COMP 3000A: Operating Systems

Carleton University
Fall 2014 Final Exam Solutions

Total points: 25

1. [1] Each file has read, write, and execute permissions specified three times, e.g., a file may have "rw-r–r–". Why three times and not just once?
   A: In UNIX the first set of permissions are for the file's owner, the next set is for members of the file's group, and the third is for everyone else. (Half a point for at least one of the three; full credit for specifying all three.)

2. [1] What is the kernel mechanism that allows a user to forcibly terminate a running process in UNIX? Illustrate with an example.
   A: You send a KILL or -9 signal to a process to forcibly terminate it. You can send such a signal to process 547 by running the command `kill -SIGKILL 547` **or** `kill -9 547`**. Note that the kill command can send any signal. By default it sends SIGTERM, which requests termination of a process. (Half a point for specifying signals as the mechanism, half for the example command or call. Half credit on the whole quetion if SIGTERM is specified rather than SIGKILL.)**

3. [1] If we have a system where virtual address 0x722B2104 mapped to physical address 0x16AB2104, what is the largest page size that could be used for this mapping? Explain briefly.
   A: The largest page size that could be used is the set of low order bits that are identical between the two addresses. B2104 is the common suffix of both addresses, so the largest page size is at least 2^{20} (because each hex digit can be represented by exactly 4 bits). We then look to the first non-identical digit. 2 (from the virtual address) has a binary representation of 0010 while A (from the physical address) is 1010. These have 3 bits in common, so the largest page size that could be used is 2^{23} for this pair of virtual and physical addresses.

4. [1] Do pointers in userspace C programs contain virtual or physical addresses on Linux? Explain briefly.
   A: All pointers in userspace contain virtual addresses because userspace programs each run inside of a virtual address space. Physical addresses can only be accessed while in supervisor mode in the CPU, not in user mode. (Half a point for saying it is a virtual address, half for the explanation.)

5. [1] What data structure allows the kernel to determine when a process is accessing an invalid memory area?
   A: A process's page table is what allows the kernel to determine what memory is and isn't valid (allocated) for a process.

6. [1] In C, the function call stack is used to store three distinct kinds of values, only one of which is required to be there by the CPU. Which value is required to be there by the CPU?
   A: The return address is required by the CPU. (The other values are function argument and local variables, but the question did not ask for those to be specified.)

7. [1] In a system with a 64-bit address space, does the physical address space have more than 64 bits, 64 bits exactly, or less than 64 bits? Explain.

A: It has a physical address space of (much) less than 64 bits because it is technologically infeasible to have anything close to 2^{64} bytes of memory. For example, current AMD chips have a 48-bit physical address space, allowing for 256 terabytes of RAM. (Half a point for saying less than 64 bits, the other half for the explanation.)

8. [1] Can concurrency primitive such as mutexes be implemented without the use of special instruction such as `xchg` on modern CPUs? In other words, can such concurrency primitives be written purely in C on current processors? Explain.
   A: Concurrency primitives must use special instructions because of the depth of the memory hierarchy on modern processors and the associated costs of keeping all the levels in sync. For exmaple, two cores can keep local copies of a memory location in L1 cache; while these copies are periodically synchronized, it is still possible to access an old value (i.e., a core won't immediately see the writes performed by another core). Instructions such as xchg force the affected memory locations to be synchonized across all cores, thus allowing the state of the concurrency primitive to remain coordinated even when accessed by multiple cores. (Half a point for saying "no", the other half for the explanation.)

9. [2] A fork bomb can be a simple as "`while (1) fork();`". Why are fork bombs so dangerous? Explain why a fork bomb can kill system performance (assuming a system that does not have built-in defenses against fork bomb-like attacks).
   A: Fork bombs are dangerous because they can create an unbounded number of processes. Because CPU schedulers normally try to give every process some CPU time to execute, an overwhelming number of processes means that the few legitimate processe will get very CPU time. In addition, they can create so many processes that the OS will not permit any more processes to be created; thus, it can be impossible to run the kill command that would terminate the fork bomb! (Normally the root user has a few process entries reserved for it to deal with precisely such a contingency.) (One point for something about an unbounded number of processes, another point for a roughly correct description of the fork bomb problem.)

10. [2] What is the relationship between function calls, system calls, and library calls?
    A: A: Function calls and library calls happen within a process; system calls are invocations of the kernel and thus cause execution of code outside of the process (in the kernel). Because system calls cannot refer to addresses (because the addresses of kernel memory are not an accessible part of a process's address space), they instead make use of special CPU instructions that cause the CPU to switch into supervisor mode and then execute pre-specified code in the kernel, namely the system call dispatcher. Library calls are just function calls made to dynamically linked libraries. There is a difference because calls to dynamically linked libraries are indirect because the location of the library can change from execution to execution; calls to other functions are statically linked and so use absolute addresses. (One point for properly separating system calls from the rest (you didn't explain everything about system calls, just be clear that they weren't running code in the process's address space), one for identifying the difference between library and function calls.)

