COMS W4118 Final
May 15, 2008

Prof. Erich Nahum

This exam is closed book and closed notes. No sharing of material and no electronic equipment other than a calculator is allowed.
Write all answers in the space provided. If you run out of space use the back of the page. If you need scratch paper, ask a TA or the professor.
Exam starts at 9:00 AM and ends at 11:59 AM.
Total number of points is 100.
Write your name and email address on the top of any *loose* page.
It is recommended that you read the entire exam before proceeding.
SOME QUESTIONS ARE FOR *COMPS TAKERS ONLY.*
SOME QUESTIONS ARE FOR *4118 STUDENTS ONLY.*
V2 *Plan your time.*

Name: _____

CUID: _____

1 (10)	2 (10)	3 (15)	4 (10)	5 (10)	6 (10)	7 (10)	8 (10)	9 (05)	10 (10)	11 (05)	Total(100)

1. (10 points) For each of the following, mark **TRUE** or **FALSE**. No justification is necessary (and no partial credit will be given for wrong answers). (1 point each)

   (a) Deadlock can be prevented in dining philosophers by reducing the number of diners that are allowed to eat at the same time by one. **FALSE**

   (b) On a uniprocessor system, the critical section problem can be solved simply by disabling interrupts while a shared variable is being modified. **TRUE**

   (c) Threads are generally more lightweight than processes because they share address spaces while processes do not. **TRUE**

   (d) LRU is generally used for page replacement because it performs at least as well as FIFO for all possible memory reference strings. **FALSE**

   (e) Imprecise interrupts are easier for an OS to implement correctly than precise interrupts. **FALSE**

   (f) Hardware locking primitives can only be utilized in kernel mode. **FALSE**

   (g) When an interrupt occurs, the operating system needs to save all general-purpose registers. **TRUE**

   (h) IPC using messages tends to be faster than shared memory. **FALSE**

   (i) Shared libraries are more difficult to implement on a segmentation-based architecture. **FALSE**

   (j) In general, accessing a file via a hard link is usually faster than accessing via a soft link. **TRUE**

2. (15 points) Synchronization. The goal of this exercise is to implement a solution to a variant of the Dining Philosophers problem. In this variant, rather than chopsticks being placed between philosophers, they are all in a pile in the center of the table.

   Using *only semaphores*, solve this problem. *Use* semaphores, do not *implement* semaphores. Your solution *may not* use locks, monitors, or any other synchronization primitives.

   Create a function `dine()`, which waits until a diner has two chopsticks and can eat, then calls `eat()`, and then releases the chopsticks before returning. Your solution should allow multiple philosophers to eat at the same time (as long as there are sufficient chopsticks). Assume you are given the variables `chopsticks` and `philosophers`, which are initialized to the total number of chopsticks and philosophers available. Your solution should avoid deadlock.

   Simplest solution:

```
Semaphore sticks = new Semaphore(floor(chopsticks/2));
Dine() {
sticks.P(); // This acquires two chopsticks
Eat();
sticks.V(); // This releases two chopsticks
}
```

   Another solution:

```
Semaphore Lock = new Semaphore (1); // mutex
Semaphore Sem1 = new Semaphore (chopsticks); // scheduling
Dine {
Lock.P(); // This acquires mutex
Sem1.P(); // This acquires one chopstick
Sem1.P(); // This acquires one chopstick
Lock.V(); // This releases mutex
Eat();
Sem1.V(); // This releases one chopstick
Sem1.V(); // This releases one chopstick
}
```

3. (10 points) Paging.

   Consider a system with a 32-bit logical address space, a two-level paging scheme, 4 byte page table entries, and 4 KB pages. A 32-entry TLB is present with a 95% average hit rate and 10 ns access time. RAM access time is 100 ns and you should assume that all page tables are already stored in RAM. RAM is byte addressable and physical frames are numbered starting from 0.

   (a) (2 pts) How many address bits are needed for the page offset?

   Since pages are 4 KB, 12 bits (2^{12} == 4 KB).

   (b) (2 pts) How many address bits are used for indexing into the outer-level page table?

   Page size is 4 KB, or 12 bits, so that leaves 20 bits of addressing to find the page. The inner page directory thus must have 2^{20} entries. Each entry is 4 (2^2) bytes, which means 1024 of them fit on a 4 KB page. That means there must be $2^{(20-10)}$ pages of inner page table entries. Thus the first 10 bits of the virtual address are used to index into the outer page table.

