PROCESS MANAGEMENT IN A DISTRIBUTED ENVIRONMENT

MCA Sem IV
Paper Code-CS4T16

-- Dr. Bhawna Sinha
Department of MCA, PWC

Process Management in a Distributed Environment

- Main goal of process management in DS is to make best possible use of existing resources by providing mechanism and polices for sharing them among processors
- **This achieve by providing :**
  - **Process allocation** : decide which process should assign to which process in any instance of time for better utilization of resources .
  - **Process migration** : move process to new node for better utilization of resources
  - **Thread facilities** : provide mechanism for parallelism for better utilization of processor capabilities.

Types of process migration

Process Management in a Distributed Environment

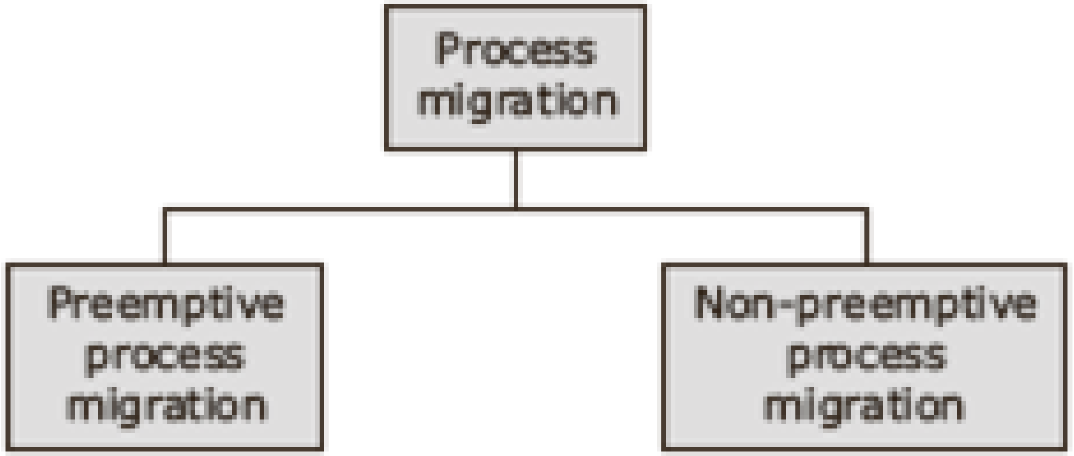
Main Functions of distributed process management

- **Process allocation :**decide which process should assign to which process in any instance of time
- **Process Migration:** Change of location and execution of a process from current processor to the destination processor for better utilization of resources and balancing load in distributed system.

3.6 Process Management in a Distributed Environment

Process Migration classified into :

- **Non-preemptive : process is migrate before start execution in source node .**
- **Preemptive : process is migrate during its execution .**


Process migration
Preemptive process migration
Non-preemptive process migration

Types of process migration

Desirable features of a good process migration mechanism

- Transparency:
- Minimal interference:
  - To the progress of process and system
  - Freezing time : time period for which the execution of the process is stopped for transferring its info to destination node
- Minimal residual dependencies:
  - Migrated process should not depend on its previous node

Desirable features of a good process migration mechanism Cont.

- Efficiency:
  - Issues
    - Time required to migrate process
    - Cost of locating the object
    - Cost of supporting remote execution once the process is migrated
- Robustness :
  - Failure of any other node should not affect the accessibility or execution of the process
- Ability to communicate between co processes of the job:
  - Communication directly possible irrespective of location

3.7 PROCESS MIGRATION

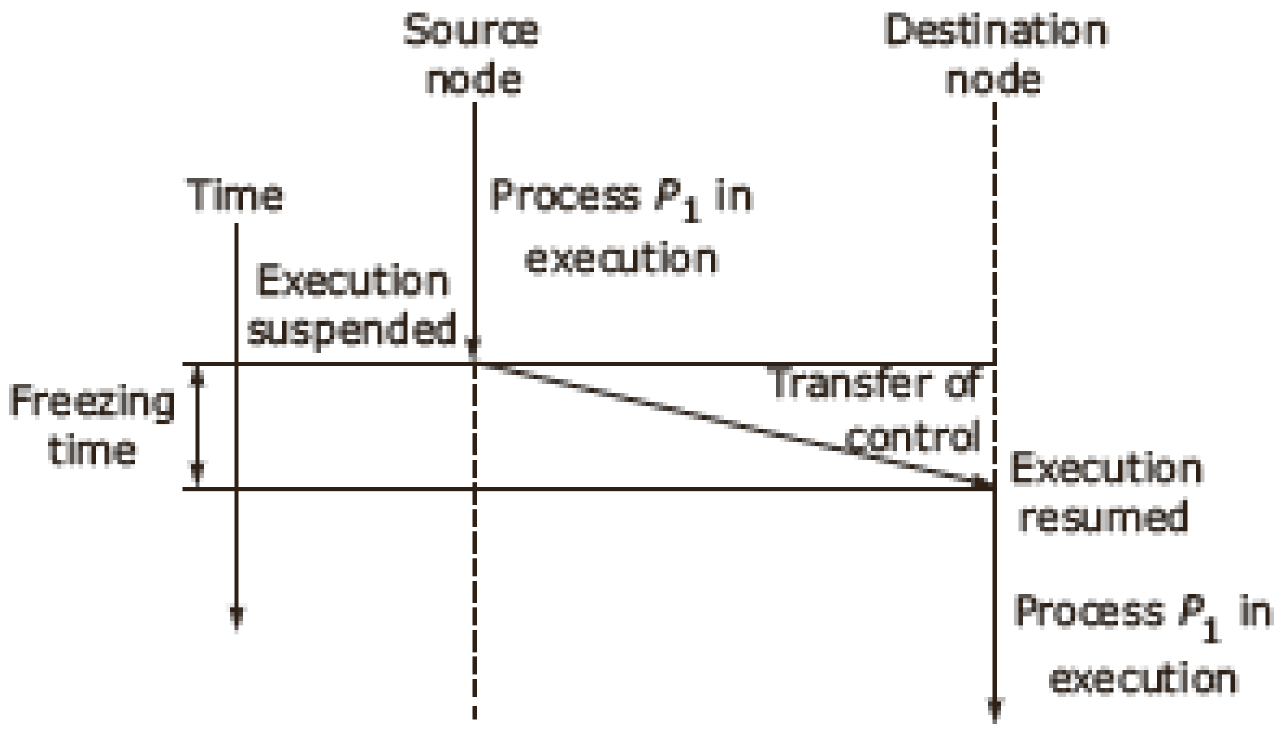
3.7 Process Migration

- In computing, **process migration** is a specialized form of **process** management whereby **processes** are moved from one computing environment to another.

