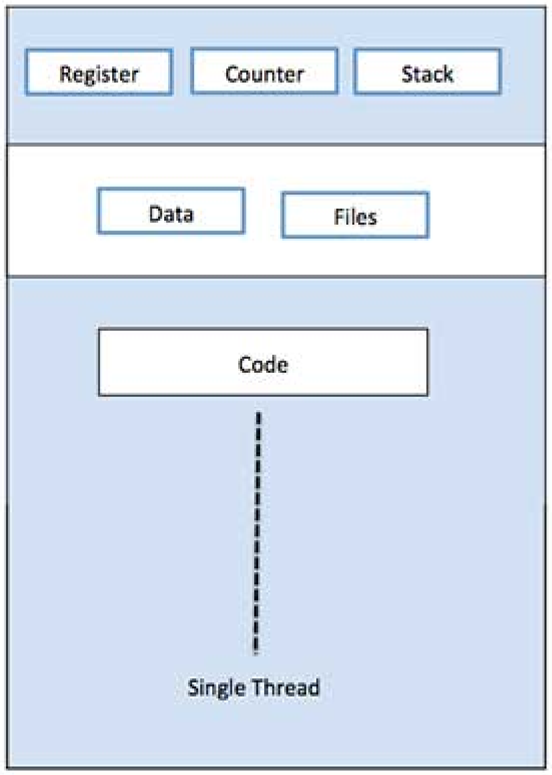
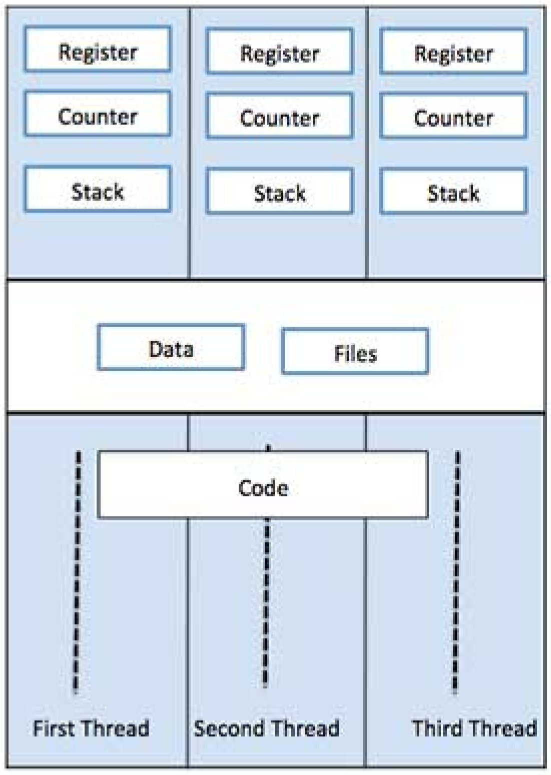
Unit -2

Thread

A thread is a flow of execution through the process code, with its own program counter that keeps track of which instruction to execute next, system registers which hold its current working variables, and a stack which contains the execution history.
A thread shares with its peer threads few information like code segment, data segment and open files. When one thread alters a code segment memory item, all other threads see that.
A thread is also called a **lightweight process**. Threads provide a way to improve application performance through parallelism. Threads represent a software approach to improving performance of operating system by reducing the overhead thread is equivalent to a classical process.
Each thread belongs to exactly one process and no thread can exist outside a process. Each thread represents a separate flow of control. Threads have been successfully used in implementing network servers and web server. They also provide a suitable foundation for parallel execution of applications on shared memory multiprocessors. The following figure shows the working of a single-threaded and a multithreaded process.



Single Process P with single thread



Single Process P with three threads

Difference between Process and Thread

	Process	Thread
1	Process is heavy weight or resource intensive.	Thread is light weight, taking lesser resources than a process.
2	Process switching needs interaction with operating system.	Thread switching does not need to interact with operating system.
3	In multiple processing environments, each process executes the same code but has its own memory and file resources.	All threads can share same set of open files, child processes.
4	If one process is blocked, then no other process can execute until the first process is unblocked.	While one thread is blocked and waiting, a second thread in the same task can run.
5	Multiple processes without using threads use more resources.	Multiple threaded processes use fewer resources.
6	In multiple processes each process operates independently of the others.	One thread can read, write or change another thread's data

Advantages of Thread

- Threads minimize the context switching time.
- Use of threads provides concurrency within a process.
- Efficient communication.
- It is more economical to create and context switch threads.
- Threads allow utilization of multiprocessor architectures to a greater scale and efficiency.

