Operating Systems

Lab report

Assignment 5

Group: 11

Team Members:

Yatharth Sameer(20CS30059)

Hardik Praveen Soni(20CS30023)

Subhajyoti Haldar(20CS10064)

Archit Mangrulkar(20CS10086)

Title: Hotel Room Allocation and Cleaning Simulation

Overview:

This program simulates a hotel where guests request rooms and cleaning staff clean the rooms. The simulation is implemented using three threads: one main thread and two worker threads. The main thread initializes the data structures, creates the worker threads, and waits for their completion. The worker threads are responsible for room allocation and cleaning.

Worker Thread 1: Room Allocation

The first worker thread is responsible for room allocation. It runs the guest_thread function, which waits for a room allocation using a condition variable and a mutex. The function uses a vector of Room structures to represent the hotel like the following information: priority of guest who stayed, number of occupants, previous guest ID, time of last cleaning and the time the last guest left the room. The guest_thread function uses the following data structures and synchronization primitives:

Data Structures:

- std::vector<Room> hotel: A vector of Room structures to represent the hotel.
- std::vector<int> guests_priority: A vector of integers to represent the priority of each guest.

Synchronization Primitives:

- std::mutex mtx: A mutex to protect shared data structures.
- std::condition_variable cv: A condition variable to wait for a room allocation.
- std::vector<sem_t> cleaning_semaphores: A vector of semaphores to signal cleaning staff when cleaning is needed.
- std::vector<bool> cleaning_in_progress: A vector of flags to indicate if a room is being cleaned.

Worker Thread 2: Room Cleaning

The second worker thread is responsible for room cleaning. It runs the cleaning_thread function, which waits for cleaning semaphores using sem_wait. The function uses the same vector of Room structures as the guest_thread function to represent the hotel. The cleaning_thread function uses the following data structures and synchronization primitives:

Data Structures:

- std::vector<Room> hotel: A vector of Room structures to represent the hotel.

Synchronization Primitives:

- std::mutex mtx: A mutex to protect shared data structures.
- std::condition_variable cv: A condition variable to signal guests when cleaning is complete.
- std::vector<sem_t> cleaning_semaphores: A vector of semaphores to wait for cleaning requests.
- std::vector<bool> cleaning_in_progress: A vector of flags to indicate if a room is being cleaned.

Main Thread:

The main thread initializes the data structures, creates the worker threads, calls them and waits for their completion. The main thread uses the following data structures and synchronization primitives:

Data Structures:

- std::vector<Room> hotel: A vector of Room structures to represent the hotel.
- std::vector<int> guests_priority: A vector of integers to represent the priority of each guest.
- std::vector<vector<int>> cleaner_pre: A vector of vectors to assign rooms to cleaning staff.

Synchronization Primitives:

- std::vector<sem_t> cleaning_semaphores: A vector of semaphores to signal cleaning staff when cleaning is needed.
- std::mutex mtx: A mutex to protect shared data structures.
- std::condition_variable cv: A condition variable to wait for room allocation requests.

Semaphores used:

In this program, semaphores are used to signal cleaning staff when cleaning is needed. Specifically, a vector of semaphores is used to notify each cleaning staff that cleaning is needed for a specific room.

When a guest releases a room and cleaning is needed, the guest signals the cleaning staff by calling sem_post on the corresponding semaphore. The cleaning staff thread waits for the semaphores using sem_wait and then cleans the assigned rooms. Once cleaning is complete, the cleaning staff signals the guests by calling notify_all on the condition variable.

The semaphores are initialized in the main function using the sem_init function and destroyed using sem_destroy at the end of the program. Each semaphore is initialized with an initial value of 0, which means that a cleaning staff thread will wait for a signal from a guest thread before cleaning the assigned room.

Overall, the use of semaphores in this program allows for efficient and synchronized communication between the guest and cleaning staff threads, ensuring that rooms are cleaned only when necessary and that guests are notified when cleaning is complete.

Conclusion:

In summary, this program simulates a hotel using three threads and various data structures and synchronization primitives. The guest_thread function is responsible for room allocation, the cleaning_thread function is responsible for room cleaning, and the main thread initializes the data structures and manages the worker threads. Overall, this program provides a simple example of how to use threads and synchronization primitives to simulate a real-world scenario.