University Of Stirling

CSCU9V6 – Operating Systems Concurrency and Distribution

Distributed Systems Assignment

CSCU9V6- Operating Systems Concurrency and Distribution

Assignment Name: Distributed Systems Assignment

Student Number: 3058038

Use of AI

I acknowledge that :

I used ChatGPT (https://chat.openai.com/) on 31/03/2024 to generate comments and resolve some errors in my code that are included within my submission

Presentation, structure, and proofreading

I acknowledge that:

I used ChatGPT (https://chat.openai.com/) on 02/04/2024 to fix the structure of my code

Introduction

Distributed systems often involve multiple processes accessing shared resources concurrently. This can lead to data inconsistency and race conditions if not properly coordinated. Distributed Mutual Exclusion (DME) is a technique used to ensure exclusive access to a critical section of code that modifies shared resources. This report describes an implementation of DME using a token ring algorithm.

The token ring algorithm utilizes a virtual token that circulates among participating processes. Only the process holding the token is allowed to enter the critical section. Other processes requesting access must wait until they receive the token. This report details the design and implementation of a DME system using the token ring approach.

This system consists of 7 classes:

- Node
- Coordinator
- C_mutex
- C_receiver
- C_connection_r
- C_buffer
- Logger

System Design

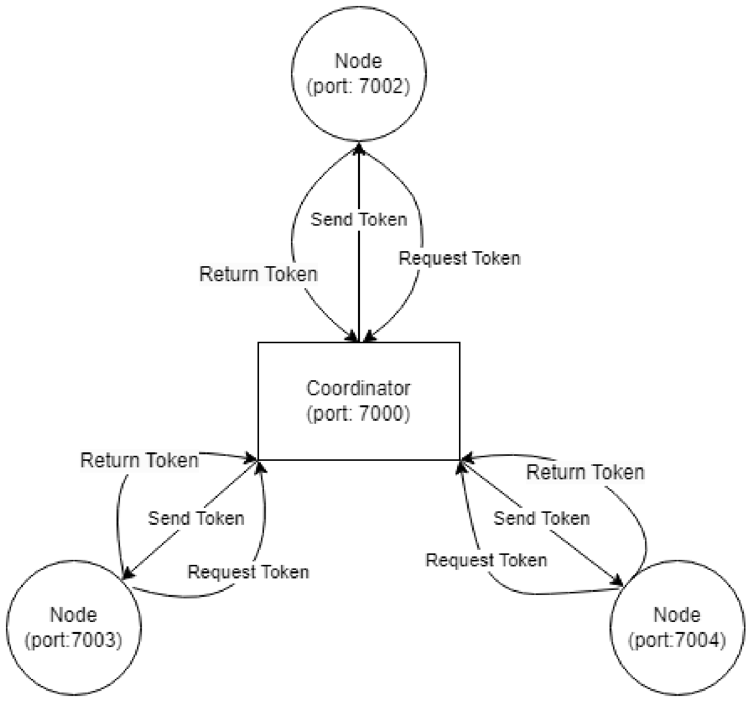
The system consists of two main components:

Coordinator: This process acts as a central authority responsible for managing the token and granting access to the critical section.

Node: This represents a process that wishes to enter the critical section to access a shared resource.

The interaction between these components can be summarized as follows:

1. A node sends a request message containing its hostname and port to the coordinator.
2. The coordinator grants the token to the requesting node, allowing it to enter the critical section.
3. The node executes its critical section operations and finishes accessing the shared resource.
4. The node returns the token back to the coordinator, signalling completion of its critical section.
5. The coordinator makes the token available to the next requesting node.



Assumptions

The following assumptions are made in this implementation:

- Reliable communication channels exist between nodes and the coordinator. Messages are not lost or corrupted during transmission.
- All nodes are aware of the coordinator's hostname and port for sending requests.

Code Description

The system is implemented using several Java classes:

Node.java:

This class represents a node process. It takes the node's hostname, port, critical section execution time (in milliseconds), and an optional shutdown command as arguments. The *Node* class performs the following tasks:

- Requests the token from the *coordinator*.
- Upon receiving the token, enters the critical section and simulates its execution for a random time.
- Returns the token back to the *coordinator* after finishing the critical section.
- Handles the optional shutdown command to gracefully terminate the program.

