EAST WEST UNIVERSITY

# CSE 325: Operating Systems

## Project Report

## Group: 03

## Title: Restaurant Reservation System


**Submitted By:**

Shamima Sharmin Ananna

ID: 2022-1-60-148 Roll: 27

Tasnuva Tasnim Nova

ID: 2022-1-60-266 Roll: 30

Taniz Fatema Jarin

ID: 2022-1-60-065 Roll: 26

**Submitted To:**

Dr. Md. Nawab Yousuf Ali

Professor

Department of Computer Science and Engineering

East West University, Bangladesh


Date of Submission: 29.05.2024

# Restaurant Reservation System

Using multithreads, mutex locks, and semaphores, we have implemented a solution that synchronizes the customers in a restaurant table reservation system where customers can reserve, cancel or modify their table reservation.

## Project Description

The Restaurant Table Reservation System is a multi-threaded application designed to manage the process of customers reserving and vacating tables in a restaurant. The system utilizes semaphores and mutexes to control access to shared resources and ensure thread safety, providing a robust simulation of a real-world restaurant reservation system. Initially, all the tables are empty. The customers will come and reserve the table one by one. If there are more tables available, customers can modify their tables. Any customer can cancel their reservations as well. If the customer number is greater than the table numbers, there will be no option for modification. After modification or cancellation, if I want to proceed, the customers will leave the tables after 2 seconds and the tables will be available again. Once all the customer threads are terminated, the table threads the main program will be terminated. This program works for any number of customers and tables. We have got a special case where customer number is more than the table number, in that case, customers can not modify anything. We have used first come first serve method in this case. Customers will come and reserve table, the remaining will wait till any customer leave the table. When every customer can reserve a table, the program will be terminated. We have allocated memory for data structures dynamically based on the input parameters.

# Overview

Customers will maintain first in first service method. When all the customers get tables, they will leave after a certain time and the tables will be available again and the program wil be terminated.

Methods are:

- pthread_mutex_lock()
- pthread_mutex_unlock()
- sem_post()
- pthread_exit()
- sem_wait()
- sem_init()
- pthread_mutex_init()
- pthread_create()
- pthread_join()
- sem_destroy()
- pthread_mutex_destroy()

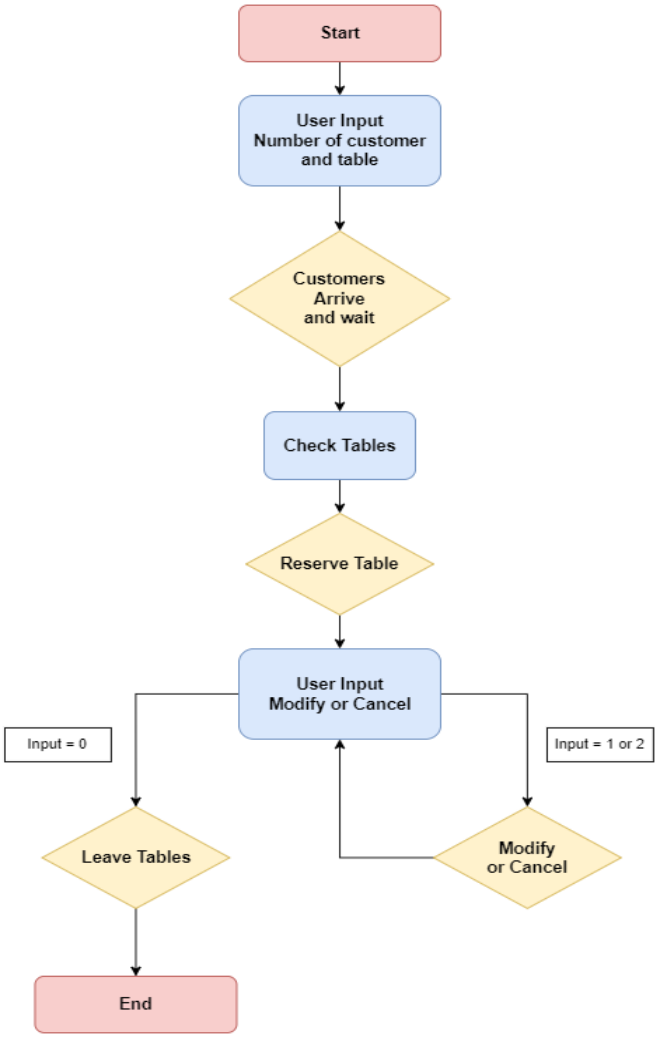
**Operating System:** Linux

To run this “project.c” file, we used these command line arguments in terminal:

