PLAGIARISM STATEMENT <Include it in your report>

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:Sai Balaram K , D Krishna Pawan

Date: 25-2-20

Signature: KSB,DKP

We need to implement a musical chairs game with player threads and umpire thread with nplayers and nplayers-1 chairs at each lap a thread should exit and a chair should be removed by the umpire. The umpire can sleep between music start and music stop. The player sleep is also an optional command which is given between lap start and music start.

Implementation

In the program for musical chairs game, initialise global variables which will be used in both the umpire function and player function to indicate when players should sit and when to make players sleep (since they have to sleep when the music starts) using boolean variables **sit** and **start respectively**. Also declare the variables **lap, chairs, count** which correspond to the lap we are in, total number of chairs in a particular lap and the count of chairs occupied by players respectively. Declare two conditional variables **w1, w2** which are used for synchronisation of the player threads. Use mutex locks **player_sleep, umpire_sleep** which help in synchronisation and solving the critical section problem. Initialise a global array **player_sleep_time[MAX]** = {0} to store the player sleep times. MAX here is defined as 100000.

- **musical_chairs(int nplayers):**

  Assign the number of chairs using chair variable to nplayer-1 (according to the game rules). Create an umpire thread and pass the function umpire_main to and the only

parameter- ‘nplayers’ to the umpire thread. Create n player threads using thread players[nplayers] and pass the function player_main and corresponding plid as the argument to it in a for loop with ‘**i’** as iterator over the for loop. Then using another for loop, make the main thread wait for player threads and umpire thread using join().

- **umpire_main( ) function:**

  Run a while loop for nplayers-1 number of times. Using a string type variable input, read the input commands from the file.

1. **lap_start:** Make count =0 and sit =0. This implies the chairs occupied are currently zero and players aren’t yet to sit on the chairs.
2. **music_start**: start =1 is used to sleep the players which are specified to sleep after music starts and notify the players waiting on the condition variable w1 that music has started and hence you can continue with the execution.
3. **umpire_sleep**: Umpire is put to sleep for the specified amount of time using **“**this_thread::sleep_for(chrono::microseconds(<sleep_time>)**”.**
4. **player_sleep**: Update the global array of player sleep times so as to make the specified player sleep when the music starts.
5. **music_stop:** Declare a unique lock lck1 on mutex variable umpire_lock and make sit =1 telling that music has stopped and player threads are now supposed to acquire chairs. Also make the umpire thread wait on condition variable w2 on unique lock lck1 so as to make the umpire thread wait until all the threads finish in attempting to acquire the chairs.
6. **lap_stop:** Decrement the count variable(number of chairs decreases by one at the end of each lap) and increment the lap value by one and re-assign start value to zero implying the lap has finished and music is yet to start in the next lap. If now the current lap value is nplayers-1, this implies there’s only one thread left and hence we notify that winner thread waiting on condition variable w1 to print its plid as winner.

- **player_main( ) function:**

  The below if statements are kept in a while loop which player threads wait for music start condition boolean from umpire thread.

if (start) block:

Player threads execute this block only in between music start and lap stop. Now since the music has started, the player threads which are supposed to sleep will be put to sleep using the same function used to put umpire thread into sleep.

If (sit) block:

The music has stopped and now players are supposed to acquire chairs. Since incrementing count (number of chairs occupied) by various player threads can lead to synchronization issues, unique lock plck is used here. After a player thread acquires a chair, if there are still chairs left to be occupied, then the player thread simply waits on condition variable w1 until remaining player threads finish their attempt in acquiring the chairs. The last player thread will not receive any chair and hence it would execute the **if (count > chairs) block** where it prints its plid and a message that it couldn't acquire a chair, unlocks the lock plck lock so as the player threads can get ready for next round, notifies the umpire that the lap has finished and now umpire thread should take the control and then breaks from while loop and returns (the player thread terminates).

If the number of chairs is zero, then we have a winner thread who would then print his plid and a message that he is the winner and then breaks from the while loop and returns (the winner thread terminates).

Output:

1. **$ cat input4rand.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
```

Output:

1.

```
Musical Chairs: 4 player game with 3 laps.
======= lap# 1 =======
3 could not get chair
**********************
======= lap# 2 =======
0 could not get chair
**********************
======= lap# 3 =======
1 could not get chair
**********************
Winner is 2
Time taken for the game: 1259 us
```

2.

```
Musical Chairs: 4 player game with 3 laps.
======= lap# 1 =======
3 could not get chair
**********************
======= lap# 2 =======
1 could not get chair
**********************
======= lap# 3 =======
2 could not get chair
**********************
Winner is 0
Time taken for the game: 1294 us
```

3.
```
Musical Chairs: 4 player game with 3 laps.
======= lap# 1 =======
2 could not get chair
**********************
======= lap# 2 =======
3 could not get chair
**********************
======= lap# 3 =======
0 could not get chair
**********************
Winner is 1
Time taken for the game: 1410 us
```

2. **$ cat input4rand.txt**

