Agreement Protocols Distributed Resource Management

INTRODUCTION

- A kind of co-operation or unity or accordance among processes.
- As we know that all the nodes in a distributed system are working together in a cooperation to achieve a common goals. Hence cooperation among the process is very necessary .
- Agreement protocol is used to ensure that DS is able to achieve the common goal even after occurrence of various failures in Distributed system.
- There are some standard agreement problem in DC. We will see each problem and try to find some protocol or algorithm to solve the agreement problem.

- **Why Agreement Protocol?**

- Agreement is always required to achieve a common goal in distributed system.

- When there are some types of faulty processes present in the distributed system at that time we need to make sure that performance of distributed system should not be affected at that time we implement some agreement protocol (algorithms) ,so that output of distributed should not be incorrect or should not be affected.

- To achieve reliability of Distributed system. So, mainly agreement protocols are used for fault (failure) tolerance in DS

SYSTEM MODEL

Agreement Problems have been studied under following System Model:

1. 'n' processors and at most 'm' of the processors can be faulty.
2. Processors can directly communicate with other processors by message passing.
3. Receiver knows the identity of the sender.
4. Communication medium is reliable.

CLASSIFICATION OF AGREEMENT PROBLEMS

There are three well known agreement problems in distributed systems:

-The Byzantian agreement problem

-The consensus problem

-Interactive consistency problem

In all three problems, all non faulty processors must reach a common agreement.

In the **Byzantian** and the **consensus problems**, the agreement is about a **single value.**

whereas in the **interactive consistency** problem, the agreement is about a **set of common values.**

Problem	Who Initiates Value	Final Agreement
Byzantine Agreement	One Processor	Single Value
Consensus	All Processors	Single Value
Interactive Consistency	All Processors	A Vector of Values

1. The Byzantian agreement problem

An arbitrarily chosen processor, called the *source processor*, broadcasts its initial value to all other processors.

Agreement: All non-faulty processors should agree on the same value.

Validity: If the source processor is non-faulty, then the common agreed upon value by all non-faulty processors must be same as the initial value of the source.

Solution for Byzantian agreement problem

Lamport et. al proposed an algorithm for byzantine agreement problem which is known as Lamport-Shostak-Pease Algorithm.

- Source Broadcasts its initial value to all other processors.
- Processors send their values to other processors and also received values from others.
- During Execution faulty processors may confuse by sending conflicting values.
- However if faulty processors dominate in number, they can prevent non-faulty processors from reaching an agreement.
- So, the no of faulty processors should not exceed certain limit.

2. The consensus problem

Here all process have some initial value they broadcast their initial values to all others process and satisfy the following condition:

Agreement: All non-faulty processes must agree on same single values.

Validity: if all non faulty processes have the same initial value , then the agreed value by all the non-faulty processes must be that same value.

3. Interactive consistency problem

Every processor broadcasts its initial value to all other processors. The initial values of the processors may be different . A protocol for the interactive consistency problem should meet the following conditions:

Agreement: All non-faulty processes must agree on the same array of values A[v1 : : : vn].

Validity: If processor Pi is non-faulty and its initial value is vi , then all non-faulty processes agree on vi as the ith element of the array A. If process j is faulty, then the non-faulty processes can agree on any value for A[j].

APPLICATION OF AGREEMENT PROTOCOL

1. Clock Synchronization in Distributed Systems:

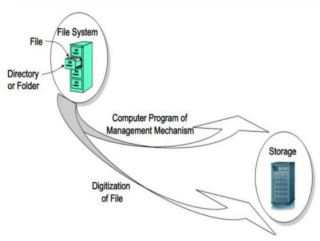
- Distributed Systems require physical clocks to synchronized but physical clocks have drift problem. So, they must periodically resynchronized.
- Such periodically synchronization becomes extremely difficult if the Byzantine failures are allowed.
- This is due to the fact that faulty processors can report different clock value to different processors.
- Agreement Protocols may help to reach a common clock value.

Distributed File Systems

- File System work as the resource management component, which manages the availability of files in distributed system.
- A common file system that can be shared by all the autonomous computers in the system. i.e. **files can be stored at any machine and the computation can be performed at any machine.**

Two important goals :

1. **Network transparency** – to access files distributed over a network. Ideally, users do not have to be aware of the location of files to access them.
2. **High Availability** - to provide high availability. Users should have the same easy access to files, irrespective of their physical location.



