Process Management

Introduction

- In conventional (centralized) operating system, process management deals with mechanisms and policies for sharing the processor of the system among all processes.
- In a distributed operating system, the main goal of process management is to make the best possible use of the processing resources of the entire system by sharing them among all processes.
- **Three important concepts are used to achieve this goal:**

  -Processor allocation

  -Process migration

  -Threads

Cont…

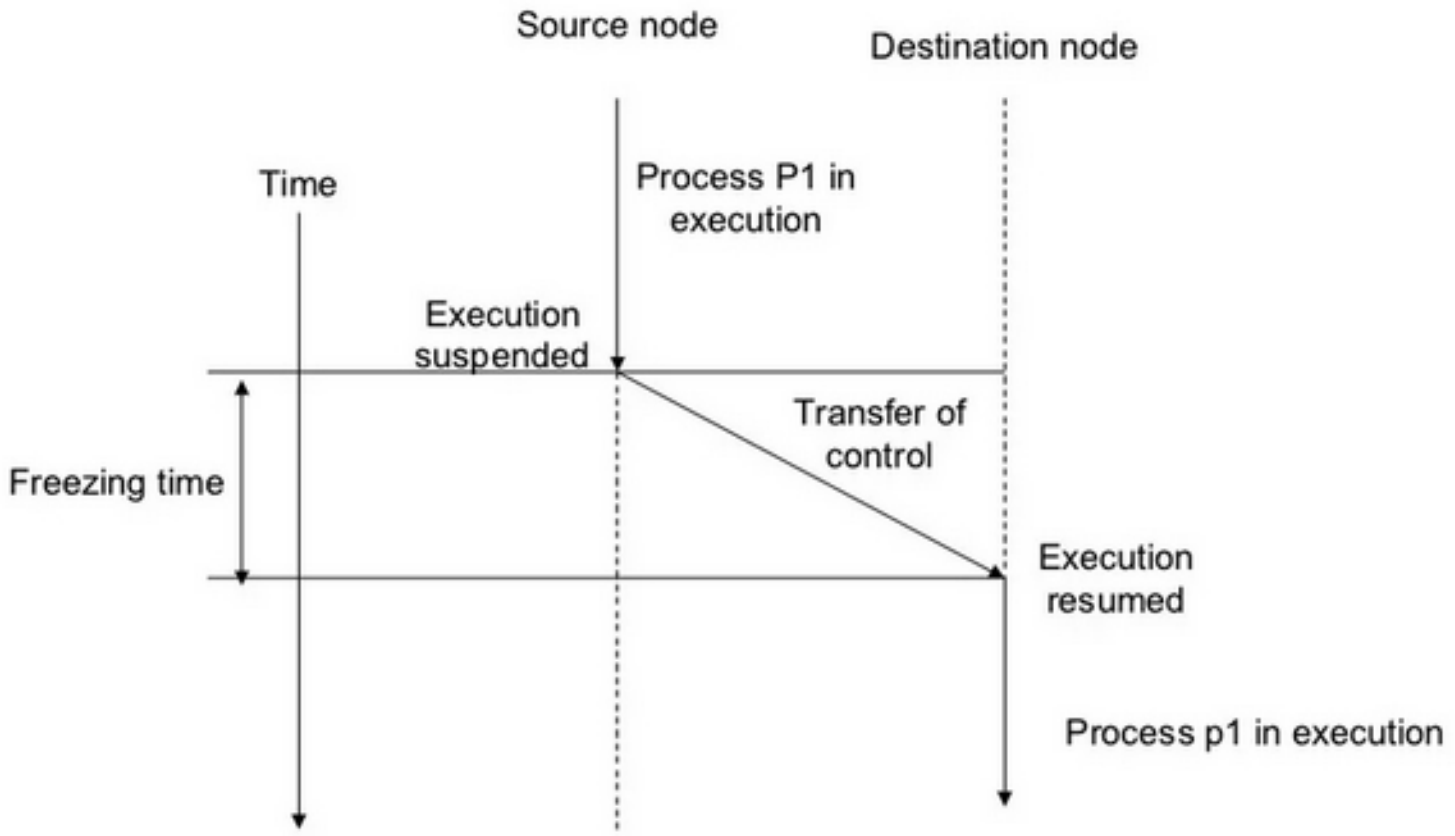
- **Process allocation** deals with the process of deciding which process should be assigned to which processor.
- **Process migration** deals with the movement of a process from its current location to the processor to which it has been assigned.
- **Threads** deals with fine-grained parallelism for better utilization of the processing capability of the system.

Process Migration

- Relocation of a process from its current location to another node.

- Process may be migrated
  - either before it start executing on its source node
    - known as **non-preemptive process migration.**
  - or during the course of its execution
    - Known as preemptive process migration.

Cont… : Flow of execution of a migration process



Cont…

- Preemptive process migration is costlier.

- **Process Migration Policy**
  - Selection of a process that should be migrated.
  - Selection of the destination node to which the selected process should be migrated.

- **Process Migration Mechanism**
  - Actual transfer of the selected process to the destination node.

Desirable features of a good process migration mechanism

- Transparency
- Minimal Interference
- Minimal Residual Dependencies
- Efficiency
- Robustness
- Communication between coprocesses of a job

Transparency

Level of transparency:

- Object Access Level
- System Call & Interprocess Communication level

Cont… : Object Access level Transparency

- Minimum requirement for a system to support non-preemptive process migration facility.
- Access to objects such as files and devices can be location independent.
- Allows free initiation of program at arbitrary node.
- Requires transparent object naming and locating.

Cont… : System Call & IPC level

- For migrated process, system calls should be location independent.
- For transparent redirection of messages during the transient state of a process.
- Transparency must be provided to support preemptive process migration facility.

Minimal Interference

- Migration of a process should cause minimal interference of progress of the process involved.

- Achieve by minimizing the freezing time of the process being migrated.
  - Freezing time is the time period for which the execution of the process is stopped for transferring its information to the destination node.

Minimal Residual Dependencies

- No residual dependency should be left on previous node.

Efficiency

- Minimum time required for migrating a process.
- Minimum cost of locating an object.
- Minimum cost of supporting remote execution once the process is migrated.

Robustness

- The failure of a node other than the one on which a process is currently running should not in any way affect the accessibility or execution of that process.

Communication between coprocessors of a job

- To reduce communication cost, it is necessary that coprocesses are able to directly communicate with each other irrespective of their locations.

Process Migration Mechanisms

Four major activities

- Freezing the process on its source node and restarting it on its destination node.
- Transferring the process's address space from its source node to its destination node.
- Forwarding massages meant for the migrant process.
- Handling communication between cooperating processes that have been separated as a result of process migration.

