**[Unit 4] Resource Management**

**Desirable Features of a Good Global Scheduling Algorithm**

**Fault Tolerance:**

A good global scheduling algorithm should not be stopped when system nodes are crashed or temporarily crashed. Algorithm configuration should also be even if the nodes are separated by multiple nodes.

**Scalability:**

A good global scheduling algorithm should be used for marketing, which means that the algorithm should work well even as the number of nodes increases. The scheduling algorithm will query the workload of all categories in the corrupted system and select a node with the least configuration load. For distributed applications, the configuration algorithm will balance the load between nodes but this is not the case that nodes that are more efficient as they are assigned will have a better response time and less loaded nodes may have a negative response time. so the load should be shared instead of balancing i.e. resources in the nodes should be shared among the nodes until the tasks being performed are not affected.

**No apriori knowledge about the processes:**

Scheduling algorithms work based on the characteristics and resource requirement of the processes this information should be provided by the user. This will obviously put extra overhead on users, so a good global scheduling algorithm should require less amount prior knowledge.

### ****Quick decision-making capability:****

Scheduling algorithms should be fast and should provide the best possible optimal decision in less amount of time. A good process-scheduling algorithm must make quick decisions about the assignment of processes to processors. This is an extremely important aspect of the algorithms and makes many potential solutions unsuitable. For example, an algorithm that models the system by a mathematical program and solves it online is unsuitable because it does not meet this requirement. Heuristic methods requiring less computational effort while providing near-optimal results are therefore normally preferable to exhaustive (optimal) solution methods.

### ****Stability:****

The useless migration of processes should be prevented ie if node n1 is idle and n2 and n3 have multiple processes, then node n2 and node n2 will send processes to node n1 causing node n1 to overload node n1 will move these processes to other nodes. which is a useless overhead to the system, so good scheduling algorithms should be stable and prevent useless migration.

### ****Dynamic in nature:****

Process allocations should be able to mean that allocations should be made based on the current load of the system but not based on specific planned conditions. The scheduler should also be able to move processes from one node to another duplicate so that the distribution is based on the current system load.

**Task assignment Approach**

### Resource Management:

One of the functions of system management in distributed systems is Resource Management. When a user requests the execution of the process, the resource manager performs the allocation of resources to the process submitted by the user for execution. In addition, the resource manager routes process to appropriate nodes (processors) based on assignments.

Multiple resources are available in the distributed system so there is a need for system transparency for the user. There can be a logical or a physical resource in the system. For example, data files in sharing mode, Central Processing Unit (CPU), etc.

As the name implies, the task assignment approach is based on the division of the process into multiple tasks. These tasks are assigned to appropriate processors to improve performance and efficiency. This approach has a major setback in that it needs prior knowledge about the features of all the participating processes. Furthermore, it does not take into account the dynamically changing state of the system. This approach’s major objective is to allocate tasks of a single process in the best possible manner as it is based on the division of tasks in a system. For that, there is a need to identify the optimal policy for its implementation.

**Working of Task Assignment Approach:**

In the working of the Task Assignment Approach, the following are the assumptions:

* The division of an individual process into tasks.
* Each task’s computing requirements and the performance in terms of the speed of each processor are known.
* The cost incurred in the processing of each task performed on every node of the system is known.
* The IPC (Inter-Process Communication) cost is known for every pair of tasks performed between nodes.
* Other limitations are also familiar, such as job resource requirements and available resources at each node, task priority connections, and so on.

**Goals of Task Assignment Algorithms:**

* Reducing Inter-Process Communication (IPC) Cost
* Quick Turnaround Time or Response Time for the whole process
* A high degree of Parallelism
* Utilization of System Resources in an effective manner

The above-mentioned goals time and again conflict. To exemplify, let us consider the goal-1 using which all the tasks of a process need to be allocated to a single node for reducing the Inter-Process Communication (IPC) Cost. If we consider goal-4 which is based on the efficient utilization of system resources that implies all the tasks of a process to be divided and processed by appropriate nodes in a system.

**Note:** The possible number of assignments of tasks to nodes:

For m tasks and n nodes= m x n

But in practice, the possible number of assignments of tasks to nodes < m x n because of the constraint for allocation of tasks to the appropriate nodes in a system due to their particular requirements like memory space, etc.

