VINAYAK JOSHI Reg No:210905270 Roll No:43

SAKSHAM SHARMA Reg No:210905248 Roll No:39

MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

Mini Project Report
of
Operating Systems Lab(CSE 3163)

HOTEL MANAGEMENT SYSTEM

SUBMITTED BY

DHRUV BAJAJ (210905202)

SAMARTH PARASHAR (210905206)

ADITYA SINGHVI (210905216)

LAKSHAY SAXENA (210905384)

Department of Computer Science and EngineeringManipal Institute of Technology, Manipal.

November 2023

MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Manipal
14/11/2023

CERTIFICATE

This is to certify that the project titled **HOTEL MANAGEMENT SYSTEM** is a record of the bonafide work done by **Aditya Singhvi, Dhruv Bajaj, Lakshay Saxena and Samarth Parashar** submitted in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology (B.Tech.) in COMPUTER SCIENCE & ENGINEERING of Manipal Institute of Technology, Manipal, Karnataka, (A Constituent Institute of Manipal Academy of Higher Education), during the academic year 2022-2023.

Name and Signature of Examiners:

1. **Mrs.Sucharitha Shetty - Assistant Professor, CSE Dept.**

TABLE OF CONTENTS

ABSTRACT

In the realm of hospitality, a hotel serves as a dynamic hub where guests, services, and accommodations intersect. The Hotel Management System project delves into the intricacies of efficiently managing rooms and guest interactions within this multifaceted environment. Much like orchestrating diverse traffic in a junction, where vehicles navigate shared space, hotel rooms contend for occupancy by guests with varying preferences and needs.

The Hotel Management System addresses the growing challenges in the hospitality industry, focusing on optimal room allocation, streamlined reservation processes, andthe seamless coordination of hotel facilities. Similar to addressing real-time dynamicsand queues in traffic management, this project tackles the dynamic nature of guest interactions, diverse room types, and the need for effective communication between guests and hotel staff.

Given the unpredictable nature of guest demands and the multifaceted factors influencing room availability, the project aims to unravel the complexities inherent in hotel management. Through careful identification and strategic addressing of these intricate elements, the proposed solution seeks to provide a practical, efficient, and adaptable system. The Hotel Management System recognizes the necessity for nuanced solutions that navigate the ever-changing landscape of guest requirements, ensuring a seamless and satisfying experience for both patrons and hotel staff.

INTRODUCTION

This project revolves around the creation of a Hotel Management System, incorporatingadvanced synchronization techniques like semaphores and shared memory to refine operations within the hospitality sector. In a manner reminiscent of how semaphores avert deadlocks in various scenarios, these mechanisms are harnessed here to improve efficiency, manage conflicts, and optimize the allocation of hotel resources.

Central to this project is the application of shared memory, establishing a collaborative space where essential hotel data is seamlessly shared among different components. Similar to the principles behind avoiding deadlocks in system processes, the Hotel Management System meticulously handles room allocation, reservation queues, and real-time guest interactions.

The incorporation of semaphores into the system mirrors the way control mechanisms are employed in various applications, serving as sentinels to regulate access to shared resources. These semaphores are critical in preventing conflicts, ensuring exclusive access to crucial sections, and streamlining the flow of information and services within the hotel environment.

Much like the objective of avoiding deadlocks and optimizing system flow in other contexts, the Hotel Management System aims to demonstrate how these synchronization mechanisms contribute to a seamless and efficient guest experience.Through careful design, strategic implementation, and collaborative shared memory,this project seeks to enhance the management of hotel resources, providing a robust solution that improves overall operational efficiency.

PROBLEM STATEMENT

To develop a comprehensive hotel management system that incorporates various operating system concepts to ensure efficient and reliable hotel operations management.

OBJECTIVES

The primary objectives of the Traffic Junction project are:

1) Synchronization and Deadlock Understanding:

- Showcase a solid grasp of synchronization and deadlock concepts within Operating Systems.

