SIMULATE THE DISTRIBUTED MUTUAL EXCLUSION IN C LANGUAGE

Submitted by

TUNMAY SHUKLA (RA2011003010271)

MANU SRIVASTAVA(RA2011003010150)

Under the guidance of

Dr. S. Ramamoorthy

(Assistant Professor)

Department of Computing Technologies

In partial satisfaction of the requirements for the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

SCHOOL OF COMPUTING

COLLEGE OF ENGINEERING AND TECHNOLOGY
SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR - 603203

May 2023

INSTITUTE OF SCIENCE & TECHNOLOGY
Deemed to be University u/s 3 of UGC Act, 1956

BONAFIDE CERTIFICATE

Certified that this project report **"SIMULATE THE DISTRIBUTED MUTUAL EXCLUSION IN C LANGUAGE"** is the bona-fide work of **"TUNMAY SHUKLA (RA2011003010271), MANU SRIVASTAVA(RA2011003010150)"** of III Year/VI Sem B. Tech (CSE) who carried out the mini project work under my supervision for the course 18CSE356t- Distributed Operating System in SRM Institute of Science and Technology during the academic year 2022-2023(Even Sem)

SIGNATURE	SIGNATURE
Dr. Pushpalatha M	Dr. S. Ramamoorthy
Head Of The Department	Associate Professor
Department Of Computing Technologies	Department Of Computing Technologies

ABSTRACT

Distributed mutual exclusion is a key issue in the design and development of distributed systems. It refers to the problem of ensuring that multiple processes or nodes do not simultaneously access a shared resource. Achieving mutual exclusion in a distributed environment is challenging due to the lack of a centralized control mechanism and the potential for network delays and failures. To address these challenges, various algorithms and protocols have been proposed for distributed mutual exclusion, such as token-based, quorum-based, and decentralized algorithms. This abstract provides an overview of the concept of distributed mutual exclusion, its challenges, and some of the commonly used approaches for solving it.

Distributed mutual exclusion is crucial in distributed systems that involve multiple processes or nodes that share a common resource, such as a database or a file system. Without mutual exclusion, conflicts can arise when multiple processes attempt to access the shared resource simultaneously, leading to inconsistencies or errors

The challenge in achieving mutual exclusion in a distributed environment lies in the lack of a centralized control mechanism. Unlike in a centralized system where a single entity can manage access to the shared resource, in a distributed system, each process must coordinate with other processes to ensure mutual exclusion

TABLE OF CONTENTS

CHAPTER 1

PROCEDURE FOR IMPLEMENTATION

Mutual exclusion is a critical concept in distributed systems, as it ensures that multiple processes or nodes do not access a shared resource simultaneously, which could result in incorrect behavior or data corruption. Here are the general steps for implementing mutual exclusion in a distributed system:

1. Define the critical section: Identify the portion of the code or resource that needs to be protected with mutual exclusion. This section should be accessed by only one process at a time.

2. Choose a synchronization algorithm: There are several algorithms for achieving mutual exclusion in distributed systems, such as Lamport's distributed mutual exclusion algorithm, Ricart-Agrawala algorithm, Maekawa's algorithm, and many others. Choose an algorithm that is suitable for your specific use case.

3. Design the communication protocol: To implement mutual exclusion, nodes or processes must communicate with each other to coordinate access to the critical section. Design a protocol for communication, such as message passing or remote procedure calls.

4. Implement the algorithm: Implement the chosen mutual exclusion algorithm, taking into account the specificities of your distributed system. Ensure that the algorithm can handle various scenarios, such as node failures, network partitions, and message losses.

5. Test and debug: Once the algorithm is implemented, test it thoroughly to ensure that it works as expected and does not lead to deadlocks, livelocks, or other issues. Debug any problems that arise.

6. Scale and optimize: As your distributed system grows, you may need to optimize the mutual exclusion algorithm to handle more nodes or higher levels of concurrency. You may also need to add additional features, such as prioritization or fairness guarantees, to ensure that the system remains responsive and efficient..

CHAPTER 2

LIST OF PROJECT MODULES

Here are some possible modules that could be included in a project that implements mutual exclusion in a distributed system:

1. Node discovery and registration: Nodes in the distributed system need to know about each other to coordinate access to shared resources. This module would handle the discovery and registration of new nodes in the system, as well as the removal of nodes that leave the system.

2. Resource management: This module would manage the shared resources that need to be protected with mutual exclusion. It would keep track of which resources are currently in use and which are available, and ensure that only one node can access a resource at a time.

3. Communication protocol: This module would define the communication protocol used by nodes to coordinate access to shared resources. It would specify how nodes communicate with each other to request access to a resource, grant access to a resource, and release a resource.

