EAST WEST UNIVERSITY

Department of CSE

Course Code
CSE325

Course Title
Operating Systems

Project Report

WILL & LOU'S CAFÉ PROBLEM

Section : 02 **Semester : Fall2024**

Submitted By

Group : 9		
Name	**ID**	**Roll**
Abrar Hossain Zahin	**2022-2-60-040**	**22**

Submitted To

Dr. Md. Nawab Yousuf Ali
Professor
Department of Computer Science and Engineering

Date of Report Submitted : 22 January, 2025

Table of Contents :

Acknowledgement

We would like to express our deepest gratitude to Dr. Nawab Yousuf Ali Sir for his invaluable guidance, encouragement, and support throughout the development of this project, *"Will & Lou's Café Simulator."* His expertise and constructive feedback have been instrumental in enhancing our understanding of **multithreading**, **synchronization**, & **resource management**. Thank you, Sir, for inspiring us to explore and excel in the field of computer science.

Abstract

The program simulates the operations of a charming café, "Will & Lou's," where customers enjoy with limited tables and the unique ability to borrow books while savoring their treats. The simulation efficiently handles concurrent customer activities and ensuring smooth operations through effective synchronization techniques. **Semaphore-based Resource Management** ensures table availability for customers while managing a queue of waiting customers. **Threaded Customer Simulation e**ach customer's actions (seating, borrowing books, and leaving) are handled in individual threads for realistic simulation. **Mutex-based Synchronization:** Prevents data inconsistencies in shared resources like book inventory and table statuses. **Interactive User Interface** Allows dynamic addition of customers, providing flexibility in testing various scenarios. The simulation emphasizes resource sharing, fair access, and a delightful experience reminiscent of the café's charming atmosphere.

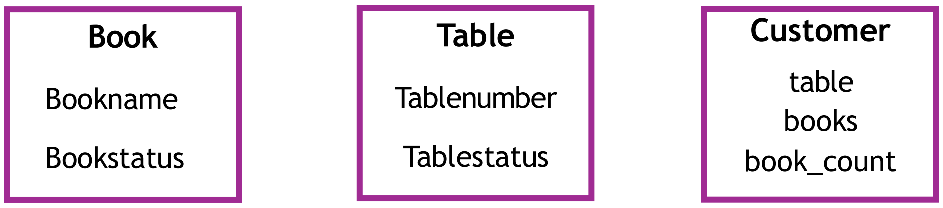
Problem Statement

'Will and Lou' are opened a charming café where customers can enjoy reading books while dining. To provide a seamless experience for every customer. So we've designed a system with the following guidelines :

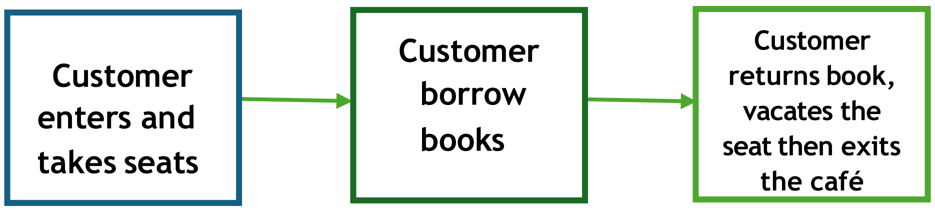
1) A customer may enter the café only if a table is available for them to sit.
2) The system will maintain a record of book borrowing and returning to efficiently manage the book inventory.
3) Once customers have finished reading, eating, or drinking, they should vacate the table, allowing new customers to use it.

Project Description

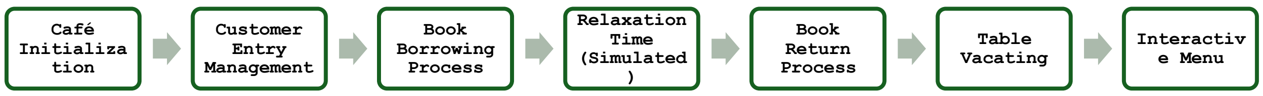
We created a program to address the café management challenge. The program simulates the café environment, which includes a fixed number of tables and a fixed inventory of books. In the simulation, tables, books, and customers are treated as objects, each with its own attributes.



