Operating Systems Lab

Mariam Fatima
22F-3168
Lab 11

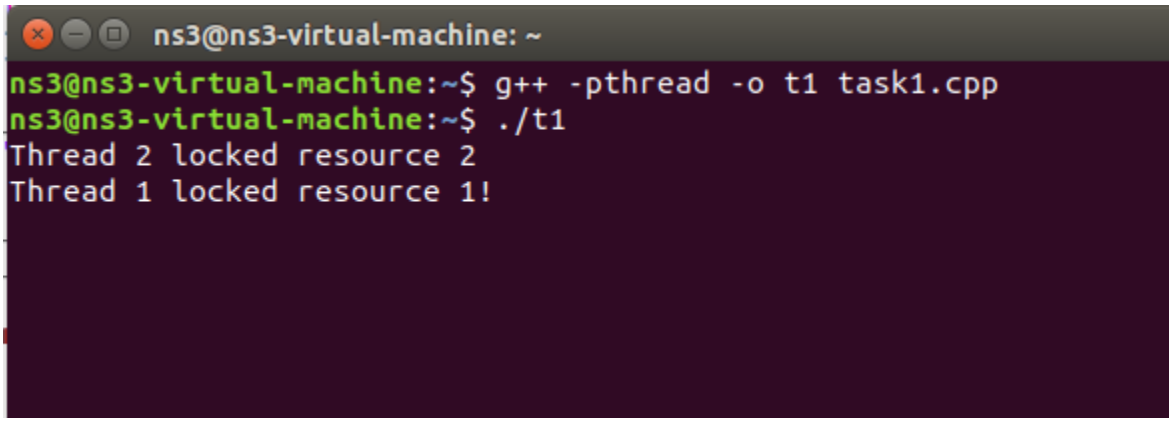
Q no 1:

Code:

```
#include <iostream>
#include <pthread.h>
#include <unistd.h>
using namespace std;

pthread_mutex_t r1 = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t r2 = PTHREAD_MUTEX_INITIALIZER;

void* thread1(void*) {
    pthread_mutex_lock(&r1);
    cout << "Thread 1 locked resource 1!" << endl;
    sleep(1);

    pthread_mutex_lock(&r2);
    cout << "Thread 1 locked resource 2!" << endl;

    pthread_mutex_unlock(&r2);
    pthread_mutex_unlock(&r1);

    return nullptr;
}

void* thread2(void*) {
    pthread_mutex_lock(&r2);
    cout << "Thread 2 locked resource 2" << endl;
    sleep(1);

    pthread_mutex_lock(&r1);
    cout << "Thread 2 locked resource 1" << endl;
```

```
    pthread_mutex_unlock(&r1);
    pthread_mutex_unlock(&r2);

    return nullptr;
}

int main() {
    pthread_t t1, t2;

    // Create threads
    pthread_create(&t1, nullptr, thread1, nullptr);
    pthread_create(&t2, nullptr, thread2, nullptr);

    // Wait for threads to finish
    pthread_join(t1, nullptr);
    pthread_join(t2, nullptr);

    return 0;
}
```

Output:



Q no 2:

Code:

```
#include<iostream>

#include<pthread.h>

#include<unistd.h>



using namespace std;



const int N = 4; // Number of nodes (p0, r1, p1, r2)

int graph[N][N];

bool visited[N];

bool recStack[N];

bool deadlockFlag = false;

pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;



void* detectCycle(void* arg) {

    int node = *(int*)arg;



    pthread_mutex_lock(&lock);
```

