Jisy Raju
Assistant Professor, CE Cherthala

Module 6

Distributed mutual exclusion – central server algorithm – ring based algorithm- Maekawa's voting algorithm –Election: Ring -based election algorithm – Bully algorithm

6.1 DISTRIBUTED MUTUAL EXCLUSION

Distributed processes often need to coordinate their activities. A **Critical Section** is a code segment that accesses shared variables and has to be executed as an atomic action. It means that in a group of cooperating processes, at a given point of time, only one process must be executing its **critical section**. If any other process also wants to execute its critical section, it must wait until the first one finishes.

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. The application-level protocol for executing a critical section is as follows:

- **enter()**: enter critical section – block if necessary
- **resourceAccesses()** : access shared resources in critical section
- **exit():** leave critical section – other processes may now enter

Our essential requirements for mutual exclusion are as follows:

- **ME1: (safety)** At most one process may execute in the critical section (CS) at a time.
- **ME2: (liveness)** Requests to enter and exit the critical section eventually succeed.

ME2 implies freedom from both deadlock and starvation. A deadlock would involve two or more of the processes becoming stuck indefinitely while attempting to enter or exit the critical section, by virtue of their mutual interdependence. But even without a deadlock, a poor algorithm might lead to starvation: the indefinite postponement of entry for a process that has requested it. The absence of starvation is a fairness condition.

- **ME3: (ordering)** If one request to enter the CS happened-before another, then entry to the CS is granted in that order.

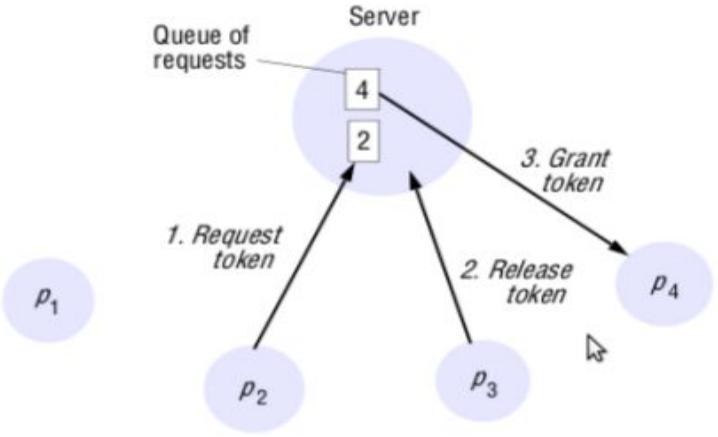
We evaluate the performance of algorithms for mutual exclusion according to the following criteria:

• the **bandwidth** consumed,
• the **client delay** incurred by a process at each entry and exit operation;
• the algorithm's effect upon the **throughput** of the system

Algorithms for mutual exclusion

- A central server algorithm
- A ring-based algorithm
- An algorithm using multicast and logic clocks (Ricart and Agarwal)
- Maekawa's voting algorithm

A central server algorithm



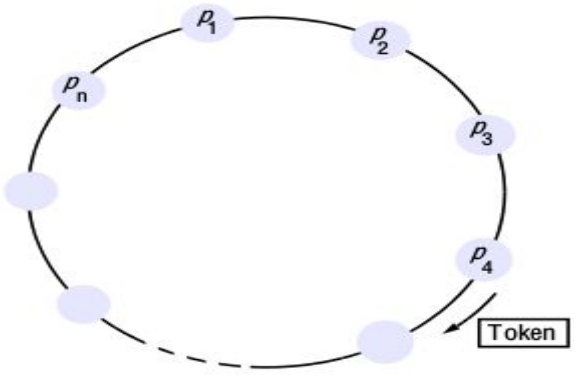
The simplest way to achieve mutual exclusion is to employ a server that grants permission to enter the critical section. To enter a critical section, a process sends a request message to he server and awaits a reply from it. Conceptually, the reply constitutes a token signifying permission to enter the critical section. If no other process has the token at the time of the request, then the server replies immediately, granting the token. If the token is currently held by another process, then the server does not reply, but queues the request. When a process exits the critical section, it sends a message to the server, giving it back the token.

