University of Dhaka
Department of Computer Science & Engineering

CSE 3211 : Operating Systems Lab

Assignment 01 : Investor Producer Synchronization Problem

Group Members:

- Shamim Bin Zahid
  Roll: SH-**43**
- Aishwarya Ghosh Bristy
  Roll: SK-**50**

Submitted to:

Dr. Md Mamun-Or-Rashid.
Professor, Department of Computer Science and Engineering,
University of Dhaka

Table of Contents

Problem Statement

The problem at hand is the investor-producer synchronization. We need to use semaphores to handle the critical regions so that concurrent processes can be executed smoothly. A customer orders products from the producer. The producer receives the order and accordingly takes a loan from the bank and supplies the customer with the ordered products. Customers make the payment after receiving their orders. And after that the producer has to return the loan back to the bank with some additional fees. The main problem to solve here is that there are a number of concurrent processes running, and we need to ensure that none of the threads become stuck in a race condition or deadlock.

Definitions

- **Race Condition:**
  A race condition occurs when two threads access a shared variable at the same time. The first thread reads the variable, and the second thread reads the same value from the variable.
- **Deadlock:**
  A deadlock is caused when two or more threads come into conflict over some resource, in such a way that no execution is possible.
- **Semaphore:**
  A semaphore is a variable or abstract data type used to control access to a common resource by multiple processes and avoid critical section problems in concurrent processes. There are two operations in a semaphore, P( ) to decrease the value (denoting to wait) and V( ) to increase the value (denoting to signal to continue)
- **Critical Region:**
  When a segment has a variable / chunk of code that is shared across multiple concurrent processes, it is called a critical region. We need to make sure every process has to run the "entry" section to get access to some resource and must run an "exit" section when it is done.

The Components of the System

Customer Thread:

order item()

This function takes an item as input. After taking the item we add the new item to our linked list and increase the value of the total item by 1. We used semaphore sem_queue to make sure that no other process is working with the item list at that moment.

consume_item()

This function takes customerNumber as an input. We check the req_serv_item list first. After that we try to see if any items are requested by this customer. If the order has already been SERVICED we just add the money for the correspondent customer in our customer_spending_amount[customerNumber]. There are two shared entities used in this function. One is the customer_spending_amount[] array and another one is req_serv_item.

end_shopping()

At the end of shopping, this function is executed. Before going home every customer had a variable (stoppedCustomer) which we maintained to see whether all customers went home or not.

Producer Thread:

take_order():

This function waits for customers to order. Here each producer takes one third of the items at a time and changes them to the form requested.

produce_item():

This function produces an item. Mostly the producer sets a price in i_price variable according to adjusting the profit and bank loan.

serve_order():

This function is pretty self explanatory. Producer serves the ordered item to the customer. After serving, the order_type attribute is set to SERVICED.

calculate_loan_amount():

This function helps the producer to calculate the bank loan. After deciding the prices of the items in the producer_item() function, we calculate and return the total loan here by simply multiplying the item_quantity and item_price.

loan_request():
This function takes two parameters. The total loan(amount) that is being requested and the producer(producernumber) who is requesting it. Then, it selects a random bank to request the loan from.

loan_reimburse():
The producer has to pay his debt to the bank at the end. This function calculates the total amount of money the producer has to give back(with additional charge). Each producer repays the loan as it took to produce the item and update bank account variables. It also updates producer income with producer profit.

initialize():
We initialize the important variables that are used all around the program here. We also create the semaphores with appropriate values.

finish():
We destroy the semaphores and free the used memory after everything is done

Identifying Critical Regions & Using Semaphores

- In the order_item() function, the code segment which inserts into the req_serv_item linked list is a critical region. When more than one customer tries to insert itself into the linked list it will create a concurrency problem. So before getting inserted into the linked list we use P() to lock the critical region and after inserting N_item_type we use V() to signal the release of the critical region. Also, in the consume_item() function it loops through the req_serv_item to see which orders are served, which is another critical region itself. In both cases, the same semaphore was used.

```
mutex = sem_create("mutex", 1);
```

- In the take_order() function, No two producers can produce products for an order at the same item. Which may form a critical region. P( ) was used before executing the req_serv_item linked list to get the requested item and after getting the requested item we used V() to release the critical region.

```
TO_empty = sem_create("TO_empty",1);
```

- In the take_order() function each producer should wait for an order to be placed by a customer. To handle this, a semaphore was used which is initialized to 0. We used P( ) right after entering the take_order() function and V() right after the customer places an order in the order_item() function. Also, when a customer ends shopping it also signals the take_order() function using V() so that producer thread doesn't end up in deadlock.

```
TO_full = sem_create("TO_full",0);
```

- A semaphore variable stopped_customer was used which was initialized with N_CUSTOMER. Everytime a customer places an order it decreases the value of the stopped_customer by 1 so that the producer thread can know if a customer has ended the shopping or not.

```
CLA = sem_create("CLA",1);
```

- Two critical regions are formed in the loan_request() and loan_reimburse() function we need to access the bank_account array. Which was solved using another semaphore.

```
WFP = sem_create("WFP",1);
```

- A semaphore variable service_arr[ ] where i'th index of the array denotes how many orders of i'th customer has been served. As soon as a producer serves an order in the serve_order() function, the service_arr's index for that customer increments by 1 solving the critical region with the semaphore.

```
mutex3 = sem_create("mutex3",1);
```

- A customer has to wait until an active order has finished before they can create a new one. To handle this critical region, we used an array of semaphores CONSUME_ITEM_SEM where the i'th semaphore is used for obviously the i'th customer. The whole semaphores array is initialized with 0s. As soon as a customer goes through the consume_item() function it waits for the serve_order() function to signa it. The serve_order() function signals customers if their index in the service_arr is equal to N_ITEM_TYPE.

```
for (int i = 0; i < NCUSTOMER; i++) {
    CONSUME_ITEM_SEM[i] =sem_create("CONSUME_ITEM_SEM",0);
}
```