Steps involved in process migration

- ➢Freezing the process on its source and restarting it on its destination
- ➢Transferring the process's address space (program code – data –stack program) from its source to destination
- ➢Forwarding messages meant for the migrant process
- ➢Handling communication between cooperating process

Mechanism



Process migration mechanism

Freezing process on source node

- Take snapshot of process's state on its source node
  - Reinstate the snapshot on the destination node
  - Issues
    - Immediate and delayed blocking of the process
    - Fast and slow I/O operations
      - After process is blocked wait for completion of fast I/O operations , then process is frozen
      - Slow I/O performed after migration
    - Information about open files
      - Use of Links to remote files
      - Use of local files as far as possible
    - Reinstating the process on its destination node
    - Constructing a new copy in the destination

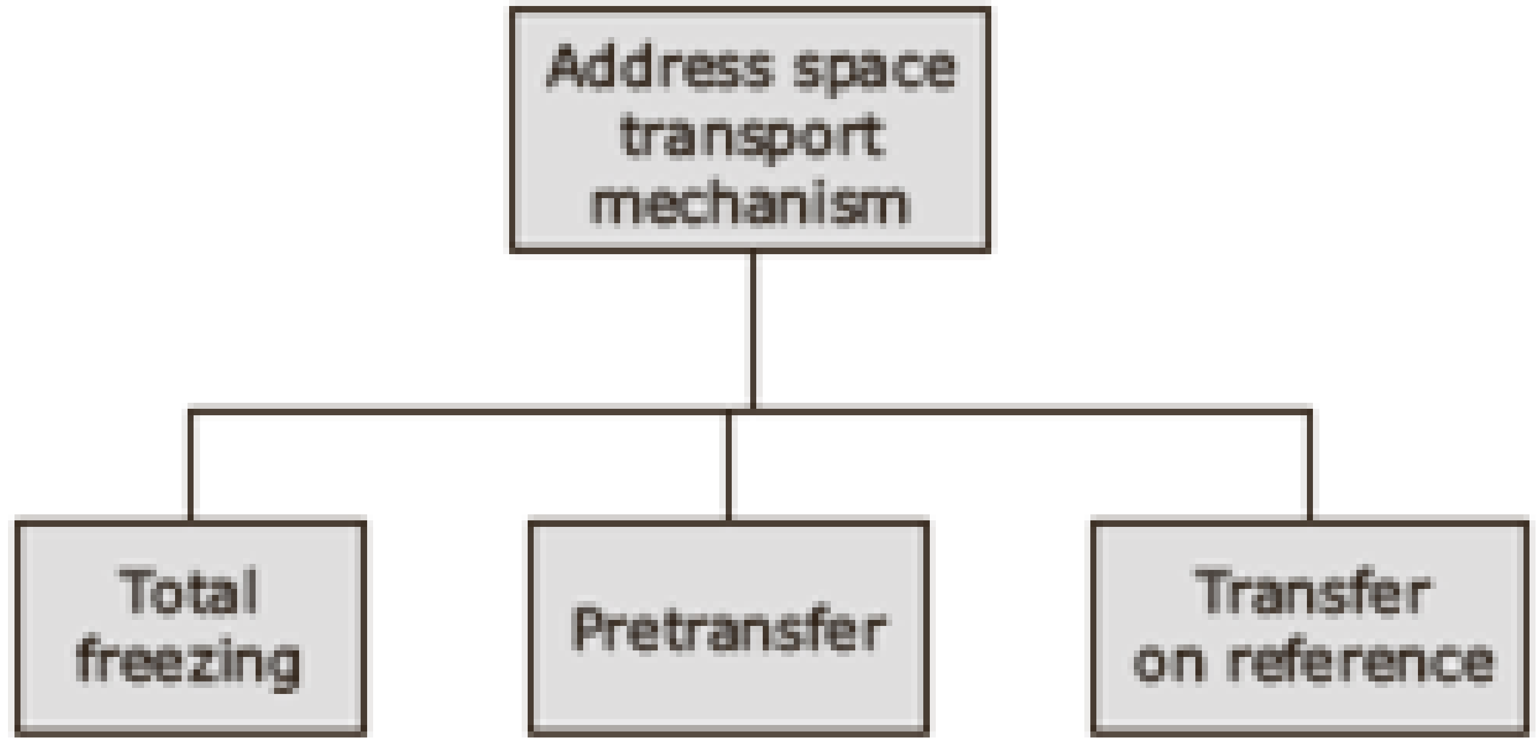
Address space transport mechanisms-1

- Information to be transferred

- Process's state
  - Execution status
  - Scheduling info
  - Info about RAM
  - I/O states
  - Objects for which the process has access rights
  - Info about files opened etc.

- Process's address space (code, data & stack)
  - Higher in size than process's state info
  - Can be transferred after migration, before or after process starts executing
  - **Address space transfer mechanisms**
    - Total freezing
    - Pretransferring
    - Transfer on reference

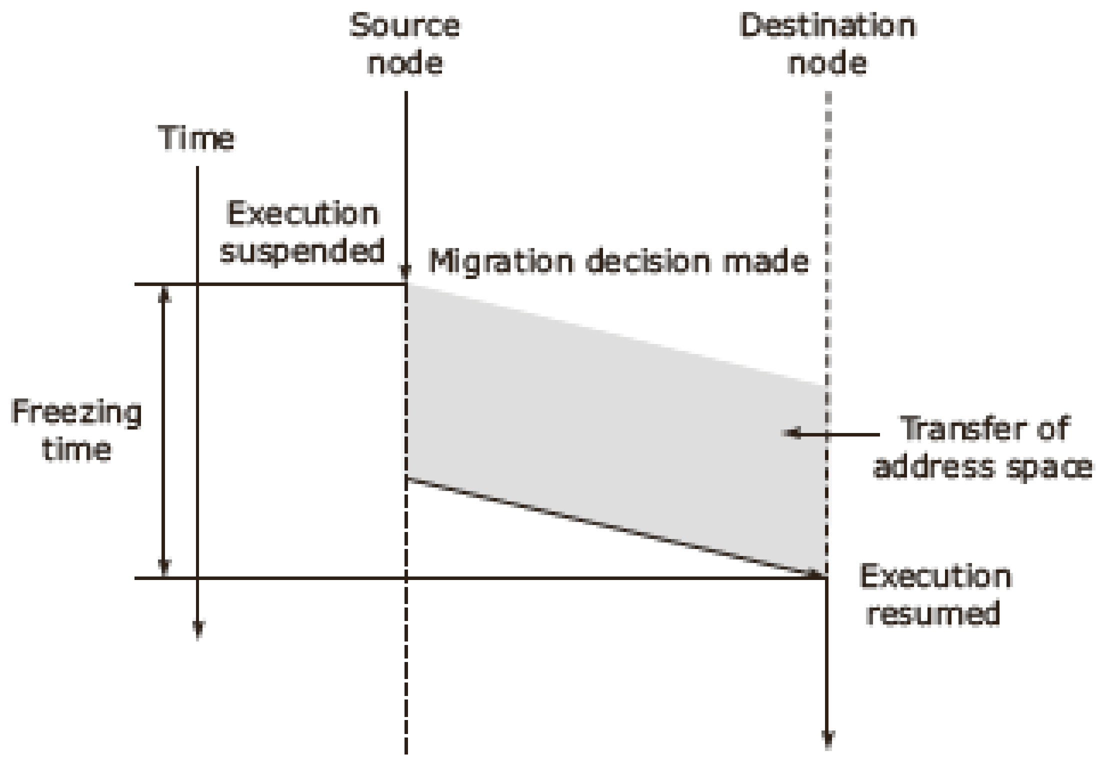
Address space transport mechanisms-1



Address space transport mechanism

Address space transport mechanisms-2

- Total Freezing
  - Process's execution is stopped while transferring the address space
  - Disadvantage that process may be suspended for a long time



Total freezing mechanism

Address space transport mechanisms-3

- Pretransfering or Precopying
  - Address space transferred while process is running on the source node