Four process p1, p2, p3, p4

- p1- no need to enter critical section (CS)
- p4 & p2 send request the token to enter CS
- But p3 is accessing CS. So p4 & p2 enter the queue.
- P3 released its token
- p4 get the permission to enter CS
- After p4 released , permission grant to p2

A ring-based algorithm

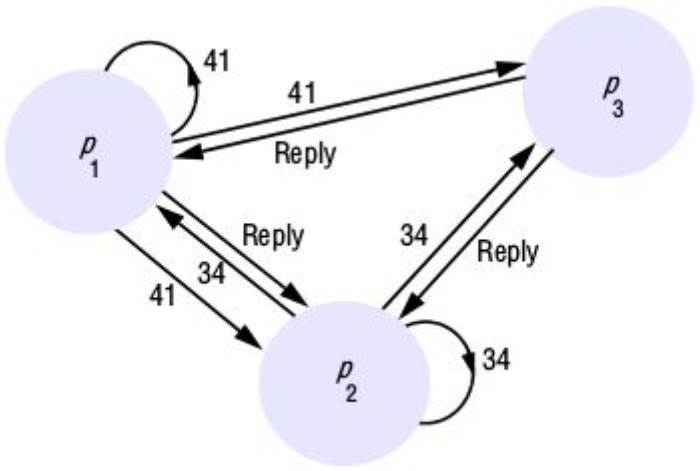
One of the simplest ways to arrange mutual exclusion between the N processes without requiring an additional process is to arrange them in a logical ring. This requires only that each process p i has a communication channel to the next process in the ring, p (i + 1)mod N . The idea is that exclusion is conferred by obtaining a token in the form of a message passed from process to process in a single direction- clockwise. If a process does not require to enter the critical section when it receives the token,
then it immediately forwards the token to its neighbour. A process that requires the token waits until it receives it, but retains it. To exit the critical section, the process sends the token on to its neighbour.



An algorithm using multicast and logic clocks (Ricart and Agarwal)

Ricart and Agrawala [1981] developed an algorithm to implement mutual exclusion

between N peer processes that is based upon multicast. The basic idea is that processes that require entry to a critical section multicast a request message, and can enter it only when all the other processes have replied to this message. The conditions under which a process replies to a request are designed to ensure that conditions ME1–ME3 are met.



To illustrate the algorithm, consider a situation involving three processes, p 1 , p 2 and p 3.

- p 3 is not interested in entering the critical section
- p 1 and p 2 request entry concurrently. Timestamp of p 1 's request is 41, and that of p 2 is 34
- p 3 receives their requests, it replies immediately
- p 2 receives p 1 's request, it finds that its own request has the lower timestamp and so does not reply
- p 1 finds that p 2 's request has a lower timestamp than that of its own request and so replies immediately.
- On receiving this second reply, p 2 can enter the critical section. When p 2 exits the critical section, it will reply to p 1 's request and so grant it entry.

Ricart and Agrawala's algorithm

```
On initialization
    state := RELEASED;

To enter the section
    state := WANTED;
    Multicast request to all processes;          } Request processing deferred here
    T := request's timestamp;
    Wait until (number of replies received = (N – 1));
    state := HELD;

On receipt of a request <Ti, pi> at pj (i ≠ j)
    if (state = HELD or (state = WANTED and (T, pj) < (Ti, pi)))
    then
                queue request from pi without replying;
    else
                reply immediately to pi;
    end if

To exit the critical section
    state := RELEASED;
    reply to any queued requests;
```

Maekawa's voting algorithm

Maekawa's Algorithm is quorum based approach to ensure mutual exclusion in distributed systems. As we know, In permission based algorithms like Lamport's Algorithm, Ricart-Agrawala Algorithm etc. a site request permission from every other site but in quorum based approach, A site does not request permission from every other site but from a subset of sites which is called **quorum**.

A request set or Quorum in Maekawa's algorithm must satisfy the following properties:

- $R_i \cap R_j \neq \varnothing$ **i.e** there is at least one common site between the request sets of any two sites.
- $S_i \in R_i$
- $|R_i| = K$ S_i is contained in exactly K sets
- $N = K(K - 1) + 1$ and $|R_i| = \sqrt{N}$ Maekawa's Algorithm requires invocation of $3\sqrt{N}$ messages per critical section execution as the size of a request set is $\sqrt{N}$. These $3\sqrt{N}$ messages involves. $\sqrt{N}$ request messages, $\sqrt{N}$ reply messages, $\sqrt{N}$ release messages

Algorithm:

- **To enter Critical section:**
  - When a site S_i wants to enter the critical section, it sends a request message **REQUEST(i)** to all other sites in the request set $\mathbf{R_i}$.
  - When a site S_j receives the request message **REQUEST(i)** from site S_i, it returns a **REPLY** message to site S_i if it has not sent a **REPLY** message to the site from the time it received the last **RELEASE** message. Otherwise, it queues up the request.
- **To execute the critical section:**
  - A site S_i can enter the critical section if it has received the **REPLY** message from all the site in request set $\mathbf{R_i}$
- **To release the critical section:**
  - When a site S_i exits the critical section, it sends **RELEASE(i)** message to all other sites in request set $\mathbf{R_i}$
  - When a site S_j receives the **RELEASE(i)** message from site S_i, it send **REPLY** message to the next site waiting in the queue and deletes that entry from the queue
  - In case queue is empty, site S_j update its status to show that it has not sent any **REPLY** message since the receipt of the last **RELEASE** message

