Name: KEY

Operating Systems
Fall 2014
Test 3
December 03, 2014

Closed books, notes, cell phones, PDAs, iPods, laptops, etc. No headphones, please. You may use a simple calculator.

You have 75 minutes to solve 7 problems. You get 10 points for making it all the way to the bitter end. As with any exam, you should read through the questions first and start with those that you are most comfortable with. If you believe that you cannot answer a question without making some assumptions, state those assumptions in your answer.

Partial credit will be offered only for meaningful progress towards solving the problems.

Please read and sign below if you agree with the following statement:

In recognition of and in the spirit of the academic code of honor, I certify that I will neither give nor receive unpermitted aid on this exam.

Signature:_____

0	10/10
1	/10
2	/15
3	/10
4	/10
5	/15
6	/15
7	/15
Total	/100

1. [10 points] Short attention span:

   a. A _____ occurs when multiple processes or threads read and write data items so that the final result depends on the order of execution of instructions in the multiple processes.

      A) atomic operation B) race condition
      C) livelock D) deadlock

   b. A situation in which a runnable process is overlooked indefinitely by the scheduler, although it is able to proceed, is _____ .

      A) mutual exclusion B) deadlock
      C) starvation D) livelock

   c. True or False: Deadlock avoidance is more restrictive than deadlock prevention.

   d. The strategy of deadlock _____ is to design a system in such a way that the possibility of deadlock is excluded.

      A) prevention B) detection
      C) diversion D) avoidance

   e. The _____ condition can be prevented by requiring that a process request all of its required resources at one time and blocking the process until all requests can be granted simultaneously.

      A) mutual exclusion B) hold and wait
      C) circular wait D) no preemption

   f. True or False: RAID is a set of physical disk drives viewed by the operating system as a single logical drive.

   g. An inode is a control data structure that contains file attributes and pointers to disk.

   h. True or False: In Unix Fast File System, a new inode is created every time an existing file is opened.

   i. The Unix File System keeps track of all files in an inode table.

   j. True or False: fsck needs to know exactly the structure of the file system.

2. [15 points] For the Fast File System, what reads and writes for inodes and blocks would occur to create a new file `/foo/sparse` and write the blocks 1 and 2,000,000 of that file? Assume that inodes have 11 direct pointers, 1 indirect pointer, 1 double-indirect pointer, and 1 triple-indirect pointer, and assume 4KB blocks with 4-byte block pointers.

Important points here:

- Updating the free block bitmap; updating the inode bitmap when allocating a new inode for the file "sparse".
- Block 2,000,000 belongs to the triply indirect pointer.
- Since just created (thus, in memory) no point to read the newly created inode for sparse.
- When writing the block 2,000,000 the file system needs to allocate the intermediate blocks indirectly pointing to the data for block 2,000,000.

3. [10 points] Consider a variation of the Dining Philosophers problem where all unused forks are placed in the center of the table and any philosopher can eat with any two forks. One way to prevent deadlock in this system is to provide sufficient resources. For a system with n philosophers, what is the minimum number of forks that ensures deadlock freedom? Why?

   Note: The original Dining Philosophers problem is stated as follows: there are five "philosophers" sitting around a table. Between each pair of philosophers is a single fork (and thus, five total). The philosophers each have times when they think, and don't need any forks, and times where they eat. In order to eat, a philosopher needs two forks, both the one on their left and the one on their right.

n+1 will ensure that there is no circular wait, as one philosophers (at least) will have both forks, thus will not close the cycle by waiting for another resource.

4. [10 points] For the rotating parity RAID (RAID 5) in figure below, suppose you update data block 8. What disk operations must occur?

Disk 0	Disk 1	Disk 2	Disk 3	Disk 4
0	1	2	3	P0
5	6	7	P1	4
10	11	P2	8	9
15	P3	12	13	14
P4	16	17	18	19

A read to block 8 (needed for updating P2)
A write to block 8
A read to P2 (needed for updating P2 as a function of old block 8, new block 8 and old P2)
A write to P2

Alternatively (conceptually possible, typically not used):
Write block 8
Read blocks 9, 10, 11
Write block P2 as a function of blocks 9, 10, 11 and new block 8.

5. [15 points, 5 each]: Consider the following sequence of disk track requests: 27, 129, 110, 186, 147, 41, 10, 64, 120. Assume that the disk head is initially positioned over track 100 and is moving in the direction of decreasing track number. Analyze the FIFO, Elevator and SSTF algorithms in terms of number of tracks traversed. Show your work.

Most of you got it write. Note that the direction in which the head moves does not matter for FIFO or SSTF, but only for elevator.

6. [15 points] Your clumsy roommate trips over the power cord of your desktop while the file system was performing an append to a file.

    a. Describe the scenarios that could result from this unfortunate event that would jeopardize the consistency of your file system (and your relationship with your roommate).
    b. Does `fsck` fix the problem? How? Explain your answer.

Detailed scenarios expected here, that would include failures at different points when updating the free block list, the inode of the file being appended, and the data block being modified. Also, fsck works for some cases and not for others. Journaling is not assumed in this scenario.

7. [15 points] Below is an attempt to solve the producer-consumer problem. Is this a correct solution? If yes, what is achieved and how? If not, what is the problem? Explain your answer in either case. (Note: Assume no syntax or compilation errors. You're asked about the concurrency logic/correctness and the use of semaphores for this.)

```
1   sem_t empty;
2   sem_t full;
3   sem_t mutex;
4
5   void *producer(void *arg) {
6       int i;
7       for (i = 0; i < loops; i++) {
8           sem_wait(&mutex);           // line p0 (NEW LINE)
9           sem_wait(&empty);           // line p1
10          put(i);                     // line p2
11          sem_post(&full);            // line p3
12          sem_post(&mutex);           // line p4 (NEW LINE)
13      }
14  }
15
16  void *consumer(void *arg) {
17      int i;
18      for (i = 0; i < loops; i++) {
19          sem_wait(&mutex);           // line c0 (NEW LINE)
20          sem_wait(&full);            // line c1
21          int tmp = get();            // line c2
22          sem_post(&empty);           // line c3
23          sem_post(&mutex);           // line c4 (NEW LINE)
24          printf("%d\n", tmp);
25      }
26  }
27
28  int main(int argc, char *argv[]) {
29      // ...
30      sem_init(&empty, 0, MAX); // MAX buffers are empty to begin with...
31      sem_init(&full, 0, 0);    // ... and 0 are full
32      sem_init(&mutex, 0, 1);   // mutex=1 because it is a lock (NEW LINE)
33      // ...
34  }
```

Deadlock, of course. (Incorrect) Solution discussed in class.