2. PROCESS MANAGEMENT in DISTRIBUTED SYSTEMS

References [Cho97], [Col94], [Gos91]

2. PROCESS MANAGEMENT in DISTRIBUTED SYSTEMS
PROCESSES in DISTRIBUTED SYSTEMS
- **Processes and system structure**
- **Types of processes**

OPERATIONS on PROCESSES in DISTRIBUTED SYSTEMS
- **Creation of a new process**
- **Termination of a process**
- **Distributed process scheduling and dispatching**

THREADS in DISTRIBUTED SYSTEMS
- **Processes & threads**

MODELS of THREAD APPLICATION
- **Dispatcher-worker model**
- **Team model**
- **Pipeline model (or producer-consumer model)**

OPERATIONS on THREADS
APPLICATION of THREADS in CLIENTS and SERVERS
- **Server with multiple threads**
- **Client with multiple threads**

THREADS PACKAGE
- **Services of the threads package**
- **Thread calls in a threads package**

IMPLEMENTATION of THREAD PACKAGE
- **User level threads package**
- **Kernel level threads package**
- **Kernel-user level thread package**

REMOTE EXECUTION
REQUIREMENTS of REMOTE EXECUTION
REMOTE EXECUTION POLICIES and MECHANISMS
- **Activities of the remote execution policy**
- **Activities of remote execution mechanism**

REMOTE EXECUTION in HETEROGENEOUS SYSTEMS
ADVANTAGES of REMOTE EXECUTION

PROCESS MIGRATION
FEATURES of PROCESS MIGRATION
PROCESS MIGRATION POLICIES and MECHANISMS
- **Activities of process migration mechanism**
- **Implementation of process migration**

PROCESS MIGRATION in HETEROGENEOUS SYSTEMS
ADVANTAGES of PROCESS MIGRATION

PROCESSES in DISTRIBUTED SYSTEMS

Processes and system structure

a. processes in the same address space

1. shared code segments:

- **address spaces of processes are protected from corruption and it can be shared by processes,**
- **the address space in each process is a virtual address space mapped onto the same physical memory,**

2. shared data segment:

- **processes have an arbitrary sharing mechanism that affords sharing some data areas while other areas are private,**

b. processes in separate address space

1. distributed processes:

- **processes are on separate computers and they occupy separate address spaces,**
- **their control blocks and states are distributed among computers of DS,**
- **processes communicate between them through communication network,**
- **processes of a single task may be running on one computer , or they can be spread among several computers (remote execution or process migration),**

Types of processes

a. heavy-weight process

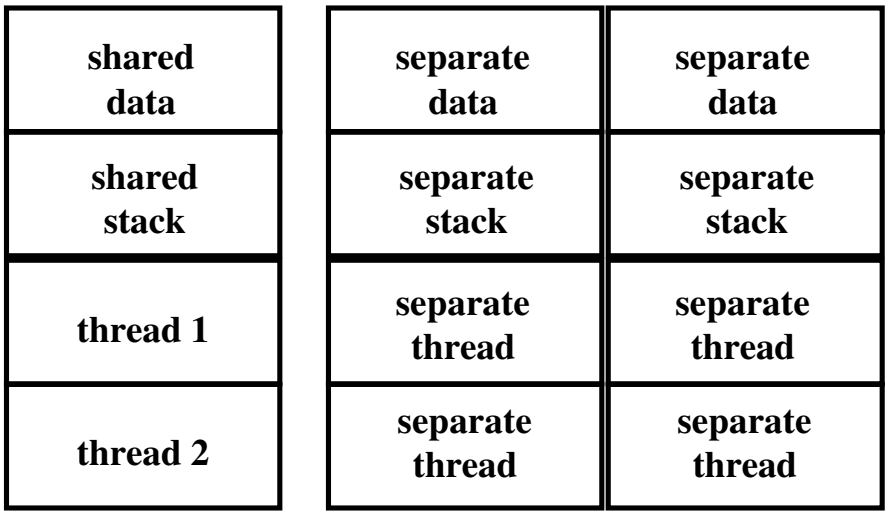
- **they run in separate address space, and have separate entry in a process control block, i.e. they do not share address space with other processes,**
- **there is only one thread in each process,**

b. medium-weight process

- **they are compromise between lightweight and heavyweight processes,**
- **there are several threads of control, where each thread is considered as a separate process,**

c. light-weight process

- **they contain more than one thread and all of them use the same data, stack and text,**
- **it is permitted to threads to read and write the same memory locations,**
- **they are executed on a uniprocessor or on a single shared memory multiprocessor,**



types of processes

OPERATIONS on PROCESSES in DISTRIBUTED SYSTEMS

Creation of a new process

process creation procedure

1. the choice of a target host

- **a process is allocated to a processor,**
- **there are two approaches to allocate processes to processors:**

  a. the system allocates all processes and initiates and controls the process migration,

  b. the programmer or the user creates the processes on specific processors, i.e. he/she makes the allocation decisions,

2. the creation of the execution environment

- **execution environment of a process, which should be created, contains:**

  - address space with initialised contents,

  - initial set of ports for communicating with services,

approaches to define and initialise the address space:

a. statically defined execution environment

- address space regions are created from a list specifying their extent,

b. dynamically defined execution environment

- the existing execution environment is taken into account,

- each region of the parent process may be inherited or omitted by the child process,

- an inherited region may either be shared with or logically copied from the parent's region,

<u>approaches to create and allocate ports:</u>

- the interprocess communication is based upon the passing messages between ports,

- a process may be created with a data port which it owns and a service port which is owned by the system in order to communicate with the system,

- only the creator of a port may receive messages on that port, they can be but are usually not passed on,

- these rights can be passed on by creator to any processes with which it wishes to communicate

a. ports can be created when a process is created:

- **the parent process may give a child process a set of ports, including input and output ports, that can be used immediately**

b. ports can be created when a process demands it:

- **the parent process gives a child process ports to communicate with the system only,**

- **when the child process needs to communicate with other processes it has to demand that ports be created for this communication,**

3. the creation a thread within the process

Termination of a process

- processes in some systems may exist forever, thus process termination mechanism is required,

- there are several process termination mechanisms:

1. a process can terminates automatically when the command list has been executed,

2. a parent process can terminate one of its children processes by issuing the command "kill",

- a port of a process can be terminated:

1. at any time by the owner process,

2. when a process is destroyed all ports allocated to it are also destroyed,

Distributed process scheduling and dispatching

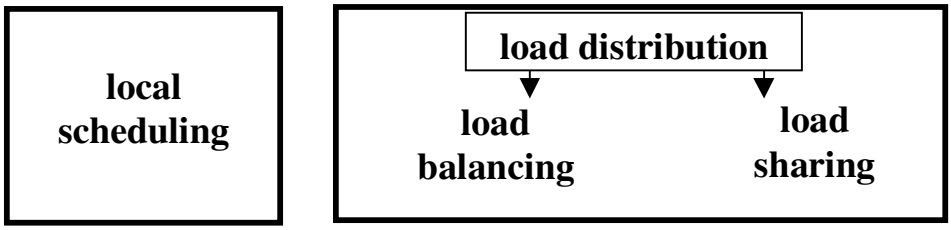
- scheduler = allocates a ready process to an available processor

scheduling is performed globally and locally, i.e. distributed scheduling

- dispatcher = removes the process selected by the scheduler from the ready queue, changes it status and loads the processor state,

dispatching is performed locally,

- distributed scheduling can be divided into:



distributed scheduling

a. local scheduling that determines how a processor is allocated among its resident processes