2) Effective Semaphore Implementation:

- Implement semaphores for synchronization, ensuring mutual exclusion and preventing deadlock situations.

3) Shared Memory Integration:

- Demonstrate proficiency in integrating shared memory mechanisms within the project, facilitating efficient communication and data exchange between different components.

METHODOLOGY

1) For Semaphores:

Semaphores are integral components in the hotel management system, specifically employed to synchronize access to shared resources related to room bookings. The room_semaphore is initialized to enforce mutual exclusion, ensuring that concurrent users interact with shared data in a coordinated manner. These semaphores play a pivotal role in preventing data inconsistencies and conflicts during critical sections such as room booking, releasing, and request queue manipulation. By utilizing sem_wait and sem_post, the system signals exclusive access to shared data for a user, facilitating the seamless execution of operations within the hotel management system. This strategic use of semaphores ensures that only one user at a time can modify or query shared data, maintaining the integrity of the system's operations in a multi-user environment.

2) Shared Memory:

In the hotel management system, shared memory serves as a critical mechanism for seamless communication and data sharing among various components of the program. The shared memory segment, embodied by the shared_rooms structure, is initiated and mapped into the process's address space. This shared memory facility allows dynamic information exchange and real-time updates across different functions and users within the system.

During the initialization phase, a shared memory segment is established using shm_open, and its size is defined by SHARED_MEMORY_SIZE. This segment is then mapped into the process's address space through mmap. The initializeRooms function subsequently populates this shared memory segment with initial room details, encompassing room numbers, occupancy status, types, timestamps, and user information.

The booking and releasing of rooms, executed through the bookRoom and releaseRoom functions, involve modifications to the shared memory segment. These modifications reflect changes in room occupancy, user details, and timestamps. To ensure exclusive access and prevent conflicts among concurrent users, these operations are protected by semaphores.

3) For Deadlocks:

In the hotel management system, deadlocks are preemptively addressed through the meticulous use of semaphores, notably the room_semaphore. This semaphoreensures exclusive access during critical operations like room booking and releasing, minimizing the risk of concurrent conflicts. The strategic implementation of sem_wait and sem_post operations sequences access to sharedresources, preventing circular wait conditions that can lead to deadlocks.

The prevention strategy extends to resource allocation, where semaphores are acquired and released in an orderly fashion, eliminating the potential for cyclic dependencies.

RESULTS AND SNAPSHOTS

Code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <fcntl.h>
#include <unistd.h>
#include <time.h>
#include <string.h>

#define SINGLE 1
#define DOUBLE 2
#define SUITE 3

typedef struct {
    int room_number;
    int room_type;
    int is_occupied;
    time_t last_occupied_time;  // New field for timestamp
    char user_name[50];  // New field for user name
} Room;

// Define the size of shared memory
#define SHARED_MEMORY_SIZE (sizeof(Room) * 10)

// Shared memory pointer
Room *shared_rooms;

sem_t *room_semaphore;

```

```c
// Queue structure for booking requests
typedef struct {
    int room_type;
    char user_name[50];
    struct RequestQueueNode* next;
} RequestQueueNode;

typedef struct {
    RequestQueueNode* front;
    RequestQueueNode* rear;
} RequestQueue;

RequestQueue request_queues[3];  // One queue for each room type

void initializeRequestQueues() {
    for (int i = 0; i < 3; i++) {
        request_queues[i].front = NULL;
        request_queues[i].rear = NULL;
    }
}

void enqueueRequest(int room_type, const char *user_name) {
    RequestQueueNode* new_node = (RequestQueueNode*)malloc(sizeof(RequestQueueNode));
    if (new_node == NULL) {
        fprintf(stderr, "Failed to allocate memory for request queue node.\n");
        exit(EXIT_FAILURE);
    }

    new_node->room_type = room_type;
    snprintf(new_node->user_name, sizeof(new_node->user_name), "%s", user_name);
    new_node->next = NULL;
```