gcc project.c -lpthread

./a.out

# Working Flowchart




At first, it will take user input as number of customer and number of tables. After that, customers will wait and reserve tables. Then we will get the option, 1 to modify tables, 2 to cancel reservation, 0 to proceed. If there are tables available, customers can modify. If we insert 0, the customers will leave after 2 seconds, all the table will be empty and the program will be terminated.

## Source Code

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>

int tablesAvailable;
sem_t tableSem; //control access the tables
pthread_mutex_t mutex; //protect CS
int *tableStatus;
int tableNum;

typedef struct {
   long id;
   int tableNum;
} Customer;

void *customer(void* arg) {
   long id = (long) arg + 1;
   printf("Customer[%ld]\tArrives and wants to reserve a table.\n", id);
   sem_wait(&tableSem);
   pthread_mutex_lock(&mutex);
   tablesAvailable--;
   printf("Customer[%ld]\tReserved a table. Tables Available: %d\n", id,
tablesAvailable);
   pthread_mutex_unlock(&mutex);
   sleep(2);
   pthread_mutex_lock(&mutex);
   tablesAvailable++;
   printf("Customer[%ld]\tLeft the table. Tables Available: %d\n", id,
tablesAvailable);
   pthread_mutex_unlock(&mutex);
   sem_post(&tableSem);
```

```
    pthread_exit(NULL);
}

void *leaveTable(void *arg) {
    Customer *customer = (Customer *) arg;
    long id = customer->id;
    int tableNum = customer->tableNum;
    sleep(2);
    pthread_mutex_lock(&mutex);
    tableStatus[tableNum - 1] = -1;
    tablesAvailable++;
    printf("Customer[%ld]\tLeft the table. Tables Available: %d\n", id,
tablesAvailable);
    pthread_mutex_unlock(&mutex);
    sem_post(&tableSem);
    free(customer);
    pthread_exit(NULL);
}

Customer *reserveTable(void *arg) {
    long id = (long) arg + 1;
    printf("Customer[%ld]\tArrives and wants to reserve a table.\n", id);
    sem_wait(&tableSem);
    pthread_mutex_lock(&mutex);
    int reservedTable = -1;
    for (int i = 0; i < tableNum; i++) {
        if (tableStatus[i] == -1) {
            tableStatus[i] = id;
            tablesAvailable--;
            reservedTable = i + 1;
            printf("Customer[%ld]\tReserved table %d.\nTables Available: %d.\nTable
NO: ", id, reservedTable, tablesAvailable);
            for (int j = 0; j < tableNum; j++) {
                if (tableStatus[j] == -1) {
                    printf("%d ", j + 1);
```

```
            }
        }
        printf("\n");
        break;
    }
  }
  pthread_mutex_unlock(&mutex);

  Customer *customer = (Customer *) malloc(sizeof(Customer));
  customer->id = id;
  customer->tableNum = reservedTable;
  return customer;
}

void displayAvailableTables() {
  pthread_mutex_lock(&mutex);
  printf("Available Tables: ");
  int available = 0;
  for (int i = 0; i < tableNum; i++) {
    if (tableStatus[i] == -1) {
      printf("%d ", i + 1);
      available++;
    }
  }
  if (available == 0) {
    printf("No tables available");
  }
  printf("\n");
  pthread_mutex_unlock(&mutex);
}

int main() {
  int customerNum;
  printf("Enter number of customers: ");
  scanf("%d", &customerNum);
```

