AOS Project – Phase 1

Project Members:

- Sruthi Chappidi (sxc105920)
- Ramya Naraparedi (rxn106120)
- Diksha Sharma (dxs134530)
- Kathryn Patterson (knp061000)

Introduction

The purpose of the project is to design a new distributed mutual exclusion algorithm by combining quorum based and token based approaches and compare it to Maekawa's algorithm.

Project Description (From Project Requirements Document)

The new distributed mutual exclusion algorithm adheres to the following:

- The algorithm is quorum-based. Like Maekawa's algorithm, algorithm messages are only be sent between members of the same quorum. The maximum quorum size is $2\sqrt{N}$ where N = total number of nodes is in the system.
- The algorithm is token-based. Like Raymond's algorithm, mutual exclusion is granted once the requesting node has the token. Only one token is used, and it can only be passed with an algorithm message (which restricts the token from being passed between nodes that are not in the same quorum).
- Our design includes code to clearly explain the logic behind the algorithm. The code has been written as pseudo code in Java.
- Our design includes an example of the algorithm when two nodes make requests at the same time. The example is diagrammed and we show all messages passed during the algorithm's operation. Since this is a distributed mutual exclusion algorithm, one node's request will obviously be delayed, but eventually is granted.

In addition to designing a new algorithm, our team is designed a testbed for comparing the new algorithm to Maekawa's algorithm while adhering to the following:

- The testbed is configurable, with the ability to indicate which nodes make requests and when.
- The testbed is able to support a minimum of 16 system nodes for both algorithms.

- The testbed is able to demonstrate that mutual exclusion is not violated.
- Our design must include a description and diagram of how the testbed will be implemented.

What is Mutual Exclusion?

In order for mutual exclusion to be satisfied in a system, a critical resource granted to a process is released before it can be granted to another process. There are two conditions for distributed **mutual exclusion**.

- Requests for a resource is granted in the order in which they were made.
- Every granted resource must eventually be released, to ensure every request will eventually be granted.

In the proposed algorithm, mutual exclusion is primarily granted via the use of a single token in the distributed system. In order to enter critical section, a node should become the holder of the token.

Requests are granted in the order they are made through the use of a first in first out (FIFO) queue of requests.

Starvation is a condition that occurs when one process must wait indefinitely to enter its critical section even though other nodes are entering and exiting their own critical sections.

The FIFO system for requests prevents a starvation condition, by ensuring that all processes are eventually granted their requests as they progress in the queue.

Furthermore, all processes are treated fairly because of the lack of prioritization for requests. In addition to the use of the FIFO queue, a sequence number per node is used so that ties between multiple requests can be prioritized and processed.

The proposed algorithm also uses a quorum based organization structure, which further ensures **fairness** for all processes so all processes have equal opportunity to get into critical section.

Deadlock is a condition that occurs when no process in the system is in its critical section and no requesting process can ever proceed to its own critical section. The use of a single token prevents deadlock from occurring in the proposed algorithm. The holder of the token will always grant another new process the token if a request is made, regardless of the number or nature of requests.

Test Bed Configuration

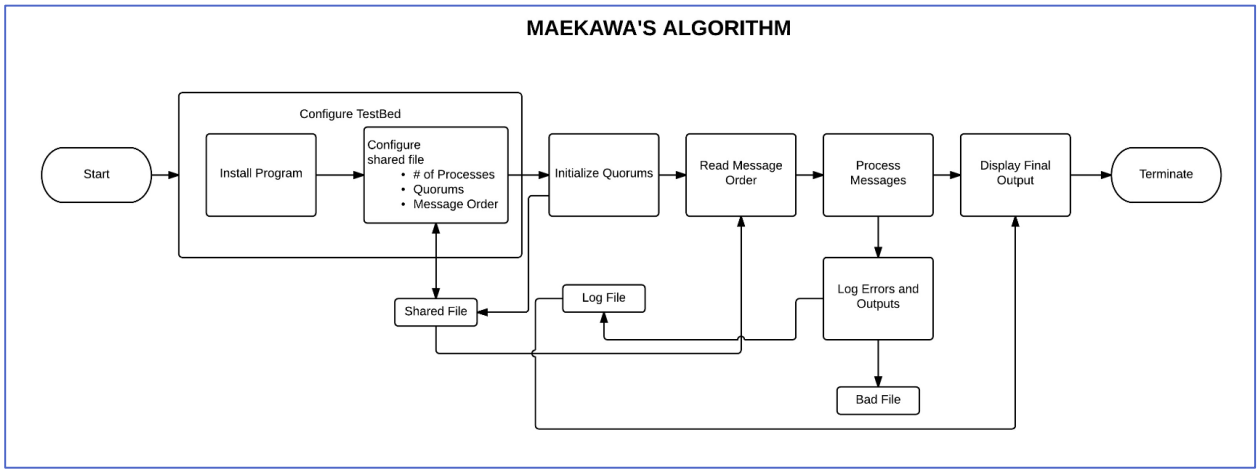
- Process ids are passed to each process instance as part of the argument list.
- The processes each access the shared text file "Config.txt" in the shared location for the quorums for each of them and also the request order for the critical section.
- Configuration File Structure is as below:

```
Quorum
{1,2,3}
{2,3,4}
{3,4,1}
{4,2,1}
Message_Order
1
3
4
2
1
3
4
2
3
4
2
1
3
4
TERMINATE
```