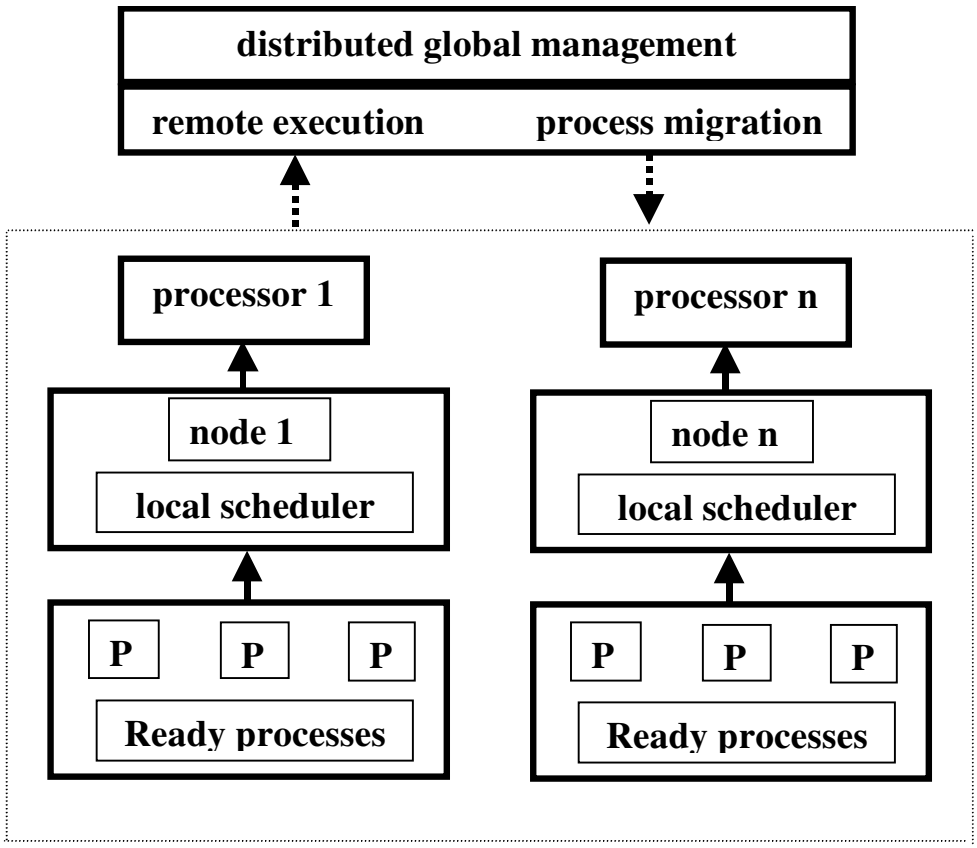
- **local scheduling does not guarantee the best performance for a process. thus processes should be scheduled globally,**

b. load distributing that defines the overall workload of DS among the processors at different computers through remote execution and process migration

- **load distribution is performed by load balancing and load sharing mechanisms,**

- ***load balancing* = equalises the workload among processors at different computers**

- ***load sharing* = assures that no computer is idle while some processors are overloaded, and processes wait for service**



local and distributed scheduling

THREADS in DISTRIBUTED SYSTEMS

Processes & threads

problem:

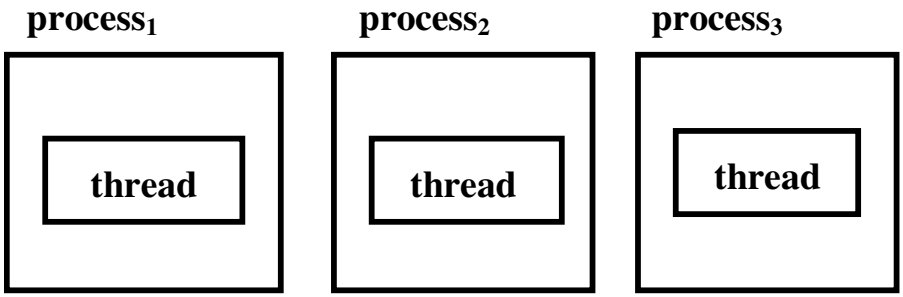
- **when one activity is assigned to a process it is an expensive resource to create and manage**

solution:

- **a process and its execution environment with one or more threads**

- execution environment of the process contains the:

- **address space,**
- **low-level resources (thread communication & synchronisation),**
- ***remark:* high level resources for example: files & windows are not included in the execution environment, they are provided by servers,**



three processes with single-thread

THREAD = the operating system level abstraction of the activity of the process,

- thread share all resources accessible within the execution environment of a process to decrease costs of process management, thus threads share the same global variables and the same set of OS resources,

- data and other resources within an execution environment are

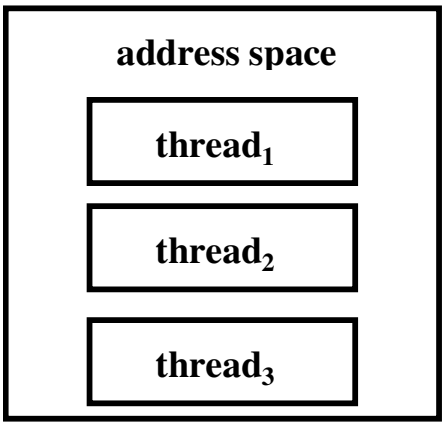
inaccessible to threads of other execution environments,

- threads maximise the concurrent execution between closely-related operations,

- threads can create and terminate child threads,

- threads can block for system calls to complete and can change their states during their execution

process components	thread components
- own program counter	- own program counter
- own register states	- own register states
- own stack	- own stack
- own address space	- common address space of the process
- different global variables	- same global variables
- protection between processes	- no protection between threads
- communication through IPC primitives	- communication through the common address space



one - multithreaded - process with three threads

MODELS of THREAD APPLICATION

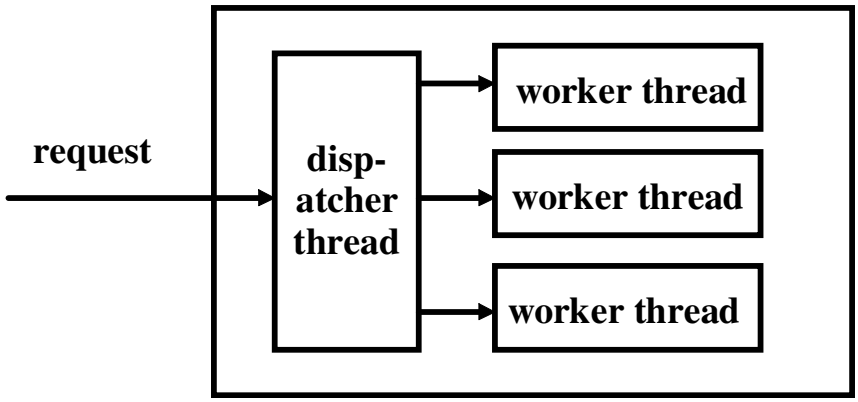
Dispatcher-worker model

- process with a single dispatcher thread and multiple worker threads,

- the dispatcher thread accepts request from clients and dispatches the request to one of the free worker threads for further processing of the request,

- it records the state of the current request and then gets the next message,

- each worker thread works on different request,



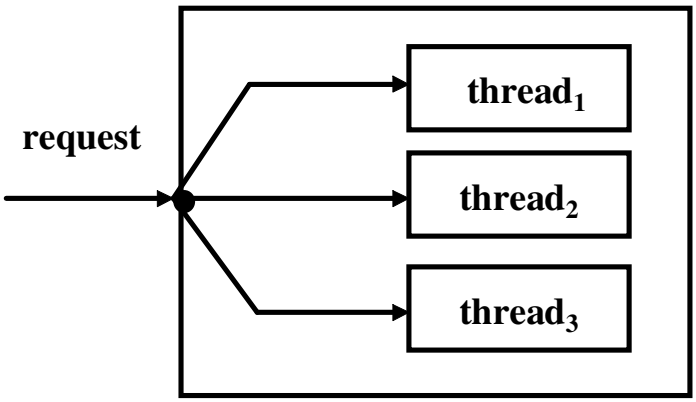
dispatcher-worker model

Team model

- client's requests are not dispatched among threads,

- each thread gets and processes client's request on its own, but threads may service special requests of clients,

- it may have a job queue to store pending requests in order to sort out special requests and distribute them among threads,

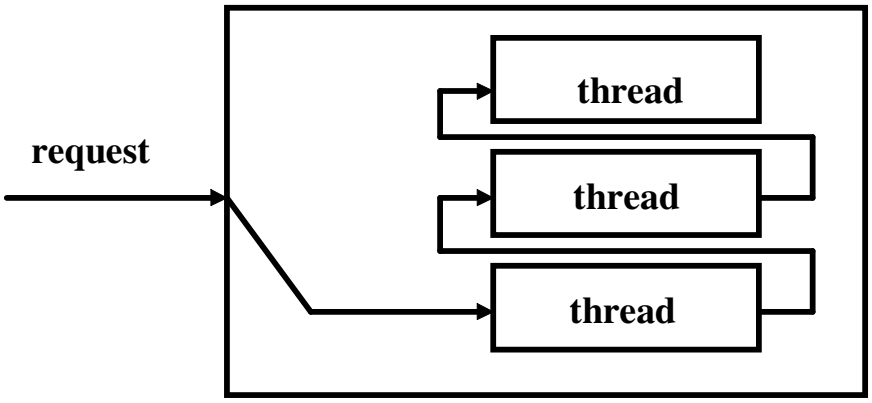


team model

Pipeline model (or producer-consumer model)