```
printf("Enter number of tables: ");
scanf("%d", &tableNum);

if (customerNum > tableNum) {

    printf("Restaurant Table Reservation System\n");
    printf("Customers: %d, Tables: %d\n", customerNum, tableNum);
    printf("-----------------------------------------\n");

    sem_init(&tableSem, 0, tableNum);
    pthread_mutex_init(&mutex, NULL);
    pthread_t customerThread[customerNum];

    tablesAvailable = tableNum;

    for(long id = 0; id < customerNum; ++id) {
        pthread_create(&customerThread[id], NULL, customer, (void*)id);
        usleep(500);
    }

    for(int id = 0; id < customerNum; ++id) {
        pthread_join(customerThread[id], NULL);
    }

    sem_destroy(&tableSem);
    pthread_mutex_destroy(&mutex);

    printf("-----------------------------------------\n");
    printf("Restaurant Table Reservation System End!\n");
    return 0;
}

printf("Restaurant Table Reservation System\n");
printf("Customers: %d, Tables: %d\n", customerNum, tableNum);
printf("-----------------------------------------\n");
```

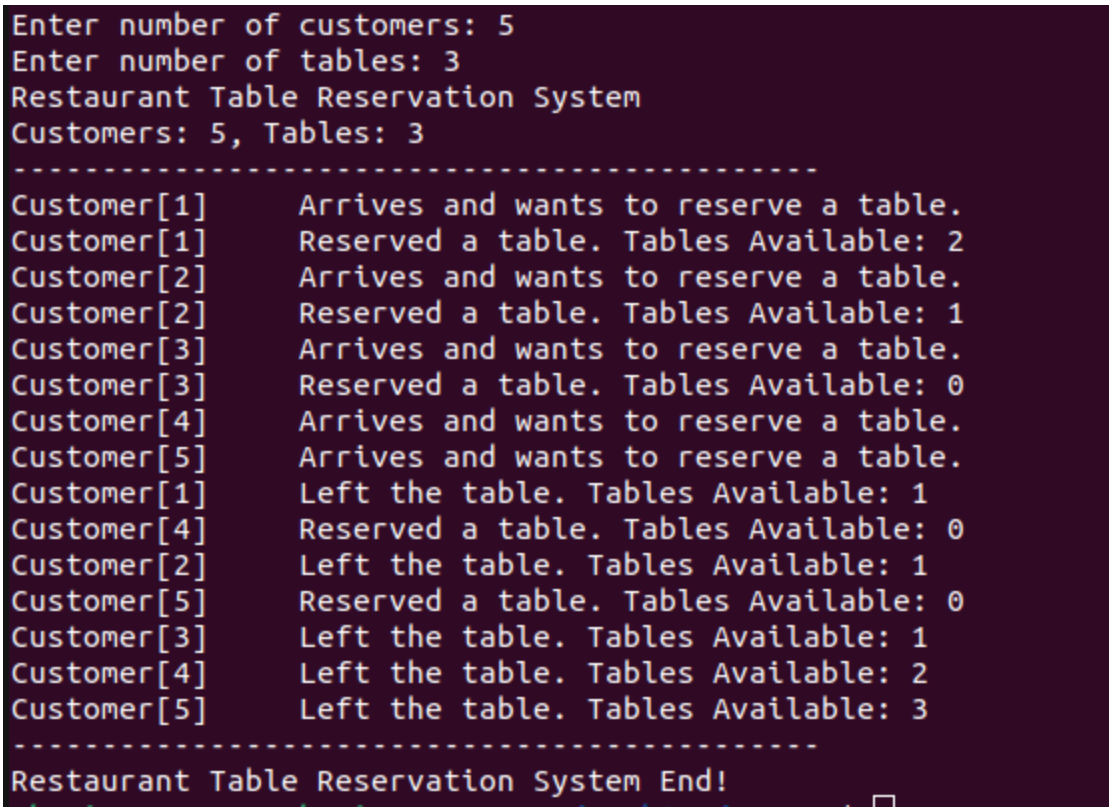
```
    sem_init(&tableSem, 0, tableNum);
    pthread_mutex_init(&mutex, NULL);
    pthread_t customerThread[customerNum];
    tableStatus = (int*) malloc(tableNum * sizeof(int));
    tablesAvailable = tableNum;

    for (int i = 0; i < tableNum; i++) {
        tableStatus[i] = -1;
    }

    Customer *customers[customerNum];
    for(long id = 0; id < customerNum; ++id) {
        customers[id] = reserveTable((void*)id);
        usleep(500);
    }

    while (1) {
        int choice, customerId, newTableNum;
        printf("1 to cancel a reservation\n2 to modify table no\n0 to proceed\nEnter
your choice: ");
        scanf("%d", &choice);
        if (choice == 0) break;
        switch (choice) {
            case 1:
                printf("Enter customer ID to cancel reservation: ");
                scanf("%d", &customerId);
                pthread_mutex_lock(&mutex);
                int cancelled = 0;
                for (int i = 0; i < tableNum; i++) {
                    if (tableStatus[i] == customerId) {
                        tableStatus[i] = -1;
                        tablesAvailable++;
                        sem_post(&tableSem);
                        printf("Reservation for customer[%d] cancelled.\nTable %d is now
```