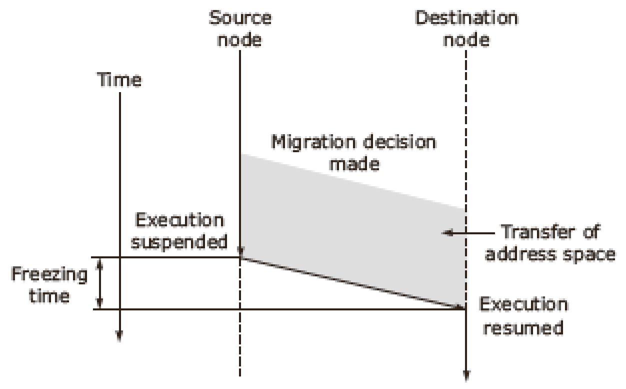


Figure 6-18 Pretransfer

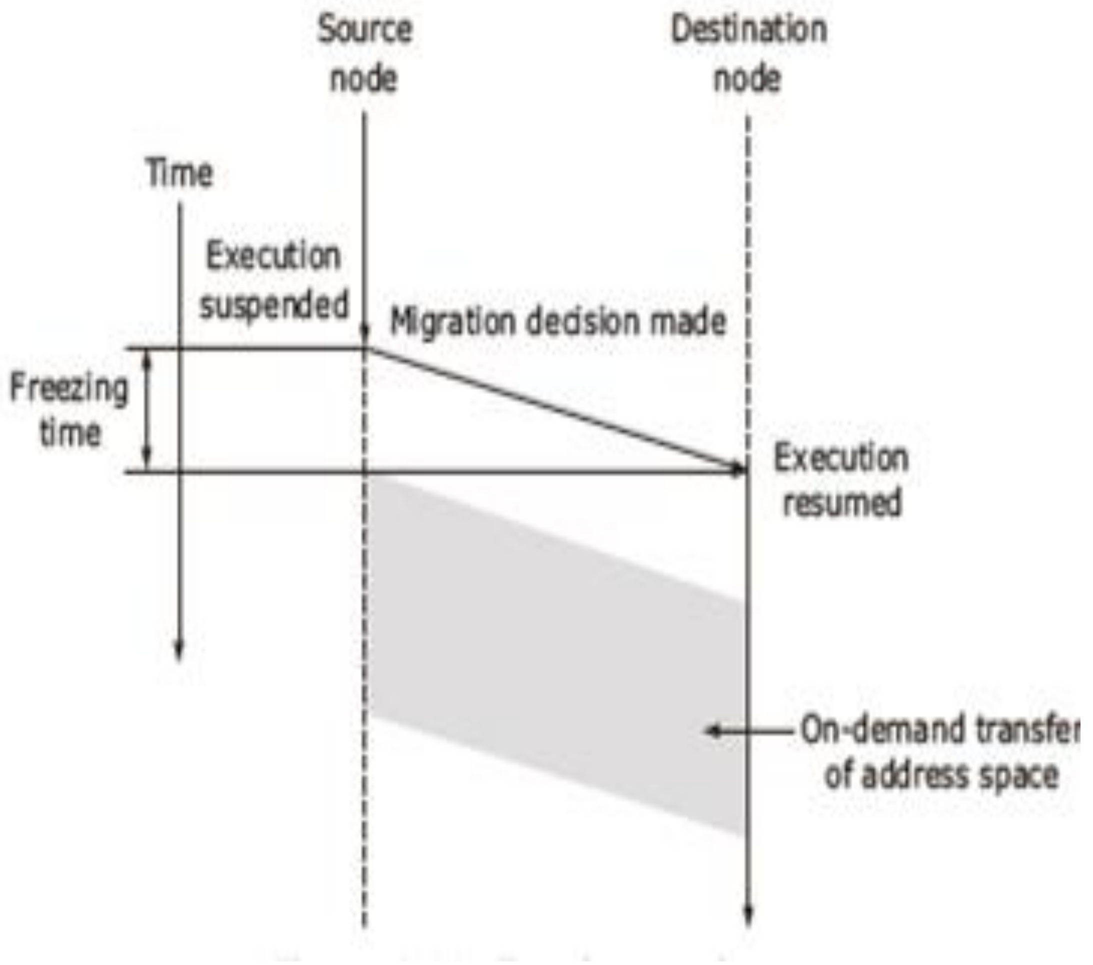
Pretransfer mechanism

Address space transport mechanisms-3

- Pretransfering or Precopying
  - Address space transferred while process is running on the source node
  - After decision for migration is made process continues to execute on source node until address space is has been transferred
  - Initially entire address space is transferred followed by repeated transfers of pages modified during previous transfer so on until no reduction in number of pages is achieved
  - The remaining pages are transferred after the process is frozen for transferring its state info
  - Freezing time is reduced
  - Migration time may increase due to possibility of redundant transfer of same pages as they become dirty while pretransfer is being done

Address space transport mechanisms-4

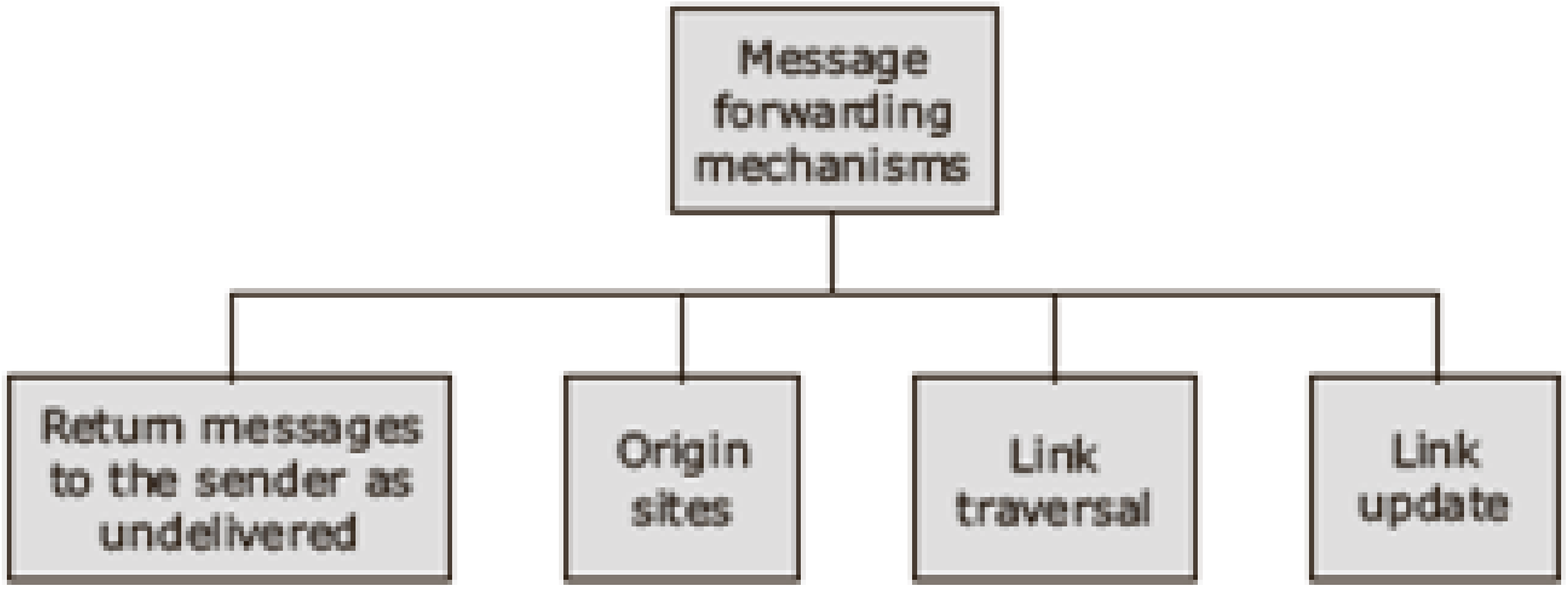
- Transfer on Reference
  - Process executes on destination
  - Address space is left behind in source node
  - Desired blocks are copied from remote locations as and when required
  - Failure of source node results in failure of process