Cont… : Mechanisms for Freezing and Restarting a Process

- Freezing the process:
  - The execution of the process is suspended and all external interactions with the process are deferred.

- Issues:
  - Immediate and delayed blocking of the process
  - Fast and slow I/O operations
  - Information about open files
  - Reinstating the process on its Destination node

Cont… :Immediate and delayed blocking of the process

- If the process is not executing a system call, it can be immediately blocked from further execution.
- If the process is executing a system call but is sleeping at an interruptible priority waiting for a kernel event to occur, it can be immediately blocked from further execution.
- If the process is executing a system call but is sleeping at an non-interruptible priority waiting for a kernel event to occur, it cannot be blocked immediately.

Cont… : Fast and Slow I/O Operations

- Process is frozen after the completion of all fast I/O operations like disk access.

- Slow I/O operation (pipe or terminal) is done after process migration and when process is executed on destination node.

Cont… : Information about open files

- Includes name and identifier of the file, their access modes and the current positions of their file pointers.
- OS returns a file descriptor to process that is used for all I/O.
- It is necessary to somehow preserve a pointer to the file so that migrated process could continue to access it.

Cont…

- Approaches :
  - Link is created to the file and the pathname of the link is used as an access point to the file after the process migrates.
  - An open file's complete pathname is reconstructed when required by modifying the kernel.
- Keeping Track of file replicas.

Cont… : Reinstating the process on its Destination Node

- On the destination node, an empty process state is created.
- Newly allocated process may or may not have the same process identifier as the migrating process.

Cont…

- Once all the state of the migrating process has been transferred from the source to destination node and copied into the empty state, new copy of the process is unfrozen and old copy is deleted.

- The process is restarted on its destination node in whatever state it was in before being migrated.

Address Space Transfer Mechanisms

- The migration of a process involves the transfer of
  - Process's state
  - Process's address space

  from the source node to the destination node.

Cont…

- **Process State consists of**
  - Execution Status – Register Contents
  - Memory Tables
  - I/O State : I/O Queue, I/O buffers, Interrupts
  - Capability list
  - Process's Identifier
  - Process's user and group identifier
  - Information about Open Files

Cont…

- **Process address space**
  - code,
  - data and
  - program stack

- The size of the process's address space (several megabytes) overshadows the size of the process's state information (few kilobytes).

Cont…

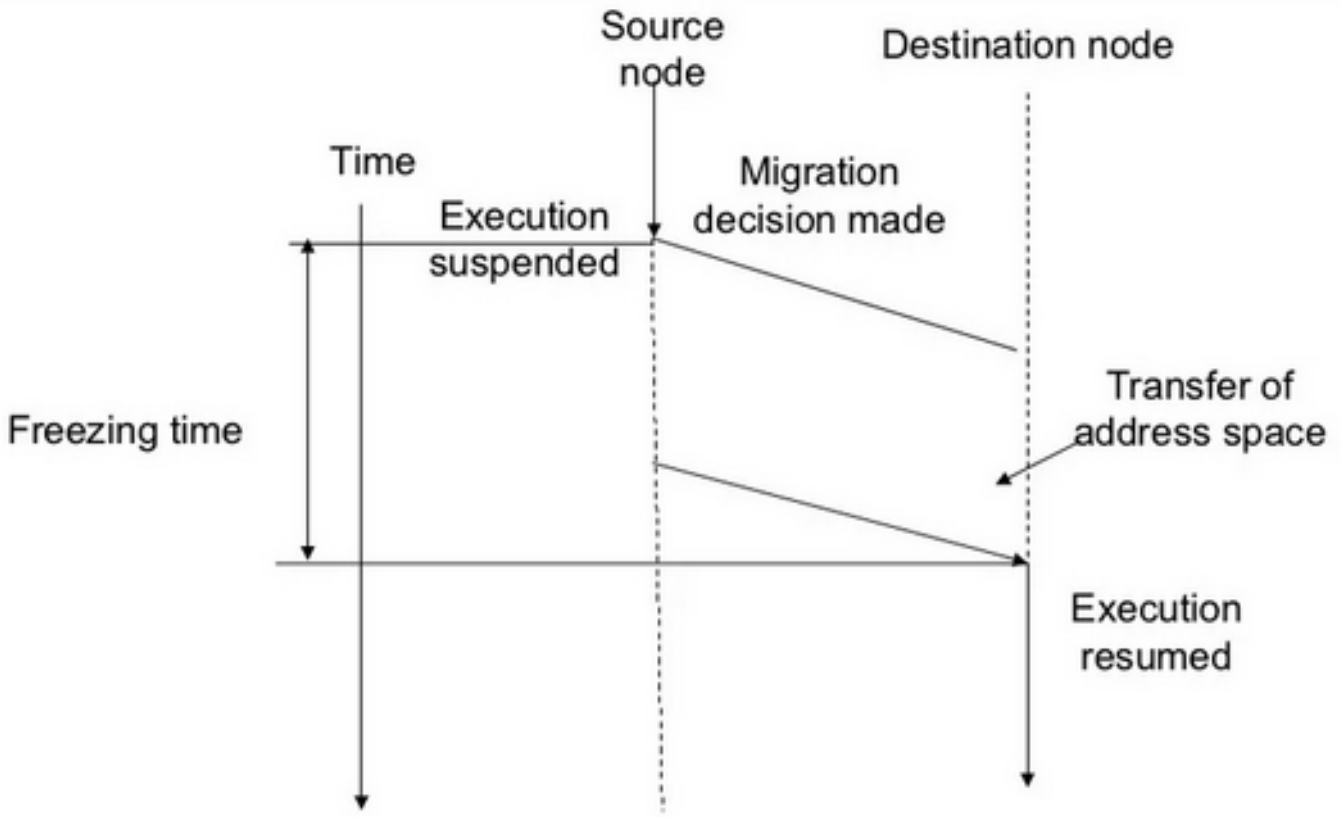
- Mechanisms for address space transfer:
  - Total freezing
  - Pretransferring
  - Transfer on reference

Cont… : Total Freezing

- A process's execution is stopped while its address space is being transferred.

- Disadvantage:
  - Process is suspended for long time during migration, timeouts may occur, and if process is interactive, the delay will be noticed by the user.

