COMP2006 - Operating Systems

CURTIN UNIVERSITY
School of Electrical Engineering, Computing and Mathematical Sciences
Discipline of Computing

Customer Queue

Due Date: **4.00 pm Monday 8th May 2023**

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.

A. Specification

The program for this customer queue simulator includes the following features. Read completely before writing your program / assignment.

1. There is 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**. Assume that initially the queue is empty, and all the four tellers are waiting for customers. Let ***m*** (in integer) be the size / length of **c_queue**.

2. When there is no customer in the queue, each teller *sleeps* / *blocks*. Arrival of a customer (to **c_queue**) will wakeup / un-block one waiting teller, which will call the customer, and provide required services. Thus, this step simulates the producer consumer problem with a bounded buffer.

3. Three alternative services are provided by each teller:
   - Cash-withdraw, which requires t_w seconds to service,
   - Cash-deposit, which requires t_d seconds to service, and
   - Ask-information, which requires t_i seconds to service.

   Assume that all tellers have the same speed for each of the service types.

4. Since a teller needs time to provide service, you must simulate this event. This can be done, for example, by using a **sleep()** system call. Note that this assignment does not require an accurate time measurement.

The need of point 6 is such that there is at least one teller that can serve any remaining customers.

To simplify the problem, you are allowed to change the following new requirement in point 6: "All tellers terminate when all customers have been placed in the queue and have been served." Thus, you can use either the original requirement in point 6 or this new requirement for point 6.

5. A customer list is stored in a file called **c_file**. A customer in the file is represented as:

   customer# service_type

   where the *customer#* is a positive integer, and the *service_type* is either **W** (for withdrawal), **D** (for deposit), or **I** (for a customer that needs information). For example, **c_file** may contain the following (*customer#* and *service_type* is separated by a space):

   1 W
   2 W
   3 I
   4 D
   5 I

   Please create your own **c_file** with a reasonable number of customers, e.g., 100.

6. Each **teller *k*** (except the last one) terminates when the queue is empty. However, one of four tellers, i.e., **the last teller**, terminates only when there is no longer incoming customer (NOT when the queue is empty).

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

B. Implementation

1. Write a function **customer()** that periodically (every t_c seconds) gets a customer from the **c_file** and puts the information in **c_queue**. Create one thread that runs function **customer()**.

2. After putting a customer in the queue, function **customer()** writes this activity into file **r_log** in the following format:

   Customer#: *service type*
   Arrival time: 13:42:51
   ---

   where customer# is the customer number, *service type* can be either **W**, **D** or **I**, and the arrival time is the time when the customer is placed into the queue (actual time). Use a system call to get the time. Note that this assignment does not require an accurate time.

3. Write a function **teller()** that simulates the operations of each teller. When there is at least one customer in the queue, the teller takes one customer from the queue, and serves the customer. Create four threads for **teller-1**, **teller-2**, **teller-3**, and **teller-4,** each of which runs function **teller()**.

4. Use pthread mutual exclusion functions, i.e., pthread_mutex_lock(), pthread_mutex_unlock(), pthread_cond_wait(), and pthread_cond_signal(), to protect any shared variables. In your report, you must describe / discuss in detail each shared variable, including its data structure, the threads that access them, and how mutual exclusion is achieved. Remember to clean up all resources created in your program.

5. When a teller (for example **teller-1**) takes one customer (for example customer number 3) from the queue, the teller writes the following information to file **r_log**:

   Teller: 1
   Customer: 3
   Arrival time: 13:42:55
   Response time: 13:42:57

   The response time is the time when the teller picked up the customer from the queue. Note that this assignment does not require an accurate time.

6. When a teller (for example **teller-1**) finishes with one customer (for example customer 3), the teller writes the following information in file **r_log**:

   Teller: 1
   Customer: 3
   Arrival time: 13:42:55
   Completion time: 13:42:57

   The completion time is the time when the teller finished servicing the customer. Note that this assignment does not require an accurate time.

7. On termination, a teller (for example **teller-1)**, writes the following information in file **r_log**:

   Termination: teller-1
   #served customers: n_1
   Start time: 13:42:51
   Termination time: 13:42:57

   The start time is the time **teller-1** is created, the termination time is the time the teller terminates, and n_1 is the total number of customers who have been served by **teller-1**. Note that this assignment does not require an accurate time.

8. **The last teller** that terminates writes the following information:

   Teller Statistic
   Teller-1 serves n_1 customers.
   Teller-2 serves n_2 customers.
   Teller-3 serves n_3 customers.
   Teller-4 serves n_4 customers.

   Total number of customers: *n* customers.

   Note:
   (i) n_1, n_2, n_3, and *n* are respectively the number of customers that have been served by **teller-1**, **teller-2**, **teller-3**, and **teller-4**, and
   (ii) *n* is the total customers that have been served by the tellers

9. YOU MUST FOLLOW THIS OUTPUT FORMAT.

10. The assignment does not require precision involving the measurement of time.

11. To test for the correctness of your program, you should run the program as follows:

   cq $\boldsymbol{m}$ $\boldsymbol{t_c}$ $\boldsymbol{t_w}$ $\boldsymbol{t_d}$ $\boldsymbol{t_i}$

   where (i) **cq** is the file name of your executable program, (ii) $\boldsymbol{m}$ is the size / length of customer queue (**c_queue**), (iii) $\boldsymbol{t_c}$ is the periodic time for the customer to arrive in the queue, and (iv) $\boldsymbol{t_w}$, $\boldsymbol{t_d}$, and $\boldsymbol{t_i}$ respectively represent the time duration to serve **w**ithdrawal, **d**eposit, and **i**nformation. Each variable is a positive integer.

C. Instruction for submission

1. Assignment submission is **compulsory**. As stated in the unit outline, the penalty for late submission is calculated as follows:
   - For assessment items submitted within the first 24 hours after the due date/time, students will be penalised by a deduction of 5% of the total marks allocated for the assessment task;
   - For each additional 24 hour period commenced an additional penalty of 10% of the total marks allocated for the assessment item will be deducted; and
   - Assessment items submitted more than 168 hours late (7 calendar days) will receive a mark of zero.

2. Please read and follow the **Academic Integrity (including plagiarism and cheating)** section in the unit outline to prevent any possible academic misconduct.

3. You must (i) submit the soft copy of the report to the unit Blackboard (**in one zip file**), and (ii) put your program files, i.e., **cq.c**, and other files, e.g., makefile, **c_file** and **r_log**, in your home directory named **OS/assignment**.

4. Your assignment **report** should include:

   - A signed **Declaration of Originality** form (available in the unit Blackboard). Please read the form carefully before you sign it. Your name should be as recorded in the student database.

   - Software solution of your assignment that includes (i) all source code for the programs with proper in-line and header documentation. Use proper indentation so that your code can be easily read. Make sure that you use meaningful variable names and delete all unnecessary comments that you created while debugging your program; and (ii) readme file that, among others, explains **how to compile your program and how to run the program**.

   - Detailed discussion on how synchronization is achieved when accessing shared resources / variables and which threads access the shared resources.

   - Description of any cases for which your program is not working correctly or how you test your program that make you believe it works perfectly.

   - Sample inputs and outputs from your running programs. Explain if the outputs are correct / incorrect.

   Your report will be assessed (worth 20% of the overall assignment mark).

5. You are required to demonstrate your working program. The demo time will be announced later. Your program must run on a computer in our Linux lab, e.g., lab 232 in building 342.

Failure to meet these requirements may result in the assignment not being marked.