CS333 Midterm Exam
Spring 2018

Name: _____

Section: M/W Tu/Th NB

This exam has 28 questions. The total number of points is 74. The back of each page can be used in place of scratch paper or for answering a question.

Instructions In the event that multiple answers appear correct, select the **most** correct answer. For multiple choice questions, *circle the **letter** of the correct answer.*

1. [1 point] Which of the following do **not** result in a *voluntary* context switch.
   A. Invoking a system call
   B. Performing I/O
   C. The process calling the `yield()` routine
   D. Timer interrupt
   E. B and D
2. [4 points] Draw and properly label the xv6 state transition diagram.

3. [8 points] Provide correct definitions for the following terms.
   1. Concurrency
   2. Critical Section
   3. Race Condition
   4. Atomicity

4. [1 point] Why do we need hardware support for mutual exclusion primitives?
    A. Only hardware designers can design a lock
    B. Atomicity
    C. Multiple CPUs can lead to concurrency problems
    D. Semaphores are inherently tricky
    E. A well-written program does not require hardware support

5. [1 point] How can the operating system be guaranteed to periodically regain control of the CPU from a user process?
    A. The user process does I/O
    B. The OS programs a timer to interrupt the CPU at a future time
    C. The user process performs a system call
    D. The user process runs at a lower privilege level
    E. The OS runs at a higher privilege level

6. [1 point] What mechanism is used to create a new process?
    A. exec followed by fork
    B. exec
    C. context switch
    D. fork
    E. fork followed by exec

7. [5 points] List the major steps in the context switch process.

8. [4 points] Define *turnaround time* and *response time* and state for what type of jobs are each an appropriate metric.

9. [1 point] When evaluating spin locks, what are the most important criteria?
   A. We only care about fairness
   B. Fairness followed by performance followed by correctness
   C. Correctness followed by fairness followed by performance
   D. We only care about correctness
   E. Fairness followed by correctness followed by performance
   F. Unfair locks are undesirable.

10. [2 points] MLFQ avoids starvation using what technique?
   A. Priority queues.
   B. Periodic priority adjustment downward.
   C. Short jobs have priority over long running jobs.
   D. Periodic priority adjustment upward.
   E. Round Robin.

11. [1 point] When user programs request services from the kernel, a system call mechanism is used rather than a simple procedure call. Why?
   A. Operating systems cannot execute procedure calls.
   B. System calls are faster than procedure calls.
   C. System calls are safer than procedure calls.
   D. To switch the CPU to privileged mode.
   E. The instruction pointer needs to be saved on the stack.

12. [1 point] What do we call a program region that accesses shared data and that may be invoked by several threads at once where the results are non-deterministic?
   A. Concurrency
   B. Deadlock
   C. Critical section
   D. Race condition
   E. Semaphore

13. [8 points] List and briefly **explain** the conditions necessary and sufficient for deadlock.

14. [1 point] The scheduler exists to:
    A. Service interrupts.
    B. Determine which blocked processes can enter the CPU.
    C. Create a new process.
    D. Select the next process to enter the CPU.
    E. Remove unused processes from the system.
    F. A and D.

15. [1 point] The principle advantage of multi-threaded over single-threaded programs is:
    A. Deadlock avoidance
    B. Interrupt disabling
    C. Concurrency
    D. CPU virtualization
    E. Race conditions

16. [1 point] What is the principle disadvantage of SJF scheduling?
    A. Shortest job may never run.
    B. It uses first-in first-out queuing.
    C. Some jobs may never run.
    D. Circular wait.
    E. Priority reset value is large.

17. [3 points] State and briefly **explain** the principle advantage of CPU virtualization.

18. [1 point] What do we call the condition where two or more threads, accessing shared data, run the same code region at the same time and the results are dependent on how the threads are scheduled?
    A. Deadlock
    B. Mutual exclusion
    C. Interrupts
    D. Concurrency
    E. Race condition

19. [3 points] On a single CPU system, what mutual exclusion primitive(s) should we generally avoid? Briefly **explain** your choice.

    A. Condition variables

    B. Binary semaphores

    C. Mutex spin locks

    D. General semaphores

    E. A and D

20. [2 points] Two threads concurrently execute `++i`. The initial value of `i` is 0. Mark all possible final values of `i`.

    A. 0

    B. 1

    C. 2

    D. 3

21. [1 point] We use the fork system call as part of creating a new process. In effect, fork returns in two places (the parent and the child). What does fork return?

    A. 1 to the parent and 0 to the child.

    B. 0 to the parent and the parent's PID to the child.

    C. The child's PID to the parent and 0 to the child.

    D. 0 on success, -1 on failure.

22. [1 point] If a parent process wants a notification of when a child process terminates, what system call can it use?

    A. sleep

    B. wait

    C. fork

    D. uptime

    E. exec

23. [2 points] In class, we discussed how a lock has no memory but a counting semaphore does have memory. Explain what is meant by the phrase "does have memory"?

24. [10 points] Explain the MLFQ scheduling algorithm. How are interactive and non-interactive processes handled? How does the algorithm avoid starvation?

Recall the mutex implemented using Test-and-Set-Lock (`TSL`) from the slides. Here, a lock is a single word variable with two values:

- 0 = FALSE = not locked
- 1 = TRUE = locked

`TSL` does the following atomically:

- Get the (old) value
- Set the lock to TRUE
- Return the old value

25. [2 points] In lock acquisition, `TSL` can return either TRUE or FALSE. Briefly explain the correct interpretation of the return value.

Recall from the scheduling slides that semaphores are an abstract data type used to control access to common resources across multiple processes in a concurrent environment. Typically a semaphore is composed of several parts:

- An integer variable
- A wait list
- Two operations: wait (which decrements the variable) and signal (which increments the variable)

26. [1 point] We attempt to acquire a semaphore with `wait(semaphore)`. How does our program know whether or not we can proceed with our work?

27. [2 points] How do we ensure that the wait() and signal() operations in a semaphore are atomic on a multi CPU system? Mark all answers that are correct.

    A. Implement wait() and signal() as system calls and disable interrupts.

    B. Use pass-by-value

    C. Use a spinlock to ensure atomic access to the semaphore variable.

    D. Use pass-by-reference

    E. Use atomic increment and decrement operations.

28. [5 points] The following xv6 program determines the execution time of a command. **Find and underline any bug(s)**. Briefly **explain** each bug and how to fix the bug.

```
1  #include "types.h"
2  #include "user.h"
3
4  int
5  main(int argc, char* argv[]) {
6    int t1, t2;  // ticks
7    int pid;
8
9    argv++;
10   pid = fork();
11   if (pid < 0) {
12     printf(2, "Error: fork failed. %s at line %d.\n", __FILE__, __LINE__);
13     exit();
14   }
15   if (pid == 0) {
16     t1 = uptime();
17     exec(argv[0], argv);
18     printf(2, "Error: exec failed. %s at line %d.\n", __FILE__, __LINE__);
19   } else {
20     wait();
21     t2 = uptime();
22     printf(1, "%s ran in %d.%d seconds.\n", argv[0],
23       (t2-t1)/100, (t2-t1)%100);
24   }
25   return();
26 }
```