   (c) (3 pts) If the logical address 10027 corresponds to a physical location on frame number 438, what is the corresponding physical address?

   10027 mod 4096 equals 1835 (the offset into the frame).
   Frame 438 is at address 1794048 (438 * 4096).
   1795883 plus 1835 is 1795883.

   (d) (3 pts) What is the effective average memory access time?

   Effective memory address time equals
   (hit ratio * hit time) + (miss ratio * miss time) =
   (.95 * 110 ns) + (0.05 * 310 ns) =
   (104.5 + 15.5) =
   120 ns.

4. (10 points) Virtual Memory.

   (a) (2 pts) Most virtual memory systems have a modified and a referenced bit in the page table. Explain why these should not be implemented as counters.

   The OS is only looking at these fields at clock ticks say, 100 – 1000 times/second. A modern CPU can execute a lot of instructions in that time, which means that we'd need a fairly large counter. The counter would have to be read from the TLB or page table, incremented, and rewritten; bit fields are simply set to 1. The extra time for this would make the process very much more expensive. Remember that any sort of memory reference is very expensive, plus we have to worry about the cost of the addition.

   (b) (2 pts) Consider the following two program fragments. Which one is more stressful on the VM system? Why?

```
#define N 10000000
int a[N], b[N], i;
for (i=0; i < N; i++)
    a[i] = b[i];
```

```
#define N 10000000
int i;
struct x {
    int a, b;
} y[N];
for (i=0; i<N; i++)
    y[i].a = y[i].b;
```

   The second is much friendlier, since each assignment statement references one page instead of two.

   (c) (6 pts) You have a system with four page frames. Suppose the page reference string is 4, 1, 7, 2, 4, 1, 6, 7, 0, 1, 5, 2, 6 and LRU is used. What does memory look like after each page reference?

4	4	4	4	4	4	4	4	**0**	0	0	0	**6**
	1	1	1	1	1	1	1	1	1	1	1	1
		7	7	7	7	**6**	6	6	6	**5**	5	5
			2	2	2	2	**7**	7	7	7	**2**	2

   Page replacements marked **in bold**.

5. (10 points) File Systems Implementation

   In this question, we consider a non-standard Unix file system, which instead of using inodes, instead stores the usual inode information in the directory entry for that file. We call our new system the Inode-Free File System, or IFFS.

   (a) (2 pts) Of the following, which are usually found in a *standard Unix inode*? Circle all that apply:

      i. *Direct pointers to data blocks*
      ii. The name of the file
      iii. *Some statistics about when the file has been accessed, updated, etc.*
      iv. The inode number of the file's parent directory
      v. The current position of the file pointer

   (b) (2 pts) In a standard Unix file system, how many disk reads would it take to read a single block from the file `/this/path/is/toolong` from disk? (assume nothing except the superblock is cached, i.e., everything starts on disk, and that each directory name fits in a block). Show reads in order to maximize partial credit.

   Read inode block for /, read directory block for /,
   read inode data for this/, read directory for this/,
   read inode data for path/, read directory data for path/,
   read inode data for is/, read directory data for is/,
   read inode data for toolong/, read file data for toolong,
   total of 10 reads.

   (c) (2 pts) Now compare this to the number of reads it would take to read a single block from the same file `/this/path/is/toolong` in IFFS. (same assumptions as above)

   01: Read directory block for /
   02: Read directory block for this/
   03: Read directory block for path/
   04: Read directory block for is/
   05: Read file data for toolong
   total of 5 reads.

   (d) (2 pts) IFFS doesn't support hard links. Why do you think so? (explain)

   A hard link is a pointer to an inode. Hard links only make sense if multiple directories can point to the same inode. No inodes, no hard links.

6. (10 points) Storage

   (a) (2 pts). What is RAID? Give a brief description (2-3 sentences)

   Redundant Array of Inexpensive/Independent Disks. A technology that employs the simultaneous use of two or more hard disk drives to achieve greater levels of performance and/or reliability. Data is striped across multiple disks in various combinations giving different tradeoffs of protection against data loss, capacity, and speed.