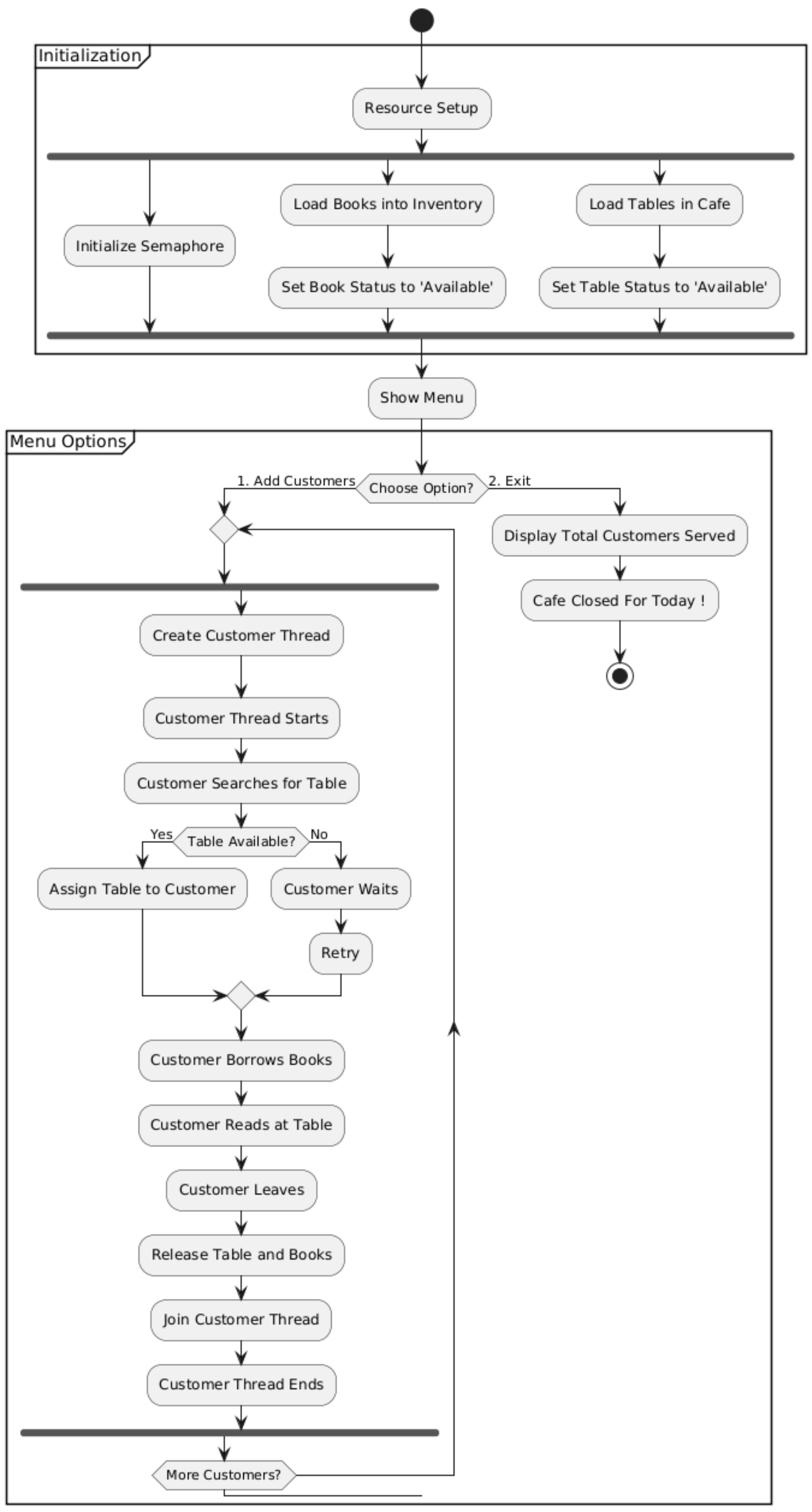
Initially, the program initializes all the tables and books into their respective **global arrays** using threads. Afterward, it begins loading customers and ensures proper synchronization as they occupy tables, borrow and return books, and eventually leave the café.


Customer enters and takes seats
Customer borrow books
Customer returns book, vacates the seat then exits the café

This project leverages **semaphores**, **mutex locks**, and **threads** to handle real-world scenarios involving concurrency and resource management. The simulator Imitation the café's unique features, ensuring a smooth experience for every visitor.