Cont…


Source node
Destination node
Time
Migration decision made
Execution suspended
Freezing time
Transfer of address space
Execution resumed

Cont … : Pretransferring

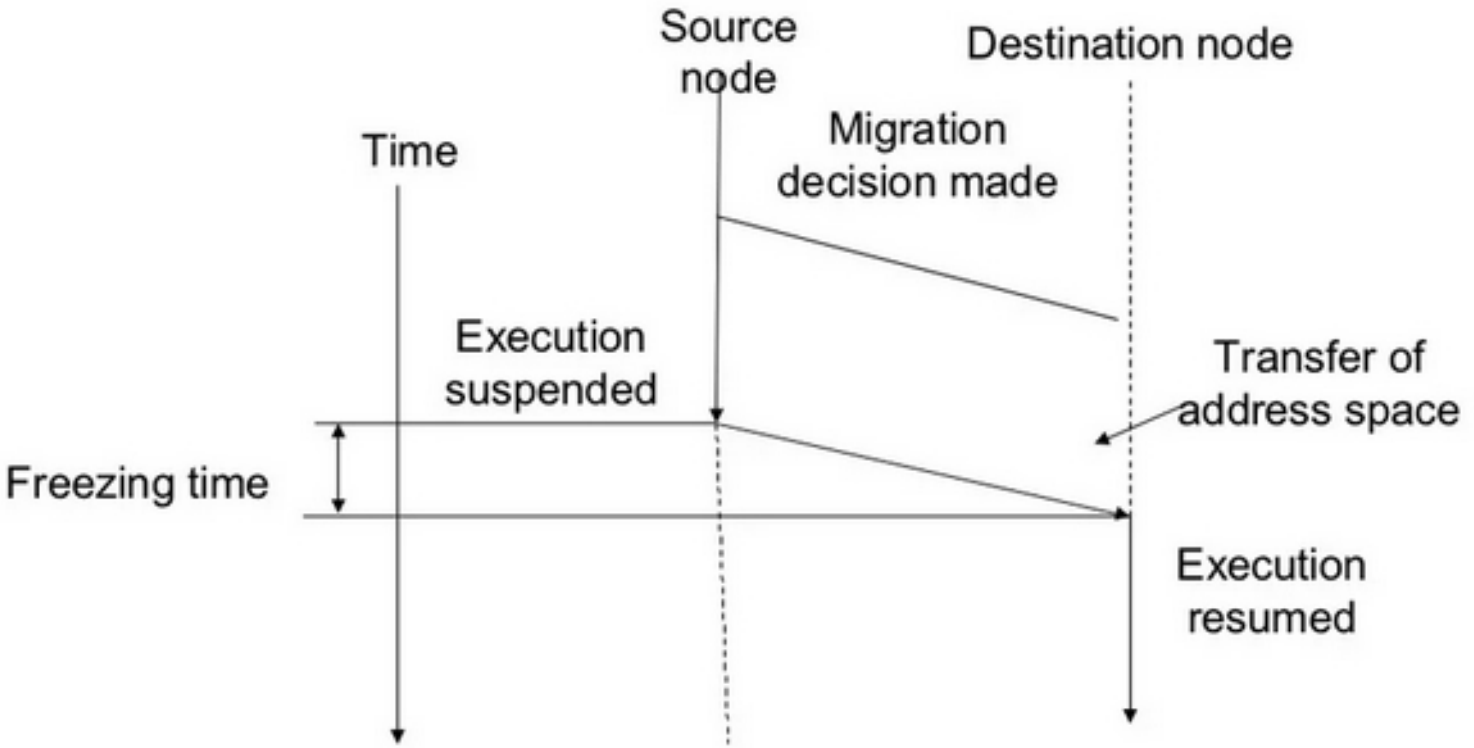
- Also known as precopying.
- The address space is transferred while the process is still running on the source node.
- It is done as an initial transfer of the complete address space followed by repeated transfers of the page modified during the previous transfer.

Cont …

- The pretransfer operation is executed at a higher priority than all other programs on the source node.

- Reduces the freezing time of the process but it may increase the total time for migrating due to the possibility of redundant page transfers.

Cont…


Source node
Destination node
Time
Migration decision made
Execution suspended
Transfer of address space
Freezing time
Execution resumed

Cont… : Transfer on Reference

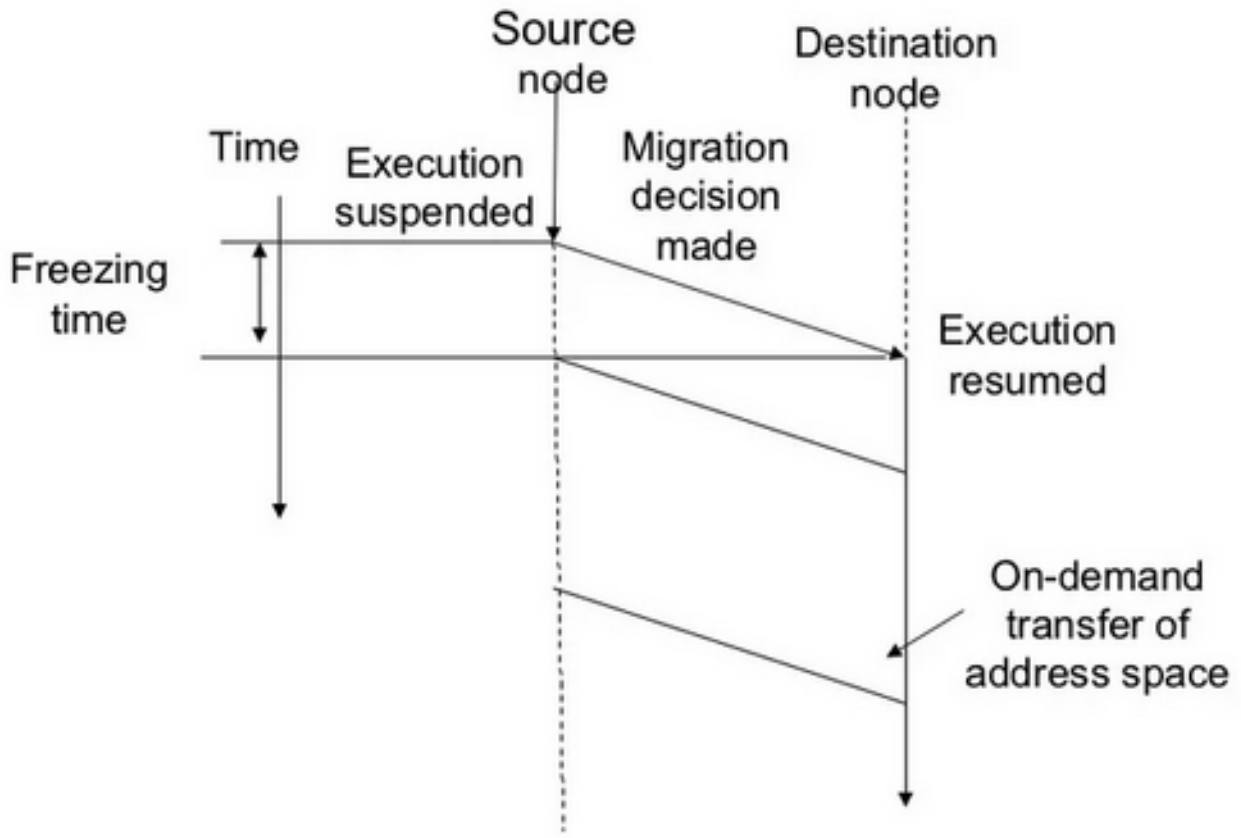
- The process address space is left behind on its source node, and as the relocated process executes on its destination node.
- Attempts to reference memory page results in the generation of requests to copy in the desired blocks from their remote location.
- A page is transferred from its source node to its destination node only when referenced.

