OS LAB 8

NAME: Aditya Anand

ROLL NO.: 20124009

BRANCH: IT

S NO.	TITLE	DATE OF IMPLEMENTATION	REMARKS
1	Implementation of solution of Dining Philosophers Problem	28-03-2022	

IMPLEMENTATION OF SOLUTION OF DINING PHILOSOPHERS PROBLEM

The Dining Philosopher Problem:

The Dining Philosopher Problem states that K philosophers seated around a circular table with one chopstick between each pair of philosophers. There is one chopstick between each philosopher. A philosopher may eat if he can pick up the two chopsticks adjacent to him. One chopstick may be picked up by any one of its adjacent followers but not both.

CODE:

```
// C++ implementation of solution of Dining Philosopher Problem using Semaphore (Process Synchronisation)

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

queue<int> ready;

struct binarySemaphore{
    bool s;

    bool down(){
       if(!s){
           return false;
       }

       s=0;

       return true;
    }

    bool up(){
        if(s){
            return false;
        }

        s=1;
        return true;
    }
};

struct countingSemaphore{
    int s;

    bool down(){
       if(s==0){
           return false;
       }

       s--;

       return true;
```

```
    }

    bool up(){
        if(s==5){
            return false;
        }

        s++;
        return true;
    }
};

void philosopherEat(int phil, vector<binarySemaphore> &fork, queue<int> &temp2){
    cout<<"Philosopher "<<phil<<" is hungry\n";
    if(fork[(phil-1)%5].down()){
        if(fork[(phil)%5].down()){
            cout<<"Philosopher "<<phil<<" has started eating\n";
            temp2.push(phil);
        }
        else{
            fork[(phil-1)%5].up();
            cout<<"Forks are not available for Philosopher "<<phil<<" to use\n";
            ready.push(phil);
            return;
        }
    }
    else{
        cout<<"Forks are not available for Philosopher "<<phil<<" to use\n";
        ready.push(phil);
    }
}

void philosopherFinish(vector<binarySemaphore> &fork, queue<int> &temp2){
    while(!temp2.empty()){
        int phil = temp2.front();
        temp2.pop();

        cout<<"Philosopher "<<phil<<" has finished eating\n";
        fork[(phil-1)%5].up();
        fork[(phil)%5].up();
    }
}

void DiningPhilosopher(){
    vector<binarySemaphore> fork(5);

    // Initially all the forks are available
    for(int i=0; i<5; i++){
        fork[i].s = true;
    }

    while(true){
```

```
        if(ready.empty()){
            break;
        }

        queue<int> temp1, temp2;

        while(!ready.empty()){
            temp1.push(ready.front());
            // temp2.push(ready.front());
            ready.pop();
        }

        while(!temp1.empty()){
            int phil = temp1.front();
            temp1.pop();

            philosopherEat(phil, fork, temp2);
        }

        while(!temp2.empty()){
            philosopherFinish(fork, temp2);
        }
    }
}

int main(){
    cout<<"SOLUTION TO DINING PHILOSOPHERS PROBLEM USING SEMAPHORE C++ IMPLEMENTATION\n";
    cout<<"Name: Aditya Anand\tRoll No.:20124009\t Branch: IT\n\n\n";

    cout<<"There are 5 philosophers(numbered 1-5) sitting on a round table and 5 forks.\n";
    cout<<"Enter the order in which Philosphers get hungry: ";

    for(int i=0; i<5; i++){
        int n=0;
        cin>>n;
        ready.push(n);
    }

    DiningPhilosopher();

    return 0;
}
```

RESULT:

```
PS C:\Users\beadi\Desktop\OS LAB> cd "c:\Users\beadi\Desktop\OS LAB\Assignment 8\" ;
SOLUTION TO DINING PHILOSOPHERS PROBLEM USING SEMAPHORE C++ IMPLEMENTATION
Name: Aditya Anand      Roll No.:20124009       Branch: IT


There are 5 philosophers(numbered 1-5) sitting on a round table and 5 forks.
Enter the order in which Philosphers get hungry: 4 2 3 1 5
Philosopher 4 is hungry
Philosopher 4 has started eating
Philosopher 2 is hungry
Philosopher 2 has started eating
Philosopher 3 is hungry
Forks are not available for Philosopher 3 to use
Philosopher 1 is hungry
Forks are not available for Philosopher 1 to use
Philosopher 5 is hungry
Forks are not available for Philosopher 5 to use
Philosopher 4 has finished eating
Philosopher 2 has finished eating
Philosopher 3 is hungry
Philosopher 3 has started eating
Philosopher 1 is hungry
Philosopher 1 has started eating
Philosopher 5 is hungry
Forks are not available for Philosopher 5 to use
Philosopher 3 has finished eating
Philosopher 1 has finished eating
Philosopher 5 is hungry
Philosopher 5 has started eating
Philosopher 5 has finished eating
```