**CS 5/7343 Fall 2021**

**Programming Homework 2**

# Thread and Synchronization Programming

**Due Date: 3/24 (Thr) 11:59pm (NO late deadlines)**

The goal of this program is to use thread programming together with synchronization constructs available to simulate a simple multi-player game.

## *The game – rules*

Each game has a dealer (who is not involved in the game), and N players (we denote them as Player 0, Player 1, … Player N-1).

The dealer will start generating random numbers between 1-50 and put them into a FIFO queue of a certain size Q. The dealer will then generate numbers to fill the queue to half-full (round down if Q is odd)

After that, the dealer will keep generating numbers and insert in into the queue until the end of the game (see below). If the queue is full, the dealer will sleep for between 1-2 seconds and then examine the queue again. If the queue is still full, it will remove the first number and insert the number generated.

Once there are number(s) in the queue, the players will try to grab the number at the front of the queue. When a player grab holds of that number (by removing it from the queue), he/she will rest for a short period of time, and then add the value of the number to his total (initialized to 0) and then discard the number. The only exception is that every 5th number the player gets, the value is subtracted from the current sum, instead of added to it. In that case, the player should also examine the queue, and if the queue is not full, insert the number back to the end of the queue. (If the queue is full, then the number is discarded).

There is a target score (S) to be reached. Once a player reaches the target score, he/she stop grabbing number from the queue.

The game ends when two players reach the target score. The first player that reach the target score first is the winner, and the second one is declared second place. (It doesn't matter if he/she get a higher score than the winner).

Here by reach first, we means the first player that grabbed the needed number to reach the target score. For example, suppose player 1 and player 2 both have 95 points (and the target score is 100), and player 1 get the next number which is 6, than player 2 get the following number 7. Player 1 should be declared the winner.

(Hint: You need to ensure the first person that reaches the target score actually is being recorded as the first person that reaches the target score).

Notice that there is no "player 1's turn", "player 2's turn" etc. Once there is a number of the queue, every player is eligible to try to grab the number. The only exception is that once one gets the number, he/she has to wait till you either discard it or put the required number back on the queue before he can get the next number available. If he/she do it fast enough, he/she can the next number. However, there cannot be two (or more) players can simultaneously get the same number.

**Base care (85 points)**

For the base case, you are to implement the game via threads and synchronization constructs.

***Main process – dealer***

The main process of your program should serve as the dealer. It will take three command line parameters (via argc, argv). The numbers are (in order): N, Q and S. Your main thread should follow logic similar to described below:

```
Do all necessary pre-processing
Create all threads
Insert number to queue so that it is half full
Signal the game to start
Repeat
    Generate a new number x
    If the queue is not full
        Insert x to the end of the queue
    Else
        Sleep for 1 sec + x milliseconds (using the function nanosleep() )
        If the queue is not full
          Insert x to the end of the queue
    Check if the game should end
Until game ends
Tell all threads to finish
Wait for all threads to finish
Print the final score for each player, and who is the winner
```

***Threads -- player***

Each player should be represented by a thread. The logic of each thread should be as follows

```
Do all necessary pre-processing
Wait for the game start signal
Repeat
    If the queue is non-empty
        Get the next number (call it x)
        Sleep for 1 sec + x milliseconds (use the function nanosleep())
        Process scoring, and put a number back to the queue if necessary
    If target reached
        Signal to the main process that I am done.
Until game ends
Exit
```

One requirement for each thread: once a thread obtained a number, it cannot block other threads from obtaining the next number before it sleeps.

***Output***

Your program should output what is being output in the sample runs. Namely

- Whenever there is a change to the queue (insert/delete), print the whole queue (on a single line) after the change is made. You should print the following line
  - <x> <i> <y> : <list>
  - Where
    - <x> : either "Dealer" or "Player k" (where k is between 0 and N-1)
    - <i> : either "insert" or "delete"
    - <y> : the number to be inserted/deleted
    - <list> : is the content of the list after the operation. It should be a list of numbers from front to back of the list. Each number is separated
  - The following are sample lines:
    - Dealer insert 10 : 4 7 6 10
    - Player 3 delete 4 : 7 6 10

- For each thread, after the score is updated, it should print the updated score. You should print the following line:
  - <x> score : <y>
  - Where
    - <x> : Player k" (where k is between 0 and N-1)
    - <y> : the score after the update
  - Sample lines
    - Player 4 score 77
    - Player 3 score -5
- When a thread reached the target score, it should print the following
  - <x> reaches the target score : <y> (z)
  - Where
    - <x> : Player k" (where k is between 0 and N-1)
    - <y> : the score after the update
    - <z> : the target score
  - Sample lines:
    - Player 2 reaches the target score : 141 (140)