- threads are organised as a pipeline so that output of a thread is used as input by an another thread,

- processing of data continue from thread to thread,

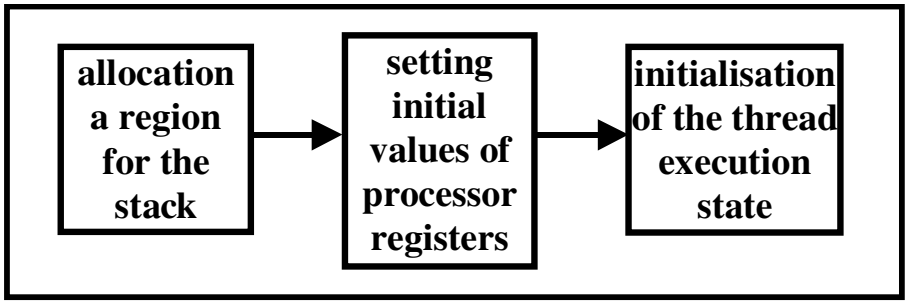


pipeline model

OPERATIONS on THREADS

1. creation of threads

- **to create a new process, its address space has to be created from scratch,**
- **creating a new thread it uses the address space of its process thus it does not needed to be created from scratch,**

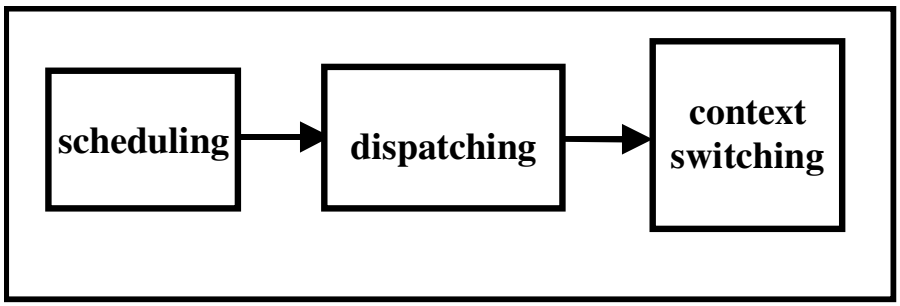


creation of a new thread

- **overheads of creating a new process is greater than those creating a new thread within a process**

2. switching between threads

- **switching between threads sharing the same address is more simple than switching between processes**
- **sharing of address space and OS resources among threads of a process decreases overheads of processor switching among threads comparing to processor switching among processes having their own address space,**



switching between threads

- **switching between threads consists of the following two processes:**

  a. scheduling

  1. non-preemptive scheduling:

  - **a thread runs until it makes a call that causes it to be rescheduled and another thread to be run,**
  - **any section of code that contains a call that might cause a rescheduling is automatically a critical section,**

  2. pre-emptive scheduling

  - **a thread may be suspended to make way for another thread, even when it is runable,**

  b. dispatching

  c. context switching

  - following tasks are performed during the context switching:

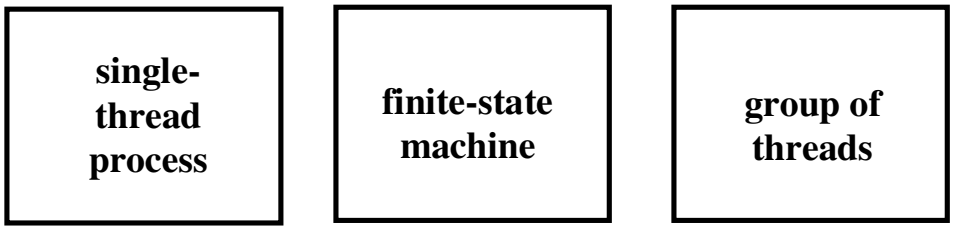
  - **saving the original processor state,**
  - **loading the new state,**
  - **domain transition (transfer to a new protection environment),**

  - context switching, between threads belonging to the same process on a kernel, occurs to and from the kernel,

  - context switching, between threads belonging to different processes, also includes a context switch to a user-level execution environment,

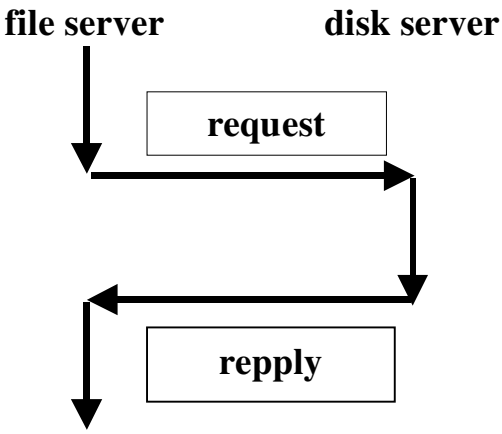
3. creation of server processes using threads

- **threads allow parallelism to be combined with sequential execution and blocking call,**
- **there are several ways of constructing a server process using threads:**



a. ***single-thread process*** **= blocking system calls without any parallelism**

example: - a file server sends a request to a disk server



- the file server blocks until a reply is received from the disk server

advantages: easy to use as a result blocking system calls

disadvantages: low performance

b. ***finite-state machine*** **= non-blocking system calls + parallelism**

example: - server is a single-thread process and operated like a finite-state machine,

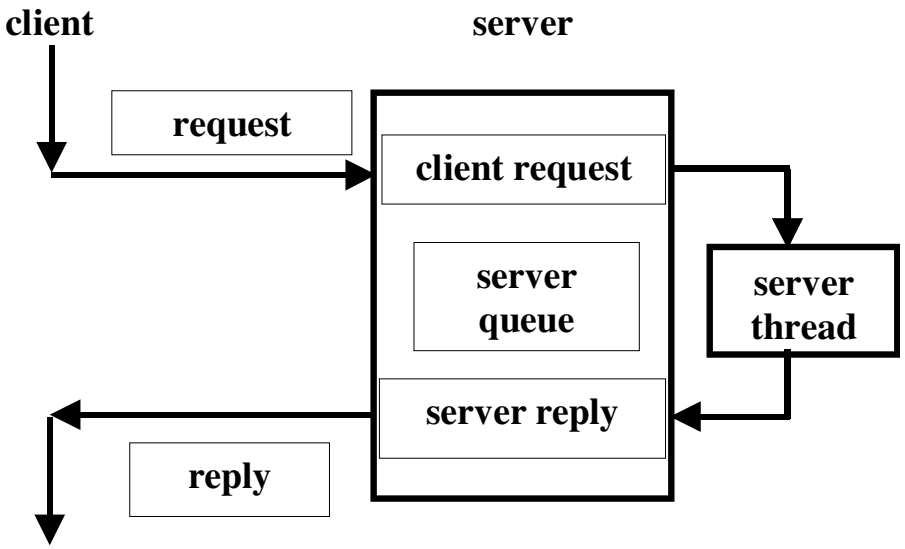
- there is an event queue for client request and server reply messages,

- whenever the server thread becomes idle, it takes the next message from the event queue,

- if it is a client request, a check is made for accession permission and need for server access

- if server access is required to service the request a request

access message is sent,



- there is no blocking, the server records the current state of client's request in a table and then goes to get the next message from the event queue,

- this message may be either a new client request or a reply from a server,

- if it is reply then the state of the client's request is retrieved from the table, and client is processed further,