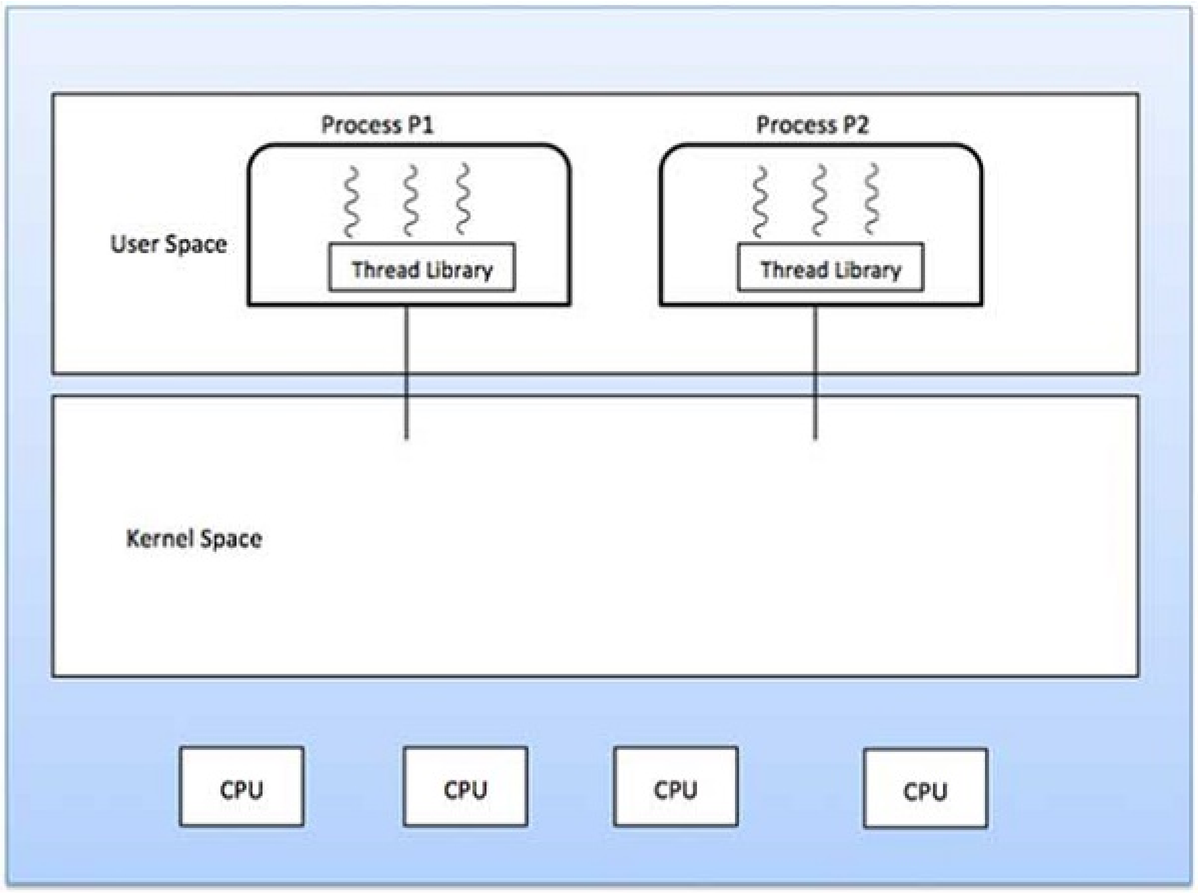
Types of Thread

Threads are implemented in following two ways −

- **User Level Threads** − User managed threads.
- **Kernel Level Threads** − Operating System managed threads acting on kernel, an operating system core.

User Level Threads

In this case, the thread management kernel is not aware of the existence of threads. The thread library contains code for creating and destroying threads, for passing message and data between threads, for scheduling thread execution and for saving and restoring thread contexts. The application starts with a single thread.



Advantages

- Thread switching does not require Kernel mode privileges.
- User level thread can run on any operating system.
- Scheduling can be application specific in the user level thread.
- User level threads are fast to create and manage.

Disadvantages

- In a typical operating system, most system calls are blocking.
- Multithreaded application cannot take advantage of multiprocessing.

Kernel Level Threads

In this case, thread management is done by the Kernel. There is no thread management code in the application area. Kernel threads are supported directly by the operating system. Any application can be programmed to be multithreaded. All of the threads within an application are supported within a single process.

The Kernel maintains context information for the process as a whole and for individuals threads within the process. Scheduling by the Kernel is done on a thread basis. The Kernel performs thread creation, scheduling and management in Kernel space. Kernel threads are generally slower to create and manage than the user threads.

Advantages

- Kernel can simultaneously schedule multiple threads from the same process on multiple processes.
- If one thread in a process is blocked, the Kernel can schedule another thread of the same process.
- Kernel routines themselves can be multithreaded.

Disadvantages

- Kernel threads are generally slower to create and manage than the user threads.
- Transfer of control from one thread to another within the same process requires a mode switch to the Kernel.

