**Full Name:** _____

**Roll Number:** _____

# IT215: Systems Software, Winter 2015-16
# First In-Sem Exam (1 hour 30 minutes)

Feb 8, 2016

**Instructions:**

- Make sure your exam is not missing any sheets, then write your name and roll number on the top of this page.
- Clearly write your answer in the space indicated. None of the questions need long answers.
- For rough work, do not use any additional sheets. Rough work will not be graded.
- Assume IA32 machine running Linux.
- The exam has a maximum score of 20 points. It is CLOSED BOOK. Notes are NOT allowed.
- Anyone who copies or allows someone to copy will receive F grade.

| |
|---|
| Problem 1 (/5): |
| Problem 2 (/2): |
| Problem 3 (/2): |
| Problem 4 (/1): |
| Problem 5 (/4): |
| Problem 6 (/6): |
| TOTAL (/20): |

## Problem 1. (5, 1 each points):

Circle the *single best* answer to each of the following questions. You will get -0.5 points for a wrong answer, so don't just guess wildly. If you circle more than one answer, you will lose the mark for the corresponding question.

1. A process exists in the zombie (also known as defunct) state because:

   (a) It is running but making no progress.

   (b) The user may need to restart it without reloading the program.

   (c) The parent may need to read its exit status.

   (d) The process may still have children that have not exited.

2. Consider a function with the following declaration:

   ```
   void foo(int a, int b, int c, int d);
   ```

   Assuming `foo` has been compiled for a x86 machine with 4-byte ints, what would be the addresss of the argument `b` in terms of `%ebp` in the stack frame for `foo`?

   (a) `%ebp` + 8

   (b) `%ebp` + 12

   (c) `%ebp` + 16

   (d) `%ebp` + 20

4. What is printed by this program?

```
int A[12] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11};
int main()
{
    char *p = (char *) (&A[0]);
    int *q = (int *) (p + 8);
    printf("%d\n", *q);
    return 0;
}
```

   (a) 0

   (b) 2

   (c) 8

   (d) It doesn't print anything; it gets a segmentation error and dies.

5. Assume a function `foo` takes two arguments. When calling `foo(arg1, arg2)`, which is the correct order of operations assuming x86 calling conventions?

   (a) `push arg1, push arg2, call foo, push %ebp`

   (b) `push arg1, push arg2, push %ebp, call foo`

   (c) `push arg2, push arg1, call foo, push %ebp`

   (d) `push arg2, push arg1, push %ebp, call foo`

## Problem 2. (2, 1 each points):

A. Consider the following snippet of IA32 code:

```
8048390:      call 8048395
8048395:      pop %eax
```

Suppose that just before the `call` instruction executes, `%esp = 0xffffd834`. Then what is the value of `%eax` after the `pop` instruction executes?

`%eax = 0x`_____

B. Consider a slightly different snippet of IA32 code:

```
8048396:      call 804839b
804839b:      ret
```

Suppose that just before the `call` instruction executes, `%esp = 0xffffd838`. Then what is the value of `%eip` after the `ret` instruction executes?

`%eip = 0x`_____

## Problem 3. (2, 1 each points):

Consider the two programs shown below.

```
// Program 1
int main()
{
    printf("PID %d running prog 1\n", getpid());
}

// Program 2
int main()
{
    char *argv[2];
    argv[0] = "prog1";
    argv[1] = NULL;
    printf("PID %d running prog 2\n", getpid());
    execv("./prog1", argv);
    printf("PID %d exiting from prog 2\n", getpid());
}
```

Assume we have Program 1 compiled to `./prog1` and there are no errors (e.g. `execv` doesn't fail, etc).

A. How many different PIDs will print out if you run Program 2? _____ PIDs

B. How many lines of output are printed? _____ lines

## Problem 4. (1 points):

In the buffer overflow lab, a buffer was allocated on the stack. When the user ran the program and typed something in, it was written into the buffer. If the user entered more characters than the buffer could fit, they could overwrite additional values on the stack. Which of the following regions of a stack could they directly overwrite in this manner? (Circle one)

A. The part of the stack with higher (i.e. larger) addresses than the buffer.

B. The part of the stack with lower (i.e. smaller) addresses than the buffer.

## Problem 5. (4=1+3 points):

Consider the program shown below.

```
int main() {
    pid_t a = 0, b = 0;

    fork();
    a = fork();
    if (a == 0)
       b = fork();
    if (a || b)
       fork();
    return 0;
}
```

Assume `fork` succeeds at every call.

A. How many *new* processes will be created when the program is executed? _____ processes

B. Draw a process graph that illustrates the processes at run-time. Hint: It usually helps to write down values for `a` and `b` for each process.

## Problem 6. (6, 2 each points):

Consider the following program. Assume that all system calls and functions return normally.

```
static pid_t pid;
static int counter = 0;

static void handlerOne(int sig) {
     counter += 7000;
     printf("counter = %d\n", counter);
}

static void handlerTwo(int sig) {
     counter += 20000;
     handlerOne(sig);
     kill(pid, SIGUSR1);
}

int main() {
     signal(SIGUSR1, handlerOne);
     if ((pid = fork()) == 0) {
         signal(SIGUSR1, handlerTwo);
         if ((pid = fork()) == 0) {
              counter += 500;
              printf("counter = %d\n", counter);
              signal(SIGUSR1, handlerOne);
              kill(getppid(), SIGUSR1);
              exit(0);
         }
         counter += 30;
         printf("counter = %d\n", counter);
    }
    counter++;
    waitpid(pid, NULL, 0);
    printf("counter = %d\n", counter);
}
```

The above program is capable of printing out something very close to the following:

```
counter = 500
counter = 30
counter = 27031
counter = 7500
counter = 27031
counter = 2
```

Answer the questions on the next page.

A. Which single line of the output is incorrect, and what should it be?

B. Which two lines might be exchanged by another test run? Why can that happen?

C. Which line might be missing altogether from another test run? Why can that happen?