EECS 111 System Software Spring 2022
Project 1: Fork & Pthread

Due Date (April 28th, 2022 11:55 PM)

In this project, you will demonstrate how to manipulate multiple processes and threads to process input data in parallel. First, please read this description and understand this project. There are several requirements that you must satisfy to get a perfect score. Then, you may start programming for this project. You will see the following files in the ***p1_student*** directory:

1. `./input/`: This directory includes csv files for input data.
2. `./output/`: This directory is for output data.
3. `main.cpp`: This is a top file for this project.
4. `p1_process.h`: This is a header file to handle processes.
5. `p1_process.cpp`: This is a file to handle processes.
6. `p1_threads.h`: This is a header file to handle threads.
7. `p1_threads.cpp`: This is a file to handle threads.
8. `Makefile`: This is the compilation makefile.

In this project, you can change the given source code (p1_process.cpp and p1_thread.cpp). Or, you can also add any new header and source files. However, please keep in mind that you will need to apply changes as required to the `Makefile` and ensure that your program can be compiled with the `'make'` command. Compiling your code using the `'make'` command should yield the `p1_exec` file as the executable. Here is the description of how your program (`p1_exec`) should behave:

- `p1_exec` receives 2 arguments: 1) the specified number of child processes and 2) the number of threads per child process.
- Your program needs to create multiple processes to handle multiple files in parallel. Each child process handles at least one file. In other words, one single child process may handle multiple files.
- For each input file, you will need to first sort the input data read from a file with a multithreaded sorting algorithm (merge sort is suggested, see https://en.wikipedia.org/wiki/Merge_sort)
- Then, calculate the following statistics: Average, Median, Standard Deviation.
- Only the child process should create multiple threads to perform the sorting. So, the process hierarchy will be the following: Main process (spawns) -> Child processes (spawns) -> Threads
- You must not waste any resources, thus you should not create any non-working processes or threads.
- The results need to be saved to the output directory with the following filename: `<input filename>_stats.csv` and `<input filename>_sorted.csv` (i.e. os_stats.csv and os_sorted.csv).
    - NOTE: The separator here is an underscore (_) not a hyphen
- When a process is created and terminated, you should print a message (see below) with its process id to indicate the process has been created/terminated (you don't need to do this for threads).
- The main process must complete its behavior after its child processes.

The complete program will provide the following output. Pay attention to the order in which processes terminate. (The main process terminates last)

```
./p1_exec 2 4
Main process is created. (pid: xxxx)
Child process is created. (pid: yyyy)
Child process is created. (pid: zzzz)
...
Child process is terminated. (pid: yyyy)
Child process is terminated. (pid: zzzz)
Main process is terminated. (pid: xxxx)
```

The complete program will provide the following output files:

os-stats.csv content: (please display at least 3 decimal places)

```
Average,Median,Std. Dev
XX.XXX,XX.XXX,XX.XXX
```

os-sorted.csv content: (Notice that rank starts at 1, and don't worry about the formatting for the # of digits to display after the decimal)

```
Rank,Student ID,Grade
 1,XXXXXXXXXX,XX.XXX
 2,XXXXXXXXXX,XX.XXX
 3,XXXXXXXXXX,XX.XXX
 …
```

Submission Guidelines

You need to compress all the files in the folder into a single archive **.zip file** (no other format will be accepted) and upload it in Canvas. The deadline for uploading the files is the project deadline. Since your submissions will be processed by a program, there are some very important things you must do, as well as things you must not do:

1. When zipping your source code, use the following command in a Linux/Unix terminal while you are inside the directory with your source code:
    a. zip -j p1_<studentId>.zip <list of your source code files>
    b. Example for above:
    `zip -j p1_1234.zip main.cpp p1_process.h p1_process.c …`
2. It is imperative to keep the output format **the same** as what you are asked for.
3. You MUST USE only the **C++98** standard. If you use any new features like C++11 or C++14, your code will **not be graded**.
4. Make sure your code can be compiled and run. If we cannot compile your code, then your code will not be tested and graded.
5. Your executable filename must be ***`p1_exec`***.
6. Your submission must contain all your source/header files/Makefile in the root of the zip file. It is suggested to remove all the compiled object files and the executable file before submitting with the following command: `make clean`.
7. Since the input file sizes are large, you **MUST NOT** include input files in your submission.

8. A sample auto-grader script which is written in python is provided for you. You can use it to double-check your output format. However, the actual auto-grader script is different. It will use other test benches, will run in a sandbox, and will have a software similarity detection feature. Similarity detection features will compare your source codes with the other source codes available online. If they are similar enough, it will give us a warning about it. So, you can talk to each other about the project, and visit online resources, but you must write your own code.

Grading

1. (5%) Following the submission format
2. (15%) Compile your code with your Makefile without any problem.
3. (20%) Command-line output matches with the description for any type of input.
4. (60%) Output files match with the description for any input. **(Passes the auto-grader test)**
5. (-25%) If there is no multithreading
6. (-25%) If there is no multiprocessing

Note

- Even though you can generate a correct output, that does not mean that your code considers some extreme cases. You should verify your code with some corner cases.
- There will always be exactly five inputs with the names currently given to you in the project 1 package.
- If two students have the same grade, the order does not matter. As long as you get a small number of differences for the benchmarks it means that your code is working. The benchmarks that are going to be used for grading will not have reparation in them.
- The size of the input files is not constant. You should not make any assumptions about the length of the input files.
- The time out for sorting an input with 1 million entries is set to be 10 minutes. Make sure that your program can carry the task under this limit.

If you have any questions regarding the project, please ask them during the discussion session and/or on Canvas Discussion Board.

Hints

- Please refer to this page for more pthread usage information:
    - https://www.cs.cmu.edu/afs/cs/academic/class/15492-f07/www/pthreads.html
- In the context of this project, multithreaded sorting can be implemented by splitting the input file and assigning each thread a slice of the file
    - Make sure that when splitting the input file, every line in the file is sorted
- It may be helpful to create a class to use for managing threads and sorting, however, pthread_create will not be able to take class member functions. See the following page for a workaround
    - https://stackoverflow.com/questions/1151582/pthread-function-from-a-class