Multithreading Models

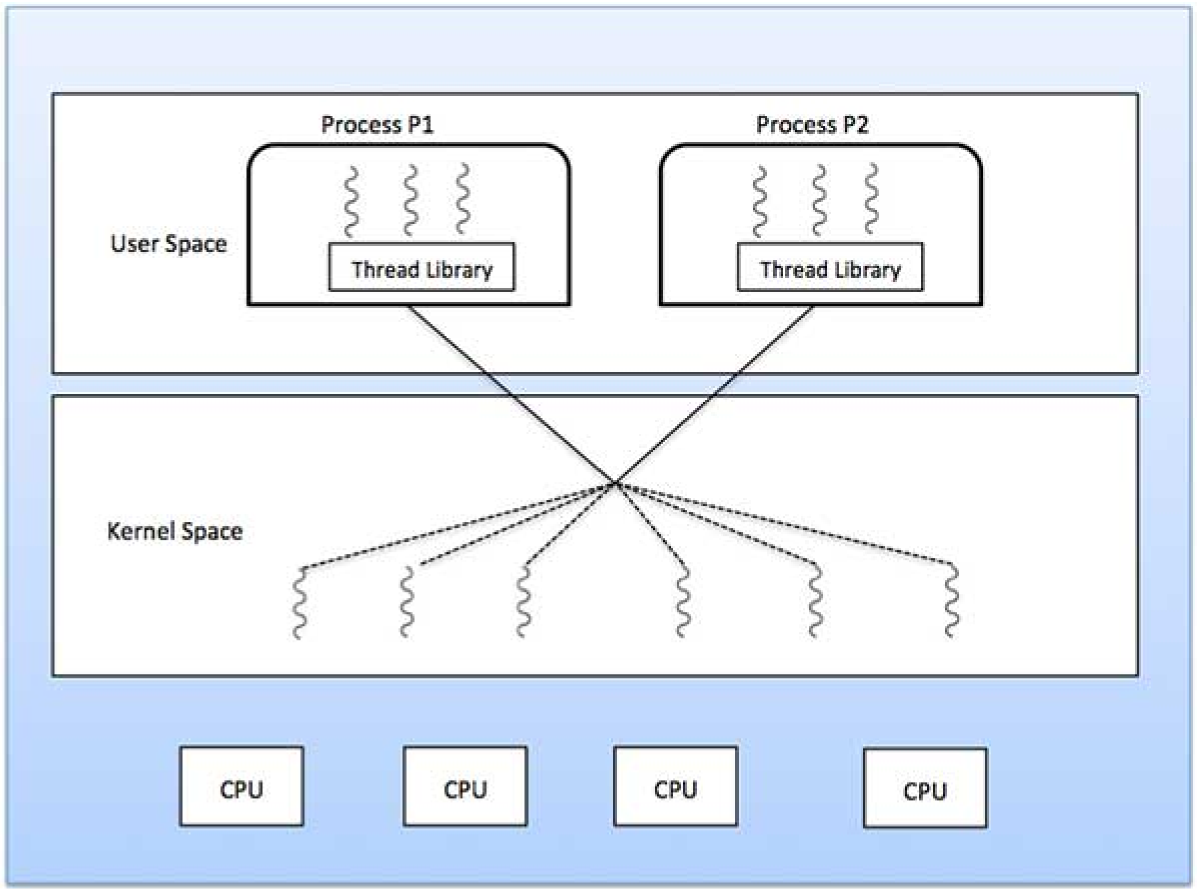
Some operating system provides a combined user level thread and Kernel level thread facility. Solaris is a good example of this combined approach. In a combined system, multiple threads within the same application can run in parallel on multiple processors and a blocking system call need not block the entire process. Multithreading models are three types

- Many to many relationship.
- Many to one relationship.
- One to one relationship.