Cont…

- Very short switching time of the process from its source node to its destination node.

- Imposes a continued load on the process's source node and results in the process if source node fails or is rebooted.

Cont…


Source node
Destination node
Time
Execution suspended
Migration decision made
Freezing time
Execution resumed
On-demand transfer of address space

Message-forwarding Mechanisms

- In moving a process, it must be ensured that all pending, re-route, and future messages arrive at the process's new location.

- Types of messages:
  1. Messages received at the source node after the process's execution has been stopped on its source node and the process's execution has not yet been started on its destination node.
  2. Messages received at the source node after the process's execution has started on its destination node.
  3. Messages that are to be sent to the migrant process from any other node after it has started executing on the destination node.

Cont…

- Mechanisms:
  - Mechanism of resending the message
  - Origin site mechanism
  - Link traversal mechanism
  - Link update mechanism

Cont… : Mechanisms of resending the message

- Messages of the types 1 and 2 are returned to the sender as not deliverable or are simply dropped, with the assurance that the sender of the message is storing a copy of the data and is prepared to retransmit it.

- Does not require any process state to be left behind on the process's source node.

Cont…

- Disadvantage:
  - The message forwarding mechanism of process migration operation is nontransparent to the processes interacting with the migrant process.

Cont… : Origin Site Mechanism

- Each site is responsible for keeping information about the current locations of all the processes created on it.
- Messages for a particular process are always first sent to its origin site.
- The origin site then forwards the message to the process's current location.

Cont…

- Disadvantage:
  - Failure of the origin site will disrupt the message forwarding mechanism.
  - Continuous load on the migrant process's origin site even after the process has migrated from that node.

Cont… : Link Traversal Mechanism

- To redirect message type 1, a message queue for the migrant process is created on its source node.
- For 2 and 3 type message, a forwarding address known as link is left at the source node pointing to the destination of the migrant process.
- Two component of link, one is unique process identifier and second is last known location.

Cont…

- Disadvantage:
  - Several link may have to be traversed to locate a process from a node and if any node in chain of link fails, the process cannot be located.

Cont… : Link Update Mechanisms

- During the transfer phase of the migrant process, the source node sends link-update messages to the kernels controlling all of the migrant process's communication partners.

- Link update message
  - Tells the new address of each link held by the migrant process.
  - Acknowledged for synchronization purposes.

Mechanisms for Handling Coprocesses

- To provide efficient communication between a process and its sub processes which might have been migrated on different nodes.

- Mechanisms :
  - Disallowing separation of coprocesses.
  - Home node or origin site concept.

Cont… : Disallowing Separation of coprocesses

- Easiest method of handling communication between coprocesses is to disallow their separation.

- Methods :
  - By disallowing the migration of processes that wait for one or more of their children to complete.
  - By ensuring that when a parent process migrates, its children process will be migrated along with it.

Cont…

- Disadvantage:
  - It does not allow the use of parallelism within jobs.

Cont… : Home node or Origin Sites Concept

- Used for communication between a process and its sub process when the two are running on different nodes.

- Allows the complete freedom of migrating a process or its sub process independently and executing them on different nodes of the system.

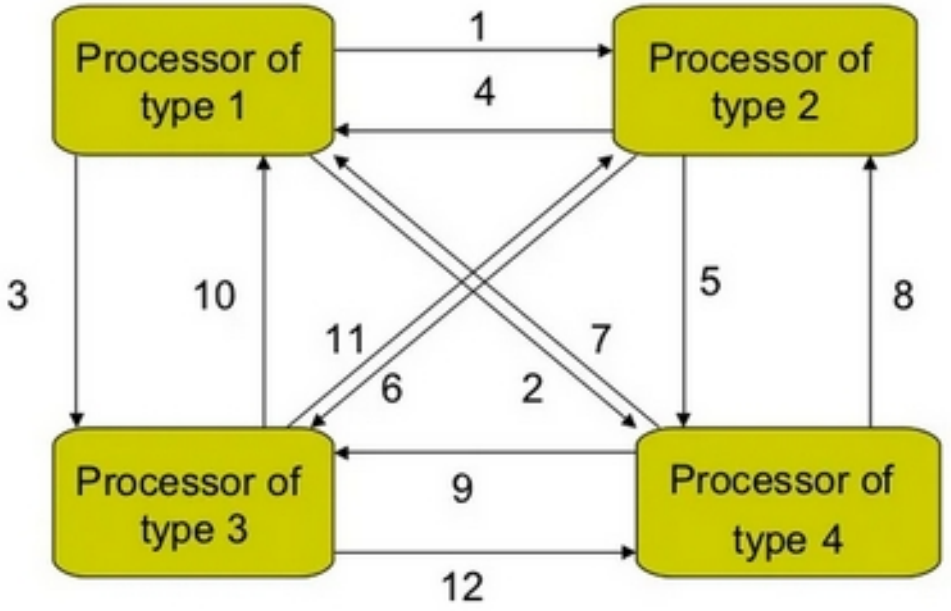
Cont…

- Disadvantage:
  - All communication between a parent process and its children processes take place via the home node.
    - The message traffic and the communication cost increase considerably.

