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: K VAMSHI KRISHNA REDDY, P SAI VARSHITTHA

Date:22-02-2020

Signature:Vamshi ,Varshittha.

Report:

We took the following points into concern before writing the code :

- Before 1st lap_start, we need to make sure that all player threads are created.
- Before lap_start, we need to make sure that all players are ready by keeping the umpire in the wait state.
- Between lap_start and music_start, there may occur at most n number of player sleep requests, just after music_start is scanned, make some of the player threads sleep, which got the sleep request and then, we need to be allowed to scan the further requests
- If umpire sleep occurs, then sleep the umpire thread for that much time and then scan the music_stop.

Following points tell about the implementation of the program :

- The unpire_main function scans all the inputs.
- The global variables used are :

  1. count - It is used to keep track of number of chairs filled.

  2. nplayers - Denotes number of players.

  3. musicstop - It is set to 1 when music is off and 0 when on.

  4. mutex variables

  5. player_sleep_times array that stores the sleep time of the players.

  6. isSleep boolean array that stores boolean values as to whether player identified by its index has sleep time or not.

  7.numberOfSleepyPlayers variable stores the number of players that want to sleep.

  8.conditional variables c1,c2,c3 - umpire waits on c1 and players on c2.

9.musicOn: it is an integer variable that is assigned 1 when music is on and zero when music is off.

- The **musical_chairs** function takes number of players as an argument and returns the time taken by the total game to complete. In this function, we create umpire threads as well as player threads as objects and pass them to umpire_main and player_main functions.
- The **umpire_main** function scans the input and does the required operations based on the string input it scans.

  lap_start refers to start of the lap. When music_start is scanned the umpire sets musicOn variable to 1 and notifies all the player threads that are waiting. However this is not required for the first lap, but further laps require notifying player threads.

  When music _stop is scanned, the musicstop will be assigned 1, and the umpire thread waits (so that it will not scan any further inputs)till it is notified by the terminating player(the player that didn't get chair).

  When lap_stop is scanned, the musicOn variable will be set to 0.If only one player is left, the program returns.

  When player_sleep is scanned, the player sleep times are updated in the corresponding indices of the player_sleep_array with respect to id as index.

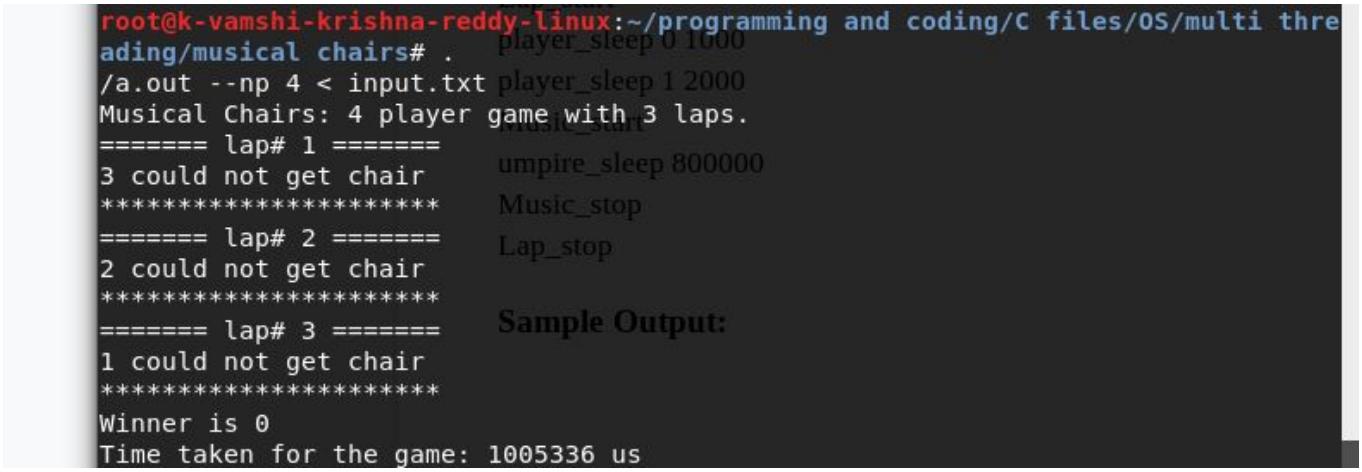
  When umpire_sleep is called, it lets the umpire sleep.

- In **player_main** function, the count variable is incremented and is used to know how many players took the chair. Each of the player checks if the count is less than (nplayers-1) increments the count variable and waits under conditional variable c2. The player that didn't get chair will reset the count to zero (so that count can be used in next laps) and prints its id telling it could not get the chair and notifies the umpire which is waiting under a conditional variable. When the number of players is 1, this function prints the winner's plid.
- **Usage of unique locks , mutex, condition variables for synchronization:**We used unique_locks and condition variables for synchronization between players and the umpire thread.
- **We used mutex locks to read as well as write to the global variables.**
- We used fflush(stdout) after every printf statement.

Sample Input:

```
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
```

Sample Output:



Explanation:

In 1st lap, since the sleep time of player 3 is more, by the time it wakes up, other 3 player threads had filled the 3 chairs and player 3 is eliminated.

In 2nd lap, since the sleep time of player 2 is more, by the time it wakes up, other 2 player threads had filled the 2 chairs and player 2 is eliminated.

In 3rd lap, since the sleep time of player 1 is more, by the time it wakes up, player threads had filled the chair and player 1 is eliminated.

Finally, 0 is the winner

Actually, there is a possibility that the player with less sleep time can also be eliminated (since we don't know the context switch timings, scheduling, and 1000 microseconds is a small-time)

Example output for the same input constraints:

```
root@k-vamshi-krishna-reddy-linux:~/programming and coding/C files/OS/multi thre
ading/musical chairs# ./a.out --np 4 < input.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: 1005253 us
```

But when there is a high difference in sleep times of players, then, with high probability, the player with high sleep time will be eliminated in each lap.

I ran it for sleep times **x10 times** of previous sample input around 5-10 times, I got the same output

```
root@k-vamshi-krishna-reddy-linux:~/programming and coding/C files/OS/multi thre
ading/musical chairs# ./a.out --np 4 < input.txt
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: 1501469 us
root@k-vamshi-krishna-reddy-linux:~/programming and coding/C files/OS/multi thre
ading/musical chairs# ./a.out --np 4 < input.txt
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: 1501757 us
root@k-vamshi-krishna-reddy-linux:~/programming and coding/C files/OS/multi thre
ading/musical chairs# ./a.out --np 4 < input.txt
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
```