Many to Many Models

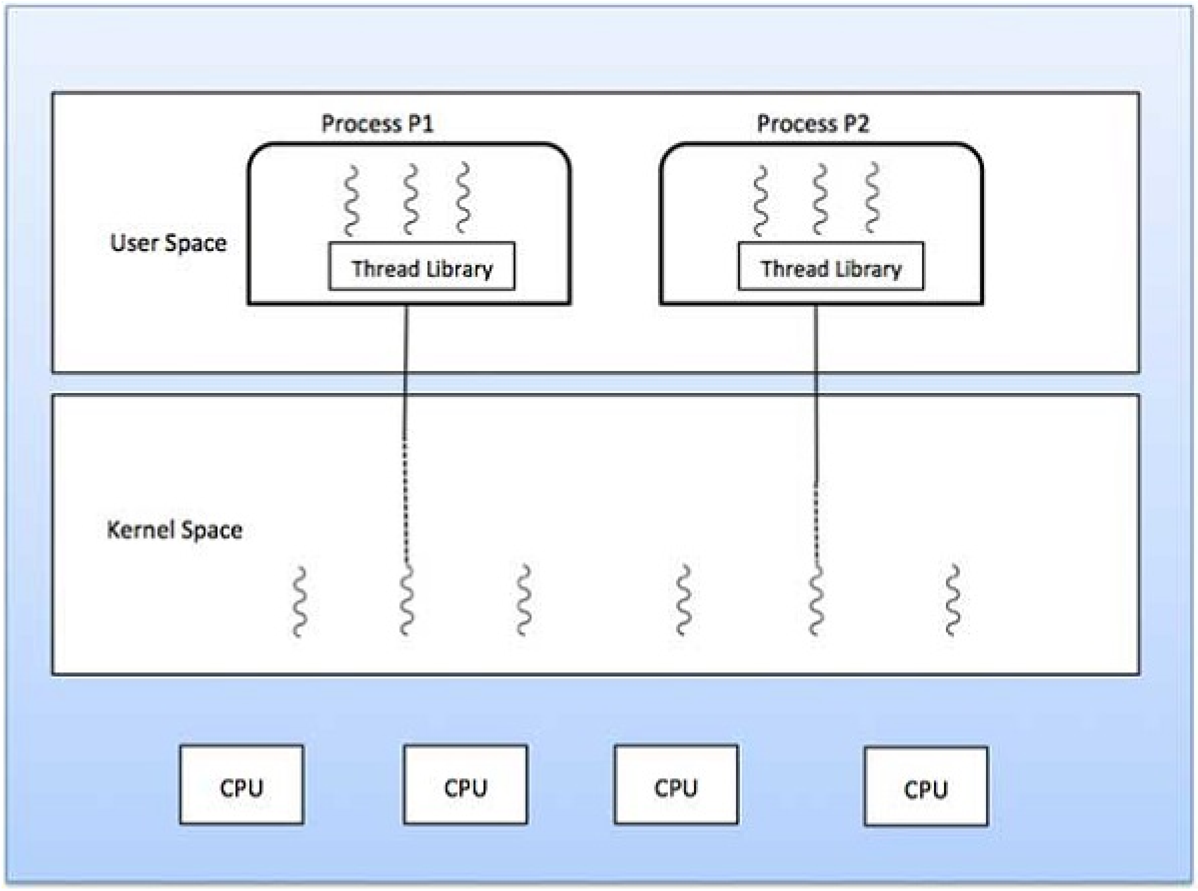
The many-to-many model multiplexes any number of user threads onto an equal or smaller number of kernel threads.

The following diagram shows the many-to-many threading model where 6 user level threads are multiplexing with 6 kernel level threads. In this model, developers can create as many user threads as necessary and the corresponding Kernel threads can run in parallel on a multiprocessor machine. This model provides the best accuracy on concurrency and when a thread performs a blocking system call, the kernel can schedule another thread for execution.