   (b) (8 pts). Disk Scheduling. Disk requests come into the disk device driver for cylinders: 10, 22, 20, 2, 40, 6, 38, in that order. The disk has 60 total cylinders and the disk head is currently positioned over cylinder 20. A seek takes 10 milliseconds per cylinder moved. What is the sequence of reads and total seek time using each of the following algorithms?

      i. (2 pts) First-come, first-served:

      10, 22, 20, 2, 40, 6, 38
      10 + 12 + 2 + 18 + 38 + 34 +32 = 146 cylinders = 1460 milliseconds.

      ii. (3 pts) Shortest Seek Time First:

      20, 22, 10, 6, 2, 38, 40
      0 + 2 + 12 + 4 + 4 + 36 + 2 = 60 cylinders = 600 milliseconds.

      iii. (3 pts) C-SCAN (initially moving upwards):

      20, 22, 38, 40, 10, 6, 2
      0 + 2 + 16 + 2 + 30 + 4 + 4 = 58 cylinders = 580 milliseconds.

7. (10 points) I/O.

   (a) (2 pts). What is programmed I/O?

   Programmed input/output (PIO) is a method of transferring data between memory and a peripheral such as a network adapter or disk drive. In PIO, the processor copies data explicitly, rather than assigning another device to perform the copying, which is the case with DMA.

   (b) (2 pts). What is memory-mapped I/O?

   Memory-mapped I/O is when a device's control status registers are mapped into regular memory. Controlling the device is done by regular reads and writes rather than explicit I/O instructions.

   (c) (2 pts). Name an example where polling would be preferable to an interrupt (and why).

   Interrupts are expensive and thus they only make sense if enough other work can be done between submitting the request and when it is available. Polling (busy-waiting) makes sense if the device will have data in less time than would be spent in servicing an interrupt. Two examples are: initializing a device (programming a control status register and busy waiting until it responds with a ready status), and retrieving subsequent packets from a network device after the first packet has generated an interrupt.

   (d) (4 pts). Outline the steps in the "life cycle" the OS needs to take to satisfy an I/O request in the form of a `read` system call. Assume the content is not cached in memory. (You do not need to discuss file systems details).

   A process makes a *read system call*. The OS determines the data is *not in memory*. The OS allocates a buffer to hold the data and pins it in memory. The OS submits the request to the device with a pointer to the allocated buffer and *blocks the process*, taking it off the run queue, marking it blocked, and placing the PCB on a device-specific list. The OS context switches to another process to run. The device *handles the request* (e.g., does a disk seek and transfer). The device *DMA's the data into the buffer*. The device *generates an interrupt*. The OS *services the interrupt*. The OS determines the block is for a particular process, The OS *unblocks the process*, marking it runnable. Eventually the *scheduler runs the process*, which will then *return from the system call*.

8. (5 points) Interrupts in Linux. Each question requires only 1-2 sentences to answer.

   (a) (1 pt). What is the difference between an IRQ and a vector?

   An IRQ is an interrupt request line. It has a number assigned to it (say by the PCI bus), but the programmable interrupt controller (PIC) can map that number to another number. The second number is the the vector, which is what the CPU sees. Sometimes this mapping is 1:1, sometimes it's not.

   (b) (1 pt). What is the difference between a trap and an exception?

   Both are synchronous and user-generated. Traps are intentional (e.g., breakpoints). Exceptions are unintentional (e.g., divide by zero).

   (c) (1 pt). Which kernel stack is used when an exception occurs?

   The kernel stack of the process causing the exception.

   (d) (1 pt). How can Linux support device sharing even if the device doesn't support it?

   By time-sharing the vector used by the interrupt service routine. When a device is unloaded, another device can use that IRQ/vector.

   (e) (1 pt). Can work queues sleep? Why or why not?

   Yes. They have their own kernel stacks.

9. (5 points) Linux Synchronization. Each question requires only 1-2 sentences to answer.

   (a) (1 pt). What is a memory barrier?

   A memory barrier is a command to the hardware and compiler to not re-order any instructions, reads or writes (depending on the barrier type).

   (b) (1 pt). What is an atomic operation? Give an example.

   An atomic operation is one that performs its operation atomically. This ensures updates are not lost due to race conditions. For example, `atomic_inc()` atomically increments a variable. Two processes executing `atomic_inc` are guaranteed to increment the variable by two, whereas if they used a normal increment, one increment might get lost.

   (c) (1 pts). Linux makes available spin lock synchronization methods that disable interrupts. Why would you want to use them, rather than the default ones (which do not disable)?

   So that the task is not preempted by an interrupt while it holds the spin lock.

   (d) (2 pt) Can writers starve in RW spin locks? RW semaphores?

   Spin locks: yes. Semaphores: No.

