ASSIGNMENT 5 DESIGN DOC

GROUP NO : 08

GROUP MEMBERS:

1.) UMANG SINGLA : 20CS10068

2.) GRACE SHARMA : 20CS30022

3.) MRADUL AGRAWAL : 20CS30034

4.) SOURABH SOUMYAKANTA DAS : 20CS10069

Data Structures used:

- class Guest:
  - priority (int) : Stores the priority alloted to the guest
  - id (int) : Stores the unique ID of the guest
- class Room:
  - guest (Guest *) : Stores the pointer to the Guest data structure of the guest currently staying in the room
  - previous_guest_time (int) : Stores the amount of time the room has been occupied by the previous guests
  - number_of_guests (int) : Stores the count of guests alloted to the room after last cleaning
  - cleaned (bool) : Stores the boolean value of whether the room is clean
- x, y, n (int) : These store the counts of cleaning staff, guests and rooms respectively
- rooms (Room *) : Stores pointer to the array of class Room of size n
- sem_id (sem_t *) : Stores pointer to the array of sem_t of size y storing the semaphore IDs
- sem_cleaner (sem_t) : Stores the semaphore for cleaning staff to modify the relevant data structures to keep them updated
- sem_clean_wait (sem_t) : Stores the semaphore for cleaning staff to start the cleaning process in all rooms
- mutex_id (pthread_mutex_t) : Mutex for locking while changing room and guest details
- cond_id (pthread_cond_t) : Conditional variable for checking the availability of rooms suitable for the guest
- guest_tid (pthread_t *) : Stores pointer to the array of pthread_t of size y storing the guest thread IDs
- staff_tid (pthread_t *) : Stores pointer to the array of pthread_t of size x storing the cleaning staff thread IDs
- guests (vector<Guest>) : Stores the details of guests in a vector of class Guest of size y

- room_priority_queue (multiset<pair<int, int>>) : Stores the guest queue for rooms based on the priority of the guest currently staying in the room and the room no.
- uncleaned_rooms (int) : Stores the count of uncleaned rooms

Design for the main thread:

- The main thread initialises the mutexes, conditional variables, semaphores, and relevant data structures.
- Then it creates the guest and cleaning staff threads.
- After the threads exit, it destroys the mutexes, conditional variables, and semaphores, and frees the relevant data structures.

Design for the guest thread:

- Initially, we obtain the guest data from the ID passed on to the thread through params.
- The guest sleeps for a random time in (10, 20) s.
- It acquires a mutex lock on mutex_id.
- If all the rooms have a higher priority guest or are unclean, then the current guest waits on cond_id, else if the room is empty, the guest is alloted the room, else an interrupt is sent to the sleeping guest to wake up.
- Modifications are done in the room_priority_queue to reflect the operations done.
- It unlocks the mutex_lock on mutex_id and broadcasts on cond_id.
- The guest sleeps for a random time in (10, 30) s on a timed wait of semaphore which can be interrupted.
- Then we obtain the mutex lock mutex_id again.
- If we observe that the guest's sleep was not interrupted, then if the room had less than 2 guests after cleaning, we remove the (guest priority, room) pair from the priority queue and add the room as empty in the priority queue else we mark the room not clean and if the unclean room count reaches n, we post on sem_clean_wait.
- It updates the room data based on the stay of the guest, releases the mutex lock and broadcasts on cond_id.

Design for the cleaning staff thread:

- It waits on the sem_clean_wait until all the rooms need cleaning.
- Then within the sem_cleaner semaphore lock, it selects at random an unclean room to clean and marks it clean.
- After releasing the lock, it cleans the room (sleeps) for the required time and then reverts the room back to its initial state.
- It acquires the sem_cleaner semaphore lock again and dequeues the guests from room_priority_queue. (change)
- It then broadcasts on cond_id and releases the semaphore lock.