Transfer on Reference mechanism

Messages Forwarding

- Track and forward messages which have arrived on source node after process migration


Message forwarding mechanisms
Return messages to the sender as undelivered
Origin sites
Link traversal
Link update

Messages Forwarding

- Messages
  - Messages received at source node after the process stopped on its source node and not started on the destination node
  - Messages received at source node after execution
  - Messages of process started at destination node

Message forwarding Mechanisms

- Return message to sender as undeliverable
  - Message type 1 and 2are returned to sender or dropped
  - Sender retries after locating the new node (using locate operation)
  - Type 3 message directly sent to new node
- Origin site mechanism
  - All messages are sent to origin site
  - Origin site forwards the messages
  - If origin site fails forwarding mechanism fails
  - Continuous load on the origin site

Messages Forwarding Message forwarding Mechanisms

- Link traversal mechanism
  - Message queue is generated at origin
  - Message Forwarded to destination node
  - After process is migrated link is left on the previous node
  - Process address has two parts process id, last known location of destination node

Advantages of process migration

- Reduce average response time of heavily loaded nodes
- Speed up of individual jobs
- Better utilization of resources
- Improve reliability of critical processes

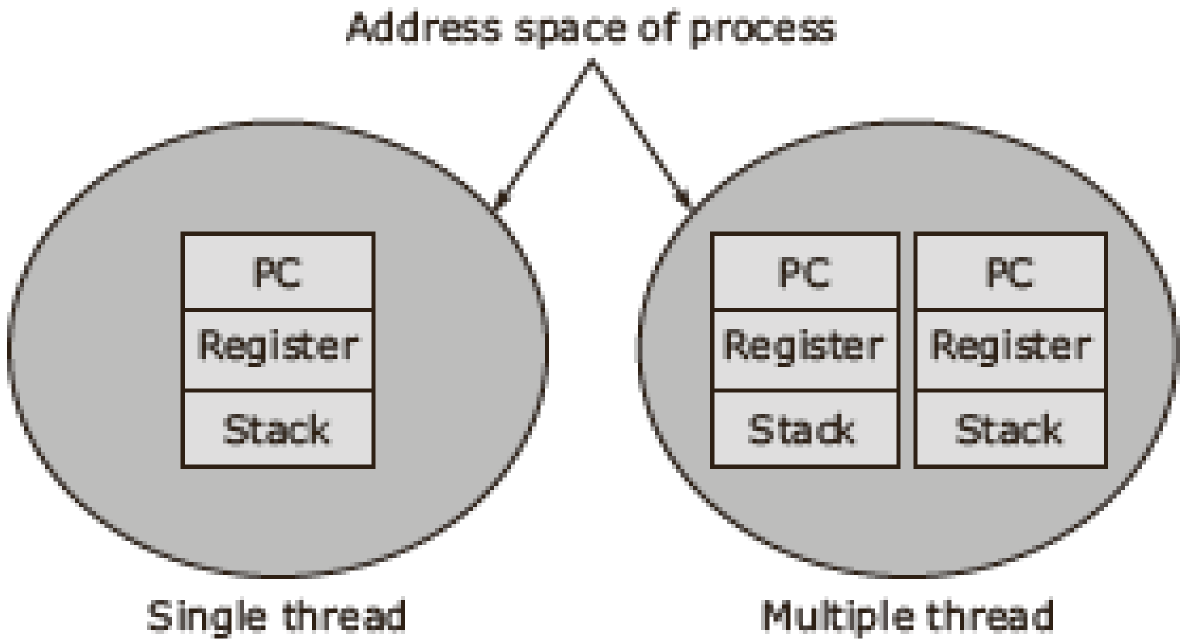
3.8 THREADS

PROCESS V/S THREADS

- Programs are divided to process , which that can be independently executed .
- Process can block itself , while waiting for some other operation to be complete , and program execution can slow down .
- We cannot use multiple processes , since processes don’t share address space. Instead if this process is divided into threads, one of the threads can go to sleep , while other threads may continue execution. thus , system throughput is increased and this in turn improve system performance .

