# CS 5/7343 Fall 2021

# Programming Homework 2

# Thread and Synchronization Programming

**Due Date: 10/26 (Tue) 11:59pm (with late deadline 10/28 (Thu))**

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 1-40 and put them into a FIFO queue. The dealer will generate a total of T numbers (which is pre-defined before the start of the game). Every time a number is generated, the dealer will look at the queue, if there is not more than N numbers in the queue, it will insert the number at the end of the queue. Otherwise it will wait until there are no more than N numbers.

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 (call him/her player k) grab hold of that number, he/she will rest for a short period of time, and then determine whether he/she will score and whether he/she will need to put the number back to the (end of the) queue: (we denote the number as x)

- If x ≤ N, then the player score x points, and the number is discarded from play
- Otherwise, if x % N = k or x % N = (k+1) % N, then the player scores $\left\lfloor \frac{2x}{5} \right\rfloor$points (notice the division is integer division). Then the player will insert the number (x - $\left\lfloor \frac{2x}{5} \right\rfloor$) back to the end of the queue.
- Otherwise, the player does not score, and will insert the number x-2 back to the end of the queue.

The game ends when there is no more number left (either on the queue or in the player's hand). The player with the highest total score wins.

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 is no two (or more) players can simultaneously get the same number.

A couple of sample runs of the game is available for you to examine.

**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 two command line parameters (via argc, argv). The first one is N (number of players) and the second one is T (the number of numbers generated). Your main thread should follow logic similar to described below:

```
Do all necessary pre-processing
Create all threads
Signal the game to start
Repeat
    If the queue currently has N or fewer numbers
        Generate a new number and add it to the queue
Until T numbers generated
Wait for all threads to finish
Print the final score for each player
```

***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 singal
Repeat
    If the queue is non-empty
        Get the next number (call it x)
        Sleep for 1 + x millseconds (use the function nanosleep())
        Process scoring, and put a number back to the queue if necessary
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
- When the dealer push an item onto the queue, print the statement ("Dealer is pushing x to the queue" – where x is the value of the number being pushed). This line should be printed before x is pushed onto the queue.
- When a thread push an item onto the queue, print the statement ("Thread t pushing x to the queue" – where x is the value of the number being pushed). This line should be printed before x is pushed onto the queue.
- When a thread pop an item off the queue, print the statement ("Thread t pop x from the queue" – where t is the player number and x is the number. It should also print (on the same line), whether the player score, and if the thread needs to push any number back on the queue (and which number to push).

**Stage 1 (15 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 three integers, which are N, T, and the total number of players for all the games (we will denote it as p). (N is the number of players 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 up a slot for the next player to fill in to start a new 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
```

One note: You should not pre-assign players to slots. Also, the player should join the game in the order of players in the file.
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:

- Whenever a game ends, print the score for each player, and who win the game.
- Before the next game starts, print the list of all players for that game.

Stage 2 (20 points, required for 7343 students, extra credit for 5343 students)

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 score 100+ points, 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 are NOT reset to 0 (unlike stage 1, where a new game is started). 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 100+ points.

Notice that in this stage, T is ignored.

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 at the base case, whenever a player leaves, you should print the player that is leaving and his/her score. Also when a new player is put into the slot, you should print the name of the player and the slot that he/she is assigned to.

**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.