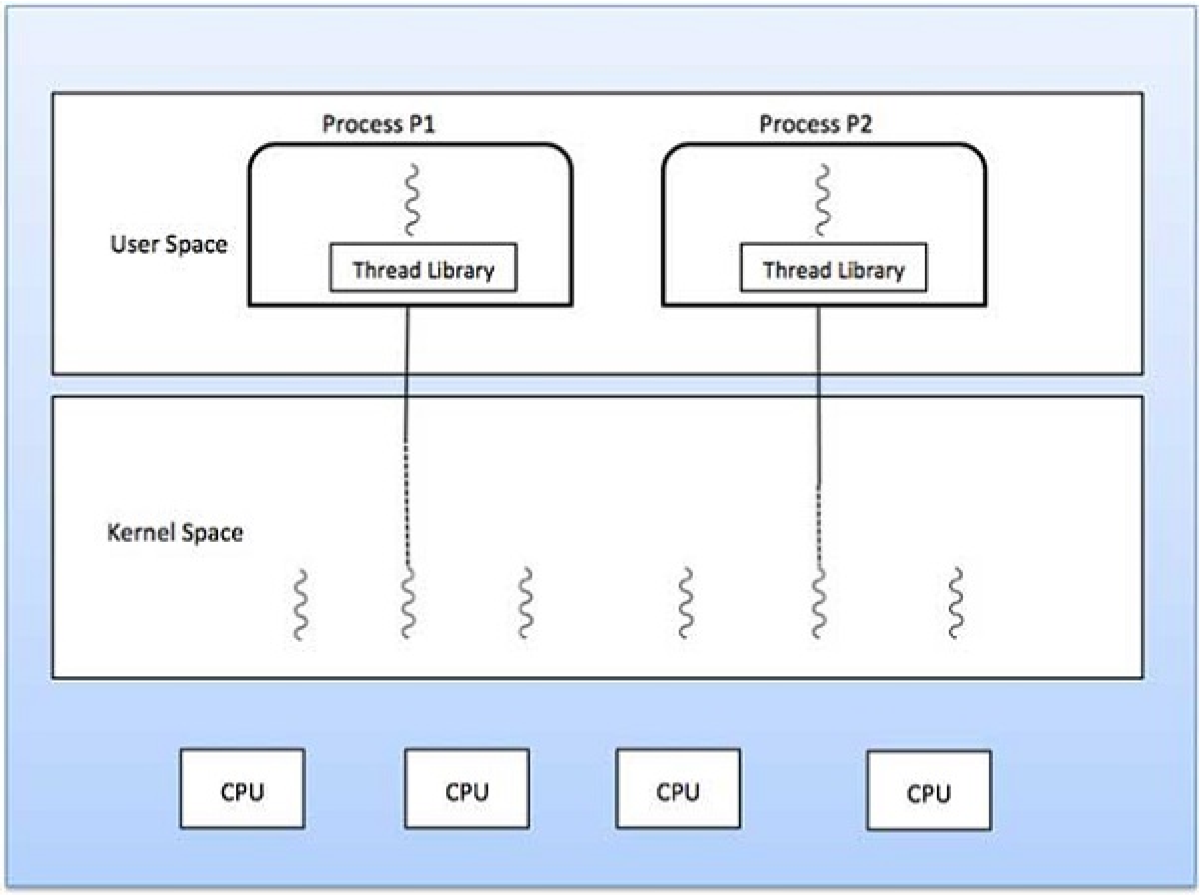
Many to One Model

Many-to-one model maps many user level threads to one Kernel-level thread. Thread management is done in user space by the thread library. When thread makes a blocking system call, the entire process will be blocked. Only one thread can access the Kernel at a time, so multiple threads are unable to run in parallel on multiprocessors.
If the user-level thread libraries are implemented in the operating system in such a way that the system does not support them, then the Kernel threads use the many-to-one relationship modes.



One to One Model

There is one-to-one relationship of user-level thread to the kernel-level thread. This model provides more concurrency than the many-to-one model. It also allows another thread to run when a thread makes a blocking system call. It supports multiple threads to execute in parallel on microprocessors. Disadvantage of this model is that creating user thread requires the corresponding Kernel thread. OS/2, Windows NT and windows 2000 use one to one relationship model.



Difference between User-Level & Kernel-Level Thread

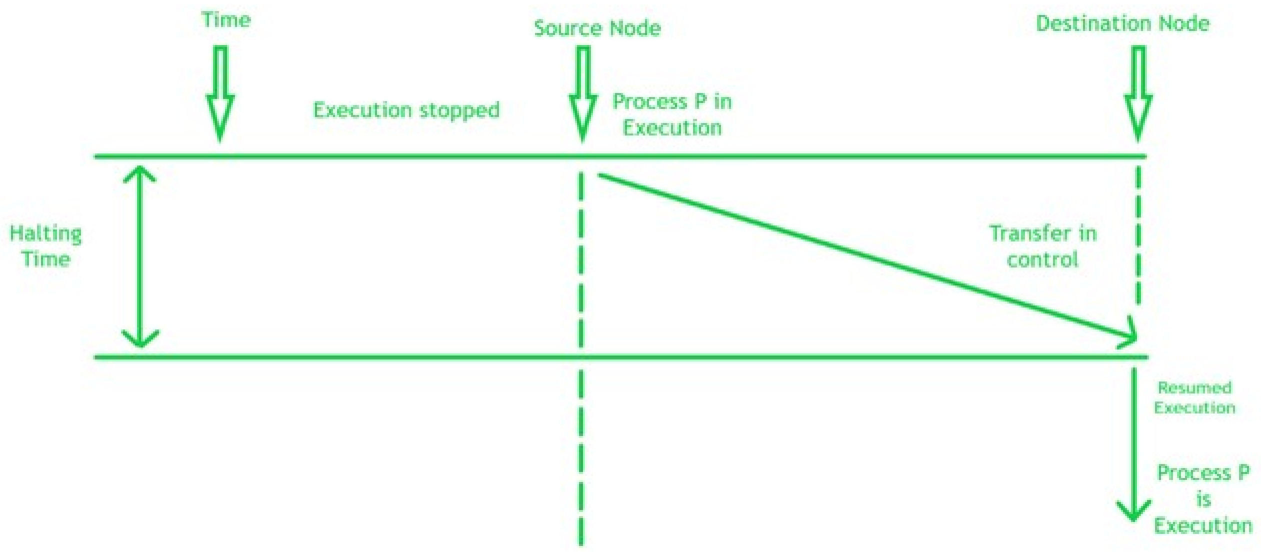
S.N.	User-Level Threads	Kernel-Level Thread
1	User-level threads are faster to create and manage.	Kernel-level threads are slower to create and manage.
2	Implementation is by a thread library at the user level.	Operating system supports creation of Kernel threads.
3	User-level thread is generic and can run on any operating system.	Kernel-level thread is specific to the operating system.

4	Multi-threaded applications cannot take advantage of multiprocessing.	Kernel routines themselves can be multithreaded

Code Migration

Process Migration in Distributed System

A process is essentially a program in execution. The execution of a process should advance in a sequential design. A process is characterized as an entity that addresses the essential unit of work to be executed in the system.
Process migration is a particular type of process management by which processes are moved starting with one computing environment then onto the next.



