**# Project 5 : Multiple Processes**

**Due date: Sunday, December 17,2020 at 11:00 pm.**

*There will be no accepting of a submission beyond this time. Grading needs to begin the next day.*

---

\*\*You may discuss any of the assignments with your classmates and instructors (or anyone else) but

all work for all assignments must be entirely your own. Any sharing or copying of assignments will be considered cheating (this includes posting of partial or complete solutions on

Piazza, GitHub public repositories or any other public forum). If you get significant help

from anyone, you should acknowledge it in your submission (and your grade will be proportional to the part that you completed on your own).

You are responsible for every line in your program: you need to know what it does and why.\*\*

**Problem 1**

Write a C Program that writes all the odd numbers from 1 to 100,000 to the output file. Open the file, write the next value, and close the file. Measure how much elapsed time it takes to complete this processing. The process should open and close the file 50000 times. Yes, it is inefficient, but we are trying to force some I/O usage.

Next have the program fork and create 10 separate processes. Each process would process a range of 1 to 100,000 . One process would process 1 to 10000, another 10001 to 20000 etc. . Each process performs the exact same tasks , in the same way, as the first part of the problem, but write the output to 10 separate files, one for each process. Next read the ten files to make one file with all the number.. Measure the elapsed time it takes for all 10 processes to complete, and to create the output file.

The goal is to show that by breaking up the task across processes (that do I/O), the total elapsed needed will be shorter. Make sure you choose elapsed time function in C to track time, versus CPU Clock time.

If your program is not finding a difference between the approaches or is taking too long, you can modify the 100,000 up or down.

**Problem 2**

Enhance the program shown in class(and attached) to be a more robust quiz program. Have the parent program pick numbers between in some range, and an operator (+,-,\*) and see if the person can answer correctly within 5 seconds If a correct answer is provided within 5 seconds, show the next question and set the timer back to 0. An incorrect answer redisplays the question. If no answer with 10 seconds, the Timer process should notify the parent class, indicating that time is up. When time is up, add 1 to the count the number of missed questions and go to the next question.

You may need to notify the timer to start.

The parent program should prompt the user at the beginning of the program for the number of questions in the quiz. At the end of the questions, the program should print correct answers, incorrect answer (wrong answers within time frame), and missed questions (no correct answer within timeframe).