CS 303 – Lab 3
September 6, 2019
Maximum Points: 25

Note:

- The answers for questions 1 and 2 need to be shown to the TAs in the lab in order to get credit.
- The take home questions 3 and 4 are both due by 11:00 PM, Thursday, September 12, 2019 on Moodle.

Q1. (in lab question, 2 points) How many processes are created by the following code? Explain your answer by creating a process tree.

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

int main ( ) {
int i;
for(i = 0; i < 4; i++)
  fork ( );
return 0;
}
```

Q2. (in lab question, 5 points) Using a Linux system, write a C program that forks a child process that ultimately becomes a zombie process. This zombie process must remain in the system for at least 10 seconds. Process states can be obtained from the command

```
ps -l
```

The process states are shown below the S column; processes with a state of Z are zombies. The process identifier (pid) of the child process is listed in the PID column, and that of the parent is listed in the PPID column. Perhaps the easiest way to determine that the child process is indeed a zombie is to run the program that you have written in the background (using the &) and then run the command `ps -l` to determine whether the child is a zombie process. Because you do not want too many zombie processes existing in the system, you will need to remove the one that you have created. The easiest way to do that is to terminate the parent process using the kill command. For example, if the process id of the parent is 4884, you would enter

```
kill -9 4884
```

Q3. (take home question, 8 points), The Collatz conjecture concerns what happens when we take any positive integer *n* and apply the following algorithm:

n = *n*/2, if *n* is even.
or
n = 3 x *n* + 1, if *n* is odd.

The conjecture states that when this algorithm is continually applied, all positive integers will eventually reach 1. For example, if *n* = 35, the sequence is

35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1

Write a C program using the `fork()` system call that generates this sequence in the child process. The starting number will be provided from the command line. For example, if 8 is passed as a parameter on the command line, the child process will output 8, 4, 2, 1. Because the parent and child processes have their own copies of the data, it will be necessary for the child to output the sequence. Have the parent invoke the `wait( )` call to wait for the child process to complete before exiting the program. Perform necessary error checking to ensure that a positive integer is passed on the command line.

Q4. (take home question, 10 points) In Q3, the child process must output the sequence of numbers generated from the algorithm specified by the Collatz conjecture because the parent and child have their own copies of the data. Another approach to designing this program is to establish a shared-memory object between the parent and child processes. This technique allows the child to write the contents of the sequence to the shared-memory object. The parent can then output the sequence when the child completes. Because the memory is shared, any changes the child makes will be reflected in the parent process as well.

This program will be structured using POSIX shared memory as described earlier in the chapter. The parent process will progress through the following steps:

A. Establish the shared-memory object (`shm_open()`, `ftruncate()`, and `mmap()`).
B. Create the child process and wait for it to terminate.
C. Output the contents of shared memory.
D. Remove the shared-memory object.

One area of concern with cooperating processes involves synchronization issues. In this exercise, the parent and child processes must be coordinated so that the parent does not output the sequence until the child finishes execution. These two processes will be synchronized using the `wait()` system call: the parent process will invoke `wait()`, which will suspend it until the child process exits.