Coordinator.java:

This class represents the coordinator process. It starts the coordinator on a specified port, creates a shared buffer object to manage token requests and returns, and spawns separate threads:

- One thread listens for incoming token requests from nodes using *C_receiver.java*.
- Another thread manages token returns using *C_mutex.java*.

C_receiver.java & C_Connection_r.java:

These classes work together to handle incoming token requests:

- *C_receiver* listens for requests on a port and creates a new thread for each request using *C_Connection_r.*
- *C_Connection_r* receives the request details (hostname and port) from a node and stores them in the shared buffer.

C_mutex.java:

This class manages the token and grants access to the critical section:

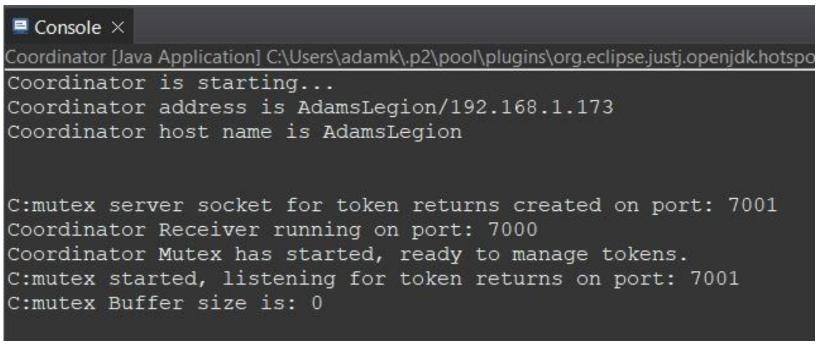
- It listens for token returns on a separate port.
- When a token return is received, it retrieves the corresponding request from the shared buffer.
- It grants the token to the requesting node by establishing a connection.
- It waits for the token to be returned by the node after the critical section execution.

C_buffer.java:

This class implements a synchronized buffer to manage token requests:

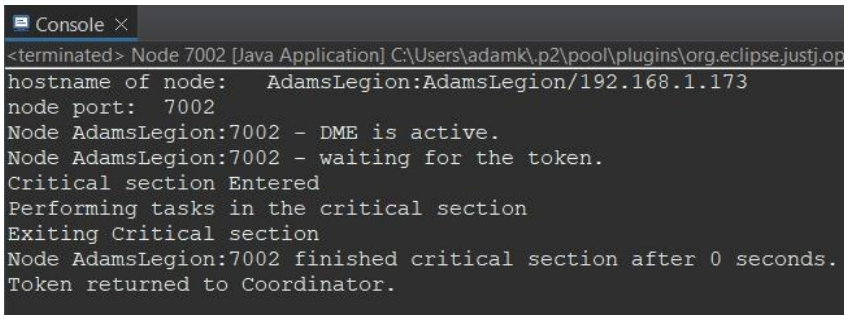
- It stores node requests (hostname and port) as string arrays.
- It provides methods for adding, removing, and accessing elements from the buffer in a thread-safe manner.
- It includes methods for checking buffer size and displaying its contents for debugging purposes.

Output:


Console
Coordinator [Java Application] C:\Users\adamk\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspo

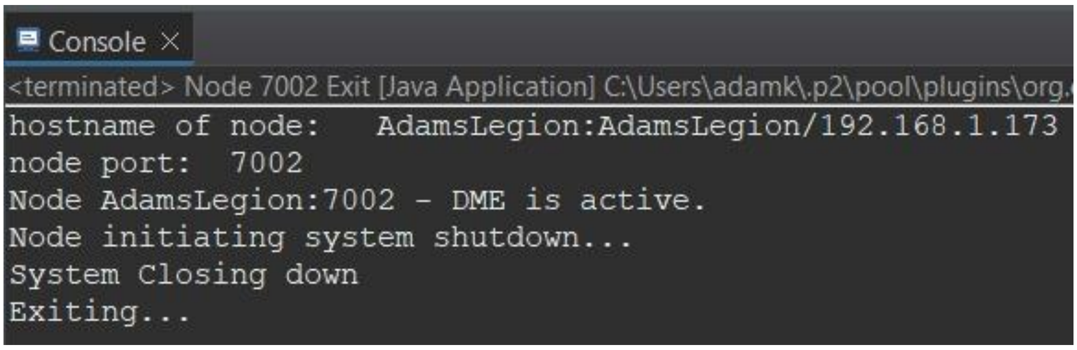
```
Coordinator is starting...
Coordinator address is AdamsLegion/192.168.1.173
Coordinator host name is AdamsLegion


C:mutex server socket for token returns created on port: 7001
Coordinator Receiver running on port: 7000
Coordinator Mutex has started, ready to manage tokens.
C:mutex started, listening for token returns on port: 7001
C:mutex Buffer size is: 0
```