```
available.\n", customerId, i + 1);
                    cancelled = 1;
                    break;
                }
            }
            if (!cancelled) {
                printf("No reservation found for customer[%d].\n", customerId);
            }
            pthread_mutex_unlock(&mutex);
            break;
        case 2:
            printf("Enter customer ID to modify table reservation: ");
            scanf("%d", &customerId);
            displayAvailableTables();
            printf("Enter new table number: ");
            scanf("%d", &newTableNum);
            pthread_mutex_lock(&mutex);

            int modified = 0;
            for (int i = 0; i < tableNum; i++) {
                if (tableStatus[i] == customerId) {
                    if (tableStatus[newTableNum - 1] == -1) {
                        tableStatus[newTableNum - 1] = customerId;
                        tableStatus[i] = -1;
                        printf("Customer[%d] moved from table %d to table %d.\n",
customerId, i + 1, newTableNum);
                        modified = 1;
                    } else {
                        printf("Table %d is not available.\n", newTableNum);
                    }
                    break;
                }
            }
            if (!modified && tableStatus[newTableNum - 1] != -1) {
                printf("No reservation found for customer[%d].\n", customerId);
```

```
            }
            pthread_mutex_unlock(&mutex);
            break;
        default:
            printf("Invalid choice. Please try again.\n");
    }
    displayAvailableTables();
  }

  for(int id = 0; id < customerNum; ++id) {
    if (customers[id] != NULL && customers[id]->tableNum != -1 &&
customers[id]->id != 1) {
        pthread_create(&customerThread[id], NULL, leaveTable,
(void*)customers[id]);
    }
  }

  for(int id = 0; id < customerNum; ++id) {
    if (customers[id] != NULL && customers[id]->tableNum != -1 &&
customers[id]->id != 1) {
        pthread_join(customerThread[id], NULL);
    }
  }

  sem_destroy(&tableSem);
  pthread_mutex_destroy(&mutex);
  free(tableStatus);
  printf("-----------------------------------------\n");
  printf("Restaurant Table Reservation System End!\n");
  return 0;
}
```

# Output

```
Enter number of customers: 3
Enter number of tables: 5
Restaurant Table Reservation System
Customers: 3, Tables: 5
-----------------------------------------
Customer[1]     Arrives and wants to reserve a table.
Customer[1]     Reserved table 1.
Tables Available: 4.
Table NO: 2 3 4 5
Customer[2]     Arrives and wants to reserve a table.
Customer[2]     Reserved table 2.
Tables Available: 3.
Table NO: 3 4 5
Customer[3]     Arrives and wants to reserve a table.
Customer[3]     Reserved table 3.
Tables Available: 2.
Table NO: 4 5
1 to cancel a reservation
2 to modify table no
0 to proceed
Enter your choice: 1
Enter customer ID to cancel reservation: 1
Reservation for customer[1] cancelled.
Table 1 is now available.
Available Tables: 1 4 5
1 to cancel a reservation
2 to modify table no
0 to proceed
Enter your choice: 2
Enter customer ID to modify table reservation: 3
Available Tables: 1 4 5
Enter new table number: 5
Customer[3] moved from table 3 to table 5.
Available Tables: 1 3 4
1 to cancel a reservation
2 to modify table no
0 to proceed
Enter your choice: 0
Customer[2]     Left the table. Tables Available: 4
Customer[3]     Left the table. Tables Available: 5
-----------------------------------------
Restaurant Table Reservation System End!
```




## Conclusion

The Restaurant Table Reservation System successfully demonstrates the use of multithreading and synchronization techniques to manage table reservations efficiently. By using semaphores and mutexes, the system ensures consistent and conflict-free access to shared resources. While the system has certain limitations, it provides a solid foundation for further development and enhancements, such as a graphical user interface and support for distributed environments. This project highlights the importance of concurrency control in real-time reservation systems and serves as a practical application of threading and synchronization concepts in C programming on the Ubuntu operating system.