Design Doc

Group-6, Assignment-5

Chirag Ghosh, 20CS10020

Sarita Singh, 20CS10053

Srishty Gandhi, 20CS30052

Yatindra Indoria, 20CS30060

Data Structures

We are using the following struct for representing a Room:

```
typedef struct _room {
      bool available;
      int pastOccupants;
      int currentOccupant;
      time_t totalTimeLived;

      bool operator<(const _room& rhs) const {
      return guestPriorities[currentOccupant] >
guestPriorities[rhs.currentOccupant];
      }
} Room;
```

We use an array of type Room to store information about all the rooms in the hotel.
We maintain a stack of availableRooms and unavailableRooms.
In availableRooms stack we push the rooms which have zero occupants.
In unavailableRooms we push the rooms which have two occupants.

We maintain a set of occupiedRooms. In this set, we insert the rooms which have one occupant. Through this, we keep track of the rooms from which a guest of lower priority can be evicted when a guest with higher priority cannot be allotted a room through the available room stack.

We maintain the global variable totalOccupiedSinceLastClean to keep track of the total occupants since the rooms were last cleaned in the hotel.

We use the array guestPriorities to store the priorities of the guests in order to aid in evicting a lower priority guest if required.

We use the roomSemaphore for mainitnaing the resource which in our case is the number of available rooms.

We have also defined a mutex_lock for each of the global data structures which may cause a race condition.

Design Implementation

In our design, an available room(room with zero occupants) is a resource. We use a semaphore for this resource.
If the semaphore function sem_trywait()

> returns 0, then it implies that we have an available resource(room) for the requesting guest. In this case, we lock the availableRooms stack and then pop a room from it. We update the details of this room like availability, currentOccupant, pastOccupants. If the pastOccupants of this room becpmes 1 then we insert the room to occupiedRooms set. We then allot this room to the guest.
>
> returns with errno == EAGAIN, then it implies that there is no room with zero occupants in the hotel. In this case, we have to evict a guest from a room with one occupant. Hence we call a function to evict a lower priority guest from a room in occupiedRooms set.
>
> returns because of the semaphore wait error, then we print the error and exit.

Each time we allot a room to the guest we also update the totalOccupiedSinceLastClean. When a room's pastOccupants value becomes 2 we push the room to the unavailableRooms stack.

The function cleaner_start_handler is used to handle the signal is_cleaning.
The cleaningStaff thread is woken up by the signal is_cleaning

> The thread then gets the semaphore value of roomSemaphore
> The thread then locks the unavailableRooms stack and pops an unavailable room to clean it.
> If the popped out currentRoom is a valid room number then:
>
> > After performing the cleaning action, the thread updates the details of room like availability, currentOccupant, pastOccupants and totalTimeLived of the room accordingly.
> > It then locks the availableRooms stack and pushes the newly cleaned room into it.
> > The thread then performs sem_post() to increment the semaphore value of roomSemaphore.
>
> Else if the currentRoom is -1 then we check the set occupiedRooms. If the set occupiedRooms is empty then we update the global variable totalOccupiedSinceLastClean to zero and update is_cleaning to zero. Then we call pthread_kill to send the signal to all the guest threads.