Operating Systems **Final Exam** ECE344, Fall 2013

Duration: 2 hour 30 min

Examiner: A. Goel

Please fill your student number and name below and then read the instructions below carefully.

Student Number: |_|_|_|_|_|_|_|_|_|

Last Name: |_|_|_|_|_|_|_|_|_|_|_|_|_|_|

First Name: |_|_|_|_|_|_|_|_|_|_|_|_|_|_|

Instructions

Examination Aids: No examination aids are allowed.

Do not turn this page until you have received the signal to start.

Do not remove any sheets from this test book. Answer all questions in the space provided. No additional sheets are permitted. Use the last blank page as scratch space. This page will not be marked.

This exam consists of 8 questions on 16 pages (including this page). The value of each part of each question is indicated. The total value of all questions is 120.

For the written answers, explain your reasoning clearly. Be as brief and specific as possible. Clear, concise answers will be given higher marks than vague, wordy answers. Marks will be deducted for incorrect statements in an answer. Please write legibly!

Work independently.

MARKING GUIDE

Q1:	_______	(10)
Q2:	_______	(10)
Q3:	_______	(10)
Q4:	_______	(20)
Q5:	_______	(18)
Q6:	_______	(20)
Q7:	_______	(12)
Q8:	_______	(20)
TOTAL:	_______	(120)

To the red pill, and the students who spent countless hours
in the rabbit-hole in Wonderland.

Question 1. **True / False** [10 MARKS]

TRUE	FALSE	A user-space scheduler needs to use a system call to implement blocking synchronization.
TRUE	FALSE	A user-space scheduler uses 2 kernel threads to multiplex 5 user-level threads. When a user-level thread makes a blocking system call, all of the other user-level threads will be blocked as well.
TRUE	FALSE	When a process issues a system call, the OS code starts by executing an instruction to change the processor mode (from user to kernel).
TRUE	FALSE	The OS has a bug if its exception handler code blocks (e.g., goes to sleep) for any reason.
TRUE	FALSE	A scheduler favoring I/O-bound processes does not significantly delay the completion of CPU-bound processes.
TRUE	FALSE	On the x86 architecture, if a given memory reference (load or store) causes a TLB miss, then that memory reference also causes a page fault.
TRUE	FALSE	The paging daemon does not eliminate the need to run a page replacement algorithm during a page fault.
TRUE	FALSE	The Elevator disk scheduling algorithm bounds the waiting time for all disk requests.
TRUE	FALSE	In a system with vast amounts of memory (e.g., greater than the disk size), a file system will issue disk requests initially to read all data from disk and then never issue disk requests.
TRUE	FALSE	An entry in the open file table maintains a pointer to an in-memory inode structure.

Question 2. **OS Design** [10 MARKS]

Part (a) [3 MARKS] For the `write` system call shown below, list 3 unique checks that the OS must perform to ensure that an application does not crash or corrupt the OS.

```
int write(int fd, const void *buf, size_t count);
```

Check 1	
Check 2	
Check 3	

Part (b) [3 MARKS] The timer interrupt is a key mechanism used by the OS. Usually, it waits some amount of time (say 10 milliseconds) and then interrupts the CPU. Suppose, the interrupt is not based on time but rather based on the number of TLB misses the CPU encounters. Once a certain number of TLB misses take place, the CPU is interrupted and the OS gets control (e.g., the page fault frequency replacement algorithm might be implemented this way). Will this design affect the timer interrupt and its usefulness?

Part (c) [4 MARKS] In a Unix file system, the `/A/B/C/D.txt` file uses one block. Assume that each directory in this file system uses one block as well. To read the file `D.txt`, list the maximum and the minimum number of disk reads for different types of blocks (e.g., inode, data, etc.). In both cases, assume that the buffer cache is initially empty.

Maximum	
Block Type	# of reads

Minimum	
Block Type	# of reads

Question 3. **Synchronization** [10 MARKS]

A user has written the snippet of code shown below. Assume that a main thread (code not shown) invokes `Init_thread` once and `Worker_thread` multiple times. All these threads are invoked concurrently.

Use blocking locks and condition variables to ensure that there are no races in this code and to ensure two conditions: 1) the initialization code ($x = 0$) in `Init_thread` runs before x is incremented ($x = x + 1$) by any `Worker_thread`, 2) after 10 worker threads have each incremented x, `Init_thread` calls `exit` so that the program exits.

