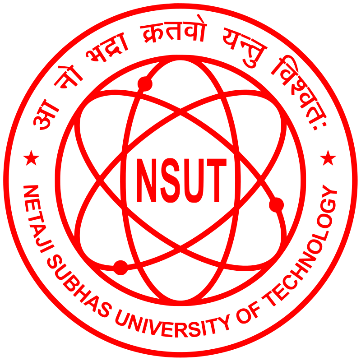
**NETAJI SUBHAS UNIVERSITY OF TECHNOLOGY**



**DISTRIBUTED COMPUTING**

**CACSC15**

**Mrs. Raina Joon**

**Sneha Gupta**

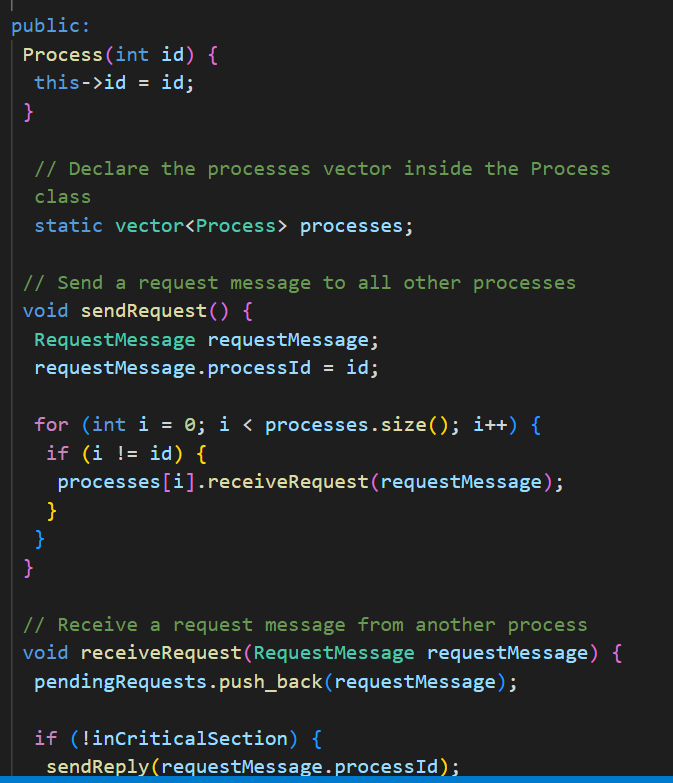
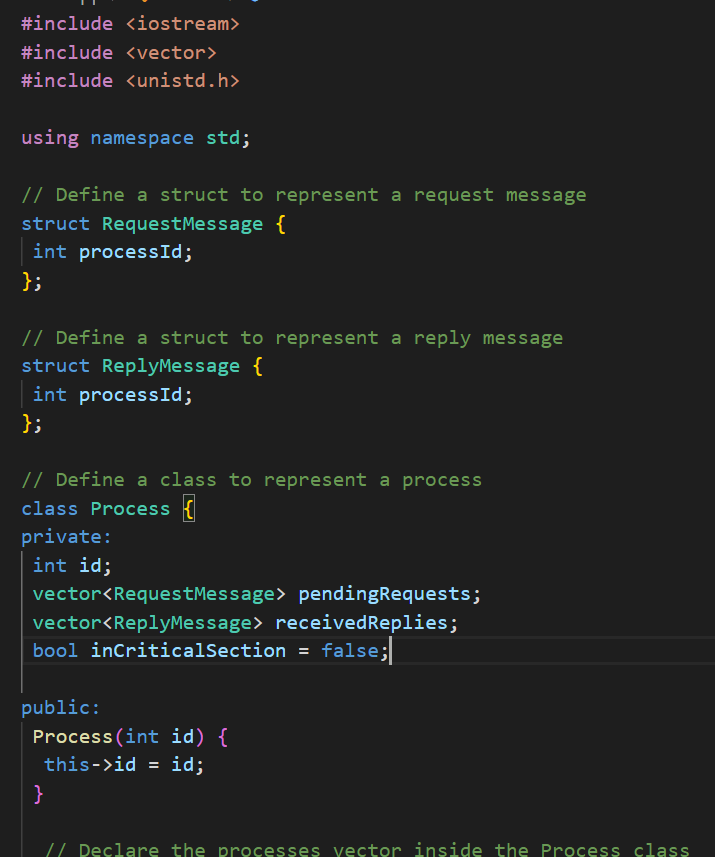
**2021UCA1859**

