Homework 1 for CS153 (Winter 2022)

Due: Wed. 1/26

***** Be brief. You will be graded for correctness, not on the length of your answers.

* Typed electronic submissions are expected. If you submit handwritten answers, you are responsible for legibility.

I. Consider a single CPU system with one or more active processes. Explain what happens in the following circumstances including any interrupts, system calls, etc.. and how they are handled until a process is back to running again (3 points)

Example:

A process executes continuously until it exhausts its scheduler allocated time slice

Example answer: The timer interrupt is used by the OS to indicate when a process' time slice it was scheduled for expires. Once the interrupt occurs, we trap to the OS and the timer interrupt handler is invoked—the sleeping beauty awakens! This gives the OS the opportunity to schedule another process. The OS places the current process on the ready queue, and picks another process to run (restoring its context, and switching control back to it).

a) A process needs to write to an open file (writes are not blocking – we don't need to wait for them to finish)

b) A process executes the system call fork()

II. Which of the following requires the operating system to run? For those where the OS is need, identify the type of event that causes the OS to start executing (system call, interrupt, fault, etc…). (4 points)

a) A program is traversing a linked list

b) A process finishes execution and needs to exit

c) The linked-list in (a) has a bug and one of the pointers is null. The process derefrences (accesses) this null pointer.

d) A network packet is received

III. Consider the following program:

```
int count = 1; //shared variable since its global

void twiddledee() {
     int i=0; //for part b this will be global and shared
     for (i=0; i<2; i++) {
     count = count*3; //assume count read from memory once
     }
}
void twiddledum() {

     int i=0; // for part b, this will be global and shared
     for(i=0; i<2; i++) { count = count - 1;}
}
void main() {
    thread_fork(twiddledee); //new thread starts twiddledee
    twiddledum(); //main thread calls twiddledum
    thread_join(); //waits until all the threads get here
    print count;
}
```

a) What are all the values that could be printed in main? (5 points)

b) Repeat part 1 considering that i is also a shared variable (Bonus 3 points – tricky!)

c) Describe a potential schedule of execution (i.e., how the threads interleave running) under the original assumptions where i is not shared that will result in the value printed out being equal to -3. (3 points)

IV. Consider the following program:

```
1.  int main()  {
2.  int count = 0;
3.  int pid=0,pid2=0;
4.  if ( (pid = fork()) ) {
5.               count=3;
6.               printf("%d ", count);
7.  }
8.  if(count == 0)
9.        {
10.        count++;
11.        pid2=fork();
12.        printf("%d ", count);
13.      } else {
14.        if(pid2 = fork())
15.            count=count+2;
16.  }
17.
18.  if(pid2 && pid) {
19.      wait(NULL);
20.      count = count+4;
21.    }
22.      printf("%d ", count);}
```

Note that on line 4, (pid = fork()) in the same line executes the fork, assigns the return value to pid and uses that return value to evaluate the condition. If the condition is above zero, the if is taken, if it is zero (i.e., the child of the fork), it is not taken.

a. How many processes are created in this program? Explain. (2 points)

b. (2 points) Draw the process tree diagram

c. List the possible outputs of the program. Full credit if you get 3 correct outputs, bonus if you get all outputs if there are more than 3. (2 + 1 bonus points)

d. (2.5 points) If we delete line 16 (the wait) show one output that is possible in the new program that is not possible in the old program.