Advanced Operating Systems

20MCAT172

Module II

Distributed Mutual Exclusion

- The problem of mutual exclusion arises in distributed systems whenever concurrent access to shared resources by several sites/process is involved.
- It is necessary that the shared resource be accessed by a single site at a time (atomic) for correctness.
- Example: Directory Management
    - Update to a directory must be done atomically.

Distributed Mutual Exclusion

- It is necessary that the shared resource be accessed by a single site (or process) at a time.

- In single-computer systems, the status of a shared resource and the status of users is readily available in the shared memory, and solutions to the mutual exclusion problem can be easily implemented using shared variables(semaphores).

- In distributes systems, the shared resources and the users may be distributed and shared memory does not exist. So approaches based on shared variables are not applicable to distributed systems and approaches based on message passing must be used.

- This problem is more complex because of lack of both shared memory and a common physical clock and because of unpredictable message delays.

Classification of mutual exclusion algorithms

- Algorithms differ in their communication topology and in the amount of information maintained by each site about other sites.
- Algorithms grouped into two classes.

1. Nontoken-based: these algorithms require two or more successive rounds of message exchanges among sites. These are assertion based because a site can enter its critical section (CS) when an assertion defined on its local variables becomes true. Mutual exclusion is enforced because the assertion becomes true only at one site at any given time.
2. Token-based :a unique token (also known as PRIVILEGE based) is shared among the sites. A site is allowed to enter its CS if it possesses the token and it continues to hold the token until the execution of the CS is over.

System model

- At any instant, a site may have several requests for CS.

- A site queues up these requests and serves them one at a time.

- A site can be in one of the following three states:
    1. Requesting CS – site is blocked and cannot make further requests of CS
    2. Executing CS
    3. Neither requesting nor executing CS(i.e. idle) – the site is executing outside CS

- In the token-based algorithms, a site can also be in a state where a site holding the token is executing outside the CS. Such a state is referred to as an idle token stage.

Requirements for mutual exclusion

- The primary objective of mutual exclusion algorithm is to guarantee that only one request accesses the CS at a time. In addition the following characteristics are important.

1. Freedom from Deadlocks – two or more site should not endlessly wait for messages that will never arrive.
2. Freedom from starvation – a site should not be forced to wait indefinitely to execute CS while other sites are repeatedly executing CS. That is every requesting site should get an opportunity to execute CS in a finite time.
3. Fairness – requests must be executed in the order they are made (or the order in which they arrive in the system)
4. Fault tolerance – a mutual exclusion algorithm is fault-tolerant if in the wake of failure, it can reorganize itself so that it continues to function without any disruptions.