This snippet shows Active Coordinator, listening for Token Requests from Nodes


Console
<terminated> Node 7002 [Java Application] C:\Users\adamk\.p2\pool\plugins\org.eclipse.justj.op

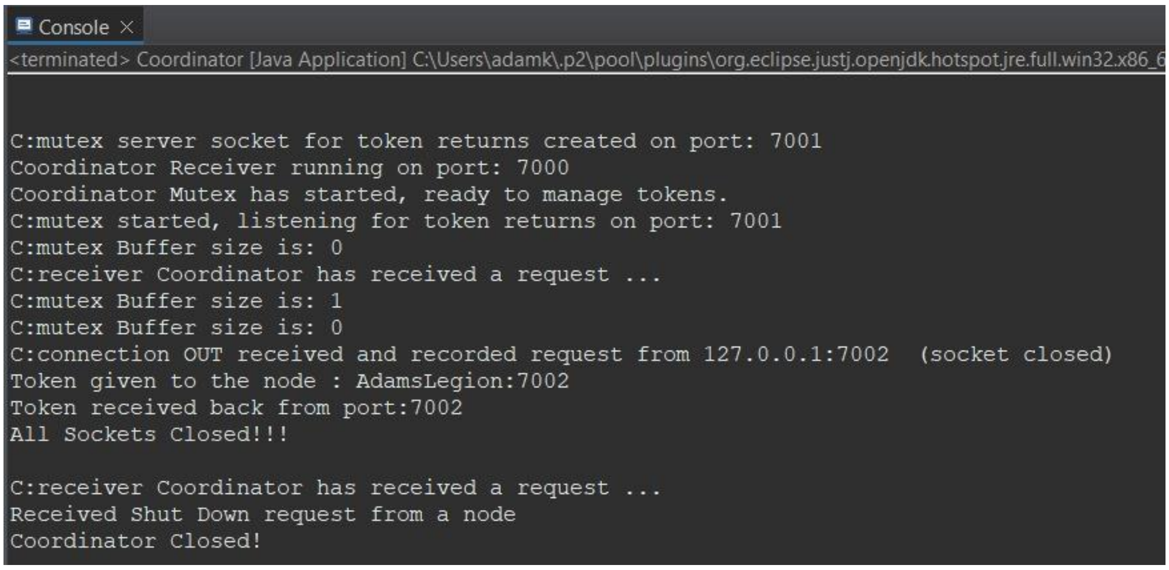
```
hostname of node:   AdamsLegion:AdamsLegion/192.168.1.173
node port:  7002
Node AdamsLegion:7002 - DME is active.
Node AdamsLegion:7002 - waiting for the token.
Critical section Entered
Performing tasks in the critical section
Exiting Critical section
Node AdamsLegion:7002 finished critical section after 0 seconds.
Token returned to Coordinator.
```

This snippet shows a Node requesting for Token to the Coordinator, after the token is granted, the Node can execute its Critical section