Flow Chart


Initialization
Resource Setup
Initialize Semaphore
Load Books into Inventory
Set Book Status to 'Available'
Load Tables in Cafe
Set Table Status to 'Available'
Show Menu
Menu Options
1. Add Customers
Choose Option?
2. Exit
Display Total Customers Served
Cafe Closed For Today !
Create Customer Thread
Customer Thread Starts
Customer Searches for Table
Yes
Table Available?
No
Assign Table to Customer
Customer Waits
Retry
Customer Borrows Books
Customer Reads at Table
Customer Leaves
Release Table and Books
Join Customer Thread
Customer Thread Ends
More Customers?

Code

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

#define MAX_BOOK 10
#define MAX_TABLE 5
#define MAX_CUSTOMER 200

// Data structures
typedef struct{
    char Bookname[200];
    int Bookstatus; // 1: Available, 0: Borrowed
} Book;
typedef struct{
    int Tablenumber;
    int Tablestatus; // 1: Available, 0: Occupied
} Table;
typedef struct{
    int table;
    int books[3];
    int book_count;
} Customer;

// Global variables
Table tables[MAX_TABLE];
Book book_inventory[MAX_BOOK];
Customer customers[MAX_CUSTOMER];
int total_tables = 0, total_books = 0,
total_customers = 0;

// Synchronization tools
sem_t table_semaphore;
sem_t
customer_semaphores[MAX_CUSTOMER];
pthread_mutex_t table_mutex =
PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t book_mutex =
PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t customer_mutex =
PTHREAD_MUTEX_INITIALIZER;
// Function prototypes
void initialize_resources();
void load_books();
void load_tables();
```

```c
        pthread_mutex_unlock(&table_mutex);
        sem_post(&table_semaphore);

        if (table_found)
            break;

        printf("#--------------------------------------#\n");
        printf("No tables available. Customer %d waiting...\n",
id);
        sleep(1);
    }
    pthread_mutex_lock(&book_mutex);
    int books_picked = 0;
    for (int i = 0; i < total_books && books_picked < 3; i++){
        if (book_inventory[i].Bookstatus == 1)
        {
            book_inventory[i].Bookstatus = 0;
            customers[id].books[books_picked++] = i;
        }
    }
    customers[id].book_count = books_picked;
    pthread_mutex_unlock(&book_mutex);

    printf("Customer %d borrowed %d books and is seated
at table %d.\n", id, books_picked, customers[id].table);
    sleep(2);

    pthread_mutex_lock(&table_mutex);
    tables[customers[id].table - 1].Tablestatus = 1;
    pthread_mutex_unlock(&table_mutex);

    pthread_mutex_lock(&book_mutex);
    for (int i = 0; i < books_picked; i++)
    {
        book_inventory[customers[id].books[i]].Bookstatus =
1;
    }
    pthread_mutex_unlock(&book_mutex);

    printf("Customer %d has left table %d.\n", id,
customers[id].table);
}
// Customer thread function
void *customer_thread(void *arg){
    int id = *(int *)arg;
    free(arg);
    handle_customer(id);
    return NULL;
```

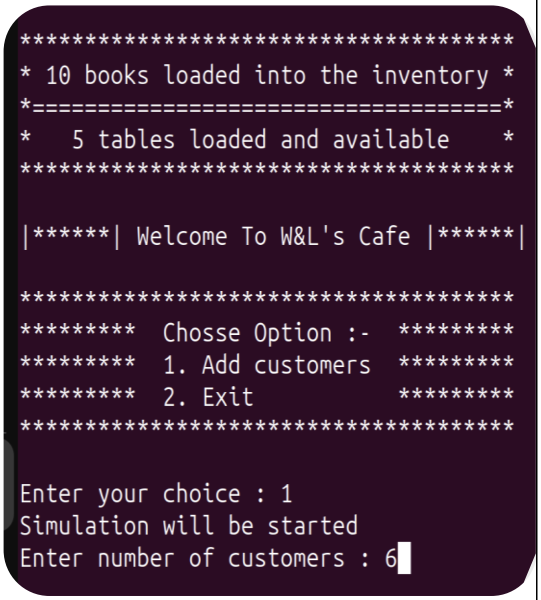
```
void handle_customer(int id);
void *customer_thread(void *arg);
void show_menu();