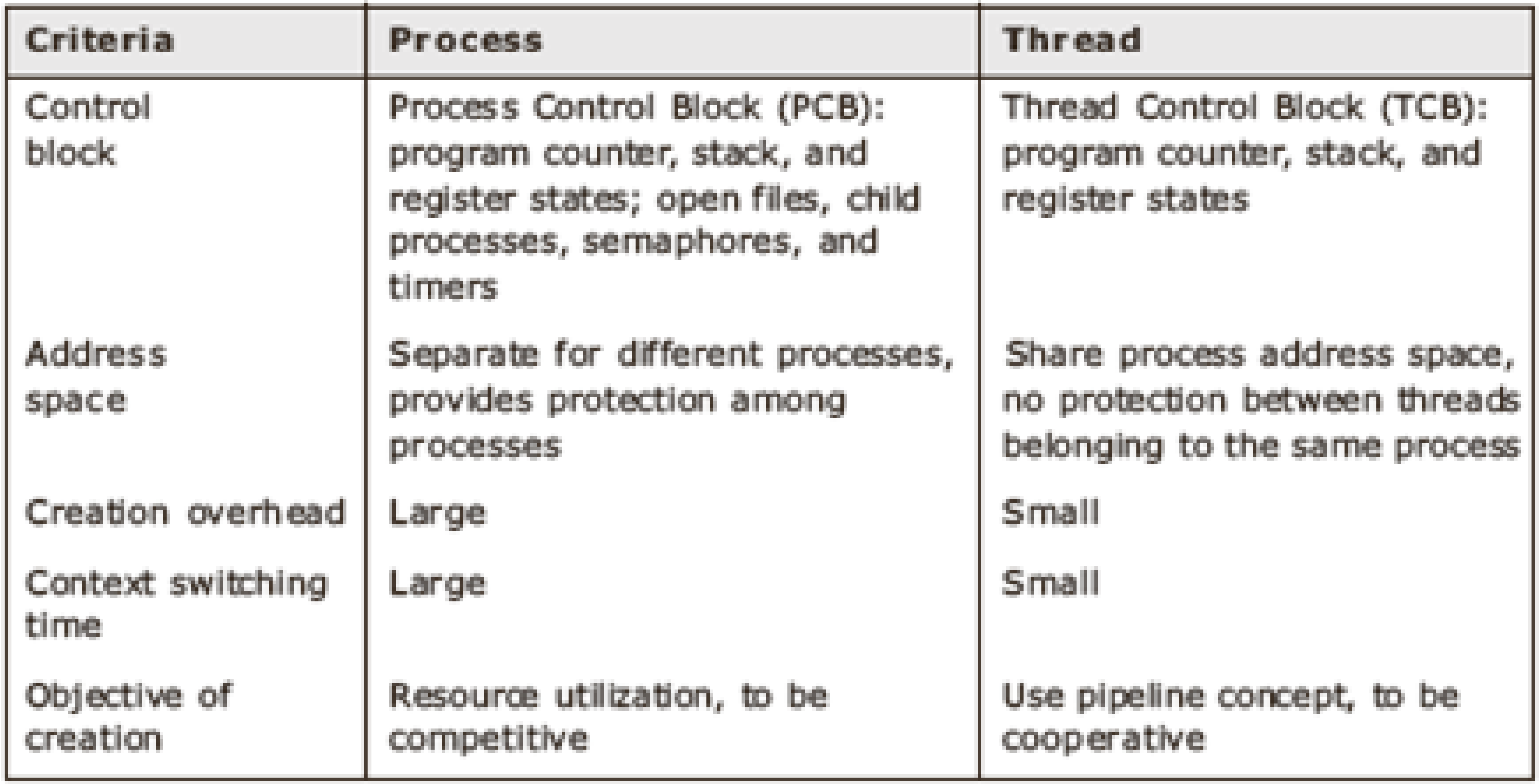
Process v/s threads

- Analogy:
  - Thread is to a process as process is to a machine



Process address space

Comparison



Criteria	Process	Thread
Control block	Process Control Block (PCB): program counter, stack, and register states; open files, child processes, semaphores, and timers	Thread Control Block (TCB): program counter, stack, and register states
Address space	Separate for different processes, provides protection among processes	Share process address space, no protection between threads belonging to the same process
Creation overhead	Large	Small
Context switching time	Large	Small
Objective of creation	Resource utilization, to be competitive	Use pipeline concept, to be cooperative

Comparison of processes and threads

Thread models

- Dispatcher worker model
- Team model
- Pipeline model

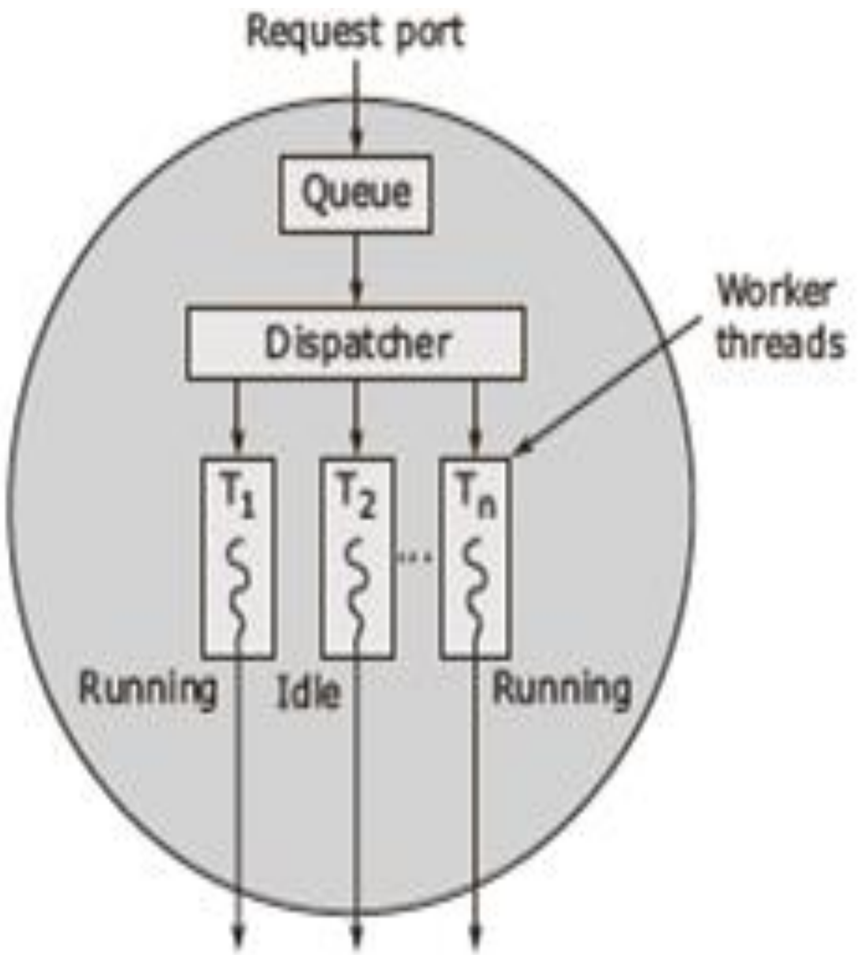
Thread: Dispatcher worker model

Typical example of this model is server process such as file server that :

1. Accept request from client .
2. Check for access permission.
3. Accordingly services the request.

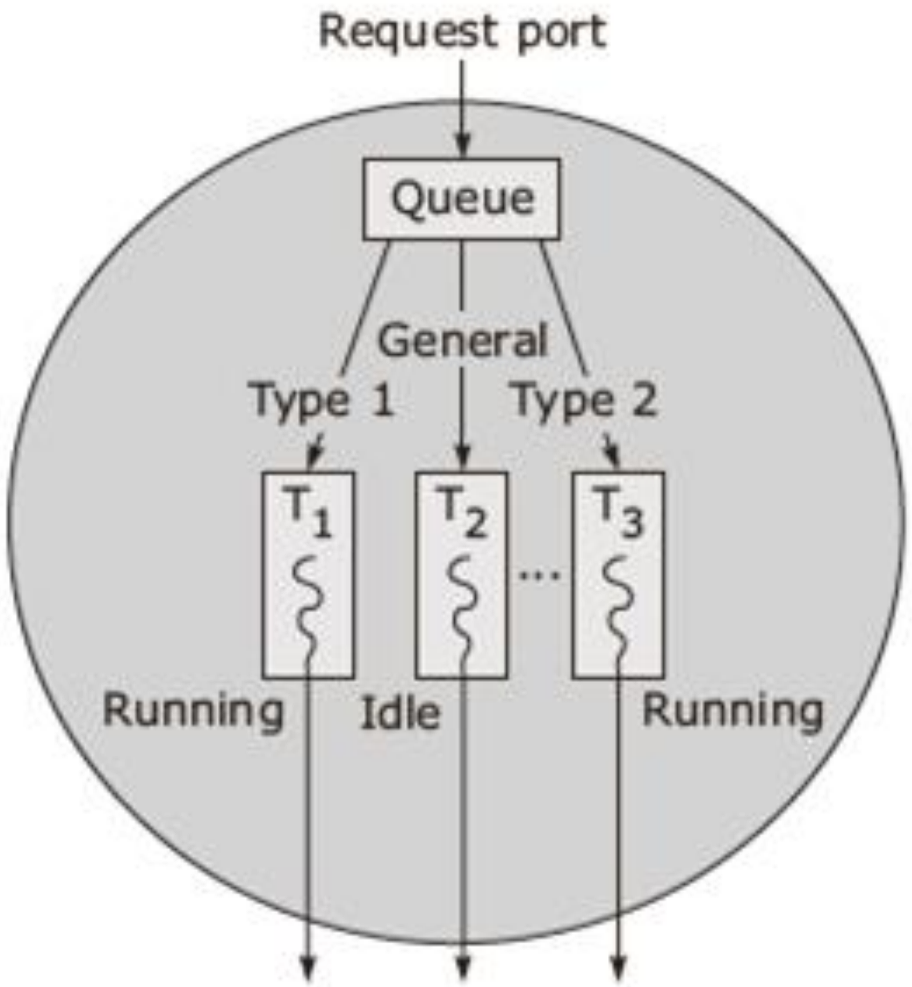
Thread: Dispatcher worker model

- Assume that Single process is divided into one dispatcher and multiple worker
- Dispatcher thread accept incoming request from client request queue.
- Examine request and choose idle worker thread to handle request.
- Thus worker thread change its state to running and dispatcher change state to ready .
- Since each worker thread processes different client request , multiple request can be processed in parallel.