<u>advantages:</u> high performance through parallelism

<u>disadvantages:</u> hard to program as a result of non-blocking calls

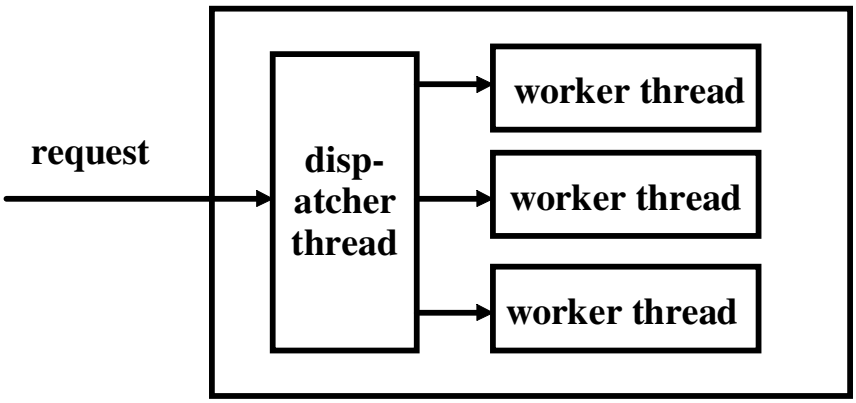
c. *group of threads* = blocking calls + parallelism

<u>example:</u> - there are single dispatcher thread + multiple worker threads,

- the dispatcher thread waits in a loop for requests from the clients,

- when a client request arrives, it checks it for access permission, if permission is allowed, it either creates a new worker thread or chooses an idle worker thread from the pool of threads and hands over the request to the worker thread,

- the control is passed on to one of the worker threads to serve the request,



- the worker thread checks whether a server access is needed for the request,

- if server access is needed, it sends a server access request to the server and the thread is blocked to wait for a reply from the server,

- an another thread will be selected by the scheduler from threads that are in the ready state,

4. resource sharing using threads

- **resource sharing is more simple because all threads of a process share the same address space**

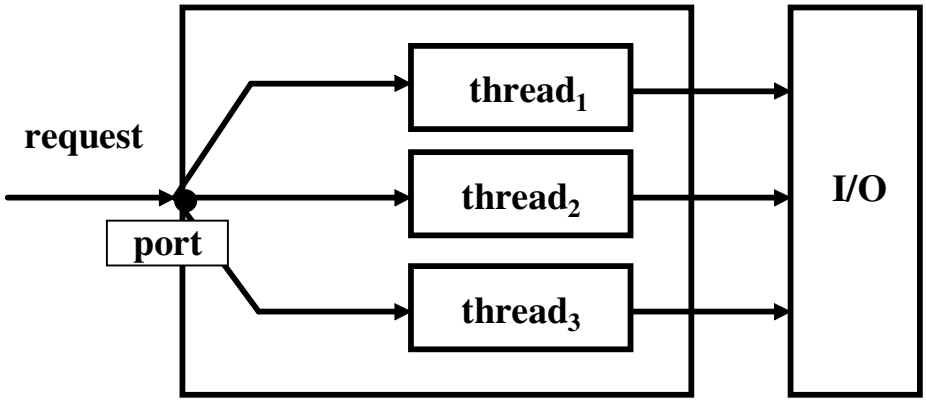
APPLICATION of THREADS in CLIENTS and SERVERS

Server with multiple threads

- server receives request messages from a port,

- it applies the same procedure to process each message,

- when one thread is blocked for in/output an another thread can be scheduled,



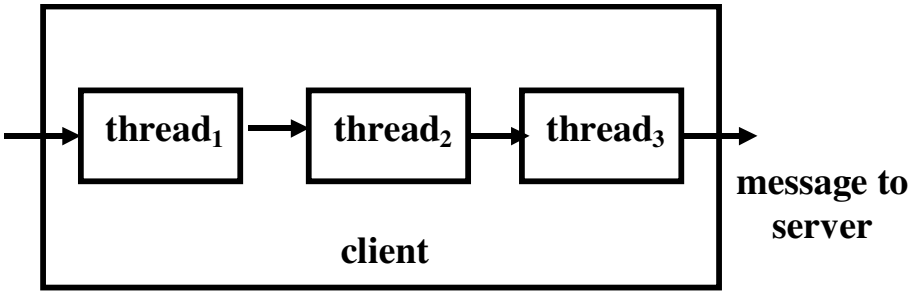
server with multiple threads

Client with multiple threads

- one thread generates a results to be passed to a server by communication procedure but communication blocks the caller - the first thread -

- to overcome this problem an another thread performs the communication and blocks

- the first thread puts its result in a buffer, which is emptied by the second thread and the first thread is able to continue computing



client with multiple threads

THREADS PACKAGE

problems:

- **each thread has a private stack, therefore local variables in procedures are always private to the thread executing the procedure, but local variables of a thread are not protected against access by other threads,**
- **co-operation and co-ordination among threads,**

solution:

THREADS PACKAGE = set of primitives to support application of threads

Services of the threads package

1. thread creation

a. *static creation*: the number of threads remains fixed for entire lifetime of a process and each thread has a fixed stack,

b. *dynamic creation*: a process is started with a single thread, new threads are created when needed during the execution of the process,

2. thread termination

- **there are two methods to terminate threads:**
  - **a thread may destroy itself when it finishes its job by making an exit call,**
  - **a thread may be killed by using the kill command from outside**

3. thread synchronisation

- **problem: multiple threads should be prevented from trying to access the same data simultaneously,**
- **solution: critical region + mutual exclusion**

a. *mutex variables* - binary semaphore with locked and unlocked states, which is used for short-term locking,

- **operations on mutex variables:**

  - <u>lock:</u> unlock → lock state change to disable access to the critical region,

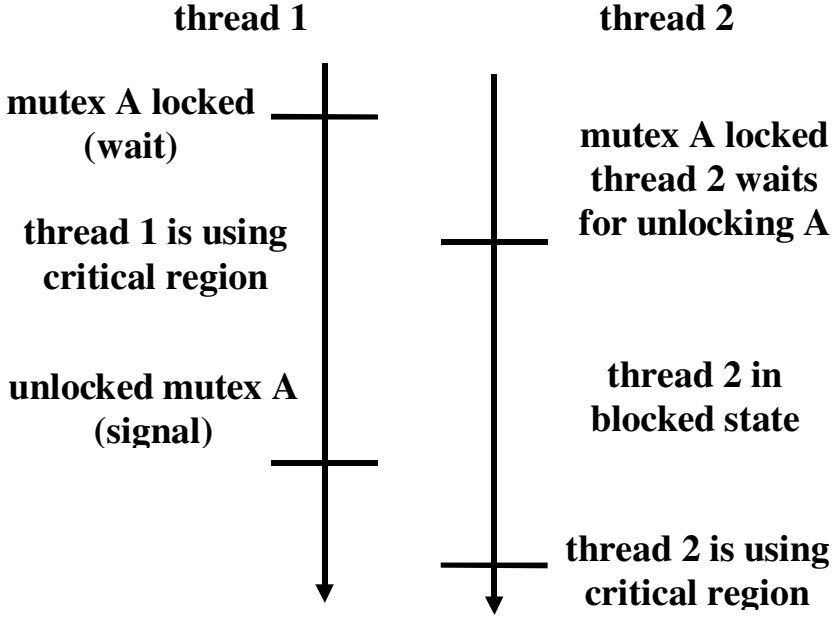
  - <u>unlock:</u> lock → unlock state change to enable access to critical region after finishing a particular execution,

b. *condition variables* - mutex variable + its Boolean state, used for long-term locking,

- **operations on condition variables:**

  - <u>wait:</u> unlock → lock state change and the thread is blocked until a signal operation is performed by some other thread on the condition variable,

  - <u>signal:</u> unlock → lock state change the thread that was blocked waiting for the condition variable starts executing in the critical region



4. thread scheduling

- **threads package supports the following scheduling features:**

a. priority assignment

- **to assign priorities to threads in order to ensure that important threads run frequently than others,**

b. dynamic scheduling

- **the size of time slots may be changed taking into account the total number of threads in the system,**

c. hand-off scheduling

- **to name successor of a thread in order to bypass the queue of runnable threads and directly switch the processor to the thread given by the currently running thread,**

d. affinity scheduling

- **a thread is scheduled on the processor it last ran supposing that part of its address space is still in that processor cache,**

5. exception and interrupt handling

- **threads package supports the following exception and interrupt handling features:**

a. exception & interrupt signal must be handled properly

- **there is a separate exception handler thread in each process that handles all exception conditions occurring in any thread of the process,**

- **when a thread receives a signal for an exception condition, it sends an exception occurrence message to the exception handler thread and waits until the exception is handled,**

b. exception & interrupt signal must be prevented from getting lost

- **each thread has its own private global variables for signalling exception conditions, so that conflicts between threads over the of such global variables never occur,**

