Module 6

Distributed Mutual Exclusion

DISTRIBUTED MUTUAL EXCLUSION

Ques 1) What is distributed mutual exclusion? Discuss about the distributed mutual exclusion algorithm.

Ans: Distributed Mutual Exclusion
A way of making sure that if one process is using a shared modifiable data, the other processes will be excluded from doing the same thing. Formally, while one process executes the shared variable, all other processes desiring to do so at the same time moment should be kept waiting; when that process has finished executing the shared variable, one of the processes waiting; while that process has finished executing the shared variable, one of the processes waiting to do so should be allowed to proceed. In this fashion, each process executing the shared data (variables) excludes all others from doing so simultaneously. This is called **mutual exclusion.**

Mutual exclusion needs to be enforced only when processes access shared modifiable data - when processes are performing operations that do not conflict with one another they should be allowed to proceed concurrently.

Distributed processes often need to coordinate their activities. If a collection of processes share a resource or collection of resources, then often mutual exclusion is required to prevent interference and ensure consistency when accessing the resources. This is the critical section problem. In a distributed system, however, neither shared variables nor facilities supplied by a single local kernel can be used to solve it, in general. One requires a solution to distributed mutual exclusion: one that is based solely on message passing.

Distributed mutual exclusion algorithms must deal with unpredictable message delays and incomplete knowledge of the system state.

In some cases shared resources are managed by servers that also provide mechanisms for mutual exclusion –how some servers synchronize client accesses to resources. But in some practical cases, a separate mechanism for mutual exclusion is required.

Distributed Mutual Exclusion Algorithm
Ricart & Agrawala came up with a distributed mutual exclusion algorithm in 1981. It requires the following:
1) Total ordering of all events in a system (e.g. Lamport's algorithm or others).
2) Messages are reliable (every message is acknowledged).

When a process wants to enter a critical section, it:
1) Composes a message containing its identifier (machine, proc#), name of critical section, current time.
2) Sends a request message to all other processes in the group (may use reliable group communication).
3) Wait until everyone in the group has given permission.
4) Enter the critical section.

When a process receives a request message, it may be in one of three states:
Case 1) The receiver is not interested in the critical section, send reply (OK) to sender.

Case 2) The receiver is in the critical section; do not reply and add the request to a local queue of requests.

Case 3) The receiver also wants to enter the critical section and has sent its request. In this case, the receiver compares the timestamp in the received message with the one that it has sent out. The earliest timestamp wins. If the receiver is the loser, it sends a reply (OK) to sender. If the receiver has the earlier timestamp, then it is the winner and does not reply. Instead, it adds the request to its queue.

When the process is done with its critical section, it sends a reply (OK) to everyone on its queue and deletes the processes from the queue.

For example, let us consider **figure 6.1** which deals with contention. Here, two processes 0 and 2. request access to the same resource (critical section). Process 0 sends its request with a timestamp of 8 and process 2 sends its request with a timestamp of 12 as shown **figure 6.1(a).**

Since process 1 is not interested in the critical section, it immediately sends back permission to both 0 and 1. Process 0 is interested in the critical section. It sees that process 2 timestamp was later than its own, so process 0 wins. It queues request as shown in **figure 6.1 (b).**

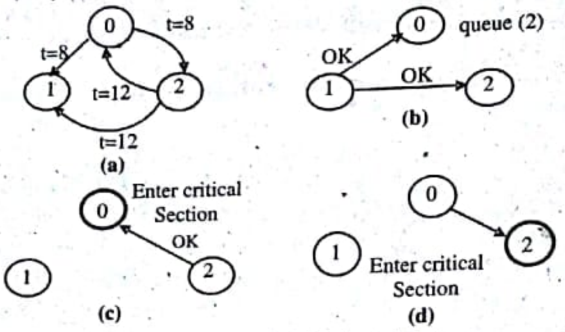

Figure 6.1: Distributed Mutual Exclusion

Process 2 is also interested in the critical section. When it compares its message's timestamp with that of the message it received from process 0, it sees that it lost, so it replies with permission to process 0 and continues to wait for everyone to give it permission to enter the critical section as shown in **figure 6.1 (c).**

As soon as process 2 send process 0 permission, process 0 received permission from the entire group and can enter the critical section. When process 0 is done with the critical section. It examines its queue of pending permissions, finds process 2 in that queue, and sends it permission to enter the critical section as shown in **figure 6.1 (d).** Now process 2 has received permission from everyone and can enter the critical section.