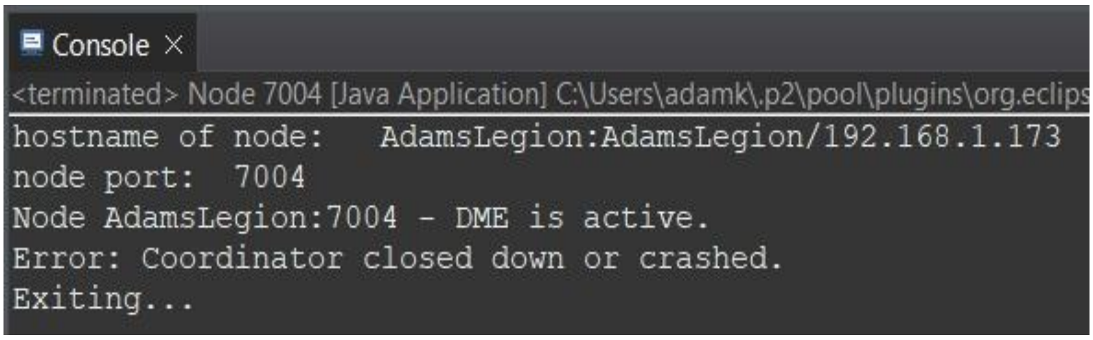
```
<terminated> Node 7002 Exit [Java Application] C:\Users\adamk\.p2\pool\plugins\org.
hostname of node:   AdamsLegion:AdamsLegion/192.168.1.173
node port:  7002
Node AdamsLegion:7002 - DME is active.
Node initiating system shutdown...
System Closing down
Exiting...
```

Above snippet shows a special Node that sends a system Shutdown instruction to the Coordinator



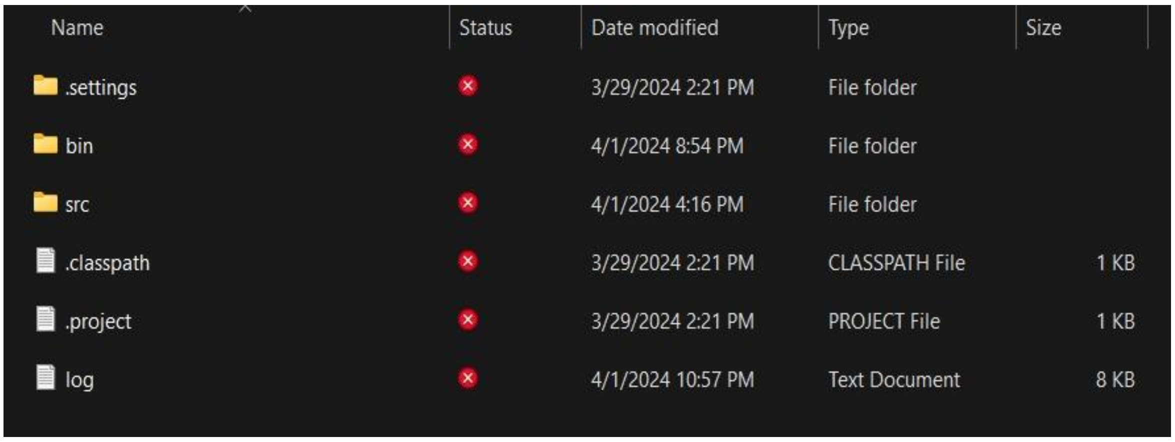
```
<terminated> Coordinator [Java Application] C:\Users\adamk\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_6


C:mutex server socket for token returns created on port: 7001
Coordinator Receiver running on port: 7000
Coordinator Mutex has started, ready to manage tokens.
C:mutex started, listening for token returns on port: 7001
C:mutex Buffer size is: 0
C:receiver Coordinator has received a request ...
C:mutex Buffer size is: 1
C:mutex Buffer size is: 0
C:connection OUT received and recorded request from 127.0.0.1:7002  (socket closed)
Token given to the node : AdamsLegion:7002
Token received back from port:7002
All Sockets Closed!!!

C:receiver Coordinator has received a request ...
Received Shut Down request from a node
Coordinator Closed!
```