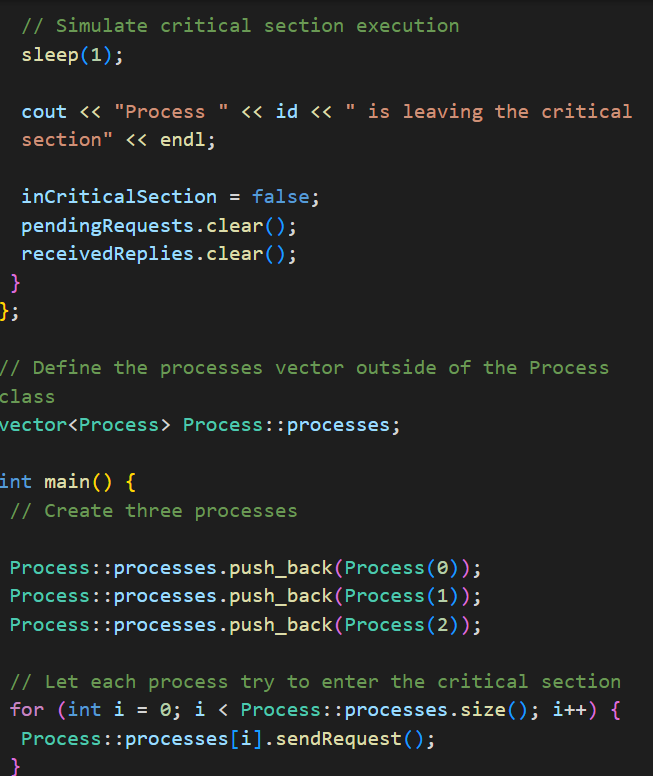
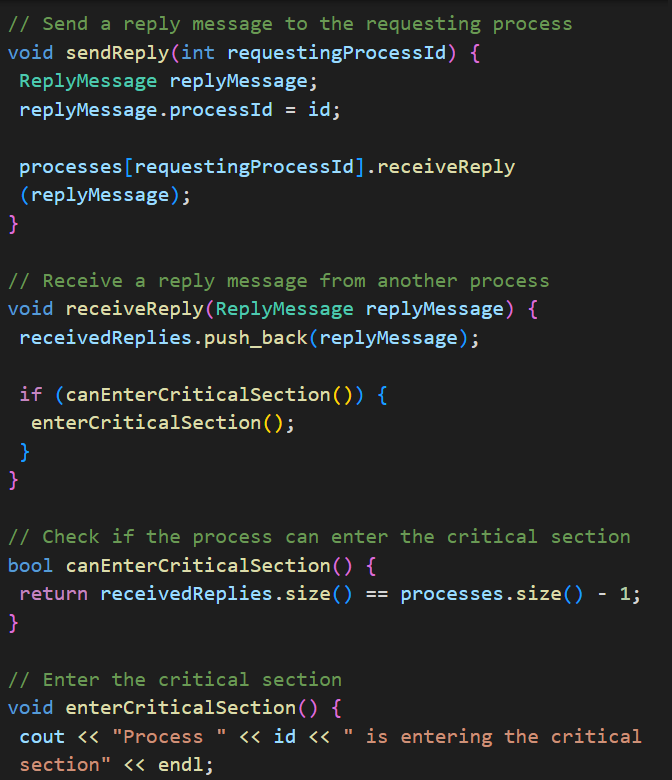
Assignment 1

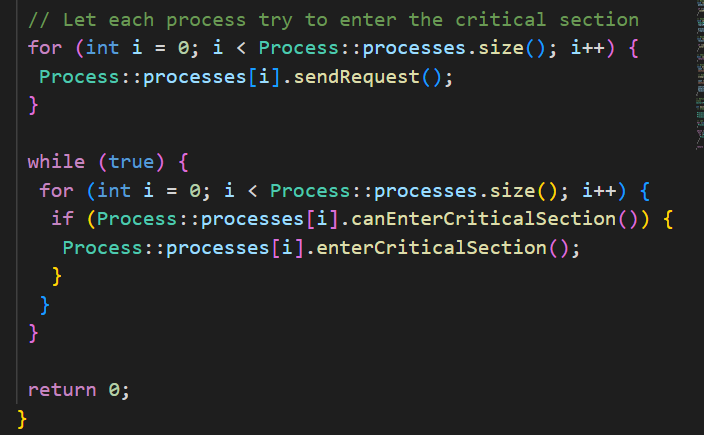
Program to implement non token based algorithm for Mutual Exclusion

Theory

In concurrent programming, mutual exclusion is a fundamental concept that ensures that only one thread or process accesses a shared resource at any given time. This is essential to prevent race conditions and maintain data consistency in multi-threaded or multi-process applications.







