College of Engineering, Trivandrum

Network Lab Exam Report - 1-A

Divya Moncy

S6 CSE Roll no. 27

University Roll no. TVE16CS028

25 April 2019

Contents

1 Problem Statement 1-A

Cold coffee : The espresso franchise in the strip mall near my house serves customers FIFO in the following way. Each customer entering the shop takes a single “ticket” with a number from a “sequencer” on the counter. The ticket numbers dispensed by the sequencer are guaranteed to be unique and sequentially increasing. When a barrista is ready to serve the next customer, it calls out the “eventcount”, the lowest unserved number previously dispensed by the sequencer. Each customer waits until the eventcount reaches the number on its ticket. Each barrista waits until the customer with the ticket makes an order. Show how to implement sequencers, tickets, and event counts using mutexes/semaphores and condition variables. Your solution should also include code for the barrista and customer threads.

2 Aim

To write a program simulating the customer and barrista threads, implement sequencers, tickets and eventcounts and to synchronise the threads using semaphores/mutexes

3 Theory

3.1 Synchronisation

Process Synchronization means sharing system resources by processes in a such a way that, Concurrent access to shared data is handled thereby minimizing the chance of inconsistent data. Maintaining data consistency demands mechanisms to ensure synchronized execution of cooperating processes.

3.2 Critical Section

A Critical Section is a code segment that accesses shared variables and has to be executed as an atomic action. It means that in a group of cooperating processes, at a given point of time, only one process must be executing its critical section. If any other process also wants to execute its critical section, it must wait until the first one finishes.

3.3 Semaphores

Semaphores are integer variables that are used to solve the critical section problem by using two atomic operations, wait and signal that are used for process synchronization.

3.4 Multithreading

Multithreading is a type of execution model that allows multiple threads to exist within the context of a process such that they execute independently but share their process resources. A thread

maintains a list of information relevant to its execution including the priority schedule, exception handlers, a set of CPU registers, and stack state in the address space of its hosting process.

4 Implementation

There are three shared resources:
custcount - to keep track of the number of customer
seqcount - to keep track of the sequencer number
eventcount - to keep track of the ticket number that was last served by the barrista
There are three semaphores:
m1 - to protect custcount
m2 - to protect eventcount
cust - to notify the barrista that there is a customer

4.1 Customer Thread

In the customer thread, it has a local variable ticketno. It waits to acquire m1, increments custcount and if it is one, it releases cust. ticketnumber is assigned the incremented value of seqcount and then m1 is released. It waits in a loop while eventcount not equal to ticketno. Then after the order has been served, it exits, decrementing custcount and releasing cust.

4.2 Barrista Thread

It works in an infinite loop. It first waits for a customer to arrive. Then it acquires m2, increments eventcount and if eventcount is greater than seqcount, it breaks out of the loop. Else customer is served and m2 is released.

4.3 Compilation and Execution

gcc barrista.c -lpthread
./a.out

5 Program

barrista.c

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <stdlib.h>

int custcount=0,seqcount=0,eventcount=0;
```

```
sem_t m1,m2,cust;
/*  custcount – to keep customer count
    seqcount – to keep sequencer value
    eventcount – to keep event counter value
    m1 – to protect custcount
    m2 – to protect eventcount
    cust – to show there is a customer */


void *customer(void *args)
{
    int f=(int)(args);
    int ticketno;
    sem_wait(&m1);//waiting for semaphore to update custcount
    custcount++;
    if(custcount==1)
       sem_post(&cust);
    ticketno=++seqcount; //giving ticketcount
    printf("Customer with ticketno %d placed order\n",ticketno);
    sem_post(&m1); // releasing m1
    while(eventcount!=ticketno); // waiting till his event number
    is called out
    sem_wait(&m1); // waiting for m1
    custcount--; // decrementing customer count
    printf("Customer with ticketno %d leaves\n",ticketno);
    sem_post(&cust); // releasing customer
    sem_post(&m1); // releasing m1
}
void *barrista(void *args)
{
    while(1)
    {
       sem_wait(&cust); //waiting for customer
       sem_wait(&m2); // waiting for m2
       eventcount++; //updating event count
       if(eventcount>seqcount) // exit condition
          break;
       printf("Customer with ticketno %d is served\n",eventcount);
       sem_post(&m2); // releasing m2

    }

}

int main()
```

```
{
      pthread_t cust[5],barr;
      sem_init(&m1,0,1);
      sem_init(&m2,0,1);
      sem_init(&cust,0,1);
      int i;
      for(i=0;i<5;i++)
        pthread_create(&cust[i],NULL,customer,(void *)0); //creating threads
      pthread_create(&barr,NULL,barrista,(void *)0);
      for(i=0;i<5;i++)
        pthread_join(cust[i],NULL); //joining threads
      pthread_join(barr,NULL);
      return 0;
}
```

6 Output

6.1 Test Run 1

```
adminisrtator@pc-16:~$ ./a.out
Customer with ticketno 1 placed order
Customer with ticketno 2 placed order
Customer with ticketno 3 placed order
Customer with ticketno 1 is served
Customer with ticketno 4 placed order
Customer with ticketno 5 placed order
Customer with ticketno 1 leaves
Customer with ticketno 2 is served
Customer with ticketno 2 leaves
Customer with ticketno 3 is served
Customer with ticketno 3 leaves
Customer with ticketno 4 is served
Customer with ticketno 4 leaves
Customer with ticketno 5 is served
Customer with ticketno 5 leaves
```

6.2 Test Run 2

```
adminisrtator@pc-16:~$ ./a.out
Customer with ticketno 1 placed order
Customer with ticketno 2 placed order
Customer with ticketno 3 placed order
Customer with ticketno 4 placed order
Customer with ticketno 5 placed order
```

```
Customer with ticketno 1 is served
Customer with ticketno 1 leaves
Customer with ticketno 2 is served
Customer with ticketno 2 leaves
Customer with ticketno 3 is served
Customer with ticketno 3 leaves
Customer with ticketno 4 is served
Customer with ticketno 4 leaves
Customer with ticketno 5 is served
Customer with ticketno 5 leaves
```

6.3 Test Run 3

```
adminisrtator@pc-16:~$ ./a.out
Customer with ticketno 1 placed order
Customer with ticketno 2 placed order
Customer with ticketno 3 placed order
Customer with ticketno 4 placed order
Customer with ticketno 1 is served
Customer with ticketno 1 leaves
Customer with ticketno 5 placed order
Customer with ticketno 2 is served
Customer with ticketno 2 leaves
Customer with ticketno 3 is served
Customer with ticketno 3 leaves
Customer with ticketno 4 is served
Customer with ticketno 4 leaves
Customer with ticketno 5 is served
Customer with ticketno 5 leaves
```

7 Result

The program to simulate the barrista and customer threads were written in C, compiled using gcc and run on Ubuntu 16.04 kernel successfully. Multiple threads were created to simulate the barrista and the customers. The output was verfified.