10. (10 points) Linux Scheduling. THESE QUESTIONS ARE FOR W4118 STUDENTS ONLY.

    (a) (2 pts). The Linux scheduler has two run queues. Why?

    To enable the scheduler to run in $O(N)$ time. The run queues are active and expired. As each task is exhausts its time slice, it is moved to the other run queue. When all the tasks are exhausted, the queues are swapped.

    (b) (2 pts). What are the Linux scheduling policies?

    There are three: SCHED_RR (real-time), SCHED_FIFO (real-time), and SCHED_OTHER (everything else).

    (c) (2 pts). What does `free_uid` do? Why do you need to call it?

    The function decrements the ref count on a user structure. When a function looks up a user, the ref count is incremented so that it cannot be freed out from under it. `free_uid` decrements the ref count to indicate the process is finished with the structure, and potentially frees it if the ref count goes to zero.

    (d) (4 pts). What does `scheduler_tick` do? How frequently is it called? Outline some pseudocode for it.

    Only Taek knows for sure.

11. (5 points) Linux File Systems. THESE QUESTIONS ARE FOR W4118 STUDENTS ONLY. Each question (except the last) requires only 1-2 sentences to answer.

    (a) (1 pt). What is a dentry?

    The in-memory representation of a path name.

    (b) (1 pt). What is a superblock?

    Global file system data.

    (c) (1 pt). Why does Ext2 divide the file system into block groups?

    Block groups are an approximation of cylinder groups. The file system tries to group related files onto the same cylinder for performance reasons (to minimize disk seeks).

    (d) (2 pts). Ext2 uses 32 bit pointers to identify blocks. An inode has 12 direct blocks, one indirect, one double-indirect, and one triple indirect. Assume a block is 4 KB. How large can an Ext2 file be?

    A block can hold 1024 entries (4 KB divided by 32 bits or 4 bytes).
    Direct space: 12 * 4 KB = 48 KB.
    Single indirect block: 1024 * 4 KB = 4 MB. Double indirect block: 1024 * 1024 * 4 KB = 4 GB. Triple indirect block: 1024 * 1024 * 1024 * 4 KB = 4 TB. Total is 4 KB + 4 MB + 4 GB + 4 TB.

12. (10 points) CPU Scheduling. THIS QUESTION IS FOR COMPS TAKERS ONLY.

Here is a table of processes and their associated running times. All of the processes arrive in alphabetical order at time 0.

Process ID	CPU Time
Process A	2
Process B	6
Process C	1
Process D	4
Process E	3

Show the scheduling order for these processes under 3 policies: First Come First Serve (FCFS), Shortest-Remaining-Time-First (SRTF), Round-Robin (RR) with timeslice quantum = 1. For RR, assume new tasks are placed at the end of the queue. Assume context switch cost is zero, and that no jobs do any I/O. For each process in the schedule above, indicate the waiting time in the queue and the turnaround time. Show all your work to maximize any partial credit.

Time	FCFS	SRTF	RR
0	A	C	A
1	A	A	B
2	B	A	C
3	B	E	D
4	B	E	E
5	B	E	A
6	B	D	B
7	B	D	D
8	C	D	E
9	D	D	B
10	D	B	D
11	D	B	E
12	D	B	B
13	E	B	D
14	E	B	B
15	E	B	B

Scheduler	A	B	C	D	E
FCFS wait	0	2	8	9	13
FCFS TRT	2	8	9	13	16
SRTF wait	1	10	0	6	3
SRTF TRT	3	16	1	10	6
RR wait	4	10	2	10	9
RR TRT	6	16	3	14	12

13. (5 points) Processes and Threads. THESE QUESTIONS ARE FOR COMPS TAKERS ONLY.

    (a) (1 pt). How does the OS preempt a process or thread?

    By some interrupt going off, usually a timer but it could be another interrupt such as a disk or network interface card.

    (b) (1 pt). A Web server uses user level threads multiplexed on top of a single kernel thread. What are two problems with this?

    First, it doesn't exploit multiple processors. Second, if any thread does any blocking I/O call (or page fault), the whole process is blocked, rather than just the thread.

    (c) (3 pts). What are the advantages and disadvantages of user threads vs. kernel threads?

    UT pros: smaller size, cheaper context switching, can create more.
    UT cons: less protection, no MP exploitation, process can block, harder to use.
    KT pros: better protection, MP exploitation, threads can block, easier to use.
    KT cons: larger size, more expensive context switching, can't create as many.