PLAGIARISM STATEMENT

We certify that this assignment/report is our own work, based on our personal study and/or research and that we have acknowledged all material and sources used in its preparation, whether they be books, articles, reports, lecture notes, and any other kind of document, electronic or personal communication. We also certify that this assignment/report has not previously been submitted for assessment in any other course, except where specific permission has been granted from all course instructors involved, or at any other time in this course, and that we have not copied in part or whole or otherwise plagiarised the work of other students and/or persons. We pledge to uphold the principles of honesty and responsibility at CSE@IITH. In addition, We understand our responsibility to report honour violations by other students if we become aware of it.

Name: J. Sai Nishith , P.V.Asish

Date: 23-02-2018

Signature: <J.S.N><P.V.A>

Report for Musical Chairs

musical_chairs function

- In this function, we are creating the player threads and the umpire thread, and we also calculate the time taken for the game, by using **chrono::steady_clock::now()** .
- Here, each thread calls their respective function, where they should be running.
- Like a player thread calls the **player_main** function, and umpire calls the **umpire_main function.**

umpire_main function

- This function is used by the Umpire thread.
- In this Function we take Input from the file, and the function loops for **nplayers times** .
- In every iteration we take input from the file and perform the respective operations.
- For **"lap_start"** input, we initialize the number of chairs occupied to be zero, and we set a flag (play) to **0**.
- For **"music_start"** input, we notify all the threads that are waiting after occupying the chair in the previous lap, we can notify them using the conditional variables **.notifyall()**.
- For **"umpire_sleep"** input, followed by the umpire_sleep input, we take one more integer as an input, which is the time of sleep for umpire, we use the command **this_thread::sleep_for(chrono::microseconds(timeOfSleep))** to make the umpire thread to sleep**.**

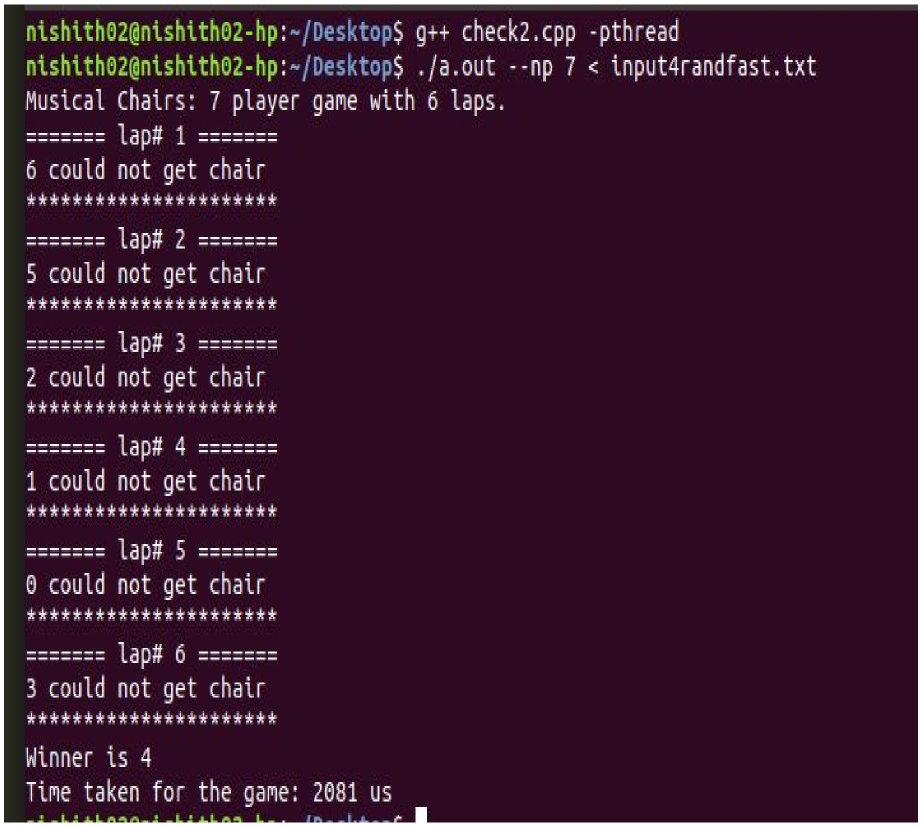
- For **“player_sleep”** input, we take the player id and time of sleep as inputs after taking the input “player_sleep”. After that, we update the array which stores the information regarding the time of sleep for player threads. The array is global and is initialized to zero. The time of sleep for the respective player will be stored in the array for that respective player id.
- For **“music_stop”** input, we acquire a lock, and we set the flag (play) to 1 which means the players are now allowed to acquire the chairs, and the umpire thread will be waiting until one of the players fails to acquire a chair, we use conditional variables **.wait()** for the umpire thread to wait.
- For **“lap_stop”** input, we increment the lap count and we update the total number of players in the game by decreasing the total players by 1, and in the case of final lap we wake up the winner thread as there will be no further input after the final lap.