- Same configuration file structure works for both Maekawa's algorithm and our algorithm.
- Quorums are listed after the title "Quorum" for each process.
- The order in which the processes request for the critical section follow the "Message_Order" title.
- The processes will stop sending more requests for the critical section once they encounter the "TERMINATE" in the shared file.
- There should be no blank lines. Program does not handle blank lines and will crash.
- Path of the shared file, log file and bad file should be passed as arguments.
- Log file will log all details like receiving, sending, popping etc.
- Bad file will store all failures.
- Processes read both log file and shared file for the request order and to find which process requested for critical section, executed critical section etc.
- Log file is defaulted to "Log.txt"
- Bad file is defaulted to "Bad.txt"

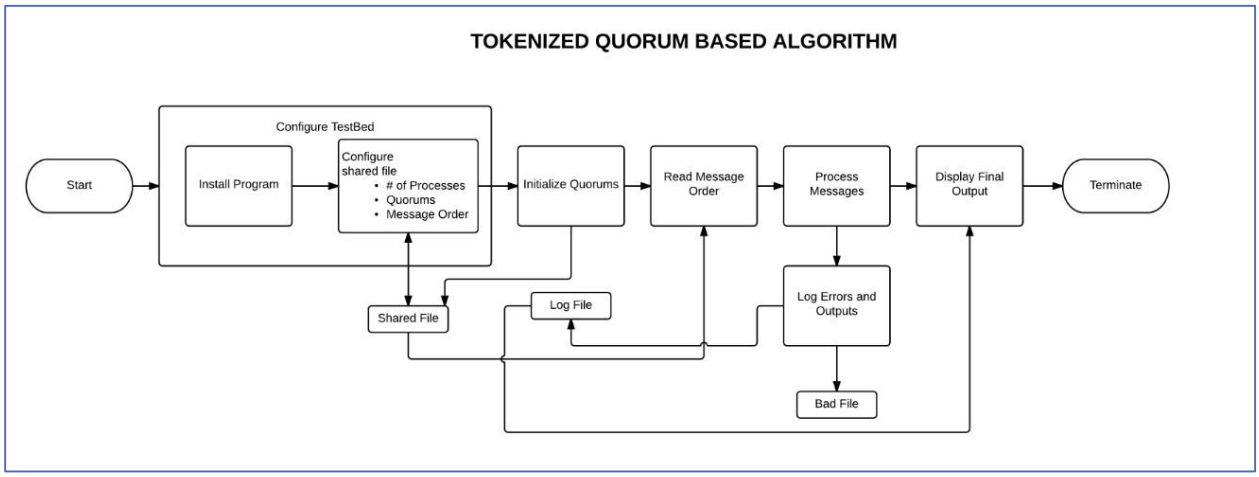
- Log file and bad file get renamed before any step for each run so as to avoid overwriting previous run's files.
- Program does not handle purging of the old log and bad files.