OUTPUT:



Assignment 2

Program to implement Lamport’s Logical Clock

Theory:

Lamport's Logical Clock is a simple algorithm for ordering events in a distributed system. It does not rely on physical time but rather assigns a logical timestamp to each event. The key idea is that if event A happened before event B, their logical timestamps should reflect this relationship.

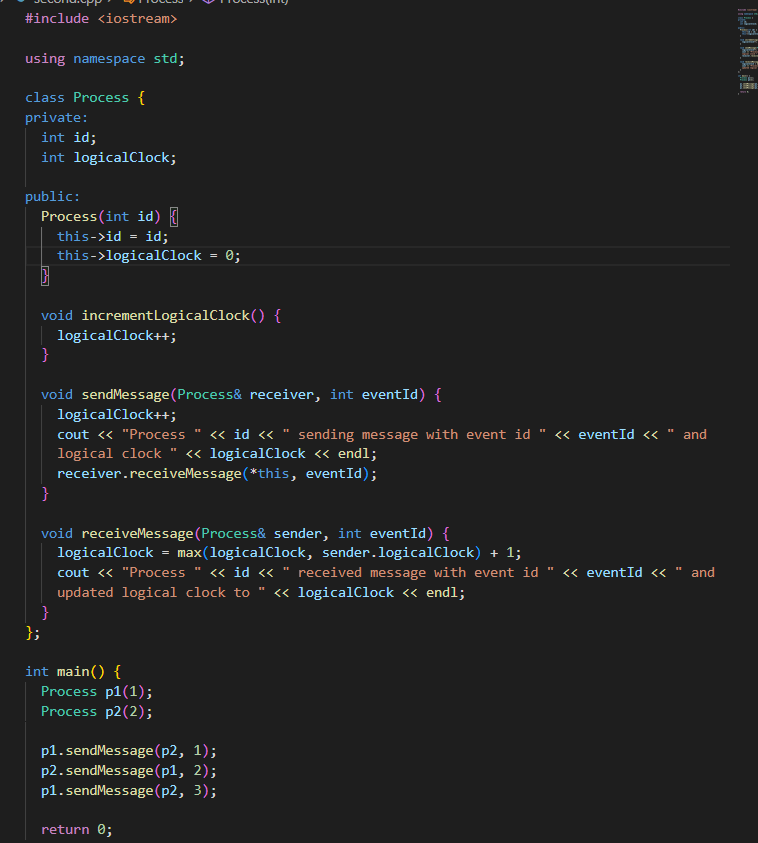
The algorithm works as follows:

1. Each process maintains a local logical clock, initially set to zero.

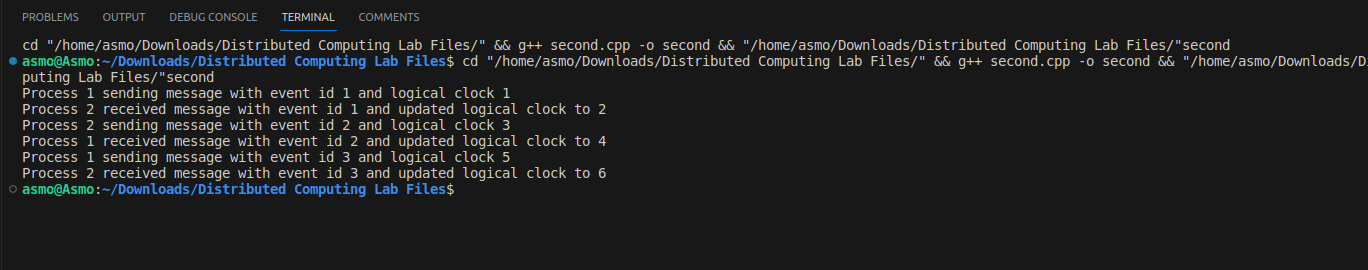
2. When a process performs an event (e.g., sending a message or receiving a message), it increments its logical clock by one and timestamps the event with the current value of its logical clock.

3. When a process receives a message with a timestamp, it updates its local logical clock to be the maximum of its current value and the timestamp received in the message, plus one. It then timestamps the event with this new logical clock value.

