EEEN30052 Concurrent Systems

Coursework Assignment

Part 3: Completing the Simulation

Dr. Peter Green
Department of Electrical and Electronic Engineering
University of Manchester
Office: Sackville E8b
peter.green@manchester.ac.uk

1. Introduction

This document outlines the final stage of the assignment.

2. The Final Task

This requires three enhancements to your programs:

(i) The time taken by each `Competitor` to run their section of the race to be stored in the corresponding `Competitor` object, which is then stored in the Thread Map. At the end of the race, `main` uses the Thread Map to determine the results of the race.
(ii) The Final Sync Object needs to be added.
(iii) The output code needs to be added.

2.1 Updating the Competitor Class

The `Competitor` class requires updating with a private data member `raceTime`, which represents the time taken for a specific `Competitor` to run their leg of the race. This is implemented by generating a random number in the appropriate interval as discussed in the first part of the assignment[1]. The random numbers should either be generated in main and passed to `run` via a parameter or generated by the Mersenne Twister code provided on Blackboard, at the start of the `run` function. `setRaceTime` and `getRaceTime` member functions should also be added to `Competitor`. The updated class interface is shown below[1].

```
class Competitor { // created in main and never updated, passed to a thread, placed in map
private:
    string teamName;
    string personName;
    int raceTime; // time in milliseconds
public:
    Competitor();
    Competitor(string tN, string pN);
    void setTeam(string tN);
    string getTeam();
    void setPerson(string pN);
    string getPerson();
    static Competitor makeNull();
    void printCompetitor();
    void setRaceTime(int rt);
    int getRaceTime();
};
```

The following functions need to be added to `Competitor.cpp`

```
void Competitor::setRaceTime(int rt) { raceTime = rt; }
int Competitor::getRaceTime() { return raceTime; }
```

After `raceTime` has been stored in the `Competitor`, the `threadID` and `Competitor` should be inserted into the `ThreadMap`. At the end of `main`, after the threads have terminated, main should iterate through the ThreadMap and determine the final results.

[1] Since the Mersenne Twister algorithm generates random `ints`, the random numbers generated should represent milliseconds rather than seconds.

The results should be printed out with one line per team, specifying the team name and the total time that its team members took to complete the race. The fastest team should be printed first, the second fastest second etc.

2.2 The Final Sync Agent

Like the other sync objects, the `FinishAgent` class inherits from `SyncAgent` as shown below. This class is incomplete with respect to your program in that **thread safety** is not considered. **You need to do this.**

```
class FinishAgent : public SyncAgent {
private:
    Results r;
    ThreadMap& tMap;
public:
    FinishAgent (ThreadMap& const t);
    void proceed();
    void pause();
    Results returnResults();
};
```

The files FinishAgent.h and FinishAgent.cpp are available in Programming/Assignments/Assignment Part 3.

The pause member function simply records the order of arrival of the threads (order of finishing of the last athletes in each team) in an object of class Results, shown below:

`Results.h`

```
class Results {
private:
    std::string resArr[4][2];
public:
    Results();
    void setPosition(Competitor c);
    void printResults();
};
```

The `proceed` member function does nothing and does not need to be called.

The files `FinishAgent.h`, `FinishAgent.cpp`, `Results.h` and `Results.cpp` are available in the folder Programming/Assignments/Assignment Part 3 on Blackboard. Note the comments about class `FinishAgent` being incomplete, that was made above.

3. The End of `main`

Once the threads exit their calls to pause in `FinishAgent`, the `run` functions terminate, and the threads are `joined` in `main`. main should use information from the `FinishAgent` and in the thread map to print out the results of the race.

These should be printed out with the team positions first, with one team per line.

Then the total time for each time should be printed, followed by the name and time of each team member on a separate line.

4. Submission Procedure

This will be detailed in a separate document which will contain an example of the expected output.