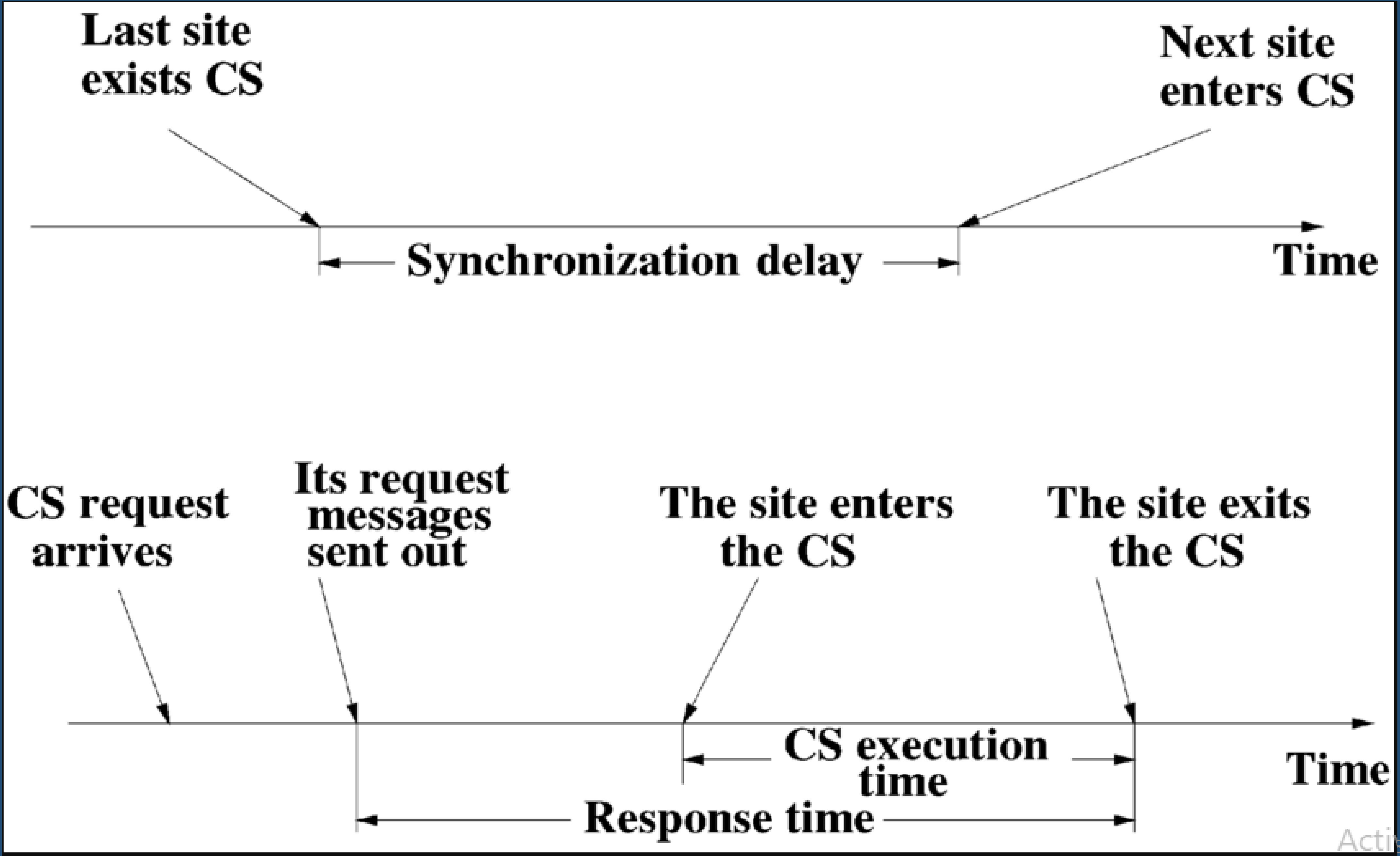
Measuring Performance

- The performance of the mutual exclusion algorithms is measured by four metrics:

1. Number of messages necessary per CS
2. Synchronization delay, which is the time required after a site leaves the CS and before the next site enters the CS.
3. Response time – which is the time interval a request waits for its CS execution to be over after its request have been sent out.
4. System throughput – which is the rate at which the system executes requests for the CS.

   If *sd* is the synchronization delay and *E* is the average critical section execution time, then the throughput is:

$$\text{SystemThroughput} = 1/(sd + E)$$


Last site
exists CS
Next site
enters CS
Synchronization delay
Time
CS request
arrives
Its request
messages
sent out
The site enters
the CS
The site exits
the CS
CS execution
time
Time
Response time

Low and high load performance

- Performance of mutual exclusion algorithms depends on loading conditions of the system., low load and high load.

- **Low load** – there is rarely more that one request for mutual exclusion simultaneously.

- **High load** – always a pending request for mutual exclusion at a time, site is in an idle state.
  - After having executed a request, a site immediately initiates next site to execute its CS.
  - A site is rarely in an idle state.

Best and Worst case performance

- Best case –prevailing conditions are such that a performance metric attains the best possible value, best value for response time is achieved when the load is low.

- Worst case – worst value for response time is achieved when the load is high

Non-token based algorithms

- A site communicates with a set of other sites to arbitrate who should execute the CS next.
- For a site Si, request set Ri contains ids of all those sites from which site Si must acquire permission before entering the CS.
- These algorithms use timestamps to order requests for the CS and to resolve conflicts between simultaneous requests for the CS.
- In these algorithms, logical clocks are maintained and updated according to Lamport's scheme.
- Each request for the CS gets a timestamp, and smaller the timestamp requests have priority over larger timestamp requests.