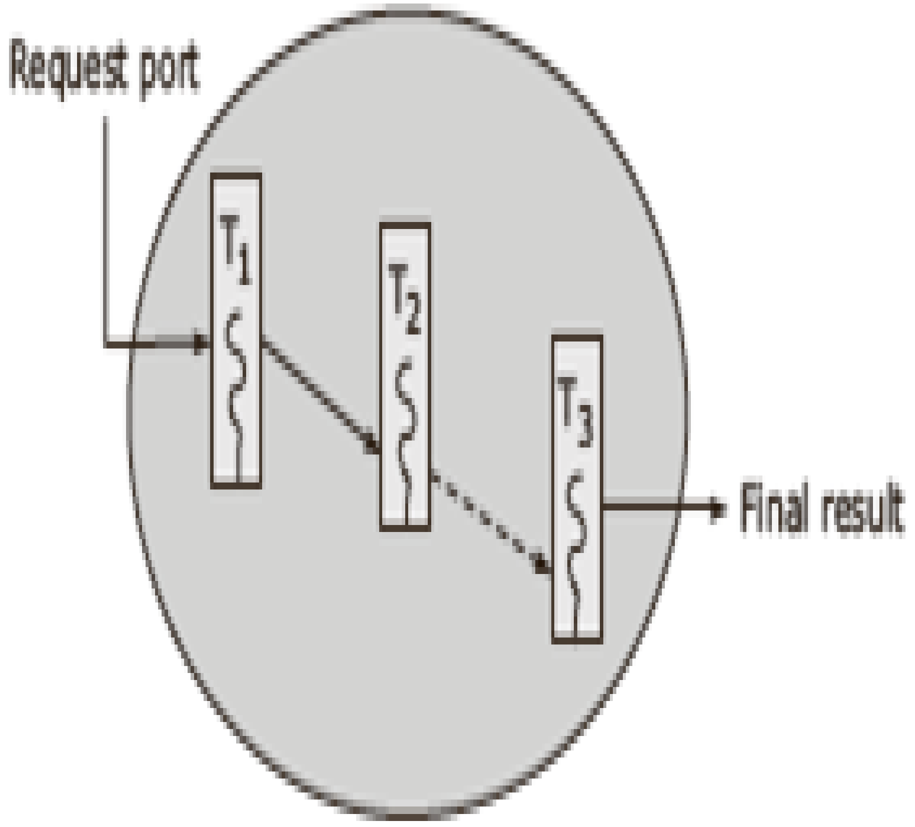
Thread: Team model

- All threads are treated equal , such that each one handle request on its own .
- In case threads are capable of performing specific distinct function , a queue can be maintained .
- When thread change state from running to idle , it take new request from the job queue and stars execution .



Thread: Pipeline model

- Used pipeline concept that used in CPU instructions executions.
- The tasks assigned to the threads are completed and the result generated by first thread are passed to next thread .
- It take this as input and start running .
- Data pass across multiple threads with each one processing it partly and the last thread giving the final result .



Design issues in threads

- Thread semantics
  - Thread creation, termination
- Thread synchronization
- Thread scheduling

Design issues in threads : Thread semantics

- The first step before using threads is **thread creation** that cab be :

  - Static .

  -Dynamic .

Design issues in threads : Thread semantics

- Static : number of threads to be created is fixed when program is written or when it is complied , and memory space is allocate to each thread.

Design issues in threads : Thread semantics

- Dynamic : threads are created as and when it is needed during the process life cycle .and they exit when task is completed .Here the stack size for the threads is specified as parameter to the system call for thread creation.

Design issues in threads : Thread semantics

- Threads termination : threads follow the same steps for termination as processes either :

  - ***EXIT*** call command : thread destroys itself on task completion by making an *EXIT* call .

  -***KILL*** call command :or thread is killed from outside using *KILL* command with thread id as parameter.

Design issues in threads

- Thread semantics
  - Thread creation, termination
- **Thread synchronization**
- Thread scheduling

Thread synchronization

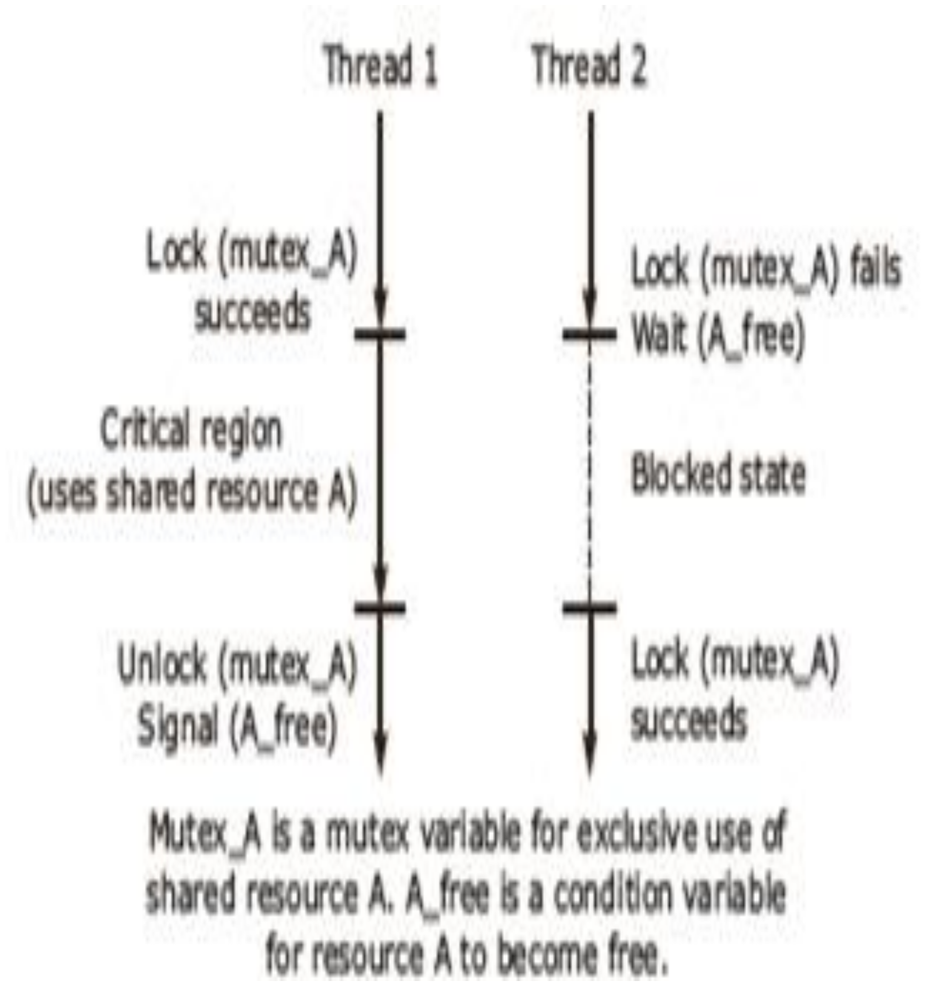
- Since threads belong to process share the same address space , thread synchronization are required to ensure that multiple threads don’t access the same data simultaneously.
- For example , if two threads want to double the same global variable , it is best done one after another.