One problem with this algorithm is that a single point of failure has now been replaced with n points of failure. A poor algorithm has been replaced with one that is essentially n times worse. All in not lost. We can patch this omission up by having the sender always send a reply to a message… either an OK or a NO. When the request or the reply is lost, the sender will time out and retry. Still, it is not a great algorithm and involves quite a bit of message traffic but it demonstrates that a distributed algorithm is at least possible.

Ques 2). What are the key properties of a useful mutual exclusion algorithm?

Ans: Properties of Mutual Exclusion Algorithm

1) **Safety:** At most one process may execute in the critical section at a time.
2) **Liveness:** Requests to enter and exit the critical section eventually succeed. This implies freedom from deadlock and starvation.
3) **Ordering:** If one requests to enter the critical section happened-before another then entry to the critical section is granted in that order.

Ques 3) Describe the Central-server algorithm of distributed mutual exclusion.

Ans: Central-Server Algorithm

The central server algorithm simulates a single processor system. One of the processes in the distributed system is first elected as the coordinator (**figure 6.2**).

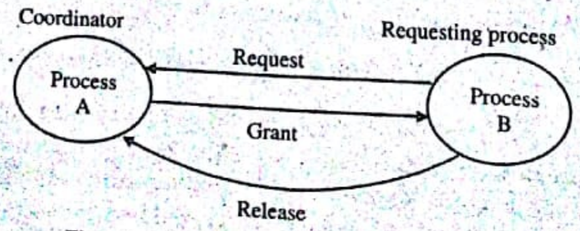

Figure 6.2: Centralized Mutual Exclusion

When a process needs to enter a critical section, it sends a request (with the identification of the critical section) to the coordinator. If no other process is currently in the critical section, the coordinator sends back a grant and marks that process as using the critical section. If another process has already claimed the critical section, the server does not reply, and hence the requesting process gets blocked and enters the queue of processes, requesting that critical section. When a process exits the critical section, it sends a release to the coordinator. The coordinator then sends a grant to the next process in the queue of processes, waiting for the critical section. This algorithm is fair, in that it processes the requests in the order they are received, and easy to implement and verify.

The major **drawback** is that the coordinator becomes the single point of failure. The centralized server can also become a bottleneck in the system.

Ques 4). Explain the Ring-Based Algorithm.

Ans: Ring-Based Algorithm

A completely different approach to achieving mutual exclusion in a distributed system is illustrated in **figure 6.3**:

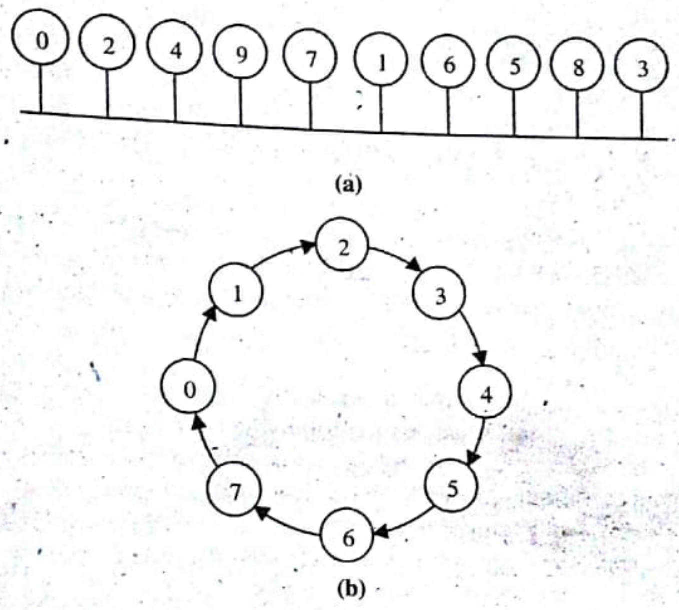

Figure 6.3: (a) Unordered Group of Processes on a Network. (b) Logical Ring Constructed in Software

Consider a bus network, as shown in **figure 6.3(a)**, (e.g., Ethernet), with no inherent ordering of the processes. In software, a logical ring is constructed in which each process is assigned a position in the ring, as shown in **figure 6.3 (b)**. The ring positions may be allocated in numerical order of network addresses or some other means. It does not matter what the ordering is. All that matters is that each process knows who is next in line after itself.

