COMP 3000: Operating Systems

Carleton University
Fall 2014 Mid-Term Exam Solutions

October 22, 2014

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 0x861B2152 mapped to physical address 0x16AB2152, what is the largest page size that could be used for this mapping?
   A: (To get full credit you just have to get the correct answer. If you say one megabyte that is also acceptable.) The last five hex digits of the addresses are both B2152, so the maximum page size is at least $2^{5\times4} = 2^{20}$ or 1 MiB (mebibyte). The next digit is 1 for the virtual address (0001 in binary) and A in the physical address (1010 in binary). Since these two numbers have no common suffix in binary the maximum page size is just 2^{20}.

4. [1] Describe one workload where a non-preemptive scheduler gives the best performance.
   A: There are many possible answers, but the key idea is that the workload should not require any interactive performance. Thus a workload where there is a set of jobs that need to be completed in any order but as quickly as possible with minimal overhead. One classic example is doing a company's payroll: it needs to be completed as soon as possible; however, intermediate results are generally not necessary so each process just needs to run to completion. (To get full credit you just have to specify an appropriate workload; you don't have to explain why it is appropriate.)

5. [1] When a C program starts, main() is given the a pointer to a set of command line arguments. Does this pointer contain a virtual or a physical address? Why?
   A: The pointer contains a virtual address because all pointers in userspace contain virtual addresses. (Half a point for saying it is a virtual address, half for the explanation.

6. [2] If you have write permission to a directory in UNIX, what is one thing that allows you to do? What is one thing such permission does **not** allow you to do?
   A: With write permission you can change what files are in the directory: you can add and remove files. It does not allow you to get a listing of the files in the directory or to access the contents of any of the files in the directory. Weird fact: on modern systems, having only write permission to a directory lets you do almost nothing with it. (Half a point for what you can do, half for what you can't.)

7. [2] What is one way in which a web browser is like an operating system? What is one way that it is not like an operating system?
   A: Web browsers are like an operating system in that web browsers provide an execution environment for running programs that abstracts away details of the underlying platform. They also manage resources, e.g. allowing users to discard loaded web pages and reclaim the memory consumed. They are

not like operating systems in that do not have full control over the underlying hardware and they do not allow (most) programs to directly execute on the system's CPU. (One point for each question.)

8. [2] How can the kernel know when a memory access (e.g., a pointer dereference) is for memory that isn't allocated to the process? Specifically, what data structures contain this information and when is this data structure accessed?
   A: A process's page table describes how virtual memory is mapped to physical memory. If a pointer is in a part of the address space (is part of a page) which has no valid mapping in the process's page table, then the kernel knows that that pointer is invalid and can send a error to the process (e.g., send a SIGSEGV signal). The page table is accessed when the TLB doesn't have an appropriate entry for virtual address's page or when the process's memory map changes, for example when more memory is allocated to the process. (One point for correctly describing page tables, the second for describing how TLBs relate to page tables.)

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 happens to response time with RR as quantum lengths increase? Explain in words and write an equation that gives the worst-case response time, given N jobs.
    A: With RR, as quantum lengths increase response time increases because the time a process must wait for its next time slice (in the worst case) is proportional to the number of other processes and the maximum amount of time they can execute. Thus on a system with n processes and a quantum size of Q a process will have to wait for $Q(N-1)$ to respond to a request (in the worst case). (One point for the verbal description, one for the equation.)

11. [2] 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. What are these three kinds of values, and which one is required by the CPU?
    A: The stack is used to store return addresses, function arguments, and local variables in C. Only the return addresses are required by the CPU. (One half point each.)

12. [2] 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. (One point for saying less than 64 bits, one for the explanation.)

13. [2] Can concurrency primitive such as mutexes be implemented without the use of special instruction such as `xchg` on modern CPUs? In other words, can 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. (One point for saying "no", the other for the explanation.)

14. [2] What is the relationship between function calls, system calls, and library calls?
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.)

15. [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.)