Drawbacks of Maekawa's Algorithm:

- This algorithm is deadlock prone because a site is exclusively locked by other sites and requests are not prioritized by their timestamp.

```
On initialization
  state := RELEASED;
  voted := FALSE;
For p_i to enter the critical section
  state := WANTED;
  Multicast request to all processes in V_i;
  Wait until (number of replies received = K);
  state := HELD;
On receipt of a request from p_i at p_j
  if (state = HELD or voted = TRUE)
  then
    queue request from p_i without replying;
  else
    send reply to p_i;
    voted := TRUE;
  end if
```

```
For p_i to exit the critical section
  state := RELEASED;
  Multicast release to all processes in V_i
On receipt of a release from p_i at p_j
  if (queue of requests is non-empty)
  then
    remove head of queue – from p_k, say
    send reply to p_k;
    voted := TRUE;
  else
    voted := FALSE;
  end if
```

ELECTIONS

An algorithm for choosing a unique process to play a particular role (coordinator) is called an election algorithm. An election algorithm is needed for this choice. It is essential that all the processes agree on the choice. Afterwards, if the process that plays the role of server wishes to retire then another election is required to choose a replacement. We say that a process calls the election if it takes an action that initiates a particular run of the election algorithm. At any point in time, a process p i is either a participant – meaning that it is engaged in some run of the election algorithm – or a non-participant – meaning that it is not currently engaged in any election.

Each process p i (i = 1 # 2 # } # N) has a variable elected i , which will contain the identifier of the elected process. When the process first becomes a participant in an election it sets this variable to the special value ‘ A ’ to denote that it is not yet defined.

Our requirements are that, during any particular run of the algorithm:

E1: (safety) A participant process p i has elected i = A or elected i = P, where P is chosen as the non-crashed process at the end of the run with the largest identifier.

E2: (liveness) All processes p i participate and eventually either set elected i z A – or crash.

Two algorithms,

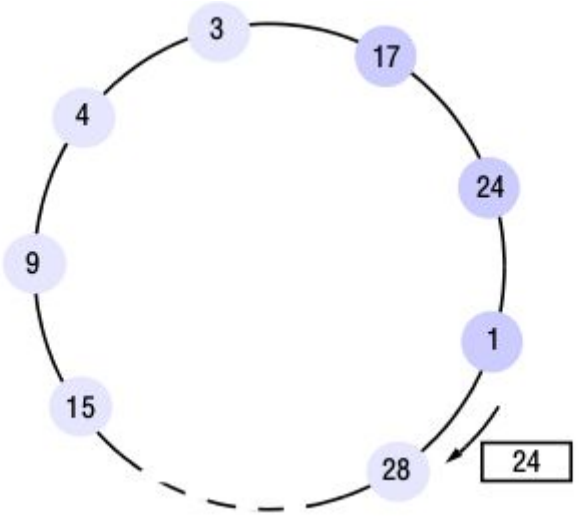
- **A ring-based election algorithm**
- **Bully algorithm**

A ring-based election algorithm

The algorithm of Chang and Roberts is suitable for a collection of processes arranged in a logical ring. Each process p i has a communication channel to the next process in the ring, p (i + 1) mod N , and all messages are sent clockwise around the ring. The goal of this algorithm is to elect a single process called the coordinator, which is the process with

the largest identifier.

Initially, every process is marked as a non-participant in an election. Any process can begin an election. It proceeds by marking itself as a participant, placing its identifier in an election message and sending it to its clockwise neighbour. When a process receives an election message, it compares the identifier in the message with its own. If the arrived identifier is greater, then it forwards the message to its neighbour. If the arrived identifier is smaller and the receiver is not a participant, then it substitutes its own identifier in the message and forwards it; but it does not forward the message if it is already a participant. On forwarding an election message in any case, the process marks itself as a participant. If, however, the received identifier is that of the receiver itself, then this process's identifier must be the greatest, and it becomes the coordinator. The coordinator marks itself as a non-participant once more and sends an elected message to its neighbour, announcing its election and enclosing its identity.



- The election was started by process 17. Process forward to neighbour with greatest identifier.
- The election message currently contains 24, and forwards
- The process 28 will replace 24 with its identifier when the message reaches it.
- The election message currently contains 28, and forwards until the received identifier is that of the receiver itself, then this process's identifier must be the greatest, and it becomes the coordinator and sends a coordinator messages to its neighbour.

The bully algorithm

There are three types of message in this algorithm:

- election message is sent to announce an election
- an answer message is sent in response to an election message
- a coordinator message is sent to announce the identity of the elected process.