Examples

- Lamport's Algorithm
- Rickart-Agrawala Algorithm

Lamport’s Algorithm for Mutual Exclusion in Distributed System

- **Lamport’s Distributed Mutual Exclusion Algorithm** is a permission based algorithm proposed by Lamport as an illustration of his synchronization scheme for distributed systems.
- In Lamport’s Algorithm critical section requests are executed in the increasing order of timestamps i.e a request with smaller timestamp will be given permission to execute critical section first than a request with larger timestamp.
- A timestamp is given to each critical section request using Lamport’s logical clock.
- Timestamp is used to determine priority of critical section requests.
- Smaller timestamp gets high priority over larger timestamp.
- The execution of critical section request is always in the order of their timestamp.

Lamport’s Algorithm for Mutual Exclusion in Distributed System

- In this algorithm:
  - Three type of messages (**REQUEST**, **REPLY** and **RELEASE**) are used and communication channels are assumed to follow FIFO order.
  - A site send a **REQUEST** message to all other site to get their permission to enter critical section.
  - A site send a **REPLY** message to requesting site to give its permission to enter the critical section.
  - A site send a **RELEASE** message to all other site upon exiting the critical section.
  - Every site S_i, keeps a queue to store critical section requests ordered by their timestamps.

    request_queue$_i$ denotes the queue of site S_i

Lamport's Algorithm for Mutual Exclusion in Distributed System

- In this algorithm:
  - Three type of messages (**REQUEST**, **REPLY** and **RELEASE**) are used and communication channels are assumed to follow FIFO order.
  - A site send a **REQUEST** message to all other site to get their permission to enter critical section.
  - A site send a **REPLY** message to requesting site to give its permission to enter the critical section.
  - A site send a **RELEASE** message to all other site upon exiting the critical section.
  - Every site S_i, keeps a queue to store critical section requests ordered by their timestamps.

    request_queue$_i$ denotes the queue of site S_i

Lamport's Algorithm for Mutual Exclusion in Distributed System

Algorithm:

- **Requesting the Critical section:**
  - When a site S_i wants to enter the critical section, it sends a request message **Request(ts_i, i)** to all other sites and places the request on **request_queue$_i$**. Here, Ts_i denotes the timestamp of Site S_i
  - When a site S_j receives the request message **REQUEST(ts_i, i)** from site S_i, it returns a timestamped REPLY message to site S_i and places the request of site S_i on **request_queue$_j$**.
- **Executing the critical section:**
  - A site S_i can enter the critical section if it has received the message with timestamp larger than **(ts_i, i)** from all other sites and its own request is at the top of **request_queue$_i$**

Lamport’s Algorithm for Mutual Exclusion in Distributed System

- **Releasing the critical section:**
  - When a site S_i exits the critical section, it removes its own request from the top of its request queue and sends a timestamped **RELEASE** message to all other sites
  - When a site S_j receives the timestamped **RELEASE** message from site S_i, it removes the request of S_i from its request queue

- When a site removes a request from its request queue, its own request may come at the top of the queue, enabling it to enter the CS.
- The algorithm executes CS requests in the increasing order of timestamps.

Ricart–Agrawala Algorithm for

Mutual Exclusion in Distributed System

- **Ricart–Agrawala algorithm** is an algorithm to for mutual exclusion in a distributed system proposed by Glenn Ricart and Ashok Agrawala.

- This algorithm is an extension and optimization of Lamport’s Distributed Mutual Exclusion Algorithm.

- Like Lamport’s Algorithm, it also follows permission based approach to ensure mutual exclusion

Ricart–Agrawala Algorithm for Mutual Exclusion in Distributed System

Algorithm:

- **Requesting the Critical section:**
  - When a site S_i wants to enter the critical section, it send a timestamped **REQUEST** message to all other sites.
  - When a site S_j receives a **REQUEST** message from site S_i, It sends a **REPLY** message to site S_i if and only if
    - Site S_j is neither requesting nor currently executing the critical section.
    - Incase Site S_j is requesting, the timestamp of Site S_i's request is smaller than its own request. Otherwise the request is deferred by site S_j.

Ricart–Agrawala Algorithm for Mutual Exclusion in Distributed System

- **Executing the critical section:**
  - Site S_i enters the critical section if it has received the **REPLY** message from all other sites.

- **Releasing the critical section:**
  - Upon exiting site S_i sends **REPLY** message to all the deferred requests.