player_main function

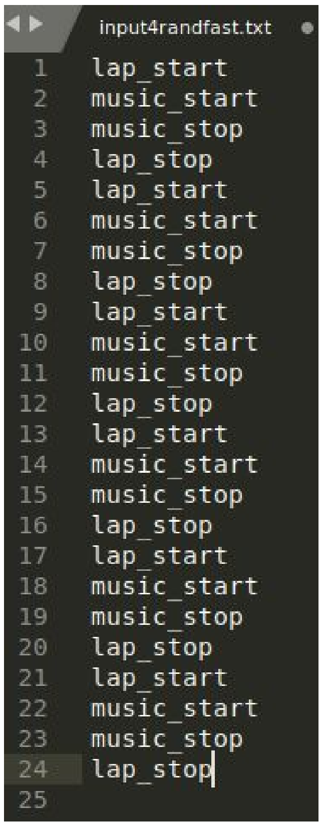
- Each thread will be running in an infinite while loop, and thread will exit the loop, whenever it loses.
- In every iteration, we first check is supposed to sleep as per the input. If the time of sleep of the corresponding thread is non-zero, then we make the thread sleep as per the entry in the array. As soon as the thread resumes, we set the time of sleep of that thread to zero in the array.
- If the total number of players is 1, then we declare that thread as the winner.
- If the flag(play) is 1 (means the players are allowed to acquire the chair), we first acquire the lock and check
  if the total number of chairs that are currently acquired is equal to 1 less than the number of players in the lap, then the thread will be declared as lost, and it will release the acquired lock and it will give a signal to the umpire(who is waiting in the **music_stop** block as mentioned above)
  else
  We just increment the number of chairs occupied by 1, and the corresponding thread will wait till we get a signal from the umpire thread.

OBSERVATIONS :

Case 1 :



```
nishith02@nishith02-hp:~/Desktop$ g++ check2.cpp -pthread
nishith02@nishith02-hp:~/Desktop$ ./a.out --np 7 < input4randfast.txt
Musical Chairs: 7 player game with 6 laps.
======= lap# 1 =======
6 could not get chair
*********************
======= lap# 2 =======
5 could not get chair
*********************
======= lap# 3 =======
2 could not get chair
*********************
======= lap# 4 =======
1 could not get chair
*********************
======= lap# 5 =======
0 could not get chair
*********************
======= lap# 6 =======
3 could not get chair
*********************
Winner is 4
Time taken for the game: 2081 us
```



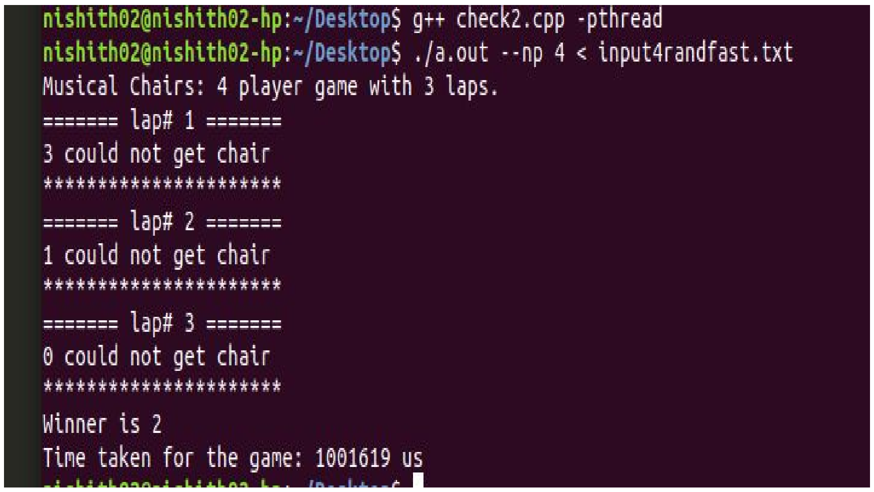
input4randfast.txt

```
lap_start
music_start
music_stop
lap_stop
lap_start
music_start
music_stop
lap_stop
lap_start
music_start
music_stop
lap_stop
lap_start
music_start
music_stop
lap_stop
lap_start
music_start
music_stop
lap_stop
lap_start
music_start
music_stop
lap_stop
```