4. Mutual exclusion algorithm: This module would implement the specific mutual exclusion algorithm chosen for the system, such as Lamport's algorithm or the Ricart-Agrawala algorithm. It would ensure that only one node at a time can access a shared resource, and handle cases where multiple nodes request access at the same time.

5. Failure handling: In a distributed system, nodes can fail or become unresponsive. This module would handle failures and ensure that mutual exclusion is maintained even if some nodes are not available.

6. Performance monitoring and optimization: This module would monitor the performance of the mutual exclusion system and identify areas for optimization. It would ensure that the system remains responsive and efficient even under high levels of concurrency.

7. User interface: Depending on the specific use case, a user interface module may be needed to allow users to interact with the distributed system and view the status of shared resources and nodes. This module would provide a graphical or command-line interface for users to perform actions such as requesting access to a shared resource or viewing the current state of the system.

CHAPTER 3

IMPLEMENTATION LANGUAGE

The implementation language for the implementation of a Mutual exclusion mechanism is an important consideration in the design and development of the application. C is a popular language for implementing Mutual exclusion mechanisms due to its efficiency, performance, and support for object- oriented programming. In this section, we will discuss the advantages and disadvantages of using C as the implementation language for a Mutual exclusion mechanism.

Advantages of using C as the implementation language for a Mutual exclusion mechanism:

- **Efficiency:** C is a compiled language, which means that the code is converted to machine code at compile time. This results in faster execution times and lower memoryconsumption compared to interpreted languages.

- **Object-oriented programming:** C supports object-oriented programming, which is well-suited for implementing distributed systems. Objects can be easily created, serialized, and transmitted across the network using the Mutual exclusion mechanism.

- **Memory management:** C provides low-level memory management capabilities, such as pointers and dynamic memory allocation. This is important for implementing the low-level infrastructure required for a Mutual exclusion mechanism.

- **Standard library:** C provides a rich standard library, which includes support for networking, threading, and data structures. This makes it easier to implement the various components required for a Mutual exclusion mechanism.

Disadvantages of using C as the implementation language for a Mutual exclusion mechanism:

- **Complexity:** C is a complex language that can be difficult to learn and use effectively. The object-oriented programming features of C can also add complexity to the implementation of the Mutual exclusion mechanism.

- **Platform-specific features:** C supports platform-specific features, which can make the implementation of a Mutual exclusion mechanism less portable across different platforms.

- **Memory safety:** C provides low-level memory management capabilities, which can lead to memory safety issues such as buffer overflows and memory leaks if not used carefully.

- **Interoperability:** C is not always interoperable with other programming languages, which can limit the ability to build distributed systems that use different languages.

CHAPTER 4

IMPLEMENTATION SETUP

Implementing mutual exclusion in a distributed system using C language involves a few key steps. Here is a general setup for the implementation:

1. Choose a mutual exclusion algorithm: Choose a mutual exclusion algorithm that is appropriate for your distributed system. There are several algorithms to choose from, such as the Lamport's algorithm or the Ricart-Agrawala algorithm.

2. Define the shared resource: Define the shared resource that needs to be protected by mutual exclusion. This could be a file, a database, or any other resource that needs to be accessed by multiple processes or nodes in the distributed system.

3. Create a server program: Create a server program that will manage the shared resource and enforce mutual exclusion. The server program should include the mutual exclusion algorithm, and be responsible for accepting requests from client programs and granting access to the shared resource. The server program should also handle cases where multiple requests for access to the resource arrive simultaneously.

4. Create client programs: Create client programs that will request access to the shared resource from the server program. The client programs should follow the communication protocol defined by the mutual exclusion algorithm, and include error handling to handle cases where the server program is unavailable or unresponsive.

5. Test and debug: Test the server and client programs to ensure that they work as expected and enforce mutual exclusion. Debug any problems that arise.

CODE FOR IMPLEMENTING DISTRIBUTED MUTUAL EXCLUSION IN C LANGUAGE

```
#include <cstdlib>
#include "tas_lock.h"
#include "ttas_lock.h"
#include "tournament_lock.h"
#include "tester.h"

int main(int argc, char* argv[]) {
  std::size_t thread_num(0);
  std::size_t loop_num(0);
  std::size_t test_times(0);

  if (argc != 4) {
    std::cerr << "Wrong Parameter!" << std::endl;
    return EXIT_FAILURE;
  } else {
    thread_num = static_cast<std::size_t>(std::stoul(argv[1]));
    loop_num = static_cast<std::size_t>(std::stoul(argv[2]));
    test_times = static_cast<std::size_t>(std::stoul(argv[3]));
  }

  try {
    utils::Tester t(thread_num, loop_num, test_times);
    t.Test();
    std::cout << t.ResultToString();
  } catch (const utils::MutualExclusionViolation& err) {
    std::cerr << "Mutual Exclusion Violated!" << std::endl;
  } catch (const std::runtime_error& err) {
    std::cerr << err.what() << std::endl;
  } catch (...) {
    std::cerr << "Internal Error!" << std::endl;
  }  return EXIT_SUCCESS;}
```