// Initialize resources
void initialize_resources(){
    sem_init(&table_semaphore, 0, 1);
    for (int i = 0; i < MAX_CUSTOMER; i++)
    {       sem_init(&customer_semaphores[i], 0,
0);
    }
}
// Load book inventory
void load_books(){
    for (int i = 0; i < MAX_BOOK; i++){
snprintf(book_inventory[i].Bookname,
sizeof(book_inventory[i].Bookname), "Book %d",
i + 1);
        book_inventory[i].Bookstatus = 1;
        total_books++;
    }
printf("\n\n****************************************");
    printf("\n* %d books loaded into the inventory
*\n", total_books);
}
// Load tables
void load_tables(){
    for (int i = 0; i < MAX_TABLE; i++){
        tables[i].Tablenumber = i + 1;
        tables[i].Tablestatus = 1;
        total_tables++;
    }
printf("*=================================
=====*\n");
    printf("*   %d tables loaded and available
*\n", total_tables);
printf("**************************************\n");
}
// Customer handling
void handle_customer(int id){
    printf("Customer %d searching for a
table...\n", id);
    while (1){
        sem_wait(&table_semaphore);
pthread_mutex_lock(&table_mutex);

        int table_found = 0;
        for (int i = 0; i < total_tables; i++){
            if (tables[i].Tablestatus == 1)
            {               tables[i].Tablestatus = 0;
```

```
}
// Show menu
void show_menu(){
    int choice;
    while (1)
    {
        printf("\n|******| Welcome To W&L's Cafe |******|\n");
        printf("\n**************************************\n");
        printf("*********  Chosse Option :-  *********\n");
        printf("*********  1. Add customers  *********\n*********
2. Exit          *********\n");
        printf("**************************************\n");
        printf("\nEnter your choice : ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
        {
            int n;
            printf("Simulation will be started\n");
            printf("Enter number of customers : ");
            scanf("%d", &n);
            pthread_t threads[n];
            for (int i = 0; i < n; i++){
                int *id = malloc(sizeof(int));
                *id = i + total_customers;
                pthread_create(&threads[i], NULL,
customer_thread, id);
            }
            for (int i = 0; i < n; i++)
            {
            pthread_join(threads[i], NULL);
            }
            break;
        }
        case 2:           printf("\n*****************************\n");
            printf("* %2d Customer Visited Today *\n",
total_customers);
            printf("** Cafe Closed For Today ! **");
printf("\n*****************************\n\n");
            return;
        default:
            printf("Invalid choice. Please try again.\n");
        }
    }
}
int main(){
    srand(time(NULL));
    initialize_resources();
    load_books();
    load_tables();
```

```
            customers[id].table =
tables[i].Tablenumber;
customers[id].book_count = 0;
            total_customers++;
            table_found = 1;
            printf("Customer %d assigned to table
%d.\n", id, tables[i].Tablenumber);
            break;
         }
      }
```

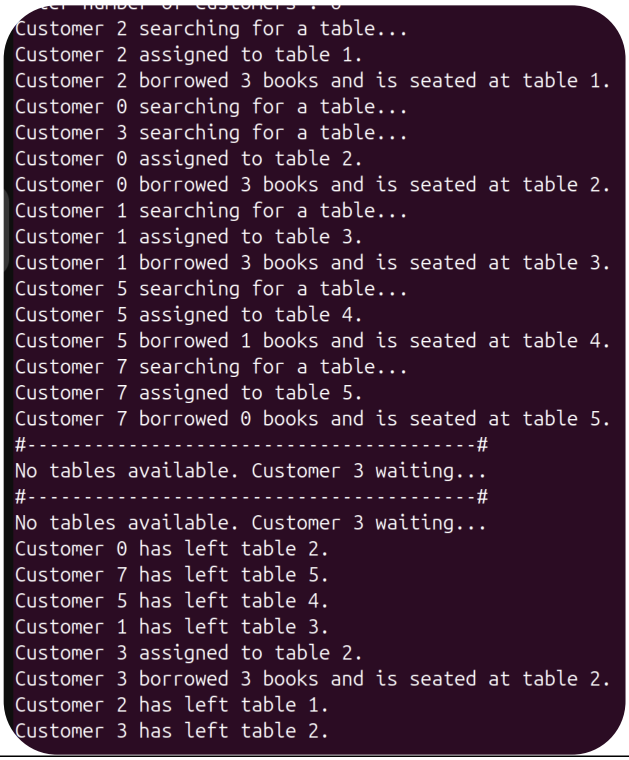
```
    show_menu();
    return 0;
}
```

Input :