There are two types of Process Migration:

- **Non-preemptive process:** If a process is moved before it begins execution on its source node which is known as a non-preemptive process.
- **Preemptive process:** If a process is moved at the time of its execution that is known as preemptive process migration. Preemptive process migration is all the more expensive in comparison to the non-preemptive on the grounds that the process environment should go with the process to its new node.
-

Why use Process Migration?

The reason to use process migration are:

- **Dynamic Load Balancing:** It permits processes to exploit less stacked nodes by relocating from overloaded ones.
- **Accessibility:** Processes that inhibit defective nodes can be moved to other perfect nodes.
- **System Administration:** Processes that inhabit a node if it is going through system maintenance can be moved to different nodes.
- **The locality of data:** Processes can exploit the region of information or other extraordinary abilities of a specific node.
- **Mobility:** Processes can be relocated from a hand-operated device or computer to an automatic server-based computer before the device gets detached from the network.
- **Recovery of faults:** The component to stop, transport and resume a process is actually valuable to support in recovering the fault in applications that are based on transactions.

What are the steps involved in Process Migration?

The steps which are involved in migrating the process are:

- The process is chosen for migration.
- Choose the destination node for the process to be relocated.
- Move the chosen process to the destination node.

The subcategories to migrate a process are:

- The process is halted on its source node and is restarted on its destination node.
- The address space of the process is transferred from its source node to its destination node.
- Message forwarding is implied for the transferred process.
- Managing the communication between collaborating processes that have been isolated because of process migration.
-

Methods of Process Migration

The methods of Process Migration are:

1. Homogeneous Process Migration: Homogeneous process migration implies relocating a process in a homogeneous environment where all systems have a similar operating system as well as architecture. There are two unique strategies for performing process migration. These are i) User-level process migration ii) Kernel level process migration.

- **User-level process migration:** In this procedure, process migration is managed without converting the operating system kernel. User-level migration executions are more simple to create and handle but have usually two issues: i) Kernel state is not accessible by them. ii) They should cross the kernel limit utilizing kernel demands which are slow and expensive.
- **Kernel level process migration:** In this procedure, process migration is finished by adjusting the operating system kernel. Accordingly, process migration will become more simple and more proficient. This facility permits the migration process to be done faster and relocate more types of processes.

Homogeneous Process Migration Algorithms:
There are five fundamental calculations for homogeneous process migration:

- Total Copy Algorithm
- Pre-Copy Algorithm
- Demand Page Algorithm
- File Server Algorithm
- Freeze Free Algorithm

2. Heterogeneous Process Migration: Heterogeneous process migration is the relocation of the process across machine architectures and operating systems. Clearly, it is more complex than the homogeneous case since it should review the machine and operating designs and attributes, as well as send similar data as homogeneous process migration including process state, address space, file, and correspondence data. Heterogeneous process migration is particularly appropriate in the portable environment where is almost certain that the portable unit and the base help station will be different machine types. It would be attractive to relocate a process from the versatile unit to the base station as well as the other way around during calculation. In most cases, this couldn't be accomplished by homogeneous migration. There are four essential types of heterogeneous migration:

- **Passive object:** The information is moved and should be translated
- **Active object, move when inactive:** The process is relocated at the point when it isn't executing. The code exists in the two areas, and just the information is moved and translated.
- **Active object, interpreted code:** The process is executing through an interpreter so just information and interpreter state need to be moved.
- **Active object, native code:** Both code and information should be translated as they are accumulated for a particular architecture.

Mutual exclusion in distributed system

Mutual exclusion is a concurrency control property which is introduced to prevent race conditions. It is the requirement that a process can not enter its critical section while another concurrent process is currently present or executing in its critical section i.e only one process is allowed to execute the critical section at any given instance of time.

Mutual exclusion in single computer system Vs. distributed system: In single computer system, memory and other resources are shared between different processes. The status of shared resources and the status of users is easily available in the shared memory so with the help of shared variable (For example: Semaphores) mutual exclusion problem can be easily solved. In Distributed systems, we neither have shared memory nor a common physical clock and there for we can not solve mutual exclusion problem using shared variables. To eliminate the mutual exclusion problem in distributed system approach based on message passing is used. A site in distributed system do not have complete information of state of the system due to lack of shared memory and a common physical clock.

Requirements of Mutual exclusion Algorithm:

- **No Deadlock:** Two or more site should not endlessly wait for any message that will never arrive.
- **No Starvation:** Every site who wants to execute critical section should get an opportunity to execute it in finite time. Any site should not wait indefinitely to execute critical section while other site are repeatedly executing critical section
- **Fairness:** Each site should get a fair chance to execute critical section. Any request to execute critical section must be executed in the order they are made i.e Critical section execution requests should be executed in the order of their arrival in the system.
- **Fault Tolerance:** In case of failure, it should be able to recognize it by itself in order to continue functioning without any disruption.