Thread synchronization : example

- if two threads want to double the same global variable , it is best done one after another.
- One thread should exclusive access to shared variable , double it , and pass control to the other thread.
- To provide exclusive access to shared variables , we define critical region .It mean that only one thread can execute in critical region at any instance of time.
- Critical region is implemented using mutex variable , which is binary semaphore: *locked* and *unlocked*

Thread synchronization

- Execution in Critical region
  - Use binary semaphore
- The lock operation attempts to lock the mutex. It successes if unlocked m and mutex become locked in single atomic action.
- If two threads try to lock the same mutex at the same time , only one successes , while the other thread is blocked .


Thread 1
Thread 2
Lock (mutex_A) succeeds
Lock (mutex_A) fails
Wait (A_free)
Critical region (uses shared resource A)
Blocked state
Unlock (mutex_A)
Signal (A_free)
Lock (mutex_A) succeeds
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.

Implementation of critical region

Design issues in threads

- Thread semantics
  - Thread creation, termination
- Thread synchronization
- **Thread scheduling**

Threads scheduling

- Another important issue in designing threads package is to decide an appropriate scheduling algorithm.
- The Threads packages provide application programmer with calls to specify scheduling policy to be used for application execution .

Threads scheduling policies/algorithms

- Priority assignment facility
- Choice of dynamic variation of quantum size
- Handoff scheduling scheme
- Affinity scheduling scheme
- Signals used for providing interrupts and exceptions

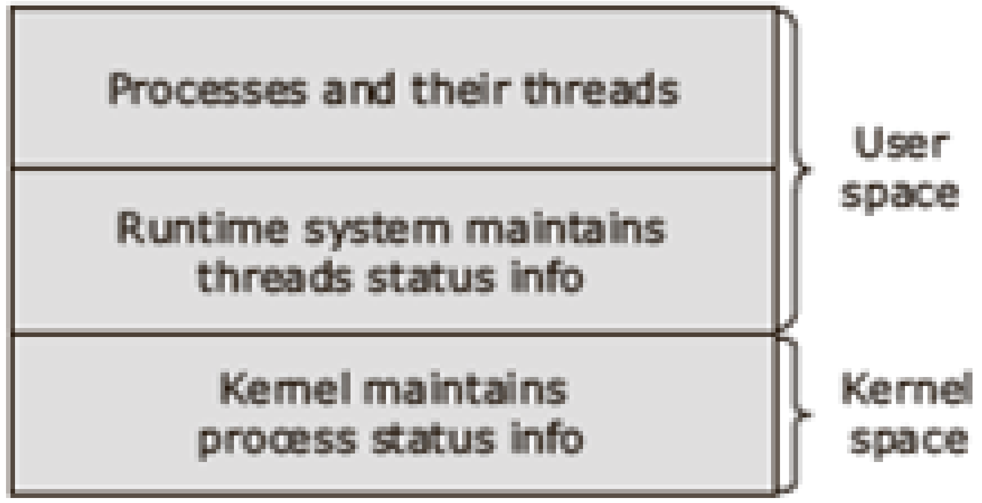
Implementing thread package

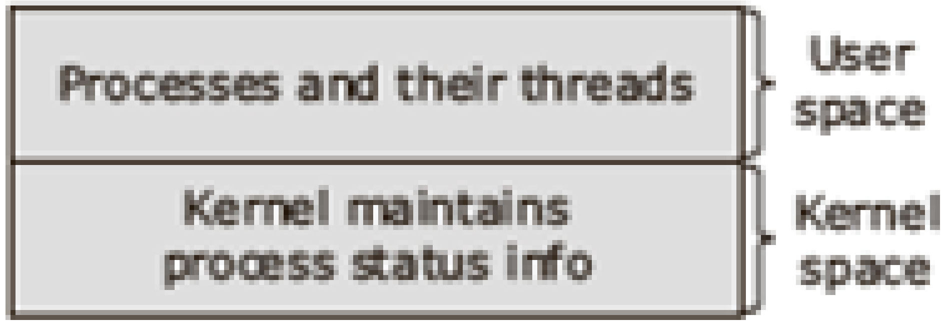
- Typically any OS divides memory into users and kernel space to store programs and data .
- **Thread package can be implemented either in**
  - user space
  - kernel space .

Implementing thread package

- **User level approach**

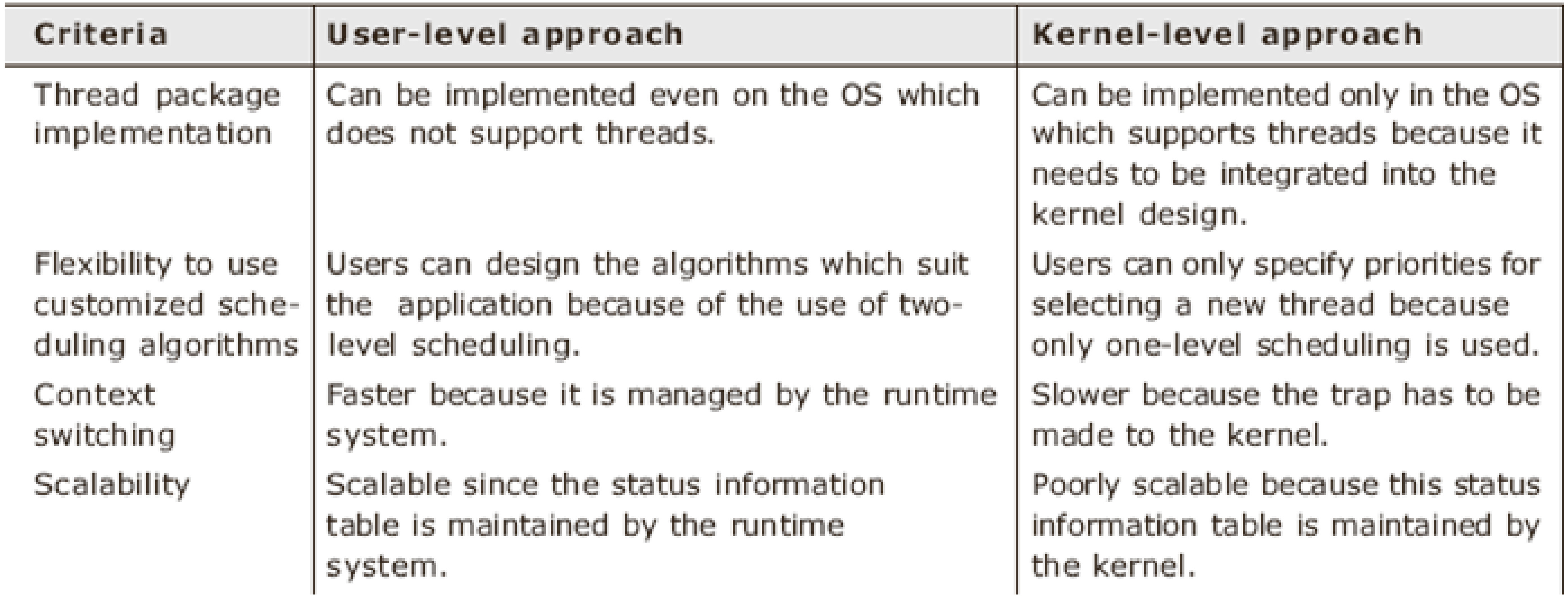
Kernel level approach


Processes and their threads
Runtime system maintains threads status info
Kernel maintains process status info
User space
Kernel space


Processes and their threads
Kernel maintains process status info
User space
Kernel space