```
***************************************
* 10 books loaded into the inventory *
*=====================================*
*   5 tables loaded and available    *
***************************************

|******| Welcome To W&L's Cafe |******|

***************************************
*********  Chosse Option :-  *********
*********  1. Add customers  *********
*********  2. Exit           *********
***************************************

Enter your choice : 1
Simulation will be started
Enter number of customers : 6
```

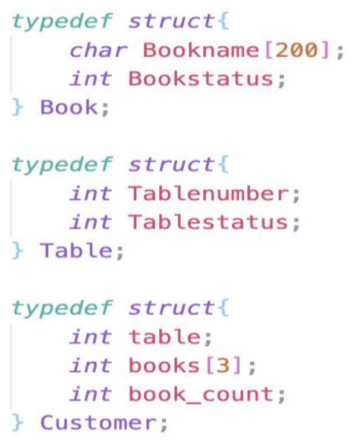
Output :



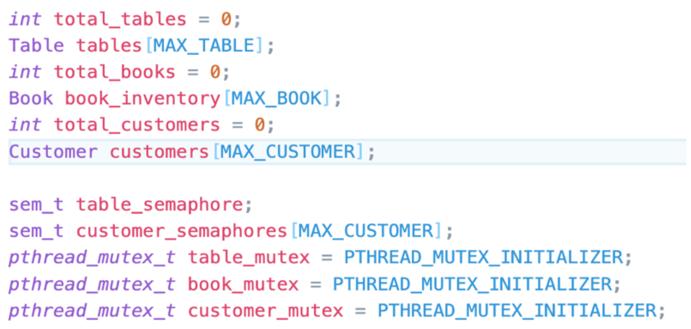
```
Customer 2 searching for a table...
Customer 2 assigned to table 1.
Customer 2 borrowed 3 books and is seated at table 1.
Customer 0 searching for a table...
Customer 3 searching for a table...
Customer 0 assigned to table 2.
Customer 0 borrowed 3 books and is seated at table 2.
Customer 1 searching for a table...
Customer 1 assigned to table 3.
Customer 1 borrowed 3 books and is seated at table 3.
Customer 5 searching for a table...
Customer 5 assigned to table 4.
Customer 5 borrowed 1 books and is seated at table 4.
Customer 7 searching for a table...
Customer 7 assigned to table 5.
Customer 7 borrowed 0 books and is seated at table 5.
#--------------------------------------#
No tables available. Customer 3 waiting...
#--------------------------------------#
No tables available. Customer 3 waiting...
Customer 0 has left table 2.
Customer 7 has left table 5.
Customer 5 has left table 4.
Customer 1 has left table 3.
Customer 3 assigned to table 2.
Customer 3 borrowed 3 books and is seated at table 2.
Customer 2 has left table 1.
Customer 3 has left table 2.
```

Code Explanation

Firstly I declared the data structures. Then global variables, shared variables semphores and mutex.

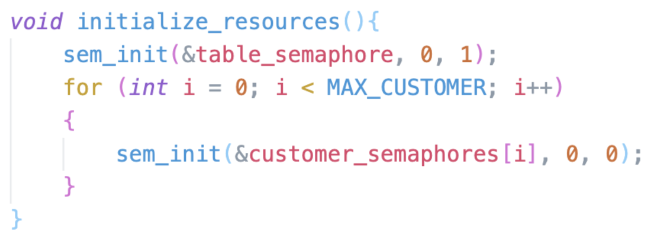