```c
    if (request_queues[room_type - 1].rear == NULL) {
        request_queues[room_type - 1].front = new_node;
        request_queues[room_type - 1].rear = new_node;
    } else {
        request_queues[room_type - 1].rear->next = new_node;
        request_queues[room_type - 1].rear = new_node;
    }
}

RequestQueueNode* dequeueRequest(int room_type) {
    if (request_queues[room_type - 1].front == NULL) {
        return NULL;  // Queue is empty
    }

    RequestQueueNode* front_node = request_queues[room_type - 1].front;
    request_queues[room_type - 1].front = front_node->next;

    if (request_queues[room_type - 1].front == NULL) {
        request_queues[room_type - 1].rear = NULL;  // Queue is now empty
    }

    return front_node;
}

void initializeRooms() {
    for (int i = 0; i < 10; i++) {
        shared_rooms[i].room_number = i + 1;
        shared_rooms[i].is_occupied = 0;
        // Assigning a random room type for demonstration purposes
        shared_rooms[i].room_type = (i % 3) + 1;
```

```c
        shared_rooms[i].room_type = (i % 3) + 1;
        shared_rooms[i].last_occupied_time = 0;  // Initialize timestamp to 0
        shared_rooms[i].user_name[0] = '\0';  // Initialize user name to an empty string
    }
}

int findNextAvailableRoom(int room_type) {
    for (int i = 0; i < 10; i++) {
        if (!shared_rooms[i].is_occupied && shared_rooms[i].room_type == room_type) {
            return i;
        }
    }
    return -1;  // No available room of the specified type
}

void bookRoom(int room_type, const char *user_name) {
    printf("\n--- Booking a Room ---\n");

    sem_wait(room_semaphore);

    int room_index = findNextAvailableRoom(room_type);

    if (room_index != -1) {
        shared_rooms[room_index].is_occupied = 1;
        shared_rooms[room_index].last_occupied_time = time(NULL);  // Update timestamp
        snprintf(shared_rooms[room_index].user_name, sizeof(shared_rooms[room_index].user_name), "%s", user_name);
        printf("Room %d (Type: %s) booked successfully by %s.\n", shared_rooms[room_index].room_number,
               room_type == SINGLE ? "Single" : (room_type == DOUBLE ? "Double" : "Suite"),
               shared_rooms[room_index].user_name);
    } else {
        printf("No available room of type %s. Adding to the queue.\n", room_type == SINGLE ? "Single" : (room_type == DOUBLE ? "Double" :
```

```c
            printf("No available room of type %s. Adding to the queue.\n", room_type == SINGLE ? "Single" : (room_type == DOUBLE ? "Double" : "Suite"));
            enqueueRequest(room_type, user_name);
        }

        sem_post(room_semaphore);
}

void releaseRoom(int room_number, const char *user_name) {
    printf("\n--- Releasing a Room ---\n");
    sem_wait(room_semaphore);

    if (!shared_rooms[room_number - 1].is_occupied) {
        printf("Room %d is already empty.\n", room_number);
        sem_post(room_semaphore);
    } else if (strcmp(shared_rooms[room_number - 1].user_name, user_name) != 0) {
        printf("You cannot release Room %d. It is booked by %s.\n", room_number, shared_rooms[room_number - 1].user_name);
        sem_post(room_semaphore);
    } else {
        shared_rooms[room_number - 1].is_occupied = 0;
        shared_rooms[room_number - 1].user_name[0] = '\0';  // Clear user name
        printf("Room %d released successfully.\n", room_number);
        sem_post(room_semaphore);

        // Check if there are queued requests for this room type
        int room_type = shared_rooms[room_number - 1].room_type;
        RequestQueueNode* front_request = dequeueRequest(room_type);

        if (front_request != NULL) {
            // Grant the oldest request
            bookRoom(room_type, front_request->user_name);
```

```c
            // Grant the oldest request
            bookRoom(room_type, front_request->user_name);
            free(front_request);
        }
    }