ARCHITECTURE OF DISTRIBUTED FILE SYSTEM

- File servers and File clients interconnected by a communication network.

- **Two most important components:**
  1. **Name Server**: map logical names to stored object's (files, directories) physical location.
  2. **Cache Manager**: perform file caching. Can present on both servers and clients.
     1. Cache on the client deals with network latency
     2. Cache on the server deals with disk latency

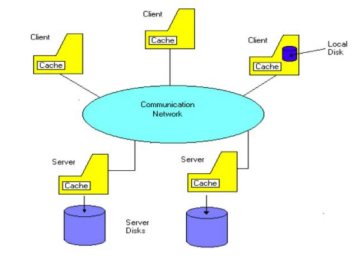


Figure:
Architecture of a Distributed File System

STEPS TO ACCESS DATA

1. check client cache, if present, return data.
2. Check local disk, if present, load into local cache, return data.
3. Send request to file server
4. ...
5. server checks cache, if present, load into client cache, return data
6. disk read
7. load into server cache
8. load into client cache
9. return data

ISSUES IN DISTRIBUTED FILE SYSTEM

1. Naming and Transparency
2. Remote file access and Caching
3. Replication and Concurrent file updates
4. Availability
5. Scalability
6. Semantics

1. Naming and Transparency

- **Issues**
  - How are files named?
  - Do file names reveal their location?
  - Do file names change if the file moves?
  - Do file names change if the *user* moves?
- **Location transparency:** the name of the file does not reveal the physical storage location.
- **Location independence:** The name of the file need not change if the file's storage location changes.
- Most naming schemes used in practice do not have location independence, but many have location transparency.

2. Remote file access and Caching

Once the user specifies a remote file, the OS can do the access either

1. remotely, on the server machine and then return the results using RPC (called *remote service*), or
2. can transfer the file (or part of the file) to the requesting host, and perform local accesses (called *caching*)

Caching Issues:

- Where and when are file blocks cached?
- When are modifications propagated back to the remote file?
- What happens if multiple clients cache the same file?

3. Concurrent file updates

- Server data is replicated across multiple machines.
- Need to ensure consistency of files when a file is updated by multiple clients.
- Changes to a file by one client should not interfere with the operations of other clients.

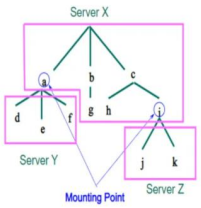
4. Availability

- how to keep replicas consistent.
- how to detect inconsistencies among replicas.
- consistency problem may decrease the availability.
- **Replica Management**: voting mechanism to read and write to replica.

MECHANISM FOR BUILDING DISTRIBUTED FILE SYSTEM

1. Mounting:

A mount mechanism allows the binding together of different filename spaces to form a single hierarchically structured name space.



Name Space Hierarchy

2. Caching

- Caching is commonly employed in distributed files systems to reduce delays in the accessing of data.
- In file caching, a copy of data stored at a remote file server is brought to the client when referenced by the client.
- Subsequent access to the data is performed locally at the client, thereby reducing access delays due to network latency.
- Caching exploits the temporal locality of reference exhibited by programs.
- The temporal locality of reference refers to the fact that a file recently accessed is likely to be accessed again in the near future.

3. Hint

- **An alternative approach is used when cached data are not expected to be completely accurate.**
- However, valid cache entries improve performance substantially without incurring the cost of maintaining cost consistency.
- The class of applications that can utilize hints are those which can recover after discovering that the cached data are invalid.
- For example, after the name of a file or directory is mapped to the physical object, the address of the object can be stored as a hint in the cache.
- If the address fails to map to the object in the following attempt, the cached address is purged from the cache.
- The file server consults the name server to determine the actual location of the file or directory and updates the cache.

4. Bulk Data Transfer

- Transferring data in bulk reduces the protocol processing overhead at both servers and clients.
- In bulk data transfer, multiple consecutive data blocks are transferred from servers to clients instead of just the block referenced by clients.
- Bulk transfers reduce file access overhead through obtaining a multiple number of blocks with a single seek; by formatting and transmitting a multiple number of large packets in a single context switch; and by reducing the number of acknowledgements that need to be sent.

5. Encryption

- Encryption is used for enforcing security in distributed systems.
- In this scheme, two entities wishing to communicate with each other establish a key for conversation with the help of an authentication server.
- It is important to note that the conversation key is determined by the authentication server, but is never spent in plain (unencrypted) text to either of the entities.