Maekawa's Algorithm

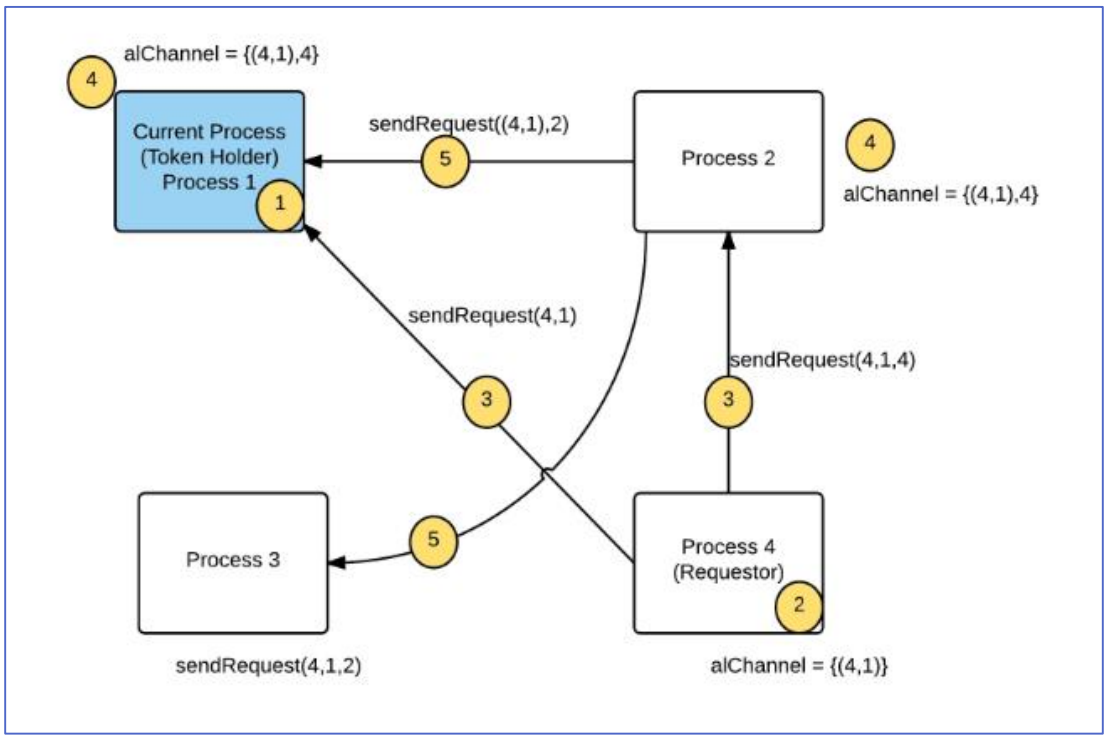


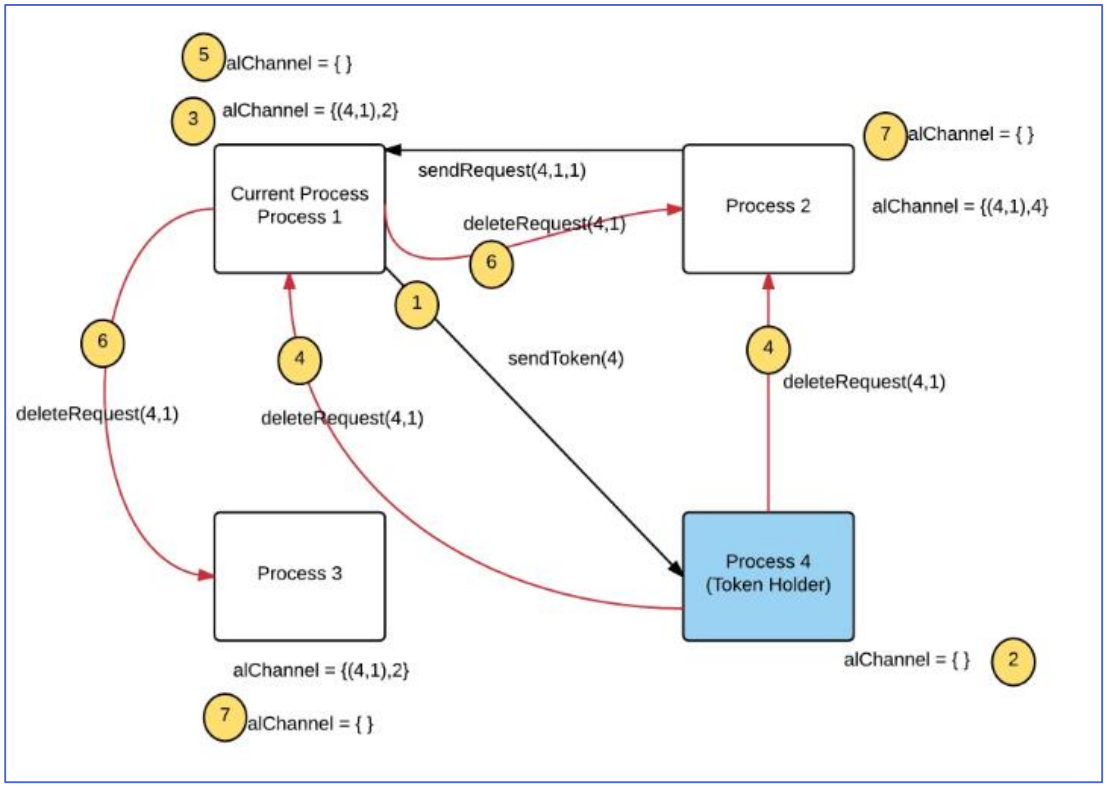
Tokenized Quorum Based Algorithm

Envisioned System:



Process Messages Stage is demonstrated in detail by the steps below:





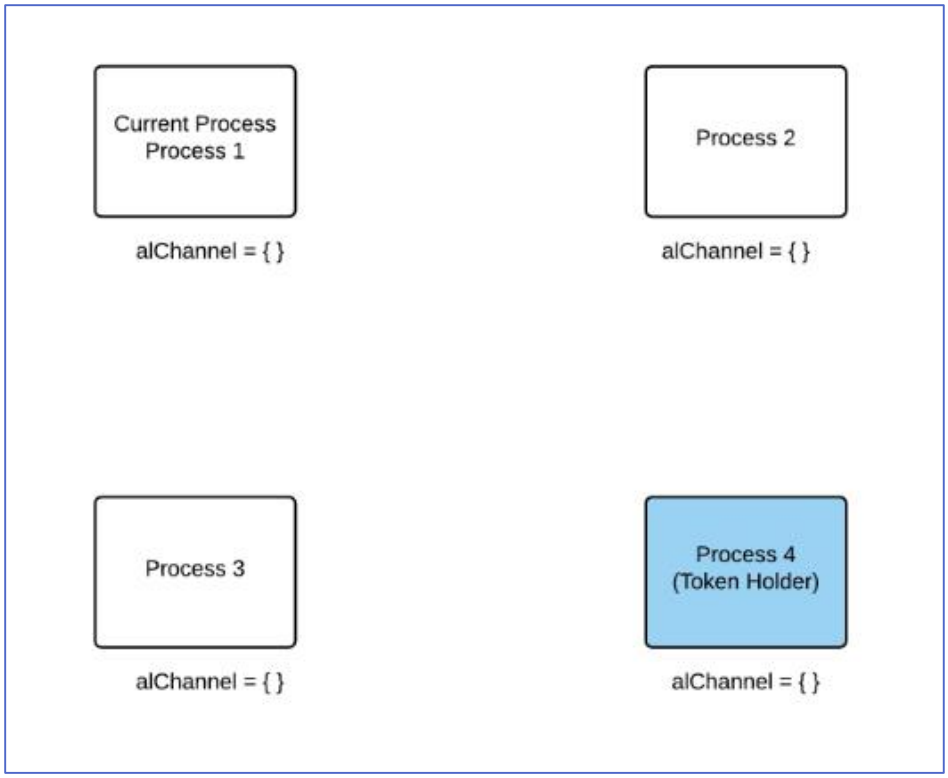
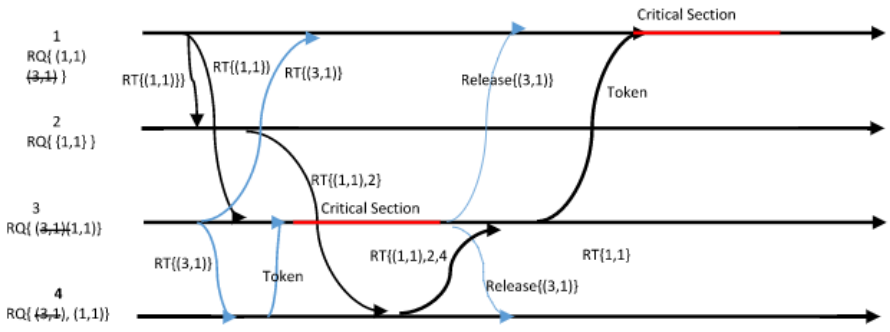


Diagram depicting competing requests:

When a request for critical section is made the message will appear like {1,1}.The first number is the initiator of the request the last number is the sequence number.

