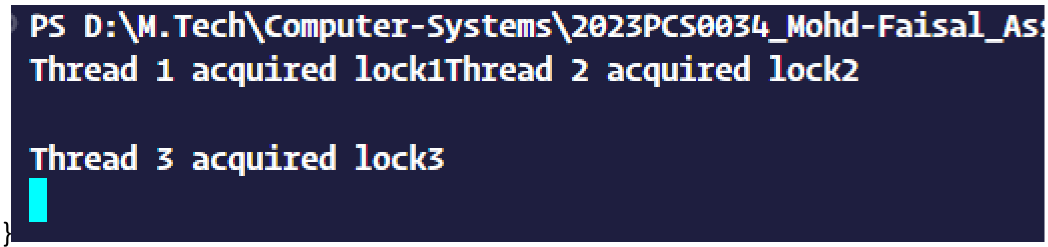
Assignment 5
Name: Mohammad Faisal Sayed

1. Create Deadlock between threads

```
#include <iostream>
#include <thread>
#include <mutex>

std::mutex mutex1, mutex2, mutex3;

void thread1()
{
   std::unique_lock<std::mutex> lock1(mutex1);
   std::cout << "Thread 1 acquired lock1" << std::endl;
   std::this_thread::sleep_for(std::chrono::milliseconds(100));
   std::unique_lock<std::mutex> lock2(mutex2);
   std::cout << "Thread 1 acquired lock2" << std::endl;
}

void thread2()
{
   std::unique_lock<std::mutex> lock2(mutex2);
   std::cout << "Thread 2 acquired lock2" << std::endl;
   std::this_thread::sleep_for(std::chrono::milliseconds(100));
   std::unique_lock<std::mutex> lock3(mutex3);
   std::cout << "Thread 2 acquired lock3" << std::endl;
}

void thread3()
{
   std::unique_lock<std::mutex> lock3(mutex3);
   std::cout << "Thread 3 acquired lock3" << std::endl;
   std::this_thread::sleep_for(std::chrono::milliseconds(100));
   std::unique_lock<std::mutex> lock1(mutex1);
   std::cout << "Thread 3 acquired lock1" << std::endl;
}

int main()
{
   std::thread t1(thread1);
   std::thread t2(thread2);
   std::thread t3(thread3);

   t1.join();
   t2.join();
   t3.join();
```

```
return 0;
```



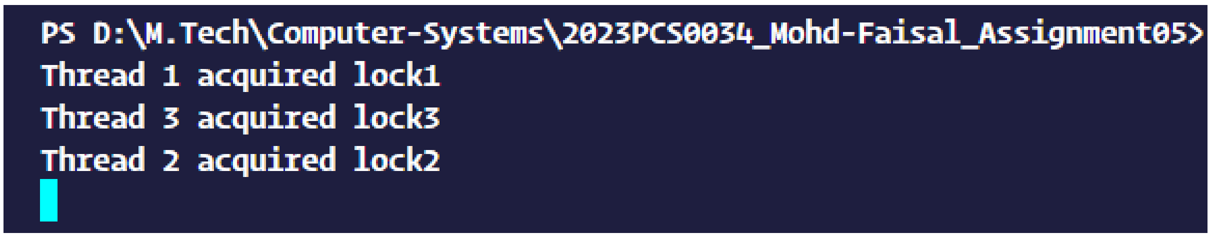
```
PS D:\M.Tech\Computer-Systems\2023PCS0034_Mohd-Faisal_As
Thread 1 acquired lock1Thread 2 acquired lock2

Thread 3 acquired lock3
```