When the ring is initialised, process 0 is given a token. The token circulates around the ring. It is passed from process k to process k + 1 (modulo the ring size) in point-to-point messages. When a process acquires the token from its neighbour, it checks to see if it is attempting to enter a critical region. If so, the process enters the region, does all the work it needs to, and leaves the region. After it has exited, it passes the token along the ring. It is not permitted to enter a second critical region using the same token.

If a process is handed the token by its neighbour and is not interested in entering a critical region, it just passes it along. As a consequence, when no processes want to enter any critical regions, the token just circulates at high speed around the ring.

The correctness of this algorithm is easy to see. Only one process has the token at any instant, so only one process can actually be in a critical region. Since the token circulates among the processes in a well-defined order, starvation cannot occur. Once a process decides it wants to enter a critical region, at worst it will have to wait for every other process to enter and leave one critical region.

A problem is occurred in this algorithm i.e., if the token is ever lost, it must be regenerated. In fact, detecting that it is lost is difficult, since the amount of time between successive appearances of the token on the network is unbounded. The fact that the token has not been spotted for an hour does not mean that it has been lost; somebody may still be using it.

The algorithm also runs into trouble if a process crashes, but recovery is easier than in the other cases. If one requires a process receiving the token to acknowledge receipt, a dead process will be detected when its neighbour tries to give it the token and fails. At that point the dead process can be removed from the group, and the token holder can throw the token over the head of the dead process to the next member down the line, or the one after that, if necessary. Doing so requires that everyone maintains the current ring configuration.

Ques 5) Explain Maekawa's voting algorithm.

Ans: Maekawa's Voting Algorithm
Maekawa's algorithm is an algorithm for mutual exclusion on a distributed system. The basis of this algorithm is a quorum like approach where any one site needs only to seek permissions from a subset of other sites.

Maekawa's algorithm is a departure from the general trend in the following two ways:

1) First, a site does not request permission from every other site, but only from a subset of the sites. This is a radically different approach as compared to the **lamport** and the **Ricart-Agrawala** algorithms, where all sites participate in the conflict resolution of all other sites. In Maekawa's algorithm, the request set of each site is chosen such that $\forall i\ \forall j: 1 \le i, j \le N :: R_i \cap R_j \ne \phi$. Consequently, every pair of sites has a site that mediates conflicts between that pair.

2) Second, in Maekawa's algorithm a site can send out only one REPLY message at a time. A site can only send a REPLY message only after it has received a RELEASE message for the previous REPLY message. Therefore, a site S_i locks all the sites in R_i in exclusive mode before executing its CS(Critical Section).

The Construction of Request Sets
Maekawa's algorithm was the first quorum–based mutual exclusion algorithm. The request sets for sites (i.e., quorums) in Maekawa's algorithm are constructed to satisfy the following conditions:
M1 $(\forall_i \forall_j: i \ne j, 1 \le i, j \le N :: R_i \cap R_j \ne \phi)$.
M2 $(\forall_i: 1 \le i \le N :: S_i \in R_i)$.
M3 $(\forall_i: 1 \le i \le N :; [R_i] = K)$,
M4 Any site S_j is contained in K number of R_is, $1 \le i, j \le N$.

Maekawa used the theory of projective planes and showed that $N = K(K-1) + 1$. This relation gives $|R_i| = \sqrt{N}$.

Since there is at least one common site between the request sets of any two sites (condition M1), every pair of sites has a common site which mediates conflicts between the pair. A site can have only one outstanding REPLY message at any time; that is, it grants permission to an incoming request if it has not granted permission to some other site. Therefore, mutual exclusion is guaranteed. This algorithm requires delivery of messages to be in the order they are sent between every pair of sites.

Conditions M1 and M2 are necessary for correctness; whereas conditions M3 and M4 provide other desirable features to the algorithm. Condition M3 states that the size of the request sets of all sites must be equal, which implies that all sites should have to do an equal amount of work to invoke mutual exclusion. Condition M4 enforces that exactly the same number of sites should request permission from any site which implies that all sites have "equal responsibility" in granting permission to other sites.

In Maekawa's algorithm, a site S_i executes the steps shown in following **Algorithm** to execute the CS.

Algorithm: Maekawa's Algorithm
Maekawa's algorithm works in the following manner:

1) **Requesting the Critical Section**
   i) A site S_i requests access to the CS by sending REQUEST (i) messages to all sites in its request set R_i.
   ii) When a site S_j receives the REQUEST (i) message, it sends a REPLY (j) message to S_i provided it hasn't sent a REPLY message to a site since its receipt of the last RELEASE message. Otherwise, it queues up the REQUEST (i) for later consideration.

