CS450: Introduction to Parallel Programming

Assignment 4: Pthreads and Optimization

Due: July 3rd, 2022, 11:59 PM MST

Preliminaries

You are expected to do your own work on all homework assignments. You may (and are encouraged to) engage in discussions with your classmates regarding the assignments, but specific details of a solution, including the solution itself, must always be your own work. See the academic dishonesty policy in the course syllabus.

Submission Instructions

You should turn in an electronic archive (.zip, .tar., .tgz, etc.). The archive must contain a single top-level directory called CS450_aX_NAME, where "NAME" is your NAU username and "X" is the assignment number (e.g., CS450_a4_ab1234). Inside that directory, you should have all your code (no binaries and other compiled code) and requested files, named exactly as specified in the questions below. In the event that I cannot compile your code, you may (or may not) receive an e-mail from me shortly after the assignment deadline. This depends on the nature of the compilation errors. If you do not promptly reply to the e-mail then you may receive a 0 on some of the programming components of the assignment. Because I want to avoid compilation problems, it is crucial that you use the software described in Assignment 0. Assignments need to be turned in via BBLearn.

Overall, I should be able to compile your code using the Makefile that is provided. You are free to use a more C or C++ approach, depending on what you prefer. Just make sure that your code works, and that I can compile it.

Turn in a single pdf document that outlines the results of each question. For instance, screenshots that show you achieved the desired program output and a brief text explanation.

If you were not able to solve a problem, please provide a brief write up (and screenshots as appropriate) that describes what you tried and why you think it does not work (or why you think it should work). You must provide this brief write up for each programming question in the assignment. The questions may provide test scripts to (try) and validate your programs. When test scripts are used, show the output of both the program and the testing script (this is 2 executions, because the script takes your program output as input).

This pdf should be independent of the source code archive, but feel free to include a copy in the top level of that archive as well. Let me know if there are problems uploading multiple files to BBLearn.

Grading

All questions are weighted equally.

Overall Constraints

All programs must not deadlock, threads must join with main, and the program should exit gracefully (e.g., must return at the end of main). Do not have threads kill other threads or use other error-prone techniques.

Question 1: Semaphore Football Simulation

Using semaphores, you will simulate a team sport that has a goal target at the end of a field/court (e.g., football, handball, basketball). Each team must perform 5 consecutive passes to score a point. But, the passes may be intercepted by the other team. The first team to score 10 points wins. The pseudocode of the simulation is as follows:

```
while (no team has scored 10 points)
{
nextid = randThread()   // Generate next thread id that will receive ball/puck/etc.
performAction(nextid)   // Pass to teammate or interception
update counters and/or scores
}
```

Problem outline:

- You will create 10 threads, enumerated from $0, 1, \ldots, 9$. Threads with ids 0–4 are "players" assigned to Team 0, and threads with ids 5–9 are "players" assigned to Team 1.
- Each thread will call a function called `do_work()`.
- You will create an array of 10 semaphores, where one semaphore is assigned to each thread.
- The game begins with thread 0 on Team 0 (*hint:* initialize the semaphore assigned to thread 0 to 1, or post it before calling `do_work()`).
- **Only one thread is active at a time.** *Hint:* 9 threads will be waiting at a `sem_wait()`, while one is active.
- At each threadś turn, the thread calls the `randThread()` function (more on this function below).
- The `randThread()` function randomly generates a thread number from 0–9 and sends the ball, puck, etc., to that thread. If the thread belongs to the same team, a pass is made. If the thread belongs to the other team, then an interception occurs.
- For simplicity, a thread can make a pass to itself.
- To simulate the pass or interception, a `sem_post()` should be called to release the thread and allow it to perform its action.
- For a team to score a point, they must successfully pass the ball, puck, etc., five times in a row to their teammates.
- Keep track of the total score of Team 0 and Team 1.
- If an interception occurs, then reset a counter that keeps track of the number of consecutive passes.
- The game ends when one team reaches 10 points.
- You must not sleep in the program that you submit. However, feel free to use it for testing/debugging.

You are given a program called `a4_q1.cpp`. The program contains the following information:

- A struct of recommended parameters. You may modify the struct as you see fit. This information is given to help conceptualize what you may need to complete the problem.
- A function called `randThread()` and an associated random seed. Do not change the seed. The function takes as input the team of the calling semaphore (0 or 1), and a minimum and maximum semaphore number to generate. When calling the function use 0 and 10, as the values are generated in [0, 10).

You must format your output exactly as shown below. The output shows several key pieces of information:

- The action that completed: a pass, an interception, and scoring a point.
- The thread ids that perform the actions (e.g., pass from thread 0 to 3).
- The values of the counters that keep track of the number of consecutive passes.
- The values of the total scores.

