

## Table of Contents

<b>Requirements.....</b>	<b>1</b>
<b>Submission .....</b>	<b>2</b>
<b>Background .....</b>	<b>2</b>
<b>The Coursework.....</b>	<b>3</b>
<b>Task 1. (40% of marks) Complete the skeleton code .....</b>	<b>3</b>
<b>Task 2. (15% of marks). Implement the relay race simulation.....</b>	<b>3</b>
<b>Task 3 (10% of marks) Dropped Baton .....</b>	<b>4</b>
<b>Task 4 (15% of marks) Inter-local-process communication using UDP or WebSockets .....</b>	<b>4</b>
<b>Task 5 (20% of marks): Document comparing overheads from threads/UDP/WebSockets .....</b>	<b>4</b>

## Requirements

C++11 code should be portable across operating systems, but there can be some differences, such as which standard library headers are required. Your code will be tested using the Microsoft Visual Studio IDE on the computer clusters. The coursework assignment must be completed independently in your own time.

Malpractice - You must write your own code and follow the code structure outlined in this document. Therefore, your code should be carefully commented. You must carefully explain its role in the synchronisation protocols.

The upload deadline is given in the Blackboard submission item. The upload link is in the Programming/Assignment folder.

You must submit your code on time, as you will be tested on it in the lab afterwards. You have multiple submission attempts, and the last submission attempt will be discussed in that lab. **Submit your code before the deadline!**

Complete the tasks in sequence, and only begin the next task once the previous one is fully finished.

You will receive marks based on your progress with the assignment, depending on which task you have successfully completed and can discuss.

Submit only code that you fully understand and can explain. All marks are awarded for explaining your working code. Having a working code that you cannot explain and discuss will not earn you marks.

You are provided with the following files. Place all the files in the same directory as your program code.

- cs\_barrier.hpp and cs\_helper\_DoNotModify.hpp.

For TCP/UDP/WebSockets coding, you should use the asio library. You can find the asio.hpp and asio folder in the week 7 module in the Lab3\_IPC.zip.

## Submission

- Your source code files must start with a comment containing your full name and student ID.
- The single file that contains the multithreaded program (tasks from 1 to 3) should have the naming convention:
  - yourID\_yourNAME\_Tasks1to3\_vVersion.cpp.
- If you have completed task 4, also submit the programming file named as:
  - yourID\_yourNAME\_Task4\_vVersion.xxx.
- If you completed task 5, then also submit the pdf document in the naming convention:
  - yourID\_yourNAME\_Task5\_vVersion.pdf.
- You must compress all the files you need to run your programs and the final document as a single zip file using the naming convention:
  - yourID\_yourNAME\_Coursework\_vVersion.zip.
- You can make up to five submission attempts. Submit an early draft to ensure you meet the deadline. Only the final zip file you submit will be marked.
- You must submit your zip file before the deadline, or we will not be able to mark you in the subsequent lab.

## Background

The coursework focuses on C++ threading techniques rather than C programming, so we have provided a lot of the C++ code for you!

The coursework simulates two types of 100 m sprints. The first part simulates 16 runners all competing against each other over 100 m (the total for the teams is also calculated). The second part simulates four teams running in a 4x100 m relay race. There are four teams, each with four competitors. In both parts, you will use sixteen separate threads to represent each runner. If you complete those two parts, you can gain extra marks by extending the code.

