COM S 352

Assignment 3

Due: February 2, 2018

3.21 (30 points) The Collatz conjecture concerns what happens when we take any positive integer *n* and apply the following algorithm:

$$n = \begin{cases} n/2, & \text{if n is even} \\ 3 \times n + 1, & \text{if n is odd} \end{cases}$$

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.

3.22 (20 points) In Exercise 3.21, 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 in Section 3.5.1. 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.

4.7(10 points) Under what circumstances does a multithreaded solution using multiple kernel threads provide better performance than a single-threaded solution on a single-processor system?

4.8 (10 points) Which of the following components of program state are shared across threads in a multithreaded process?

a. Register values
b. Heap memory
c. Global variables
d. Stack memory

4.11 (10 points) Is it possible to have concurrency but not parallelism? Explain.

4.17 (10 points) The program shown in Figure 4.16 uses the Pthreads API. What would be the output from the program at LINE C and LINE P?

4.18 (10 points) Consider a multicore system and a multithreaded program written using the many-to-many threading model. Let the number of user-level threads in the program be greater than the number of processing cores in the system. Discuss the performance implications of the following scenarios.
a. The number of kernel threads allocated to the program is less than the number of processing cores.
b. The number of kernel threads allocated to the program is equal to the number of processing cores.
c. The number of kernel threads allocated to the program is greater than the number of processing cores but less than the number of user-level threads.