Name:

COS 217 Final
Spring 2006
May 22, 2006

Pledge:

Directions:

- Please answer each question in the space provided. The amount of space should be sufficient for a correct answer. If you need more space, please use the backs of pages, and make a note to that effect. If you run out of space, exam books are provided at the front of the room.
- This exam is open-book, open-notes, and is covered by the Honor Code. Please write and sign the pledge after you finish your exam.
- There are a total of five sections, with the number of points for each shown by the question. While it is not the intent for the exam to be a race, spending too much time on a single question may preclude finishing the exam. Budget your time wisely.
- To be fair, I will try to avoid answering content-related questions during the exam, unless it's to correct a mistake on my part.
- If you feel that a question **requires** additional assumptions or information to answer, please state them. Your guiding principle should be *Occam's razor*, which loosely translated states that you should allow as few assumptions as necessary to explain the situation.
- Answers should assume C/Unix unless otherwise stated or implied
- Please first read over the entire exam and then begin to answer questions. I will wait outside the exam room for the first five minutes, and then will be available in my office (room 322).
- Please write legibly

#	Name	Points Available	Score
1	True or False	10	
2	Short answer	15	
3	Code reading	25	
4	Bug hunt	25	
5	Putting It Together	25	
	Total	100	

1. True or False (10pts) For each statement, write "true" if you believe the statement is correct, or false if you believe the statement is incorrect. If you believe the statement does not have a clear answer, give whichever choice is more appropriate and explain why.

- `movl %eax, %ebx` is generally faster than `movl (%eax), %ebx`
- for simplicity, all stack frames are the same size
- programmers try to write race conditions to make programs faster
- if you have only one local variable to allocate, you can use `pushl $0` instead of `subl $4, %esp`
- variables declared const are placed in roData

2. Short Answer (15pts) Answer each item *well* in no more than 3 sentences.

   - What is a relocation record?

   - Why are relocation records needed?

   - In what environments is it more useful to measure a program's virtual time than wall-clock time?

- In what environments is it more useful to measure a program's wall-clock time than virtual time?

- Why might a malloc library use different memory regions for bookkeeping structures and allocated buffers?

3. Code reading (25pts)

The following assembly code was automatically generated by compiling a very short, nonsensically-named, C function that takes a single parameter of type "unsigned char *" and has a return type void.

```
        .file   "wipe.c"
        .text
.globl wipe
        .type   wipe, @function
wipe:
        pushl   %ebp
        movl    %esp, %ebp
        pushl   %ebx
        subl    $4, %esp
        movl    8(%ebp), %ebx
        cmpb    $0, (%ebx)
        je      .L7
.L5:
        movzbl  (%ebx), %edx
        movb    %dl, %al
        subb    $65, %al
        cmpb    $25, %al
        ja      .L4
        movzbl  %dl, %eax
        movl    %eax, (%esp)
        call    putchar
.L4:
        incl    %ebx
        cmpb    $0, (%ebx)
        jne     .L5
.L7:
        addl    $4, %esp
        popl    %ebx
        popl    %ebp
        ret
        .size   wipe, .-wipe
        .section        .note.GNU-stack,"",@progbits
        .ident  "GCC: (GNU) 3.4.4"
```

Here are some hints to help understand this code:

1. "dl" and "al" are just the low-order byte names of the registers
2. "movzbl" transfers a byte into a long register, and sets the high-order bytes to zero
3. "ja" is the unsigned equivalent of "jg", and means "jump if above"
4. 65 is ASCII for uppercase A

- In what register is the code storing the value of the formal parameter?

- The two instructions before L5 are similar to the two before L7. In plain English, what are they doing? (i.e., give a brief, high-level description of their function).

- Describe the conditions necessary for the "ja" in L5 to fail. Remember that "ja" is an unsigned comparison

- In a simple, plain English sentence, give a high-level summary of what this code does

4. Bug hunt! (25pts)

The following program is supposed to wait around for the user to hit ctrl-c and count how many times the user tried to end this program. While intended to run on basically any system, it's full of bugs.

```
#include <signal.h>
#include <stdio.h>
#include <unistd.h>

static int numSigs = 0;

void Handler(int val)
{
  int temp = numSigs;

  temp++;
  printf("you tried to break out %d times\n", temp);
  numSigs = temp;
}

int main(int argc, char *argv[])
{
  signal(SIGINT, Handler);

  while (1);
    sleep(1);
}
```

- If you ran a correct version of this program at the console, how would you end it? (5 points)

- Identify and briefly explain as many bugs, possible bugs, and design flaws as you can. Suggest fixes where appropriate. Five right gets full credit (20 points). Using "int" for numSigs and temp is not considered a bug.

this page left intentionally blank to answer the previous question

4. Putting It Together (25pts)

Given what you know about the system as a whole, give well-rounded answers to the following questions:

- (8 points) Assume you have a particular task that needs to be performed, and it may be generally useful. What are the pros/cons of putting it (a) in your code directly, (b) in a library, and (c) in the operating system

- (8 points) Programmers often define a macro MAX as something like

  ```
  #define MAX(a,b) (a > b) ? a : b
  ```

  which roughly translates to “if a is greater than b, return a, else return b”. Give an example of an invocation of where this macro can behave differently than its corresponding function. If you decided to be safe and replace this macro with a function in an existing program, what other problems would you have to consider?

- (8 points) Assume you want to allocate some information associated with people's names and you are given a program that uses a data structure to store both the variably-sized name and one piece of information as shown:

  ```
  typedef struct Entry {
    struct Entry *e_next;
    int e_value;
    char e_name[1];
  } Entry;
  ```

  Describe how this structure works, comment on whether it is valid C, and discuss memory consumption of this approach versus a separate strdup call for the person's name, assuming a modern memory allocator.

- (1 point) Do you want to play CAVE EXPLORER?