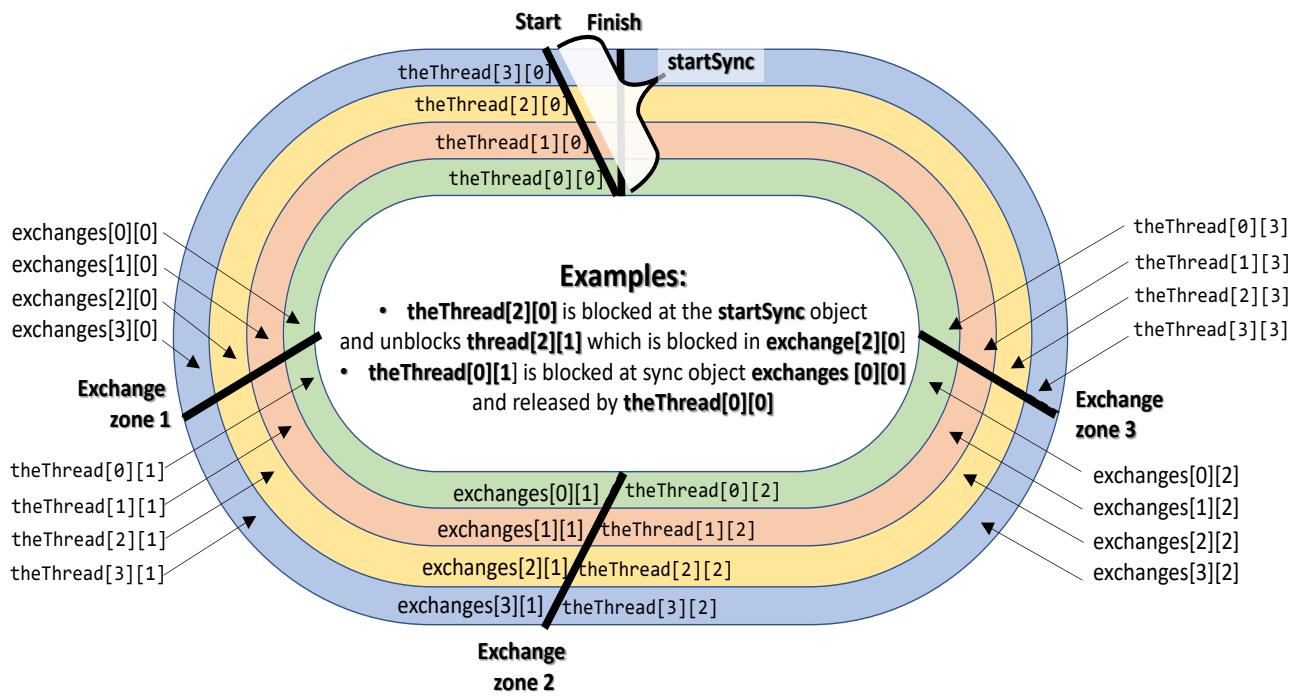


Figure 1 The 4 team, 4x100 m relay race

## The Coursework

Each athlete's progress is managed within a separate thread. That thread represents an athlete competing in the race. There is a 4x4 array of threads representing the four teams, each with four athletes. One dimension of the array indicates the team (e.g., Great Britain), and the other dimension denotes the runner within that team.

Each thread pauses for a random period, symbolising the time the runner takes to cover their 100 m segment.

Tackle the tasks in their order as each part builds on the previous.

### Task 1. (40% of marks) Complete the skeleton code

100 m sprint race running with 16 competitors. All 16 athletes are running simultaneously.

Each thread should execute the function named “thd\_runner\_16x100m”. This part concentrates on, starting the athletes and running them in parallel. Once all the athletes are in position, there is a pause before the starting pistol is fired. An example of the correct output is shown below.

The tasks are as follows. Their position and explanation are identified in the provided code.

- 0 Test the provided code to prove it compiles and runs.
- 1.1 Make the Random number generator thread-safe.
- 1.2 Make “thrd\_print” thread safe.
- 1.3 Make sure all threads wait until the race official starts the race.
- 1.4 Ensure all threads have been created before continuing (all athletes are on their starting blocks)
- 1.5 Wait for the starter gun to fire before the threads continue (athletes start running)
- 1.6 Wait for the random duration to simulate the athlete running the 100 m
- 1.7 Changing the random number generation to between 10 s and 12 s.
- 1.8 Change the demonstration a non-threaded call, to the multithreaded implementation
- 1.9 Apply a barrier in main code to wait for all the threads to be ready.
- 1.10 Wait for 2 seconds before the gun goes off.
- 1.11 Start all the competitors running.
- 1.12 Complete the program by joining the threads

### Task 2. (15% of marks). Implement the relay race simulation

The figure above illustrates the track layout for a four-team 4x100 m relay race. In this version, each thread should execute the function named “thd\_runner\_4x4x100m”. The main challenge is implementing the baton exchange – when runners complete their 100 m, they pass the baton to their next team member. The team whose first athlete crosses the finishing line wins the race. An example of the correct output is shown below.

Your tasks are as follows. Their position and explanation are identified in the provided code.

- 2.1 Ensure that the only winning thread claims to have won the race using an atomic variable.
- 2.2 Copy the code from thd\_runner\_16x100m for the two barriers
- 2.3 Create a “baton” condition\_variable with the previous athlete’s mutex and use their “bFinished” flag for the spurious check function.
- 2.5 Copy the code for the sleep\_for running time.
- 2.6 Ensure the "winner" runner only prints that they have won.
- 2.7 In the main function, change to calling the thd\_runner\_4x4x100m thread function and the associated function arguments.
- 2.8 Change this starter gun time from its fixed duration to a random delay between 1 to 3 seconds.
- 2.9 Ensure the program output is similar to the text given in the figure below.

Re-run of the women's 4x100 meter relay at the Tokyo 2020 Olympics.  
Briana Williams ready, Shelly-Ann Fraser-Pryce ready...  
The race official raises her starting pistol...  
GO !  
Teahna Daniels started, Elaine Thompson-Herah started, ...  
Imani Lansiquot (Great Britain) took the baton from Asha Philip (Great Britain)  
Leg 1: Asha Philip ran in 3.245 seconds. (Great Britain)  
Teahna Daniels (United States) took the baton from Javianne Oliver (United States)  
Leg 1: Javianne Oliver ran in 3.327 seconds. (United States)  
...  
Leg 4: Gabrielle Thomas ran in 1.553 seconds. (United States)

Team United States is the WINNER!

Leg 4: Shericka Jackson ran in 3.850907 seconds. (Jamaica)

#### TEAM RESULTS

Team Jamaica = 12.808 s

Team United States = 7.556 s

Team Great Britain = 11.826 s

Team Switzerland = 9.209 s

Program ended with exit code: 0

*Figure 2 Example output for the 4 Teams, 4x100 m relay race*

#### Task 3 (10% of marks) Dropped Baton

Simulate one of the athletes fumbling or dropping the baton (use a random number to see if they do).

- 3.1 Baton Fumble - If the percentage is below 20% down to 5% then they get a delay of 2 seconds to 0.5 seconds (ten times the percentage fraction).
- 3.2 Baton Drop - If the percentage is below 5% then their team is out of the race – disqualified.
- 3.3 Make sure the dropped baton information is clearly displayed in the text output.

#### Task 4 (15% of marks) Inter-local-process communication using UDP or WebSockets

Create a second program that communicates with the main program using either UDP or WebSockets.

This second program should display how the race is progressing, in addition to being shown on the main program. You can use an ASCII-based display like the main program, but you are also free to use your creativity.

You may write this program in a language of your choice, but you must be able to demonstrate it working on the computers in the cluster.

#### Task 5 (20% of marks): Document comparing overheads from threads/UDP/WebSockets

Create a document (up to 2 pages) in PDF format that examines communication overhead timings within shared memory for threads (intra-process), as well as inter-process communication via UDP and WebSockets. You should include quantitative timing data. The programs used to generate this data can be specifically written for this purpose (rather than using the relay race code).