2) **Executing the Critical Section:** Site S_i executes the CS only after it has received a REPLY message from every site in R_i.

3) **Releasing the Critical Section**
   i) After the execution the CS is over, site S_i sends a RELEASE(i) message to every site in R_i.
   ii) When a site S_j receives a RELEASE (i) message from site S_i, it sends a REPLY message to the next site waiting in the queue and deletes that entry from the queue. If the queue is empty, then the site updates its state to reflect that it has not sent out any REPLY message since the receipt of the last RELEASE message.

Ques 6) Prove that Maekawa's algorithm achieves mutual exclusion.

Ans: We proof the algorithm by contradiction. Suppose two sites S_i and S_j are concurrently executing the CS. This means site S_i received a REPLY message from all sites in R_i and concurrently site S_j was able to receive a REPLY message from all sites in R_j. If $R_i \cap R_j = \{S_K\}$, then site S_k must have sent REPLY messages to both S_i and S_j concurrently, which is a contradiction.

Ques 7) Discuss the problem of deadlocks in Maekawa's algorithm and how to handle it.

Ans: The Problem of Deadlocks
Maekawa's algorithm is prone to deadlocks because a site is exclusively locked by other sites and requests are not prioritized by their timestamps. Without the loss of generality, assume three sites S_i, S_j, and S_k simultaneously invoke mutual exclusion. Suppose $R_i \cap R_j = \{S_{ij}\}$, $R_j \cap R_K = \{S_{jk}\}$, and $R_k \cap R_j = \{S_{ki}\}$,).

Since sites do not send REQUEST message to the sites in their request sets in any particular order, it is possible that due to arbitrary message delays, S_{ij} has been locked by S_i (forcing S_j to wait at S_{ij}), S_{jk} has been locked by S_j (forcing S_k to wait at S_{jk}), and S_{ki} has been locked by S_k (forcing S_i to wait at S_{ki}) resulting in a deadlock involving the sites S_i, S_j, and S_k.

Handling Deadlocks
Maekawa's algorithm handles deadlocks by requiring a site to yield a lock lock if the timestamp of its request is larger than the timestamp of some other request waiting for the same lock (unless the former has succeeded in locking all the needed sites). A site suspects a deadlock (and initiates message exchanges to resolve it) whenever a higher priority request finds that a lower priority request has already locked the site. Deadlock handling requires following three of message:

1) **Failed:** A FAILED message from site S_t to site S_j indicates that S_i cannot grant S_j's request because it has currently granted permission to a site with a higher priority request.

2) **Inquire:** An INQUIRE message from S_i to S_j indicates that S_i would like to find out from S_j if it has succeeded in locking all the sites in its request set.

3) **Yield:** A YIELD message from site S_i to S_j indicates that S_i is returning the permission to S_j (to yield to a higher priority request at S_j).

Details of the deadlock handling steps are as follows:

1) When a REQUEST (ts, i) from site S_i blocks at site S_j because S_j has currently granted permission to site S_k, then S_j sends a FAILED(j) message to S_i if S_i's request has lower priority. Otherwise, S_j sends an INQUIRE(j) message to site S_k.

2) In response to an INQUIRE (j) message from site S_j, site S_k sends a YIELD(k) message to S_j, provided, S_k has received a FAILED message from a site in its request set or if it sent a YIELD to any of these sites, but has not received a new REPLY from it.

3) In response to a YIELD (k) message from site S_k, site S_j assumes it has been released by S_k, places the request of S_k at the appropriate location in the request queue, and sends a REPLY(j) to the top request's site in the queue.

Thus, Maekawa type algorithms require extra messages to handle deadlocks and may exchange these messages even though there in no deadlock. The maximum number of messages required per CS execution in this case is $5\sqrt{N}$.

ELECTION

Ques 8) What do you understand by elections? What are the assumptions of election algorithm?

Ans: Elections
The functions like replacing token controlling I/O device in the system, enforcing mutual exclusion are performed by a process called co-ordinator. The processes require one process to act as a co-ordinator. The processes require one process to act as a co-ordinator to perform these functions. Election algorithm elects co-ordinator from processes.

An algorithm for choosing a unique process to play a particular role is called an election algorithm.

There are two election algorithms are as follows:
1) The Bully Algorithm
2) A Ring Algorithm

Assumptions of Election Algorithm
The assumptions of election algorithm are as follows:
1) Unique number is assigned to each process. This will help to distinguish the process from all other processes having same characteristic and are exactly same. So for this we assume that there is one to one correspondence between processes and machine. As co-ordinator has highest number, election algorithm try to locate process with highest number.
2) Every process knows process number of every other process.