Distributed Operating Systems

Distributed systems require a network to connect all of its elements (devices, hardware, or software) so that they may exchange messages and interact.

A distributed system's ability to consistently convey messages, whether they're delivered, received, acknowledged, or how a node retries after a failure, is a key aspect.

Services and applications required to scale, and additional computers necessary to be added and managed, therefore distributed systems were born.

It connects multiple computers via a single communication channel. Furthermore, each of these systems has its own processor and memory.

Types of Distributed Operating System

1. **Client-Server Systems**
2. **Peer-to-Peer Systems**
3. **Middleware**
4. **Three-tier**
5. **N-tier**

Client-Server System

This type of system requires the client to request a resource, after which the server gives the requested resource. When a client connects to a server, the server may serve multiple clients at the same time.

Client-Server Systems are also referred to as "Tightly Coupled Operating Systems".

Client-Server Systems function as a centralized server since they approve all requests issued by client systems.

Server systems can be divided into two parts:

1. **Computer Server System**

This system allows the interface, and the client then sends its own requests to be executed as an action. After completing the activity, it sends a back response and transfers the result to the client.

2. **File Server System**

It provides a file system interface for clients, allowing them to execute actions like file creation, updating, deletion, and more.

Peer-to-Peer System

The nodes play an important role in this system. The task is evenly distributed among the nodes. Additionally, these nodes can share data and resources as needed. Once again, they require a network to connect.

The Peer-to-Peer System is known as a "Loosely Couple System".

This concept is used in computer network applications since they contain a large number of processors that do not share memory or clocks.

Middleware

Middleware enables the interoperability of all applications running on different operating systems. Those programs are capable of transferring all data to one other by using these services.

Three-tier

The information about the client is saved in the intermediate tier rather than in the client, which simplifies development. This type of architecture is most commonly used in online applications.

N-tier

When a server or application has to transmit requests to other enterprise services on the network, n-tier systems are used.

Features of Distributed Operating System

Openness

It means that the system's services are freely displayed through interfaces.

Scalability

It refers to the fact that the system's efficiency should not vary as new nodes are added to the system.

the performance of a system with 100 nodes should be the same as that of a system with 1000 nodes.

Resource Sharing

Its most essential feature is that it allows users to share resources. They can also share resources in a secure and controlled manner. Printers, files, data, storage, web pages, etc., are examples of shared resources.

Flexibility

A DOS's flexibility is enhanced by modular qualities and delivers a more advanced range of high-level services.

Transparency

It is the most important feature of the distributed operating system. The primary purpose of a distributed operating system is to hide the fact that resources are shared. Transparency also implies that the user should be unaware that the resources he is accessing are shared. Furthermore, the system should be a separate independent unit for the user.

Fault Tolerance

Fault tolerance is that process in which user may continue their work if the software or hardware fails.

Applications of Distributed Operating System

Network Applications

DOS is used by many network applications, including the Web, peer-to-peer networks, multiplayer web-based games, and virtual communities.

Telecommunication Networks

DOS is useful in phones and cellular networks. A DOS can be found in networks like the Internet, wireless sensor networks, and routing algorithms.

Distributed Deadlock Detection

In a distributed system deadlock can neither be prevented nor avoided as the system is so vast that it is impossible to do so.

Therefore, only deadlock detection can be implemented. The techniques of deadlock detection in the distributed system require the following:

- **Progress –**
  The method should be able to detect all the deadlocks in the system.
- **Safety –**
  The method should not detect false or phantom deadlocks.

There are three approaches to detect deadlocks in distributed systems.

Centralized approach –

In the centralized approach, there is only one responsible resource to detect deadlock.

The advantage of this approach is that it is simple and easy to implement, while the drawbacks include excessive workload at one node.

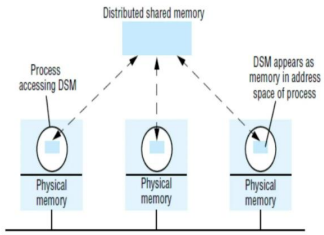
Distributed approach-
In the distributed approach different nodes work together to detect deadlocks. No single point failure (that is the whole system is dependent on one node if that node fails the whole system crashes) as the workload is equally divided among all nodes. The speed of deadlock detection also increases.