11. [2] What happens when you type a command at a shell prompt in UNIX that is not built in to the shell? Specifically: 1) what code does it run, 2) how does it find that code, and 3) what system calls (if any) does the shell do (at minimum) in order to execute that command?
    A: When you run an non-built in shell command, the shell runs a separate executable with the same name as the command in a new process. The shell searches for such a named binary in the directories specified by the PATH environment variable. The shell does a fork system call

to make a copy of the shell and an execve system call to run the command binary. (A half point each for saying it is an external program binary, looks in PATH, fork, and execve.)

Hard Disk Recovery: Your Linux computer's hard disk is starting to produce errors and you do not have a current backup of the disk. Your old disk is 500Gb; you are replacing it with a 2Tb drive (2000 Gb). You have already purchased and installed the new disk in the computer and have installed a fresh copy of Linux onto it. The new hard drive is now /dev/sda, with everything in one partition in /dev/sda1, while the old hard drive is in /dev/sdb, with all of its data in /dev/sdb1. You boot from the first disk and the old, failing disk (/dev/sdb) is unmounted.

12. [1] To make the files on /dev/sdb1 accessible in /mnt, what command should you run (as root)? Please give the full command.
    A: mount /dev/sdb1 /mnt

13. [1] When attempting to access files from the disk you find that the underlying filesystem has been heavily corrupted. So, you'd like to try and repair the damage.

    What command should you use to attempt to repair the filesystem on /dev/sdb1?
    A: fsck or fsck.ext4 (full command is "fsck /dev/sdb1" but this was optional)

14. [1] Before you attempt the repair a friend warns you that if the repair goes badly you may lose even more data from the disk. Thus you decide you want to copy the raw data from disk into a file first.

    What command could you use to make a bit-for-bit copy of /dev/sdb1 into the file /old-image?
    A: dd (optional full command is "dd if=/dev/sdb1 of=/old-image") or cp

15. [2] You decide to use the `rsync` command to copy the files from /mnt to /old. Because the old disk is failing, you have to run rsync multiple times. You notice that each time `rsync` seems to pick up right where it left off, in that it doesn't copy files that have been fully transferred but does continue to copy files that it was in the middle of copying.

    How does `rsync` know which files to copy? And is it possible for `rsync` to make a mistake (to not copy a file that is in fact different between the two directories)?
    A: rsync compares file metadata between the source and destination and only copy those with significant differences (e.g., different last modified times, different lengths, etc). By default rsync does not compare the contents of files for those with similar metadata; thus if a file at the source and destination have the same last modified time, length, and other attributes, but their contents are completely different (e.g., one is just filled with zeros because of a corrupted copy) rsync can be "fooled" into not copying a file that should be updated. (Optional: you can force rsync to compare the contents of all files with the –checksum argument. This makes the copy much, much slower if the source and destination are mostly the same already.)

YourData Interview: You are interviewing for a startup, YourData, that wants to provide a new way for people to access all of their data in the cloud, no matter where it is stored. They are considering you for a Linux development position and they have the following questions for you:

16. [1] We want to use an authentication agent-type architecture where one process will store and protect all of a user's secrets. We know that ssh has a similar architecture but we don't understand how various processes can find the authentication agent.

    How do processes know how to contact the authentication agent for the current user?
    A: The process checks its environment variables for one specifying information on the authentication agent, e.g. SSH_AUTH_SOCK.

17. [1] We've implemented a custom filesystem using FUSE. On some file operations our FUSE process crashes with a segmentation violation error. What sort of coding error is causing the segmentation violation?
    A: Most likely it is caused by dereferencing an invalid pointer, e.g. attempting to dereference a NULL pointer.

18. [1] When the FUSE process crashes the filesystem still is listed by the `df` command, even though no files can be accessed in it. How can you tell the kernel that our FUSE-based filesystem should no longer be accessible?
    A: fusermount -u <name-of-mountpoint>

19. [1] One of our new developers is saying that our filesystem will perform better if we implement it as a kernel module rather than using FUSE.

    What is one reason you would expect a kernel implementation of a filesystem to be faster than one implemented using FUSE?
    A: A kernel implementation can be a bit faster because it avoids a second context switch into the process with the FUSE-based implementation. In general, though, the cost of this context switch is small relative to the cost of the actual I/O (to network or disk). (You didn't have to say the efficiency gain wouldn't be very large.)

20. [2] If the current buggy FUSE code is ported to a kernel module, would you expect it to still generate segmentation violations? Explain.
    A: The same bad pointer dereferences would still be there; however they wouldn't generate segfaults because bad pointers don't cause signals to be generated in the kernel; instead, they produce Oops messages in the logs or even kernel panics or crashes. The kernel can't generate segfaults because signals such as SIGSEGV are only defined with respect to processes. The kernel is not a process or a set of processes; instead, it is the code that implements the process abstraction.