Some points are need to be taken in consideration to understand mutual exclusion fully :
0 seconds of 15 secondsVolume 0%

1) It is an issue/problem which frequently arises when concurrent access to shared resources by several sites is involved. For example, directory management where updates and reads to a directory must be done atomically to ensure correctness.
2) It is a fundamental issue in the design of distributed systems.
3) Mutual exclusion for a single computer is not applicable for the shared resources since it involves resource distribution, transmission delays, and lack of global information.

Solution to distributed mutual exclusion: As we know shared variables or a local kernel cannot be used to implement mutual exclusion in distributed systems. Message passing is a way to implement mutual exclusion. Below are the three approaches based on message passing to implement mutual exclusion in distributed systems:

1. Token Based Algorithm

- A unique token is shared among all the sites.
- If a site possesses the unique token, it is allowed to enter its critical section
- This approach uses sequence number to order requests for the critical section.
- Each requests for critical section contains a sequence number. This sequence number is used to distinguish old and current requests.
- This approach insures Mutual exclusion as the token is unique.

2. Non-token based approach:

- A site communicates with other sites in order to determine which sites should execute critical section next. This requires exchange of two or more successive round of messages among sites.
- This approach use timestamps instead of sequence number to order requests for the critical section.
- Whenever a site make request for critical section, it gets a timestamp. Timestamp is also used to resolve any conflict between critical section requests.
- All algorithm which follows non-token based approach maintains a logical clock. Logical clocks get updated according to Lamport's scheme

Example : Ricart–Agrawala Algorithm

3. Quorum based approach:

- Instead of requesting permission to execute the critical section from all other sites, Each site requests only a subset of sites which is called a quorum.
- Any two subsets of sites or Quorum contain a common site.
- This common site is responsible to ensure mutual exclusion.

Election Algorithms:

Distributed system is a collection of independent computers that do not share their memory. Each processor has its own memory and they communicate via communication networks. Communication in networks is implemented in a process on one machine communicating with a process on other machine. Many algorithms used in distributed system require a coordinator that performs functions needed by other processes in the system. **Election algorithms** are designed to choose a coordinator.

Election algorithms choose a process from group of processors to act as a coordinator. If the coordinator process crashes due to some reasons, then a new coordinator is elected on other processor. Election algorithm basically determines where a new copy of coordinator should be restarted.

Election algorithm assumes that every active process in the system has a unique priority number. The process with highest priority will be chosen as a new coordinator. Hence, when a coordinator fails, this algorithm elects that active process which has highest priority number. Then this number is send to every active process in the distributed system.

We have two election algorithms for two different configurations of distributed system.

1. The Bully Algorithm
2. The Ring Algorithm

NOTE:-

- Distributed algorithms require one process to act as a coordinator or initiator. To decide which process becomes the coordinator different types of algorithms are used.
- Election algorithms are meant for electing a coordinator process from among the currently running processes in such a manner that at any instance of time there is a single coordinator for all processes in the system.
- The goal of an election algorithm is to ensure that when an election starts it concludes with all the processes agreeing on who the coordinator should be.
- Therefore, whenever initiated, an election algorithm basically finds out which of the currently active processes has the highest priority number and then informs this to all other active processes.

2. Bully Algorithm :-

- This algorithm was proposed by Garcia-Molina.
- When the process notices that the coordinator is no longer responding to requests, it initiates an election. A process, P, holds an election as follows:

  (I) P sends an ELECTION message to all processes with higher numbers.

  (II) If no one responds, P wins the election and becomes the coordinator.

  (III) If one of the higher-ups answers, it takes over. P's job is done.

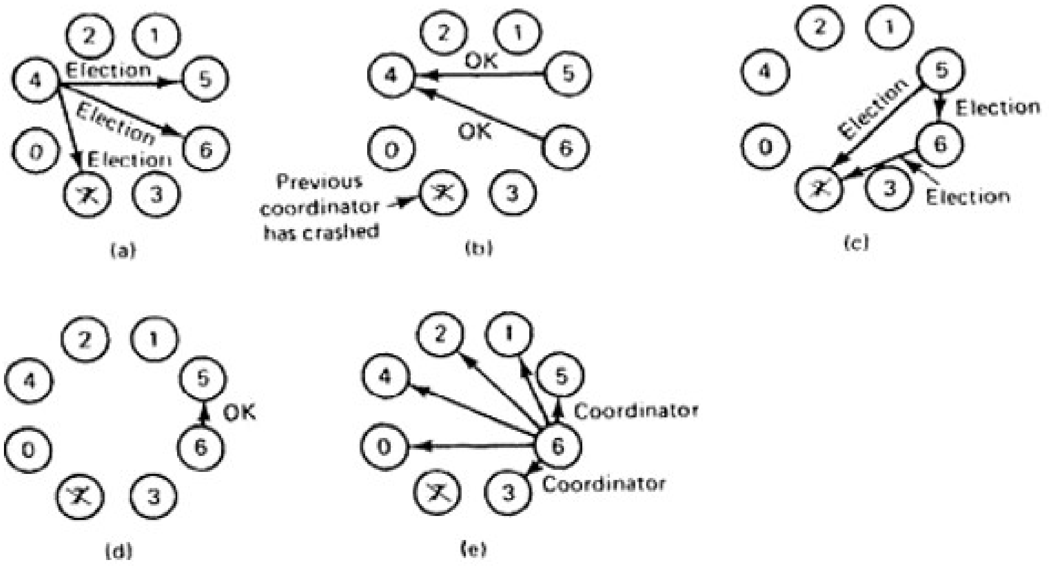
    a. A process can get an ELECTION message at any time from one of its lower numbered colleagues.