- When the dealer detected the end of the game it should print the following
  - "End of game detected"

- After all the threads has finished. The dealer should print information about the winners
  - Winner <x> with score <y>. Runner up <p> with score <q>.
  - Where
    - <x> , <p> : Player k" (where k is between 0 and N-1)

        - <y>,  <q> : score of the winner
    - Sample line:
        - Winner Player 3 with score 148, Runner up Player 8 with score 149

Notice that each output should have a '\n' at the end.

You should also have a print mutex to ensure only one thread is printing at any given time.

**Stage 1 (20 points)**

The base part allows one game to be played. In this part you are going to simulate playing multiple games with different players.

In this part you program should read in a name of a file via the command line parameter. Then the program should read that file. The file format is the following:

- The first line has four integers, which are N, Q, S  and the total number of players for all the games (we will denote it as p). (N is the number of slots for each game)
- Then each of the following p lines contains one string, which is the name of a player.

The program now consists of a list of games. The main program, in addition to being the dealer, will have to facilitate multiple games.

In this part, you should consider each thread is a slot for a player to play in the game. The main program will start by assigning players to the slots, and once all the slots are filled, then play a game among them (T is the number of numbers generated per game). At the end of each game, the winner will leave, opening a slot for the next player to fill in to start a new game.  (The runner up remains in the game)

The main process' logic will now be like this:

```
Do all necessary pre-processing
Create all threads
Create a list of all players
Assign the first N players to a slot
Repeat
    Play a game (using the logic of base part)
    Print the winner of that game (and the score)
    Replace that player with the next one
Until no more player left in the list
```

Notes:

- You should not pre-assign players to slots. Also, the player should join the game in the order of players in the file.
- ***You are not allowed to create new threads for each individual game, you should reuse the same threads that you created initially for every game.***

The logic of each thread will also need to be adjusted (left to you to figure it out).

*Output*

In additional to the output for the base case. You need to output two more things:

- Before the first game starts, print the list of all players for that game. You should output the following
    - Player for the first game: <list of names>
        - The list of names should be separated by a single blank space (in order of the slots)
- Before the start of each new game, you should output the following
    - New game: <x> replacing <y> at slot <z>
        - <x> is the name of the new player
        - <y> is the name of the player being replaced
        - <z> is the slot that the new player is assigned.
- Also, for all other output, replace "Player k" with the name of the player

**Stage 2 (20 points)**

For this stage, we modify our rules in the following way.  We will use the same input as of stage 1. However, instead of playing a series of game, you should now treat this as one continuous game.
The main process will assign slots for the first N players, and then start the game. However, whenever a player reaches the target score, he/she will immediately leave the game, and the main process will assign the slot to the next player. However, the game will continue for the other players. (Notice that if the thread does not have a player associated, it must wait until a player fill the slot before it can continue in the game).

Notice that once a player leaves, the other player's score is halved (round down). Also, while waiting for a player to join the game, the other players can continue picking up numbers. Also, it is possible that multiple threads can wait for players to come in. In such case, you should not predetermine an order of which thread will the next player join.

The game ends when there is no more player waiting, and then another player get to the target score..

I will leave it up to you to figure out how the program structures need to be updated.

***Output***

Here, in additional of what to be printed for stage 1, except you do not need to print the winners at the end of each game.

**Comment bonus (5 points)**

You will get a maximum of 5 points for providing good comments for the program. This includes:

- For each function, put comment to describe each parameter, and what is the expected output
- For each mutex/semaphore used, denote what is it used for

- For each critical section, denote when the start and end of that critical section is (you can name your critical section in various ways to distinguish among them (if necessary))
- For loops and conditional statements that do non-trivial work (you decide what is non-trivial) put comment before it describes what it does.

***What to hand in***

You should put all you source code into a zip file and upload the zip file to Canvas. For each stage you should have a separate program. You should have a separate program for each stage (base case, stage 1 and stage 2). You can reuse the code from one part to the other.

Also, you are allowed to use any publicly available third-party code for a linked list/FIFO queue. However, you need to include where you obtain the code in your comments.

**Full marks**

Full marks for 5000-level students is 100, For 7000-level students is 115.