- -The execution of CS is always in the order of their timestamps.

Token-based algorithms

- A unique token is shared among all sites.
- A site is allowed to enter its CS if it possesses the token.
- Token-based algorithms use sequence numbers instead of timestamps.
- Every request for the token contains a sequence number and the sequence numbers of sites advance independently.
- A site increments its sequence number counter every time it makes a request for the token.
- A primary function of the sequence number is to distinguish between old and current requests.

Example

- SUZUKI-KASAMI’S BROADCAST ALGORITHM

Suzuki–Kasami Algorithm for Mutual Exclusion in Distributed System

- **Suzuki–Kasami algorithm** is a token-based algorithm for achieving mutual exclusion in distributed systems.

- This is modification of Ricart–Agrawala algorithm, a permission based (Non-token based) algorithm which uses **REQUEST** and **REPLY** messages to ensure mutual exclusion.

- *In token-based algorithms, A site is allowed to enter its critical section if it possesses the unique token.*

- Non-token based algorithms uses timestamp to order requests for the critical section where as sequence number is used in token based algorithms.

Suzuki–Kasami Algorithm for Mutual Exclusion in Distributed System

- n array of integers **RN[1…N]**   **// Request Array**
  A site S_i keeps **RN_i[1…N]**, where **RN_i[j]** is the largest sequence number received so far through **REQUEST** message from site S_i.
- An array of integer **LN[1…N]**   **// Token Array**
  This array is used by the token. **LN[J]** is the sequence number of the request that is recently executed by site S_j.
- A queue **Q**
  This data structure is used by the token to keep record of ID of sites waiting for the token

Suzuki–Kasami Algorithm for Mutual Exclusion in Distributed System

Algorithm:

- **Requesting the Critical section:**

  - When a site S_i wants to enter the critical section and it does not have the token then it increments its sequence number **$RN_i[i]$** and sends a request message **REQUEST(i, sn)** to all other sites in order to request the token. Here **sn** is update value of **$RN_i[i]$**

  - When a site S_j receives the request message **REQUEST(i, sn)** from site S_i, it sets **$RN_j[i]$** to maximum of **$RN_j[i]$** and **sn** i.e **$RN_j[i]$** = max(**$RN_j[i]$**, **sn**).
  - After updating **$RN_j[i]$**, Site S_j sends the token to site S_i if it has token and **$RN_j[i]$** = **LN[i]** + 1

Suzuki–Kasami Algorithm for Mutual Exclusion in Distributed System

- **Executing the critical section:**
  - Site S_i executes the critical section if it has acquired the token.
- **Releasing the critical section:**
  After finishing the execution Site S_i exits the critical section and does following:
  - sets **LN[i]** = **RN$_i$[i]** to indicate that its critical section request **RN$_i$[i]** has been executed.
  - For every site S_j, whose ID is not present in the token queue **Q**, it appends its ID to **Q** if **RN$_i$[j]** = **LN[j]** + 1 to indicate that site S_j has an outstanding request.
  - After above updation, if the Queue **Q** is non-empty, it pops a site ID from the **Q** and sends the token to site indicated by popped ID.
  - If the queue **Q** is empty, it keeps the token.

RESOURCE SECURITY AND PROTECTION

Security & Protection

- Security and protection deals with the control of unauthorized use and access to hardware and software resources of a computer system.
- Protection is a mechanism and security is a policy.
- Protection deals with mechanisms to build secure systems and security deals with policy issues that use protection mechanisms to build secure systems.
- Policies refer to what should be done and mechanisms refer to how it should be done.
- Protection in an operating system refers to mechanisms that control user access to system resources, whereas policies decide which user can have access to what resources.
- Policies can change with time and applications.
- A protection scheme must be amenable to a wide variety of policies to ensure security in computer systems

Potential Security Violations

- **Classified the potential security violations into three categories.**

1. Unauthorized information release
2. Unauthorized information modification
3. Unauthorized denial of service

Potential Security Violations

1. **Unauthorized information release:**

    - This occurs when an unauthorized person is able to read and take advantage of the information stored in a computer system.
    - This also includes the unauthorized use of a program.

2. **Unauthorized information modification:**

    - This occurs when an unauthorized person is able to alter the information stored in a computer.
    - Eg: Changing student grades in a university database or changing account balance in a bank database.
    - An unauthorized person need not read the information before changing it.
    - Blind writes can be performed.