Process Migration in Heterogeneous Systems

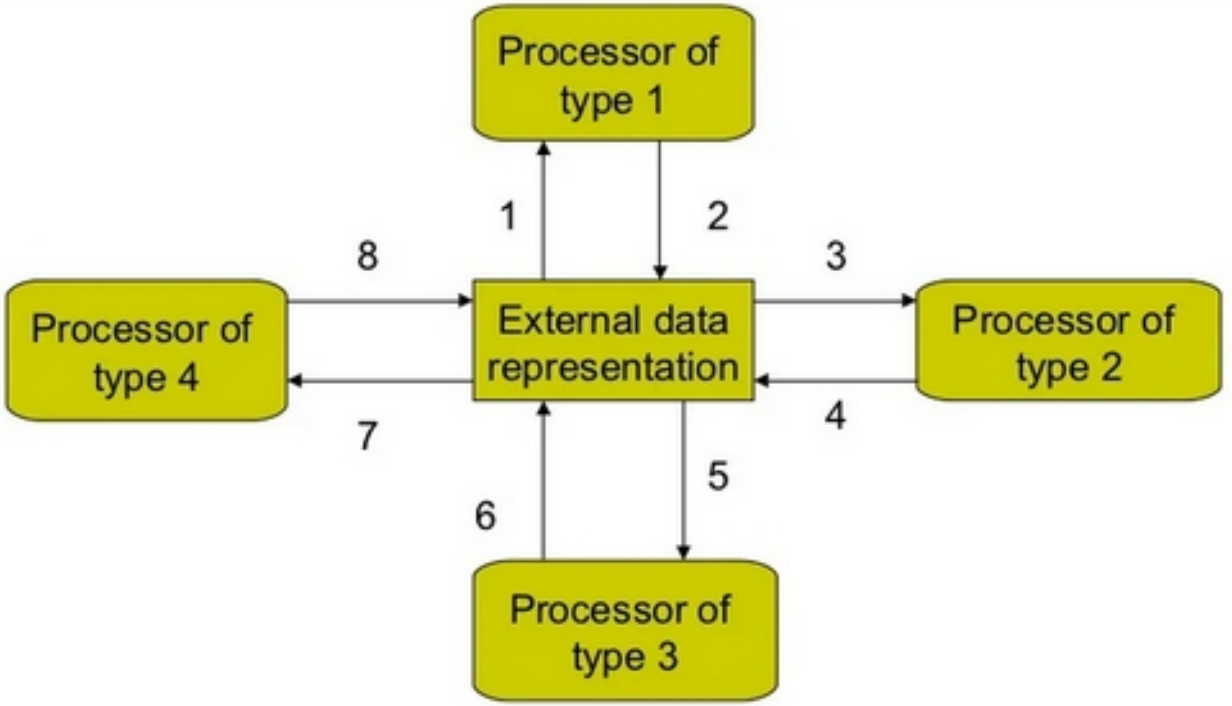
- All the concerned data must be translated from the source CPU format to the destination CPU format before it can be executed on the destination node.
- A heterogeneous system having n CPU types must have n(n-1) pieces of translation software.
- Handles problem of different data representations such as characters, integers and floating-point numbers.

Cont…



Example: The need for 12 pieces of translation software required in a heterogeneous system having 4 types of processors

Cont…



Example: The need for only 8 pieces of translation software in a heterogeneous system having 4 types of processors when the EDR mechanism is used

Cont… : Handling Exponent

- The number of bits for the exponent varies from processor to processor.
- The EDR have at least as many bits in the exponent as the largest exponent.
- Overflow or underflow can be managed upon conversion from the external data representation.

Cont…

- Some solutions:
  - Ensuring that numbers used by the programs that migrate have a smaller exponent value than the smallest processor's exponent value in the system.
  - Emulating the larger processor's value.
  - Restricting the migration of the process to only those nodes whose processor's exponent representation is at least as large as that of the source node's processor.

Cont… : Handling the Mantissa

- Suppose processor A uses 32 bits and processor B uses 64 bits.
  - No problem from A to B migration.
  - Half-Precision problem from B to A migration.

- Solution:
  - The external data representation must have sufficient precision to handle the largest mantissa.

Cont…

- Problem:
  - Loss of precision due to multiple migrations.
- Solution:
  - Remote computations can be viewed as ‘extra precision’ calculation.

Advantages of Process Migration

- *Reducing average response time of processes*
  - To reduce the average response time of the processes, processes of a heavily loaded node are migrated to idle or underutilized nodes.

- *Speeding up individual jobs*
  - A migration of job to different node is done and execute them concurrently.
  - Migrate a job to a node having a faster CPU or to a node at which it has minimum turnaround time.
  - More speed up more migration cost involved.

Cont…

- *Gaining higher throughput*
  - Process migration facility may also be used properly to mix I/O and CPU-bound processes on a global basis for increasing the throughput of the system.

- *Utilizing resources effectively*
  - Depending upon the nature of a process, it can be migrated to suitable node to utilize the system resource in the most efficient manner.

Cont…

- *Reducing network traffic*
  - Migrating a process closer to the resources it is using most heavily.
- *Improving system reliability*
  - Migrate a copy of a critical process to some other node and to execute both the original and copied processes concurrently on different nodes.
- *Improving system security*
  - A sensitive process may be migrated and run on a secure node.

Process

- A basic unit of CPU utilization in traditional OS.
- A program in execution.
- Defines a data space.
- Has its own program counter, its own register states, its own stack and its own address space.
- Has at least one associated thread.
- Unit of distribution.

Threads

- It is a basic unit of CPU utilization used for improving a system performance through parallelism.
- A process consist of an address space and one or more threads of control.
- Threads share same address space but having its own program counter, register states and its own stack.
- Less protection due to the sharing of address space.

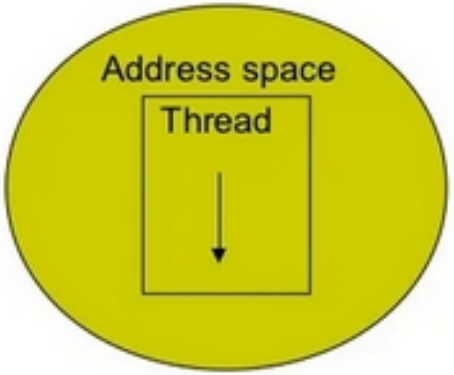
Cont…

- On a uniprocessor, threads run in quasi-parallel (time sharing), whereas on a shared-memory multiprocessor, as many threads can run simultaneously as there are processors.

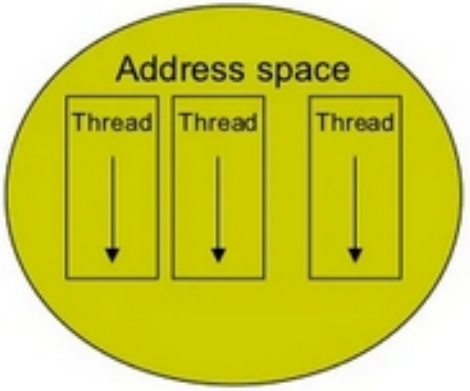
- States of Threads:
  - Running, blocked, ready, or terminated.

- Threads are viewed as miniprocesses.