This example will show what happens when process 1 and 3 making requests and the original holder is 4:

In this case we must also assume the quorums as well 1: {1, 2, 3} 2: {1, 2, 4} 3: {3, 4, 1} 4: {3, 4, 2}



One should note here are that when process three gets message RT{(1,1),2,4} it recognizes that it is in its own queue already and drops the message.

Algorithm Steps:

1. Initializing: (Each process will do this set of steps)
    a. Read the shared file.
    b. Initialize the quorum.
    c. Set the size of the quorum.
2. Requesting a Token:
    a. Read the shared file for the request order.
    b. Request the token when the next request is for the current process.
    c. Add the request to the request queue.
    d. Send message to the quorum members for token.
3. Receiving a Request:
    a. Update the sequence id.
    b. Add the request to the queue.
    c. Check if the current process is the token holder and is not using the critical section.
        i. Assign the token to the process at the head of the FIFO queue.
        ii. If the current process is the holder then execute critical section.
        iii. If not send the token to the process requesting the token.
    d. If the current process does not have the token and needs to send a request to it quorum members then send request to the quorum members.
4. Receiving the Token:
    a. Sort the queue by sequence id and in case of ties sort by process id.
    b. If the queue is not empty
        i. Assign the first process as the holder.
        ii. If the current process is the holder then execute the critical section.
        iii. Else send the token to the holder.
        iv. If the queue is not empty then send the request for token to the holder.
    c. Send a broadcast delete request to all the quorum members to delete the previously sent request for token.
5. Sending the Token:
    a. Set the current process as not the possessor of the token.
    b. Send token to the holder.

Deadlock:

The algorithm is single token based system. At any time only one process will possess the token to enter into its critical section which it should release to other processes once it is done with its critical section. The system initially allocates token to the process id 1. This resolves any deadlock issues.

Starvation:

Each request is associated with the sequence number. We update the sequence number every time a new request is received by the current process. The requests are ordered in the non-decreasing order of the sequence number. The smaller the sequence number the higher the priority of the request. In case of any ties the algorithm relies on the process ids to resolve them. This makes it starvation free.

Fairness:

The algorithm addresses all requests in the order of their non-decreasing sequence numbers and so will always address a request made first than the one that reached the process first. In case of any ties the algorithm relies on the process ids to resolve them. Thus it is fair to the request order.

Pseudo Code

Pseudo code for Maekawa's algorithm and our design of the tokenized quorum based algorithm are attached in the zip file submitted.

Maekawa's Algorithm: Maekawa_Algorithm.java

Tokenized Quorum Based Algorithm: Tokenized Quorum_Based Algorithm.java

Performance Metrics

The algorithm will collect following metrics to measure performance:

1. Number of messages exchanged for the request order – Message Complexity
2. No deadlocks - Deadlock is a condition that occurs when no process in the system is in its critical section and no requesting process can ever proceed to its own critical section. The use of a single token prevents deadlock from occurring in the proposed algorithm. The holder of the token will always grant another new process the token if a request is made, regardless of the number or nature of requests.
3. No starvation - Starvation is a condition that occurs when one process must wait indefinitely to enter its critical section even though other nodes are entering and exiting their own critical sections. The FIFO system for requests prevents a starvation condition, by ensuring that all processes are eventually granted their requests as they progress in the queue.
4. Fairness - The fairness property ensures that process requests are processed in the order in which they are made.

JIRA

Rather than using JIRA for project management we used a shared document on Google Drive to maintain tasks and responsibilities for this project.

We had regular meetings, where we discussed the progress of each major task and improved on each other's work. After every meeting we would create a list of action items that would be due at the next meeting. These tasks, action items, were also tracked on a document along with meeting minutes on the google document.