Potential Security Violations

3. **Unauthorized denial of service**

   - An unauthorized person should not succeed in preventing an authorized user from accessing the information stored in a computer.
   - The services can be denied to authorized users by some internal actions or by external actions.

Design principles for secure systems

Design principles for secure systems

- The principles for designing a secure computer system are:

1. Economy
2. Complete Mediation
3. Open Design
4. Separation of Privileges
5. Least Privilege
6. Least Common Mechanism
7. Acceptability
8. Fail-Safe Defaults

Design principles for secure systems...

1. Economy:
   - A protection mechanism should be economical to develop and use.
   - Its inclusion in a system should not result in substantial cost or overhead to the system.
   - One easy method to achieve economy is to keep the design as simple and small as possible.

2. Complete Mediation:
   - The design of a completely secure system requires that every request to access an object be checked for the authority to do so.
   -

3. Open Design:
   - A protection mechanism should work even if its underlying principles are known to an attacker.

Design principles for secure systems...

4. Separation of Privileges

- A protection mechanism requires two keys to unlock a lock or gain access to a protected object is more robust and flexible than one that allows a single key to unlock a lock.
- In computer systems, the presence of two keys mean satisfying two independent conditions before an access is allowed.

5. Least Privilege:

- A user should be given the minimum access rights that are sufficient for it to complete its task.
- If the requirement of a user changes, the user should acquire it by switching the domain.

Design principles for secure systems...

6. Least Common Mechanism:
   - According to this principle, the portion of a mechanism that is common to more than one user should be minimized.
   - As any coupling among users represents a potential information path between users and is thus a potential threat to their security.
7. Acceptability:
   - A protection mechanism must be simple to use.
   - A complex and obscure protection mechanism will deter users from using it.
8. Fail-Safe Defaults:
   - Default case should mean lack of access

The Access Matrix Model and Implementation

Access Matrix Model

- **Access Matrix** is a security model of protection state in computer system.
- It is represented as a matrix.
- A protection system consists of mechanisms to control user access to system resources or to control information flow in the system.
- *The most fundamental model of protection is the access matrix model in computer systems.*
- The original model is called access matrix since the authorization state, meaning the authorizations holding at a given time in the system, is represented as a matrix.
- The matrix therefore gives an abstract representation of protection systems.
- Access matrix is used **to define the rights** of each process executing in the domain with respect to each object.
- It is used to describe which users have access to what objects.

Access Matrix Model

- The model consists of the following three components:

    1. Current Objects

    2. Current Subjects

    3. Generic Rights

1. Current Objects

- Current objects are a finite set of entities to which access is to be controlled.
- The set is denoted by ‘O’.
- Example : File

Access Matrix Model

2. **Current Subjects**
   - Current subjects are a finite set of entities that access current objects.
   - The set is denotes by ‘S’.
   - Example : Process

3. **Generic Rights**
   - A finite set of generic rights, R = {r1, r2, r3, ... , rm}, gives various access rights that subjects can have to objects.
   - Example : read, write, own, delete etc.

The Protection State of a System

- The protection state of a system is represented by a triplet (S, O, P), where
  - S is the set of current subjects.
  - O is the set of current objects.
  - P is a matrix called Access Matrix with a row for every current subject and a column for every current object.

