(VERSION WITH ANSWERS)
CORE EXAMINATION
Department of Computer Science
New York University
February 6, 2009

This is the common examination for the M.S. program in CS. It covers core computer science topics: Programming Languages (PL), Operating Systems (OS), and Algorithms (ALG). The exam has two parts. The first part (PL and OS) lasts two and a half hours. The second part (ALG), given this afternoon, lasts one and one-half hours.

You will be assigned a seat in the examination room.

Use the proper booklet or answer sheet for each question. Each booklet is marked with the Area and Question number, in the form PL1, PL2, OS1, OS2, ALG1, ALG2, and ALG3. DO NOT put your name on the exam booklet or answer sheet. Instead, your exam number must be on every booklet.

You will be graded according to your exam number, shown on the envelope containing the booklets. Remember your exam number: when grades are given out, they will be published according to this number, not by name.

Make sure your name and signature are on the envelope. This is the only place where your name appears. Please include all the booklets inside the envelope. You can keep the exam.

Good luck!

SYSTEMS PART:
Programming Languages and Operating Systems

Question 1—Please use the Exam Booklet labeled PL1

Programming Languages: Recursive and Iterative Programming

For this question, each part will count almost equally.

The fibonacci function is defined as follows:

$$fib(n) = \begin{cases} 1 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ fib(n-1) + fib(n-2) & \text{otherwise} \end{cases}$$

1. Write, in your favorite *imperative* programming language, a recursive fibonacci function.

Answer:

```
int fib(int n)
{
  if (n==0)
    return 1;
  else if (n==1)
    return 1;
  else
    return fib(n-1) + fib(n-2);
}
```

2. In the same language, write an iterative version of the fibonacci function (using a loop, no recursion).

Answer:

```
int fib(int n)
{
  if ((n==0) || (n==1))
    return 1;
  else {
    int prev1 = 1;
    int prev2 = 1;
    int i, current;
    for(i = 2; i <= n; i++) {
      current = prev1 + prev2;
      prev2 = prev1;
      prev1 = current;
    }
    return current;
  }
}
```

3. Which of the two versions of your fibonacci function is more efficient (in terms of asymptotic complexity)?

Answer:

The iterative version, which is linear in n, is more efficient than the recursive version, which is exponential in n.

4. If your iterative solution was more efficient, rewrite your recursive version so that it has the same asymptotic complexity as your iterative solution. Be sure, though that the rewritten version is still recursive (and doesn't use a loop). If you wish, you may write use an additional auxiliary function (which also cannot use a loop).

Answer:

```
int rfib(int n, int i, int prev1, int prev2)
{
  if (i == n)
    return prev1 + prev2;
  else
    return rfib(n, i+1, prev1 + prev2, prev1);
}

int fib(n)
{
  if ((n==0) || (n==1))
    return 1;
  else
    return rfib(n,2,1,1);
}
```

5. Describe, in general, how to take a loop and translate it into a recursive function that has the same asymptotic complexity.

Answer:

A loop is easily translated into a function (called a tail recursive function), in which the recursive call is the last operation that the function performs. A very simple algorithm, given a loop L, defines a function f that has one formal parameter for each variable referenced in the loop (including the loop index variable). The jump to the top of the loop is replaced by the tail recursive call. The body of the function is the body of the loop.

6. Is it always possible to translate a recursive function into a loop (which is the inverse of what you were asked in the previous question)? If so, describe, in general, how to take a recursive function and translate it into a loop. If not, give an example of a recursive function that cannot be translated into a loop.

Answer:

It is always possible to translate recursion into iteration – this is what a compiler does, since hardware doesn't support recursion directly. A recursive function can be implemented using iteration by the explicit maintenance of a stack. A recursive call is translated into a push operation, where each record on the stack contains the value of each local variable (including formal parameters) in the function.

Question 2—Please use the Exam Booklet labeled PL2

Programming Languages: Functional Programming

1. (5 Points) Write a function called `make_sentence` in ML that takes a list of strings and returns a string which is the sentence created by appending the strings in order with spaces in between and a period at the end. Recall that `^` is the string concatenation operator.

