

CSC 469H1 F
ADVANCED OPERATING SYSTEMS

UNIVERSITY OF TORONTO
Fall 2006

Term Test #1

NO AIDS ALLOWED

Please **PRINT** in answering the following requests for information:

Family Name: _____

Given Names: _____

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

Login (@cdf): _____

Notes to students:

1. This test lasts for 110 minutes and consists of 80 marks. Budget your time accordingly.
2. This test has 7 questions and 9 pages (including this one); Check that you have all pages before starting.
3. **Write in pen. No pencils. Really, I mean it.**
4. Write your answers on this “question and answer” paper, in the spaces provided. Be concise. In general, the amount of space provided is an upper bound on the “size” of answer that is expected. If necessary, use space where available and provide explicit pointers.
5. State your assumptions and show your intermediate work, where appropriate.
6. Do **not** go beyond here until instructed to do so. Write your student number at the top of each succeeding page once you get going.

Question	Marks
1	/15
2	/12
3	/10
4	/12
5	/9
6	/10
7	/12
Total	/80

1. [15 marks, 3 each] Definitions

Define the following terms, in the context of this course:

a) microkernel

Operating system design/structure which attempts to minimize the amount of privileged code, providing only basic abstractions such as threads (or tasks), virtual memory, and interprocess communication. Everything else is a server process running at user-level using basic facilities to provide full range of expected OS services.

b) interrupt

An event external to the currently executing process that causes a change in the normal flow of instruction execution; usually generated by hardware devices external to the CPU to indicate the device needs service. Interrupts happen asynchronously with respect to the currently-executing process.

c) queuing lock

A spinlock design in which each process/CPU spins on a different memory location to reduce memory contention; waiting processes add their lock structure to the tail of the queue and the lock holder unblocks the next process in the queue when the lock is released. Guarantees FIFO access to lock (or critical section protected by lock).

d) ABA problem

Issue with compare-and-swap (CAS) atomic operation where a memory location can be read while it holds value "A", a different value "B" is stored into that location, and then "A" is stored there again before the CAS is attempted. CAS only sees the current value, and does not know that it changed, thus may succeed when it should fail. [The problem can be illustrated with a non-blocking stack as in the lecture notes. Answers that refer specifically to the stack example are fine.]

e) processor affinity

A measure of the benefit of keeping thread scheduled on same CPU it used previously due to data already loaded into the CPU cache.

2. [12 marks; 4 each] System Virtual Machines

a) Identify 2 architecture requirements for constructing a virtual machine, and briefly explain why each is needed.

i) dual-mode operation, to prevent code in virtual machine from manipulating real machine state without control of virtual machine monitor layer

ii) a way to invoke privileged operations from non-privileged mode (e.g. a trap instruction), so that virtual machine monitor can perform privileged operations on behalf of code in virtual machine

iii) memory protection/relocation, to protect virtual machine monitor from code running in virtual machines it provides.

iv) asynchronous interrupts for I/O, so virtual machine monitor can regain control and direct data to correct virtual machine

b) What is the distinction between *sensitive* and *privileged* instructions, in the context of virtual machines?

Privileged instructions can only be executed in privileged (aka supervisor, monitor, system) mode, and are required to trap if executed outside privileged mode. Sensitive instructions affect operation of machine but are not, by definition, required to trap (e.g., "enable interrupts" may simply be a no-op if executed in user mode). An instruction set is efficiently virtualizable if all sensitive instructions are also privileged.

c) The Intel Pentium is not *efficiently virtualizable*, meaning extra work is needed to create the illusion of a virtual machine, yet several VMs exist for this architecture. Identify the strategy used by VMWare and the strategy used by Xen, and identify the primary advantage of each approach.

VMWare uses binary rewriting to replace all sensitive but non-privileged instructions in the guest OS binary with trap instructions, so that the VMM can emulate their intended effect in privileged mode. The main advantage here is that we can run ANY guest operating system in a virtual machine without requiring the OS source code.

Xen requires changes to the guest operating system (paravirtualization) to deal with cases where sensitive but non-privileged instructions are used. The main advantage is that this can be more efficient than the trap + emulation approach.

3.[10 points, 5 each] Signals

You are using a FreeBSD system, and your firefox browser window has completely stopped responding. Using the `ps` command from a shell, you have identified the pid of the browser as 1234.

a) You try to kill the browser by typing "`kill 1234`" at the shell prompt, but as far as you can tell, nothing happens. A quick check with "`man kill`" tells you that the default signal for kill is `SIGTERM`. Give two reasonable explanations for this behaviour.

i) firefox is blocking/ignoring SIGTERM so the signal is never delivered to it.

ii) firefox has installed its own signal handler for SIGTERM, which catches the signal when it is delivered, and has some behaviour different from the default and desired "end program". Whatever the SIGTERM handler in firefox does, it has no observable effect.

b) You try again, this time explicitly sending `SIGKILL` with "`kill -SIGKILL 1234`" and the window disappears. A check with `ps` verifies that the process is gone. Outline the steps taken in the OS, from receiving the kill system call from the shell process, to the point where the browser process exits.

SIGKILL cannot be caught or ignored, so this time we go through normal signal delivery.