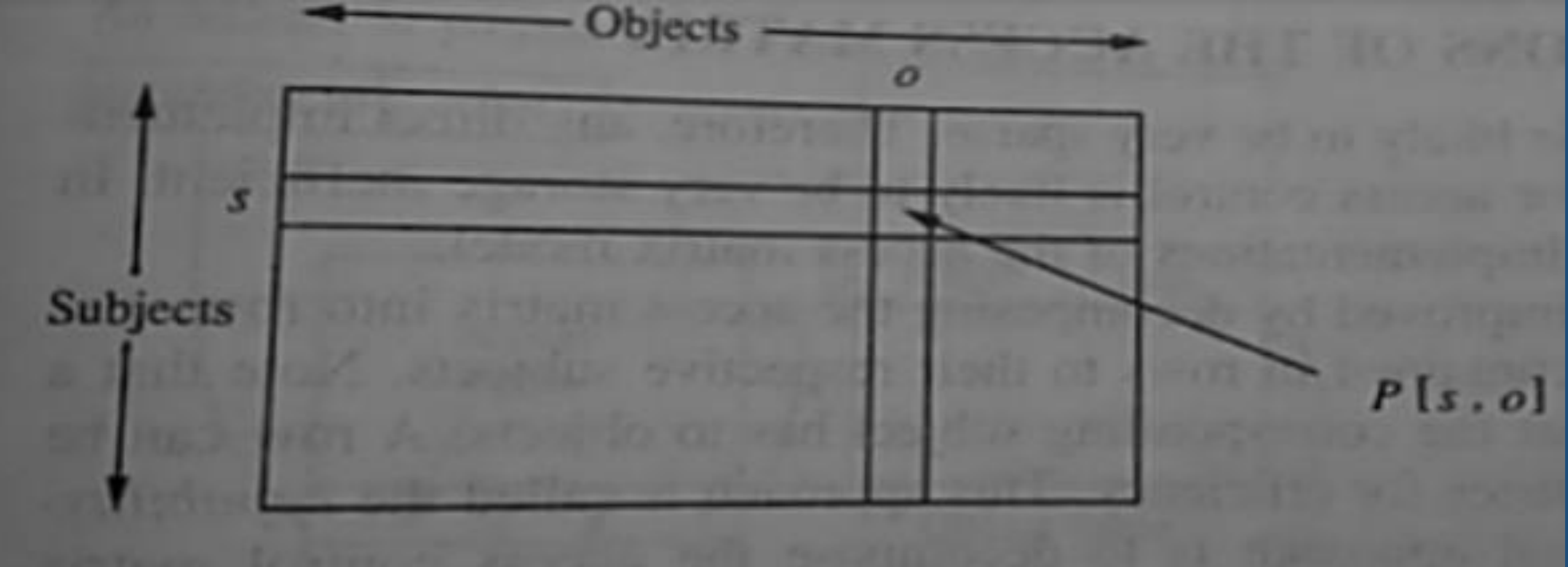
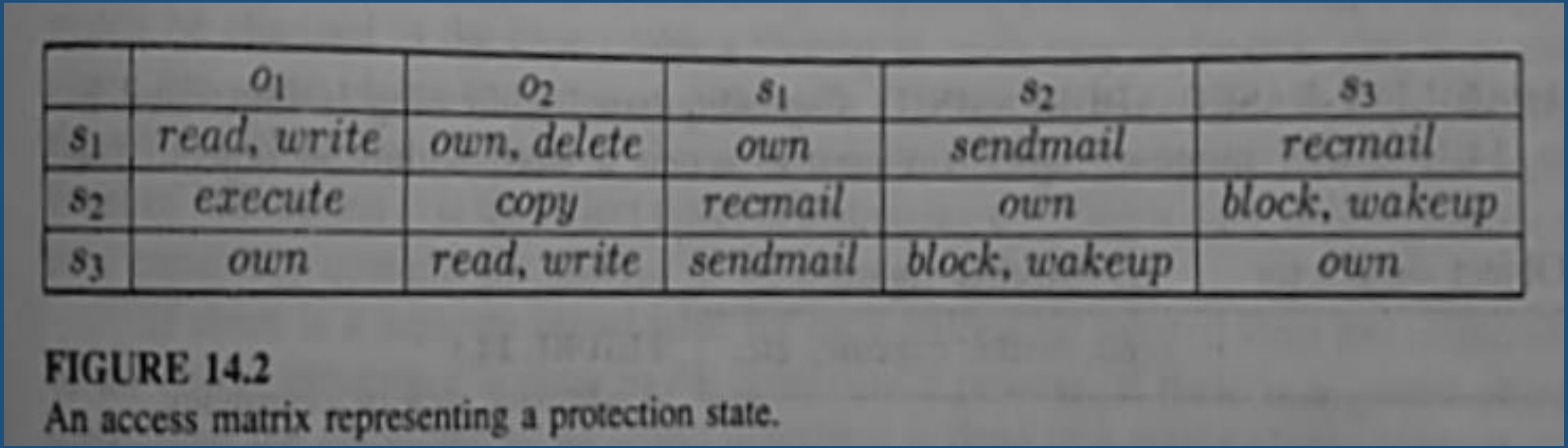


FIGURE 14.1
A schematic of an access matrix.

