5

PROCESS AND RESOURCE MANAGEMENT

5.1 PROCESS MANAGEMENT

Process Management involves the activities aimed at defining a process, establishing responsibilities, evaluating process performance, and identifying opportunities for improvement.

5.1.1 Process Migration

- Process migration involves the transfer of sufficient amount of the state of a process from one computer to another.

Process migration is the act of transferring process between two machines.

- The process executes on the target machine.
- The aim for migration is to move processes from heavily loaded to lightly load systems.
- This enhances the performance communication.
- Processes that interact intensively can be moved to the same node to reduce communications cost.
- The hope of process migration is that it will allow for better system-wide utilization of resources.
- The long-running process may be moved because the machine it is running on will be down.
- Process can take advantage of unique hardware or software capabilities.

- Process migration enables dynamic load distribution, fault resilience, eased system administration, and data access locality.
- After migration, the process on the source system and create it on the target system.
- The process image and process control block and any links must be moved.

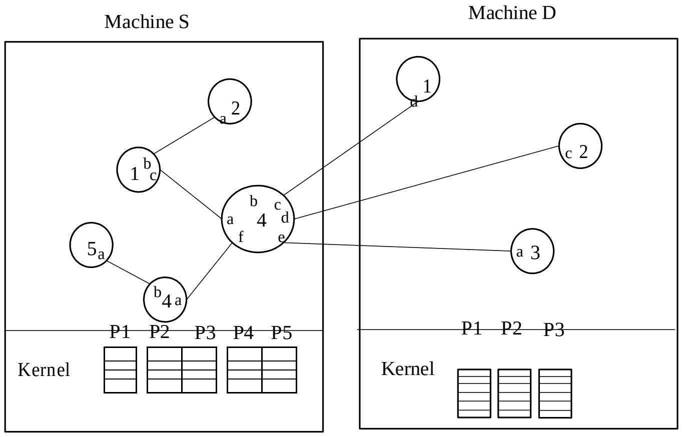


Fig 5.1 Before Migration

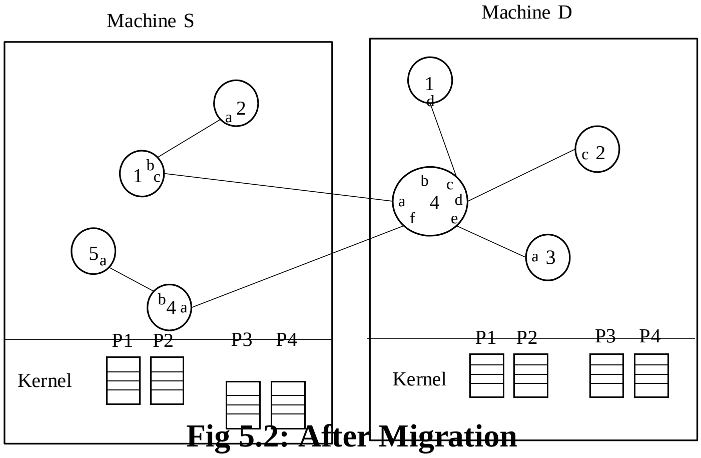


Fig 5.2: After Migration

Issues in migration:

The following questions must be addressed in process migration:

- Who initiates the migration?
- What is involved in a Migration?
- What portion of the process is migrated?
- What happens to outstanding messages and signals?

Moving PCBs:

- The movement of the process control block is straightforward and simple.
- The difficulty, from a performance point of view, concerns the process address space and any open files assigned to the process.
- There are several strategies for moving the address space and data including:
  - Eager (All):
    - This transfers the entire address space.
    - No trace of process is left behind in the original system.
    - If address space is large and if the process does not need most of it, then this approach is unnecessarily expensive.
    - Implementations that provide a checkpoint/restart facility are likely to use this approach, because it is simpler to do the check-pointing and restarting if all of the address space is localized.
  - Precopy
    - In this method, the process continues to execute on the source node while the address space is copied to the target node.
    - The pages modified on the source during the precopy operation have to be copied a second time.
    - This strategy reduces the time that a process is frozen and cannot execute during migration.
  - Eager (dirty)
    - The transfer involves only the portion of the address space that is in main memory and has been modified.
    - Any additional blocks of the virtual address space are transferred on demand.
    - The source machine is involved throughout the life of the process.
  - Copy-on-reference
    - In this method, the pages are only brought over when referenced.
    - This has the lowest initial cost of process migration.