Thread calls in a threads package

1. thread_fork(fun, arg)
- creates a new thread that executes the given function "fun" with "arg" and returns a thread identifier "tid",

2. thread_exit(res)
- terminates the current thread and returns a result "res",

3. thread_block(tid)
- a thread which created a new thread with "tid" can block until the last thread terminates,

4. thread_yield(tid)
- allows another thread, given by "tid", to run

5. thread_sdat(tid, data)
- assigns "data" to a thread given by "tid"

6. thread_rdat(tid)
- returns the data associated with a thread given by "tid"

7. mutex_lock(mid)
- blocks the thread until the mutex "mid" has been unlocked, then locks mutex

8. mutex_unlock(mid)
- unlocks the mutex "mid"

9. cond_wait(cid, mid)
- unlocks the mutex "mid" and blocks the thread, i.e. queues it on the condition variable "cid"

10. cond_sign(cid)
- wakes a thread if it is blocked on the condition variable "cid"

IMPLEMENTATION of THREAD PACKAGE

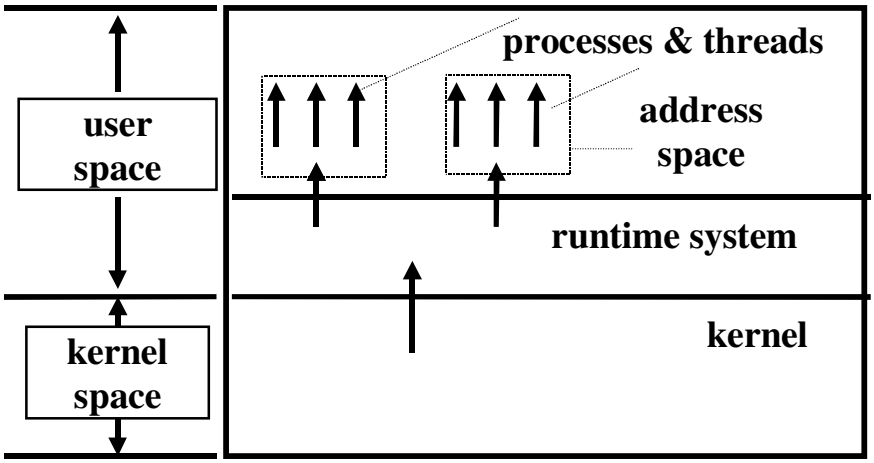
- key implementation issues of threads package are the following:

- **handling blocking system calls from threads,**
- **scheduling threads for execution in a process,**

User level threads package

- the user space consists of run-time systems that is a collection of threads management routines,

- threads run in the user space on top of the run-time system which is a set of procedures that manages threads,



user level threads package

- the kernel knows nothing about threads and manages them as single-thread processes,

- all calls are implemented as calls to the run-time system procedures that perform the functions corresponding to the calls, the run-time system maintains a status information table of threads,

- a blocking system call from a running thread is not trapped by the OS but is routed to the run-time procedure,

- when a thread performs a system call, it calls a run-time system procedure, this procedure checks whether the running thread must be

suspended, if so a thread is suspended and thread switching is executed,

- in this case only the thread is blocked, it is put into a queue maintained by a run-time support, and the execution of the process continues with other threads,

- two-level scheduling is performed; the scheduler in the kernel allocates processes, and the scheduler of the run-time system manages threads,

advantages of user level approach:

- **thread package can be implemented on top of an existing OS that does not support threads,**
- **the most suitable scheduling policy may be selected because it runs with its own run-time library,**
- **switching the context from one thread to another is faster in the user level approach because it is performed by the run-time system and there is no call to the kernel,**

disadvantages of user level approach:

- **difficult to manage time slicing, when the timer interrupt occurs because control returns to a procedure in the thread library which can decide whether the thread is out of time,**
- **blocking system calls are not allowed directly because all threads of the process will be stopped and the kernel will schedule another process to run, but there is a solution to this problem:**

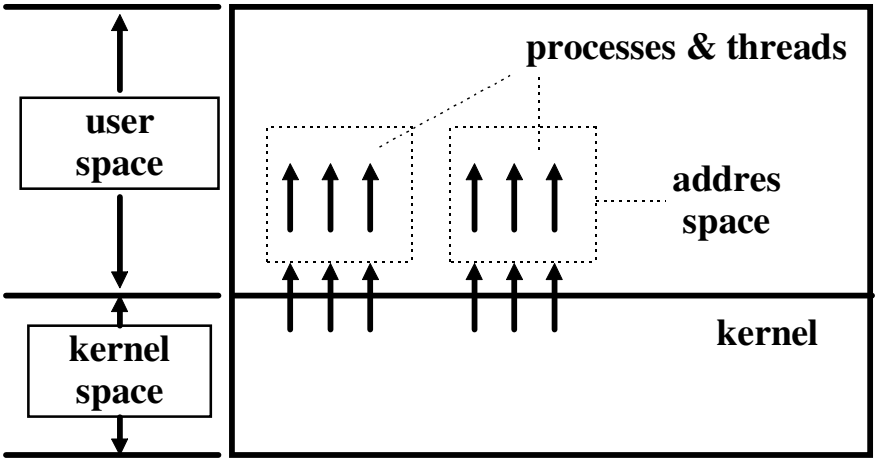
  a. if the system call returns the result, it can be passed to the calling thread which may continue,

  b. blocking systems calls may be replaced by non-blocking system calls, if the call returns with a suspended state, the calling thread is blocked and another thread is scheduled,

- **when the non-blocking call completes the calling thread may be runnable again and the results made available to it in the usual way,**
- **threads within a process cannot take advantage of a multiprocessor architecture,**

Kernel level threads package

- no run-time system is used and the threads are managed by the kernel, thus status information table of threads is maintained within the kernel, where the kernel has one table per process with one entry per thread,



kernel level threads package

- kernel thread is created for each user-level thread and all scheduling is carried out by the kernel,

- all calls that might block a thread are implemented as system calls that trap to the kernel, i.e. blocking and non-blocking of threads is carried by the kernel,

- a thread issuing a system call can be blocked without blocking all other threads in the same process, and each thread can compete for processor,

- when a thread blocks, the kernel either selects another thread from the same process or from a different process to be run,

- to decrease costs of creating and destroying of threads some systems recycle their threads,

advantages of kernel level approach

- **the kernel can keep track of the amount of processor time consumed by a thread, when a thread finishes its time-slot, it can be interrupted by the kernel,**

- **blocking calls are used by this approach because after a blocking call, it traps to the kernel, where it is suspended and the kernel starts a new**

thread,

disadvantages of kernel level approach

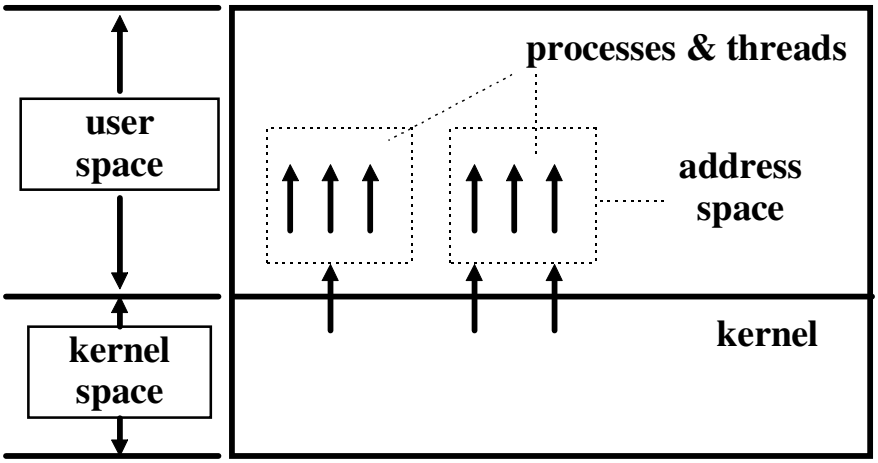
- **the concepts of threads must be incorporated in the design of the kernel of OS,**
- **context switching overhead is greater than at the user-level approach because switching is carried out by the kernel,**
- **users can select scheduling algorithm and assign priorities to threads through the system call parameters as a result of the single level schedule,**
- **poor scalability because the status information table for threads is maintained in the kernel,**