4. Events can be compared based on their logical timestamps. If Event A has a lower logical timestamp than Event B, it means A happened before B.



OUTPUT:



Assignment 3

Program to implement edge chasing distributed deadlock detection algorithm.

Theory: Lamport's Logical Clock is a simple algorithm for ordering events in a distributed system. It does not rely on physical time but rather assigns a logical timestamp to each event. The key idea is that if event A happened before event B, their logical timestamps should reflect this relationship.

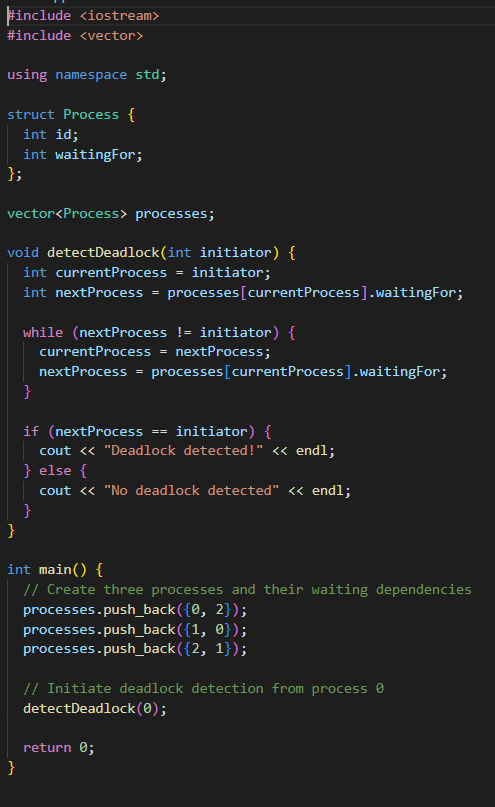
The algorithm works as follows:

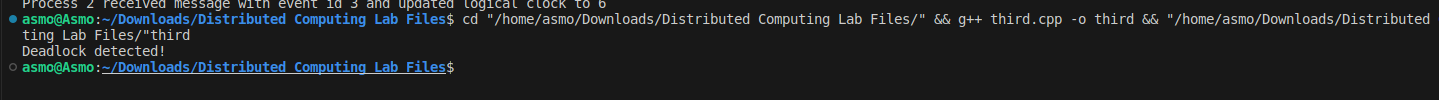
1. Each process maintains a local logical clock, initially set to zero.

2. When a process performs an event (e.g., sending a message or receiving a message), it increments its logical clock by one and timestamps the event with the current value of its logical clock.

3. When a process receives a message with a timestamp, it updates its local logical clock to be the maximum of its current value and the timestamp received in the message, plus one. It then timestamps the event with this new logical clock value.

4. Events can be compared based on their logical timestamps. If Event A has a lower logical timestamp than Event B, it means A happened before B.





Assignment 4

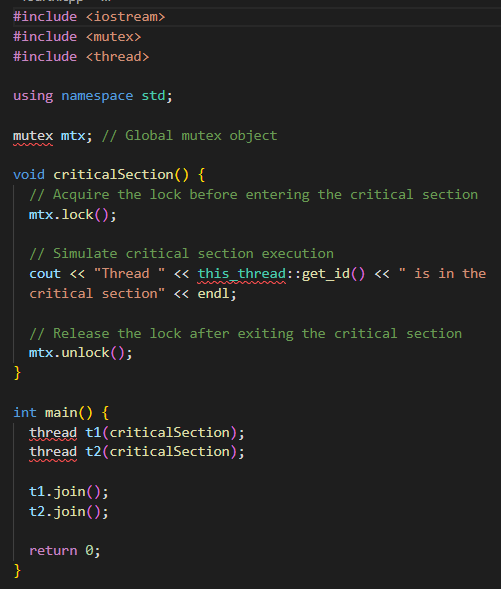
Program to implement locking algorithm.

