CS360 Final Exam - May 3, 2012 - James S. Plank

Answer all questions on the answer sheets. When you write code, you do *not* need to include any include files. I would prefer, if you have time, that you rewrite your code to make it neat. However, if you don't have time, I understand. My intended time allocation for the exam is as follows: 5 minutes to get settled, 15 minutes for Question 1, 20 minutes each for Questions 2 and 3, and a full hour for question 4. That said, the grading will be something like 10 points for Q1, 12 points for Q2, 12 points for Q3 and 20 points for Q4. So, were I you, I'd try to finish the first three questions in the first hour of the exam, and then work on Q4. If it takes more than an hour for the first three, make sure that you give me a good outline of how you're writing your code for Q4. That will help me navigate the jibberish that you write in a panicked frenzy when you're writing code.

Question 1

Explain the exact semantics of the following pthreads routines:

- **pthread_mutex_lock(pthread_mutex_t *l);**
- **pthread_mutex_unlock(pthread_mutex_t *l);**
- **pthread_cond_wait(pthread_cond_t *c, pthread_mutex_t *l);**
- **pthread_cond_signal(pthread_cond_t *c);**

Question 2

Part A: Write a program that generates the **SIGPIPE** signal, catches it by printing the string "Got it", and then exits. This program is not allowed to call **fork()**, **system()** or **popen()**.

Part B: Give me an example Unix command line that executes two standard Unix programs in such a way that one of them will exit because it generates **SIGPIPE**. Tell me why **SIGPIPE** is generated in your example, and make sure that your example generates **SIGPIPE** even if the operating system / standard IO library perform some buffering.

Part C: Consider the program to the right. Assume that standard output is going to the terminal. Tell me *every* potential output that this program can have, and how that output comes about.

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>

main()
{
  int i;

  i = write(10, "J\n", 2);
  printf("%d\n", i);
  exit(0);
}
```

Question 3

You are part of a group in CS400 (Senior Design) that is writing a very large program for its final project. The group is composed of you, Alvin, Simon and Theodore. Alvin, Simon and Theodore all want to work together on one program. You don't trust them, so you'll write a separate program that talks with theirs via pipes. With a week before the deadline, they come to you frustrated, because they can't figure out what their program is doing. You are not surprised, but you offer to help.

You distill their code, and in the end, it boils down to figuring out the output to the program that's on the next page. Assume that the program is compiled to **a.out**. Tell me the output of the following five commands:

- **Command A:** `a.out 2 2 2`
- **Command B:** `a.out 2 1 2`
- **Command C:** `a.out 1 0 0`
- **Command D:** `a.out 0 2 0`
- **Command E:** `a.out 2 0 0`

```
#include <stdio.h>
#include <setjmp.h>

jmp_buf SharedBuf;
int i, j, a;

int alvins_code()
{
  printf("A: %d\n", a);
  if (a == 0) longjmp(SharedBuf, 1);
  return a;
}

int simons_code()
{
  int s;

  if (j == 2) s = setjmp(SharedBuf); else s = j;
  printf("S: %d\n", s);
  if (s == 0) return alvins_code();
  if (s == 1) longjmp(SharedBuf, 2);
  return s;
}
```

```
int theodores_code()
{
  int t;

  if (i == 2) t = setjmp(SharedBuf); else t = i;
  printf("T: %d\n", t);
  if (t == 0) return simons_code();
  if (t == 1) longjmp(SharedBuf, 1);
  return t;
}

main(int argc, char **argv)
{
  int m;
  if (argc == 4) {
    sscanf(argv[1], "%d", &i);
    sscanf(argv[2], "%d", &j);
    sscanf(argv[3], "%d", &a);
    m = theodores_code();
    printf("M: %d\n", m);
  }
}
```

Question 4

Five days later, Alvin, Simon and Theodore are finished with their program. It is called **chpmnk**, and takes no command line arguments. **chpmnk** reads lines of input on standard input, and produces lines of output on standard output. Specifically, after reading each line from standard input, it produces one line on standard output. They have assured you that they call **fflush()** after every line (This assurance lets you make your program much easier by calling **fdopen()** and dealing with standard IO streams instead of raw file descriptors).

The first thing that your program is going to do is launch **chpmnk** and set up file descriptors so that your prorgram can talk to **chpmnk**. Your program then reads lines of input from standard input, and will produce lines of output on standard output. Like **chpmnk**, it will produce one line of output for every line of standard input. When it reads a line, it calls **preprocess_line()** on it (that's a procedure that you've already written while you were waiting for your cohorts to finish their part). It then sends the line to **chpmnk** and reads the output of **chpmnk**. Finally, it calls **postprocess_line()** on the output line from **chpmnk** and prints the result on standard output.

You are guaranteed that all lines of input, output, preprocessing and postprocessing are less than 500 characters long.

Your job is to write this program. You are to write *everything*, under the assumption that you have already written **preprocess_line()** and **postprocess_line()**. Just a little more information about them:

```
void preprocess_line(char *line);
void postprocess_line(char *line);
```

Both of these assume that **line** is a C string with less than 500 characters. They process **line** and modify it in place. They may make it bigger, but they will not make it bigger than 500 characters.

Finally: instead of just blasting code at me, first write a decently detailed outline or paragraph of how you're structuring your code, and what you want it to do. Then write code.