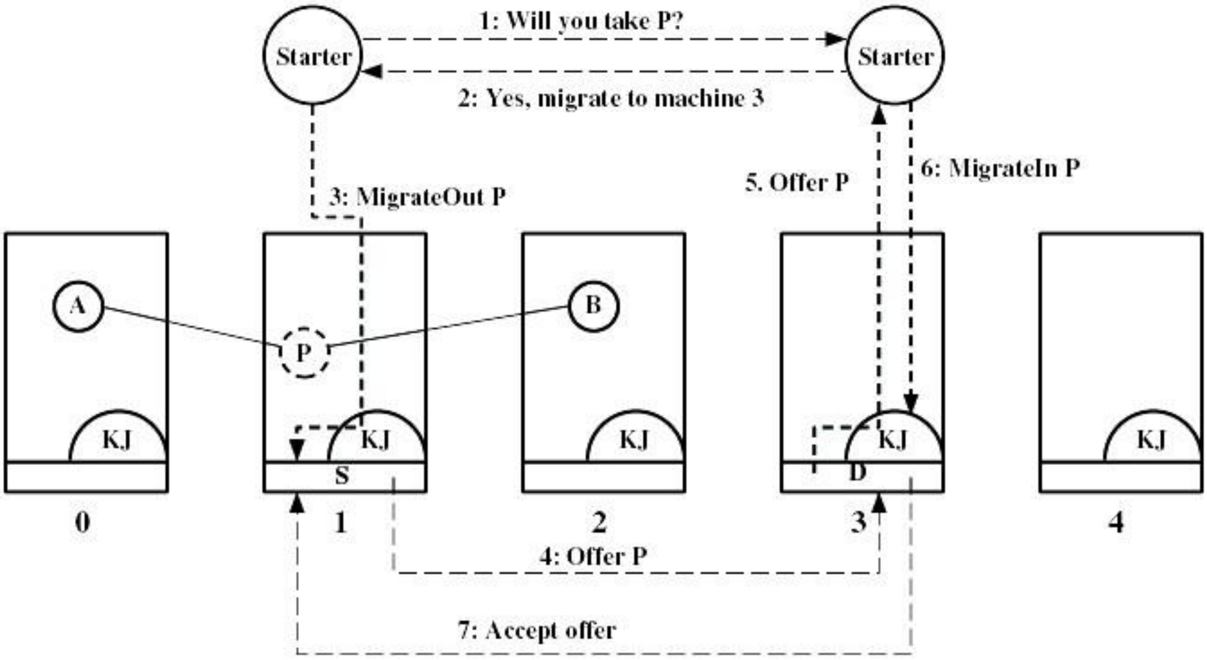


Fig 5.3: Negotiation in Migration process

Eviction

- Sometimes the destination system may refuse to accept the migration of a process to itself.
- If a workstation is idle, process may have been migrated to it.
- Once the workstation is active, it may be necessary to evict (recover) the migrated processes to provide adequate response time.

Distributed Global States

- The operating system cannot know the current state of all process in the distributed system.
- A process can only know the current state of all processes on the local system.
- Also the remote processes only know state information that is received by messages.
- This message helps to know about the state in the past.

5.2 THREADS

A thread is the entity within a process that can be scheduled for execution. All threads of a process share its virtual address space and system resources.

Each thread maintains exception handlers, a scheduling priority, thread local storage, a unique thread identifier, and a set of structures the system will use to save the thread context until it is scheduled. The ***thread context*** includes the thread's set of machine registers, the kernel stack, a thread environment block, and a user stack in the address space of the thread's process. Threads can also have their own security context, which can be used for impersonating clients.

Because threads have some of the properties of processes, they are sometimes called lightweight processes. In a process, threads allow multiple executions of streams. A process can be single threaded or multi- threaded.

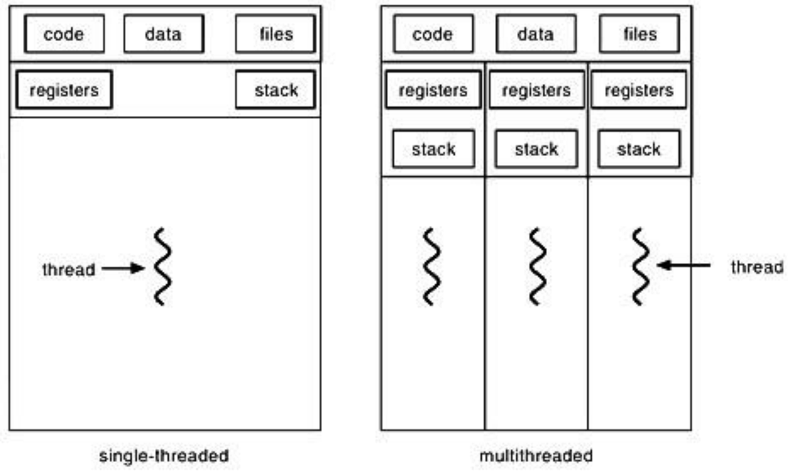


Fig 5.4: Threads

Threads provide better responsiveness, resource sharing, economy and better utilization of multiprocessor architectures.

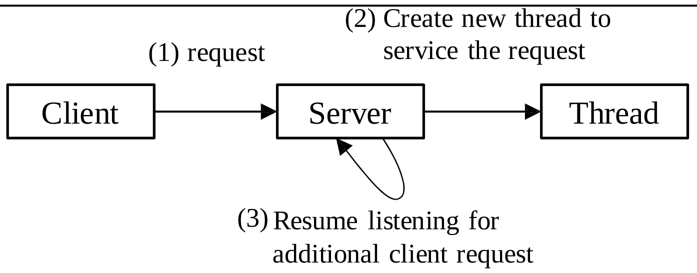

(2) Create new thread to service the request
(1) request
Client
Server
Thread
(3) Resume listening for additional client request

Fig 5.5: Multithreaded Architectures

Thread management done by user-level threads library. The actions in thread management involve:

- ❖ Creation and destruction of threads
- ❖ Message passing or data sharing between threads
- ❖ Scheduling threads
- ❖ saving/restoring threads contexts

5.2.1 Multithreading Models

There are basically three multithreading models based on how the user and kernel threads map into each other:

- ✓ Many to one
- ✓ One to one
- ✓ Many to many

Many to one Model

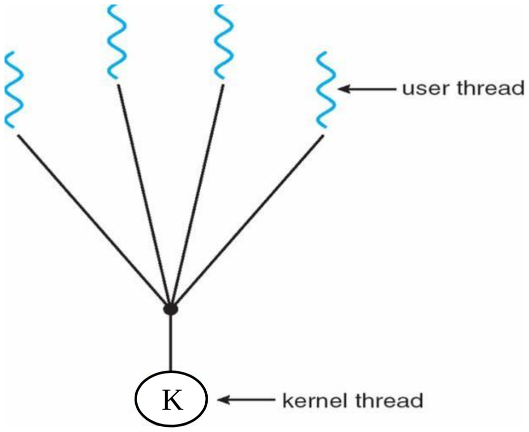


Fig 5.6: Many to one Model

In this model many user-level threads mapped to single kernel thread. Examples are Solaris Green Threads and GNU Portable Threads.

One to One Model

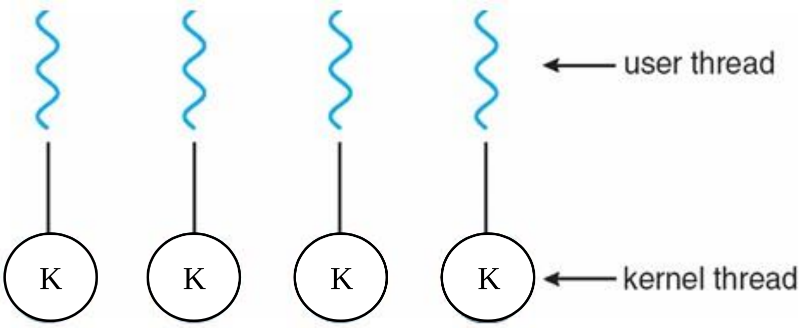


Fig 5.7: One to One Model

In this model each user-level thread maps to kernel thread. The disadvantage of this model is creating a user thread requires creating a kernel thread. Example are threads in Windows NT/XP/2000, Linux, Solaris 9 and later.