```
typedef struct{
    char Bookname[200];
    int Bookstatus;
} Book;

typedef struct{
    int Tablenumber;
    int Tablestatus;
} Table;

typedef struct{
    int table;
    int books[3];
    int book_count;
} Customer;
```



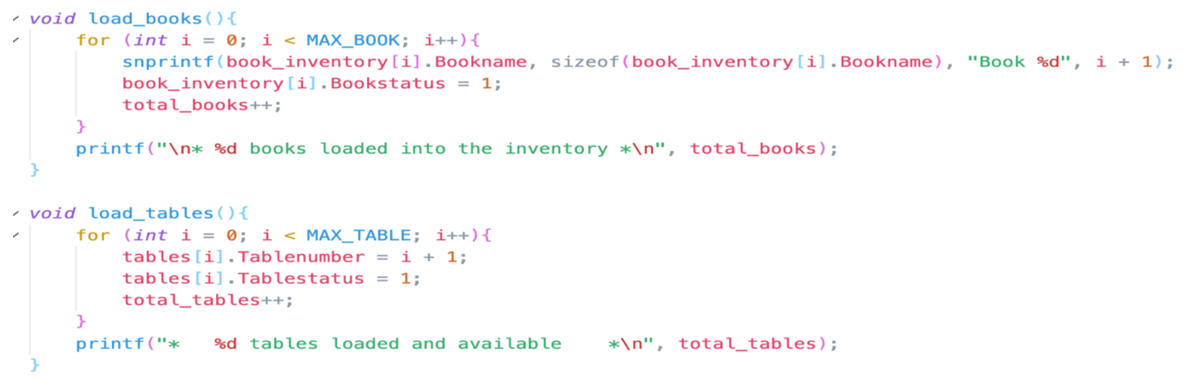
```
int total_tables = 0;
Table tables[MAX_TABLE];
int total_books = 0;
Book book_inventory[MAX_BOOK];
int total_customers = 0;
Customer customers[MAX_CUSTOMER];

sem_t table_semaphore;
sem_t customer_semaphores[MAX_CUSTOMER];
pthread_mutex_t table_mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t book_mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t customer_mutex = PTHREAD_MUTEX_INITIALIZER;
```

All the semaphore are being initiated in initialize_resources() function. Prepares semaphores and mutexes for use.



```
void initialize_resources(){
    sem_init(&table_semaphore, 0, 1);
    for (int i = 0; i < MAX_CUSTOMER; i++)
    {
        sem_init(&customer_semaphores[i], 0, 0);
    }
}
```

The load_books() and load_tables() functions work by initializing their respective resources: the book inventory and the seating tables. In load_books(), a loop iterates through the book_inventory array, assigning each book & marking it as available by setting its status to 1. Similarly, load_tables() iterates through the tables array, assigning each table a unique number and marking it as available. Both functions update global counters to track the total resources initialized and provide user feedback through printed messages, confirming the successful setup of books and tables for the simulation.



```
void load_books(){
    for (int i = 0; i < MAX_BOOK; i++){
        snprintf(book_inventory[i].Bookname, sizeof(book_inventory[i].Bookname), "Book %d", i + 1);
        book_inventory[i].Bookstatus = 1;
        total_books++;
    }
    printf("\n* %d books loaded into the inventory *\n", total_books);
}

void load_tables(){
    for (int i = 0; i < MAX_TABLE; i++){
        tables[i].Tablenumber = i + 1;
        tables[i].Tablestatus = 1;
        total_tables++;
    }
    printf("*   %d tables loaded and available    *\n", total_tables);
}
```

The handle_customer() function manages a customer's visit by first searching for an available table, using a **semaphore** and **mutex** to ensure thread-safe access. Once a table is found, it is marked as occupied, and the customer's table and book data are initialized. The customer then borrows up to **3** available books, updating their status to borrowed. After a simulated stay sleep(2), the customer releases the table and returns the books, restoring their availability. The function uses synchronization tools like semaphores and mutexes to handle **concurrent** access to shared resources efficiently and ensure proper customer flow.

