Final Examination

COS 217, Spring 2004

Name:

UNIX login:

There are five problems on this examination, each worth from 15 to 35 points. You have 120 minutes to complete this examination. Each problem should take you about its point value in minutes to complete.

You are not required to comment your code in this exam unless what you have written is difficult to understand.

Please start your solution to each problem in the reserved space of your examination booklet. This examination is open book and open notes, but no calculators or other electronic devices are permitted.

Partial credit will be given if you can show your work in arriving at solutions. Be concise.

Good luck!

Write out and sign the Honor Code pledge before turning in the test.

“I pledge my honor that I have not violated the Honor Code during this examination."

1. Computer Organization (15 points)

(a) (2 points) Describe the parts of a typical instruction and the steps in the execution of an instruction.

(b) (2 points) Describe the steps in the execution of a program.

(c) (4 points) Describe the differences between an exception and an interrupt and give two examples of each.

(d) (7 points) Briefly describe the main differences between a procedure call and a system call, and then describe the calling sequences of both.

2. Assembly Programming and Assembler (20 points)

Write a procedure sum, in IA32 assembly, that takes two arguments `x` and `y` (both of them are positive) and return the sum of all the numbers between 1 and `y` that are divisible by `x`. You should use the standard calling sequence such that a C program can call. For example, the following C code

```
int foo;
foo = sum( 3, 100 );
```

will sum up all the numbers between 1 and 100 that are divisible by 3 and put the result in variable `foo`.

3. C Programming (35 points)

(a) (15 points) Write a simple filter **`compress`**, that compresses its input into a “run-length” compressed form. The program uses an “escape” character to encode the run length. If a non-escape character in the input occurs consecutively more than twice, the program will compress the occurrences of the character into a three-byte escape sequence **`<escape-char><char><count>`**, where **`<escape-char>`** is $\mathbf{377}_8$ (a byte with all bits on), **`<char>`** is the character in the input, and <count> is the number of consecutive occurrences of the character. Your program should work correctly when an input character is **`<escape-char>`** itself (hint: use the three-byte escape sequence to encode escape character sequences).

(b) (10 points) Write a filter called **`decompress`** that decompresses a compressed file into the original input file.

(c) (10 points) Provide a test plan that includes a stress test plan and all boundary conditions you plan to test.

4. UNIX System Services (15 points)

UNIX shell allows users to use a pipe to connect two programs together. For example, it is quite common for a UNIX user to use the following shell command

```
ls –l | more
```

to list the details of files in a large directory in a page-by-page fashion.

(a) (10 points) Write a C program **`lsmore`** that implements the command above.

(b) (5 points) Show how to modify **`lsmore`** without rewriting **`more`**, such that if the **`more`** program is still running after 360 seconds, the program will terminate.

5. Software Engineering (15 points)

(a) (9 points) Identify any portions of the following C program that are not portable. Show how to make them portable on multiple hardware and software platforms.

```
#include <stdio.h>

#define N 10000

int MakeData( int x[], float y[], int size )
{
   char c;
   int i;

   for ( i = 0; i < N && ((c = getchar()) != EOF); i++)
      x[i] = c;

   size = i;
   i = 0;
   while ( i < size )
      y[i] = (float) x[i++] + i;
   return size;
}

main(void)
{
   int size;
   int *iBuf;
   float *fBuf;

   iBuf = (int *) malloc(4*N);
   fBuf = (float *) malloc(4*N);

   size = MakeData( iBuf, fBuf, N );
   fwrite( fBuf, 4, size, stdout );
}
```

(b) (6 points) After making (a) portable, please identify the portions of the resulting code that are not robust and show how to make them robust. You can label the portions you have identified and rewrite them here.