1. Data Structures Used:

i. Room

Each room is represented as a strcture with the following fields:

a. occupied (int): 0 represents empty room, 1 represents a room occupied with a guest

b. curr_guest (int): stores the id of the guest currently occupying the room (0 when there is no guest)

c. curr_stay_time (int): time the current guest stays in the room

d. curr_stay_start (time_t): start time of current guest occupying the room

e. total_occupants (int): number of occupants in the room since the last time the room was cleaned

f. occupied_time (int): total time the room has been occupied by guests since the last time the room was cleaned

2. Global Variables Used:

i. n, x, y (int): stores the number of rooms, number of cleaning stuff and number of guests respectively

ii. rooms (room*): an array of n room structures, each corresponding to a room in the hotel

iii. guest_priority (int*): an array of y integers, storing the priorities for each guest

iv. guest_removed_status (int*): an array of y integers, storing for each guest the time left to be spent in the room when it has been removed (0 if no guest present)

v. total_guests_curr (int): number of hotel rooms currently occupied by guests

vi. rooms_with_2_occupants (int): number of hotel rooms that have 2 occupants since the last time the room was cleaned

vii. allotment_possible (int): stores whether a room can be alloted to a guest requesting for it

3. Semaphores Used:

i. room_access (sem_t*): an array of n semaphores corresponding to the n rooms, each initialized to 1, which control whether a room can be accessed by a cleaning or guest thread

ii. guest_removed_status_access (sem_t*): an array of y semaphores corresponding to the y guests, each initialized to 1, which control access to the guest_removed_status array

iii. total_guests_curr_access (sem_t): controls access to the total_guests_curr variable, initialized to 1

iv. rooms_with_2_occupants_access (sem_t): controls access to rooms_with_2_occupants variable, initialized to 1

v. cleaning_mode (sem_t): helps to toggle between guest occupancy mode and cleaning mode, initialized to 0

vi. allotment_possible_access (sem_t): controls access to the allotment_possible variable

vii. print_access (sem_t): controls access to printing details about the guest and the cleaning thread

4. Thread Design:

i. Guest Thread (argument: void* &guest_id):

a. The guest sleeps for a random time between 10 and 20 seconds and wakes up.

b. If the guest was kicked out, stay_time is set to the remanining time, otherwise a random time between 10 and 30 seconds is generated. This is checked using guest_removed_status and its semaphore.

c. The thread prints the status of the guest using the print semaphore.

d. If room allotment (using allotment_possible_access semaphore) is not possible, the amount of time to spend when room is alloted is stored in guest_removed_status.

e. If there are less than n guests occupied (checked using total_guests_curr and its semaphore), then it finds an unoccupied room and stores the data in rooms, printing the status, using corresponding semaphores.

f. If the hotel is full, the guest with the least priority is found and kicked out, updating data in the rooms and guest_removed_status variables and using the semaphores.

g. If a room has been alloted, then the guest sleeps for the stay time.

h. If the room has been occupied twice, the count of 2-rooms_with_2_occupants is increased, and cleaning is invoked when all the rooms reach this mode.

ii. Cleaning Thread (argument: void* &cleaner_id):

a. The cleaning thread iterates through the rooms to find those occupied twice.

b. The thread sleeps for the total time the room has been occupied (as it should be proportional to the total time occupied).

c. After waking up, it resets the counter to 0, making the room available to be occupied.

d. It prints the status before and after the completion of the cleaning process.

iii. Main thread:

a. It takes n, x, y as inputs from the user.

b. It generates random priorities for the guests and then sorts guests based on their priorities, reassigning them priorities equal to their index

c. It then initializes all the global variables and semaphores with the default values.

d. It creates pthreads for all the y guests and x cleaning threads and launches them.

e. It then waits for the threads to join, and then terminates.