Many to many model

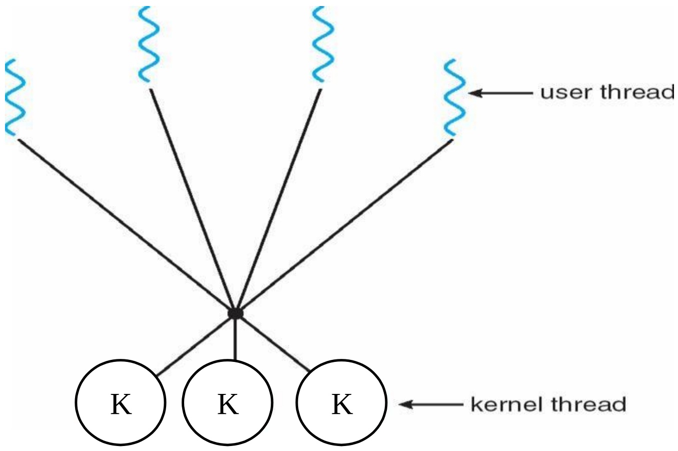


Fig 5.8: Many to many model

This model allows many user level threads to be mapped to many kernel threads. This also allows the operating system to create a sufficient number of kernel threads. Examples includes Solaris prior to version 9 and Windows NT/2000 with the *ThreadFiber* package.

Two Level Model

This is similar to Many to Many model, except that it allows a user thread to be bound to a kernel thread. Examples are threads in IRIX, HP-UX, Tru64 UNIX, Solaris 8 and earlier.

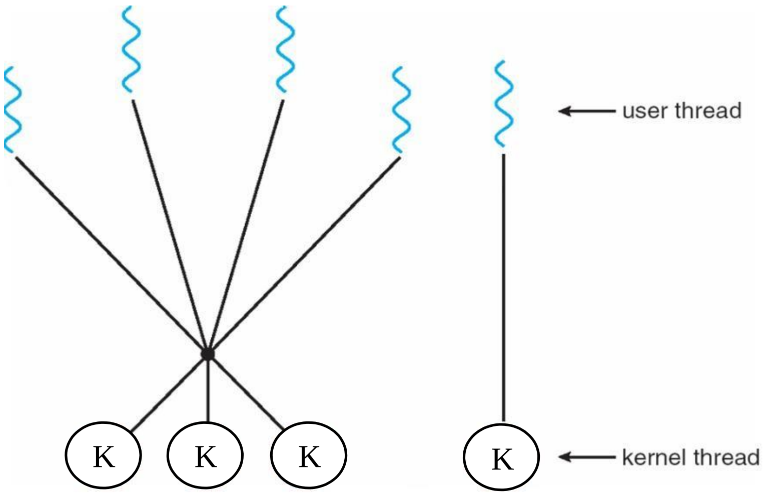


Fig 5.9: Two Level Model

5.2.2 Threading Issues

The following are the issues in threads:

- Semantics of fork() and exec() system calls

  The fork() call may duplicate either the calling thread or all threads.

- Thread cancellation
  - Terminating a thread before it has finished.
  - There are two general approaches:
    - Asynchronous cancellation terminates the target thread immediately
    - Deferred cancellation allows the target thread to periodically check if it should be cancelled

- Signal handling
  - Signals are used in UNIX systems to notify a process that a particular event has occurred
  - A signal handler is used to process signals
  - Signal is generated by particular event
  - Signal is delivered to a process
  - Signal is handled
  - The signal can be delivered :
    - to the thread to which the signal applies

        - ↑ to every thread in the process
        - ↑ to certain threads in the process
    - – Assign a specific thread to receive all signals for the process.

- ▪ Thread pools
    - – Create a number of threads in a pool where they await work.
    - – This is slightly faster to service a request with an existing thread than create a new thread
    - – This allows the number of threads in the application(s) to be bound to the size of the pool.
- ▪ Thread specific data
    - – This allows each thread to have its own copy of data
    - – This is useful when you do not have control over the thread creation process (i.e., when using a thread pool)
- ▪ Scheduler activations
    - – The Many to Many models require communication to maintain the appropriate number of kernel threads allocated to the application.
    - – The scheduler activations provide upcalls: a communication mechanism from the kernel to the thread library.
    - – This communication allows an application to maintain the correct number kernel threads.

5.2.3 Thread Implementation

The various thread implementations are listed below:

❖ **PThreads**

- This is a POSIX standard (IEEE 1003.1c).
- This is a API for thread creation and synchronization and specifies the behavior of the thread library.
- pthreads is an Object Orientated API that allows user-land multi-threading in PHP.
- It includes all the tools needed to create multi-threaded applications targeted at the Web or the Console.

- PHP applications can create, read, write, execute and synchronize with Threads, Workers and Threaded objects.
- **A Threaded Object:**
  - A Threaded Object forms the basis of the functionality that allows pthreads to operate.
  - It exposes synchronization methods and some useful interfaces for the programmer.
- **Thread:**
  - The user can implement a Thread by extending the Thread declaration provided by pthreads implementing the run method.
  - Any members of the thread can be written and read by any context with a reference to the Thread.
  - Any context can be executed in any public and protected methods.
  - The run method of the implementation is executed in a separate thread when the start method of the implementation is called from the context that created it.
  - Only the context that creates a thread can start and join with it.
- **Worker Object:**
  - A Worker Thread has a persistent state, and will be invoked by start().
  - The life of worker thread is till the calling object goes out of scope, or is explicitly shutdown.
  - Any context with a reference can stack objects onto the Worker, which will be executed by the Worker in a separate Thread.
  - The run method of a Worker is executed before any objects on the stack, such that it can initialize resources that the objects to come may need.
- **Pool:**
  - A Pool of Worker threads can be used to distribute Threaded objects among Workers.
  - The Pool class included implements this functionality and takes care of referencing in a sane manner.
  - The Pool implementation is the easiest and most efficient way of using multiple threads.