This snippet shows the coordinator receiving a Shutdown instruction from a Node



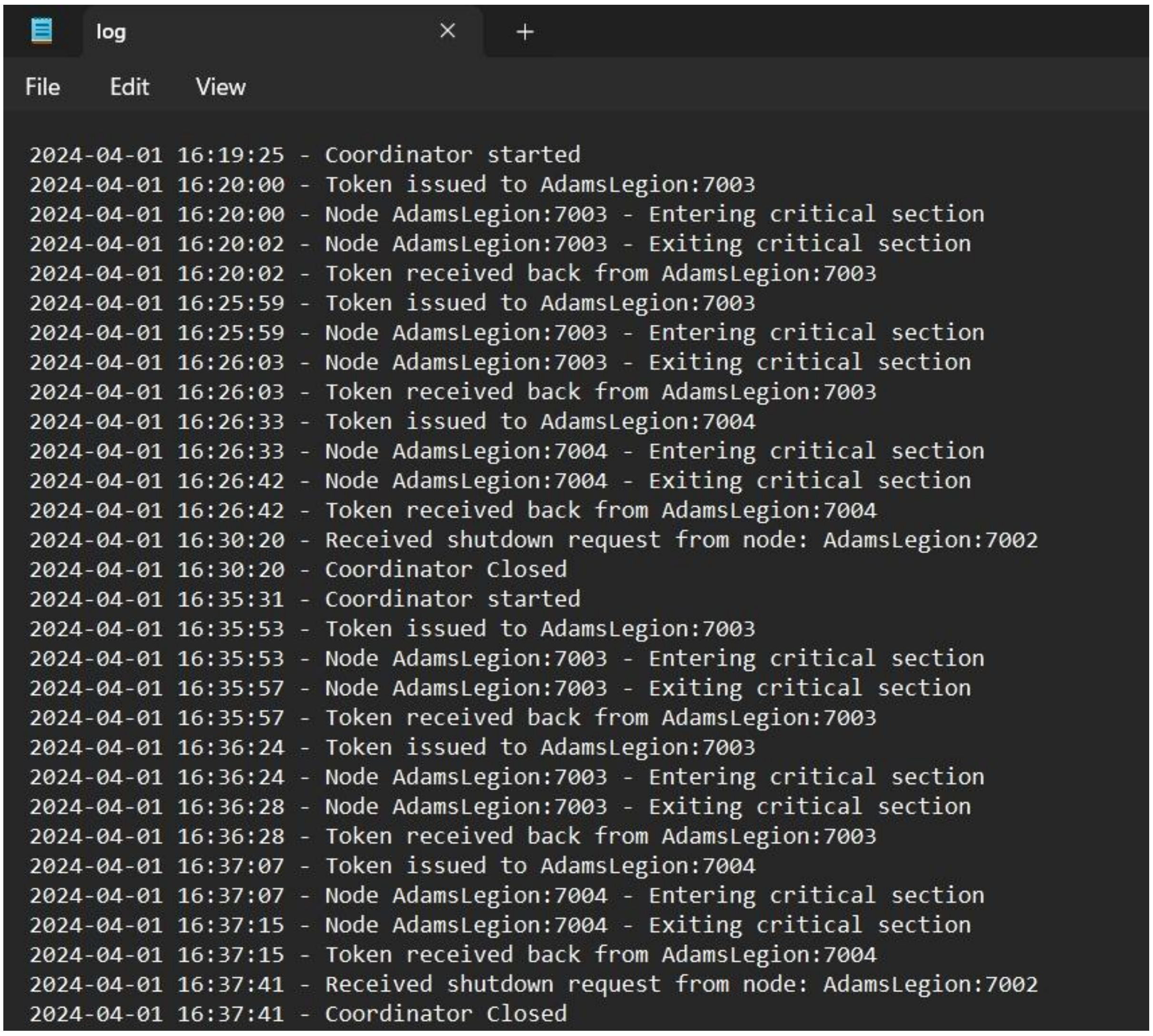
```
<terminated> Node 7004 [Java Application] C:\Users\adamk\.p2\pool\plugins\org.eclips
hostname of node:   AdamsLegion:AdamsLegion/192.168.1.173
node port:  7004
Node AdamsLegion:7004 - DME is active.
Error: Coordinator closed down or crashed.
Exiting...
```

