REPORT
ASSIGNMENT - 2

1) In main function, musical_chairs function is called, which returns the time taken for the game.
2) In musical_chairs function, one umpire thread and n player threads are created. Also, this waits for all n players and umpire thread to terminate.
3) umpire_main is created for the umpire thread and player_main is for n player threads.
4) In umpire_main input file is read.
5) Every time when the string is read as lap_start, we update temp(number of chairs occupied) to zero.
6) When the string is read as music_start, we notify all the waiting player threads.
7) When the string is read as music-stop, a boolean ready is updated to true so that threads go and occupy their chairs. Also, the umpire threads wait till the players occupy their seats and one player leaves.
8) When the string is read as lap_stop, the number of players is reduced by 1. Also, on the last lap, we notify the winner thread which is waiting in the player_main.
9) As soon as ‘ready’ is made true in umpire_main all remaining player threads go one by one to occupy the chairs.
10) The player which did not get the chair makes ready false and notifies the sleeping umpire thread and leaves the game.
11) Umpire_sleep: In case if the input string read is ‘umpire_sleep,’ then we will make umpire thread to sleep for a given time in the input.
12) Player_sleep: In case if the input string read is ‘player_sleep,’ then in umpire_main, we will update the given time in the array indexed at given thread no. Each player thread will check the time in the array indexed at its number, and sleeps for that time.
13) In player_main, the final winner thread will print its number and then terminates.
14) Every time when the global variable is accessed (read or write), then for synchronization, we used locks.

Observations:

- Measurement of CPU time :

  For 4 players: Time taken for the game: 569 us
  For 5 players: Time taken for the game: 1500 us
  For 10 players: Time taken for the game: 10165 us
  For 14 players: Time taken for the game: 17426 us

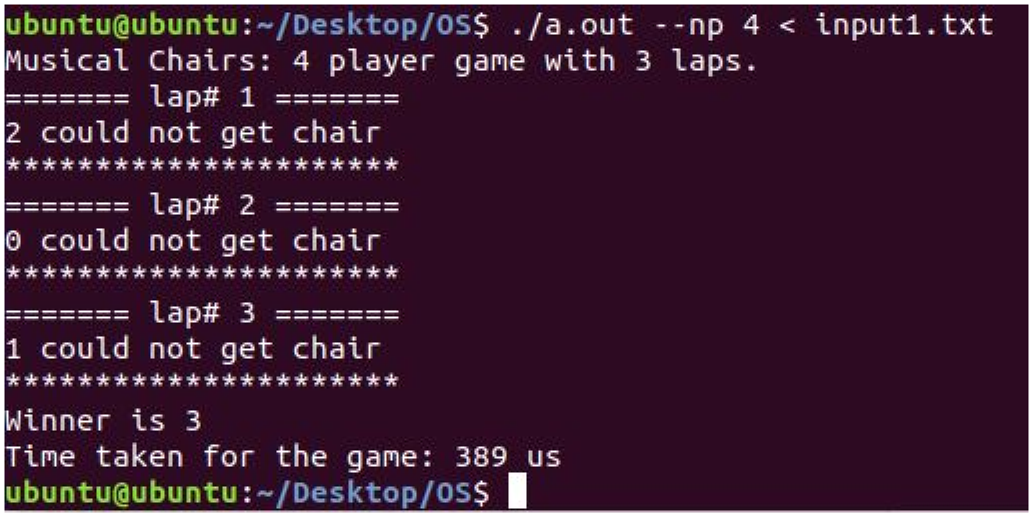
  => The time taken for the game has increased with no.of player threads.
- The chairs are occupied randomly by the threads and the winner is nondeterministic. It completely depends on job scheduler.

- When umpiresleep and playersleep are given in the input the program is a bit more when compared with the ones without sleep.
- We also observed that if the players are made to sleep for some time, then the player with longer sleep would lose the chair in the respective lap.