**Need for Task Assignment in a Distributed System:**

The need for task management in distributed systems was raised for achieving the set performance goals. For that optimal assignments should be carried out concerning cost and time functions such as task assignment to minimize the total execution and communication costs, completion task time, total cost of 3 (execution, communication, and interference), total execution and communication costs with the limit imposed on the number of tasks assigned to each processor, and a weighted product of cost functions of total execution and communication costs and completion task time. All these factors are countable in task allocation and turn, resulting in the best outcome of the system.

**Example of Task Assignment Approach:**

Let us suppose, there are two nodes namely n1 and n2, and six tasks namely t1, t2, t3, t4, t5, and t6. The two task assignment parameters are:

* **execution cost: xab**refers to the cost of executing a task an on node b.
* **inter-task communication cost: cij**refers to inter-task communication cost between tasks i and j.

| **Tasks** | **t1** | **t2** | **t3** | **t4** | **t5** | **t6** |
| --- | --- | --- | --- | --- | --- | --- |
| **t1** | 0 | 6 | 4 | 0 | 0 | 12 |
| **t2** | 6 | 0 | 8 | 12 | 3 | 0 |
| **t3** | 4 | 8 | 0 | 0 | 11 | 0 |
| **t4** | 0 | 12 | 0 | 0 | 5 | 0 |
| **t5** | 0 | 3 | 11 | 5 | 0 | 0 |
| **t6** | 12 | 0 | 0 | 0 | 0 | 0 |

| **Execution Cost** | | |
| --- | --- | --- |
| **Tasks** | **Nodes** | |
| **n1** | **n2** |
| **t1** | 5 | 10 |
| **t2** | 2 | infinity |
| **t3** | 4 | 4 |
| **t4** | 6 | 3 |
| **t5** | 5 | 2 |
| **t6** | infinity | 4 |

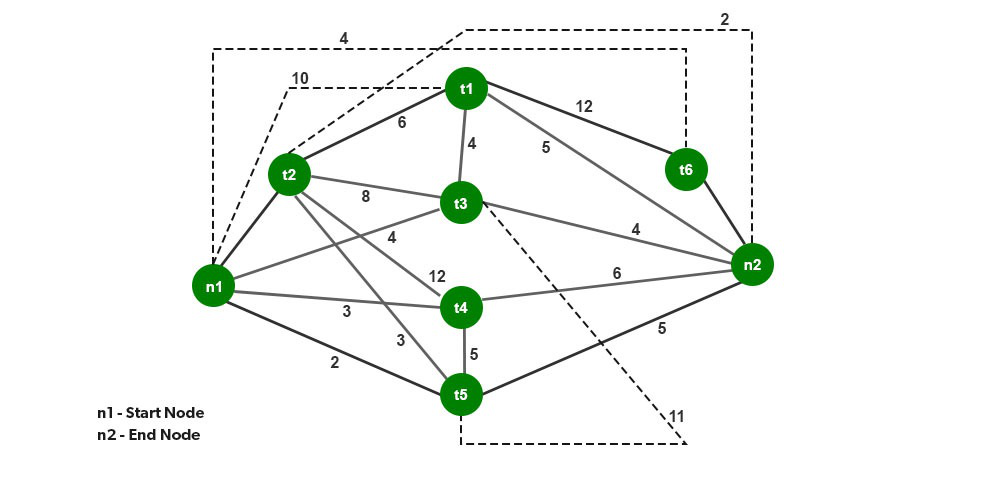
* **Note:** The execution of the task (t2) on the node (n2) and the execution of the task (t6) on the node (n1) is not possible as it can be seen from the above table of Execution costs that resources are not available.
* **Case1: Serial Assignment**

| **Task** | **Node** |
| --- | --- |
| t1 | n1 |
| t2 | n1 |
| t3 | n1 |
| t4 | n2 |
| t5 | n2 |
| t6 | n2 |

* **Cost of Execution in Serial Assignment:**
* t11 +t21 + t31 + t42 + t52 + t62 = 5 + 2+ 4 + 3 + 2 + 4
* = 20 (Refer Execution Cost table)
* **Cost of Communication in Serial Assignment:**
* = c14 + c15 +c16 + c24 + c25 + c26 + c34 + c35 + c36
* = 0 + 0+ 12 + 12 + 3 + 0 + 0 + 11 + 0
* = 38 (Refer Inter-task Communication Cost table)
* Hence, Total Cost in Serial Assignment
* = 20 + 38
* = 58
* **Case2: Optimal Assignment**

