OS Lab Assignment-5
Design Doc

Group No: 2

Ritwik Ranjan Mallik (20CS10049)
Saptarshi De Chaudhury (20CS10080)
Nikumbh Sarthak Sham (20CS30035)
Ayush Kumar Dwivedi (20CS10084)

Data Structures:

struct room: stores information about a room which includes room number, availability status, number of occupancies after last cleaning, guest id and time for which the room was occupied.

vector<room *> roomlist: stores pointers to struct room.

vector<int> priority: stores the random priority allocated to each guest.

We have included a header file ‘structs.h’ containing the definition of the room structure.

Semaphores and Mutexes used:

roomAvailabilitySemaphore: This semaphore controls the allocation of rooms such that there cannot be more than 2N occupancies. If the number of occupancies is 2N then the guest threads will keep waiting and not allocate rooms. It is initialised with value 2N.

roomAllocationSemaphore: This is a binary semaphore which controls the allocation of rooms to guests, such that no two threads can access the list of rooms at the same time. It prevents the race condition of two threads trying to allocate the same room to two different guests.

roomCleanSemaphore: This is a binary semaphore which controls the allotment of rooms to cleaners, such that no two threads can access the list of rooms at the same time. It prevents the race condition of two threads trying to allot the same room to two different cleaners.

printSemaphore: This is a binary semaphore which controls the printing of output to the terminal.

cleanerMutex: This is a vector of mutexes, one for each cleaner thread. It prevents the execution of the cleaner threads until the number of occupancies is 2N.

Threads:

Main thread:-

It initially takes three inputs N,X and Y then initialises all the required mutex and semaphores. After that it allocates random priority to all the guests and creates the required number of rooms. Then it creates the required number of guest and cleaner threads and waits till they complete their execution. Finally it destroys all the mutexes and semaphores, also it frees the memory allocated to the rooms.

Cleaner thread:-

Initially the thread waits on the cleanerCondition using the cleanerMutex until all the rooms have been occupied by 2 guests. Then it chooses one unclean room and sleeps for the total stay time of guests. After that it decrements the dirty_rooms counter by 1. If all rooms get cleaned then one of the cleaner threads marks all the rooms as available by signalling the roomAvailabilitySemaphore and then keeps waiting on the cleanerCondition else it chooses another room and starts the cleaning process again.

Guest Thread:-

The guest number is passed as an argument to the thread. Then it waits on the roomAvailabilitySemaphore and it sleeps for a random time between 10 and 20 seconds. After that it waits on the roomAllocationSemaphore to choose a room. When it gains access to the room list, then it finds a suitable room to allocate to the guest through the allotroom( ) function.

If the room is not allotted then the roomAvailabilitySemaphore is signalled. Otherwise the room is allotted and the guest thread checks whether the number of occupancies is 2N. If so, then all the sleepSemaphores are signalled to evict all the occupied rooms. Then it broadcasts the cleanerCondition to activate the cleaner threads. If not,

then the guest thread occupies the room and sleeps until the sleep is completed or some other thread evicts it.

allotroom( ) function:

If the current number of guests is less than N then the first room which is unoccupied is allotted to the guest.
Otherwise, we find the room which contains the guest with the highest priority that is less than the current guest, to evict.
This will allow the maximum number of room allotments to be made successfully.

For example, suppose there are two rooms and two guests with priority 1 and 4 currently occupying them, and a guest with priority 5 and another guest with priority 2 arrive in that order.
Firstly guest 5 will try to evict one of the guests, after that guest 2 will try to evict one of them. If guest 5 evicts guest 1, then the rooms will be occupied by guests 4 and 5. Now guest 2 will not be able to evict either of them and it will not be allotted a room. However if guest 5 evicts guest 4, then guest 2 will be able to evict guest 1 and it will also be allotted a room. Hence this strategy maximises the number of room allotments made.