Threads



(a) Single-threaded



(b) Multithreaded processes

Motivations for using Threads

- Overheads involved in creating a new process is more than creating a new thread.
- Context switching between threads is cheaper than processes due to their same address space.
- Threads allow parallelism to be combined with sequential execution and blocking system calls.
- Resource sharing can be achieved more efficiently and naturally between threads due to same address space.

Different models to construct a server process: As a single-thread process

- Use blocking system calls but without any parallelism.
- If a dedicated machine is used for the file server, the CPU remains idle while the file server is waiting for a reply from the disk space.
- No parallelism is achieved in this method and fewer client requests are processed per unit of time.

Cont… : As a finite state machine

- Model support parallelism but with nonblocking system calls.
- Implemented as a single threaded process and is operated like a finite state machine.
- An event queue is maintained for request & reply.
- During time of a disk access, it records current state in a table & fetches next request from queue.
- When a disk operation completes, the appropriate piece of client state must be retrieved to find out how to continue carrying out the request.

Cont… : As a group of threads

- Supports parallelism with blocking system calls.
- Server process is comprised of a single dispatcher thread and multiple worker threads.
- Dispatcher thread keeps waiting in a loop for request from the clients.
- A server process designed in this way has good performance and is also easy to program.

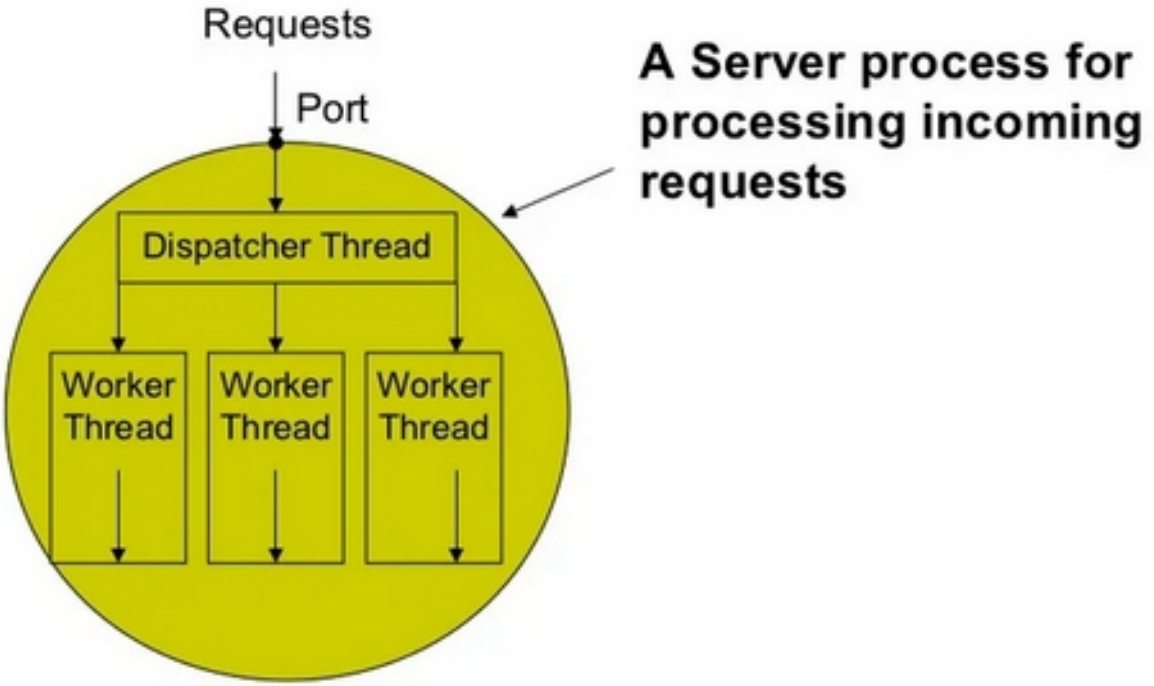
Models for organizing threads

- Dispatcher-workers model
- Team model
- Pipeline model

Cont… : Dispatcher-worker model

- Single dispatcher thread and multiple worker threads.
- Dispatcher thread accepts requests from clients and after examining the request, dispatches the request to one of the free worker threads for further processing of the request.
- Each worker thread works on a different client request.

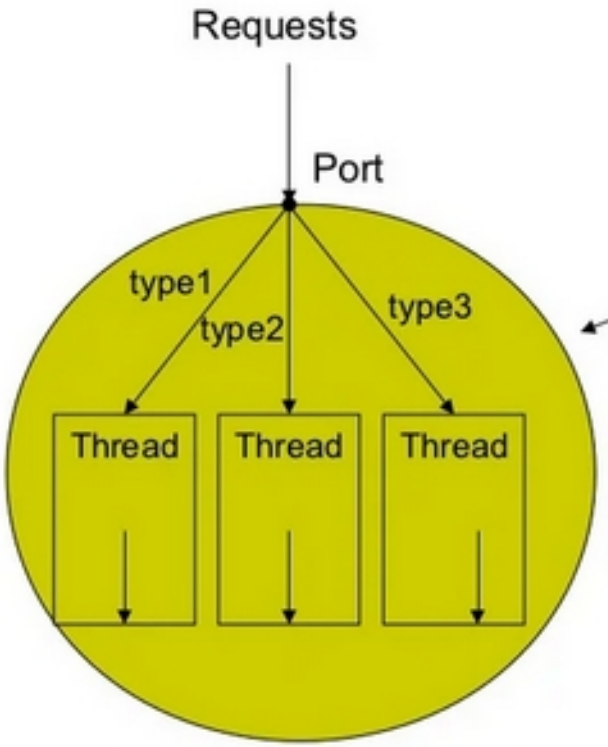
Cont…


Requests
Port
Dispatcher Thread
Worker Thread
Worker Thread
Worker Thread
A Server process for processing incoming requests

Cont… : Team Model

- There is no dispatcher-worker relationship for processing clients requests.
- Each thread gets and processes clients' requests on its own.
- Each thread of the process is specialized in servicing a specific type of request.
- Multiple types of requests can be simultaneously handled by the process.