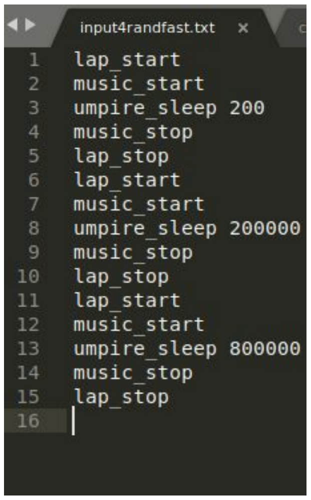
- For the above images, the input file is attached to the right, and the output of the program is to the left, This is the output for the case of having no sleep in the input.

- Here there is no sleep instruction for both umpire thead and player thread, so the output is completely random i.e, it depends on the scheduler. It’s non-deterministic.

Case 2 :



```
nishith02@nishith02-hp:~/Desktop$ g++ check2.cpp -pthread
nishith02@nishith02-hp:~/Desktop$ ./a.out --np 4 < input4randfast.txt
Musical Chairs: 4 player game with 3 laps.
======= lap# 1 =======
3 could not get chair
*********************
======= lap# 2 =======
1 could not get chair
*********************
======= lap# 3 =======
0 could not get chair
*********************
Winner is 2
Time taken for the game: 1001619 us
```

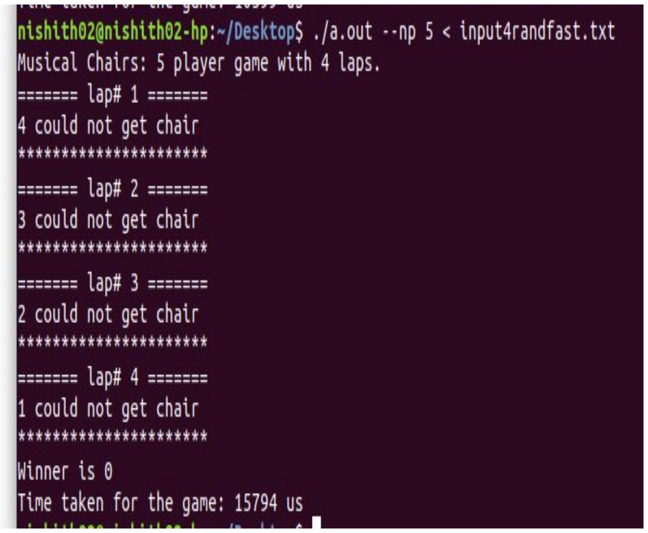


```
lap_start
music_start
umpire_sleep 200
music_stop
lap_stop
lap_start
music_start
umpire_sleep 200000
music_stop
lap_stop
lap_start
music_start
umpire_sleep 800000
music_stop
lap_stop
```

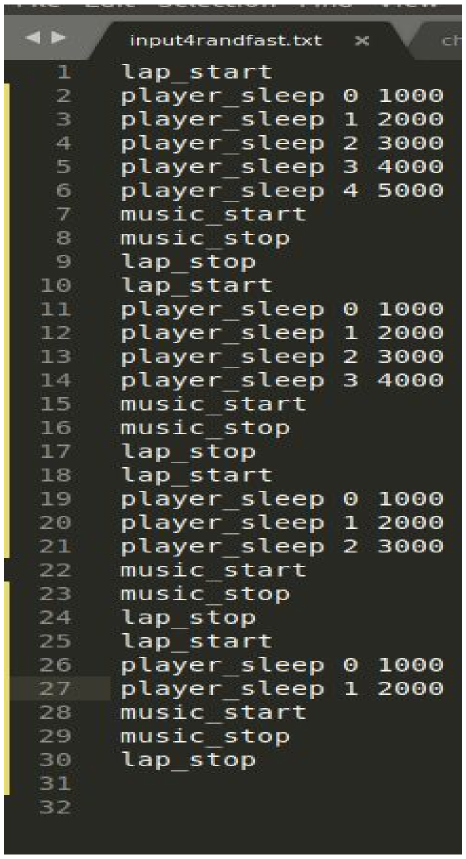
- The above is the output for the case, when there are sleep instructions for umpire thread.

- Here there is sleep instruction only for the umpire thread, the output is still random i.e, it depends on the scheduler, but the time taken for the completion of game increases.