Matrix itself ia a protected object.
S - Subject
O – Object
P [s,o] – Access right which s has to object o

The Protection State of a System

- The protection state of a system is represented by a triplet (S, O, P), where
  - S is the set of current subjects.
  - O is the set of current objects.
  - P is a matrix called Access Matrix with a row for every current subject and a column for every current object.



	o_1	o_2	s_1	s_2	s_3
s_1	read, write	own, delete	own	sendmail	recmail
s_2	execute	copy	recmail	own	block, wakeup
s_3	own	read, write	sendmail	block, wakeup	own

FIGURE 14.2
An access matrix representing a protection state.

The Protection State of a System

- The Access matrix model of a protection system is very popular because of its simplicity, elegant structure and amenability to various implementations.

Implementation of the Access Matrix

- The Access matrix is very sparse and takes up a large chunk of memory.
- So, any direct implementation of the access matrix for access control is likely to be very storage inefficient.
- Three implementations of the access matrix model:

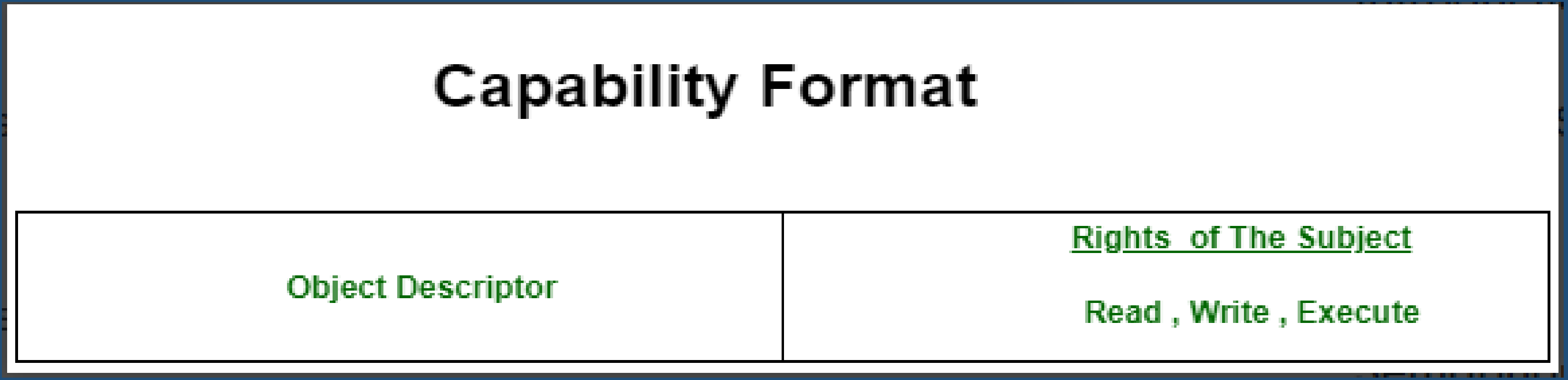
  1. **Capabilities**
  2. **The Access Control List Method**
  3. **The Lock – Key Method**

1. Capabilities

- This method refers to **row wise decomposition** of the access matrix.
- Each Subject is assigned with a list of tuples (o, P[s, o]) for all objects o that it is allowed to access.
- This tuples are called Capabilities.
- If a subject s possess a capability (o, P[s, o]) then it is authorized/allowed to access object o in the manner which is described in P[s, o].
- A subject is allowed to access any objects for which it holds the capabilities.
- Possession of a capability is treated as the evidence to access the object in the ways specified in the capability.
- Capabilities are not meant to be forged.

1. Capabilities

- Capabilities contain two fields:
    - **(i)** Object Descriptor
    - **(ii)** Access Rights
- Object Descriptor contain the address/identifier for the objects
- Access Rights contain the rights which the subject has on object, mainly read write, execute.
- Since object Descriptor contains address it may be used as an addressing mechanism also.
- Below is the format of capability.