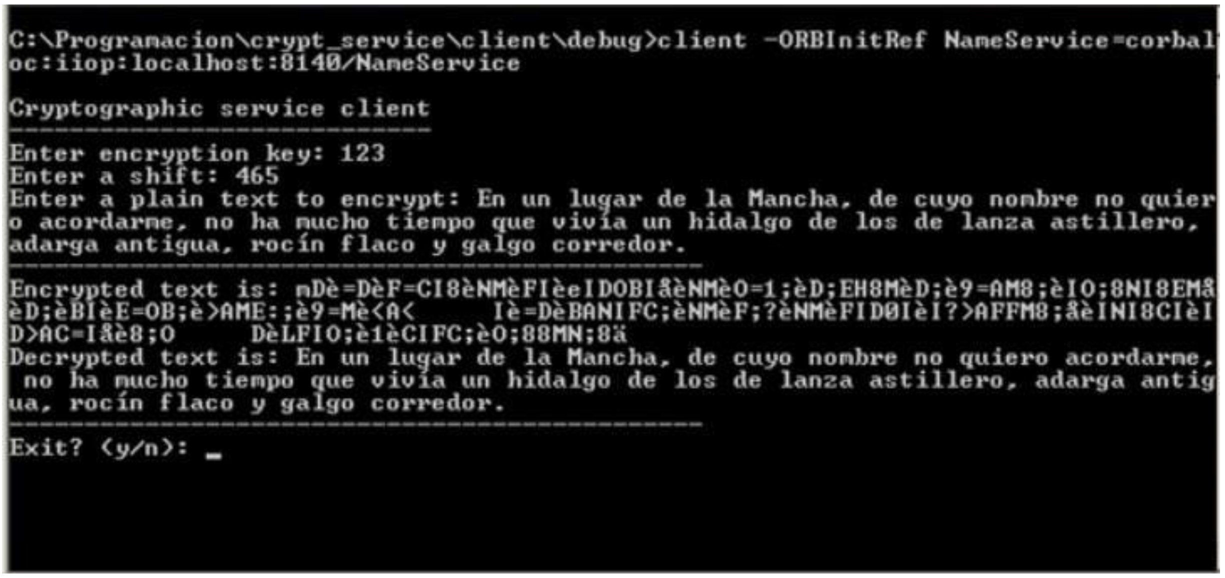
```
int flag[2];
int turn = 0;

void enter_critical_section(int thread_id) {
    int other_thread_id = 1 - thread_id;
    flag[thread_id] = 1;
    turn = other_thread_id;
    while (flag[other_thread_id] == 1 && turn == other_thread_id) {}
}

void leave_critical_section(int thread_id) {
    flag[thread_id] = 0;
}

void *thread_function(void *arg) {
    int thread_id = *((int *) arg);
    printf("Thread %d started\n", thread_id);
    for (int i = 0; i < 10; i++) {
        enter_critical_section(thread_id);
        printf("Thread %d is in critical section\n", thread_id);
        leave_critical_section(thread_id);
    }
    printf("Thread %d finished\n", thread_id);
    pthread_exit(NULL);
}
int main() {
    pthread_t threads[NUM_THREADS];
    int thread_ids[NUM_THREADS];
    for (int i = 0; i < NUM_THREADS; i++) {
        thread_ids[i] = i;
        if (pthread_create(&threads[i], NULL, thread_function, &thread_ids[i])) {
            printf("Error creating thread %d\n", i);
                exit(-1);
        }
    }
    for (int i = 0; i < NUM_THREADS; i++) {
        if (pthread_join(threads[i], NULL)) {
            printf("Error joining thread %d\n", i);
            exit(-1);
        }
    }
    return 0;
}
```