}



void displayRooms() {
    printf("\n--- Displaying Rooms ---\n");
    printf("%-10s %-15s %-15s %-25s %-15s\n", "Room", "Type", "Status", "Time Room Occupied", "User Name");
    printf("------------------------------------------------------------------------------------\n");
    for (int i = 0; i < 10; i++) {
        // Limit the width of the user name to 15 characters
        printf("%-10d %-15s %-15s %-25.*s %-15s\n", shared_rooms[i].room_number,
               shared_rooms[i].room_type == SINGLE ? "Single" : (shared_rooms[i].room_type == DOUBLE ? "Double" : "Suite"),
               shared_rooms[i].is_occupied ? "Occupied" : "Available",
               24, shared_rooms[i].is_occupied ? ctime(&shared_rooms[i].last_occupied_time) : "N/A",
               shared_rooms[i].is_occupied ? shared_rooms[i].user_name : "N/A");
    }
}



int main() {
    // Create a shared memory file descriptor
    int shared_memory_fd = shm_open("/hotel_shared_memory", O_CREAT | O_RDWR, S_IRUSR | S_IWUSR);
    if (shared_memory_fd == -1) {
```

```c
    if (shared_memory_fd == -1) {
        perror("shm_open");
        exit(EXIT_FAILURE);
    }

    // Set the size of the shared memory
    if (ftruncate(shared_memory_fd, SHARED_MEMORY_SIZE) == -1) {
        perror("ftruncate");
        exit(EXIT_FAILURE);
    }

    // Map the shared memory into the address space of the process
    shared_rooms = mmap(NULL, SHARED_MEMORY_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shared_memory_fd, 0);
    if (shared_rooms == MAP_FAILED) {
        perror("mmap");
        exit(EXIT_FAILURE);
    }

    // Initialize the semaphore in shared memory
    room_semaphore = mmap(NULL, sizeof(sem_t), PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0);
    sem_init(room_semaphore, 1, 1);

    // Initialize rooms
    initializeRooms();

    char user_name[10];

    printf("Enter your name: ");
    scanf("%s",user_name);

    int choice, room_number, room_type;
```

```c
    int choice, room_number, room_type;

    do {
        printf("\n--- HOTEL MANAGEMENT SYSTEM ---\n");
        printf("1. Display Rooms\n");
        printf("2. Book a Room\n");
        printf("3. Release a Room\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                displayRooms();
                break;
            case 2:
                printf("Enter room type (1: Single, 2: Double, 3: Suite): ");
                scanf("%d", &room_type);
                bookRoom(room_type, user_name);
                break;
            case 3:
                printf("Enter room number to release: ");
                scanf("%d", &room_number);
                releaseRoom(room_number, user_name);
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
```

```c
            case 3:
                printf("Enter room number to release: ");
                scanf("%d", &room_number);
                releaseRoom(room_number, user_name);
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }

    } while (choice != 4);

    // Clean up: Close the shared memory file descriptor and unlink the shared memory object
    close(shared_memory_fd);
    shm_unlink("/hotel_shared_memory");

    // Destroy the semaphore
    sem_destroy(room_semaphore);

    return 0;
}
```

Output:

```
--- HOTEL MANAGEMENT SYSTEM ---
1. Display Rooms
2. Book a Room
3. Release a Room
4. Exit
Enter your choice: 2
Enter room type (1: Single, 2: Double, 3: Suite): 1

--- Booking a Room ---
Room 1 (Type: Single) booked successfully by Saksham.