- **Synchronization:**
  - All of the objects that pthreads creates have built in synchronization in the (familiar to java programmers) form of ::wait and ::notify.
  - Calling ::wait on an object will cause the context to wait for another context to call ::notify on the same object.
  - This allows for powerful synchronization between Threaded Objects in PHP.
- **Method Modifiers:**
  - The protected methods of Threaded Objects are protected by pthreads, such that only one context may call that method at a time.
  - The private methods of Threaded Objects can only be called from within the Threaded Object during execution.
- **Data Storage:**
  - Any data type that can be serialized can be used as a member of a Threaded object, it can be read and written from any context with a reference to the Threaded Object.
  - Not every type of data is stored serially, basic types are stored in their true form. Complex types, Arrays, and Objects that are not Threaded are stored serially; they can be read and written to the Threaded Object from any context with a reference.
  - With the exception of Threaded Objects any reference used to set a member of a Threaded Object is separated from the reference in the Threaded Object; the same data can be read directly from the Threaded Object at any time by any context with a reference to the Threaded Object.
- **Static Members:**
  - When a new context is created , they are generally copied, but resources and objects with internal state are nullified .
  - This allows them to function as a kind of thread local storage.
  - Allowing the new context to initiate a connection in the same way as the context that created it, storing the connection in the same place without affecting the original context.
  - These threads are common in UNIX operating systems (Solaris, Linux, Mac OS X).

❖ Windows Threads

- Microsoft Windows supports preemptive multitasking, which creates the effect of simultaneous execution of multiple threads from multiple processes.
- On a multiprocessor computer, the system can simultaneously execute as many threads as there are processors on the computer.
- A **job object** allows groups of processes to be managed as a unit.
- Job objects are namable, securable, sharable objects that control attributes of the processes associated with them.
- Operations performed on the job object affect all processes associated with the job object.
- An application can use the **thread pool** to reduce the number of application threads and provide management of the worker threads.
- Applications can queue work items, associate work with waitable handles, automatically queue based on a timer, and bind with I/O.
- **User-mode scheduling (UMS)** is a lightweight mechanism that applications can use to schedule their own threads.
- An application can switch between UMS threads in user mode without involving the system scheduler and regain control of the processor if a UMS thread blocks in the kernel.
- Each UMS thread has its own thread context instead of sharing the thread context of a single thread.
- The ability to switch between threads in user mode makes UMS more efficient than thread pools for short-duration work items that require few system calls.
- A **fiber** is a unit of execution that must be manually scheduled by the application.
- Fibers run in the context of the threads that schedule them.
- Each thread can schedule multiple fibers.
- In general, fibers do not provide advantages over a well-designed multithreaded application.
- However, using fibers can make it easier to port applications that were designed to schedule their own threads.

❖ Java Threads

- Java is a multi threaded programming language which means we can develop multi threaded program using Java.
- A multi threaded program contains two or more parts that can run concurrently and each part can handle different task at the same time making optimal use of the available resources especially when your computer has multiple CPUs.
- Java threads may be created by: Extending Thread class or by implementing the Runnable interface.
- Java threads are managed by the JVM.

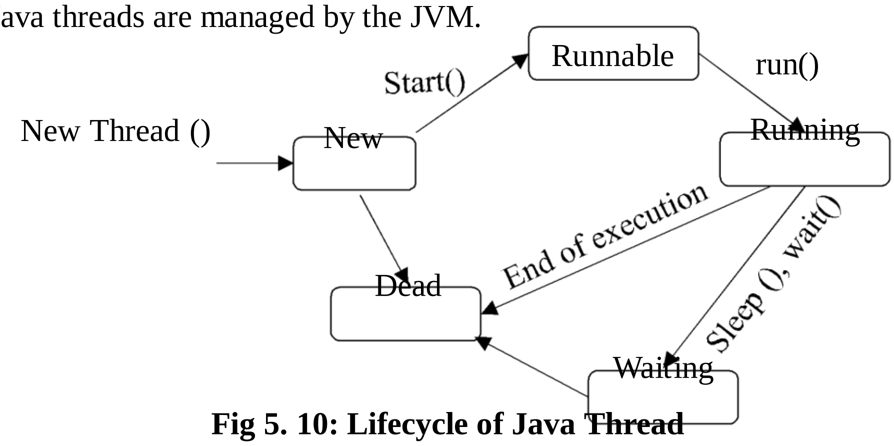


Fig 5. 10: Lifecycle of Java Thread

- Thread class provide constructors and methods to create and perform operations on a thread.
- Thread class extends Object class and implements Runnable interface.
- The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread.
- Runnable interface have only one method named run().

5.3 RESOURCE MANAGEMENT IN DISTRIBUTED SYSTEMS

- ✓ Distributed systems contain a set of resources interconnected by a network.
- ✓ Processes are migrated to fulfill their resource requirements.
- ✓ The Resource manager is responsible to control the assignment of resources to processes.
- ✓ The resources can be logical (shared file) or physical (CPU) resource.
- ✓ Scheduling is the way in which processors are assigned to run on the available resources.

- ✓ In a distributed computing system, the scheduling of various modules on particular processing nodes may be preceded by appropriate allocation of modules of the different tasks to various processing nodes and then only the appropriate execution characteristic can be obtained.
- ✓ The task allocation becomes the important most and major activity in the task scheduling within the operating system of a DCS.

5.3.1 Features of Scheduling algorithms

The following are the desired features of scheduling algorithms:

- ➢ **General purpose**
  - ✓ A scheduling approach should make few assumptions about and have few restrictions to the types of applications that can be executed.
  - ✓ Interactive jobs, distributed and parallel applications, as wellas non-interactive batch jobs, should all be supported with good performance.
  - ✓ This property is a straightforward one, but to some extent difficult to achieve.
  - ✓ Because different kinds of jobs have different attributes, their requirements to the scheduler may contradict.
  - ✓ To achieve the general purpose, a tradeoff may have to be made.
- ➢ **Efficiency:**
  - ✓ It has two meanings: one is that it should improve the performance of scheduled jobs as much as possible; the other is that the scheduling should incur reasonably low overhead so that it would not counter attack the benefits.
- ➢ **Fairness:**
  - ✓ Sharing resources among users raises new challenges in guaranteeing that each user obtains his/her fair share when demand is heavy is fairness.
  - ✓ In a distributed system, this problem could be exacerbated suchthat one user consumes the entire system.
  - ✓ There are many mature strategies to achieve fairness on a single node.
- ➢ **Dynamic:**
  - ✓ The algorithms employed to decide where to process a task should respond to load changes, and exploit the full extent of the resources available.

