HW Programming Assignment 2 - Musical Chairs

CS18BTECH11042 | CS18BTECH11047

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: Shreyas Jayant Havaldar, Vedant Singh
Date: 24-02-2020
Signature: S.J.H, V.S.

Note: Please refer to the program comments for details.

- Solution Implementation:
  - Approach:
    - C++ synchronization primitives to ensure synchronization between the umpire and the player threads:
      - **std::thread**: For creation of *nplayer* player threads and umpire thread.
      - **std::mutex**: To ensure proper printing of I/O and manipulation of variables to prevent race conditions among the players themselves and the umpire and player threads by ensuring the control of the critical section stays with only one thread. Also used for utilising condition variables.
      - **std::unique_lock**: Created to establish control over condition variables.

- **std::condition_variable**: Used to make a particular thread wait until another thread sends a signal using a predicate.
- **std::atomic_flag**: Used to make an array which corresponds to the total chairs in the game. Only one player can be in the process of acquiring the i^{th} chair at a particular point of time in the game.
- **std::atomic:** The arrays like *chair_array* and *sl,* and integers like *winner* and *running* were made atomic to ensure all operations on these variables happen in one step without context switch.

■ Acquiring chairs:

- We ensured maximum resemblance to real world musical chairs and ensured utmost fairness between all player threads while trying to acquire a chair.
- We have created an array of atomic_flag with each atomic flag corresponding to a chair. This is done to ensure that if two different players target the same chair, only one of them is able to acquire it with the rest of the players proceeding as usual.
- We ensure that if, without loss of generality, player 1 is trying to acquire chair 1 then independently player 3 can still try to acquire chair 3. Multiple players may try to acquire different chairs independently practically at the same time.

○ <u>Functions</u>:

■ main():

- The *main()* function deals with the allocation of memory to the global variables and also initializes them.
- Calls *musical_chairs()* function to allow for the simulation of the game, prints the value returned(time taken for game completion) and exits the program.
- Allocates appropriate memory to the various arrays used in the program.
- Time taken for the game to complete is printed if the program proceeds correctly and exits as success.
- Otherwise prints error and exits as failure. (Also if proper command line arguments are not provided.)

■ musical_chairs():

- The time is noted and stored in variable *t1*.

        - Umpire thread is created and executed the function *umpire_main()*.
        - An array of *n_players* player threads is created, with each thread calling the *player_main()* function passing the *plid* to the function for identifying the player number which is executing the function currently.
        - Joins all the created threads.
        - The time after the game is completed and the winner player no is printed is noted in *t2.*
        - Returns the difference *t2 - t1* to *main().*
    - umpire_main():
        - Reads input and makes the game proceed as required.
        - Initializes the *sl* and *chair_array* at the beginning of the lap.
        - Fills in *sl* according to the input.
        - Prints out the winner of the game after all laps are completed and returns.
    - player_main():
        - Waits for the umpire to mark all the player threads ready.
        - Waits until the music starts and then sleep if necessary otherwise try to acquire a chair according to the above described process.
        - Exits if it doesn't get a chair printing its player no.
        - Else waits for the next lap signal from the umpire.
        - Exits after the last lap is completed, if a player is not kicked out until then.
- Precautions taken:
    - Use of *cout.flush()* after every *cout* to clear the buffer.
    - Use of atomic variables to prevent context switching causing problems in manipulation of values.
    - Use of control mutex in required places to prevent synchronization issues and proper I/O.
    - Taken into consideration the case of only one player, where there is no lap and player 0 wins everytime.

Observations:

- No sleep for players and umpire:

        - After running the program for various players, it was observed that the winner is completely random, as should be.
        - No bias was observed towards the winning of any player even for a small total player number. (eg. We ran the two player game 20 times and observed that both player 1 and player 0 won it 10 times each.)
        - The time taken for the game to run appears to be linear in the number of players as can be seen from the sample output in readme.txt with the 4 player game taking 496 us and 14 player game taking 1506 us.
    - Sleeping umpire:
        - No marked difference in the observations from the above case, as players wait until the umpire signals that the music has stopped to try and acquire chairs.
        - The time taken for the game now includes the time the umpire slept and if we subtract it from the total game time, similar time as the above case is observed.
    - Sleeping players and sleepy umpire:
        - The umpire sleeps more than all the players:
            - The player who has slept the most among the still alive players is the last one in the queue to enter the code to acquire the chair.
        - At least one player sleeps more than the umpire:
            - The player who sleeps for the most duration among all the still alive players and is still sleeping when the umpire has signalled music stop will always fail to acquire the chair.
            - This will be a deterministic case as observed in the sample input mentioned in readme.txt for 4 players, where for every execution of the program, player 3 was kicked first, followed by player 2, then 1 with 0 being the winner every time.
            - This occurs because the other players have acquired all the chairs by the time this player awakens and then tries to look for a free chair.
            - The time taken for execution excluding the maximum sleeping time is identical to the no sleeping case.
            - All the threads sleep concurrently, thus the sleeping time does not add up.

Thus, the game of musical chairs was simulated successfully.