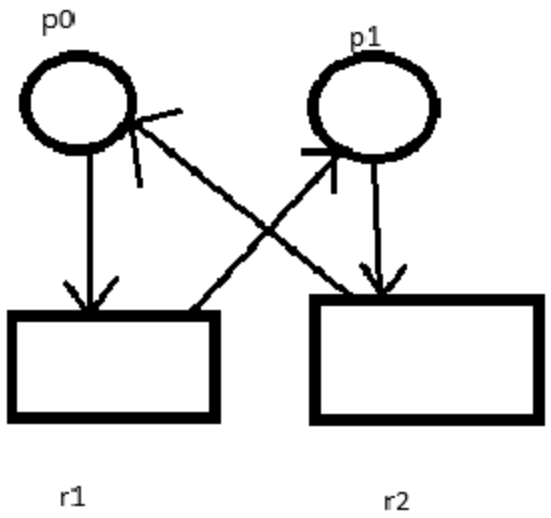
```
if (visited[node]) {
    pthread_mutex_unlock(&lock);
    pthread_exit(nullptr);
}

visited[node] = true;
recStack[node] = true;
pthread_mutex_unlock(&lock);

for (int i = 0; i < N; i++) {
    if (graph[node][i]) {
        pthread_mutex_lock(&lock);
        if (recStack[i]) {
            deadlockFlag = true;
            pthread_mutex_unlock(&lock);
            pthread_exit(nullptr);
        }
        pthread_mutex_unlock(&lock);
```

```
            if (!visited[i]) {
                detectCycle(&i);
            }
        }
    }

    pthread_mutex_lock(&lock);
    recStack[node] = false;
    pthread_mutex_unlock(&lock);
    pthread_exit(nullptr);
}

int main() {
    pthread_t threads[N];
    int threadArgs[N];

    // Initialize graph and visited arrays
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
```

```
        graph[i][j] = 0;
    }
    visited[i] = false;
    recStack[i] = false;
}

graph[0][1] = 1; // p0 -> r1
graph[1][2] = 1; // r1 -> p1
graph[2][3] = 1; // p1 -> r2
graph[3][0] = 1; // r2 -> p0

// Create threads
for (int i = 0; i < N; i++) {
    threadArgs[i] = i;
    pthread_create(&threads[i], nullptr, detectCycle, (void*)&threadArgs[i]);
}

// Wait for all threads to complete
for (int i = 0; i < N; i++) {
```

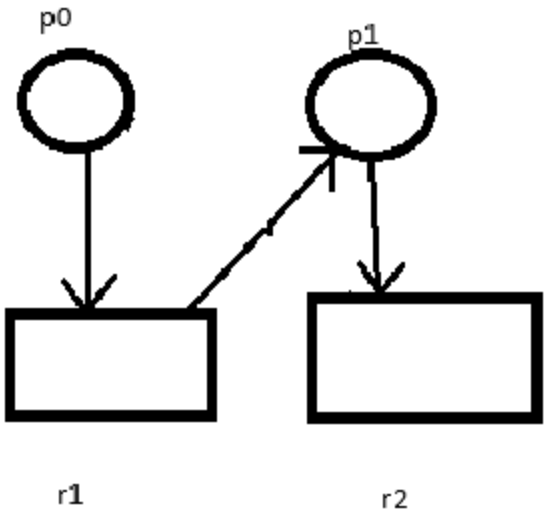
```
        pthread_join(threads[i], nullptr);
    }

    // Output result
    if (deadlockFlag) {
        cout << "Deadlock Detected" << endl;
    } else {
        cout << "Deadlock not Detected" << endl;
    }

    return 0;
}
```

Output:




```
graph[0][1] = 1; // p0 -> r1
graph[1][2] = 1; // r1 -> p1
graph[2][3] = 1; // p1 -> r2
graph[3][0] = 1; // r2 -> p0
```

```
ns3@ns3-virtual-machine:~$ g++ -pthread -o t2 task2.cpp
ns3@ns3-virtual-machine:~$ ./t2
Deadlock Detected
ns3@ns3-virtual-machine:~$
```




```
graph[0][1] = 1; // p0 -> r1
graph[1][2] = 1; // r1 -> p1
graph[2][3] = 1; // p1 -> r2
```

```
ns3@ns3-virtual-machine:~$ g++ -pthread -o t2 task2.cpp
ns3@ns3-virtual-machine:~$ ./t2
Deadlock not Detected
ns3@ns3-virtual-machine:~$
```