Name: _____ **Student ID #:** _____

COMP 3000B: Operating Systems

Fall 2007 Test 1 Solutions

November 18, 2007

1. [1] "Zombie" processes are caused by:

   (a) An execve of a non-existent program
   (b) **A failure to wait for terminating processes**
   (c) A fork when system resources are low
   (d) All of the above

2. [1] When a regular process is running on a single-CPU system, all of the following are true **except**:

   (a) The kernel is not running.
   (b) Disks can be writing data to memory.
   (c) The CPU will generate an exception (software interrupt) if certain addresses are accessed.
   (d) **The CPU is in supervisor mode.**

3. [1] Monitors:

   (a) use condition variables for synchronization
   (b) prevent multiple threads from executing monitor code at the same time
   (c) hide mutual exclusion details from calling routines
   (d) **all of the above**

4. [1] System calls must be used to:

   (a) modify global variables
   (b) call a user-written function
   (c) **write to a file**
   (d) All of the above

5. [1] Pipelining. . .

   (a) . . .involves dividing machine instructions into smaller units of work.
   (b) . . .allows modern CPUs to execute more than one instruction in parallel.
   (c) . . .slows down context switches and increases interrupt latency.
   (d) **All of the above**

6. [1] When choosing whether to implement a device driver as a block or a serial device, which of the following information about the new device is **most** relevant?

   (a) **The same data is likely to be read multiple times.**
   (b) The device returns data in fixed-sized chunks.
   (c) The device is fast.
   (d) The device is attached by USB.

7. [1] Which UNIX command will show you the files in the current directory?

   (a) chmod
   (b) ps
   (c) more
   (d) **ls**

8. [1] If "ls -l hello" outputs the following, which of the following is true about hello?

```
-rwxr-xr--  2 soma sys 82021 2005-11-20 10:06 hello
```

   (a) The user soma may execute hello.
   (b) Members of the group "sys" may execute hello.
   (c) The file hello was last modified on November 20, 2005.
   (d) **All of the above**

9. [1] When a web browser requests a web page, it is performing a type of inter-process communication. This communication is best described as:

   (a) **message passing**
   (b) shared memory IPC
   (c) semaphore-based synchronization
   (d) None of the above

10. [1] Disabled interrupts are effective at enforcing mutual exclusion in which of the following contexts?

    (a) an OS kernel on a a symmetric multiprocessor (SMP) system
    (b) an OS kernel on a cluster of networked computers
    (c) **an OS kernel on a single processor system**
    (d) a web-based distributed application

11. [1] "Elevator" scheduling is used by:

    (a) Ethernet cards
    (b) Flash drive controllers
    (c) **Hard disk controllers**
    (d) Operating system CPU schedulers

12. [1] Producer/consumer programs use mutual exclusion mechanisms (semaphores, monitors, etc.) to:

    (a) slow down consumers that run faster than producers
    (b) slow down producers that run faster than consumers
    (c) prevent the shared queue from becoming corrupted
    (d) **all of the above**

13. [8] **Environment Variables**: I have two `gnome-terminal` terminal windows open, A and B, each running `bash` shell processes. In window A I have the following dialog:

```
lab01:~ $ echo $PATH
/usr/bin:/bin
lab01:~ $ echo $HOME/bin
/home/student/bin
lab01:~ $ cp /bin/ls $HOME/bin
lab01:~ $ export PATH=$HOME/bin:$PATH
```

Assume no other commands have been executed in either the A or B windows.

(a) [1] What is the value of the `PATH` environment variable in A's bash process (after the commands above have been entered)?

/home/student/bin:/usr/bin:/bin

(b) [1] What is the value of `PATH` in B's bash process?

/usr/bin:/bin

(c) [1] What is the value of `PATH` for `gnome-terminal`, the parent process of both bash processes?

/usr/bin:/bin

(d) [2] What is the value of `PATH` for `top` when run in window A? Window B?

Window A: PATH=/home/student/bin:/usr/bin:/bin

Window B: PATH=/usr/bin:/bin

(e) [3] If I run the command `ls`, which binary will get executed in window A? Window B? Why?

Window A will run /home/student/bin/ls in response to the command "ls" because it is the first version of ls in A's path. Since /home/student/bin is not in window B's path, it will just run the regular /bin/ls binary.

14. [4] **Semaphores**

(a) [2] What is the basic strategy for using a binary semaphore (mutex) to prevent concurrent access to a shared data structure?

First grab the semaphore (lock it) before accessing the data structure (this may require sleeping if the lock isn't initially available). Then, release the semaphore (unlock it) when the access is complete.

(b) [2] If the semaphore is used incorrectly, what problems can arise? State two simple errors and the problems these errors can cause with the execution of concurrent threads and/or the state of the shared data structure.

If the semaphore isn't grabbed before accessing the data structure interleaved concurrent accesses may occur, corrupting the data structure. If the semaphore isn't released, other accessing threads will wait forever for access (they'll starve).

15. [7] **File Permissions**

    (a) [3] On UNIX directories, what operation(s) do read, write, and execute access enable?

    Read allows the listing of the contents of a directory. Write allows directory entries to be added, modified, or removed. Execute allows the contents of files (its inode) to be accessed.

    (b) [2] Why is the "sticky" bit useful for directories shared by multiple users? Explain.

    The sticky bit ensures that only the owner of a file is allowed to remove it from the directory. This prevents users from removing each other's files in world-writable directories such as /tmp.

    (c) [2] Which are more flexible, standard UNIX or Windows file permissions? Explain.

    Windows file permissions are more flexible because on Windows files can belong to more than one group or user.