- **Transparency:**
  - ✓ The behavior and result of a task’s execution should not be affected by the host(s) on which it executes.
  - ✓ In particular, there should be no difference between local and remote execution.
  - ✓ No user effort should be required in deciding where to execute a task or in initiating remote execution; a user should not even be aware of remote processing.
  - ✓ Further, the applications should not be changed greatly.
  - ✓ It is undesirable to have to modify the application programs in order to execute them in the system.
- **Scalability**
  - ✓ A scheduling algorithm should scale well as the number of nodes increases.
  - ✓ An algorithm that makes scheduling decisions by first inquiring the workload from all the nodes and then selecting the most lightly loaded node has poor scalability.
  - ✓ This will work fine only when there are few nodes in the system.
  - ✓ This is because the inquirer receives a flood of replies almost simultaneously, and the time required to process the reply messages for making a node selection is too long as the number of nodes (N) increase.
  - ✓ Also the network traffic quickly consumes network bandwidth.
  - ✓ A simple approach is to probe only m of N nodes for selecting a node.
- **Fault tolerance**
  - ✓ A good scheduling algorithm should not be disabled by the crash of one or more nodes of the system.
  - ✓ Also, if the nodes are partitioned into two or more groups due to link failures, the algorithm should be capable of functioning properly for the nodes within a group.
  - ✓ Algorithms that have decentralized decision making capability and consider only available nodes in their decision making have better fault tolerance capability.

- **Quick decision making capability**
  - ✓ Heuristic methods requiring less computational efforts (and hence less time) while providing near-optimal results are preferable to exhaustive (optimal) solution methods.
- **Balanced system performance and scheduling overhead**
  - ✓ Algorithms that provide near-optimal system performance with a minimum of global state information (such as CPU load) gathering overhead are desirable.
  - ✓ This is because the overhead increases as the amount of global state information collected increases.
  - ✓ This is because the usefulness of that information is decreased due to both the aging of the information being gathered and the low scheduling frequency as a result of the cost of gathering and processing the extra information.
- **Stability**
  - ✓ Fruitless migration of processes, known as processor thrashing, must be prevented.
  - ✓ E.g. if nodes n1 and n2 observe that node n3 is idle and then offload a portion of their work to n3 without being aware of the offloading decision made by the other node.
  - ✓ Now if n3 becomes overloaded due to this it may again start transferring its processes to other nodes.
  - ✓ This is caused by scheduling decisions being made at each node independently of decisions made by other nodes.

5.3.2 Task Assignment Approach

The following assumptions are made in this approach:

- A process has already been split up into pieces called **tasks.**
- This split occurs along natural boundaries (such as a method), so that each task will have integrity in itself and data transfers among the tasks are minimized.
- The amount of computation required by each task and the speed of each CPU are known. The cost of processing each task on every node is known. This is derived from assumption 2.
- The IPC cost between every pair of tasks is already known.
- The IPC cost is 0 for tasks assigned to the same node.

- This is usually estimated by an analysis of the static program.
- If two tasks communicate *n* times and the average time for each inter-task communication is *t*, them IPC costs for the two tasks is *n* * *t*.
- Precedence relationships among the tasks are known.
- Reassignment of tasks is not possible.

Goal of this method is to assign the tasks of a process to the nodes of a distributed system in such a manner as to achieve goals such as:

- ❖ Minimization of IPC costs
- ❖ Quick turnaround time for the complete process
- ❖ A high degree of parallelism
- ❖ Efficient utilization of system resources in general

- These goals often conflict. E.g., while minimizing IPC costs tends to assign all tasks of a process to a single node, efficient utilization of system resources tries to distribute the tasks evenly among the nodes.
- So also, quick turnaround time and a high degree of parallelism encourage parallel execution of the tasks, the precedence relationship among the tasks limits their parallel execution.
- In case of *m* tasks and *q* nodes, there are m^q possible assignments of tasks to nodes .
- In practice, however, the actual number of possible assignments of tasks to nodes may be less than m^q due to the restriction that certain tasks cannot be assigned to certain nodes due to their specific requirements.

5.3.3 Classification of Load Balancing Algorithms

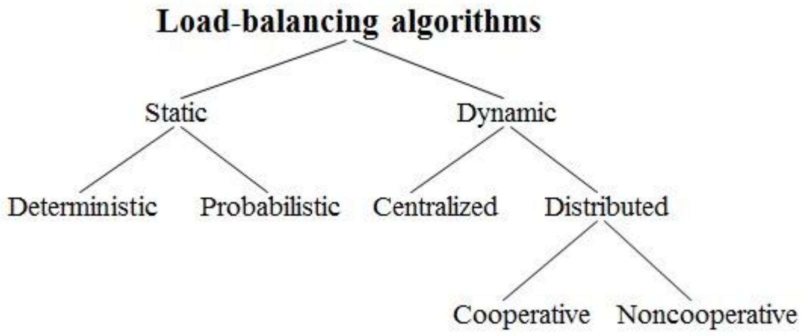


Fig 5.11: Classification of Load Balancing Algorithms

- **Static versus Dynamic**

Static Algorithms	**Dynamic Algorithms**
Static algorithms use only information about the average behavior of the system.	Dynamic algorithms collect state information and react to system state if it changed.
Static algorithms ignore the current state or load of the nodes in the system.	Dynamic algorithms are able to give significantly better performance.
Static algorithms are much simpler.	They are complex

- **Deterministic versus Probabilistic**

Deterministic Algorithms	**Probabilistic Algorithms**
Deterministic algorithms use the information about the properties of the nodes and the characteristic of processes to be scheduled.	Probabilistic algorithms use information of static attributes of the system (e.g. number of nodes, processing capability, topology) to formulate simple process placement rules
Deterministic approach is difficult to optimize.	Probabilistic approach has poor performance

- **Centralized versus Distributed**

Centralized Algorithms	**Distributed Algorithms**
Centralized approach collects information to server node and makes assignment decision.	Distributed approach contains entities to make decisions on a predefined set of nodes
Centralized algorithms can make efficient decisions, have lower fault-tolerance	Distributed algorithms avoid the bottleneck of collecting state information and react faster

- **Cooperative versus Non-cooperative**

Cooperative Algorithms	**Non-cooperative Algorithms**
In Co-operative algorithms distributed entities cooperatewith each other.	In Non-cooperative algorithms entities act as autonomous ones and make scheduling decisions independently from other entities.

Cooperative algorithms are more complex and involve larger overhead.	They are simpler.
Stability of Cooperative algorithms are better.	Stability is comparatively poor.

