OPERATING SYSTEM LAB

ASSIGNMENT 5

GROUP: 5

Design and Data Structure Report

MEMBERS:

Abhibhu Prakash 20CS10002
Anirban Haldar 20CS10008
Shah Dhruv Rajendrabhai 20CS10088
Suhas A M 20CS10066
Vivek Jaiswal 20CS10077

Data structure Details:

```
typedef struct g
{
    int id;
    int priority;
    int evict;
    int roomid;
}guest_ds;
```

id: stores the guest id
priority: stores the priority of the guest
evict: flag to denote whether guest is thrown out of the room or not
roomid: stores the room id where this particular guest is staying

```
typedef struct r
{
    int id;
    int tot_time;
    int state;
    guest_ds Guest;
}room;
```

id: stores the id of the room
tot_time: denotes the total time elapsed since it was cleaned
state: stores the state of the room. We are using the following flag values for the states of the room:

- ***0*** for no one has used this room after it is cleaned
- ***1*** for someone is occupying the room since it was last cleaned
- ***2*** for one guest has used it since last cleaned, and the room is currently unoccupied
- ***3*** for 2nd guest is staying here
- ***4*** 2nd guest left and no one is occupying the room right now

`extern int N,X,Y;`
These denote the number of rooms, cleaners and guests respectively (Y > N > X>1).

`extern unordered_set <int> rooms_set;`
This stores the set of indices of the rooms yet to be cleaned.

`extern unordered_set <int> Rset;`
This contains the indices of the rooms that are currently free. Rooms with states 0 or 2.

`extern int guest_count;`
This is used to keep track of the number of guests.

`extern int K;`
Proportionality constant for cleaning time (later initialized to 1 in main.cpp).

`extern set <pair<int, int>> pq;`
Priority queue to find the index of the room currently occupied by the guest with the lowest priority. Rooms with state 1.

Semaphore Usage:

`extern sem_t sem_tot;`
A counting semaphore to protect the occupancy. As there are N rooms and each can be occupied twice, we initialize this with 2N.

`extern sem_t sem_clean;`
A counting semaphore that is initially unavailable. When a guest function makes the sem_post() call Y times, the cleaner thread runs for a total of Y times.

`extern sem_t* per_room;`
An array of semaphores to protect the data read/ write for every individual room.

`extern sem_t* per_guest;`
An array of semaphores to protect the data read/ write for every individual guest.

`extern sem_t rset_sem;`
A semaphore that protects the rooms_set set, which records the rooms that need to be cleaned.

`extern sem_t print_sem;`
A semaphore to protect the printf/ cout operation, so that only a single thread can perform the said I/O operation at a time.

`extern sem_t sem_Rset;`
A semaphore that protects the Rset, which records the rooms that are currently free.

`extern sem_t sem_pq;`
A semaphore that protects the priority queue ‘pq’, that records the room index and the priority of the guest residing in that room.

Overall Design:

We have defined our data structures, declared the semaphores, the guest and cleaner function signatures and some variables inside a header file named “ds.h”.

Main Process:
Here we first initialize some global variables. Inside the main process, we make the guest and cleaner threads, and the per_room and per_guest arrays (to be used later). Then we initialize the semaphores and the rooms. Next, we create the guest threads and assign a random priority to each of them. After this, we create the cleaner threads and pass id of the cleaner in each case as an argument. Finally we just wait for the threads to join.

Cleaner Thread:

```
while (rooms to be cleaned exists)
     Select a room
     remove it from list of rooms to be cleaned
     Cleaning time initialized with total time the guests stayed

     if (room.state is 3)
          guest is evicted and corresponding flag is set
          display the evicted guest’s id

     reinitialize the room’s state and tot_time to 0
     cleaner cleans (goes to sleep)

if(cleaner ID is 0)
     cleaner frees the rooms
     adds the room to list of rooms that can be occupied
```

Guest Thread:
When there is enough space in the queue, or if there is an empty room ready to be occupied:
Randomly select a time between 10 to 20 seconds for sleep timer.

```
If(empty rooms available)
    Allot an empty room
    If(no room which is only occupied once)
        Call cleaners
    Else if(current guest is the first occupant)
        Insert guest id and room id in pq
    Sleep for sleep timer
    If(guest was evicted)
        Do Nothing
    Else
        Guest frees room
        Increments the state of room
        If(room.state == 2)
            Add room in Rset
Else
    If(No occupied rooms with their first occupant)
        Call cleaners
    Else
        if(guest with lower priority exists)
            Evict the guest with lowest priority in pq
            if(pq is empty after eviction)
                Call Cleaners
                Sleep for sleep timer
            Sleep for sleep timer
            If(guest was evicted)
                Do Nothing
            Else
                Guest frees room
                Increments state of room
        Else
            Waits for other guests to vacate
```