National University of computer and emerging sciences

Course Name:	Operating System	Course Code:	CS2006
Degree Program:	BS-CS	Semester:	Spring 2023
Exam Duration:	180 minutes	Total Marks:	40
Paper Date:	2-May-2023	Weight	`40
Section:	J	Page(s):	3
Exam Type:	Final		

Instructions:

- **The quality of the code will affect the marks.**
- Students will receive ZERO marks if the answers are plagiarized.
- Use of ANY helping material/code, or cell phones, **INTERNET** and flash drive are strictly prohibited.
- You can use Linux man pages for help.
- You must ensure proper submission of your code following the file naming instructions (given below).
- No queries will be entertained.
- Your submission will contain your **CODE**.
- Submission location: Cactus.

Cactus Link: \\cactus1\Xeon\Spring 2023\OS Lab Spring 2023\Section J\ Saba Tariq
\\cactus1\Xeon\Spring 2023\OS Lab Spring 2023\Section J\ Muhammad Irtaza

File Naming Instructions:

- Name your each individual file as **ROLLNUMBER_QUESTION_FILENAME.c**
- **2 absolute marks will be deducted if the naming instructions are not followed.**

Question 1

20 marks

You are tasked with developing a real-time system for automated order processing in a restaurant. The system consists of multiple processes that need to coordinate their activities to ensure correct and efficient order processing. The system uses multiple threads and synchronization mechanisms to handle customer orders.

The system is designed to handle three customer orders. Each customer can place an order, and once the order is processed, it is delivered to the customer.

The program should contain following **methods:**

1. Order Placement:

- Customers can place orders concurrently (multiple customers can place their orders at the same time).
- Each customer is assigned a unique token number when placing an order.
- Only three concurrent orders can be placed at a time.
- If the maximum number of concurrent orders is reached, subsequent customers should wait until a slot becomes available.
- Once a customer places an order, they should be assigned a token number, starting from 1 and incrementing by 1 for each new order.
- Display the customer ID and the assigned token number when an order is placed.

2. Order Processing:
   - Once an order is placed, it should be processed.
   - Only one order can be processed at a time.
   - If an order is being processed, other orders should wait until the processing is completed.
   - Display the token number when an order starts processing.
3. Order Delivery:
   - Once an order is processed, it should be delivered to the customer.
   - Order delivery can happen concurrently with order processing (Once a order is processed it can be delivered).
   - Display the token number and customer ID when an order is being delivered.

Code Template:

```
Void placeOrder (Void* arg) {
// implement order placing logic here
}
Void processOrder (Void* arg) {
// implement order processing logic here
}
Void deliverOrder (Void* arg) {
// implement order delivery logic here
}
Int main ( ) {
Pthread_t processOrderTh[3], deliverOrderTh[3], placeOrderTh[3];
//create threads
//wait for all threads to finish here
// destroy semaphores here

Return 0;
}
```

char * (
float * (
int * (
)

Scanned with CamScanner

Implement the system using the provided code template, which includes the necessary synchronization mechanisms using the pthread and semaphore libraries.

Your task is to complete the code to satisfy the requirements mentioned above. Ensure that the system operates correctly and efficiently, adhering to proper synchronization.

Instructions:

- Complete the code in the provided template to implement the order placement, order processing, and order delivery logic.
- Use proper synchronization mechanisms, such as semaphores, to ensure correct and efficient operation.
- Test the system to ensure that it satisfies the requirements mentioned above.
- Clearly label and comment your code to demonstrate your understanding.

Question 2 — 20 marks

- Write a C program using the pthread library to perform **matrix multiplication using multiple threads.**
- The program initializes two matrices of size 100x100 with random values between 1 and 10.
- Each thread is responsible for computing the multiplication **for a specific row in the output matrix.**
- The program measures the **execution time** of each thread.
- Each thread should calculate its row **independently** and return the result to the main thread.
- The main thread will collect the results from all threads and store them in the resultant matrix.
- Finally, the program will display the total execution time taken by all threads.
- You can use gettimeofday( ) to calculate execution time.
- However, the threads will have to send their execution time to main thread via **Shared Memory.**
- The main thread then collects these values and adds them up before displaying it.
- Ensure that the matrix multiplication is performed correctly, each thread returns its result to the main thread using **pthread_exit**, and the main thread collects the results using **pthread_join.**

```
struct timeval start_time, end_time;
    gettimeofday(&start_time, NULL);
    // Perform matrix multiplication here
    gettimeofday(&end_time, NULL);
execution_time = (end_time.tv_sec - start_time.tv_sec) * 1000000 + (end_time.tv_usec -
start_time.tv_usec);
```

Once you have completed the code, compile and run it to verify the correctness of your implementation and measure the execution time of the parallel matrix multiplication.

M

--- GOOD LUCK ---