ASSIGNMENT REPORT

Musical Chairs

24.02.2020
OPERATING SYSTEMS

G. Vishal Siva Kumar

CS18BTECH11013

T. Krishna Prashanth

CS18BTECH11045

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: T. Krishna Prashanth, G. Vishal Siva Kumar

Date: 25 -Feb - 2020

Signature: TKP, GVSK

PROBLEM

The school arranges n-1 chairs at random places in the ground. n players will be standing in the ground at random places. The game iterates n-1 times. In each of the ith lap, the umpire starts lap by ensuring that all players are ready. Once all players are ready the umpire starts the game by starting music. Each of the players keeps running around in the ground until the music plays. When music is stopped by the umpire, each of the players quickly chooses a chair to occupy it. But if the chair is already occupied, the player steps back and chooses another free chair, and so on. One player will not get any chair as there is one less chair, so this player is out of the game. The umpire stops the lap by ensuring that one player is knocked out, all other players are up from the chairs and a chair is removed from the game to start the next lap.

Write a program to mimic this game by creating one thread for the umpire and one thread for each of n players. The umpire thread works with all the player threads in lockstep synchronization. Each iteration, one player thread exits. The main thread waits for umpire thread and the n players to join back and declares who has won the game.

APPROACH

We used std::mutex, std::unique_lock, std::conditional variable for synchronization. And std::thread for thread creation. The umpire thread works with all the player threads in lockstep synchronization.

WORKING

We create one umpire thread and ‘n’ player threads. Each player thread executes player_main function and umpire executes umpire_main function. Umpire takes the input commands and signals the players accordingly. Player_main has a while loop that all player threads execute. Umpire starts the lap by ensuring all the players are ready, then players start running as the music plays. When music_stop command is given, the umpire signals all the players to start acquiring chairs by setting the global variable ‘i’ to 1 and umpire waits in the condition variable ‘music’. All player threads acquire chairs

and wait in the conditional variable ‘laps’. The last thread that could not acquire chair, prints the result of the lap, updates the running player count, notifies the umpire waiting on the condition variable, and then breaks out of the while loop and terminates. In this manner, umpire ensures that one player is kicked out and restores the variable ‘i’ to 0 so that other players don’t try to acquire chairs. Now, umpire stops the lap and notifies all the player threads for the next lap. This happens till all but one player thread is left. When umpire_sleep <sleeptime> is taken as input, the umpire sleeps for the specified time. For player_sleep input, we maintain a global array with size as the number of players that contains the sleep time corresponding to each player indexed with the respective player id (plid). When umpire reads player_sleep, it populates this array with specified sleep time. Each player before starting lap checks this array and sleeps for the respective time in the array. After waking up, it puts zero in its position of the array and starts execution.

DATA

INPUT TYPE	NUMBER OF PLAYER THREADS	UMPIRE SLEEP	PLAYER SLEEP	TIME TAKEN(us)
Input rand fast	500	no	no	4364960
Input rand	4	yes	no	1001596
Input player sleep	4	no	yes	11084
Input all	7	yes	yes	2009483

TEST CASES

1

```
krishna@krishna-Inspiron-7572: ~/Desktop/OS/musicalchairs
File  Edit  View  Search  Terminal  Help
krishna@krishna-Inspiron-7572:~$ cd Desktop/OS/musicalchairs/
krishna@krishna-Inspiron-7572:~/Desktop/OS/musicalchairs$ g++ uncharted.cpp -pth
read -o musicalchairs
krishna@krishna-Inspiron-7572:~/Desktop/OS/musicalchairs$ ./musicalchairs --np 5
00 < input500randfast.txt > output.txt
krishna@krishna-Inspiron-7572:~/Desktop/OS/musicalchairs$ ./musicalchairs --np 5
00 < input500randfast.txt > output.txt
krishna@krishna-Inspiron-7572:~/Desktop/OS/musicalchairs$
```

```
Open ▾                    output.txt
                    ~/Desktop/OS/musicalchairs
Musical Chairs: 500 player game with 499 laps.
======= lap# 1 =======
499 could not get chair
**********************
======= lap# 2 =======
481 could not get chair
**********************
======= lap# 3 =======
473 could not get chair
**********************
```