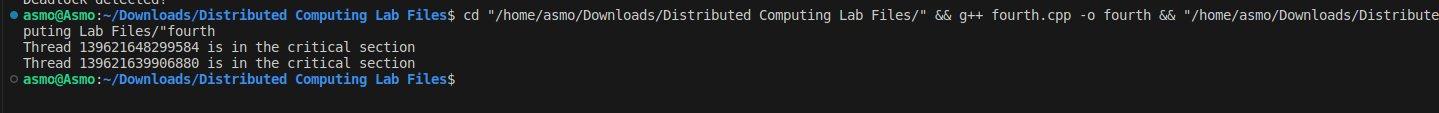
Theory:

A locking algorithm ensures that only one thread can access a critical section of code at a time, preventing data races and conflicts that may lead to incorrect results or program crashes. Locking mechanisms come in various forms, with mutex locks being one of the most common. Here's how they work:

Mutex (Mutual Exclusion): A mutex is a synchronization primitive that allows threads to lock access to a shared resource. When a thread locks a mutex, it enters a critical section. If another thread tries to lock the same mutex while it's locked by another thread, it will block until the first thread releases the lock.

Lock and Unlock: Threads must explicitly lock the mutex before entering the critical section and unlock it when they exit. This ensures mutual exclusion and prevents multiple threads from accessing the critical section simultaneously.





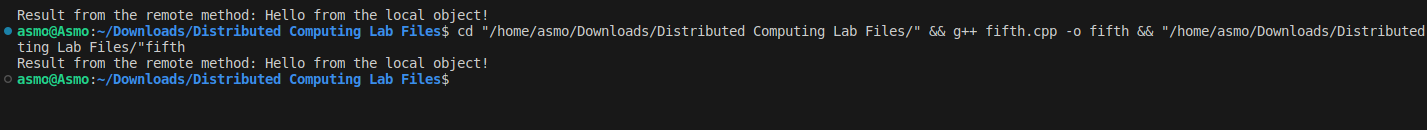
Assignment 5

Program to implement Remote Method Invocation.

Theory:

Remote Method Invocation (RMI) is a mechanism that allows an object in one address space (usually on a remote machine) to invoke methods on an object in another address space, possibly on a different machine. RMI is a fundamental concept in distributed computing and is typically used in client-server applications. It enables communication between objects in different address spaces by making method calls appear as if they were local.





Assignment 6

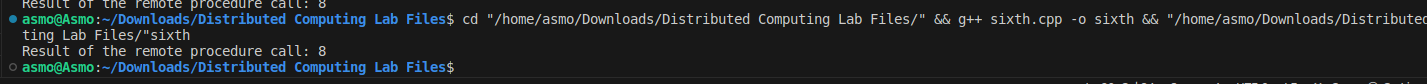
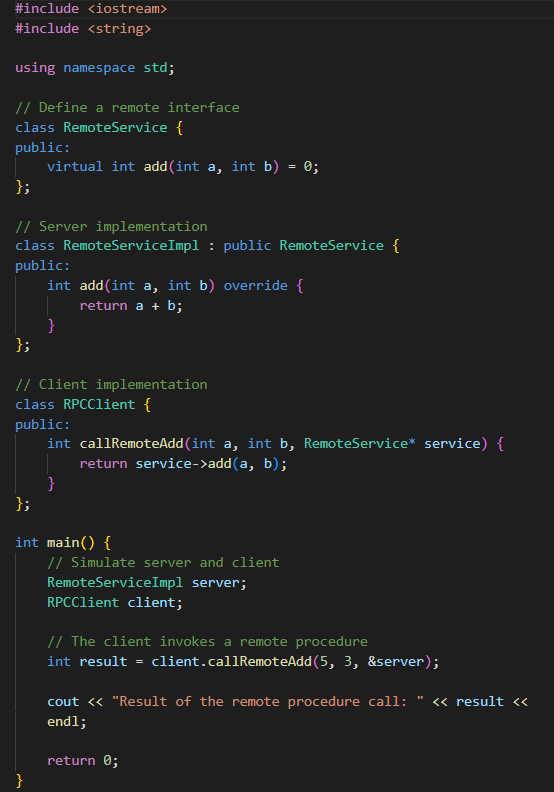
Program to implement Remote Procedure Call. Remote Procedure Call (RPC) is a powerful mechanism for invoking procedures or methods in a different address space, typically on a remote machine. RPC allows distributed systems to make a local procedure call on a remote system as if it were a local call. In C++, this is often implemented using technologies such as gRPC or Apache Thrift.