```c
void handle_customer(int id){
    printf("Customer %d searching for a table...\n", id);
    while (1){
        sem_wait(&table_semaphore);
        pthread_mutex_lock(&table_mutex);

        int table_found = 0;
        for (int i = 0; i < total_tables; i++){
            if (tables[i].Tablestatus == 1)
            {
                tables[i].Tablestatus = 0;
                customers[id].table = tables[i].Tablenumber;
                customers[id].book_count = 0;
                total_customers++;
                table_found = 1;
                printf("Customer %d assigned to table %d.\n", id, tables[i].Tablenumber);
                break;
            }
        }

        pthread_mutex_unlock(&table_mutex);
        sem_post(&table_semaphore);

        if (table_found)
            break;

        printf("No tables available. Customer %d waiting...\n", id);
        sleep(1);
    }
```

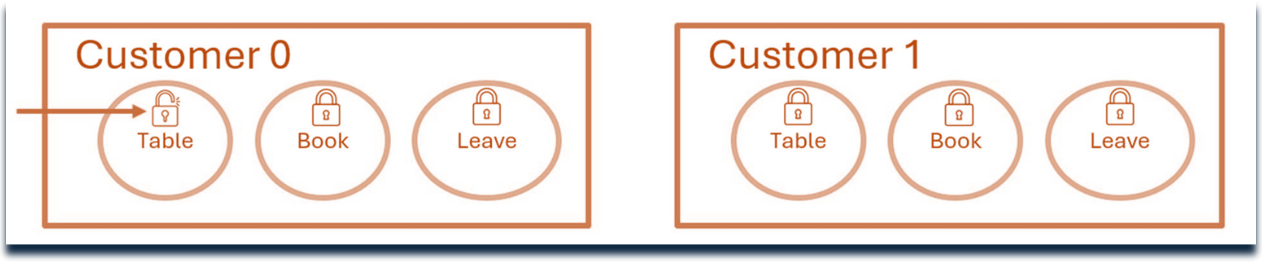
```c
pthread_mutex_lock(&book_mutex);
int books_picked = 0;
for (int i = 0; i < total_books && books_picked < 3; i++){
    if (book_inventory[i].Bookstatus == 1)
    {
        book_inventory[i].Bookstatus = 0;
        customers[id].books[books_picked++] = i;
    }
}
customers[id].book_count = books_picked;
pthread_mutex_unlock(&book_mutex);

printf("Customer %d borrowed %d books and is seated at table %d.\n", id, books_picked, customers[id].table);
sleep(2);

pthread_mutex_lock(&table_mutex);
tables[customers[id].table - 1].Tablestatus = 1;
pthread_mutex_unlock(&table_mutex);

pthread_mutex_lock(&book_mutex);
for (int i = 0; i < books_picked; i++)
{
    book_inventory[customers[id].books[i]].Bookstatus = 1;
}
pthread_mutex_unlock(&book_mutex);