16. [4] **Process and Thread Management**

    (a) [2] When a thread exits, does a process exit? When a process exits, does a thread exit? Explain.

    When a thread exits, a process does not necessarily exit because it may have other threads executing. If a process exits, it implies that all of its threads exit (so at least one thread must exit).

    (b) [2] Kernels in many operating systems support multiple execution contexts within the kernel, i.e. part of the kernel may be blocked waiting for an I/O event while another is processing a system call. Are these "execution contexts" best thought of as threads or processes? Why?

    They are better throught of as threads, as they all share a single address space.

17. [3] **Shared Memory**

    (a) [2] If a program calls only calls `shmget()`, is it possible for a program to successfully use a shared memory segment? Explain. (Assume that it inherited no shared memory segments from its parent.)

    It is not possible to access the contents the shared memory segment using only shmget() because shmget() does not return a pointer. shmat() must also be called to map the shared memory segment into the process's address space.

    (b) [1] Why are permissions necessary for UNIX shared memory segments?

    Permissions are necessary because shared memory segments can potentially be accessed by any process on the system (unlike standard process memory, which is strictly private). Permissions allow access to be limited to processes owned by specific users and/or groups.

18. [5] **Signal Handling**

(a) [1] Who is the "caller" of a signal handler? In other words, what part of a UNIX system causes a signal handler to be invoked?

The kernel is the caller of a signal handler. (While some signals can be initiated by other processes, other signals, such as SIGCHLD and SIGSEGV, are initiated directly by the kernel.)

(b) [2] The `kill` command and the `kill()` function can both be used to send signals to processes. What are two reasons for a user to send a signal to a process?

A user may want to send a signal to request a process to terminate (SIGTERM), to force a process to terminate (SIGKILL), to stop (SIGSTOP) and restart (SIGCONT) a process, and to send it a programmer-definde message (SIGUSR1 and SIGUSR2). SIGUSR1 are often used to tell a daemon (service) to reload its configuration.

(c) [2] Signals has long been a standard feature of UNIX; however, threads are a relatively recent addition. How could one use a signal handler to implement userspace threads (simulate multiple execution contexts)? Explain. (Hint: there exists an "alarm" signal.)

Threads can be implemented in userspace by setting an interval timer using setitimer(), which will cause a process to receive a periodic SIGALRM signal. The signal handler for SIGALRM then runs a scheduler which rewrites the stack such that different execution contexts are simulated. (E.g. if there are 2 threads then on one SIGALRM the stack is updated to be running thread 1 while on the next SIGALRM the stack is update to thread 2's state.)

Note that if threads are implemented in userspace all system calls must be "wrapped" such that they are only executed in non-blocking mode; otherwise, if one thread made a blocking read request, no other threads would execute until that read completed.

19. [2] **UNIX pipes**

(a) [1] Pipes can be simulated through the use of temporary intermediate files. Give an example of how the command `ls | more` can be simulated using the $<$ (read from file) and $>$ (write to file) redirection operators.

ls $>$ /tmp/foo; more $<$ /tmp/foo; rm /tmp/foo

(The semicolon just allows more than one command to be entered on the same line.)

(b) [1] What is a disadvantage to simulating pipes with intermediate files?

The main disadvantage is that intermediate files force the processes in the pipe to run in series rather than running in parallel. Having to wait for the "pipe" writing process to complete can be problematic, especially if it never terminates!

Because of file I/O caching, for small data transfers, simulating them with intermediate files will not necessarily result in much of a performance impact. For large data transfers, however, the speed hit and the space consumed by intermediate files can be significant.

20. [5] **Kernel structure**

(a) [1] Why are device drivers typically executed in CPU supervisor mode?

Drivers are typically executed in CPU supervisor mode to give device drivers direct access to hardware resources (I/O ports, DMA memory, etc.) and low-level resource management code in the kernel.

(b) [2] What is the difference between CPU supervisor mode and programs running as "root" or "administrator"? Explain.

CPU supervisor mode is a protection mechanism implemented by the CPU that allows kernels to have greater hardware access than other programs. Root or administrator are special categories of processes (users) that the kernel recognizes as priviledged. A process running as root is still a userspace process and runs in CPU user mode; however, it can make systems calls that can result in arbitrary changes to the kernel and other parts of the system.

There do exist systems which have no root user (i.e. there is no one privilege that gives access to all resources). Such systems must still rely upon CPU supervisor mode, however, in order to separate kernel and user code.

(c) [2] In microkernel-based systems, device drivers are run in userspace processes. How can such an arrangement improve system reliability and security? And, what is a problem that cannot be solved with userspace device drivers?

Microkernels can improve system reliability and security by reducing the amount of code running in the kernel. Less code in kernel mode means fewer software flaws, something that potentially translates into fewer crashes due to coding errors and fewer kernel-level security vulnerabilities. Further, a buggy or malicious device driver, by running in userspace, is limited in the amount of damage it can do (e.g. it cannot directly access or modify the memory of other device drivers or of the kernel itself).

Because userspace device drivers must make system calls to access many hardware resources, they can be slower than kernel level drivers. Through the use of memory management tricks and other optimizations, however, the performance of microkernels can be quite competitive (albeit still slower).

One major problem microkernels do not fix is the centralization of critical resources in a few devices that are heavily interconnected. If a hard disk or video card driver is flawed, it can cause a system to be non-functional even if the rest of the system is running.

Further, because userspace device drivers must have special access to hardware, they can often still crash a system simply by misusing those resources (e.g. by causing a hard disk to do dangerous DMA).