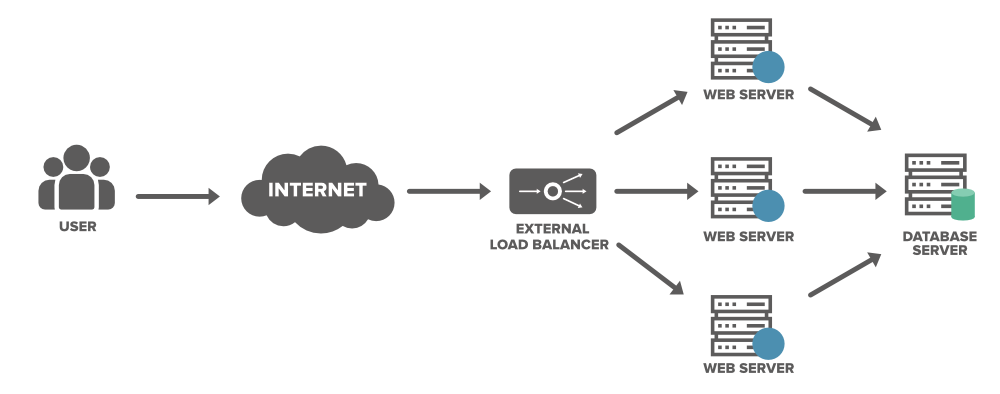
| **Task** | **Node** |
| --- | --- |
| t1 | n1 |
| t2 | n1 |
| t3 | n1 |
| t4 | n1 |
| t5 | n1 |
| t6 | n2 |

* **Cost of Execution in Optimal Assignment:**
* = t11 + t21 + t31 + t41 + t51 + t62
* = 5 + 2+ 4 + 6 + 5 + 4
* = 26 (Refer Execution Cost table)
* **Cost of Communication in Optimal Assignment:**
* = c16 + c26 + c36 + c46 + c56
* = 12 + 0+ 0 + 0 + 0
* = 12 (Refer Inter-task Communication Cost table)
* Hence, Total Cost in Optimal Assignment
* = 26 + 12
* = 38
* **Optimal Assignment using Minimal Cutset:**
* **Cutset:**The cutset of a graph refers to the set of edges that when removed makes the graph disconnected.
* **Minimal Cutset:**The minimal cutset of a graph refers to the cut which is minimum among all the cuts of the graph.



**Load Balancing Approach in Distributed System**

A load balancer is a device that acts as a reverse proxy and distributes network or application traffic across a number of servers. Load adjusting is the approach to conveying load units (i.e., occupations/assignments) across the organization which is associated with the distributed system. Load adjusting should be possible by the load balancer. The load balancer is a framework that can deal with the load and is utilized to disperse the assignments to the servers. The load balancers allocates the primary undertaking to the main server and the second assignment to the second server.



**Purpose of Load Balancing in Distributed Systems:**

* **Security:**A load balancer provide safety to your site with practically no progressions to your application.
* **Protect applications from emerging threats:**The Web Application Firewall (WAF) in the load balancer shields your site.
* **Authenticate User Access:**The load balancer can demand a username and secret key prior to conceding admittance to your site to safeguard against unapproved access.
* **Protect against DDoS attacks:**The load balancer can distinguish and drop conveyed refusal of administration (DDoS) traffic before it gets to your site.
* **Performance:**Load balancers can decrease the load on your web servers and advance traffic for a superior client experience.
* **SSL Offload:**Protecting traffic with SSL (Secure Sockets Layer) on the load balancer eliminates the upward from web servers bringing about additional assets being accessible for your web application.
* **Traffic Compression:**A load balancer can pack site traffic giving your clients a vastly improved encounter with your site.