}

2. Write a program to create four threads: thread1, thread2, thread3, and thread4. Create a deadlock situation between thread1 and thread3.

```
#include <iostream>
#include <thread>
#include <mutex>

std::mutex mutex1, mutex3;

void thread1()
{
   std::unique_lock<std::mutex> lock1(mutex1);
   std::cout << "Thread 1 acquired lock1\n";
   std::cout << "Now trying to acquire lock on mutex 3....\n";
   std::this_thread::sleep_for(std::chrono::milliseconds(100));
   std::unique_lock<std::mutex> lock3(mutex3);
   std::cout << "Thread 1 acquired lock3\n";
}

void thread2()
{
   std::this_thread::sleep_for(std::chrono::milliseconds(50));
   std::unique_lock<std::mutex> lock1(mutex1);
   std::cout << "Thread 2 acquired lock1\n";
}

void thread3()
{
   std::unique_lock<std::mutex> lock3(mutex3);
   std::cout << "Thread 3 acquired lock3\n\n";
   std::cout << "Now trying to acquire lock on mutex 1....\n";
   std::this_thread::sleep_for(std::chrono::milliseconds(100));
   std::unique_lock<std::mutex> lock1(mutex1);
   std::cout << "Thread 3 acquired lock1\n";
}

void thread4()
{
   std::this_thread::sleep_for(std::chrono::milliseconds(50));
   std::unique_lock<std::mutex> lock3(mutex3);
```

```
    std::cout << "Thread 4 acquired lock3\n";
}

int main()
{
    std::thread t1(thread1);
    std::thread t2(thread2);
    std::thread t3(thread3);
    std::thread t4(thread4);

    t1.join();
    t2.join();
    t3.join();
    t4.join();

    return 0;
}
```

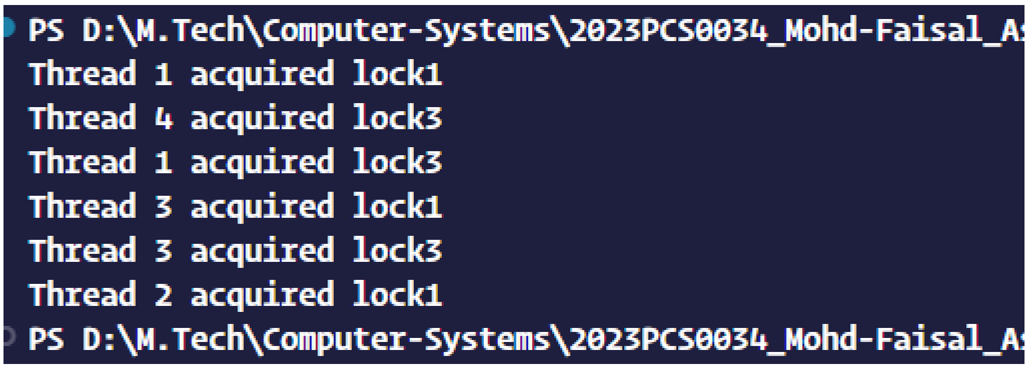


```
PS D:\M.Tech\Computer-Systems\2023PCS0034_Mohd-Faisal_Assignment05>
Thread 1 acquired lock1
Thread 3 acquired lock3
Thread 2 acquired lock2
```

3. Revise the program to avoid deadlock

```
#include <iostream>
#include <thread>
#include <mutex>

std::mutex mutex1, mutex3;

void thread1()
{
    std::unique_lock<std::mutex> lock1(mutex1);
    std::cout << "Thread 1 acquired lock1" << std::endl;
    std::this_thread::sleep_for(std::chrono::milliseconds(100));
    std::unique_lock<std::mutex> lock3(mutex3);
    std::cout << "Thread 1 acquired lock3" << std::endl;
}

void thread2()
{
    std::this_thread::sleep_for(std::chrono::milliseconds(50)); // Simulate some work
    std::unique_lock<std::mutex> lock1(mutex1);
    std::cout << "Thread 2 acquired lock1" << std::endl;
}

void thread3()
```