Issues in Load Balancing Algorithms

- ❖ **Load estimation policy**: determines how to estimate the workload of a node.
- ❖ **Process transfer policy:** determines whether to execute a process locally or remote.
- ❖ **State information exchange policy:** determines how to exchange load information among nodes.
- ❖ **Location policy:** determines to which node the transferable process should be sent.
- ❖ **Priority assignment policy:** determines the priority of execution of local and remote processes.
- ❖ **Migration limiting policy:** determines the total number of times a process can migrate.

Load Estimation Policies:

Policy I:

- ➢ To balance the workload on all the nodes of the system, it is necessary to decide how to measure the workload of a particular node.
- ➢ Some measurable parameters (with time and node dependent factor) can be the following:
  - ∗ Total number of processes on the node
  - ∗ Resource demands of these processes
  - ∗ Instruction mixes of these processes
  - ∗ Architecture and speed of the node’s processor

- ➢ Several load-balancing algorithms use the total number of processes to achieve big efficiency.

Policy II:

- In some cases the true load could vary widely depending on the remaining service time, which can be measured in several way:

  - **Memory less** method assumes that all processes have the same expected remaining service time, independent of the time used so far.
  - **Past repeats** assumes that the remaining service time is equal to the time used so far.
  - **Distribution** method states that if the distribution service times is known, the associated process's remaining service time is the expected remaining time conditioned by the time already used.

Policy III:

- None of the previous methods can be used in modern systems because of periodically running processes and daemons.
- An acceptable method for use as the load estimation policy in these systems would be to measure the CPU utilization of the nodes.
- Central Processing Unit utilization is defined as the number of CPU cycles actually executed per unit of real time.
- It can be measured by setting up a timer to periodically check the CPU state (idle/busy).

Threshold Policy

- Most of the algorithms use the **threshold policy** to decide on whether the node is lightly-loaded or heavily-loaded.
- Threshold value is a limiting value of the workload of node which can be determined by:

  - Static policy: predefined threshold value for each node depending on processing capability.
  - Dynamic policy: threshold value is calculated from average workload and a predefined constant

- Below threshold value node accepts processes to execute, above threshold value node tries to transfer processes to a lightly-loaded node.

Single threshold Policy

- Single-threshold policy may lead to unstable algorithm because under loaded node could turn to be overloaded right after a process migration.

- To re ed which is also known

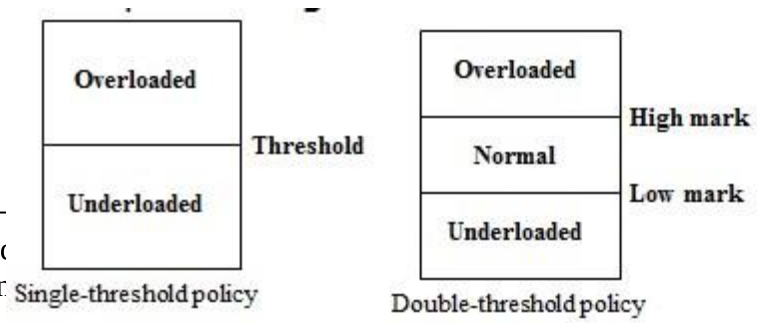


Fig 5.12: Process Threshold Policy

Double threshold Policy

- When node is in overloaded region new local processes are sent to run remotely, requests to accept remote processes are rejected.
- When node is in normal region new local processes run locally, requests to accept remote processes are rejected.
- When node is in under loaded region new local processes run locally, requests to accept remote processes are accepted

Location Policies:

- **Threshold method:**

  This policy selects a random node, checks whether the node is able to receive the process, then transfers the process. If node rejects, another node is selected randomly. This continues until probe limit is reached.

- **Shortest method**
  - L distinct nodes are chosen at random, each is polled to determine its load. The process is transferred to the node having the minimum value unless its workload value prohibits to accept the process.
  - Simple improvement is to discontinue probing whenever a node with zero load is encountered.
- **Bidding method**
  - The nodes contain managers (to send processes) and contractors (to receive processes).

- The managers broadcast a request for bid, contractors respond with bids (prices based on capacity of the contractor node) and manager selects the best offer.
- Winning contractor is notified and asked whether it accepts the process for execution or not.
- The full autonomy for the nodes regarding scheduling.
- This causes big communication overhead.
- It is difficult to decide a good pricing policy.

- **Pairing**
  - Contrary to the former methods the pairing policy is to reduce the variance of load only between pairs.
  - Each node asks some randomly chosen node to form a pair with it.
  - If it receives a rejection it randomly selects another node and tries to pair again.
  - Two nodes that differ greatly in load are temporarily paired with each other and migration starts.
  - The pair is broken as soon as the migration is over.
  - A node only tries to find a partner if it has at least two processes.

Priority assignment policy

- **Selfish:** Local processes are given higher priority than remote processes. Worst response time performance of the three policies.
- **Altruistic:** Remote processes are given higher priority than local processes. Best response time performance of the three policies.
- **Intermediate:** When the number of local processes is greater or equal to the number of remote processes, local processes are given higher priority than remote processes. Otherwise, remote processes are given higher priority than local processes.

Migration limiting policy

This policy determines the total number of times a process can migrate

- **Uncontrolled:** A remote process arriving at a node is treated just as a process originating at a node, so a process may be migrated any number of times
- **Controlled:**
  - Avoids the instability of the uncontrolled policy.

- ∗ Use a migration count parameter to fix a limit on the number of time a process can migrate.
- ∗ Irrevocable migration policy: migration count is fixed to 1.
- ∗ For long execution processes migration count must be greater than 1 to adapt for dynamically changing states

5.3.4 Load-sharing approach

- ▪ The following problems in load balancing approach led to load sharing approach:
  - – Load balancing technique with attempting equalizing the workload on all the nodes is not an appropriate object since big overhead is generated by gathering exact state information.
  - – Load balancing is not achievable since number of processes in a node is always fluctuating and temporal unbalance among the nodes exists every moment

Basic idea:

- ❖ It is necessary and sufficient to prevent nodes from being idle while some other nodes have more than two processes.
- ❖ Load-sharing is much simpler than load-balancing since it only attempts to ensure that no node is idle when heavily node exists.
- ❖ Priority assignment policy and migration limiting policy are the same as that for the load-balancing algorithms.