.....

```
======= lap# 497 =======
78 could not get chair
**********************
======= lap# 498 =======
171 could not get chair
**********************
======= lap# 499 =======
225 could not get chair
**********************
Winner is 417
Time taken for the game: 4364960 us
                                    Plain Text ▾   Tab Width: 8 ▾
```

2

```
krishna@krishna-Inspiron-7572:~/Desktop/OS/musicalchairs$ ./musicalchairs --np 4
 < input4rand.txt > output.txt
krishna@krishna-Inspiron-7572:~/Desktop/OS/musicalchairs$ ./musicalchairs --np 4
 < input4rand.txt > output.txt
krishna@krishna-Inspiron-7572:~/Desktop/OS/musicalchairs$
```

Open ▾ | input4rand.txt ~/Desktop/OS/musicalchairs | Save

input4rand.txt × | output.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
```

Plain Text ▾ Tab Width: 8 ▾ Ln 16, Col 1 ▾ INS

Open ▾ output.txt ~/Desktop/OS/musicalchairs Save

input4rand.txt × output.txt ×

```
Musical Chairs: 4 player game with 3 laps.
======= lap# 1 =======
0 could not get chair
**********************
======= lap# 2 =======
3 could not get chair
**********************
======= lap# 3 =======
1 could not get chair
**********************
Winner is 2
Time taken for the game: 1001596 us
```

Plain Text ▾ Tab Width: 8 ▾ Ln 12, Col 36 ▾ INS

3

```
krishna@krishna-Inspiron-7572:~/Desktop/OS/musicalchairs$ ./musicalchairs --np 4
 < input4deterministic.txt > output.txt
krishna@krishna-Inspiron-7572:~/Desktop/OS/musicalchairs$ ./musicalchairs --np 4
 < input4deterministic.txt > output.txt
krishna@krishna-Inspiron-7572:~/Desktop/OS/musicalchairs$
```

Open ▾ input4deterministic.txt ~/Desktop/OS/musicalchairs Save

```
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
```

Plain Text ▾ Tab Width: 8 ▾ Ln 21, Col 9 ▾ INS

Open ▾ output.txt ~/Desktop/OS/musicalchairs Save

```
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: 11084 us
```

Plain Text ▾ Tab Width: 8 ▾ Ln 12, Col 34 ▾ INS

4

```
krishna@krishna-Inspiron-7572:~/Desktop/OS/musicalchairs$ ./musicalchairs --np 7
 < input7deterministic.txt > output.txt
krishna@krishna-Inspiron-7572:~/Desktop/OS/musicalchairs$ ./musicalchairs --np 7
 < input7deterministic.txt > output.txt
krishna@krishna-Inspiron-7572:~/Desktop/OS/musicalchairs$
```

Open ▾ input7deterministic.txt ~/Desktop/OS/musicalchairs Save

```
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
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
```

```
Musical Chairs: 7 player game with 6 laps.
======= lap# 1 =======
3 could not get chair
**********************
======= lap# 2 =======
4 could not get chair
**********************
======= lap# 3 =======
6 could not get chair
**********************
======= lap# 4 =======
2 could not get chair
**********************
======= lap# 5 =======
1 could not get chair
**********************
======= lap# 6 =======
0 could not get chair
**********************
Winner is 5
Time taken for the game: 2009483 us
```

OBSERVATIONS

1. With no sleep commands, the program runs normally and the winner is non-deterministic.
2. When umpire sleeps, the players get no signal so, keep running (or waiting). It just delays the program.
3. When players sleep, the umpire does not wait for all threads to be ready for executing music_start instruction because umpire only does synchronization and waits for all players to be ready when lap_start is called, not during music_start.
4. So, in a lap, when music_start is called, it just signals the players irrespective of whether they are sleeping.
5. But again, umpire synchronizes all players when lap_stop is called.

6. However, in the 3rd test case above, the order is deterministic since, in each particular lap, the thread with larger id waits for a longer time than the others.