   Example:

```
> make_sentence ["This", "is", "a", "sentence"];
val it = "This is a sentence." : string
> make_sentence ["a", "b", "c", "d"];
val it = "a b c d." : string
```

2. (5 Points) Write a program called `maximize` in Scheme that takes two arguments: a list and a function. It should return the element of the list that gives the maximum value when the function is applied to it (of all the elements of the list).

   Examples:

```
> (maximize '((a b) () (c d e) (f)) length)
(c d e)
> (maximize '(2 5 3 1 9) (lambda (x) (- x)))
1
```

Answer:

```
fun make_sentence [] = "."
|   make_sentence [x] = x ^ "."
|   make_sentence (x::l) = x ^ " " ^ make_sentence l;


(define (helper l f max)
  (cond
    ((null? l) max)
    (#t (let* ((first (car l))
               (rest  (cdr l))
               (newmax (if (> (f max) (f first)) max first)))
          (helper rest f newmax)))))

(define (maximize l f)
  (helper (cdr l) f (car l)))
```

Question 3—Please use the Exam Booklet labeled OS1

Operating Systems: I/O and Filesystems

To make the arithmetic easier, assume for this entire question that each disk block is 1,000 bytes in size.

Part A (2 Points)

For parts A and B assume the OS uses the FAT (File Allocation Table) file system.

Consider an existing file with 100 blocks (hence 100,000 bytes). Assume the file has not yet been accessed since the OS has been started. Assume 1,000 new bytes (i.e., the size of a block) are to be added to the **beginning** of the file. These 1,000 bytes are already in memory, but not yet on disk What actions must the OS take, and what I/Os are necessary?

Part B (2 Points)

The same question as part A except that the 1,000 new bytes are to be added to the **end** of the 100 block file file.

Part C (2 Points)

For parts C, D, and E assume the OS uses the Unix i-node based file system. An i-node in this system has pointers to 5 direct blocks, 1 single-indirect, and 1 double indirect block. There are **no** triple indirect blocks.

Answer the same question as part A, but for this different file system. That is, we are to add 1,000 bytes to the **beginning** of a 100 block, not yet accessed file.

Part D (2 Points)

The same question as part C except that the 1,000 new bytes are to be added to the **end** of the file.

Part E (2 Points)

Assume all pointers are 4 bytes in length. Consider the largest possible file in this i-node based file system. What is the total number of **direct blocks** in this file.

$$\overbrace{\hspace{12cm}}$$

Answer:

`To appear.`

$$\underbrace{\hspace{12cm}}$$

Question 4—Please use the Exam Booklet labeled OS2

Operating Systems: User/Kernel separation and Processes

Answer the following multiple-choice questions. Circle *all* answers that apply.

For these questions all 21 statements count equally

A. In a typical modern OS, which of the following statements are true about user applications and the Kernel?

1. The Kernel runs at superviser privilege level.
2. Applications run at superviser privilege level.
3. Applications may directly invoke any function calls in the Kernel.
4. The Kernel executes an application's machine instructions the way a JVM executes Java bytecode.

B. Which of the following instructions should be protected, i.e., can execute only when the processor is running at superviser level?

1. MOV cr3, src; (move the value in *src* to control register CR3. CR3 contains the physical address of the base of the page directory).
2. TRAP; (jump to kernel's syscall dispatcher with kernel privilege)
3. ADD; (add numbers)
4. JMP; (jump to a different instruction)
5. INB (contact I/O device).

C. Which of the following are elements of a typical process descriptor? (Recall that a process descriptor or process control block is the per-process state kept by the Kernel.)

1. Process state (blocked/runnable);
2. Address space;
3. Disk driver;
4. File descriptor table.

D. Which of the following statements are true about processes?

1. Any process can access any other process's memory by default.
2. Timer interrupts are generally required for a kernel to correctly isolate processes.
3. The kernel must run one process to completion before it starts another process.
4. A user-level process can crash the kernel if not programmed carefully.

E. Which of the following statements are true when the kernel switches execution from the current process to another process?

1. The kernel *must* save the register values (including IP (Instruction Pointer), SP (stack pointer) etc.) of the current process to memory.
2. The kernel *must* save the content of the current process's memory on disk.
3. The kernel *must* close all the files opened by the current process.
4. The kernel *must* make the page table of the next process active.

Answer:

```
A: 1
B: 1,5
C: 1,2,4
D: 2
E: 1,4
```