Example output:

```
[Interception: Team: 0] Thread: 0, Intercepted by Thread: 7, Counter Team 0: 0
[Pass: Team: 1] Thread: 7, Pass to Thread: 9, Counter Team 1: 1
[Interception: Team: 1] Thread: 9, Intercepted by Thread: 0, Counter Team 1: 0
[Pass: Team: 0] Thread: 0, Pass to Thread: 2, Counter Team 0: 1
[Interception: Team: 0] Thread: 2, Intercepted by Thread: 7, Counter Team 0: 0
[Pass: Team: 1] Thread: 7, Pass to Thread: 7, Counter Team 1: 1
[Pass: Team: 1] Thread: 7, Pass to Thread: 5, Counter Team 1: 2
[Pass: Team: 1] Thread: 5, Pass to Thread: 7, Counter Team 1: 3
[Interception: Team: 1] Thread: 7, Intercepted by Thread: 0, Counter Team 1: 0
[Interception: Team: 0] Thread: 0, Intercepted by Thread: 9, Counter Team 0: 0
[Interception: Team: 1] Thread: 9, Intercepted by Thread: 3, Counter Team 1: 0
[Pass: Team: 0] Thread: 3, Pass to Thread: 4, Counter Team 0: 1
[Pass: Team: 0] Thread: 4, Pass to Thread: 0, Counter Team 0: 2
[Interception: Team: 0] Thread: 0, Intercepted by Thread: 8, Counter Team 0: 0
[Pass: Team: 1] Thread: 8, Pass to Thread: 8, Counter Team 1: 1
[Interception: Team: 1] Thread: 8, Intercepted by Thread: 4, Counter Team 1: 0
[Pass: Team: 0] Thread: 4, Pass to Thread: 4, Counter Team 0: 1
[Interception: Team: 0] Thread: 4, Intercepted by Thread: 9, Counter Team 0: 0
[Pass: Team: 1] Thread: 9, Pass to Thread: 9, Counter Team 1: 1
[Pass: Team: 1] Thread: 9, Pass to Thread: 8, Counter Team 1: 2
[Pass: Team: 1] Thread: 8, Pass to Thread: 7, Counter Team 1: 3
[Pass: Team: 1] Thread: 7, Pass to Thread: 8, Counter Team 1: 4
[Pass: Team: 1] Thread: 8, Pass to Thread: 7, Counter Team 1: 5
[Team 1 Scored A Goal!] Score Team 0: 0, Score Team 1: 1, Counter Team 1: 5
[Team 1] Counter Reset: 0
[Pass: Team: 1] Thread: 7, Pass to Thread: 5, Counter Team 1: 1
[Interception: Team: 1] Thread: 5, Intercepted by Thread: 4, Counter Team 1: 0
[Pass: Team: 0] Thread: 4, Pass to Thread: 2, Counter Team 0: 1
[Pass: Team: 0] Thread: 2, Pass to Thread: 2, Counter Team 0: 2
[Pass: Team: 0] Thread: 2, Pass to Thread: 3, Counter Team 0: 3
[Pass: Team: 0] Thread: 3, Pass to Thread: 1, Counter Team 0: 4
[Pass: Team: 0] Thread: 1, Pass to Thread: 3, Counter Team 0: 5
[Team 0 Scored A Goal!] Score Team 0: 1, Score Team 1: 1, Counter Team 0: 5
[Team 0] Counter Reset: 0
[Interception: Team: 0] Thread: 3, Intercepted by Thread: 6, Counter Team 0: 0
[Pass: Team: 1] Thread: 6, Pass to Thread: 5, Counter Team 1: 1
...
[Pass: Team: 0] Thread: 0, Pass to Thread: 3, Counter Team 0: 5
[Team 0 Scored A Goal!] Score Team 0: 9, Score Team 1: 7, Counter Team 0: 5
[Team 0] Counter Reset: 0
[Pass: Team: 0] Thread: 3, Pass to Thread: 2, Counter Team 0: 1
```

```
[Pass: Team: 0] Thread: 2, Pass to Thread: 2, Counter Team 0: 2
[Pass: Team: 0] Thread: 2, Pass to Thread: 2, Counter Team 0: 3
[Pass: Team: 0] Thread: 2, Pass to Thread: 3, Counter Team 0: 4
[Pass: Team: 0] Thread: 3, Pass to Thread: 4, Counter Team 0: 5
[Team 0 Scored A Goal!] Score Team 0: 10, Score Team 1: 7, Counter Team 0: 5
[Team 0] Counter Reset: 0
```

Question 2: Code Optimization

The file `a4_q2_to_optimize.cpp` contains code that is purposefully not well written, and that is not very efficient. This program computes, given 4 data-structures A, B, C and D, $D = A \times B + C$. This computation is implemented using lists (from the C++ standard library), which are filled with random integers. Overall, the lists contain 10,000,000 elements, and the code uses 20 threads (plus a semaphore and a pthread barrier). The number of elements and threads should not be changed for the submission. But feel free to change these values for your tests. There is a random number generator used to generate random values, do not change it or the seed. Use `genRandom()` to generate random integers to fill the data structures. Most of the optimizations should take place in the `doWork()` function, but you may adapt the rest of the code accordingly. The function `checkResult()` checks that the result of the computation is correct. You may change this function accordingly to your code, so it fulfills its role correctly (i.e., that it verifies that $D = A \times B + C$).
Objectives:

- Implement some performance measurement (e.g., execution time, total work computed by the threads, etc.) in the code.
- Improve the overall performance of the code. You should at least get a 2× speedup, and a speedup of 3× is "easily" reachable. For example, if the base code runs in 15s, your code should not take more than 7.5s on average (2× speedup), and ideally 5s (3× speedup).
- There are several possibilities to improve the performance, so you can be creative! And you should get some inspirations/solutions from the lectures.
- Overall, do not be afraid of making a lot of modications to the code. What matter the most are the correct result, and the performance (and a somewhat clean/smart code)
- This question is somewhat open-ended, so do not hesitate to ask for further indications/hints.
- You may not use compiler optimizations for this question (e.g., you cannot use the -O3 compiler flag).
- Submit the optimized version of the code in a file called `a4_q2_optimized.cpp`.