Sample Output:

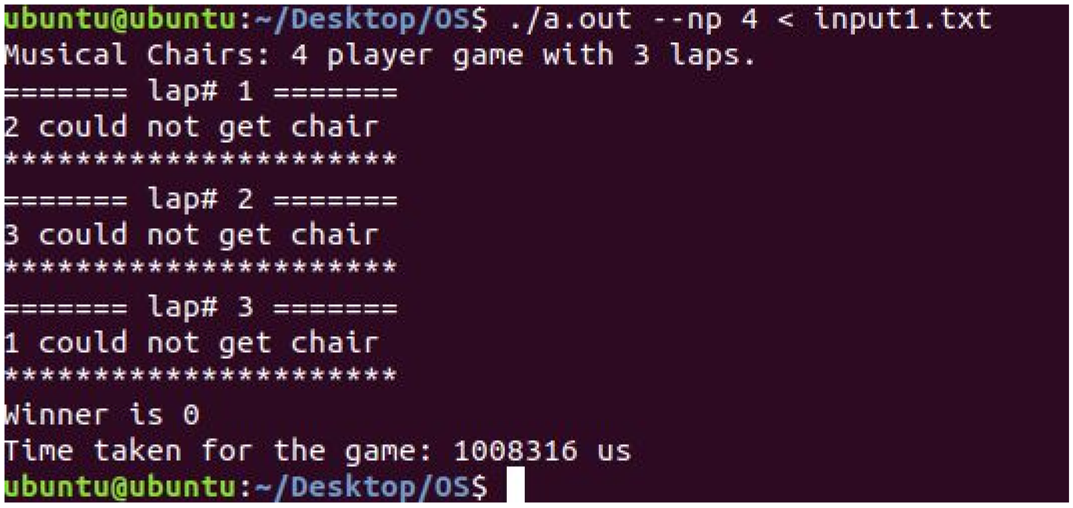
Note: The input is from the given document in the classroom.

- For 4 players with no sleep for players and umpire.



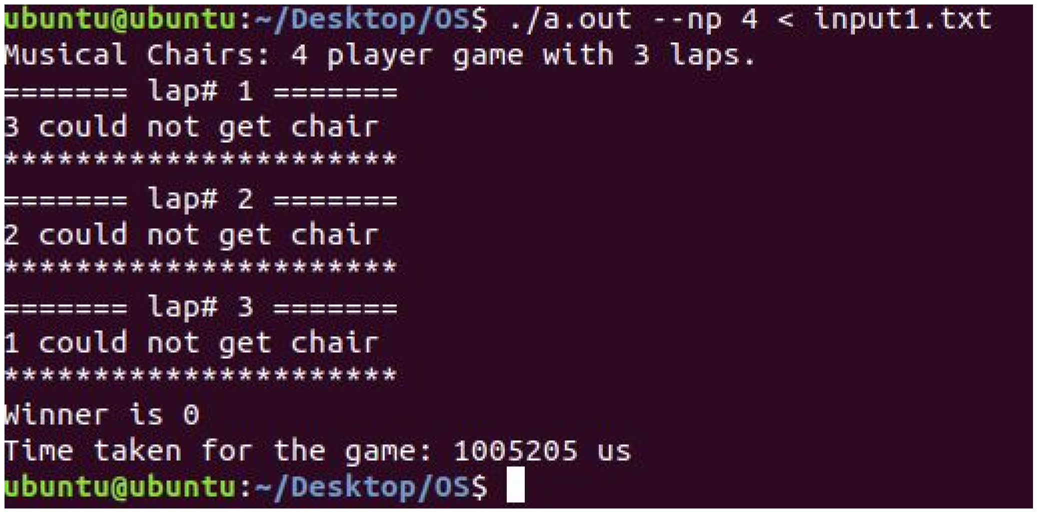
```
ubuntu@ubuntu:~/Desktop/OS$ ./a.out --np 4 < input1.txt
Musical Chairs: 4 player game with 3 laps.
======= lap# 1 =======
2 could not get chair
**********************
======= lap# 2 =======
0 could not get chair
**********************
======= lap# 3 =======
1 could not get chair
**********************
Winner is 3
Time taken for the game: 389 us
ubuntu@ubuntu:~/Desktop/OS$
```

- For 4 players with sleeping umpire.



```
ubuntu@ubuntu:~/Desktop/OS$ ./a.out --np 4 < input1.txt
Musical Chairs: 4 player game with 3 laps.
======= lap# 1 =======
2 could not get chair
**********************
======= lap# 2 =======
3 could not get chair
**********************
======= lap# 3 =======
1 could not get chair
**********************
Winner is 0
Time taken for the game: 1008316 us
ubuntu@ubuntu:~/Desktop/OS$
```

- For 4 players with sleeping players and the sleepy umpire.



```
ubuntu@ubuntu:~/Desktop/OS$ ./a.out --np 4 < input1.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: 1005205 us
ubuntu@ubuntu:~/Desktop/OS$
```