Load Estimation Policies

- Since load-sharing algorithms simply attempt to avoid idle nodes, it is sufficient to know whether a node is busy or idle.
- Thus these algorithms normally employ the simplest load estimation policy of counting the total number of processes.
- In modern systems where permanent existence of several processes on an idle node is possible, algorithms measure CPU utilization to estimate the load of a node

Process Transfer Policies

- ✓ The load sharing algorithms normally use all-or-nothing strategy.
- ✓ This strategy uses the threshold value of all the nodes fixed to 1.
- ✓ Nodes become receiver node when it has no process, and become sender node when it has more than 1 process.

- ✓ To avoid processing power on nodes having zero process load-sharing algorithms uses a threshold value of 2 instead of 1.

- ✓ When CPU utilization is used as the load estimation policy, the double-threshold policy should be used as the process transfer policy

Location Policies

- ❖ The location policy decides whether the sender node or the receiver node of the process takes the initiative to search for suitable node in the system, and this policy can one of the following:

- ❖ **Sender-initiated location policy:** Sender node decides where to send the process. Heavily loaded nodes search for lightly loaded nodes

- ❖ **Receiver-initiated location policy**: Receiver node decides from where to get the process. Lightly loaded nodes search for heavily loaded nodes

Sender-initiated location policy

- ➢ Node becomes overloaded, it either broadcasts or randomly probes the other nodes one by one to find a node that is able to receive remote processes.

- ➢ When broadcasting, suitable node is known as soon as reply arrives

Receiver-initiated location policy

- ➢ Nodes becomes underloaded, it either broadcast or randomly probes the other nodes one by one to indicate its willingness to receive remote processes.

- ➢ Receiver-initiated policy require preemptive process migration facility since scheduling decisions are usually made at process departure epochs

  - ✓ Both policies gives substantial performance advantages over the situation in which no load-sharing is attempted.

  - ✓ Sender-initiated policy is preferable at light to moderate system loads.

  - ✓ Receiver-initiated policy is preferable at high system loads.

  - ✓ Sender-initiated policy provide better performance for the case when process transfer cost significantly more at receiver-initiated than at sender-initiated policy due to the pre-emptive transfer of processes.

State information exchange policies

- ∗ In load-sharing algorithms it is not necessary for the nodes to periodically exchange state information, but needs to know the state of other nodes when it is either underloaded or overloaded.

- The following are the two approaches followed when there is state change:
- **Broadcast when state changes**
  - In sender-initiated/receiver-initiated location policy a node broadcasts State Information Request when it becomes overloaded/ underloaded.
  - It is called broadcast-when-idle policy when receiver-initiated policy is used with fixed threshold value of 1
- **Poll when state changes**
  - In large networks polling mechanism is used.
  - Polling mechanism randomly asks different nodes for state information until find an appropriate one or probe limit is reached.
  - It is called poll-when-idle policy when receiver-initiated policy is used with fixed threshold value value of 1.

REVIEW QUESTIONS

PART - A

1. **Define process management.**

   Process Management involves the activities aimed at defining a process, establishing responsibilities, evaluating process performance, identifying and opportunities or improvement.

2. **Define process migration.**

   Process migration is the act of transferring process between two machines.

3. **What is Eager (All)?**

   ∗ This transfers the entire address space.

   ∗ No trace of process is left behind in the original system.

   ∗ If address space is large and if the process does not need most of it, then this approach is unnecessarily expensive.

   ∗ Implementations that provide a checkpoint/restart facility are likely to use this approach, because it is simpler to do the check-pointing and restarting if all of the address space is localized.

4. **What is Precopy?**

   ∗ In this method, the process continues to execute on the source node while the address space is copied to the target node.

   ∗ The pages modified on the source during the precopy operation have to be copied a second time.

   ∗ This strategy reduces the time that a process is frozen and cannot execute during migration.

5. **What is Eager (dirty)?**

   ∗ The transfer involves only the portion of the address space that is in main memory and has been modified.

   ∗ Any additional blocks of the virtual address space are transferred on demand.

   ∗ The source machine is involved throughout the life of the process.

6. **What is Copy-on-reference?**

   ∗ In this method, the pages are only brought over when referenced.

   ∗ This has the lowest initial cost of process migration.

7. What is Flushing?

- The pages are cleared from main memory by flushing dirty pages to disk.
- This relieves the source of holding any pages of the migrated process in main memory.

8. Define thread.

A thread is the entity within a process that can be scheduled for execution. All threads of a process share its virtual address space and system resources.

9. What is thread context?

The *thread context* includes the thread's set of machine registers, the kernel stack, a thread environment block, and a user stack in the address space of the thread's process.

10. What is thread pool?

Thread pool consists of a number of threads in a pool where they await work. This is slightly faster to service a request with an existing thread than create a new thread. This allows the number of threads in the application(s) to be bound to the size of the pool.

11. What is worker thread?

A Worker Thread has a persistent state, and will be invoked by start(). The life of worker thread is till the calling object goes out of scope, or is explicitly shutdown

12. What is job object?

A job object allows groups of processes to be managed as a unit.

13. What is UMS?

User-mode scheduling (UMS) is a lightweight mechanism that applications can use to schedule their own threads.

14. What is a fiber?

A fiber is a unit of execution that must be manually scheduled by the application.

15. What is multithreaded program?

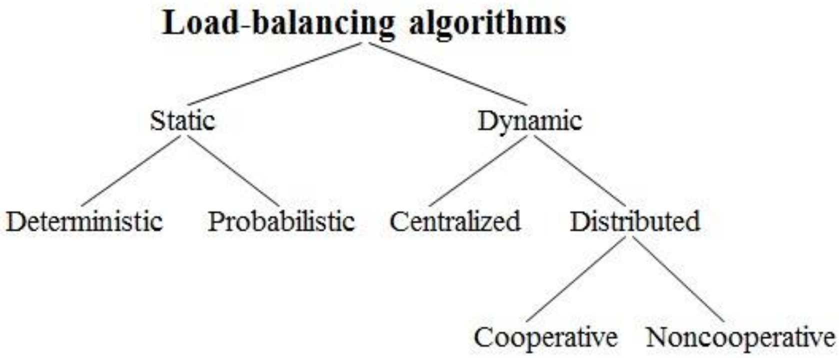
A multi- threaded program contains two or more parts that can run concurrently and each part can handle different task at the same time making optimal use of the available resources especially when your computer has multiple CPUs.