Hierarchical approach –

It is the combination of both centralized and distributed approaches of deadlock detection in a distributed system. In this approach, some selected nodes or clusters of nodes are responsible for deadlock detection and these selected nodes are controlled by a single node.

DISTRIBUTED SHARED MEMORY

- Idea of distributed shared memory is to provide an environment where computers support a shared address space that is made by physically dispersed memories.
- It refers to shared memory paradigm applied to loosely coupled distributed memory systems. **It gives the systems illusion of physically shared memory.**
- Memory mapping manager is responsible for mapping between local memories and the shared memory address space.
- Any processor can access any memory location in the address space directly.
- Chief responsibility is to keep the address space coherent at the times.



- Each node of the system consist of one or more CPUs and memory unit.
- Nodes are connected by high speed communication network.
- Simple message passing system for nodes to exchange information.
- Main memory of individual nodes is used to cache pieces of shared memory space.
- **Shared memory exist only virtually.**
- Memory mapping manager routine maps local memory to shared virtual memory.

Advantages of distributed shared memory

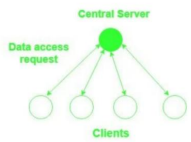
1. **Simpler Abstraction:** shields the application programmers from low level concern.
2. **Better portability of distributed application programs:** The access protocol used in case of DSM is consistent with the way sequential application access data this allows for a more natural transition from sequential to distributed application.
3. **Better performance:** due to Locality of data, On demand data moment, Large memory space as total memory size is the sum of the memory size of all the nodes in the system.
4. **Flexible communication environment**
5. **On demand migration of data between processors.**

Algorithm for implementation of DSM

1. The Central Server Algorithm
2. The Migration Algorithm
3. The Read-Replication Algorithm
4. The Full–Replication Algorithm

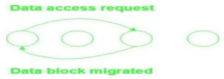
1. The Central Server Algorithm

- **Central server maintains all shared data:**
  - Read request: returns data item.
  - Write request: updates data and returns acknowledgement message.



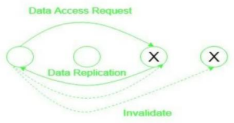
2. The Migration Algorithm

- Every data access request is forwarded to location of data while in this data is shipped to location of data access request which allows subsequent access to be performed locally.
- It allows only one node to access a shared data at a time.
- The whole block containing data item migrates instead of individual item requested.
- This algorithm provides an opportunity to integrate DSM with virtual memory provided by operating system at individual nodes.



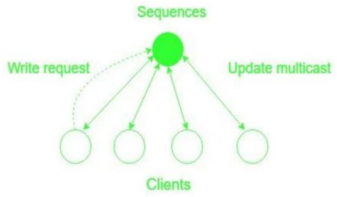
3. The Read-Replication Algorithm

- This extends the migration algorithm by replicating data blocks to multiple nodes and allowing multiple nodes to have read access or one node to have both read write access.
- After a write, all copies are invalidated or updated.
- DSM has to keep track of locations of all copies of data objects.
- Advantage:
  - The read-replication can lead to substantial performance improvements if the ratio of reads to writes is large.
  - It improves system performance by allowing multiple nodes to access data concurrently.



4. The Full–Replication Algorithm

- It is an extension of read replication algorithm which allows multiple nodes to have both read and write access to shared data blocks.
- Issue: Since many nodes can write shared data concurrently, the access to shared data must be controlled to maintain it's consistency.



Synchronization Mechanisms

The Critical Section Problem

The critical section problem is used to design a set of protocols which can ensure that the Race condition among the processes will never arise.

Critical Section is the part of a program which tries to access shared resources. That resource may be any resource in a computer like a memory location, Data structure, CPU or any IO device.

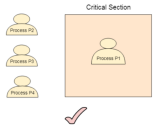
The critical section cannot be executed by more than one process at the same time; operating system faces the difficulties in allowing and disallowing the processes from entering the critical section.

In order to synchronize the cooperative processes, our main task is to solve the critical section problem. We need to provide a solution in such a way that the following conditions can be satisfied.

Requirements/Condition of Synchronization mechanisms

1. **Mutual Exclusion**

Our solution must provide mutual exclusion. By Mutual Exclusion, we mean that if one process is executing inside critical section then the other process must not enter in the critical section.



2. **Progress**

Progress means that if one process doesn't need to execute into critical section then it should not stop other processes to get into the critical section.