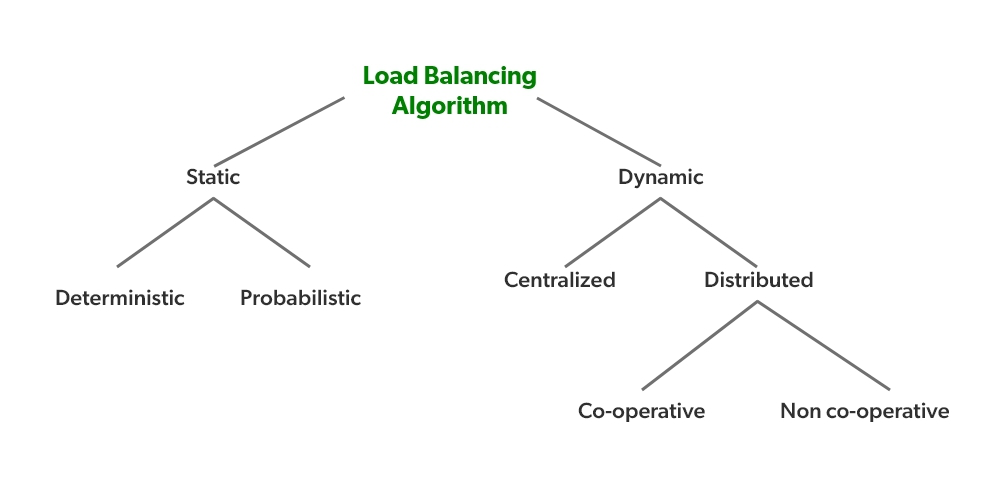
**Load Balancing Approaches:**

* Round Robin
* Least Connections
* Least Time
* Hash
* IP Hash

**Classes of Load Adjusting Calculations:**

Following are a portion of the various classes of the load adjusting calculations.

* **Static:**In this model assuming any hub/node is found with a heavy load, an assignment can be taken arbitrarily and move the undertaking to some other arbitrary system. .
* **Dynamic:**It involves the present status data for load adjusting. These are better calculations than static calculations.
* **Deterministic:**These calculations utilize processor and cycle attributes to apportion cycles to the hubs.
* **Centralized:**The framework states data is gathered by a single hub.

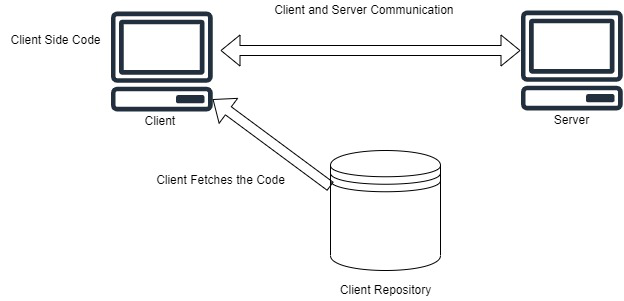


**Advantages of Load Balancing:**

* Load balancers minimize server response time and maximize throughput.
* Load balancer ensures high availability and reliability by sending requests only to online servers
* Load balancers do continuous health checks to monitor the server’s capability of handling the request.

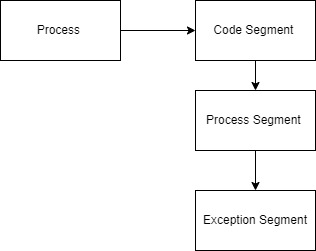
**Migration:**

Another important policy to be used by a distributed operating system that supports process migration is to decide about the total number of times a process should be allowed to migrate.



**Migration  Models:**

* Code section
* Resource section
* Execution section



* **Code section:**It contains the real code.
* **Resource fragment:**It contains a reference to outer resources required by the interaction.
* **Execution section:**It stores the ongoing execution condition of interaction, comprising private information, the stack, and the program counter.
* **Powerless movement:**In the powerless relocation just the code section will be moved.
* **Solid relocation:**In this movement, both the code fragment and the execution portion will be moved. The relocation additionally can be started by the source.

**Load-Sharing Approach in Distributed System**

Load sharing basically denotes the process of forwarding a router to share the forwarding of traffic, in case of multiple paths if available in the routing table. In case there are equal paths then the forwarding process will follow the load-sharing algorithm. In load sharing systems, all nodes share the overall workload, and the failure of some nodes increases the pressure of the rest of the nodes. The load sharing approach ensures that no node is kept idle so that each node can share the load.

For example, suppose there are two connections of servers of different bandwidths of 500Mbps and another 250Mbps. Let, there are 2 packets. Instead of sending the 2 packets to the same connection i.e. 500Mbps, 1 packet will be forwarded to the 500Mbps and another to the 250Mbps connection. Here the goal is not to use the same amount of bandwidth in two connections but to share the load so that each connection can sensibly deal with it without any traffic.

