Project 1: Evaluation of Thread Schedulers

Description

In this project, you will implement four thread schedulers for a uni-processor environment: (i) First Come First Served (FCFS) - no preemption, (ii) Shortest Remaining Time First (SRTF) - with preemption, (iii) Priority Based Scheduling (PBS) - with preemption, and (iv) Multi Level Feedback Queue (MLFQ) - with preemption and ageing.

The project is to be done in groups of one or two, either C/C++ or Java is allowed. And the project is due on Monday October 13 by 11:59PM.

Your scheduler should implement the following interface in C programming language:

```
void init_scheduler(int sched_type);
int scheduleme(float currentTime, int tid, int remainingTime, int tprio);
```

The above two functions should be provided in a file called 'scheduler.c'. You will be provided with a file called 'project1.c' which creates threads using the pthread library and invokes the above two functions. You are **NOT** supposed to make any edits to 'project1.c' - all your code goes in to 'scheduler.c'. The application (the main function in 'project1.c') will take an integer command line argument specifying the type of scheduler to be used ([0-3]: 0-FCFS, 1-SRTF, 2-PBS, 3-MLFQ) which will be passed to the init_scheduler function.

The scheduleme() function implements the actual scheduler functionality based on the initialized scheduler type in the init_scheduler() function. The scheduleme() function takes 4 input arguments from the calling thread:

1. **currentTime** - indicates the time at which the call to scheduleme() is made by some thread. No two scheduleme() invocations will have the same 'currentTime' value. However, it is possible that when a thread is executing your scheduleme() function, another thread invokes the same function, albeit with a higher 'currenttime'. Hence your implementation should be multithread safe. The 'currentTime' can take float values.
2. **tid** - indicates the thread identification number.
3. **remainingTime** - indicates the amount of CPU time requested by the thread. The 'remainingTime' will always be an integral number.
4. **tprio** - indicates the priority of the thread. 'tprio' can take a value between 1 and 5, 1 being the highest priority.

Your scheduler should maintain a global current time value which will be updated with the 'currentTime' value passed by the threads' scheduleme() invocations. Upon returning from the scheduleme() function, your scheduler should return the global current time value to the thread. Note that we are simplifying your implementation so that time quanta are always assigned at integral values, and these quanta will never start at non-integral time values (e.g. a thread will never relinquish the CPU for I/O at non-integral time values, even if it arrives in the system at non-integral time values).

Your scheduler should maintain a READY queue with all the threads that are currently waiting for the CPU. When a thread initially makes a call to the scheduleme() function, the 'remainingTime' argument specifies the amount of CPU time needed by the thread. When the scheduler decides to grant CPU for a

thread, the scheduleme() should return to the calling thread. The selected thread runs for exactly 1 integral time unit before making another call to scheduleme() function with the updated 'remainingTime' and 'currentTime' value. Scheduling decisions should be made only at these integral time units. The scheduler can now decide either to preempt the currently running thread and schedule some other thread from the READY queue or continue to schedule the current thread (depending on the scheduling criteria).
When a thread invokes scheduleme() function and the CPU is currently being used by another thread, your scheduler should block the calling thread until the next integral time unit (using a pthread routine). The blocked thread will be made to wait in the READY queue. When a thread invokes scheduleme() with a non-integral 'currentTime' and there is no thread currently running on the CPU and also there are no threads residing in the READY queue of the scheduler, the scheduleme() function should return the next highest integral 'currentTime' back to the thread - this is as if the time has advanced to that point. Your scheduler should not use any system timer interrupt functionality - the only notion of time for your scheduler is the 'currentTime' indicated by the threads. When a thread has finished its CPU requirements, it invokes the scheduleme() function with 'remainingTime'=0, after which the thread is no longer part of the READY queue - it has either left the system or gone for I/O. Return from the scheduleme() function indicates that the thread is either granted the CPU or has completed its CPU burst (when it calls the scheduleme() function with 'remainingTime'=0). Since multiple threads could be simultaneously accessing common data structures (like the READY queue), make sure you use pthread synchronization primitives at appropriate places to ensure atomicity. pthread calls should also be used to block threads until they are supposed to get the CPU.

Below is a brief description of the four kind of schedulers that you will implement:

1. **FCFS:** This scheduling scheme is non-preemption based and schedules the threads based on who arrives first at the scheduler. The threads are scheduled for the entire 'remainingTime' duration without preemption in between. Assume no two threads arrive at the same time.
2. **SRTF:** This scheduling scheme is preemption based and schedules the threads based on who has the least 'remainingTime' value. If two or more threads have the same 'remainingTime' value, then the thread with least 'currentTime' is chosen for scheduling. Further note that when a thread with a shorter execution/remaining time arrives in the system at a non-integral time value, it still waits for the next integral boundary before pre-empting the currently running thread.
3. **PBS:** This scheduling scheme is preemption based and schedules the threads based on who has the highest priority. If two or more threads have the same 'tprio' value, then the thread with least 'currentTime' is chosen for scheduling.
4. **MLFQ:** This scheduling scheme is preemption based with multi-level feedback queues. The READY queue should be implemented as a multi-level queue with 5 levels. The time quantum for the 5 levels (starting from the highest level) are 5, 10, 15, 20, and 25 time units respectively. Threads initially join the READY queue at the highest level. When a thread completes its time quantum on a particular level, it will be moved to the next lower level. Your scheduler should select threads from a lower level queue only when there are no threads in any of the higher levels. A thread entering the READY queue at the highest level will preempt threads at the lower levels at the next integral time unit. Within a level, you should use FCFS to select the threads. You will use Round-robin at the last level.

Note that tprio is only used in PBS, and is a don't care value for the other 3 schemes.

You are allowed to use the pthread (**pthread.h**) thread library to implement the scheduler functionality. You may find the following pthread functions useful when implementing the scheduler.

```
int pthread_cond_init(pthread_cond_t *, const pthread_condattr_t *);
int pthread_cond_signal(pthread_cond_t *);
int pthread_cond_wait(pthread_cond_t *, pthread_mutex_t *);
int pthread_cond_destroy(pthread_cond_t *);
pthread_t pthread_self(void);
int pthread_mutex_init(pthread_mutex_t*, const pthread_mutexattr_t *);
int pthread_mutex_lock(pthread_mutex_t* ) ;
int pthread_mutex_unlock(pthread_mutex_t* );
int pthread_mutex_destroy(pthread_mutex_ t* );
```

The file 'project1.c' parses an input file which indicates the thread arrivals and their execution times (burst length) and issues corresponding scheduleme() calls. 'project1.c' also spits out an output file which indicates the actual schedule order (Gantt chart) of the threads.
The file, **project1.c** would be provided to you on Sakai.

Testing your code
For testing your work, input files and their corresponding output files will be provided.

Input files are available in the directory 'TestInputs' which can be used by all schedulers.
As you can see, each line in the input file is represented in the following format,
"[arrival time][tab][thread id][tab][burst length][tab][priority]".
And each line of the output file is represented in the following format,
"[start time] - [end time]: [thread id]"

The output files corresponding to the input files are available in the directory 'TestOutputs' for each scheduler type.
The format of the output files is:
"[time slot beginning] - [time slot end]: T[Thread ID of thread running in that time slot]

Compiling and Running
To compile your code (project1.c and scheduler.c) run the provided script 'mymake'. Ensure that it has execute privileges. This will result in the generation of the binary 'out'.
To run the binary use the command: out [scheduler_type] [input_file]
where [scheduler_type] is 0,1,2 or 3 indicating FCFS, SRTF, PBS and MLFQ respectively. [input_file] is the path to the input file.