3. **Bounded Waiting**

We should be able to predict the waiting time for every process to get into the critical section. The process must not be endlessly waiting for getting into the critical section.

4. **Architectural Neutrality**

Our mechanism must be architectural natural. It means that if our solution is working fine on one architecture then it should also run on the other ones as well.

Lock Variable

This is the simplest synchronization mechanism. This is a busy waiting solution which can be used for more than two processes.

In this mechanism, a Lock variable **lock** is used. Two values of lock can be possible, either 0 or 1. Lock value 0 means that the critical section is vacant while the lock value 1 means that it is occupied.

A process which wants to get into the critical section first checks the value of the lock variable. If it is 0 then it sets the value of lock as 1 and enters into the critical section, otherwise it waits.

```
Entry Section →
While (lock! = 0);
Lock = 1;
//Critical Section
Exit Section →
Lock =0;
```

if we look at the Pseudo Code, we find that there are three sections in the code.

Entry Section, Critical Section and the exit section.

Initially the value of **lock variable** is **0**. The process which needs to get into the **critical section**, enters into the entry section and checks the condition provided in the while loop.

The process will wait infinitely until the value of **lock** is 1 (that is implied by while loop). Since, at the very first time critical section is vacant hence the process will enter the critical section by setting the lock variable as 1.

When the process exits from the critical section, then in the exit section, it reassigns the value of **lock** as 0.

Introduction to Deadlock

Every process needs some resources to complete its execution. However, the resource is granted in a sequential order.

1. The process requests for some resource.
2. OS grant the resource if it is available otherwise let the process waits.
3. The process uses it and release on the completion.

A Deadlock is a situation where each of the computer process waits for a resource which is being assigned to some another process.

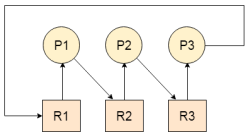
In this situation, none of the process gets executed since the resource it needs, is held by some other process which is also waiting for some other resource to be released.

EXAMPLE

Let us assume that there are three processes P1, P2 and P3. There are three different resources R1, R2 and R3. R1 is assigned to P1, R2 is assigned to P2 and R3 is assigned to P3.

After some time, P1 demands for R1 which is being used by P2. P1 halts its execution since it can't complete without R2. P2 also demands for R3 which is being used by P3. P2 also stops its execution because it can't continue without R3. P3 also demands for R1 which is being used by P1 therefore P3 also stops its execution.

In this scenario, a cycle is being formed among the three processes. None of the process is progressing and they are all waiting. The computer becomes unresponsive since all the processes got blocked.



Necessary conditions for Deadlocks

1. **Mutual Exclusion**

   A resource can only be shared in mutually exclusive manner. It implies, if two process cannot use the same resource at the same time.

2. **Hold and Wait**

   A process waits for some resources while holding another resource at the same time.

3. **No preemption**

   The process which once scheduled will be executed till the completion. No other process can be scheduled by the scheduler meanwhile.

4. **Circular Wait**

   All the processes must be waiting for the resources in a cyclic manner so that the last process is waiting for the resource which is being held by the first process.

Strategies for handling Deadlock

Deadlock Ignorance

In this approach, the Operating system assumes that deadlock never occurs. It simply ignores deadlock. This approach is best suitable for a single end user system where User uses the system only for browsing and all other normal stuff.

In these types of systems, the user has to simply restart the computer in the case of deadlock. Windows and Linux are mainly using this approach.

Deadlock prevention

Deadlock happens only when Mutual Exclusion, hold and wait, No preemption and circular wait holds simultaneously. If it is possible to violate one of the four conditions at any time then the deadlock can never occur in the system.

Deadlock avoidance

the operating system checks whether the system is in safe state or in unsafe state at every step which the operating system performs.

The process continues until the system is in safe state. Once the system moves to unsafe state, the OS has to backtrack one step.

The OS reviews each allocation so that the allocation doesn't cause the deadlock in the system.

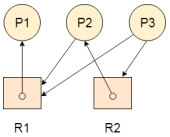
the process should declare the maximum number of resources of each type it may ever need. The Deadlock avoidance algorithm examines the resource allocations so that there can never be a circular wait condition.

Example

3 processes P1, P2 and P3, and two types of resources R1 and R2. The resources are having 1 instance each.