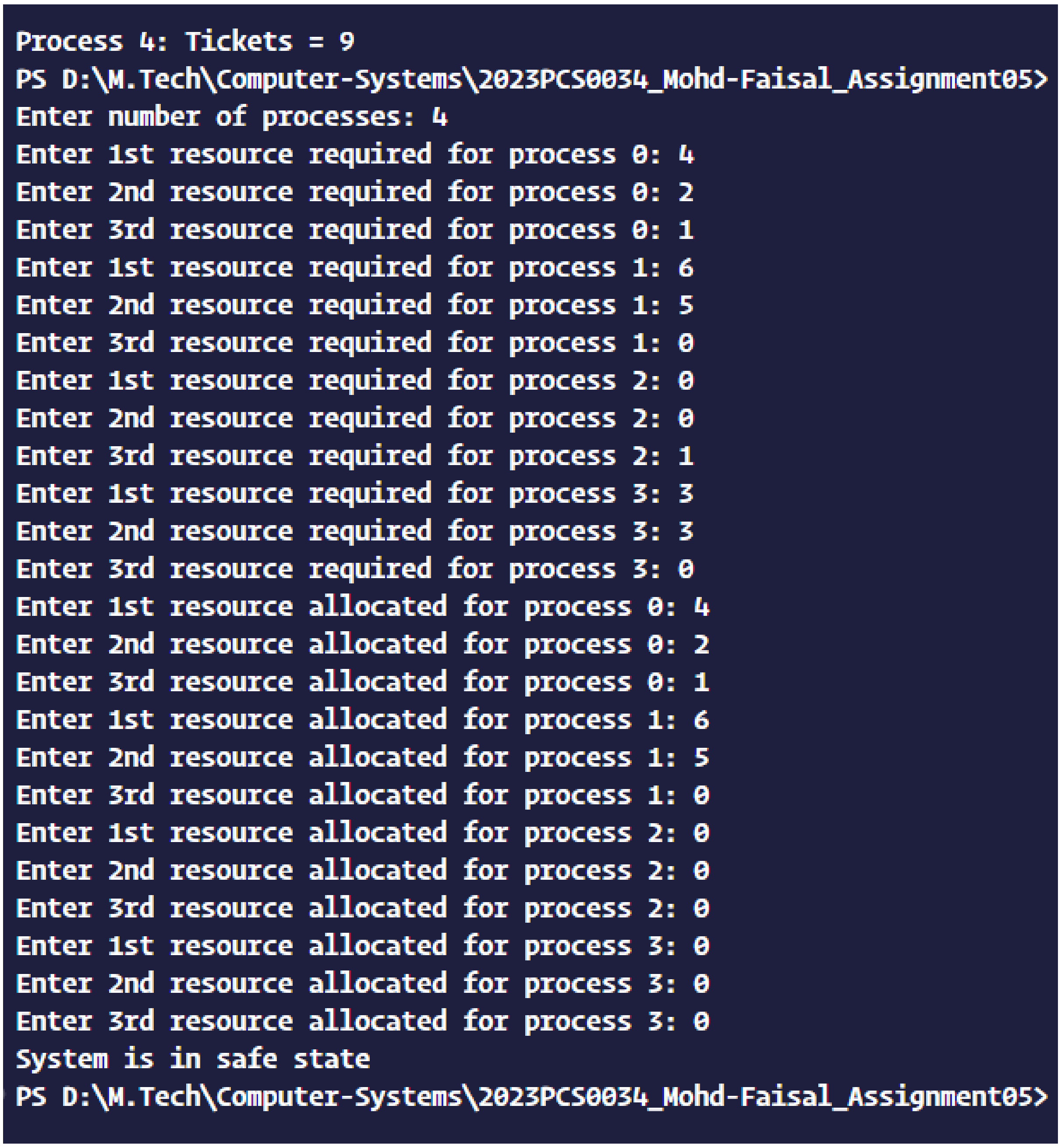
```
{
    std::unique_lock<std::mutex> lock1(mutex1);
    std::cout << "Thread 3 acquired lock1" << std::endl;
    std::this_thread::sleep_for(std::chrono::milliseconds(100));
    std::unique_lock<std::mutex> lock3(mutex3);
    std::cout << "Thread 3 acquired lock3" << std::endl;
}

void thread4()
{
    std::this_thread::sleep_for(std::chrono::milliseconds(50)); // Simulate some work
    std::unique_lock<std::mutex> lock3(mutex3);
    std::cout << "Thread 4 acquired lock3" << std::endl;
}

int main()
{
    std::thread t1(thread1);
    std::thread t2(thread2);
    std::thread t3(thread3);
    std::thread t4(thread4);

    t1.join();
    t2.join();
    t3.join();
    t4.join();

    return 0;
}

/*
both thread1 and thread3 lock mutex1 before attempting to lock mutex3, ensuring that they lock
the mutexes in the same order and avoiding a potential deadlock. thread2 and thread4 still
simulate some work to delay the acquisition of locks, but now the program will not enter a
deadlock state.
*/
```



```
PS D:\M.Tech\Computer-Systems\2023PCS0034_Mohd-Faisal_A
Thread 1 acquired lock1
Thread 4 acquired lock3
Thread 1 acquired lock3
Thread 3 acquired lock1
Thread 3 acquired lock3
Thread 2 acquired lock1
PS D:\M.Tech\Computer-Systems\2023PCS0034_Mohd-Faisal_A
```