Cont…



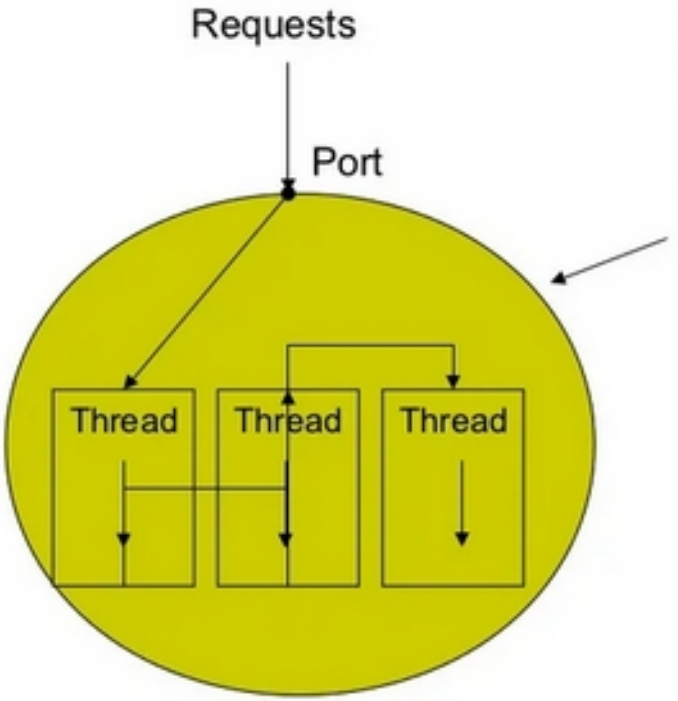
A Server process for processing incoming requests that may be of three different types, each types of request being handled by a different thread.

Cont… : Pipeline Model

- Useful for application based on the producer-consumer model.

- The threads of a process are organized as a pipeline so that the output data generated by the first thread is used for processing by the second thread and so on.

Cont…



A Server process for processing incoming requests, each request processed in three steps, each step handled by a different thread and output of one step as input to the next step

Issues in designing a Thread Package

1. Threads creation
2. Thread termination
3. Threads synchronization
4. Threads scheduling
5. Signal handling

Threads Creation

- Created either statically or dynamically.

- Static approach:
  - Threads remain fixed for its entire lifetime.
  - Fixed stack is allocated to each thread.
  - No. of threads of a process is decided at the time of writing a program or during compilation.

- Dynamic approach:
  - Number of threads changes dynamically.

Threads Termination

- A thread may either destroy itself when it finishes its job by making an exit call

or

- be killed from outside by using the kill command and specifying the thread identifier as its parameter.

Threads Synchronization

- Threads share a common address space, so it is necessary to prevent multiple threads from trying to access the same data simultaneously.

- Mechanism for threads synchronization:
  - Use of global variable within the process.
    - mutex variable and condition variable.

Cont…

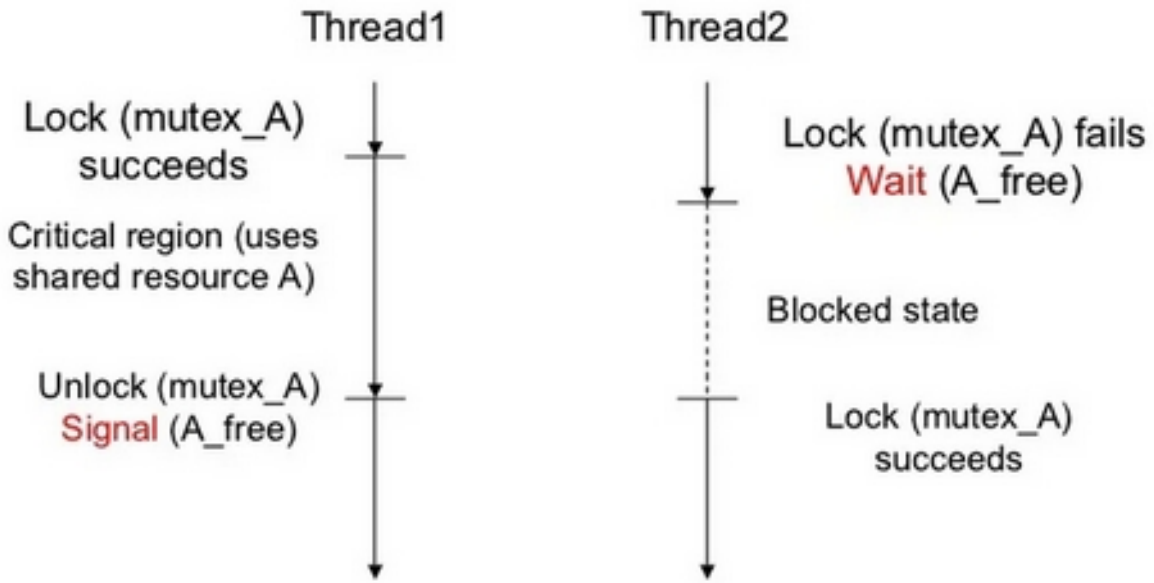
- Segment of code in which a thread may be accessing some shared variable is called a critical region.

- Two commonly used mutual exclusion techniques in a threads package are mutex variables and condition variables.

Cont…

- Mutex have only 2 states, hence simple to implement.
- Condition variable associated with mutex variable & reflects Boolean state of that variable.
- Condition – wait and signal operation.

Cont…



Mutex_A is a mutex variable for exclusive use of shared resource A.
A_free is a condition variable for resource A to become free.

Threads Scheduling

- Special features for threads scheduling :
  1. Priority assignment facility
  2. Flexibility to vary quantum size dynamically
  3. Handoff scheduling
  4. Affinity scheduling

Cont… : Priority assignment facility

- Threads are scheduled on the first-in, first-out basis or the round-robin policy is used.
- Used to timeshares the CPU cycles.
- To provide more flexibility priority is assigned to the various threads of the applications.
- Priority thread maybe non-preemptive or preemptive.

Cont… : Flexibility to vary quantum size dynamically

- Use of round-robin scheduling scheme.
- Varies the size of fixed-length time quantum to timeshare the CPU cycle among the threads.
- Not suitable for multiprocessor system.
- Gives good response time to short requests, even on heavily loaded systems.
- Provides high efficiency on lightly loaded systems.

Cont… : Handoff Scheduling

- Allows a thread to name its successor if it wants to.
- Provides flexibility to bypass the queue of runnable threads and directly switch the CPU to the thread specified by the currently running thread.

Cont… : Affinity scheduling

- A thread is scheduled on the CPU it last ran on in hopes that part of its address space is still in that CPU's cache.

- Gives better performance on a multiprocessor system.

Signal handling

- Signals provide software-generated interrupts and exceptions.

- Issues :
  - A signal must be handled properly no matter which thread of the process receives it.
  - Signals must be prevented from getting lost.

Cont…

- Approach:
  - Create a separate exception handler thread in each process.
  - Assign each thread its own private global variables for signaling conditions.

Thread Implementation

- Often provided in form of a thread package.
- Two important approaches:
  - First is to construct thread library that is entirely executed in user mode.
  - Second is to have the kernel be aware of threads and schedule them.

