OS LAB 4

NAME: Aditya Anand

ROLL NO.: 20124009

BRANCH: IT

S NO.	TITLE	DATE OF IMPLEMENTATION	REMARKS
1	Program to handle the Critical Section Problem using semaphore	09-02-2022	

CRITICAL SECTION PROBLEM SOLUTION USING SEMAPHORE

The Critical Section is the part of the program where shared resources are accessed by multiple processes.

When two or more process try to access the critical section at the same time, errors such as race condition may arise. This is known as the **critical section problem**.

Semaphore is an integer variable which is used in mutually exclusive manner by various concurrent cooperative processes in order to achieve process synchronization.

CODE:

```
#include<bits/stdc++.h>
using namespace :: std;

queue<int> ready;

class semaphore{
    int value;
    public:
        queue<int> blocked_list, CS;

        semaphore(){
            this->value = 9; // Last digit of my roll number.
        }

        void down(int p){         // Entry code before entering Critical Section
            this->value--;
            if(this->value<0){
                // Put the process in the blocked list
                blocked_list.push(p);
                cout<<"Process "<<p<<" has been put in the blocked list\n";
            }
            else{
                // Put the process in the Critical Section
                CS.push(p);
                cout<<"Process "<<p<<" has entered the Critical Section\n";
            }
        }

        void up(){
            this->value++;
            if(!CS.empty()){
                int p=CS.front();
                cout<<"Process "<<p<<" has exited the Critical Section\n";
                CS.pop();
            }
            if(this->value<=0){
                // Wake up a sleeping process from the blocked list
                int p=blocked_list.front();
                cout<<"Process "<<p<<" has been removed from the blocked list\n";
                blocked_list.pop();
                // Put the blocked process back in the ready queue
```

```
                ready.push(p);
            }
        }

        void inCS(){
            if(CS.empty()){
                cout<<"The Critical Section is empty\n";
            }
            else{
                queue<int> temp = CS;
                cout<<"Processes present in Critical Section are: ";
                while(!temp.empty()){
                    cout<<temp.front()<<" ";
                    temp.pop();
                }
                cout<<"\n";
            }
        }

        void inBlockedList(){
            if(blocked_list.empty()){
                cout<<"The Blocked List is empty\n";
            }
            else{
                queue<int> temp = blocked_list;
                cout<<"Processes present in Blocked List are: ";
                while(!temp.empty()){
                    cout<<temp.front()<<" ";
                    temp.pop();
                }
                cout<<"\n";
            }
        }
};

int main(){

    cout<<"PROCESS SYNCHRONISATION USING SEMAPHORE C++ IMPLEMENTATION\n";
    cout<<"Name: Aditya Anand\tRoll No.:20124009\t Branch: IT\n\n\n";

    int n=0;
    cout<<"Enter number of processes: ";
    cin>>n;
    for(int i=1; i<=n; i++){
        ready.push(i);
    }

    semaphore S;

    int itr=0;

    while(!ready.empty()){
        itr++;
        while(!ready.empty()){
```

```
            int process = ready.front();
            ready.pop();
            S.down(process);       // Each process tries to enter the critical section
            if(itr!=1){
                break;
            }
        }

        S.inCS();
        cout<<"\n";
        S.inBlockedList();
        cout<<"\n\n";

        while(!S.CS.empty()){
            S.up();
        }
    }

    return 0;
}
```

RESULT:

```
PS C:\Users\beadi\Desktop\OS LAB\Assignment 4> cd "c:\Users\beadi\Desktop\OS LAB\Assignment 4\" ;
PROCESS SYNCHRONISATION USING SEMAPHORE C++ IMPLEMENTATION
Name: Aditya Anand      Roll No.:20124009       Branch: IT


Enter number of processes: 10
Process 1 has entered the Critical Section
Process 2 has entered the Critical Section
Process 3 has entered the Critical Section
Process 4 has entered the Critical Section
Process 5 has entered the Critical Section
Process 6 has entered the Critical Section
Process 7 has entered the Critical Section
Process 8 has entered the Critical Section
Process 9 has entered the Critical Section
Process 10 has been put in the blocked list
Processes present in Critical Section are: 1 2 3 4 5 6 7 8 9

Processes present in Blocked List are: 10


Process 1 has exited the Critical Section
Process 10 has been removed from the blocked list
Process 2 has exited the Critical Section
Process 3 has exited the Critical Section
Process 4 has exited the Critical Section
Process 5 has exited the Critical Section
Process 6 has exited the Critical Section
Process 7 has exited the Critical Section
Process 8 has exited the Critical Section
Process 9 has exited the Critical Section
Process 10 has entered the Critical Section
Processes present in Critical Section are: 10

The Blocked List is empty


Process 10 has exited the Critical Section
```