Programming Assignment-2: Operating Systems-II

By CS18BTECH11041 and CS18BTECH11017

PLAGIARISM STATEMENT

I certify that this assignment/report is my own work, based on my personal study and/or research and that I 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. I 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 I have not copied in part or whole or otherwise plagiarised the work of other students and/or persons. I pledge to uphold the principles of honesty and responsibility at CSE@IITH. In addition, I understand my responsibility to report honour violations by other students if I become aware of it.

Name: Sandeep Kumar and Vijay Reddy
Date: 24/02/2019
Signature: SK, VJ

Report:

In this assignment, we have made use of mutex locks and condition variables to simulate a musical chairs games based on the concepts of multi threading. Depending on the input given, n player threads and 1 umpire thread are created for each instance of the game. Here, we try to study the effects of the players and umpires sleeping on the results of the game. In the next two sections, we discuss how this program was made.

Important Global Variables and their role:

Some of the major global variables used in this program are *ready*, *seat*, *sleepers* and *n_searching.* The *ready* array and the *sleepers* array are filled up with 1s and 0s respectively at the beginning of every lap of the game by the umpire. Each element of the *sleepers* array corresponds to how much the respective player must sleep in that lap once music starts. The *ready* array is primarily used to ensure that the same player doesn't try to occupy multiple seats in the same lap. Once the i^{th} player has finished searching for a chair in a lap, he sets his respective ready[i] to 0. Now the player will busy-wait till the umpire denotes the start of the next lap by setting ready[i] to 1 again.

The *seat* array is used to keep track of empty seats in each lap. Once the lap has begun and the umpire has let go of the *mtx* lock, it busy waits till the value of *n_searching* becomes 0. The value of *n_searching* indicates the number of players still searching if a chair is available and therefore when its value is 0 it means that the lap is essentially over.

Important locks and their role:

The main mutex lock used in this program is *mtx* and is initially held by the umpire and is taken by the players when they search for the chairs, once music stops.

Three condition variables have been used mainly for the purpose of busy waiting. The first one is the most trivial one and is used to stop the *umpire_main* from starting until all the player

threads have been spawned. The second condition variable is used by the *umpire_main* to busy wait on the value of *n_searching* and waits till the only remaining thread in the lap sets n_searching to 0. The third condition variable makes the player threads wait till the start of the next lap, once they are done searching for their seat in the current lap. It does so by busy-waiting on the value of ready[i] and waits till it is set to 1 by the umpire thread during the next lap.
The last lock, *mtx_sleep* is used to synchronize read and write accesses to the *sleepers* array by the players and umpires respectively.

Sample Outputs and Observations:

CASE 1:

Input:

lap_start
music_start
music_stop
lap_stop
lap_start
music_start
music_stop
lap_stop
lap_start
music_start
music_stop
Lap_stop

Command: ./hi --nplayers 4 < input.txt

Output:

Musical Chairs: 4 player game with 3 laps.
======= lap# 1 =======
0 could not get chair
****
======= lap# 2 =======
2 could not get chair
****
======= lap# 3 =======
1 could not get chair
****
Winner is 3
Time taken for the game: 1216 us

Observations: In a case with no sleeping, the winner is non-deterministic and the order of players getting knocked out is random.

CASE 2:

Input:

lap_start

```
music_start
umpire_sleep 400
music_stop
lap_stop
lap_start
music_start
umpire_sleep 20000
music_stop
lap_stop
lap_start
music_start
umpire_sleep 800000
music_stop
lap_stop
lap_start
music_start
umpire_sleep 5000
music_stop
lap_stop
lap_start
music_start
umpire_sleep 20000
music_stop
lap_stop
lap_start
music_start
umpire_sleep 8000
music_stop
lap_stop
```

Command: ./hi --nplayers 7 < input.txt
Output:

```
Musical Chairs: 7 player game with 6 laps.
======= lap# 1 =======
0 could not get chair
**********
======= lap# 2 =======
6 could not get chair
**********
======= lap# 3 =======
1 could not get chair
**********
```

======= lap# 4 =======
2 could not get chair
****
======= lap# 5 =======
4 could not get chair
****
======= lap# 6 =======
3 could not get chair
****
Winner is 5
Time taken for the game: 857701 us
Observation: Even if the umpire is sleeping during the laps, the results are still non-deterministic although the time taken for the game obviously increases.

