

CS 167 Midterm Exam Solutions

Fall 2008

Do all of questions 1 through 3.

1. *Recursion, in the context of programming languages, refers to a function that calls itself. For example, the following is a simple example (assume that it's called only with appropriate argument values):*

```
int factorial(int n) {
    if (n == 1)
        return n;
    else
        return n*factorial(n-1);
}
```

Tail recursion is a restriction of recursion in which the result of a recursive call is simply returned — nothing else may be done with the result. For example, here's a tail-recursive version of the factorial function:

```
int factorial(int n) {
    return f2(n, 1);
}

int f2(int a1, int a2) {
    if (a1 == 1)
        return a2;
    else
        return f2(a1-1, a1*a2);
}
```

- a. *Why is tail recursion a useful concept? (Hint: consider memory use.)*
With tail recursion, one can write recursive functions that use a minimum of stack space.
 - b. *Explain how tail recursion might be implemented so as to obtain the advantages mentioned in part a.*
Each tail call reuses the current stack frame rather than pushing a new one on the stack. Thus placing a call simply involves restoring the stack and registers to the state they were in when the current subroutine was entered (on the x86 this means that the local variables are popped off the stack, saved registers are restored, and the saved ebp is popped from the stack and into the ebp register). [You don't need to supply the x86 details.]
2. *We have a new architecture for interrupt handling. There are n possible sources of interrupts. A bit vector is used to mask them: if bit i is 1, then interrupt source i is masked. The operating system employs n threads to handle interrupts, one per interrupt source. When interrupt i occurs, thread i handles it and interrupt source i is automatically masked. When the thread completes handling of the interrupt, interrupt source i is unmasked. Thus if interrupt source i attempts to send an interrupt while a previous interrupt from i is being handled, the new interrupt is masked until the handling of the previous one is completed. In other words, each interrupt thread handles one interrupt at a time.*

Threads are scheduled using a simple priority-based scheduler. It maintains a list of runnable threads (the exact data structure is not important for this problem). There's a global variable `CurrentThread` that refers to the currently running thread.

- a. *When an interrupt occurs, on which stack should the registers of the interrupted thread be saved? Explain. (Hint: there are two possibilities: the stack of the interrupted thread and the stack of the interrupt-handling thread.)*

They should be saved on the stack of the interrupted thread. This way this thread's stack contains all the thread's context, allowing the thread to be scheduled independently of the interrupt thread.

- b. *After the registers are saved, what further actions are necessary so that the interrupt-handling thread and the interrupted thread can be handled by the scheduler? (Hint: consider the scheduler's data structures.)*

`CurrentThread` must be set to refer to the interrupt-handling thread and the interrupted thread must be put back into the list of runnable threads.

- c. *Recall that Windows employs DPCs (deferred procedure calls) so that interrupt handlers may have work done when there is no other interrupt handling to be done. How could this be done in the new architecture? (Hint: it's easily handled in the new architecture.)*

The purpose of a DPC is to allow an action to be performed without interfering with the handling of further occurrences of the interrupt that generated it. This might be done by having the interrupt threads run at a high priority. DPCs could be handled by one or more threads that run at a lower priority. When an interrupt thread generates a DPC, it might append the request to a DPC request queue that is served by the lower-priority DPC threads.

- d. *If there are multiple threads at the same priority, we'd like their execution to be time-sliced — each runs for a certain period of time, then yields to the next. In Windows, this is done by the clock interrupt handler's requesting a DPC, which forces the current thread to yield the processor. Explain how such time-slicing can be done on the new architecture.*

When there's a clock interrupt, its thread will preempt whatever thread is running at the moment. Assuming that the interrupted thread is put back on the list of runnable threads so that it is next chosen to run *after* all other threads of equal priority, then nothing else needs to be done to implement time slicing.

3. *Linux and other Unix operating systems provide the `madvise` system call with which a thread can tell the operating system how it will be referencing memory. For example, a thread can specify that its references to a particular region of memory will be random, meaning that there's no predicting whether any particular page will be referenced or not at any given moment. It can also specify that its references to a particular region of memory will be sequential, meaning that pages are going to be accessed in sequential order.*

Explain how the operating system might use this information in both of the above cases with both of the above flags. In particular, how might it affect page-ins and page-outs?

If a thread's referencing behavior is random, then there's no point to paging in any page but the one faulted on — pre-paging is useless, since any page that might be fetched is unlikely to be needed soon. If a page must be removed to create free space, no information is available as to which one is needed least, and thus removal of a random page is fine is as good of a choice as any.

If, however, a thread's referencing behavior is sequential, then it does make sense to pre-page — once a page is faulted on, not only should that page be fetched, but as many following it (sequentially) as is reasonable to bring in — they're sure to be referenced soon. If a page must be paged out to make free space available, then the page that's been in the longest (furthest behind the current reference) is the one least likely to be referenced again soon.

If you do all of the following correctly, you'll get an A regardless of how well you do on the first three problems. If you miss any of the following, your grade will be based solely on how well you do on the first three problems.

4. *The origin of the term foobar is disputed. However, one possibility is that it comes from fubar, an acronym used by members of the U.S. military in World War II.*
 - a. *What does fubar stand for? (Hint: there's the correct version and the not-so-correct but more polite version. Either is acceptable as an answer.)*

The polite version is “fouled up beyond recognition” (or perhaps “fouled up beyond repair”).
 - b. *In what was clearly an attempt at humor by the designers of the Digital VAX-11 computer, fubar was the acronym for the name of a diagnostic register. What did it stand for here?*

Failed unibus address register.
5. *The CIT building was completed in 1988 and CS moved into the fourth and fifth floors. Renovations to the third floor were completed in 2005 and CS expanded into it. Who was the prior occupant of the third floor?*

CIS (computer and information services)
6. *Who was the computer-industry pioneer noted for having the motto “think”?*

Thomas J. Watson Sr., the founder of IBM.
7. *Which Brown alumnus succeeded Steve Jobs as CEO of Apple? (He had absolutely no CS background.)*

John Sculley
8. *Which Brown alumnus was the ninth employee of Microsoft? (He definitely had a CS background.)*

Bob Wallace
9. *Which Brown CS faculty member has been on the Brown faculty the longest? In which department did he or she start?*

Andy van Dam. He started in Applied Mathematics.
10. *Early CRT-based (CRT stands for cathode-ray tube) computer terminals displayed strictly text and provided a single window. What were the dimensions of the window in terms of number of lines of text and characters per line?*

24 lines and 80 characters per line.