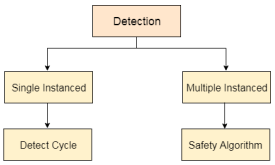
According to the graph, R1 is being used by P1, P2 is holding R2 and waiting for R1, P3 is waiting for R1 as well as R2.

The graph is deadlock free since no cycle is being formed in the graph.



Deadlock detection and recovery

This approach let the processes fall in deadlock and then periodically check whether deadlock occur in the system or not. If it occurs then it applies some of the recovery methods to the system to get rid of deadlock.



Distributed Deadlock Detection

Distributed deadlocks can occur when distributed transactions or concurrency control are utilized in distributed systems.

In a distributed system, deadlock cannot be prevented nor avoided because the system is too vast. As a result, only deadlock detection is possible. The following are required for distributed system deadlock detection techniques:

1. **Progress**

The method may detect all the deadlocks in the system.

2. **Safety**

The approach must be capable of detecting all system deadlocks.

Approaches to detect deadlock in the distributed system

1. **Centralized Approach**

   Only one resource is responsible for detecting deadlock in the centralized method, and it is simple and easy to use.

   The excessive workload on a single node and single-point failure (i.e., the entire system is dependent on one node, and if that node fails, the entire system crashes), making the system less reliable.

2. **Hierarchical Approach**

   it is the integration of both centralized and distributed approaches to deadlock detection. In this strategy, a single node handles a set of selected nodes or clusters of nodes that are in charge of deadlock detection.

3. **Distributed Approach**

   In this technique various nodes work to detect deadlocks. There is no single point of failure as the workload is equally spread among all nodes. It also helps to increase the speed of deadlock detection.

Synchronization Mechanisms

Why process cooperation?

Information sharing:
Several users may need to access same piece of information, so there should be an environment to allow concurrent access to these types of resources.

Computation speedup:
To make any task run faster , It should be divided into subtasks, each of which will be executing in parallel with the others. Speedup can achieved if computer has multiple processing elements such as CUP or I/O channels.

Modularity: Constructing system in a modular fashion, by dividing the system function into separate processes.

Cooperating processes using producer-consumer problem

- In Producer – Consumer problem a producer process produces information that is consumed by a consumer process. For example, a print program produces characters that are consumed by the printer driver.
- A producer can produce one item while the consumer is consuming another item. The producer and consumer must be synchronized, so that the consumer does not try to consume an item that has not yet been produced. In this situation, the consumer must wait until an item is produced.

Process Synchronization

Process Synchronization means coordinating the execution of processes such that no two processes access the same shared resources and data.

It is required in a multi-process system where multiple processes run together, and more than one process tries to gain access to the same shared resource or data at the same time.

Changes made in one process aren't reflected when another process accesses the same shared data. It is necessary that processes are synchronized with each other as it helps avoid the inconsistency of shared data.

For example: A process P1 tries changing data in a particular memory location. At the same time another process P2 tries reading data from the same memory location. Thus, there is a high probability that the data being read by the second process is incorrect.

Race Condition in OS

When more than one processes execute the same code or access the same memory/shared variable, it is possible that the output or value of the shared variable is wrong.

In this condition, all processes race ahead in order to prove that their output is correct. This situation is known as race condition.

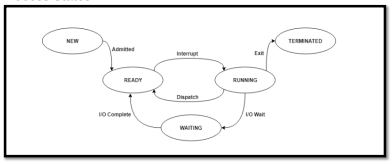
When multiple processes access and manipulate the same data concurrently the outcome depends on the order in which these processes accessed the shared data. When the output of multiple thread execution differs according to the order in which the threads execute, a race condition occurs.

Unit I

TOPIC :- Process Synchronization

Define Process: A process is an active program i.e a program that is under execution. It is more than the program code as it includes the program counter, process stack, registers, program code etc.

Process States



- **New** - The process is in the new state when it has just been created.
- **Ready** - The process is waiting to be assigned the processor by the short-term scheduler.
- **Running** - The process instructions are being executed by the processor.
- **Waiting** - The process is waiting for some event such as I/O to occur.
- **Terminated** - The process has completed its execution.

Types of Process

Processes are two types based on their types of categories.

1. **Independent process.**
2. **Cooperating process.**

Independent process is the process that can not affect or be affected by the other processes. Independent processes **does not share any data** like temporary or persistent with any other process.

Cooperating processes Execution of one process affects the execution of the other. Thus, it is necessary that these processes are synchronized in order to guarantee the order of execution.