Program with multiple threads/processes

Submission Instructions

1. Submit two files for each problem: 1) your C program, and 2) a screenshot in jpg format showing that your program really works.

2. Name your scripts using the pattern NJITID#_Problem#.c, and name your screenshots using the pattern NJITID#_Problem#.jpg. NJITID# is the eight-digit NJIT ID (Not your UCID, Rutgers students also have NJIT IDs). Problem# is the problem number (e.g., 1, 2, 3, 4etc).

3. Submit individual files. DO NOT SUBMIT A ZIP FILE.

Objectives

1. To learn how to divide computation tasks and distribute tasks into different threads/processes.

2. To learn how to solve problems using multiple threads, particularly how to synchronize the execution of threads to avoid race condition.

2. To learn how to solve problems using multiple processes, particularly how to exchange data between processes.

3. To gain more experience with accessing files using memory mapping.

Problem 1: Write a program that creates **multiple worker threads** to calculate the sum of float point values saved in a file. Each thread calculates a partial sum, and the main thread adds up the partial sums. Your program will read the float point values from a file and print out the sum on the screen. The number of worker threads, as well as the file containing the float point values, are specified in the command line. For example.

problem_1   4   ./file_containing_values

You may use gendata.c attached with the assignment to generate the file. Your main thread can memory-map the file, such that the worker threads can access the float point values in the file easily.

Problem 2: Write a program that creates **multiple processes** to multiply two matrices of float point values. One matrix is saved in a file. The other matrix is an identity matrix (https://en.wikipedia.org/wiki/Identity_matrix) of the same size. Your program should save the product matrix into another file. Because the product should be the same as the first matrix, you can easily check whether your program produce correct results by comparing the product matrix against the first matrix. Use cmp command to compare (https://linux.die.net/man/1/cmp).

The number of processes, the file containing the first matrix, and the file saving the product matrix, should be specified in the command line. The identity matrix should be generated dynamically (determine the size based on the size of the input file (i.e., the one containing the first matrix)).

problem_2   4   ./file_containing_one_matrix ./file_saving_product_matrix

You may use gendata.c attached with the assignment to generate the file for the first matrix. To ensure correctness, your program can assume that each matrix is square (NxN). So, when you generate the input file, ensure that the number of float point values in the file is a square number (https://en.wikipedia.org/wiki/Square_number).

To check the results, use cmp

cmp ./file_containing_one_matrix ./file_saving_product_matrix

Read manual of cmp for more on how to use it (https://linux.die.net/man/1/cmp).

Hint:

Your processes can memory-map the files to share data easily.

Refer to the multi-threaded matrix multiplication program in the slides for how to multiple two matrices in parallel.

Problem 3: Write a program that create **multiple processes** to calculate an approximation of **π.** Refer to the multi-threaded program calculating an an approximation of **π** in the slides. The number of processes and the number of terms required to calculate the approximation should be specified in the command line (refer to the program in the slides). You may choose an IPC method (pipe, FIFO, or shm) you like in your implementation. But, pipe fits most. FIFO and shm are over-kills.