    b. When such a message arrives, the receiver sends an OK message back to the sender to indicate that he is alive and will take over. The receiver then holds an election, unless it is already holding one.

    c. All processes give up except one that is the new coordinator. It announces its victory by sending all processes a message telling them that starting immediately it is the new coordinator.

    d. If a process that was previously down comes back up, it holds an election. If it happens to the highest numbered process currently running, it will win the election and take over the coordinator's job. Thus the biggest guy in town always wins, hence the name "bully algorithm".

    e. Example
- In fig(a) a group of eight processes taken is numbered from 0 to 7. Assume that previously process 7 was the coordinator, but it has just crashed. Process 4 notices if first and sends



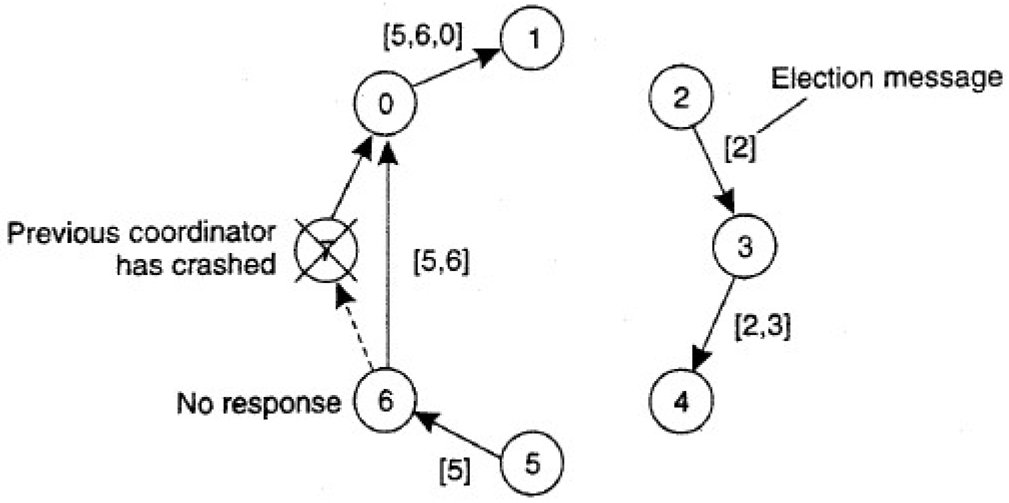
ELECTION messages to all the processes higher than it that is 5, 6 and 7.

- In fig (b) processes 5 and 6 both respond with OK. Upon getting the first of these responses, process4job is over. It knows that one of these will become the coordinator. It just sits back and waits for the winner.
- In fig(c), both 5 and 6 hold elections by each sending messages to those processes higher than itself.
- In fig(d), process 6 tells 5 that it will take over with an OK message. At this point 6knows that 7 is dead and that (6) it is the winner. It there is state information to be collected from disk or elsewhere to pick up where the old coordinator left off, 6 must now do what is needed. When it is ready to take over, 6 announce this by sending a COORDINATOR message to all running processes. When 4 gets this message, it can now continue with the operation it was trying to do when it discovered that 7 was dead, but using 6 as the coordinator this time. In this way the failure of is handled and the work can continue.
- If process 7 is ever restarted, it will just send all the others a COORDINATOR message and bully them into submission.

3. Ring Algorithm :-

- This algorithm uses a ring for its election but does not use any token. In this algorithm it is assumed that the processes are physically or logically ordered so each processor knows its successor.
- When any process notices that a coordinator is not functioning, it builds an ELECTION message containing its own process number and sends the message to its successor. If the successor is down the sender skips over the successor and goes to the next member along the ring until a process is located.

- At each step the sender adds its own process number to the list in the message making itself a candidate to elected as coordinator
- The message gets back to the process that started it and recognizes this event as the message consists its own process number.
- At that point the message type is changed to COORDINATOR and circulated once again to inform everyone who the coordinator is and who are the new members. The coordinator is selected with the process having highest number.
- When this message is circulated once it is removed and normal work is preceded.


[5,6,0]
1
0
2
Election message
[2]
Previous coordinator has crashed
7
[5,6]
3
[2,3]
No response
6
4
[5]
5