Ques 9) Write the Bully algorithm and analyse them.
Or
What is assumption of bully algorithm?

Ans: Bully Algorithm
In distributed computing, the bully algorithm is a method for dynamically electing a coordinator or leader from a group of distributed computer processes. The process with the highest process ID number from amongst the non-failed processes is selected as the coordinator.

Assumption of Bully Algorithm are as follows:
1) The system is synchronous.
2) Processes may fail at any time, including during execution of the algorithm.
3) A process fails by stopping and returns from failure by restarting.
4) There is a failure detector which detects failed processes.
5) Message delivery between processes is reliable.
6) Each process knows its own process id and address, and that of every other process.

The **algorithm** uses the following message types:
1) **Election Message:** Sent to announce election.
2) **Answer (Alive) Message:** Responds to the Election message.
3) **Coordinator (Victory) Message:** Sent by winner of the election to announce victory.

When a process P recovers from failure or the failure detector indicates that the current coordinator has failed, P performs the following actions:
1) If P has the highest process id, it sends a Victory message to all other processes and becomes the new Coordinator. Otherwise, P broadcasts an Election message to all other processes with higher process IDs than itself.
2) If P receives no Answer after sending an Election message, then it broadcasts a Victory message to all other processes and becomes the Coordinator.
3) If P receives an Answer from a process with a higher ID, it sends no further messages for this election and waits for a Victory message. (If there is no Victory message after a period of time, it restarts the process at the beginning.)

4) If P receives an Election message from another process with a lower ID it sends an Answer message back and starts the election process at the beginning, by sending an Election message to higher-numbered processes.
5) If P receives a Coordinator message, it treats the sender as the coordinator.

Analysis of Bully Algorithm

1) **Safety:** The safety property expected of leader election protocols is that every non-faulty process either elects a process Q, or elects none at all. Note that all processes that elect a leader must decide on the same process Q as the leader. The Bully algorithm satisfies this property (under the system model specified), and at no point in time is it possible for two processes in the group to have a conflicting view of who the leader is, except during an election. This is true because if it were not, there are two processes X and Y such that both sent the Coordinator (victory) message to the group.

   This means X and Y must also have sent each other victory messages. But this cannot happen, since before sending the victory message, Election messages would have been exchanged between the two, and the process with a lower process id among the two would never send out victory messages. We have a contradiction, and hence our initial assumption that there are two leaders in the system at any given time is false, and that shows that the bully algorithm is safe.

2) **Liveness:** Liveness is also guaranteed in the synchronous, crash-recovery model. Consider the would-be leader failing after sending an Answer (Alive) message but before sending a Coordinator (victory) message. If it does not recover before the set timeout on lower id processes, one of them will become leader eventually (even if some of the other processes crash). If the failed process recovers in time, it simply sends a Coordinator (victory) message to all of the group.

3) **Network Bandwidth Utilization:** Assuming that the bully algorithm messages are of a fixed (known, invariant) size, the most number of messages are exchanged in the group when the process with the lowest id initiates an election. This process sends (N – 1) Election messages the next higher id sends (N – 2) messages, and so on, resulting in $\Theta(N^2)$ election messages. There are also the $\Theta(N^2)$ Alive messages, and $\Theta(N)$ co-ordinator messages, thus making the overall number messages exchanged in the worst case be $\Theta(N^2)$.

Ques 10)Explain the ring-based election algorithm.

Ans: Ring-Based Election Algorithm

The goal of this algorithm is to elect a single process called the coordinator, which is the process with the largest identifier. The ring algorithm uses the same ring arrangement as in the token ring mutual exclusion algorithm, but does not employ a token. Processes are physically or logically ordered so that each knows its successor:

1) If any process detects failure, it constructs an election message with its process I.D. (**e.g.,** network address and local process I.D.) and sends it to its successor.
2) If the successor is down, it skips over it and sends the message to the next party. This process is repeated until a running process is located.
3) At each step, the process adds its own process I.D. to the list in the message.

Eventually, the message comes back to the process that started it:

1) The process sees its ID in the list.
2) It changes the message type to **coordinator.**
3) The list is circulated again, with each process selecting the highest numbered ID in the list to act as coordinator.
4) When the **coordinator** message has circulated fully, it is deleted.

Multiple messages may circulate if multiple processes detected failure. This creates a bit of overhead but produces the same results.