CS 252 00 : Spring 2012 : Quiz 2

Name:

Date:

1. What does the following shell script do? Answer succinctly.

```
for i in *
do
   for j in $i/*
   do
      echo $j
   done
done
```

Ans. One to a line, it prints all files exactly one level deep in the file hierarchy. Full credit: It also prints foo/* for any non-directory or empty directory. Bonus: for files in the current directory with a space in the name, it prints word/* for each word.

2. If the function below is executed by two threads, will the final value of the counter be 10000? If not, fix the code so it does.

```
int counter = 0;
void increment() {
   pthread_mutex_t lock;
   pthread_mutex_init(&lock, NULL);
   for (int i = 0; i < 5000; i++) {
      pthread_mutex_lock(&lock);
      counter++;
      pthread_mutex_release(&unlock);
   }
}
```

Ans. No: the loop increment was wrong, the lock was not shared (move outside the fn). Bonus: the lock release was incorrectly named, and the release function was incorrect.

```
int counter = 0;
pthread_mutex_t lock;
pthread_mutex_init(&lock, NULL);
void increment() {
   for (int i = 0; i < 5000; i++) {
      pthread_mutex_lock(&lock);
      counter++;
      pthread_mutex_unlock(&lock);
   }
}
```

3. How many times will "Hello World!" be printed in the following code?

```
01: void* printHello(void* arg) {
02:   fork();
03:   printf("Hello World!\n");
04:}

05: int main() {
06:   printHello(NULL);
07:   pthread_t thr;
08:   pthread_create(&thr, NULL, printHello, NULL);
09:   printHello();
10:}
```

Ans: We start with process 1, which creates process 2 on 06,02. Then, these two processes print [1,2] at 06,03. On line 08 we, process 1 creates thread 1.2, and process 2 creates thread 2.2, both of which run the printHello function. On 08,02, 1.2 creates process 3(.2), and 2.2 creates process 4(.2) the two new threads, and the two new processes each now print at 08,03 [3,4,5,6], but neither the threads nor their forks continue. Only 2 processes (process 1 and 2), then execute line 9, which on 09,02 create processes 5 and 6, respectively. All 4 processes (1,2,5,6) print at 09,03 [7,8,9,10]. A total of 10

printed Hello World's.

4. Spin locks are possible thanks to what machine provided instruction?

   Ans. test_and_set

5. Describe the difference between mutual exclusion and atomicity.

   Ans. Atomicity means only one thread at a time may execute instructions from a single block of code. Mutual exclusion means that only one thread at a time may execute instructions from any of a set of blocks of code. Problem 2 was a sort-of example of atomicity without mutual exclusion [you could think of each instantiation of the code as a different copy because it had a different mutex].

6. Name 2 advantages of threads, and 2 disadvantages, as compared to processes.

   Ans. Many possible answers – see slides. However, threads are (+) faster to create, (+) faster to context switch between, (+) allow faster sharing of data, but (-) have bigger synchronization problems, and (-) have fate sharing [robustness issue].

7. In the command, based on your shell experience:
   `$ ls -l | grep txt | wc -l > x &`

   How many processes will be executed? How many times will the "pipe" system call be invoked? Where does the first process's input come from? Where does the last process's output go? Will the shell wait for any of the processes to exit before printing a new prompt?

   3 processes, 2 calls to pipe(), input from standard input (not that any will be used), output to a file named 'x', and the shell will print the prompt without waiting for any processes to

exit.

8. Does lex or yacc handle tokenization?

Lex.