CASE 3:
Input:
lap_start
player_sleep 0 1000
player_sleep 1 2000
player_sleep 2 3000
player_sleep 3 4000
player_sleep 4 5000
player_sleep 5 6000
player_sleep 6 7000
music_start
umpire_sleep 200
music_stop
lap_stop
lap_start
player_sleep 0 1000
player_sleep 1 2000
player_sleep 2 3000
player_sleep 3 4000
player_sleep 4 5000
player_sleep 5 6000
music_start
umpire_sleep 200000
music_stop
lap_stop
lap_start
player_sleep 0 1000
player_sleep 1 2000
player_sleep 2 3000

player_sleep 3 4000
player_sleep 4 5000
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

Command: ./hi --nplayers 7 < input.txt

Output:

Musical Chairs: 7 player game with 6 laps.
======= lap# 1 =======
6 could not get chair
****
======= lap# 2 =======
5 could not get chair
****
======= lap# 3 =======
4 could not get chair
****
======= lap# 4 =======

3 could not get chair
****
======= lap# 5 =======
2 could not get chair
****
======= lap# 6 =======
1 could not get chair
****
Winner is 0
Time taken for the game: 2015213 us
Observation: Here, the pattern of *player_sleep* used makes the result deterministic as the player that sleeps the most in each lap is always the last to check for an empty chair. By the time this player is done sleeping and starts searching for a seat, all the available seats would have already been taken. So every lap, the player that sleeps the most is knocked out.

CASE 5:
Input:
lap_start
player_sleep 0 10
player_sleep 1 20
player_sleep 2 30
music_start
music_stop
lap_stop
lap_start
player_sleep 0 10
player_sleep 1 20
music_start
music_stop
lap_stop
lap_start
player_sleep 0 10
music_start
music_stop
lap_stop
Command: ./hi --nplayers 4 < input.txt
Output:
Musical Chairs: 4 player game with 3 laps.
======= lap# 1 =======
1 could not get chair
****
======= lap# 2 =======
0 could not get chair

****
======= lap# 3 =======
2 could not get chair
****
Winner is 3
Time taken for the game: 1482 us

Observations:
Here, although one might that the results should be deterministic they aren't as the sleep values are too small for them to actually make a difference. Thus, as the sleeping times of the players decrease they become less and less predictable.

CASE 5:
Input:
lap_start
player_sleep 0 1000
player_sleep 1 2000
player_sleep 2 3000
player_sleep 3 4000
music_start
umpire_sleep 50000
music_stop
lap_stop
lap_start
player_sleep 0 1000
player_sleep 1 2000
player_sleep 2 3000
music_start
umpire_sleep 50000
music_stop
lap_stop
lap_start
player_sleep 0 1000
player_sleep 1 2000
music_start
umpire_sleep 50000
music_stop
lap_stop
Command: ./hi --nplayers 4 < input.txt
Output:
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: 152177 us

Observation: Here, the umpire_sleep is more than all the player_sleep values but still the output is deterministic. This is because, although the threads start searching for seats only after music_stop is encountered(by then all threads would've completed sleeping due to the excessive umpire sleep time), the compiler remembers the order in which the threads began to wait at the *mtx* lock and gives them thread access in that very order.

Conclusion:

We can clearly conclude that sleeping times of the players definitely do have an effect on the results of the game. The effect they have on the game reduces with the values of the sleeping times itself, as the non-determinacy of the scheduler begins to play a larger role. The umpire sleeping doesn't really impact the game in any manner.