Full Name: _____

Roll Number: _____

IT215: Systems Software, Winter 2013-14

Second In-Sem Exam (2 hours)

March 13, 2014

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 50 points. It is CLOSED BOOK. Notes are NOT allowed.
- Anyone who copies or allows someone to copy will receive F grade.

  Good luck!

1 (15):
2 (5):
3 (6):
4 (6):
5 (6):
6 (6):
7 (6):
TOTAL (50):

Problem 1. (15, 3 each points):

Circle the *single best* answer to each of the following questions. You will get -2 points for a wrong answer, so don't just guess wildly.

1. For the Unix linker, which of the following accurately describes the difference between global symbols and local symbols?

   (a) There is no functional difference as to how they can be used or how they are declared.

   (b) Global symbols can be referenced by other modules (files), but local symbols can only be referenced by the module that defines them.

   (c) Global symbols refer to variables that are stored in `.data` or `.bss`, while local symbols refer to variables that are stored on the stack.

   (d) Both global and local symbols can be accessed from external modules, but local symbols are declared with the "static" keyword.

2. Due to the simplistic mechanism used within the kernel to track pending signals for a given process, it is possible that

   (a) a process terminates without becoming a zombie process

   (b) a parent process does not receive a `SIGCHLD` signal for every terminated child process

   (c) a terminated child process cannot be reaped by the parent with the `wait` system call

   (d) a parent process is notified of a child's termination before it has actually occurred

3. Which of the following is NOT true of ELF files?

   (a) The `.symtab` section contains entries for local nonstatic variables.

   (b) The `.bss` section occupies no actual space in the object file.

   (c) The `.rel.data` section holds relocation information for initialized global pointer variables.

   (d) The `.data` section contains initialized global variables.

4. Assuming all the system calls return normally, which of the following pieces of code will print the string "Hello" to `stdout`? Recall that the `dup2(int srcd, int dstfd)` function copies descriptor table entry `srcfd` to descriptor table entry `dstfd`.

   (a)
   ```
   int fd = open("hoola.txt", O_RDWR);
   dup2(fd, STDOUT_FILENO);
   printf("Hello");
   ```

   (b)
   ```
   int fd = open("hoola.txt", O_RDWR);
   dup2(fd, STDOUT_FILENO);
   write(STDOUT_FILENO, "Hello", 5);
   ```

   (c)
   ```
   int fd = open("hoola.txt", O_RDWR);
   dup2(STDOUT_FILENO, fd);
   write(fd, "Hello", 5);
   ```

   (d)
   ```
   int fd = open("hoola.txt", O_RDWR);
   dup2(fd, STDOUT_FILENO);
   write(fd, "Hello", 5);
   ```

5. In the malloc project, we provided code for an implicit list allocator. Many students improved this code by creating an explicit linked list of free blocks. Which of the following reason(s) explain(s) why an explicit linked list implementation has better performance?

   I. Immediate coalescing when freeing a block is significantly faster for an explicit list.

   II. The implicit list had to include every block in the heap, whereas the explicit list could just include the free blocks, making it faster to find a suitable free block.

   III. Inserting a free block into an explicit linked list is significantly faster since the free block can just be inserted at the front of the list, which takes constant time.

   (a) I only.

   (b) II only.

   (c) III only.

   (d) II and III only.

   (e) All I, II, and III.

Problem 2. (5, 1 each points):

Write T (for TRUE) or F (for FALSE) in the blanks provided. You do not have to give a reason. NOTE: Correct answers receive +1 mark, incorrect answers -1 mark, unanswered parts receive 0.

A. ____ Under normal conditions, `exec()` system call will never return.

B. ____ The `exec()` system call creates a new process running a program that is different from the original process.

C. ____ A linker does not need access to the source code to determine what global variables are referenced by an object file.

D. ____ A zombie process is a stopped process that is waiting for a specific signal to resume execution.

E. ____ A user process cannot modify the default action for the receipt of a `SIGKILL` signal.

Problem 3. (6 points):

When creating programs to automatically test student assignments, we want to detect if a student's program gets into an infinite loop. Write a C program that takes the name of a second program to run as a command line argument. Your program will create a process to execute the second program (which itself takes no arguments). If the second program does not terminate after 10 seconds, the original process will terminate it. Your program will print an error message if it must terminate the second process.

```
int main(int argc, char *argv[])
{









}
```

Problem 4. (6 points):

Consider the following C program. Assume there are no errors (e.g. `fork` doesn't fail), but make no assumptions about how the processes are scheduled. Assume each call to `printf` flushes its output out just before it returns.

```
int main() {
   if (fork() != 0) {
       wait(NULL);
       printf("0");
   } else if (fork() == 0) {
       printf("1");
   } else {
       fork();
       printf("2");
       exit(0);
   }
   printf("3");

   return 0;
}
```

For each of the following strings, circle whether (Y) or not (N) this string is a possible output of the program. You will be graded on each sub-problem as follows:

- If you circle no answer, you get 0 points.
- If you circle the right answer, you get 1 points.
- If you circle the wrong answer, you get -1 points.