printf("Customer %d has left table %d.\n", id, customers[id].table);
```

The *customer_thread(*void* *arg)function retrieves the customer ID from the argument, frees the memory, and calls handle_customer() to manage the customer's actions. It enables multithreaded handling of customer interactions.

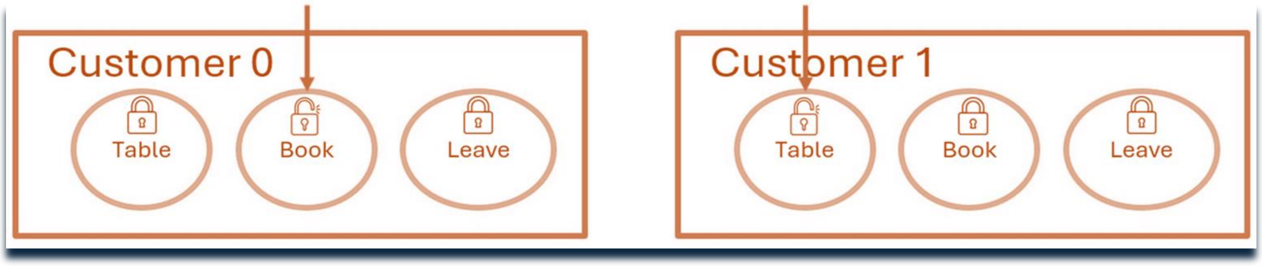
```c
void *customer_thread(void *arg){
    int id = *(int *)arg;
    free(arg);
    handle_customer(id);
    return NULL;
}
```

This below code handles the simulation of customer interactions. It prompts the user to enter the number of customers (n) and creates an array of pthread_t to store thread identifiers. For each customer, it dynamically allocates memory for their ID and starts a thread using pthread_create() to run the customer_thread() function. After all threads are created, it waits for their completion using pthread_join() ensuring all customer interactions finish before proceeding.

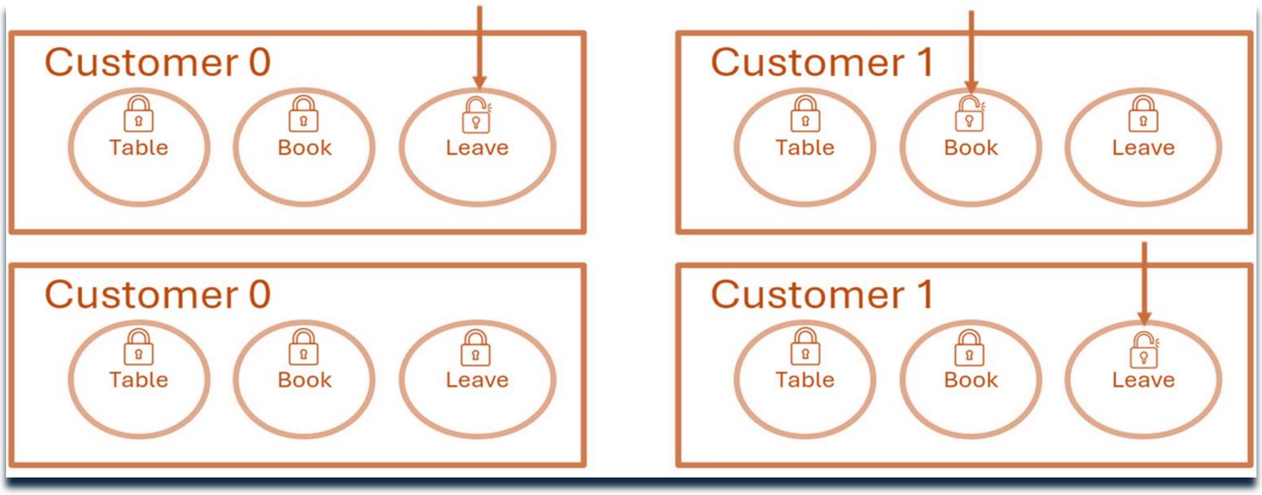
```c
case 1:
{
    int n;
    printf("Simulation will be started\n");
    printf("Enter number of customers : ");
    scanf("%d", &n);
    pthread_t threads[n];
    for (int i = 0; i < n; i++){
        int *id = malloc(sizeof(int));
        *id = i + total_customers;
        pthread_create(&threads[i], NULL, customer_thread, id);
    }
    for (int i = 0; i < n; i++)
    {
        pthread_join(threads[i], NULL);
    }
    break;
}
```



Firstly Customer 0: process with sem_wait and pthread_mutex_lock being used to ensure exclusive access to a table. Customer 0 searches for a free table and occupies it (Tablestatus = 0). Then Customer 1 is blocked, waiting for the semaphore to become available as no free table exists.



Then Once seated, Customer 0 proceeds to borrow books (Bookstatus= 0 for up to 3 books). Mutex ensures thread-safe access to the book inventory then Customer 1 finds a free table and occupies it while Customer 0 is handling books.



Finally, After using the table and books, Customer 0 leaves, releasing the table (Tablestatus = 1) and resetting book statuses (Bookstatus= 1) then Customer 1 proceeds to borrow books in the same synchronized manner after acquiring a table.

Limitation

1) **No Timeout for Waiting :** Customers wait indefinitely for tables, with no timeout mechanism to leave the queue after prolonged waiting.
2) **Concurrency Conflicts :** Insufficient optimization for managing concurrent access to resources (e.g., overlapping book assignments under heavy load).
3) **Unoptimized Queue Management :** The system lacks customer prioritization or termination logic if tables remain unavailable.
4) **Fixed Resource Limits :** The hardcoded number of books and tables restricts scalability. Allowing dynamic resource adjustment could overcome this.
5) **Limited Error Handling :** The program doesn't validate or handle exceptional scenarios like invalid inputs or system resource exhaustion effectively.

Conclusion

Our system provides a fundamental multi-threaded approach to managing tables and books, utilizing semaphores and mutexes effectively. It ensures customers are assigned tables and books and allows them to exit after their allotted time. However, the lack of dynamic interfaces limits the system's potential. Enhancing these aspects could make the simulation more realistic and efficient, aligning it more closely with a real-world café environment.

“THE END”