```
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
```

Output:

1.
```
Musical Chairs: 4 player game with 3 laps.
======= lap# 1 =======
0 could not get chair
**********************
======= lap# 2 =======
1 could not get chair
**********************
======= lap# 3 =======
3 could not get chair
**********************
Winner is 2
Time taken for the game: 1001120 us
```

2.
```
Musical Chairs: 4 player game with 3 laps.
======= lap# 1 =======
0 could not get chair
**********************
======= lap# 2 =======
1 could not get chair
**********************
======= lap# 3 =======
2 could not get chair
**********************
Winner is 3
Time taken for the game: 1001315 us
```

3.
```
Musical Chairs: 4 player game with 3 laps.
======= lap# 1 =======
0 could not get chair
**********************
======= lap# 2 =======
3 could not get chair
**********************
======= lap# 3 =======
2 could not get chair
**********************
Winner is 1
Time taken for the game: 1001222 us
```

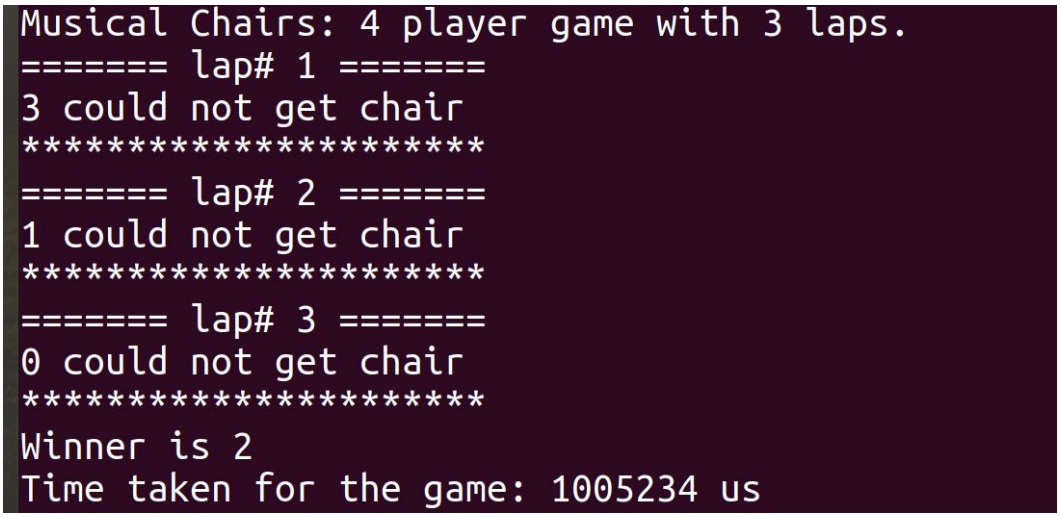
3.

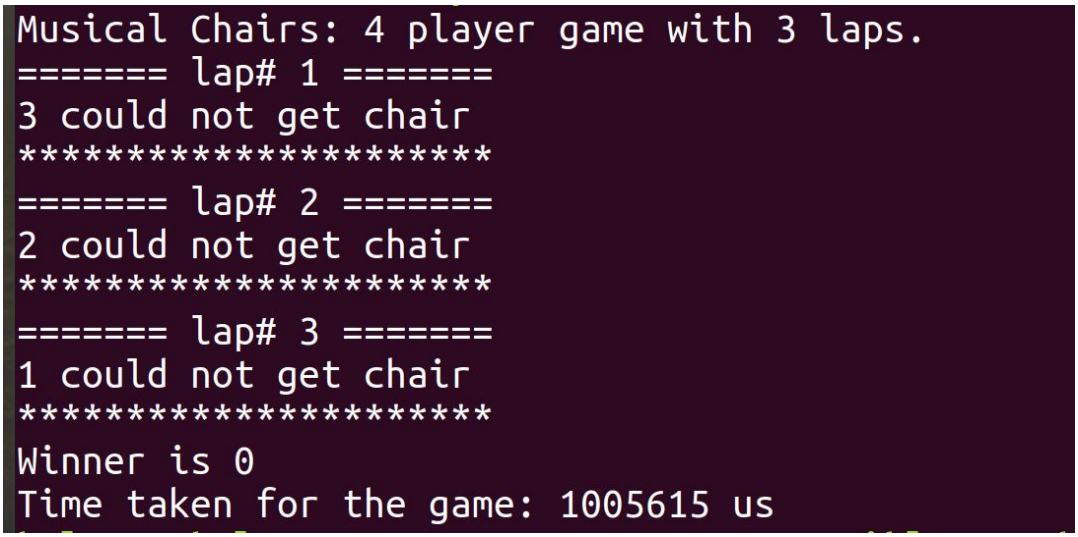
```
lap_start
player_sleep 0 1000
player_sleep 1 2000
player_sleep 2 3000
```

```
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
```

Output:

1.
```
Musical Chairs: 4 player game with 3 laps.
======= lap# 1 =======
3 could not get chair
**********************
======= lap# 2 =======
0 could not get chair
**********************
======= lap# 3 =======
2 could not get chair
**********************
Winner is 1
Time taken for the game: 1005624 us
```

2. 

3. 

Observations:

In case of no umpire_sleep or player_sleep commands, the outcome of which player thread terminates in each lap and which player thread wins the whole game is completely random as all the player threads are starting their execution simultaneously. Even if the umpire thread is made to sleep, the outcome of the thread that wins is still random. But if the player threads are made to sleep, then we may make some deductions based on the time quantums they are made to sleep for. If some player thread is made to sleep for a large time quantum, then the probability of it not acquiring a chair in that round increases. Hence, the probability of a player thread with less time

quantum winning is more as its probability of acquiring a chair in a round is more. For example, if the process is executed several times with input no. 3 given above, we observe that more often player thread 0 is winner as it has the least time quantum. Also since player thread 3 has a large quantum in the first lap, it is highly probable for it to exit out of the game first. But in case it didn't terminate in the first lap, then it's probability of winning the game is improved as now in the second lap it is not sleeping whereas other players are.The program when run cannot determine which player wins.