1. The process that knows it has the highest identifier can elect itself as the coordinator simply by sending a coordinator message to all processes with lower identifiers.
2. On the other hand, a process can begin an election by sending an election message to those processes and awaiting answer messages in response. If none arrives within time T, the process considers itself as the coordinator and sends a coordinator message to all processes with lower identifiers announcing this. Otherwise, the other process start election for a coordinator.

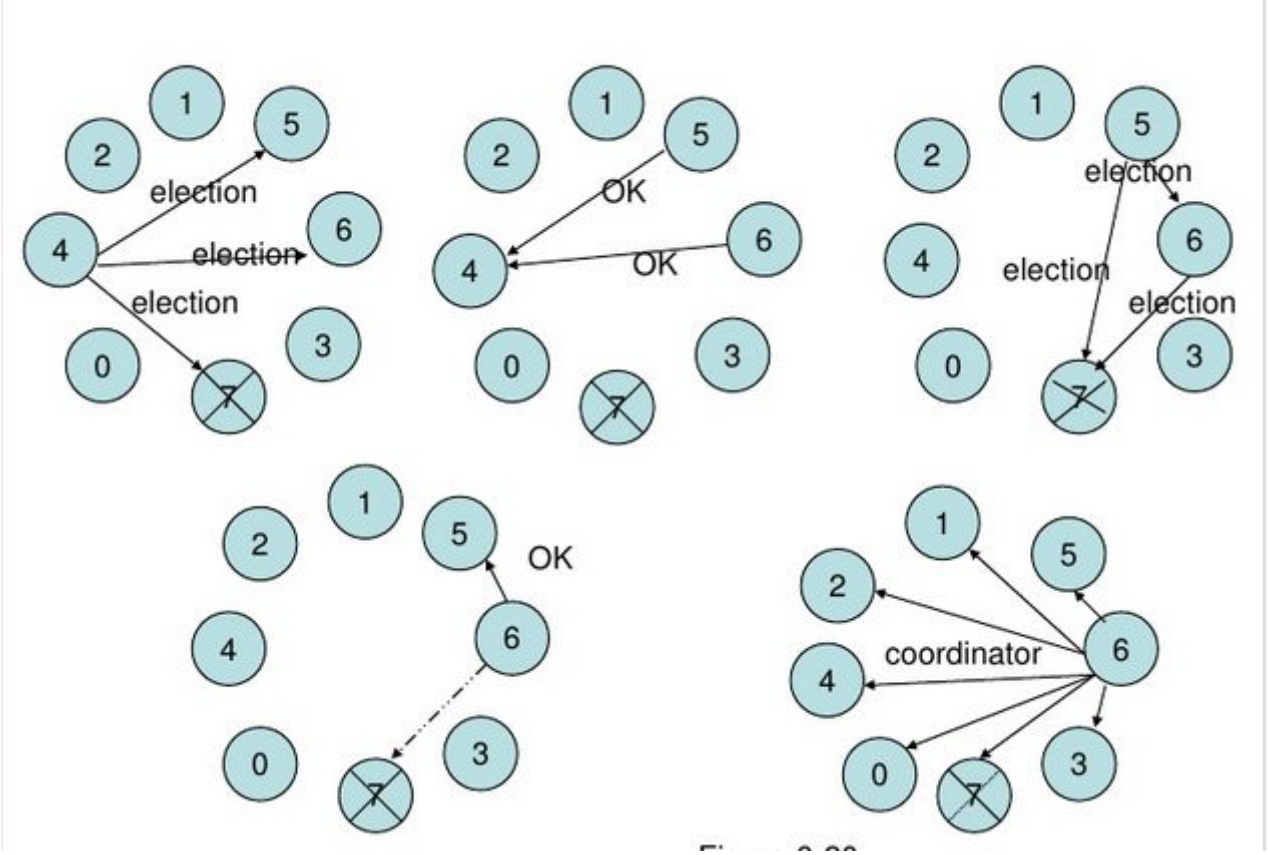
3. When a process, P, notices that the coordinator is no longer responding to requests, it initiates an election. P sends an ELECTION message to all processes

   If no one responds, P wins the election and becomes a coordinator. The new coordinator announces its victory by sending all processes a message telling them that starting immediately it is the new coordinator.

4. If a process that was previously down comes back:

- It take over the coordinator job. Biggest guy" always wins and hence the name " **bully**" algorithm.

Example



- P0 to P7 process are there. The coordinator P7 has just crashed.Process 4 notices if first and sends ELECTION messages to all the processes
- P5 and 6 both respond with answer message to P4 . Upon getting these responses, P4 job is over.
- Then P5 sends ELECTION messages to all the processes andP6 responds with answer message to P5.
- P6 send election message and there is no response from other P, then P6 wins the election and becomes a coordinator.
- If P7 is ever restarted, it will just send all the others a COORDINATOR message and bully P6.