16. List the features of scheduling algorithm.

- General purpose
- Efficiency

- Fairness
- Dynamic
- Transparency
- Scalability
- Fault tolerance
- Quick decision making capability
- Balanced system performance and scheduling overhead
- Stability

17. Classify load balancing algorithms.



18. Differentiate static and dynamic algorithms.

Static Algorithms	**Dynamic Algorithms**
Static algorithms use only information about the average behavior of the system.	Dynamic algorithms collect state information and react to system state if it changed.
Static algorithms ignore the current state or load of the nodes in the system.	Dynamic algorithms are able to give significantly better performance.
Static algorithms are much simpler.	They are complex

19. Differentiate deterministic and probabilistic algorithms.

Deterministic Algorithms	**Probabilistic Algorithms**
Deterministic algorithms use the information about the properties of the nodes and the characteristic of processes to be scheduled.	Probabilistic algorithms use information of static attributes of the system (e.g. number of nodes, processing capability, topology) to formulate simple process placement rules
Deterministic approach is difficult to optimize.	Probabilistic approach has poor performance

20. Differentiate centralized and distributed algorithms.

Centralized Algorithms	**Distributed Algorithms**
Centralized approach collects information to server node and makes assignment decision.	Distributed approach contains entities to make decisions on a predefined set of nodes
Centralized algorithms can make efficient decisions, have lower fault-tolerance	Distributed algorithms avoid the bottleneck of collecting state information and react faster

21. Differentiate cooperative and non co operative algorithms.

Cooperative Algorithms	**Non-cooperative Algorithms**
In Co-operative algorithms distributed entities cooperatewith each other.	In Non-cooperative algorithms entities act as autonomous ones and make scheduling decisions independently from other entities.
Cooperative algorithms are more complex and involve larger overhead.	They are simpler.
Stability of Cooperative algorithms are better.	Stability is comparatively poor.

22. Give the issues in Load Balancing Algorithms

- ❖ Load estimation policy: determines how to estimate the workload of a node.
- ❖ Process transfer policy: determines whether to execute a process locally or remote.
- ❖ State information exchange policy: determines how to exchange load information among nodes.
- ❖ Location policy: determines to which node the transferable process should be sent.
- ❖ Priority assignment policy: determines the priority of execution of local and remote processes.
- ❖ Migration limiting policy: determines the total number of times a process can migrate.

23. What is threshold policy?

Most of the algorithms use the threshold policy to decide on whether the node is lightly-loaded or heavily-loaded. Threshold value is a limiting value of the workload of node which can be determined by:

- ∗ Static policy: predefined threshold value for each node depending on processing capability.
- ∗ Dynamic policy: threshold value is calculated from average workload and a predefined constant

24. What is single threshold policy?

Single-threshold policy may lead to unstable algorithm because under loaded node could turn to be overloaded right after a process migration.

25. What is double threshold policy?

When node is in overloaded region new local processes are sent to run remotely, requests to accept remote processes are rejected. When node is in normal region new local processes run locally, requests to accept remote processes are rejected.When node is in under loaded region new local processes run locally, requests to accept remote processes are accepted

26. What are the location policies?

- ➢ Threshold method
- ➢ Shortest method
- ➢ Bidding method
- ➢ Pairing
- ➢ Threshold method:

27. What is threshold policy?

This policy selects a random node, checks whether the node is able to receive the process, then transfers the process. If node rejects, another node is selected randomly. This continues until probe limit is reached.

28. What is Shortest method?

* L distinct nodes are chosen at random, each is polled to determine its load. The process is transferred to the node having the minimum value unless its workload value prohibits to accept the process.
* Simple improvement is to discontinue probing whenever a node with zero load is encountered.

29. What is Bidding method?

* The nodes contain managers (to send processes) and contractors (to receive processes).
* The managers broadcast a request for bid, contractors respond with bids (prices based on capacity of the contractor node) and manager selects the best offer.
* Winning contractor is notified and asked whether it accepts the process for execution or not.
* The full autonomy for the nodes regarding scheduling.
* This causes big communication overhead.
* It is difficult to decide a good pricing policy.

30. What is Pairing?

* Contrary to the former methods the pairing policy is to reduce the variance of load only between pairs.
* Each node asks some randomly chosen node to form a pair with it.
* If it receives a rejection it randomly selects another node and tries to pair again.
* Two nodes that differ greatly in load are temporarily paired with each other and migration starts.
* The pair is broken as soon as the migration is over.
* A node only tries to find a partner if it has at least two processes.

31. What is Priority assignment policy?

- ❖ Selfish: Local processes are given higher priority than remote processes. Worst response time performance of the three policies.
- ❖ Altruistic: Remote processes are given higher priority than local processes. Best response time performance of the three policies.
- ❖ Intermediate: When the number of local processes is greater or equal to the number of remote processes, local processes are given higher priority than remote processes. Otherwise, remote processes are given higher priority than local processes.

32. What are the issues in load balancing approach?

The following problems in load balancing approach led to load sharing approach:

- – Load balancing technique with attempting equalizing the workload on all the nodes is not an appropriate object since big overhead is generated by gathering exact state information.
- – Load balancing is not achievable since number of processes in a node is always fluctuating and temporal unbalance among the nodes exists every moment

33. What is load sharing approach?

It is necessary and sufficient to prevent nodes from being idle while some other nodes have more than two processes. Load-sharing is much simpler than load-balancing since it only attempts to ensure that no node is idle when heavily node exists. Priority assignment policy and migration limiting policy are the same as that for the load-balancing algorithms.

34. What are location policies?

- ❖ The location policy decides whether the sender node or the receiver node of the process takes the initiative to search for suitable node in the system, and this policy can one of the following:
- ❖ Sender-initiated location policy: Sender node decides where to send the process. Heavily loaded nodes search for lightly loaded nodes
- ❖ Receiver-initiated location policy: Receiver node decides from where to get the process. Lightly loaded nodes search for heavily loaded nodes

PART-B

1. Explain in detail about process migration.
2. Define threads. Explain all the multithreading thread models.
3. What are the issues in threads?
4. Discuss the various thread implementations.
5. Describe the resource management in distributed systems.
6. Write about task assignment.
7. Classify load balancing algorithms.
8. Explain load sharing approaches.