This snippet shows the Node’s Status when Coordinator is closed



Name	Status	Date modified	Type	Size
.settings	✗	3/29/2024 2:21 PM	File folder	
bin	✗	4/1/2024 8:54 PM	File folder	
src	✗	4/1/2024 4:16 PM	File folder	
.classpath	✗	3/29/2024 2:21 PM	CLASSPATH File	1 KB
.project	✗	3/29/2024 2:21 PM	PROJECT File	1 KB
log	✗	4/1/2024 10:57 PM	Text Document	8 KB

Whenever the System starts a log txt file is created (if not already existing) in the working directory.



```
2024-04-01 16:19:25 - Coordinator started
2024-04-01 16:20:00 - Token issued to AdamsLegion:7003
2024-04-01 16:20:00 - Node AdamsLegion:7003 - Entering critical section
2024-04-01 16:20:02 - Node AdamsLegion:7003 - Exiting critical section
2024-04-01 16:20:02 - Token received back from AdamsLegion:7003
2024-04-01 16:25:59 - Token issued to AdamsLegion:7003
2024-04-01 16:25:59 - Node AdamsLegion:7003 - Entering critical section
2024-04-01 16:26:03 - Node AdamsLegion:7003 - Exiting critical section
2024-04-01 16:26:03 - Token received back from AdamsLegion:7003
2024-04-01 16:26:33 - Token issued to AdamsLegion:7004
2024-04-01 16:26:33 - Node AdamsLegion:7004 - Entering critical section
2024-04-01 16:26:42 - Node AdamsLegion:7004 - Exiting critical section
2024-04-01 16:26:42 - Token received back from AdamsLegion:7004
2024-04-01 16:30:20 - Received shutdown request from node: AdamsLegion:7002
2024-04-01 16:30:20 - Coordinator Closed
2024-04-01 16:35:31 - Coordinator started
2024-04-01 16:35:53 - Token issued to AdamsLegion:7003
2024-04-01 16:35:53 - Node AdamsLegion:7003 - Entering critical section
2024-04-01 16:35:57 - Node AdamsLegion:7003 - Exiting critical section
2024-04-01 16:35:57 - Token received back from AdamsLegion:7003
2024-04-01 16:36:24 - Token issued to AdamsLegion:7003
2024-04-01 16:36:24 - Node AdamsLegion:7003 - Entering critical section
2024-04-01 16:36:28 - Node AdamsLegion:7003 - Exiting critical section
2024-04-01 16:36:28 - Token received back from AdamsLegion:7003
2024-04-01 16:37:07 - Token issued to AdamsLegion:7004
2024-04-01 16:37:07 - Node AdamsLegion:7004 - Entering critical section
2024-04-01 16:37:15 - Node AdamsLegion:7004 - Exiting critical section
2024-04-01 16:37:15 - Token received back from AdamsLegion:7004
2024-04-01 16:37:41 - Received shutdown request from node: AdamsLegion:7002
2024-04-01 16:37:41 - Coordinator Closed
```

This snippet shows the log txt file, this file contains all the log records with time stamps

Critical Reflection

This implementation of DME using the token ring algorithm ensures exclusive access to the critical section for each node.

Strengths:

- Ensures serialized access to the critical section, preventing race conditions and data inconsistency.
- Relatively simple to understand and implement.

Limitations:

- Relies on reliable communication channels. Lost or corrupted messages can disrupt

Future Scope

- **Scalability Enhancements:** Explore alternative DME algorithms like Lamport's Bakery Algorithm or Bully Algorithm for better scalability when dealing with a large number of nodes. These algorithms can reduce message overhead compared to the token ring approach.
- **Fault Tolerance Mechanisms:** Implement mechanisms to handle node failures or network disruptions. This could involve timeouts for waiting nodes, leader election protocols for coordinator failures, or heartbeat messages to detect unresponsive nodes.
- **Performance Optimization:** Analyze the performance bottlenecks and explore optimizations. Consider techniques like adapting the token circulation strategy or exploring dynamic buffer sizing in the coordinator for improved efficiency.
- **Security Integration:** Integrate security measures like authentication and authorization to ensure only authorized nodes can access critical sections and prevent unauthorized access to shared resources.
- **Distributed Deadlock Detection:** Investigate methods for detecting and resolving deadlocks that might arise in complex scenarios involving multiple critical sections and dependencies between nodes.