**Why use Load Sharing?**

There are several issues in designing Load Balancing Algorithms. To overcome these issues we use the load-sharing algorithm. The issues are:

* **Load assessment:** It decides how to evaluate the workload of a node in a distributed framework.
* **Process transfer:** It concludes whether the process can be executed locally or from a distance.
* **Static information exchange:** It decides how the framework loads information that can be exchanged among the nodes.
* **Location policy:** It decides the determination of an objective hub during process migration.
* **Priority assignment:** It decides the priority of execution of a bunch of nearby and remote processes on a specific node.
* **Migration restricting policy:** It decides the absolute number of times a process can move starting with one hub then onto the next.

Load Sharing algorithm includes policies like location policy, process transfer policy, state information exchange policy, load estimation policy, priority assignment policy, and migration limiting policy.

**1. Location Policies:**The location policy concludes the sender node or the receiver node of a process that will be moved inside the framework for load sharing. Depending upon the sort of node that steps up and searches globally for a reasonable node for the process, the location strategies are of the accompanying kinds:

* **Sender-inaugurated policy:** Here the sender node of the process has the priority to choose where the process has to be sent. The actively loaded nodes search for lightly loaded nodes where the workload has to be transferred to balance the pressure of traffic. Whenever a node’s load turns out to be more than the threshold esteem, it either communicates a message or arbitrarily tests different nodes individually to observe a lightly loaded node that can acknowledge at least one of its processes. In the event that a reasonable receiver node isn’t found, the node on which the process began should execute that process.
* **Receiver-inaugurated policy:**Here the receiver node of the process has the priority to choose where to receive the process. In this policy, lightly loaded nodes search for actively loaded nodes from which the execution of the process can be accepted. Whenever the load on a node falls under threshold esteem, it communicates a text message to all nodes or tests nodes individually to search for the actively loaded nodes. Some vigorously loaded node might move one of its processes if such a transfer does not reduce its load underneath the normal threshold.

**2.** **Process transfer Policy:**All or nothing approach is used in this policy. The threshold value of all the nodes is allotted as 1. A node turns into a receiver node if there is no process and on the other side a node becomes a sender node if it has more than 1 process. If the nodes turn idle then they can’t accept a new process immediately and thus it misuses the processing power  To overcome this problem, transfer the process in such a node that is expected to be idle in the future. Sometimes to ignore the processing power on the nodes, the load-sharing algorithm turns the threshold value from 1 to 2.

**3. State Information exchange Policy:**In load-sharing calculation, it is not required for the nodes to regularly exchange information, however, have to know the condition of different nodes when it is either underloaded or overloaded. Thus two sub-policies are used here:

* **Broadcast when the** **state changes:**The nodes will broadcast the state information request only when there is a change in state. In the sender-inaugurated location policy, the state information request is only broadcasted by the node when a node is overloaded. In the receiver-inaugurated location policy, the state information request is only broadcasted by the node when a node is underloaded.
* **Poll when the state changes:**In a large network the polling operation is performed. It arbitrarily asks different nodes for state information till it gets an appropriate one or it reaches the test limit.

**4. Load Estimation Policy:**Load-sharing algorithms aim to keep away from nodes from being idle yet it is adequate to know whether a node is occupied or idle. Consequently, these algorithms typically utilize the least complex load estimation policy of counting the absolute number of processes on a node.

**5. Priority Assignment Policy:** It uses some rules to determine the priority of a particular node. The rules are:

* **Selfish:**Higher priority is provided to the local process than the remote process. Thus, it has the worst response time performance for the remote process and the best response time performance for the local process.
* **Altruistic:**Higher priority is provided to the remote process than the local process. It has the best response time performance.
* **Intermediate:**The number of local and remote processes on a node decides the priority. At the point when the quantity of local processes is more or equivalent to the number of remote processes then local processes are given higher priority otherwise remote processes are given higher priority than local processes.

**6. Migration limiting policy:**This policy decides the absolute number of times a process can move. One of the accompanying two strategies might be utilized.