Case 3:



```
nishith02@nishith02-hp:~/Desktop$ ./a.out --np 5 < input4randfast.txt
Musical Chairs: 5 player game with 4 laps.
======= lap# 1 =======
4 could not get chair
*********************
======= lap# 2 =======
3 could not get chair
*********************
======= lap# 3 =======
2 could not get chair
*********************
======= lap# 4 =======
1 could not get chair
*********************
Winner is 0
Time taken for the game: 15794 us
```

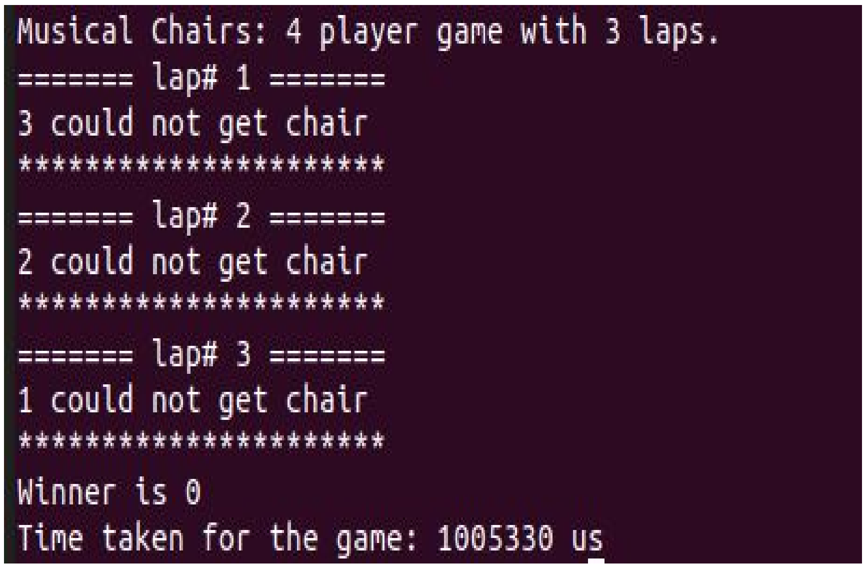


```
lap_start
player_sleep 0 1000
player_sleep 1 2000
player_sleep 2 3000
player_sleep 3 4000
player_sleep 4 5000
music_start
music_stop
lap_stop
lap_start
player_sleep 0 1000
player_sleep 1 2000
player_sleep 2 3000
player_sleep 3 4000
music_start
music_stop
lap_stop
lap_start
player_sleep 0 1000
player_sleep 1 2000
player_sleep 2 3000
music_start
music_stop
lap_stop
lap_start
player_sleep 0 1000
player_sleep 1 2000
music_start
music_stop
lap_stop
```

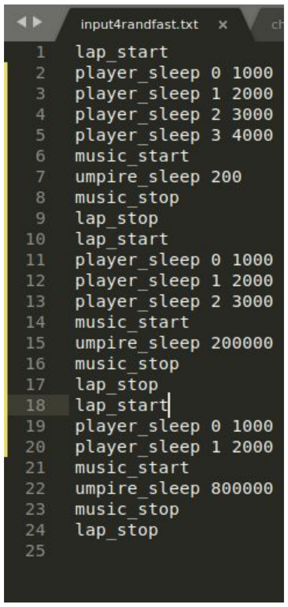
- The above is the output for the case, when there is sleep instruction only for the player threads, but not the umpire thread.

- In Every lap, the player thread with maximum sleep time will be out of the game.

Case 4 :



```
Musical Chairs: 4 player game with 3 laps.
======= lap# 1 =======
3 could not get chair
**********************
======= lap# 2 =======
2 could not get chair
**********************
======= lap# 3 =======
1 could not get chair
**********************
Winner is 0
Time taken for the game: 1005330 us
```



input4randfast.txt

```
lap_start
player_sleep 0 1000
player_sleep 1 2000
player_sleep 2 3000
player_sleep 3 4000
music_start
umpire_sleep 200
music_stop
lap_stop
lap_start
player_sleep 0 1000
player_sleep 1 2000
player_sleep 2 3000
music_start
umpire_sleep 200000
music_stop
lap_stop
lap_start
player_sleep 0 1000
player_sleep 1 2000
music_start
umpire_sleep 800000
music_stop
lap_stop
```

- The above is the output for the case, when there are sleep instructions to both the umpire threads and player threads.

- We can clearly observe in case 1, case 2 that the total time taken to complete the game is greater than the time of sleep of umpire thread and time of sleep of player threads.

- On executing the program for different number of player threads and different time quantum of sleep for both player threads and umpire threads, we didn’t encounter any deadlock.