Kernel-user level thread package

- combination of kernel-level and user-level approach to achieve flexibility and maximum potential performance,

- user threads are assigned to kernel threads taking into account:

- **how many processors are available, and which threads should run parallel on separate processors,**
- **which threads will make blocking calls,**



kernel-user level thread package

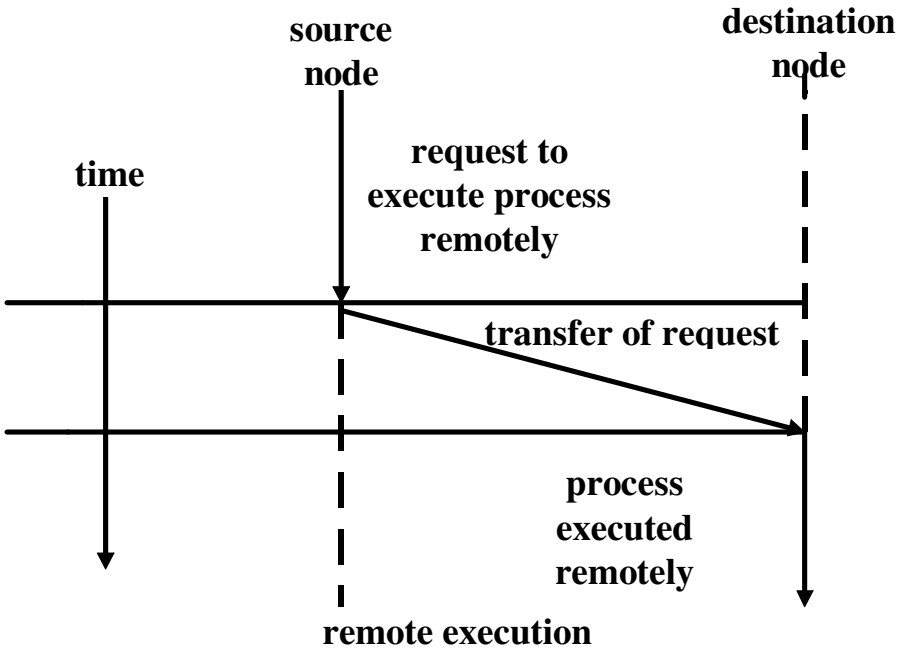
REMOTE EXECUTION

REMOTE EXECUTION = transfer and execution a process on a remote host

- every computer in the DS has a process server that are responsible for maintaining the load information, negotiating a remote host, invoking remote operations and for linking clients and servers, each process server can maintain a list of registered hosts that are ready to host processes,

- remote execution may be initiated by a process or by the local process server,

- after selecting the remote host, a negotiation process begins when the client process server indicates the resource requirements to the process server at the remote site



- if the client is authenticated and its requirements can be met, the server grants permission for remote execution,

- next, code of the process is transferred and the server creates the remote process and initialises the process at the remote site,

- remote execution does not allow to move a process once created i.e. a process should be completed where it was created,

REQUIREMENTS of REMOTE EXECUTION

1. information propagation

- **there should be mechanism for propagating information on the availability of idle computers, or for identifying such computers in the DS,**

2. pre-emption

- **remote use of a computer must be able to be stopped when the owner of the computer claims it,**

- **the pre-emption prevents degradation of performance when remote process is executed on the computer,**

3. location independence

- **remote processes should be performed just as local processes**

4. load sharing

- **computation load should be distributed in the DS using load balancing algorithms**

5. transparency

- **execution is transparent if programs need not be written with special provisions for executing remotely,**

6. system heterogeneity

- **if a program is to be invoked in a heterogeneous host, it is necessary to recompile the program**

- **to the compiling overhead a canonical machine independent intermediate language is used for program execution for example JAVA,**

REMOTE EXECUTION POLICIES and MECHANISMS

-remote execution is a load sharing and balancing method, that could be divided into:

-remote execution can divided into two main phases:

a. *remote execution policy* - selection of a process to execute remotely and determination of when and where to it is transferred and initiated, and to transfer the process to the destination computer,

- **this policy is supported by collecting necessary statistics using local databases and making the corresponding decisions,**

b. *remote execution mechanism* - mechanism to initiate the selected processes on the destination computer,

Activities of the remote execution policy

- the remote execution policy could be divided into the following phases:

a. transfer policy - to determine whether to execute a process locally or remotely,

b. selection policy - to determine which process should be transferred,

c. location policy - to determine to which computer a selected process should be transferred

- the remote execution policy determines data to be collected and the frequency of this operation,

- having this data the remote execution policy also deals with the definition of which process to transfer, when and to where to transfer,

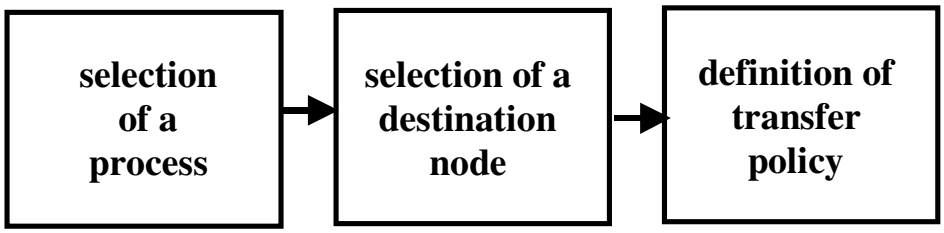
collecting data - the following data is collected to make the required decisions:

1. *computer load*- number of processes and communication links, whether currently active or not, the average CPU load, and the average network load,

2. *process resource usage* - the average and total CPU and network utilisation of each process, and its total communication to total and remote processes,

3. *communication data* - the total number of packets sent and received over each link for the most active links per process,

<u>policy execution</u> - there are three activities performed to make the corresponding decisions:



1. selection of a process that should be executed remotely

- **"which" process to transfer**

2. selection of a destination node - execution site -

- **there are two approaches to select the execution site:**

a. using information about idle computers, servers and requests for a server by a client

- this information is stored by each computer in a locally maintained database,

- when a server decides to make itself available, it broadcasts a message to all the clients, all clients register the availability of the server,

- when a server decides to withdraw itself, it broadcasts a message to all the clients, a client, after receiving the withdrawal message, removes the server from its available server lists,

b. using information provided by resource managers

- each computer has a network server process, when a computer is a server, a remote execution dispatcher process is created at this node,

- the remote execution dispatcher process contains information about the load of the corresponding server,

3. definition of transfer policy

- **"when" and "how" to transfer the selected process from the source computer to the destination computer,**

Activities of remote execution mechanism

- a process is moved to the destination site after selection the new execution site,

- next step is the initialisation of the new process's environment, it requires sending a request to the local program manager to create a new address space by the "invoker" process,

- the program manager uses the kernel server to set up the address space and create an initial process,

- the new process waits for a reply from the program manager and it turns over control to the "invoker" process by forwarding the new process to it,

- next, the "invoker" process initialises the new program space with program arguments, default I/O and environment variables,

- finally, it starts execution of the new process,

REMOTE EXECUTION in HETEROGENEOUS SYSTEMS

requirements:

- **variety of resources requires a remote execution facility which can accommodate the different environments and also deals with the problems of any homogenous environment,**

problems:

- **it is impossible to assume the existence of global naming**

solution: attributed names,

- **binary translation of data as it moves from one architecture to another**

problems:

- **service description as a result of differences in syntax and semantics among systems,**

solution: translation among systems

ADVANTAGES of REMOTE EXECUTION

1. reducing average response time of processes

- **remote execution may reduce the average response time of the processes of overloaded nodes,**

2. speeding up individual tasks

- **transferring a process to a faster or idle processor,**

3. gaining higher throughput

- **capabilities of processors are better used as a result of load balancing among processors,**

4. utilising resources effectively

- **processes can be transferred to the most suitable node to utilise the system resources in the most efficient manner,**

5. reducing communication load

- **transferring a process closer to the resources, it requires, may reduce communication load,**