* **Uncontrolled:**On arrival of a remote process at a node is handled similarly as a process emerging at a node because of which any number of times a process can migrate.
* **Controlled:**A migration count parameter is used to fix the limit of the migration of a process. Thus,  a process can migrate a fixed number of times here. This removes the instability of uncontrolled strategy.

**Difference Between Load Balancing and Load Sharing in Distributed System**

### Load Balancing

[Load balancing](https://www.geeksforgeeks.org/load-balancing-in-cloud-computing/) is the practice of spreading the workload across distributed system nodes in order to optimize resource efficiency and task response time while avoiding a situation in which some nodes are substantially loaded while others are idle or performing little work.

### ****Load Sharing****

Load balancing solutions are designed to establish a dispersed network in which requests are evenly spread across several servers. Load sharing, on the other hand, includes sending a portion of the traffic to one server and the rest to another.

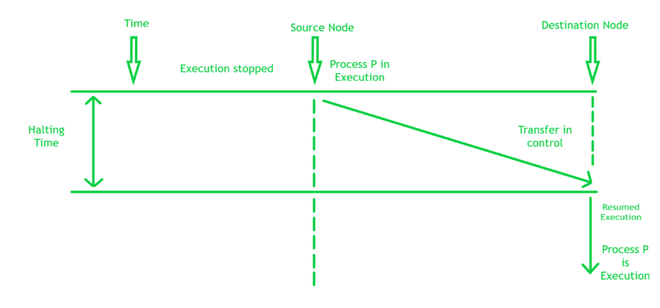
| **Load Balancing** | **Load Sharing** |
| --- | --- |
| 1. | Load balancing equally distributes network traffic or load across different channels and can be achieved using both static and dynamic load balancing techniques. | Load sharing delivers a portion of the traffic or load to one connection in the network while the remainder is routed through other channels. |
| 2. | Focuses on the notion of traffic dispersion across connections. | Works with the notion of traffic splitting across connections. |
| 3. | The creation of Ratios,  Least connections, Fastest, Round robin, and observed approaches are used in load balancing. | Load Sharing is based on the notion of sharing traffic or network load among connections based on destination IP or MAC address selections. |
| 4. | It is Uni-Directional. | It is Uni-Directional. |
| 5. | No instance is load sharing. | All instances are load sharing. |
| 6. | Accurate Load Balancing is not an easy task. | Load sharing is easy compared with load balancing. |

Load sharing is a more specific phrase that refers to the distribution of traffic across different routes, even if in an uneven manner. If you compare two traffic graphs, the two graphs should be almost the same with load balancing. However, they may be comparable with load sharing, but the traffic flow pattern will be different.

**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.

**Thread in Operating System**

**What is a Thread?**  
A thread is a path of execution within a process. A process can contain multiple threads.  
**Why Multithreading?**  
A thread is also known as lightweight process. The idea is to achieve parallelism by dividing a process into multiple threads. For example, in a browser, multiple tabs can be different threads. MS Word uses multiple threads: one thread to format the text, another thread to process inputs, etc. More advantages of multithreading are discussed below  
**Process vs Thread?**  
The primary difference is that threads within the same process run in a shared memory space, while processes run in separate memory spaces.  
Threads are not independent of one another like processes are, and as a result threads share with other threads their code section, data section, and OS resources (like open files and signals). But, like process, a thread has its own program counter (PC), register set, and stack space.  
***Advantages of Thread over Process***  
1. Responsiveness: If the process is divided into multiple threads, if one thread completes its execution, then its output can be immediately returned.

2. Faster context switch: Context switch time between threads is lower compared to process context switch. Process context switching requires more overhead from the CPU.

3. Effective utilization of multiprocessor system: If we have multiple threads in a single process, then we can schedule multiple threads on multiple processor. This will make process execution faster.

4. Resource sharing: Resources like code, data, and files can be shared among all threads within a process.  
Note: stack and registers can’t be shared among the threads. Each thread has its own stack and registers.

5. Communication: Communication between multiple threads is easier, as the threads shares common address space. while in process we have to follow some specific communication technique for communication between two process.

6. Enhanced throughput of the system: If a process is divided into multiple threads, and each thread function is considered as one job, then the number of jobs completed per unit of time is increased, thus increasing the throughput of the system.  
**Types of Threads**  
There are two types of threads.  
User Level Thread  
Kernel Level Thread