COLLEGE OF ENGINEERING TRIVANDRUM

NETWORKING PROGRAMMING LAB

Lab Exam Report - Q2A

NIKHIL JOJI
Roll No: 46
TVE16CS047

25 April 2019

Contents

1 Problem 1 **3**
1.1 Problem Statement 3

2 Theory **3**
2.1 Synchronisation 3
2.2 Mutex 3
2.3 Threads 3

3 Implementation **3**

4 Program **4**

5 Output **6**
5.1 Test Run 1 6
5.2 Test Run 2 6
5.3 Test Run 3 7

6 Result **7**

1 Problem 1

1.1 Problem Statement

Three engineering professors have gone to the faculty club to eat curdrice.Each plate of curdrice costs 35. To buy a plate of curdrice, a professor needs one 20, one 10, and one 5. The first professor has a pocket full of 5, the second has 10, and the third a supply of 20. A wealthy alumni walks up to the restaurant and lays down at random two of the three coins needed for a plate of curdrice. The professor with the third coin takes the money and buys the curdrice. The cycle then repeats. Show how to synchronize the professors and the alumni.

2 Theory

2.1 Synchronisation

The problem of Synchronisation is a standard problem in Operating System concepts. When there are multiple processes accessing a particular section of the program called the Critical section, there is a chance that mismatch of information may result (Inconsistent values for certain variables etc). To bypass such inconsistency while running multiple processes simultaneously, we use Synchronisation.

2.2 Mutex

Mutex or Mutual Exclusion Object in programming is a lock like mechanism implemented so that we can bring about synchronisation while programming using multiple processes. When a thread requires a particular resource, it acquired the mutex and locks it, preventing any other thread from accessing it. When another thread comes to acquire the same resource, the mutex cannot be obtained therefore is prevented from accessing the resource.

2.3 Threads

A thread is a path of execution within a process (program). There can be more than one threads in a process. Threads of a process share memory inside the process, unlike between multiple processes. Threads run simultaneously in memory.

3 Implementation

The program was written in c++ using the standard libraries. The main function creates 3 threads that simultaneously try to let the professor eat curdrice. These threads run the function eat(). Here the 3 cases that the alumni can donate money is indexed at 0,1,2. A random number is generated and if it is 0, we proceed as Alumni has donated 5 and 10 coins. If the random number generated is 1, we proceed as Alumni has donated 10,20.

If the random number generated is 2, we proceed as Alumni has donated 5,20. The random number is generated after the alock mutex is acquired and then accordingly the professor eat critical section is executed, after acquiring the mutex m. 3 threads execute the same function repeatedly with Synchronisation.
The program is compiled using g++ compiler along with the option -pthread as we're using threads. Since the library function as best supported in c++11 with also use the option -std=c++11. Therefore to run the program named Q2A.cpp:

```
g++ Q2A.cpp -pthread -std=c++11
```

is the command that is used. Then ./a.out is used to run the compiled code.

4 Program

```
#include<iostream>
#include<mutex>
#include<thread>
#include<stdlib.h>
#include<unistd.h>
#include<time.h>

using namespace std;

mutex m,alock;
int a;

//if a is 0 then 5,10
//if a is 1 then 10,20
//if a is 2 then 5,20

void eat()
{
        while(1)
        {

                while(!alock.try_lock()); //lock acquired to protect
                                          //random variable a

                a = rand();
                a = (a%10)/4;

                if (a == 0)
                {
                        while(!m.try_lock());   //lock protecting
```

```
                                                    //eating section for
                                                    //prof with 20
                        cout<<"Professor with 20 is eating."<<endl;
                        //sleep(1);
                        cout<<"Professor with 20 finished eating."<<endl;
                        //sleep(1);
                        m.unlock();
                }
                if (a == 1)
                {
                        while(!m.try_lock());   //lock protecting eating
                                                //section for prof with 5
                        cout<<"Professor with 5 is eating."<<endl;
                        //sleep(1);
                        cout<<"Professor with 5 finished eating."<<endl;
                        //sleep(1);
                        m.unlock();
                }
                if (a == 2)
                {
                        while(!m.try_lock());   //lock protecting
                                                //eating section for prof
                                                //with 10
                        cout<<"Professor with 10 is eating."<<endl;
                        //sleep(1);
                        cout<<"Professor with 10 finished eating."<<endl;
                        //sleep(1);
                        m.unlock();
                }
                alock.unlock();  //lock for variable a released

        }
}

int main()
{
        thread e1(eat);  //3 threads that run simultaneously
        thread e2(eat);
        thread e3(eat);
        exit(0);
}
```

5 Output

5.1 Test Run 1

```
Professor with 20 is eating.
Professor with 20 finished eating.
Professor with 5 is eating.
Professor with 5 finished eating.
Professor with 5 is eating.
Professor with 5 finished eating.
Professor with 5 is eating.
Professor with 5 finished eating.
```

5.2 Test Run 2

```
Professor with 20 is eating.
Professor with 20 finished eating.
Professor with 5 is eating.
Professor with 5 finished eating.
Professor with 5 is eating.
Professor with 5 finished eating.
Professor with 20 is eating.
Professor with 20 finished eating.
Professor with 10 is eating.
Professor with 10 finished eating.
```

5.3 Test Run 3

```
Professor with 20 is eating.
Professor with 20 finished eating.
Professor with 20 is eating.
Professor with 20 finished eating.
Professor with 5 is eating.
Professor with 5 finished eating.
Professor with 20 is eating.
Professor with 20 finished eating.
Professor with 10 is eating.
Professor with 10 finished eating.
Professor with 20 is eating.
Professor with 20 finished eating.
Professor with 5 is eating.
Professor with 5 finished eating.
Professor with 20 is eating.
Professor with 20 finished eating.
Professor with 5 is eating.
Professor with 5 finished eating.
Professor with 20 is eating.
Professor with 20 finished eating.
Professor with 5 is eating.
Professor with 5 finished eating.
Professor with 20 is eating.
Professor with 20 finished eating.
```

6 Result

The synchronisation problem has been solved using mutex and threads. The program was implemented in c++ on an Ubuntu 16.04 LTS system and compiled using g++ compiler.