OUTPUT SCREENSHOTS



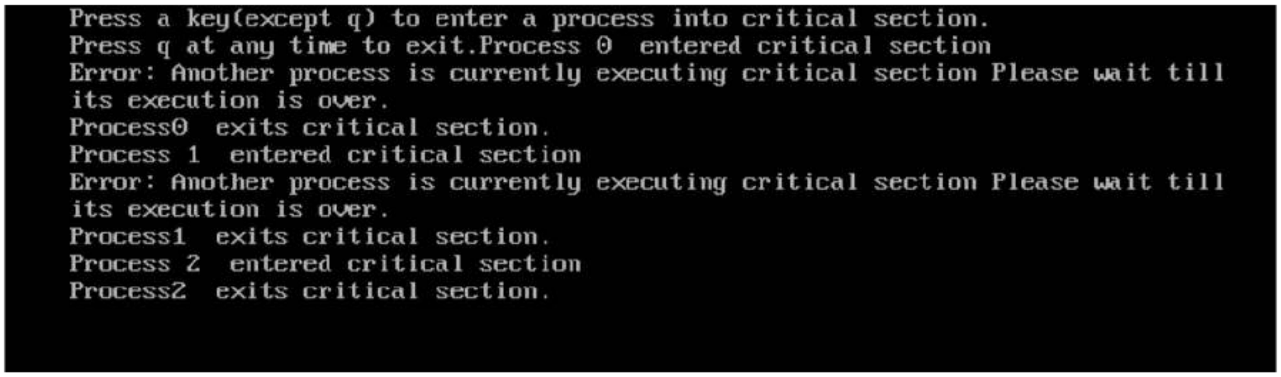
```
C:\Programacion\crypt_service\client\debug>client -ORBInitRef NameService=corbal
oc:iiop:localhost:8140/NameService

Cryptographic service client
----------------------------

Enter encryption key: 123
Enter a shift: 465
Enter a plain text to encrypt: En un lugar de la Mancha, de cuyo nombre no quier
o acordarme, no ha mucho tiempo que vivia un hidalgo de los de lanza astillero,
adarga antigua, rocín flaco y galgo corredor.
-----------------------------------------------

Encrypted text is: mDè=DèF=CI8èNMèFIèeIDOBIåèNMèO=1;èD;EH8MèD;è9=AM8;èIO;8NI8EMå
èD;èBIèE=OB;è>AME:;è9=Mè<A<      Iè=DèBANIFC;èNMèF;?èNMèFID0IèI?>AFFM8;åèINI8CIèI
D>AC=Iåè8;O      DèLFIO;è1èCIFC;èO;88MN;8ä
Decrypted text is: En un lugar de la Mancha, de cuyo nombre no quiero acordarme,
 no ha mucho tiempo que vivia un hidalgo de los de lanza astillero, adarga antig
ua, rocín flaco y galgo corredor.
-----------------------------------------------

Exit? (y/n): _
```



```
Press a key(except q) to enter a process into critical section.
Press q at any time to exit.Process 0  entered critical section
Error: Another process is currently executing critical section Please wait till
its execution is over.
Process0  exits critical section.
Process 1  entered critical section
Error: Another process is currently executing critical section Please wait till
its execution is over.
Process1  exits critical section.
Process 2  entered critical section
Process2  exits critical section.
```

CHAPTER 5

RESULT

The result of implementing a mutual exclusion mechanism using C is a distributed application that allows objects to communicate and exchange data across different machines in a network. By using the mutual exclusion middleware, developers can build distributed systems that are independent of the underlying hardware and operating system.

The implementation of a mutual exclusion mechanism using C involves several steps, including setting up the development environment, implementing the server and client applications, building and compiling the application, testing it, and documenting it. The choice of software packages and libraries used can affect the performance and scalability of the application.

One of the key benefits of using a mutual exclusion mechanism is its ability to allow objects to communicate seamlessly across different programming languages and platforms. This means that developers can create distributed systems that integrate different components built using different programming languages and platforms.

Another benefit of using mutual exclusion is the ability to scale distributed systems easily. Developers can addnew servers or clients to the system without having to modify the application's code. This makes it possible to handle large volumes of requests and improve the system's overall performance.

In conclusion, the implementation of a mutual exclusion mechanism using C can lead to the creation of a distributed system that is flexible, scalable, and independent of the underlying hardware and operating system. With proper design and implementation mutual exclusion -based applications can provide reliable and efficient communication between distributed objects, making them a valuable tool for building complex and mission-critical systems.

REFERENCES

1. Advanced C Programming Styles and Idioms by James O. Coplien
2. Object-Oriented Programming with C and OSF/Motif by Douglas A. Young
3. Object-Oriented Programming with /: Essential Concepts and Applications by Stephen J. Mellor
4. / Programming with C by Michi Henning and Steve Vinoski
5. The ACE ORB (TAO) User's Guide by Douglas C. Schmidt, Aniruddha Gokhale, and Irfan Pyarali
6. Object Management Group. (2021). / - Common Object Request Broker
7. The Object Management Group. (2021). IDL Data Types. Retrieved from https://www.omg.org/spec/IDL/4.2/PDF/
8. The Object Management Group. (2021). / Component Model. Retrieved from https:// https://www.studocu.com/in/document/delhi-technological-university/parallel-and-distributed-systems-lab/write-a-program-for-simulating-distributed-mutual-exclusion/19536736 www.omg.org/spec /4.2/PDF/
9. The Object Management Group. (2021). / Services. Retrieved from https://www.omg.org/spec /4.2/PDF/
10. https://www.studocu.com/in/document/delhi-technological-university/parallel-and-distributed-systems-lab/write-a-program-for-simulating-distributed-mutual-exclusion/19536736