--- HOTEL MANAGEMENT SYSTEM ---
1. Display Rooms
2. Book a Room
3. Release a Room
4. Exit
Enter your choice: 1

--- Displaying Rooms ---
Room       Type            Status          Time Room Occupied         User Name
-----------------------------------------------------------------------------------
1          Single          Occupied        Tue Nov 14 10:14:17 2023   Saksham
2          Double          Available       N/A                        N/A
3          Suite           Available       N/A                        N/A
4          Single          Available       N/A                        N/A
5          Double          Available       N/A                        N/A
6          Suite           Available       N/A                        N/A
7          Single          Available       N/A                        N/A
8          Double          Available       N/A                        N/A
9          Suite           Available       N/A                        N/A
10         Single          Available       N/A                        N/A

--- HOTEL MANAGEMENT SYSTEM ---
1. Display Rooms
2. Book a Room
3. Release a Room
4. Exit
Enter your choice: 
```

```
--- HOTEL MANAGEMENT SYSTEM ---
1. Display Rooms
2. Book a Room
3. Release a Room
4. Exit
Enter your choice: 1

--- Displaying Rooms ---
Room      Type          Status        Time Room Occupied        User Name
-----------------------------------------------------------------------------------
1         Single        Occupied      Tue Nov 14 10:14:17 2023  Saksham
2         Double        Occupied      Tue Nov 14 10:15:38 2023  Saksham
3         Suite         Available     N/A                       N/A
4         Single        Available     N/A                       N/A
5         Double        Occupied      Tue Nov 14 10:15:44 2023  Saksham
6         Suite         Available     N/A                       N/A
7         Single        Available     N/A                       N/A
8         Double        Occupied      Tue Nov 14 10:15:49 2023  Saksham
9         Suite         Available     N/A                       N/A
10        Single        Available     N/A                       N/A

--- HOTEL MANAGEMENT SYSTEM ---
1. Display Rooms
2. Book a Room
3. Release a Room
4. Exit
Enter your choice: 2
Enter room type (1: Single, 2: Double, 3: Suite): 2

--- Booking a Room ---
No available room of type Double. Adding to the queue.
```

```
--- Displaying Rooms ---
Room      Type          Status        Time Room Occupied        User Name
-----------------------------------------------------------------------------------
1         Single        Occupied      Tue Nov 14 10:14:17 2023  Saksham
2         Double        Occupied      Tue Nov 14 10:15:38 2023  Saksham
3         Suite         Available     N/A                       N/A
4         Single        Available     N/A                       N/A
5         Double        Occupied      Tue Nov 14 10:15:44 2023  Saksham
6         Suite         Available     N/A                       N/A
7         Single        Available     N/A                       N/A
8         Double        Occupied      Tue Nov 14 10:15:49 2023  Saksham
9         Suite         Available     N/A                       N/A
10        Single        Available     N/A                       N/A

--- HOTEL MANAGEMENT SYSTEM ---
1. Display Rooms
2. Book a Room
3. Release a Room
4. Exit
Enter your choice: 3
Enter room number to release: 1

--- Releasing a Room ---
Room 1 released successfully.
```

```
--- HOTEL MANAGEMENT SYSTEM ---
1. Display Rooms
2. Book a Room
3. Release a Room
4. Exit
Enter your choice: 2
Enter room type (1: Single, 2: Double, 3: Suite): 1

--- Booking a Room ---
Room 1 (Type: Single) booked successfully by Saksham.

--- HOTEL MANAGEMENT SYSTEM ---
1. Display Rooms
2. Book a Room
3. Release a Room
4. Exit
Enter your choice: 1

--- Displaying Rooms ---
Room      Type        Status      Time Room Occupied        User Name
---------------------------------------------------------------------------------
1         Single      Occupied    Tue Nov 14 10:14:17 2023  Saksham
2         Double      Available   N/A                       N/A
3         Suite       Available   N/A                       N/A
4         Single      Available   N/A                       N/A
5         Double      Available   N/A                       N/A
6         Suite       Available   N/A                       N/A
7         Single      Available   N/A                       N/A
8         Double      Available   N/A                       N/A
9         Suite       Available   N/A                       N/A
10        Single      Available   N/A                       N/A