Make sure to clearly declare and initialize any synchronization variables. You will not need any additional non-synchronization variables. Since you are using blocking synchronization primitives, do not use `sleep`, `wakeup`, tight loops, or interrupt disabling.

```
int x = -1;




Init_thread() {




        x = 0;




        exit(0);


}

Worker_thread() {




        x = x + 1;




}
```

Question 4. **Multiprocessor Synchronization** [20 MARKS]

We have seen how the semaphore primitives P and V are implemented for uniprocessors. In this problem, we will implement these primitives for multiprocessors. The relevant code for the uniprocessor version of these primitives is shown below. The `schedule()` function removes the current thread from the run queue and chooses another thread to run.

```
P(sem) {
    spl = splhigh();
    while (sem->count==0) {
        thread_sleep(sem);
    }
    sem->count--;
    splx(spl);
}
```

```
V(sem) {
    spl = splhigh();
    sem->count++;
    thread_wakeup(sem);
    splx(spl);
}
```

```
thread_sleep(void *cond) {
    add_to_wait_queue(cond);
    schedule();
}
```

```
thread_wakeup(void *cond) {
    thread = remove_from_wait_queue(cond);
    add_to_run_queue(thread);
}
```

Part (a) [16 MARKS] In the four questions shown below, circle all (one or more) correct answers. Be careful, two marks will be deducted for each incorrect answer. In the answers below, "corrupt" means an incorrect update.

1) Would the semaphore code shown above work for multiprocessors?

1. No, it could corrupt the `sem->count` variable.
2. No, it could corrupt the wait queue.
3. No, it could corrupt the run queue.
4. No, it could cause a thread to wait forever.
5. Yes, it would work.

2) Your partner argues that the code would clearly not work for multiprocessors and suggests a minimal change: add spin locks in the `thread_sleep` and `thread_wakeup` implementation:

```
thread_sleep(void *cond) {
    spin_lock(schedlock);
    add_to_wait_queue(cond);
    schedule();
    spin_unlock(schedlock);
}
```

```
thread_wakeup(void *cond) {
    spin_lock(schedlock);
    thread = remove_from_wait_queue(cond);
    add_to_run_queue(thread);
    spin_unlock(schedlock);
}
```

Would the semaphore code shown earlier now work for multiprocessors?

1. No, it could corrupt the `sem->count` variable.
2. No, it could corrupt the wait queue.
3. No, it could corrupt the run queue.
4. No, it could cause a thread to wait forever.
5. Yes, it would work.

3) Your friend from Barkly University argues that the code would clearly still not work for multiprocessors and suggests another minimal change: replace interrupt disabling with spin locks in the `P` and `V` implementation:

```
P(sem) {
    spin_lock(sem->spinlock);
    while (sem->count==0) {
        thread_sleep(sem);
    }
    sem->count--;
    spin_unlock(sem->spinlock);
}
```

```
V(sem) {
    spin_lock(sem->spinlock);
    sem->count++;
    thread_wakeup(sem);
    spin_unlock(sem->spinlock);
}
```

Would this semaphore code work for multiprocessors?

1. No, it could corrupt the `sem->count` variable.
2. No, it could corrupt the wait queue.
3. No, it could corrupt the run queue.
4. No, it could cause a thread to wait forever.
5. Yes, it would work.

4) Your partner looks at the code, mutters something about Barkly, and suggests adding `spin_unlock` and `spin_lock` around `thread_sleep` in the `P` implementation:

```
P(sem) {
    spin_lock(sem->spinlock);
    while (sem->count==0) {
        spin_unlock(sem->spinlock);
        thread_sleep(sem);
        spin_lock(sem->spinlock);
    }
    sem->count--;
    spin_unlock(sem->spinlock);
}
```

You are convinced this code will not work for multiprocessors?

1. No, it could corrupt the `sem->count` variable.
2. No, it could corrupt the wait queue.
3. No, it could corrupt the run queue.
4. No, it could cause a thread to wait forever.

Part (b) [4 MARKS] You know that the previous code is almost correct, but it does cause problems occasionally. How would you go about fixing the problem? Use the bullet points shown below to state your solution. Be as precise as possible. Hint: the implementation of the semaphore primitives should be tightly coupled with the sleep/wakeup primitives.

1.

2.

3.

4.

Question 5. TLB [18 MARKS]

Part (a) [4 MARKS] An early RISC processor uses a software-managed TLB with a 4 KB page size, and 4 byte integer size. It only provides a single control bit in the TLB entry. This control bit is the valid bit. You wish to implement swapping on this processor and would like to track referenced and dirty pages. Can you simulate each of these bits in the OS? If so, explain how. If not, explain why. If you make any assumptions, state it clearly below.

Bits	Yes/No	How/Why
Referenced		
Dirty		

Part (b) [4 MARKS] For the processor described above, consider the following code snippet:

```
int a[4096];
int b[4096];

int simple() {
    int c = 0;
cswitch:    /* <- context switch takes place */
    for (i = 0; i < 4096; i++) {
        c = a[i] + b[i];
    }
    return c;
}
```

How many TLB misses will take place when the `simple()` function is run for the first time? Ignore the context switch comment in the code. Provide a justification for each TLB miss. State any assumptions that you make.

Part (c) [2 MARKS] Now suppose a context switch occurs at the `cswitch:` label in **Part (b)**. At this context switch, the TLB is flushed completely. Will there be additional TLB misses when this code is run? If so, how many? Clearly explain why.