- the kernel, handling the kill system call from the shell process, locates the process structure for pid "1234"

- the bit for signal SIGKILL is marked in the process structure for "1234" (`p_siglist` in FreeBSD, but that detail is not needed) and the process is set to run next (since we know the signal succeeded, we can surmise that if "1234" was sleeping in the kernel, it must have been an interruptible sleep, and it was awoken and set to run)

- shell process in kernel does a return from syscall, which in FreeBSD will cause rescheduling to happen first... reschedule may happen at next clock interrupt instead on other types of systems

- when process "1234" is scheduled, it checks for pending signals, sees that SIGKILL is marked, and takes the default action for SIGKILL, which is to exit

Notes: *killing the process is not done immediately upon posting the signal*

4. [12 points; 4 each] Performance

You need to tune the performance of an application that you wrote. You have profiled the application and the OS and find that it spends 50% of its execution time executing application instructions, 36% handling page faults, and 14% handling TLB misses (the system uses a software TLB miss handler).

a) Where should you focus your code optimization efforts first, and why?

Focus on application first, since that is where most of the time is spent. [It's also probably easier to optimize the application than the page fault handling code in the OS.]

b) Suppose all the page fault time is due to I/O, and you do almost no I/O outside of page fault handling. What speedup do you expect if you buy a disk that is twice as fast? Show your calculations. You may express your final answer as a fraction.

New disk will cut I/O time, and thus page fault handling time, in half.

Speedup = $T_{old}/T_{new} = 100 / (50 + 18 + 14) = 100 / 82 = 1 \frac{18}{82} = 1 \frac{9}{41}$

This is somewhat less than 1.25. (numerical answer is 1.22)

c) Instead of spending a lot of time coding, you buy a new CPU with twice the clock speed. You find, however, that your speedup is much less than you were expecting. What is the most likely explanation?

You should only expect your new CPU to speed up the 64% of the time that is non-I/O. If your measured speedup is less than what this would give, the most likely explanation is that your application is limited by something other than the CPU speed, probably memory access time, which didn't change when you installed the new CPU.

5. [10 points] Memory Reclamation

L.T. tries to implement quiescent-state-based memory reclamation. Assume that he has written a function `quiescent_state()` which announces to all other threads that the calling thread has gone through a *quiescent state*, and always detects a *grace period* if one exists. (L.T. sometimes writes brilliant code.)

However, L.T. has given each thread exactly one limbo list, which it empties on every grace period. (L.T. sometimes writes sloppy code.) **Show a simple schedule in which this causes a problem, and explain what the problem is.**

6. [9 points, 3 each]**Multiprocessors: Locking Issues**

In the lectures, we looked at 4 ways to improve the performance of spinlocks on multiprocessors.

a) Why do we focus on spinlocks, rather than a lock with a sleep queue for waiting threads?

- *we need them to implement other types of locks on a multiprocessor anyway*
- *certain parts of the OS (like interrupt handlers) are not allowed to block/sleep so spinlocks are the only option for synchronization [related to previous point - we need them anyway]*
- *cost of spinning when we have multiple CPUs may often be less than cost of putting thread to sleep*
- *we can use them in user-level code without any additional OS support [this is true on uniprocessor too, except with multiprocessor, lock holder can run and release while a thread spins, so this relates to previous point]*

b) All four strategies address the main performance issue (which is not wasted CPU cycles in the spinning thread) in different ways. Explain the problem.

Main issue is memory contention as all waiting threads repeatedly attempt to modify the same memory location in trying to acquire the lock. Each such access creates extra cache and memory bus traffic, and may slow the execution of other memory operations unrelated to the lock variable.

c) Pick any one of the improved spinlocks, and explain how it addresses the performance issue of (b).

- *test-and-test-and-set repeatedly reads the lock variable (which can be cached) and only attempts to acquire (modify the lock variable) when it is seen to be available, reducing memory contention to points where lock changes ownership*
- *exponential backoff reduces the number of attempts to acquire if the lock is seen to be unavailable, spinning on a private location during backoff period instead*
- *ticket locks give each thread own memory location (their ticket) and the shared now_serving variable is only modified by one thread on lock release, so most reads by spinners are served from cache*
- *queueing locks as explained in 1(c)*

7. [12 points] Parallel Job Scheduling

At a particular point in time, all 4 CPUs on a space-shared multiprocessor become idle. The job scheduler queue is shown below. Show the scheduling decisions that result if we are using FCFS with the EASY backfilling algorithm, on the CPUxTime matrix below, up to the point where all jobs have run.

Write a 1-line description of how the scheduling decision was made for each job.

Scheduler Queue (Job ID, # CPUs, Time per CPU):

empty	F, 2, 2	E, 1, 4	D, 3, 2	C, 4, 4	B, 2, 3	A, 1, 8
-------	---------	---------	---------	---------	---------	---------

Queue head

Schedule:

CPU ₀	A	A	A	A	A	A	A	A	C	C	C	C				
CPU ₁	B	B	B	F	F	D	D		C	C	C	C				
CPU ₂	B	B	B	F	F	D	D		C	C	C	C				
CPU ₃	E	E	E	E		D	D		C	C	C	C				

Time

- 1) *A is at head of queue and CPUs are available, so it is started (pick CPU₀ for simplicity)*
- 2) *B is now at head of queue and enough CPUs are available, so it is started.*
- 3) *C is at head of queue, but can't start yet, so a reservation is made for it at the earliest possible time that it could start (at end of A's execution).*
- 4) *Backfill algorithm considers D, can't be started immediately, but EASY only makes one reservation, so we leave D in queue and move on to E.*
- 5) *E can be started now on last unallocated CPU.*
- 6) *B finishes, C is at head of queue still with reservation, and D still can't be started (we only have the 2 CPUs freed by B at this point). Backfill finds F in queue, which only needs 2 CPUs however, so F is started.*
- 7) *E finishes, but still can't run D until F finishes. After F finishes, we now have 3 CPUs free for 3 time units before C's reservation time, so D is started.*
- 8) *D finishes, then A finishes and we have all 4 CPUs available to run C.*

Extra space. Please indicate clearly which question(s) you are answering here, if any.

Total marks = (80)

End of test