CS446 Assignment 1
Operating Systems

Question 1 [10 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.

Note:

- There are two deliverables for this question (submitted to Canvas):
  - Your `C` file named `q1.c`
  - A screenshot of the output.
- You may use your solution for lab test 1 as a source code for this question.
- You may use your solution of lab test 2 to pass parameters on the command line.
- To compile your code, enter: `gcc q1.c`
- To run your code, enter: `./a.out`

Question 2 [10 points]:

In Question 1, 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 establishing 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 3.7.1. (Chapter 3 slide #3.49) The parent process will progress through the following steps:

- Establish the shared-memory object (`shm open()`, `ftruncate()`, and `mmap()`).
- Create the child process and wait for it to terminate.
- Output the contents of shared memory.
- Remove the shared-memory object.

One area of concern with cooperating processes involves synchronization issues. In this question, 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.

Note:

- There are two deliverables for this question (submitted to Canvas):
  - Your `C` files are named `producerQ2.c` and `consumerQ2.c`
  - A screenshot of the output.
- You may use the example of the `POSIX` Shared Memory (discussed in class) as a source code for this question. To access the example:
  - Go to Canvas > CS 446 01 WI 23 > Files > Coding Examples > Chapter 3 > POSIX Shared Memory.
- To compile your code, enter the following:
  - `gcc -o producer producerQ2.c -lrt`
  - `gcc -o consumer consumerQ2.c -lrt`
- First, run the producer, which generates the sequence in shared memory:
  - `./producer <starting number>`
- Followed by running the consumer, which outputs the sequence:
  - `./consumer`

Question 3 [10 points]:

The Fibonacci sequence is the series of numbers 0, 1, 1, 2, 3, 5, 8, ... Formally, it can be expressed as:

$$fib_0 = 0$$
$$fib_1 = 1$$
$$fib_n = fib_{n-1} + fib_{n-2}$$

Write a multithreaded program that generates the Fibonacci sequence. This program should work as follows: On the command line, the user will enter the number of Fibonacci numbers that the program is to generate. The program will then create a separate thread that will generate the Fibonacci numbers, placing the sequence in data that can be shared by the threads (an array is probably the most convenient data structure). When the thread finishes execution, the parent thread will output the sequence generated by the child thread. Because the parent thread cannot begin outputting the Fibonacci sequence until the child thread finishes, the parent thread will have to wait for the child thread to finish.

Note:

- There are two deliverables for this question (submitted to Canvas):
  - Your `C` file named `q3.c`
  - A screenshot of the output.
- You may use your solution for lab test 2 as a source code for this question.
- To compile your code, enter: `gcc q3.c -lpthread`
- To run your code, enter: `./q3 <number of the Fibonacci numbers>`

Question 4 [10 points]:

(Multithreaded Sorting Application) Write a multithreaded sorting program that works as follows: A list of integers is divided into two smaller lists of equal size. Two separate threads (which we will term ***sorting threads***) sort each sublist using a sorting algorithm of your choice. The two sublists are then merged by a third thread—a ***merging thread***—which merges the two sublists into a single sorted list.

Because global data are shared across all threads, perhaps the easiest way to set up the data is to create a global array. Each sorting thread will work on one-half of this array. A second global array of the same size as the unsorted integer array will also be established. The merging thread will then merge the two sublists into this second array. Graphically, this program is structured according to the below figure:

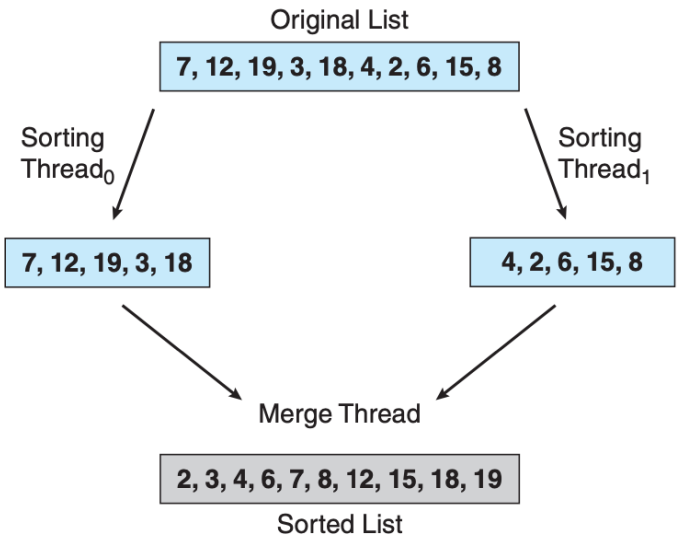

Figure: Multithreaded sorting

This assignment question will require passing parameters to each of the sorting threads. In particular, it will be necessary to identify the starting index from which each thread is to begin sorting. The parent thread will output the sorted array once all sorting threads have exited.

Note:

- There are two deliverables for this question (submitted to Canvas):
  - Your `C` file named `q4.c`
  - A screenshot of the output.
- You may use your solution for lab test 2 as a source code for this question.
- To sort, you may use any algorithm of your choice.
- To merge, you may use merge sort for merging the two sorted sublists.
- To compile your code, enter: `gcc q4.c -lpthread`
- To run your code, enter: `./a.out`