4. Bankers algorithm

```
#include <iostream>
#include <algorithm>
using namespace std;
int main() {
 int n;
 cout << "Enter number of processes: ";
 cin >> n;
 int* finish = new int[n];
 int** maxi = new int*[n];
 int** alloc = new int*[n];
 int** need = new int*[n];
 int work[3] = {3, 2, 2};
 for (int i = 0; i < n; i++) {
 maxi[i] = new int[3];
 cout << "Enter 1st resource required for process " << i << ": ";
 cin >> maxi[i][0];
 cout << "Enter 2nd resource required for process " << i << ": ";
 cin >> maxi[i][1];
 cout << "Enter 3rd resource required for process " << i << ": ";
 cin >> maxi[i][2];
 }
 for (int i = 0; i < n; i++) {
 alloc[i] = new int[3];
 cout << "Enter 1st resource allocated for process " << i << ": ";
 cin >> alloc[i][0];
 cout << "Enter 2nd resource allocated for process " << i << ": ";
 cin >> alloc[i][1];
 cout << "Enter 3rd resource allocated for process " << i << ": ";
 cin >> alloc[i][2];
 }
 for (int i = 0; i < n; i++) {
 need[i] = new int[3];
 for (int j = 0; j < 3; j++) {
 need[i][j] = maxi[i][j] - alloc[i][j];
 }
 }
 int i = 0;
 int j = 0;
 while (j < 3 * n && (std::find(finish, finish + n, 0) != finish + n)) {
 int flag = 0;
 j++;
 if (need[i][0] <= work[0] && need[i][1] <= work[1] && need[i][2] <= work[2]) {
 work[0] += alloc[i][0];
 work[1] += alloc[i][1];
 work[2] += alloc[i][2];
 finish[i] = 1;
 flag = 1;
 i = (i + 1) % n;
 }
 if (flag == 0) {
 i = (i + 1) % n;
 }
 }
 int nflag = 0;
 for (int i = 0; i < n; i++) {
 if (finish[i] == 0) {
 nflag = 1;
 break;
 }
 }
 if (nflag == 0) {
 cout << "System is in safe state" << endl;
 } else {
 cout << "System is in deadlock" << endl;
 }
 // Free allocated memory
 delete[] finish;
 for (int i = 0; i < n; i++) {
 delete[] maxi[i];
 delete[] alloc[i];
 delete[] need[i];
 }
 delete[] maxi;
 delete[] alloc;
 delete[] need;
 return 0;
             }
```



```
Process 4: Tickets = 9
PS D:\M.Tech\Computer-Systems\2023PCS0034_Mohd-Faisal_Assignment05>
Enter number of processes: 4
Enter 1st resource required for process 0: 4
Enter 2nd resource required for process 0: 2
Enter 3rd resource required for process 0: 1
Enter 1st resource required for process 1: 6
Enter 2nd resource required for process 1: 5
Enter 3rd resource required for process 1: 0
Enter 1st resource required for process 2: 0
Enter 2nd resource required for process 2: 0
Enter 3rd resource required for process 2: 1
Enter 1st resource required for process 3: 3
Enter 2nd resource required for process 3: 3
Enter 3rd resource required for process 3: 0
Enter 1st resource allocated for process 0: 4
Enter 2nd resource allocated for process 0: 2
Enter 3rd resource allocated for process 0: 1
Enter 1st resource allocated for process 1: 6
Enter 2nd resource allocated for process 1: 5
Enter 3rd resource allocated for process 1: 0
Enter 1st resource allocated for process 2: 0
Enter 2nd resource allocated for process 2: 0
Enter 3rd resource allocated for process 2: 0
Enter 1st resource allocated for process 3: 0
Enter 2nd resource allocated for process 3: 0
Enter 3rd resource allocated for process 3: 0
System is in safe state
PS D:\M.Tech\Computer-Systems\2023PCS0034_Mohd-Faisal_Assignment05>
```