6. improving system reliability

- **transfer of a process to a node whose reliability is higher,**

7. improving system security

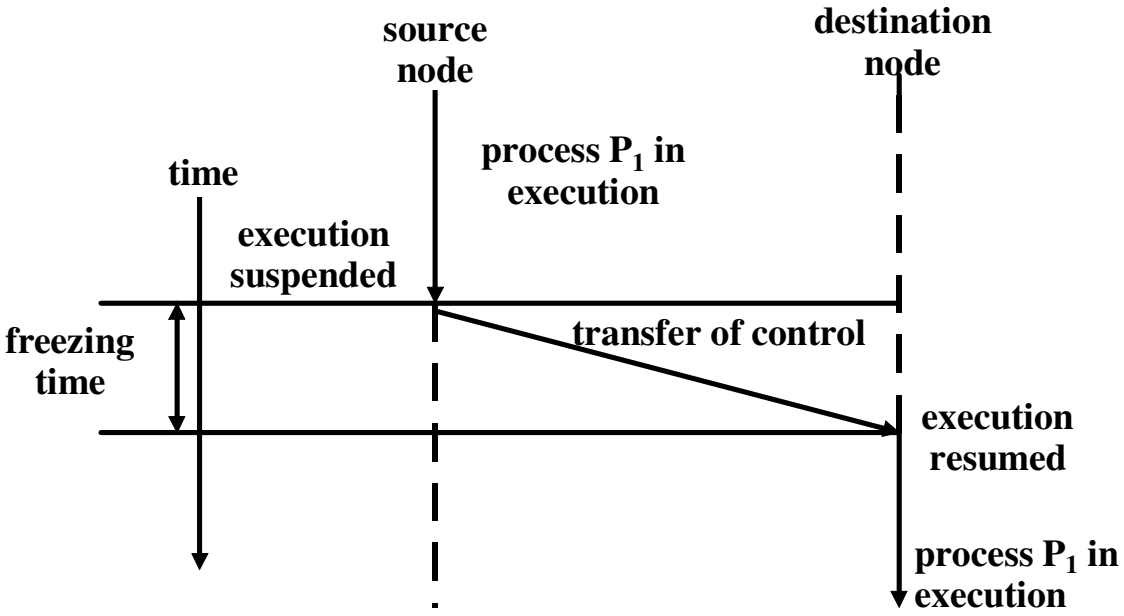
- **a process may be transferred and run on a node that is not directly accessible to general users,**

PROCESS MIGRATION

PROCESS MIGRATION = relocation of a process from its current node to another node

- process migration moves a process during its execution, so that it continues on another processor, with continuous access to all its resources,

- it may be initiated without the knowledge of the running process or any other process interacting with it,



process migration

- similar to remote execution, a process migration needs to locate and negotiate a remote host, transfer the cod image, and initialise the remote operation,

- since the process transferred to the destination node is pre-empted, its state information must be also transferred,

- the state information, which should be transferred, consists of two parts: computation state and communication state,

FEATURES of PROCESS MIGRATION

1. transparency

- **levels of transparency:**

  a. object access level - supports non-preemptive process migration

  - allows free initiation of programs at an arbitrary computer,

  - provides access to objects in location independent manner,

  b. system call & interprocess communication level - supports pre-emptive process migration

  - all system call and interprocess communication should be location independent in order not to depend upon its originating node after being migrated,

2. minimal interference

- **migration should cause minimal interference to the progress of the process and the whole system,**

- **it is achieved by minimising the time of the process being migrated - freezing time -,**

3. minimal residual dependencies

- **migrated process should not any way continue to depend on its previous node once it has started executing on its new node,**

4. efficiency

- **main sources of inefficiency in process migration:**

  - **the time required for migrating a process,**

  - **the cost of locating an object,**

  - **the cost of supporting remote execution once the process is migrated,**

5. robustness

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

PROCESS MIGRATION POLICIES and MECHANISMS

- process migration can be divided into two main phases:

a. *process migration policy* - selection of a process to migrate and determination of when and where to it is migrated,

- **collecting necessary statistics and making decision relating to process migration, and**

b. *process migration mechanism* - mechanism to migrate the selected processes to a destination computer,

- **performing process transfer taking into account decisions made by process migration policy,**

Activities of process migration mechanism

- information transferred during migration is the process's state and the process's address space:

- **process's state: - execution state of the process**

  - memory being used by the process

  - I/O states

  - process's identifier

  - process's user & group identifiers

  - list of objects to which the process has a right to access

  - list of files opened by the process

process's address space: code, data and stack of the program

- the size of the address space is larger than the size of the process's state information,

- transferring of the process's state requires complete stopping the execution of the migrant process, but in some systems it is possible to transfer process's address space without stopping its execution

- there are four activities performed by the process migration policy:

1. <u>blocking, completing fast I/O operations and freezing the process</u> on its source node

***blocking* = before process can be frozen, its execution must be blocked,**

- **blocking may be immediately or be delayed until the process reaches a state when it can be blocked,**

- **after blocking the next step is to wait for the completion of all fast I/O operations associated with the process, but there is no waiting for slow I/O operations that are continued after the process starts executing on its destination node,**

***completing fast I/O operation* = the process is frozen after the completion of all fast I/O operations**

- **files opened by the process are included in the process's state information for example by pointers to the files,**

- **these pointers should be preserved so that the migrated process could continue to access to it,**

***freezing* = after blocking the process and complementing fast I/O operations the state of the process is saved and all external interactions with the process are deferred,**

- **to determine the state of a process is complicated because it is defined by the contents of the**

  a. process address space,

  b. local message queues, and

  c. state of communication with remote processes

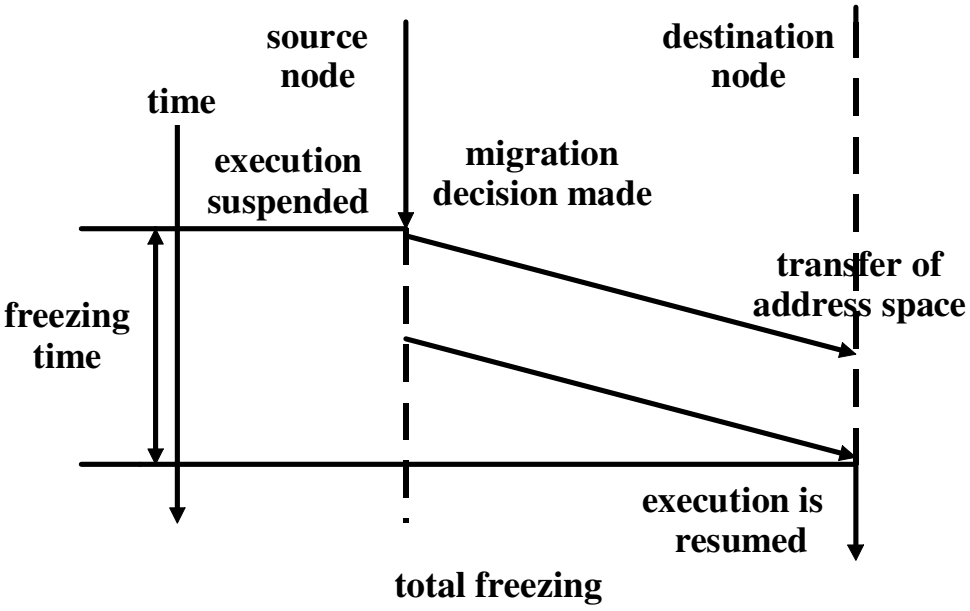
2. <u>transferring the process's address space and state</u> from its source node to the destination node

- **it makes the new location of the process transparent to other processes performing the transition without affecting the behaviour of the migrating process or other processes,**

- **address space transfer methods:**

a. total freezing

- **a process execution is stopped while its address space is being transferred**