Capability Format

Object Descriptor	Rights of The Subject Read , Write , Execute

2. Access Control List

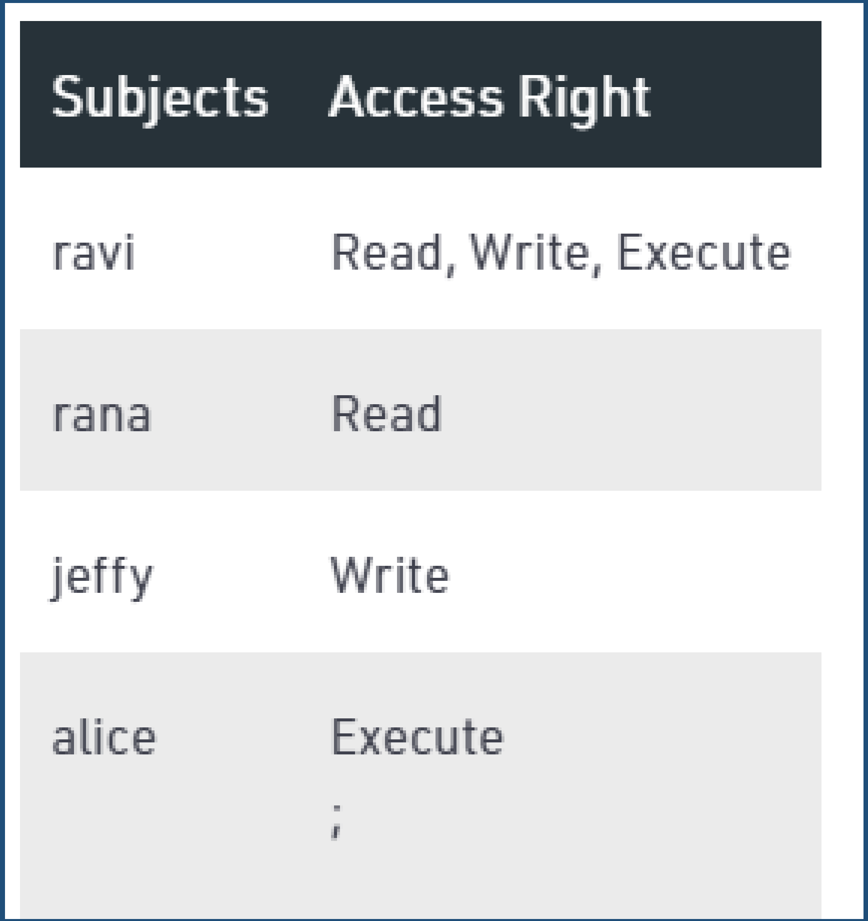
- This method refers to **column wise decomposition** of the access matrix .
- Each object o has a list containing tuples like (s, P[s, o]) for all subjects s which can access the object.
- P[s, o] denotes the rights of the subject s on the object o.
- when a subject s request to access to the object o it is executed in the following manner.
    - The system searches the access control list of o to find out if an entry (s,) exist for subject s
    - If and entry (s, ⫛) exists for subject s then the system checks to see if the requested access is permitted or not.(i.e., α **belong to** ⫛)
    - If the requested access is permitted then the request is executed else an appropriate exception is raised.

Access Control List

- **Features:**
  - Easy Revocation
  - Easy Review of an Access

- **Implementation Considerations:**
  - Efficiency of Execution
  - Efficiency of Storage

A schematic of an access control list



Subjects	Access Right
ravi	Read, Write, Execute
rana	Read
jeffy	Write
alice	Execute ;

3. The Lock – Key Method

- The lock and key method is an **hybrid** of the access control list and capabilities method.
- This method has both the features of both previous methods.
- In the lock and key method, every subject has a capability list that contains tuples of the form (o, key), indicating the subject can access object o using key *k*.
- Every object has an access control list that contains tuples of the form (l, ሐ), called a lock entry indicating that any subject which can open the lock l can access this object in modes contained in the set ሐ.

3. The Lock – Key Method

- When the subject makes the request to access object o in mode α , the system executes in the following manner.
    - The system locates the tuple (o, k) in the capability list of the subject. If no such tuple is found, the access is not permitted.
    - Otherwise the access is permitted only if there exists a lock entry (l, ⫛) in the access control list of object o such that *k*=*l* and α belongs to ⫛.

Thank You