C (data structures, libraries, etc.) to process the results of your system call.
- Implement whatever command-line parameters you like, to make your programs more flexible or powerful. These are your commands now; make them whatever you'd like them to be.

Commit your work (source code only—no build products please) under *homework/my-own-commands*.