--- HOTEL MANAGEMENT SYSTEM ---
1. Display Rooms
2. Book a Room
3. Release a Room
4. Exit
Enter your choice:
```

FUTURE WORK

1) **Multi-User Support :** Implementing robust support for multiple users to ensure the system can handle concurrent booking and release requests fromdifferent users effectively.

2) **Dynamic Room Allocation :** Developing an advanced room allocationalgorithm that considers user preferences, special requests, and real-time availability to optimize the allocation process.

3) **Security Measures :** Integrating user authentication and authorization mechanisms to enhance the overall security of the system and protect sensitiveuser and room data.

4). **Data Persistence :** Implementing a reliable database or file system for storing room and user data, enabling data persistence across system restarts and ensuringa more stable storage solution.

5) **Reporting and Analytics:** Integrating reporting and analytics tools to generate valuable insights into room occupancy trends, booking patterns, and other metrics, empowering hotel management with informed decision-making capabilities.

LIMITATIONS

While the hotel management system is functional and effective, it does have some limitations:

1. Limited Room Types:

   - The system currently supports three room types (Single, Double, Suite). Future enhancements could explore a more diverse range of room categories to accommodate varying user needs.

2. Basic User Interaction:

   - The user interaction in the current system is relatively simple, mainly involving entering a name and choosing options. Enhancements could include amore user-friendly graphical interface and additional functionalities.

3. Single-User Input:

   - The system assumes a single user interacting with it. In a real-world scenario,multiple users may be accessing the system simultaneously. Extending the system to handle concurrent users could enhance its practicality.

4. Limited Error Handling:

   - While the system incorporates basic error handling, more robust mechanisms could be implemented to handle unforeseen situations, ensuring a smoother userexperience and system stability.

5. Static Room Allocation:

   - The current system allocates rooms based on availability without considering user preferences or specific requirements. A more sophisticated room allocation algorithm could be implemented to consider user preferences and optimize roomassignments.

6. No Persistence:

- The system does not include a mechanism for data persistence. Implementing a database or file system for storing room and user data would allow informationto persist across system restarts.

7. Lack of Security Measures:

- Security features, such as user authentication and authorisation, are not implemented in the current version. Integrating these measures would enhancethe system's overall security and protect sensitive user data.

8. Limited Reporting and Analytics:

- The system lacks advanced reporting and analytics capabilities. Integrating tools for generating reports on room occupancy trends, user booking history, andother metrics would provide valuable insights for hotel management.

9. Scalability Concerns:

- The system's scalability for a larger number of rooms or concurrent users isnot thoroughly tested. Ensuring scalability is crucial for the system's effectiveness in a real-world hotel environment.

10. No Cancellation Mechanism:

- The system does not include a mechanism for users to cancel their bookings. Implementing a cancellation feature would enhance user flexibility and accommodate changes in travel plans.

CONCLUSION

In conclusion, the hotel management system presented herein provides a robust platform for exploring concurrent programming principles, shared memory usage, and efficient resource coordination in a real-world scenario. The project successfully manages room bookings, releases, and displays room status through the implementationof threads, semaphores, and shared memory. The system adeptly handles scenarios suchas room availability, user requests, and orderly processing of booking queues, demonstrating its effectiveness in a multi-user environment.

The hotel management system, while achieving its goals, has identified areas for improvement. Future enhancements could focus on refining the user experience, bolstering input validation, and fortifying error handling.Looking forward, there's significant potential to elevate the system by addressing these limitations and integrating advanced features like real-time analytics and personalized interactions. Incorporating artificial intelligence and machine learning may further enhance decision-making, offering insights into occupancy patterns and user preferences.In an era of constant technological progress, the hotel management system serves as a foundation for innovative solutions in the hospitality industry. Its adaptability and growth potential position it as a promising candidate for further research and development, contributing to the evolution of more efficient and user-centric hotel management systems.

REFERENCES

- Dijkstra, E. W. (1971). "Hierarchical Ordering of SequentialProcesses." Acta Informatica, 1(2), 115–138.
- Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). "Operating System Concepts." John Wiley & Sons.
- Lamport, L. (1978). "Time, Clocks, and the Ordering of Events in a Distributed System." Communications of the ACM, 21(7), 558–565.