Theory:

RPC involves two main components:

1. Client: The client initiates a procedure call, and the local stub (proxy) prepares a request message. This message is then sent to the server using a network protocol. Distributed Computing 11

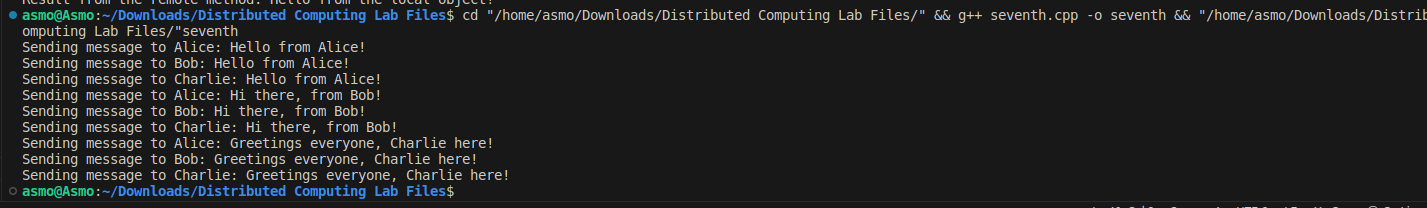
2. Server: The server receives the request message and forwards it to the local stub (skeleton). The local stub calls the actual procedure/method on the server. The result is then sent back to the client in a response message.



Assignment 7

Program to implement Chat Server

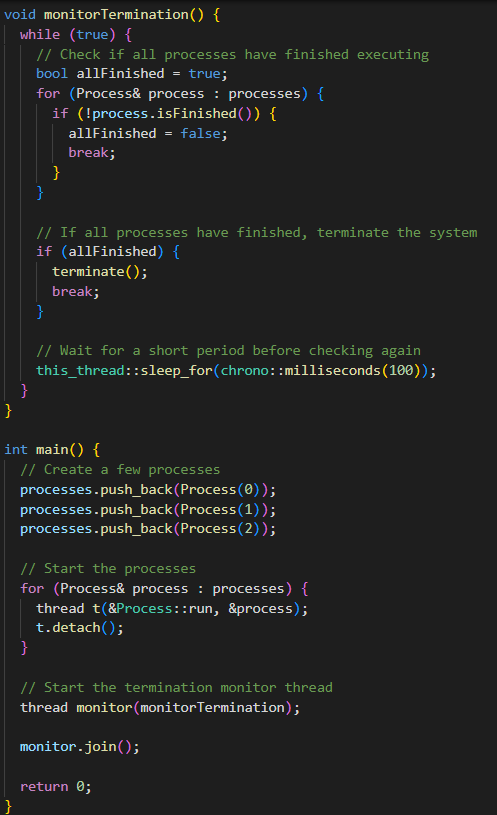
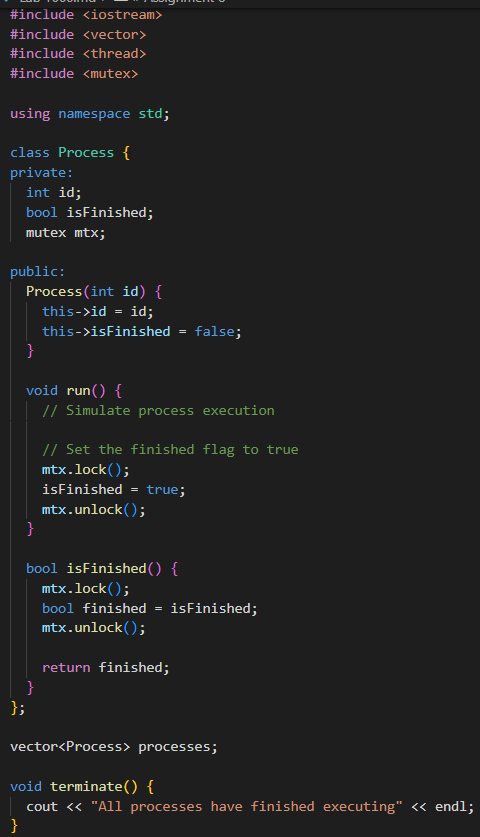




Assignment 8

Program to implement termination detection

Termination detection is an essential concept in distributed systems, ensuring that the system can determine when a distributed computation or set of processes has finished. Termination detection can be useful for various purposes, such as resource reclamation, signaling completion, or initiating another phase of a distributed computation.



Assignment 9

To implement CORBA mechanism by using C++ program at one end and Java Program on the other.

Theory:

CORBA (Common Object Request Broker Architecture) is a standard for distributed object computing that allows objects on different machines to communicate with each other as if they were local. CORBA uses an IDL (Interface Definition Language) to define the interfaces of distributed objects, which can then be implemented in different programming languages.

