OPERATING SYSTEMS LABORATORY

CS39002

Assignment 5:

Usage of Semaphores to synchronize between threads

Designing Documentation

Group No. 21

Nikhil Sarashwat (20CS10039)

Abhijeet Singh (20CS30001)

Amit Kumar (20CS30003)

Gopal (20CS30021)

➢ Explanation of Threads

➔ Main Thread

- The main thread first initializes all the semaphores present.
- All the rooms are inserted in the available_0 vector which stores the information about all the rooms which have zero occupancy till now.
- Then it used to initialize the ids of the rooms,guests and cleaning staff.
- It then creates the guests threads and cleaning staff threads.
- The main thread then waits on guests and cleaning staff threads to complete their execution.
- At last it destroys all the semaphores and free all the data structures used.

➔ Guest Thread

- The guest thread sleeps for a random amount of time and then wakes up and then waits on the semaphore sem_cleaning to check whether the cleaning is under process or not and then signals the semaphore sem_cleaning. If it is not under cleaning it proceeds further.
- sem_getvalue(&sem_guests[guestId], &temp) is done in order to ensure that a signal from previous attempt is not pending and it only moves forward when value of semaphores becomes zero. Indirectly we are resetting the value of semaphore to zero to avoid the effect of delayed signal.
- It then generates the random amount of time for which it would stay in the room and request for a room.
- If it is unavailable to find a room then it goes for the next iteration.
- When it acquires the room it waits for the sem_terminal semaphore and prints the appropriate message on the terminal.
- After that it waits for the semaphore sem_room_to_clean and checks if all the rooms are dirty and if all the rooms are dirty then firstly all the guests are kicked out of the room by sending a signal to semaphore sem_guests of respective guests. After that signals are sent to all the semaphores named sem_cleaningStaff to wake up all the cleaning threads.
- If all the rooms are not dirty it goes on the timed_wait using sem_timedwait for the time it has to stay in the room.
- If the return value of the timed_wait is 0 it means that some other guest of higher priority has kicked him out or cleaning has started.
- Otherwise if the errno is ETIMEDOUT or return value is -1 then this means that guest has completed his stay and will leave the room.
- After this information of the room is updated after waiting for the semaphore of that particular room also if guest_count is one then the information of the guest is also erase from the set occupied and that room is inserted in the vector available_1 which stores the information of all the rooms which have 1 guests uptill now.
- What happens when a guests makes request for a room:
  On making requests waiting is done on the semaphore sem_room_ptr since we may access any information about any room.
  - ➔ Firstly we check in the available_0 vector and if it has at least one element we allot that room in the available_0 vector to the guest and then remove that room from the available_0 vector and update the information about the room in the struct. We also add the information about the guest and room in the occupied set to later use this information for allotting the same room to a guest of higher priority. And then return the room id to the guest.
  - ➔ Similarly we check in the available_1 vector and if there is at least one room in this vector we allot that room and also update information about the room and also mark that room as dirty. Also update the count of the dirty rooms and return the room id to the guest.
  - ➔ And if there is no available room we check the guest ids of the guest occupying room which have at max 1 occupancy with the help of occupied set and if the lowest priority value among all such guests is lower than the current guest we kick out that guest by sending a signal. And allot that room to the current guest and update information about the room and mark that room dirty and send back the room id.

➔ Cleaning Thread

- It first waits for the semaphore of the respective cleaning thread.
- And when it acquires it, it then checks whether cleaning is under process or not by acquiring another semaphore. If it is not it comes out by sending a signal .
- Otherwise it cleans until all the rooms have been cleaned.
- It first waits for the semaphore of the room_to_clean and if size of room_to_clean is not zero it proceeds.

- Room_to_clean contains indexes of all the rooms which are remaining to be cleaned.
- It selects one of the indexes of the room from this vector randomly and removes that index from the vector room_to_clean.
- If cleaning of that room hasn't yet been done it is cleaned by the cleaning thread by taking a time proportional to the time guests have stayed in that room and all the information of that room is set to default.
- It also checks if all the rooms have been cleaned then all the data structures are set to their default values.

➢ Data Structure Used

```
struct Guest
{
    int guestID;
    int priority;
    Guest(int guestID, int priority)
    {
        this->guestID = guestID;
        this->priority = priority;
    }
};
```

For implementation of Guest we define above structure:
guestID: an integer that represents the ID of the guest
priority: an integer (randomly generated between 0 and Y-1) that represents the priority level of the guest
We have created an array of Guests named as guests for storing information about all the y guests.
We have an array of semaphores named as sem_guests one corresponding to each of the guests to avoid race conditions on them.

```
struct Room
{
    int roomID;      // room number
    bool isOccupied; // is occupied or not
    int GuestId;     // guest id if occupied
    int GuestCount;  // number of guest in room till now
    int SpentTime;   // time spent in room
    int room_status; // 0: dirty, 1: cleaning, 2: clean
    Room(int roomID)
    {
        this->roomID = roomID;
        GuestCount = 0;
        isOccupied = false;
        SpentTime = 0;
        GuestId = -1;
        room_status = CLEANING_DONE;
    }
};
```

For implementation of Room we define above structure:
roomID: an integer that represents the room number
isOccupied: a boolean that indicates whether the room is currently occupied or not
GuestId: an integer that represents the ID of the guest who is currently occupying the room
GuestCount: an integer that represents the total number of guests who have occupied the room so far
SpentTime: an integer that represents the total amount of time to which the cleaning will be proportional to
room_status: an integer that represents the current cleaning status of the room, which can be either "DIRTY", "CLEANING", or "CLEANING_DONE".

We have an array of Rooms storing information about the n rooms.
We also have an array of semaphores named sem_rooms one corresponding to each of the rooms.

We also have a semaphore named sem_room_ptr which is used when we are accessing all the rooms at once.

cleaningStaffNo

We have an array named cleaningStaffNo storing the ids of the cleaning staff.
We also have an array of semaphores named sem_cleaningStaff one corresponding to each cleaning.