EECS 111 System Software Spring 2023
Project 3: Thread Scheduling

Due Date (Jun 11th, 2023 11:55 PM)

Instructions

In this project, your job is to understand thread conditional variables and write a code to solve thre ad scheduling problems. By unzipping the project file you will see the following files in the ***p3_student*** directory:

1. `main.cpp`: This is a top file for this project
2. `p3_threads.h`: This is a header file to handle threads
3. `p3_threads.cpp`: This is a file to handle threads
4. `utils.h`: This is a header file for utility functions
5. `utils.cpp`: This is a file header file for utility functions
6. `types_p3.h`: This is a header file for Person class
7. `types_p3.cpp`: This is a source code file for Person class
8. `Makefile`: This is a compilation script

For this project, you only need to complete the blank parts in `main.cpp`. After building using `mak e`, **you will have `p3_exec` as an executable.** It is suggested to remove all the compiled object files and executable files with the following command before submitting your code: `make clean`. In this project, your program will take 1 argument and print an error message if the wrong number of argum ents is given to it. This is an example of an error message.

```
[ERROR] Expecting 1 argument, but got (X).
[USAGE] ./p3_exec <number>
```

Problem Description

The assignment code will create 4 worker threads. Each thread has its own task execution time, per iod (deadline), and so on. (These parts are already implemented). You need to give 1 argument. Based on the argument value, your program uses a different scheduling function (0: FIFO scheduling, 1: Ear liest deadline first (EDF) scheduling, and 2: Rate-Monotonic (RM) scheduling). For example, if 0 is t he input argument, then your program should use FIFO scheduling.

- Your job is to implement two thread scheduling functions in main.cpp, `fifo_schedule`, and `edf_schedule`. RM scheduling is NOT mandatory. **You must read all of the starter codes a nd understand the behavior of the program before starting the assignment.**
    - You will need to implement the FIFO scheduling function first. Then check whether t he FIFO scheduling satisfies all the deadlines or not.
    - After that, you need to implement your own scheduling function based on EDF sched uling. Your EDF-based scheduling function MUST satisfy all the deadlines.
    - In order to satisfy all the deadlines, you will need to use preemption, this is supported in the starter code and can be accomplished as follows

```
// preempt the current running thread

preempt = 1;

pthread_cond_wait(&preempt_task, &mutex);

// thread is preempted here
```

    - If you complete the RM scheduling function, you will get extra points.

- You will need to understand *pthread conditional variables* to complete this assignment.
    - You will need to use `pthread_cond_wait` and `pthread_cond_signal`.
- The `ready_queue` vector stores all ready threads by their id.
- The `running_thread` variable keeps track of what thread is currently working, a value of -1 means no thread is currently working.
- Your scheduling function should schedule threads from the `ready_queue` vector by signaling t o the thread using the corresponding pthread conditional variable `resume[x]` where `x` is the thr ead id.
- **You can ONLY change main.cpp**. You MUST NOT change the other files.
    - **Do NOT** modify any of the print statements in the original code.

Submission

You need to compress all the files into a single archive .**zip file** (no other format will be accepted) and upload it. 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. It is imperative to keep the output format the **EXACT** same as what you are asked for.
2. You MUST USE only the C++98 standard. If you use any new features like C++11 or C++1 4, your code will not be graded.
3. Make sure your code can be compiled and run. If we cannot compile your code, then your cod e will not be tested and graded.
4. Your executable filename must be **p3_exec**.
5. Your code **MUST SHOW** a parallel behavior. Otherwise, your code will not be graded prope rly.
6. Your submissions must contain all your source code and Makefile in the **root** of the zip file.
7. 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 p3_<studentId>.zip <list of your source code files>
    b. Example for above:
    `zip -j p3_1234.zip main.cpp p3_threads.h p3_threads.cpp …`

8. The codes that the students submit will be compared with each other using a program. If they are similar enough, the program 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. (10%) Compile your code with your Makefile without any problem.
3. (5%) Command-line output matches with the description for any type of input.
4. (40%) Complete FIFO scheduling function.
5. (40%) Complete modified EDF scheduling function
6. (10%) Complete RM scheduling function.

Note

Even though you can generate correct output, that does not mean that your code considers extreme cases. You should verify your code with some corner cases as well.

If you have any questions regarding the project, please ask them in the discussion session.