Project 1: Thread scheduling

Requirements:

Your team will create thread-based schedulers for an n-processor architecture, n > 1, using the following scheduling policies:

1. First Come, First Served (FCFS).
2. Shortest Remaining Time First (SRTF), without preemption.
3. SRTF, with preemption.
4. Priority Based Scheduling (PBS), with preemption.

You shall use standard C++ Object-oriented programming. You may use either Linux or Windows development platforms. An initial Visual Studio project will be provided for THIS PROJECT, but do not expect we will always provide such a well-defined starting point.

We have defined an abstract base class from which your scheduler needs to derive (this guarantees API compatibility and ensures your code will work with the TA's testbenches). This initial code is available on Canvas. YOU MAY NOT CHANGE THIS CLASS! The code in scheduler.h is NOT for you to change!

You and your team are to implement (1) the 'MyScheduler' class skeletonized for you in the Visual Studio project. Students must also (2) implement your own test bench to self-verify your scheduler works as required, evaluate their work, and to ensure schedulers implement policies correctly.

Projects will be graded by running the students' schedules against the TAs testbenches.

Teams of three students for this effort are required. Teams may be formed from registered members of either section.

What and When to submit: Your myschedule.h and myschedule.cpp files ONLY are to be submitted on Canvas by 11:59 pm Wednesday, March 1st. Do NOT submit your testbenches or any write-ups, but DO be sure all of your team's names are on both files in the comment headers.

Grading: If the TAs are unable to compile your projects by dropping in your .h and .cpp files, you will have a mandatory 50% penalty – that is, 50% is the maximum possible score for a good effort that doesn't compile. TAs will scale grades between 50% and 100% for those teams whose scheduler compiles and runs based on (20%) how well document and designed your source code is and (30%) how many of the TA's testbench scores are correct. The average project score is expected to be between 70 and 80%.

You ARE allowed to exchange testbenches by posting your testbenches to the Canvas discussion. You ARE NOT allowed to share source code! ANY case of plagiarism between teams will result in a zero being entered for all members of all teams involved on the first offense, and on the second offense the teams will be failed for the course. Reports of academic misconduct will be made in all cases.

```
////////////////////////////////////////////////
/*This file should not be modified by the students*/
////////////////////////////////////////////////

#include <iostream>
#include <cassert>
using namespace std;


/* Thread Structure, to be used to create thread obects
which will be processed by the scheduler. Students should
use their own data structure to store threads whic should
be defined in 'myschedule.h'.*/
struct ThreadDescriptorBlock {
          int tid;
          int remaining_time;
          int arriving_time;
          int priority;
};

enum Policy {
          FCFS,               //First Come First Serve
          STRFwoP,            //Shortest Time Remaining First, without preemption
          STRFwP,             //Shortest Time Remaining First, with preemption
          PBS                 //Priority Based Scheduling, with preemption
};

/*The 'schedule' class is an abstract class which will be
extended by 'myschedule' class in 'myschedule.h'.
Please note, the minimum number of CPUs allowed is 2.
*/
class Scheduler{
public:
          Scheduler(Policy p, unsigned int n) : policy(p), num_cpu(n) {
                    //Base class constructor
          }

          void Go() //This function initializes the scheduling process once all the threads are created.
          {
                    while(Dispatch())
                    {
                              //stuff happens
                    }
                    cout << "All the Threads have been executed !! .. Exiting Scheduler ..\n";
                    system("pause");
          }

          ///Dispatch schedules the next thread to execute based on scheduler policy & current work load
          virtual bool Dispatch() = 0;

          ///Method to create a new thread with initial conditions as defined.
          ///all threads must be called before the test bench invokes the Go() method to start simulation
          ///of the scheduler.
          virtual void CreateThread(int arriving_time, int remaining_time, int priority, int tid) = 0;

protected:
          const Policy policy; //To denote the scheduling policy

private:
          unsigned int num_cpu; //To denote number of processors
          ThreadDescriptorBlock **CPUs; //Pointer array to let processors access the threads
          int timer;//Global Timer for all the processors, single increment in timer equals one cpu cycle

};
```