User level threads

- It is cheap to create and destroy threads.
- Cost of creating or destroying thread is determined by the cost for allocating memory to set up a thread stack.
- Switching thread context is done in few instructions.
- Only CPU registers need to be stored & subsequently reloaded with the previously stored values of the thread to which it is being switched.
- There is no need to change memory maps, flush the TLB, do CPU accounting etc.

Cont…

- Drawback: Invocation of a blocking system call will immediately block the entire process to which the thread belongs.

User and Kernel Level

- Threads are useful to structure large application into parts that could be logically executed at the same time.
- In user level thread, blocking on I/O does prevent other parts to be executed in the meantime.
- This can be circumvented by implementing threads in the OS Kernel.

Cont…

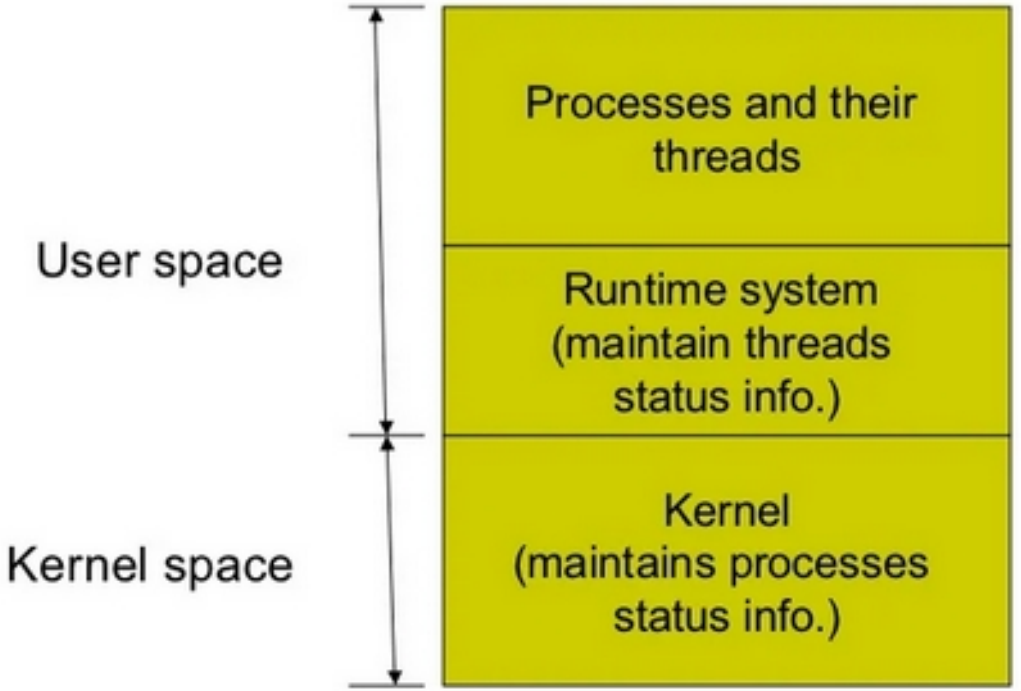
- Price: Every thread operation will have to be carried out by kernel requiring system calls. Hence benefits of using threads disappear.

Implementing a Threads Package

User-level approach

- Consists of a runtime system that is a collection of threads management routines.
- Threads run in the user space on the top of the runtime system and are managed by it.
- System maintains a status information table having one entry per thread.
- Two-level scheduling is preformed in this approach.

Cont… : User level

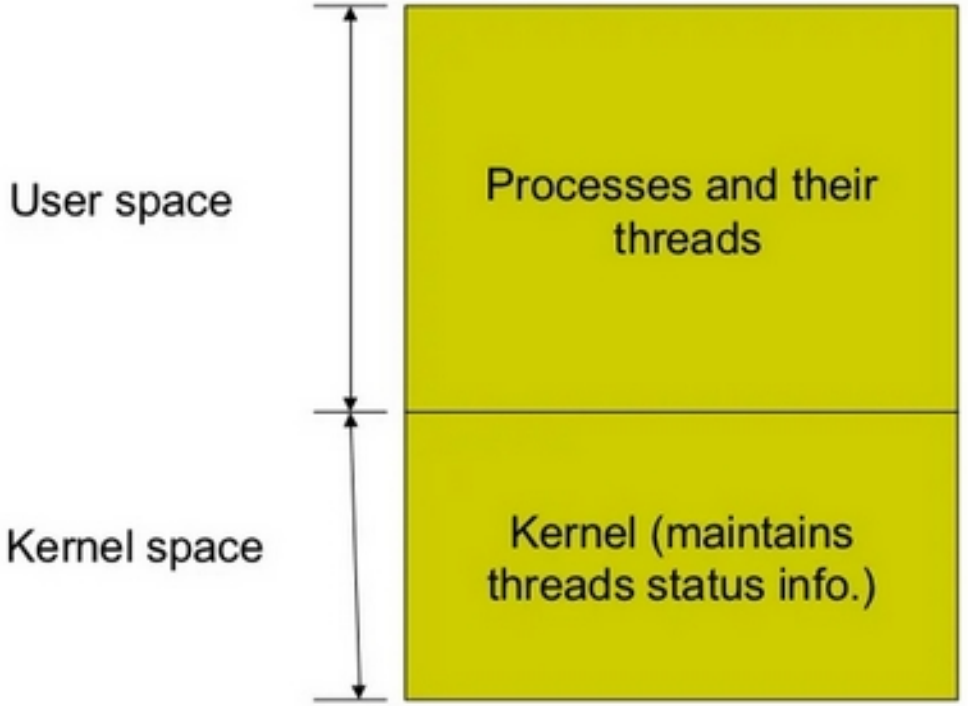


Cont…

Kernel-level approach

- No runtime system is used and threads are managed by the kernel.
- Threads status info. table maintained within kernel.
- Single-level scheduling is used in this approach.
- All calls that might block a thread are implemented as system calls that trap to the kernel.

Cont… : Kernel level



Cont…

Advantages of the approaches

- User-level approach can be implemented on top of an existing OS that does not support threads.
- In user-level approach due to use of two-level scheduling, users have the flexibility to use their own customized algorithm to schedule the threads of a process.
- Switching the context from one thread to another is faster in the user-level approach than in the kernel approach.
- No status information cause poor scalability in the kernel as compared to the user-level approach.

Cont…

Disadvantages of the approaches:

- Since there is no clock interrupt within a single process, so once CPU is given to a thread to run, there is no way to interrupt it.

- To solve this runtime system can request a clock interrupt after every fixed unit of time