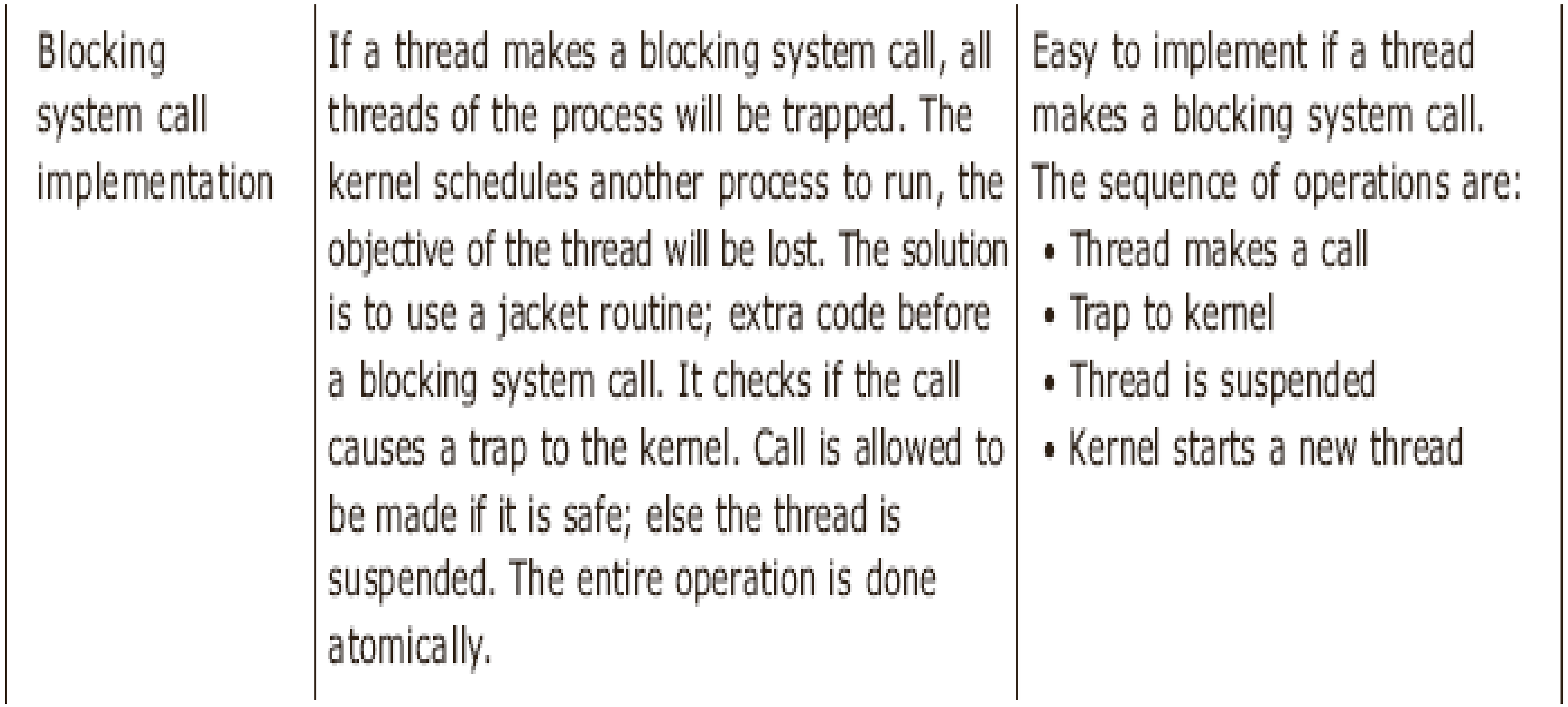
Comparison of thread implementation-1

User- level vs. kernel-level thread implementation



Criteria	User-level approach	Kernel-level approach
Thread package implementation	Can be implemented even on the OS which does not support threads.	Can be implemented only in the OS which supports threads because it needs to be integrated into the kernel design.
Flexibility to use customized sche-duling algorithms	Users can design the algorithms which suit the application because of the use of two-level scheduling.	Users can only specify priorities for selecting a new thread because only one-level scheduling is used.
Context switching	Faster because it is managed by the runtime system.	Slower because the trap has to be made to the kernel.
Scalability	Scalable since the status information table is maintained by the runtime system.	Poorly scalable because this status information table is maintained by the kernel.

Comparison of thread implementation-2

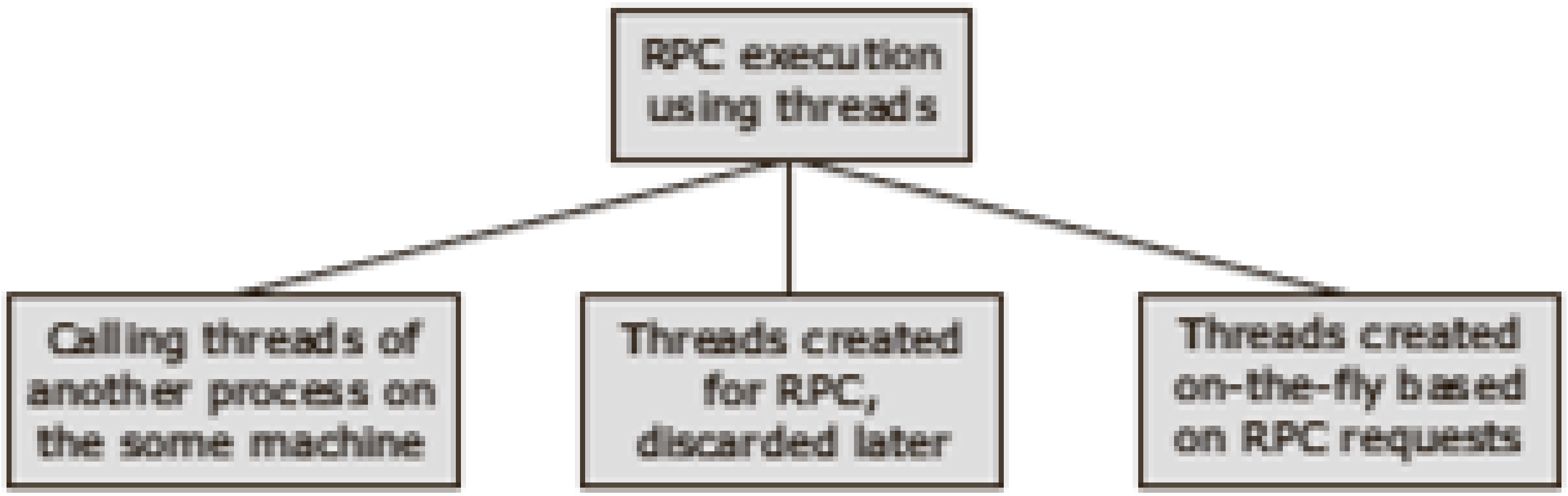


Blocking system call implementation	If a thread makes a blocking system call, all threads of the process will be trapped. The kernel schedules another process to run, the objective of the thread will be lost. The solution is to use a jacket routine; extra code before a blocking system call. It checks if the call causes a trap to the kernel. Call is allowed to be made if it is safe; else the thread is suspended. The entire operation is done atomically.	Easy to implement if a thread makes a blocking system call. The sequence of operations are: • Thread makes a call • Trap to kernel • Thread is suspended • Kernel starts a new thread

Threads and Remote execution

There are two different ways of remote execution of thread :

- RPC : distributed systems commonly use RPC (remote procedure call).
- RMI (remote method invocation) and Java threads


RPC execution using threads
Calling threads of another process on the some machine
Threads created for RPC, discarded later
Threads created on-the-fly based on RPC requests

Types of RPC execution in distributed system