A. `213032` Y N

B. `123023` Y N

C. `130322` Y N

D. `132203` Y N

E. `220133` Y N

F. `022133` Y N

Problem 5. (6 points):

There are 4 questions to answer for this problem. Assume there are no errors (e.g. `Open` doesn't fail, etc).

Part 1

Consider the following C program, where `foo.txt`, `bar.txt` and `baz.txt` are existing disk files.

```
int main()
{
    int fd1, fd2, fd3;

    fd1 = Open("foo.txt", O_RDONLY, 0);
    fd2 = Open("bar.txt", O_RDONLY, 0);
    Close(fd1);
    fd3 = Open("baz.txt", O_RDONLY, 0);
    printf("fd3 = %d", fd3);
    exit(0);
}
```

The output is: `fd3 =` _____

Part 2

Consider the following C program, where the disk file `barfoo.txt` consists of the 6 ASCII characters "`barfoo`".

```
int main()
{
    int fd1, fd2;
    char c;

    fd1 = Open("barfoo.txt", O_RDONLY, 0);
    fd2 = Open("barfoo.txt", O_WRONLY, 0);
    Read(fd1, &c, 1);
    c = 'z';
    Write(fd2, &c, 1);
    Read(fd1, &c, 1);
    printf("c = %c", c);
    exit(0);
}
```

The output is: `c =` _____

Part 3

Consider the following C program, where the disk file `barfoo.txt` consists of the 6 ASCII characters "`barfoo`".

```
int main()
{
    int fd;
    char c;

    fd = Open("barfoo.txt", O_RDONLY, 0);
    Read(fd, &c, 1);
    if (Fork() == 0)
    {   Read(fd, &c, 1);
        exit(0);
    }
    Wait(NULL);  /* wait for child to terminate */
    Read(fd, &c, 1);
    printf("c = %c", c);
    exit(0);
}
```

The output is: `c = ______`

Part 4

Consider the following C program, where the disk file `barfoo.txt` consists of the 6 ASCII characters "`barfoo`".

```
int main()
{
    int fd1, fd2;
    char c;

    fd1 = Open("barfoo.txt", O_RDONLY, 0);
    fd2 = Open("barfoo.txt", O_RDONLY, 0);
    Read(fd1, &c, 1);
    Dup2(fd2, fd1);
    Read(fd1, &c, 1);
    printf("c = %c", c);
    exit(0);
}
```

The output is: `c = ______`

Problem 6. (6 points):

Consider the following two .c files which both include the same .h file.

```
/* a.h */

_________ int inc(int x) { return x + 1; } /* (1) */

_________ int x; /* (2) */

_________ int z; /* (3) */

_________ void b(void); /* (4) */
```

```
/* a.c */
#include "a.h"
#include <stdio.h>

_________ int w;     /* (5) */

_________ int v = 5; /* (6) */

int main()
{
   x = inc(0);
   z += 4;
   b();
   printf("x = %d z = %d "
          "w = %d v = %d\n",
          x, z, w, v);
   return 0;
}
```

```
/* b.c */
#include "a.h"

_________  int w = 4; /* (7) */

_________  int v = 5; /* (8) */

void b() {
   x = inc(x);
   z--;
   v++;
}
```

When compiled, linked, and executed, the following output results:

```
$ gcc a.c b.c -o a
$ ./a
x = 1 z = 3 w = 4 v = 5
```

Assuming that this program compiled and linked successfully, and based on the output show above, add `static` and/or `extern` modifiers to the blank lines (1) through (8)! If neither modifier would be appropriate, then write "no modifier" next to the comment. Any correct solution that results in successful compilation and the output shown will be accepted.

Problem 7. (6 points):

Consider the following C program. Assume all system calls return predictable values, but make no assumptions about how the processes are scheduled. Assume no signals are sent from other processes, `printf` is atomic, and that each call to `printf` flushes its output just before it returns. Recall that the `pause` function suspends the calling process until a signal is caught.

```
int a = 1;

void handler(int sig) {
     a = 0;
}

void emptyhandler(int sig) { }

int main() {
   Signal(SIGINT, handler);
   Signal(SIGCONT, emptyhandler);

   int pid = Fork();
   if (pid == 0)
   {
         while (a == 1)
            pause();
         printf("a");
   }
   else
   {
       kill(pid, SIGCONT);
       printf("b");
       kill(pid, SIGINT);
       printf("c");
   }
}
```

For each of the following outcomes, circle whether (Y) or not (N) this outcome is a possible outcome of the program. You will be graded on each sub-problem as follows:

- If you circle no answer, you get 0 points.
- If you circle the right answer, you get 1 points.
- If you circle the wrong answer, you get -1 points.

i. Nothing is printed. Y N

ii. The string `ba` is printed. Y N

iii. The string `abc` is printed. Y N

iv. The string `bac` is printed. Y N

v. The string `bca` is printed. Y N

vi. A process does not terminate. Y N

Blank page for rough work.