Part (d) [2 MARKS] Suppose you decide to use a linear (single-level) page table in your OS for the processor described earlier. How many memory accesses would occur to the page table when the code shown in **Part (b)** is run without any context switches?

Part (e) [4 MARKS] Now let's imagine that the processor didn't have a TLB and accessed your linear page table directly when translating addresses. How many memory accesses would occur to the page table when the code shown in **Part (b)** is run without any context switches? Circle the best answer below and provide a justification.

1. Same as the answer to **Part (d)**
2. Somewhere between 4K-8K accesses
3. Somewhere between 8K-12K accesses
4. Somewhere between 12K-16K accesses
5. Somewhere between 16K-20K accesses
6. More than 20K accesses

Part (f) [2 MARKS] Now suppose a context switch occurs at the `cswitch:` label. Would your answer in **Part (e)** be the same or not? Why?

Question 6. Page Tables [20 MARKS]

Suppose that a computer system consists of a processor that uses 50 bit virtual addresses and its paging hardware supports 16 KB page frames. The size of each page table entry is 4 bytes. Each entry has a valid bit, a referenced bit, a dirty bit and two bits for protection that are located in the high order bits of the entry. Please answer the questions shown below. If a question below asks you to calculate a size, show the size in a human readable unit (e.g., KB=2^{10} bytes, MB=2^{20} bytes, GB=2^{30} bytes or TB=2^{40} bytes), unless indicated otherwise.

Part (a) [3 MARKS] If the paging hardware uses a single-level page table, calculate the total size of the page table.

Part (b) [3 MARKS] Calculate the maximum size of physical memory that can be supported by this system.

Part (c) [4 MARKS] You realize that a single-level page table is not the best design for this processor. Design a multi-level page table for the paging hardware by answering the following questions: 1) how many levels does your page table design need, 2) how many page table entries will each level have? Ensure that each page table fits within a page frame.

Part (d) [2 MARKS] Suppose a process is using 4 MB of memory and no pages are swapped to disk. What is the minimum amount of memory that your multi-level page table could be using at this time? Show your answer in pages.

Part (e) [4 MARKS] Suppose a process is using 4 MB of memory and no pages are swapped to disk. What is the maximum amount of memory that your multi-level page table could be using at this time? Show your answer in pages.

Part (f) [4 MARKS] Given the pages being accessed by the process in **Part (e)**, how would you redesign your page table to reduce the memory needed for page tables?

Question 7. **Disks** [12 MARKS]

The shortest-access-time-first (SATF) scheduler is a variant of the shortest-seek-first (SSF) algorithm. It picks the disk IO request from the IO scheduler queue that has the shortest access time from the current head position and services this request. In this question, we'll perform some simple calculations on a highly-simplified disk.

Part (a) [3 MARKS] Assume a simple disk that has only a single track, and a simple FIFO scheduling policy. The rotational delay on this disk is R. There is no seek cost (only one track!), and transfer time is so fast that we just consider it to be free. What is the (approximate) worst case execution time for three (3) requests (to different blocks)?

Part (b) [3 MARKS] Now assume that a SATF scheduler is being used by the disk (but it still only has a single track). What is the worst-case time for three requests (to different blocks) now?

Part (c) [3 MARKS] Now assume the disk has three tracks. The time to seek between two adjacent tracks is S; it takes twice that to seek across two tracks (e.g., from the outer to the inner track). Given a FIFO scheduler, what is the worst-case time for three requests?

Part (d) [3 MARKS] The same question above, but for a SATF scheduler.

Question 8. **Feature Full File System (FFFS)** [20 MARKS]

You have just joined a storage startup that has designed a high-end, one petabyte (1024 TB = 2^{50} bytes) storage array for a modern data center. You are in charge of designing a modern, feature-full file system (FFFS) for this storage system. To do so, you start with the traditional Unix file system (UFS) and address many of its limitations.

Part (a) [3 MARKS] Assume the standard UFS supports an 8KB block size, and an inode has 12 direct pointers, one single indirect block pointer, and one double indirect block pointer. Assume also that each pointer is 32 bits in size. What is the approximate maximum file size supported in UFS? State your number in the most appropriate human readable unit (e.g., MB, TB).

Part (b) [3 MARKS] In FFFS, we want to support files that can be as large as 128 TB. What would be the minimal changes you would make to UFS to support these huge files.

Part (c) [2 MARKS] In UFS, the number of inodes that can be allocated is fixed when the file system is first created. Describe why it is fixed (i.e., describe how the allocation/deallocation of inodes is managed).

Part (d) [6 MARKS] In FFFS, we want to support growing the number of inodes over time. Describe the changes you would make to UFS to implement this feature, as precisely as possible.

Part (e) [6 MARKS] In UFS, the file system is created in a fixed size partition or disk. In FFFS, we would like to add new 1 PB storage arrays to the file system dynamically (e.g., starting with a 1 PB file system, grow it to a 8 PB file system over time). Describe the changes you would make to UFS to implement this feature, as precisely as possible.

[Use the space below for rough work.]