Homework Assignment #3 – Processes and fork() [50 pts]

You are expected to do your own work on all homework assignments. (See the statement of Academic Dishonesty on the Syllabus.)

Check the Syllabus for the late assignment policy for this course.

How to turn in?

Assignments need to be turned in via Laulima. Check the Syllabus for the late assignment policy for the course.

What to turn in?

You should turn in single **plain text** file named README.txt with your answers to the assignment's questions. Your file must be readable "as is" and points will be removed if the report is not readable.

Exercise #1 [20 pts]

Consider the following C program (assume that there are no compilation or execution errors of any kind):

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <stdlib.h>
4
5  int main(int argc, char **argv)
6  {
7      int count;
8
9      if ((argc != 2) || (sscanf(argv[1],"%d",&count) != 1)) {
10         fprintf(stderr,"Usage: %s <integer>\n", argv[0]);
11         exit(1);
12     }
13
14     pid_t pid1, pid2;
15     while (count > 0) {
16         pid1 = fork();
17         if (pid1 > 0) {
18             pid2 = fork();
19             count = count - 2;
20         } else if (pid1 == 0) {
21             count = count - 1;
```

```
25 }
```

Question #1 [2 pts]

How many child processes does this program create if the command-line argument is **1**?

Question #2 [3 pts]

How many child processes does this program create if the command-line argument is **2**?

Question #3 [12 pts]

Let *T(n)* be the number of child processes this program creates when its input is *n* (the answer to Question #1 above is thus T(1)).

Write an *recursive formula* (i.e., a *recurrence relation*) that gives *T(i)* as a function of one or more values of *T* for smaller input (i.e., smaller *i*). Explain your reasoning.

Feel free to double-check your formula by actually running the program and possibly augmenting it so that it allows you to count processes in whichever way you want.

Question #4 [3 pts]

Say the maximum number of processes that can be created on your machine is 100,000 (you can find out the actual number of your machine with `ulimit -u`).

What is the smallest value of the command-line argument for which the above program would experience failed `fork()` calls?

Exercise #2 [30 pts]

Consider the following C program (assuming that there are no compilation or execution errors)

```
1  #include <unistd.h>
2  #include <stdlib.h>
3  #include <stdio.h>
4  #include <sys/time.h>
5
6
7  int main(int argc, char **argv)
8  {
9          pid_t pid;
10
11         // Main process's PID=42
12
13         pid = fork(); // creates process with PID=36
14         if (pid == 0) {
15                 pid_t pid2 = fork(); // creates process with PID=99
```

```
19                      exit(0);
20              } else {
21                      sleep(30);
22                      printf("** ONE **\n");
23                      exit(0);
24              }
25      } else {
26              pid_t pid3 = fork(); // creates process with PID=71
27              if (pid3 == 0) {
28                      sleep(30);
29                      exit(0);
30              }
31              pid_t pid4 = fork(); // creates process with PID=123
32              if (pid4 > 0) {
33                      waitpid(pid3, NULL, 0);
34                      sleep(20);
35                      printf("** TWO **\n");
36                      exit(0);
37              }
38              exit(0);
39      }
40
41      sleep(20);
42      exit(0);
43 }
```

Assume that the PID of the main process is 42. Each call to fork() in the code is commented to show the PID of the newly created process (The PIDS are thus 42, 36, 99, 71, and 123). All calls to all functions / systems calls succeed.

In this exercise we assume that all operations take zero time but for sleep() (which sleeps a prescribed number of seconds) and waitpid(), which obviously might block for a while. We also assume that the execution starts at time zero. Therefore, in all the questions below, whenever a time is asked, it's always a perfect multiple of 10 (since all calls to sleep in the above program are for numbers of seconds that are multiples of 10).

Answer the following questions:

- **q1** [3 pts]: At what time is "** ONE **" printed to the terminal?
- **q2** [3 pts]: At what time is "** TWO **" printed to the terminal?
- **q3** [3 pts]: What is the PID of the process that prints "** TWO **"?
- **q4** [3 pts]: What is the initial PPID of the process with PID 99?
- **q5** [3 pts]: At what time does the process with PID 123 terminate?

- **q7** [5 pts]: What is the PPID of the process that prints **ONE** (when it prints it)
- **q8** [5 pts]: Which processes are “alive” (i.e., not terminated/zombies) at time 25