COMP2006 – Operating Systems

Customer Queue Assignment Report

Freddy Khant, 20618166

Due: 14/05/22 23:59 (Extension Granted)

Aim:

The goal of this assignment is to give experiences in using pthreads, including thread creations, synchronizations, and using system calls. You are asked to write a program in C in Linux environment to simulate the operations of a hypothetical customer queue in a bank.

Specifications:

The program implements the following features:

1. A First-in First-out (FIFO) customer queue, called c_queue, where customers queue up waiting for four bank tellers: teller-1, teller-2, teller-3, and teller-4.

2. Arrival of a customer wakes up a waiting teller, which then calls the customer and provides the required services.

3. Three alternative services are provided by each teller: Cash-withdraw, Cash-deposit, and Ask-information.

4. Customers are stored in a file called c_file, where each customer is represented by a customer number and a service type.

5. All activities of the queue and the tellers are recorded in a file named r_log

Implementation:

The program is implemented in C and consists of the following files:

- bank.c: The main source code file containing the implementation of the customer queue simulator.

- makefile: A makefile to compile the program.

Usage:

Refer to README.

Output:

The program generates an output log file named r_log, which contains the activities of the customer queue and tellers. The log file includes the arrival time, response time, completion time, and statistics for each teller.

Libraries:

Mutual Exclusion - The program uses the pthread library to provide mutual exclusion for shared variables and critical sections. Mutex locks (pthread_mutex_lock and pthread_mutex_unlock) are used to protect the customer queue, customer count, teller statistics, and log file operations.

Discussion:

Synchronization and Shared Resources

There are multiple shared resources and variables that need to be accessed and modified by multiple threads. To ensure proper synchronization and avoid race conditions, the program uses mutexes and condition variables.

1. Customer Queue: Represented by the c_queue array, the queue accessed by both the customer and teller threads. To prevent simultaneous access to the queue, queue_mutex is used. The customer thread acquires the queue_mutex before enqueuing a customer, and the teller thread acquires the queue_mutex before dequeuing a customer. This ensures that only one thread can modify the queue at a time.

2. Log File: Represented by the r_log file pointer, is accessed by both the customer and teller threads when logging customer information. To synchronize access to the log file, log_mutex is used. Each thread acquires the log_mutex before opening the log file, writing to it, and closing it. This ensures that the log entries from different threads are properly serialized and avoid conflicts.

3. Condition Variable: The condition variable queue_cond is used to signal when the customer queue is NOT empty. The teller threads wait on queue_cond when the queue is empty and there are no customers to serve. The customer thread signals queue_cond whenever it enqueues a customer. This mechanism allows the teller threads to sleep efficiently when there are no customers and wake up when new customers arrive.

Testing

1. Case 1: Small Input
   Input: m = 5, tc = 1, tw = 2, td = 3, ti = 4
   Output: The program runs without errors and outputs all customer arrivals, teller processing times and information correctly on the log file but is missing the Teller Statistics at the end.

2. Case 1: Large Input
   Input: m = 100, tc = 2, tw = 3, td = 4, ti = 5
   Output: The program runs without errors and outputs all customer arrivals, teller processing times and information correctly on the log file but is missing the Teller Statistics at the end.

3. Case 3: Stress Test
   Input: m = 100, tc = 1, tw = 1, td = 1, ti = 1
   Output: A stress test with minimal service times. The program handles this satisfactorily, running fine until reaching the 50-60 customer range, the final teller terminates, and customers arrive with nowhere to go. This is ultimately where it fails.