CSL373/CSL633 Major Exam
Operating Systems
Sem II, 2013-14

Answer all 11 questions (14 pages) Max. Marks: 62

For **True/False** questions, please provide a brief justification for your answer. No marks will be awarded for no/incorrect justification.

1. Does a logging-based filesystem (like ext3) make sense if the buffer cache is write-through. State your assumptions about how logging will be implemented with a write-through buffer cache to answer this question. [3]

2. In xv6, consider the following situation:
   a. A thread (say thread1) is executing on CPU0, and acquires spinlock X
   b. A timer interrupt occurs and the thread1 gets context-switched out (while holding X)
   c. Another thread (say thread2) starts executing on CPU0 and tries to acquire spinlock X, and spins (because thread1 is holding X)

Assume that threads in xv6 can share address space both within the kernel and in user-space (i.e., multiple processes can map the same pages in the user address space and access them concurrently). Answer the following questions:

(i) Based on this sequence of events and your knowledge of the xv6 kernel, can you say if the thread was executing in user-space or kernel-space at the time of the context-switch? [3]

(ii) How long will thread2 spin on CPU0? Why? [2]

(iii) If there are 100 threads that all acquire spinlock X, and the xv6 scheduler is round-robin, then what is the problem? [2]

(iv) Give two potential solutions to this problem. Your solution could involve change in the xv6 design/abstractions. Discuss which solution is better in what situations. [4]

3. Consider the following pseudo-code for exit/wait synchronization (only relevant parts shown, this pseudo-code is incomplete) using sleep/wakeup primitives, taken from xv6 lines 2354-2430:

```
void exit(void) {
    // release some resources, e.g., fds
    // wakeup(my parent process)
    // set my state to ZOMBIE
}

int wait(void) {
   // iterate over all my child processes
   // if find a child process that is ZOMBIE, free its resources and return its pid
   // if we find a child process that is currently not ZOMBIE, sleep(myself)
}
```

Answer the following questions:
a. What happens if the child calls exit() before the parent calls wait()? [3]

b. What happens if the parent calls wait() just after the child calls wakeup()? [3]

4. When an executable program is loaded into memory using the exec() system call, it is possible that only a few pages are loaded initially and the rest are demand-paged.

a. What are the advantages of demand paging? [2]

b. Is it possible that the overall performance of the system could have been better if demand paging was disabled, i.e., all pages of the executable are loaded apriori at load time? Give an example to justify. [4]

c. Explain why the copy-on-write optimization allows UNIX to separate process creation into two system calls, namely fork and exec? [3]

5. True/False: - The spinlocks implemented using the xchg() instruction (as in xv6) obey arrival order, i.e., if thread A reaches the acquire function before thread B (by reaching the acquire function, we mean the first execution of the atomic xchg instruction by the thread), then thread A will definitely obtain the lock before thread B (for example, even if the lock was held by another thread C). Explain. [3]

6. Fix the following program code such that all concurrent calls to the function foo() by multiple threads, are effectively serialized. [2]

```
void foo(void) {
    struct lock mutex;
    acquire(&mutex);
    // access a global variable
    release(&mutex);
}
```

7. The hardware provides the “accessed bit” per page to implement a cache replacement algorithm that accounts for access patterns, e.g., CLOCK.

a. CLOCK is also called a 1-bit approximation to LRU. Why? [2]

b. What is the advantage of using a two-hand clock over a single-hand clock? Explain clearly with example. [2]

8. Security: Consider a UNIX system, in which a special ‘root’ user has all privileges. Also assume that all common utility files (e.g., executables, configuration files, etc.), are owned by the ‘root’ user. Answer the following questions:

The executable file ‘/bin/ls’ is one such file owned by the root user. As you know this executable displays the contents of a directory.

a. Should the setuid bit on the ‘/bin/ls’ executable be one or zero? What are some bad things that can happen if the setuid bit of the ‘/bin/ls’ executable is set to one? Does it matter? [3]

b. A special program called ‘sudo’ allows a non-root user to execute commands using root privileges. The commands to be executed are provided as arguments to the ‘sudo’ program. Can the sudo program be implemented without using the setuid bit? Who is the owner of the ‘sudo’ executable? [3]

9. Many operating systems provide a ‘disk defragmenter’ utility. This utility can be invoked by the user to reorganize her disk contents, for better future disk performance.

a. What kind of reorganization could this tool do, to try and improve future disk performance? Assume that the filesystem remains the same. [3]

b. Assuming best possible implementation, what is the approximate time it will take to run such a tool on a disk with a filesystem with 100GB worth of files, in the worst case? Assume that you have 500GB RAM in your system. Show how you arrived at your answer. Does your answer depend on the filesystem being used? [4]

10. Consider the following sequence of shell commands on the Linux/ext3 filesystem:

```
$ echo hello > a &
$ echo world > b &
$ ls > c &
```

Notice that the ‘&’ symbol means that all these three commands execute concurrently, and can finish in any order. Which of the following are consistent states after a crash, assuming ext3 journaled mode? In other words, which of these filesystem states are possible after a crash on Linux ext3 journaled mode? Explain briefly. [6]

1. File ‘b’ exists, file ‘a’ exists, and file ‘c’ is empty.

2. File ‘b’ exists, file ‘a’ does not exist, and file ‘c’ contains two entries (for ‘a’ and ‘b’).

11. Give examples (qualitatively) of situations when :

a. Big reader locks perform better than reader-writer locks [2.5]

b. Reader-writer locks perform better than big reader locks [2.5]