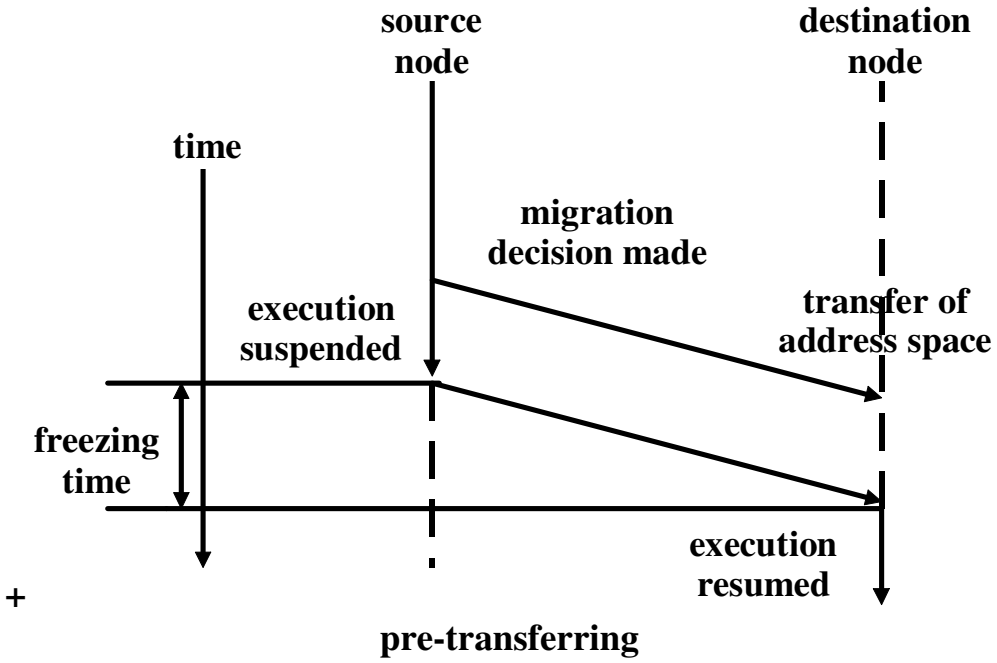


b. pre-transferring

- **firstly, an initial transfer of the complete address space is done while the process is still running on the source node,**

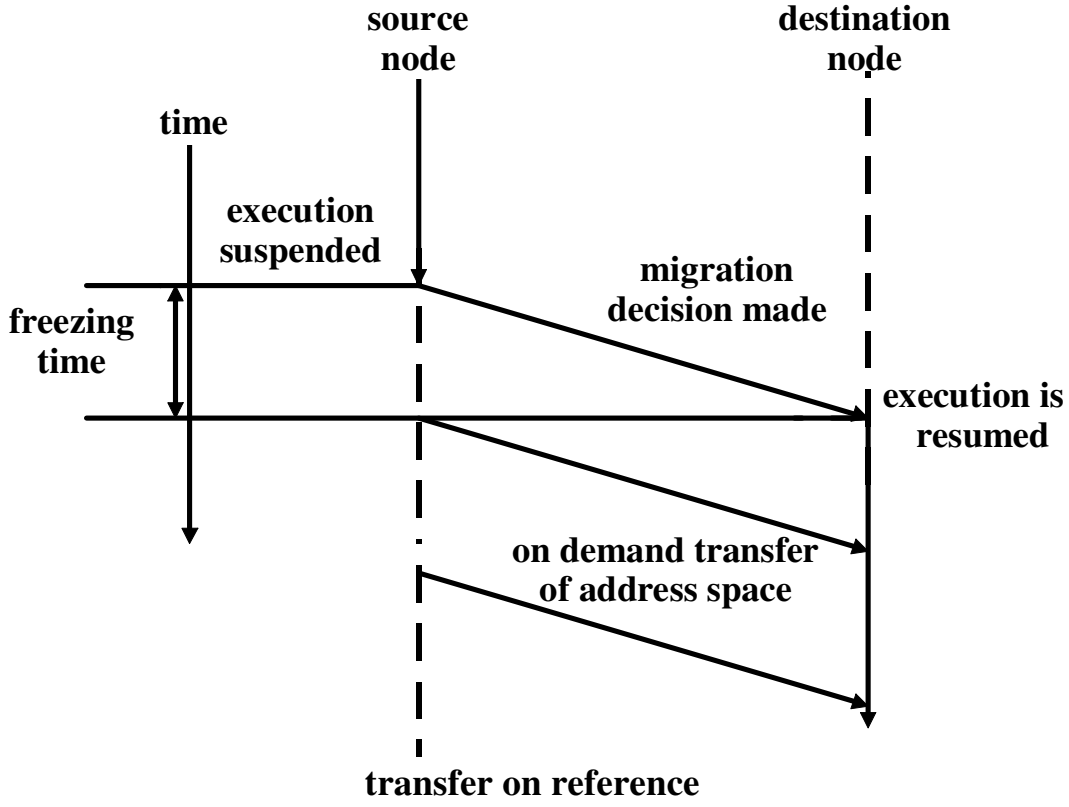
- **secondly, the process is frozen for transferring the modified state of the process and those pages that were modified during the last transfer,**

- **thirdly, those pages of the address space are moved that were modified during the first transfer,**



c. transfer on reference

- **the process's address space left behind on its source node, and as the relocated process executes on its destination space, attempts to reference memory pages results in the generation of request to copy in the desired blocks from their remote locations,**



3. restarting a process on its destination node

- ***restart* = a process does not begin execution on the destination node until all its address space and state is transferred,**

  - **an empty process state is created during the process migration on the destination node,**

  - **two copies; the old and the new copy exist and are accessible during the process migration,**

  - **the state of the migrating process is transferred from the old copy to the new one,**

  - **after copying all the state of migrating process the new copy is unfrozen and the old copy is deleted,**

  - **next, the migrated process is restarted on its destination node in whatever state it was in before,**

problem: **to define time instant when a migrating process can begin its execution on a destination node,**

4. handling communication between processes that have been separated by the process migration

- **there are special links at the source processor to transfer all messages between the migrated processes and all other processes,**

- **there are several approaches to handle problems associated with the communication state:**

a. message forwarding mechanisms

- **all pending, en-route and future messages should arrive at the process's new location,**

- **when a migrating process is suspended on the source node, all known names and locations of remote processes with which a migrating process communicates are saved in the process state with the destination node address,**

- **the process is moved to the destination node but processes communicating with the migrating process are not informed about the migration,**

- **processes communicating with the migrating process will still send messages to the source node,**

- **thus, when a process is suspended and migrating all messages addressed to it are saved,**

- **after the process is resumed on the destination node, these buffered messages are forwarded to its new location,**

- **types of messages to be forwarded to the new location:**

  message No. 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 started on its destination node,

  message No. 2. messages received at the source node after the process's execution has started on its destination node,

  message No. 3. messages that are sent to the migrant process from any other node after it has started executing on the destination node,

- **message forwarding (redirection) methods:**

I. resending the message

- **messages of No. 1. and No. 2 are returned to the sender as not deliverable or simply dropped assuming that the sender of the message is storing a copy of the data and is prepared to retransmit it,**

- **the sender is prompted to resend messages to process's new node,**

- **this approach is non-transparent to the processes interacting with the migrant process,**

II. origin node

- **each node keeps information about the current locations of all the processes created on it,**
- **messages for a process are always sent to the node where the process was created,**
- **the origin node forwards the message to the process's current node,**

III. link traversal

- **message queue is for messages No. 1. is for the migrant process on its source node and all messages are placed in this message queue,**
- **messages No. 1. in the queue are sent to the current node as part of the migration procedure,**
- **to redirect messages No. 2. and No. 3. a link is left at the source code pointing to the destination node of the migrant process,**
- **the link contains two components; the first one is the identifier of the node where the process was created, the second one is the identifier of the last known node of the process,**
- **forwarding a message the location of a migrant process is located through series of links,**

IV. link update

- **during migration the source node sends link-update messages to the kernel controlling communication of migrant processes,**
- **these link-update messages contain the new address of each link held by the migrant process,**
- **messages sent to the migrant process on any of its links will be directly forwarded to the migrant process's current node,**

b. message loss prevention

- **a migration facility on the source node informs processes with which a migrating process communicates about the intent of a moving a process - this takes place before the migration -,**
- **when a process is migrating messages are buffered,**
- **after the process resumes, a migration facility on the destination node resumes communication with migration facilities on other nodes,, the new address is passed,**

c. message loss recovery

- **a migration facility does not save the destination node address,**
- **processes with which a migrating process communicates are not informed about the intent to move a process,**
- **all messages in transit during the migration are ignored, i.e. they are lost,**
- **however, the addresses and names of remote processes with which a migrating process communicates are saved,**
- **after resuming process on the destination node, the process establishes links with remote processes cooperating with it before migration and message recovery is initiated,**

Implementation of process migration

- there are three different level of implementation:

- **1. the operating system kernel**
- **2. outside the kernel,**
- **3. embedded in a compiler and the runtime package.**

PROCESS MIGRATION in HETEROGENEOUS SYSTEMS

problem:

- **- different data representation at different nodes,**

solution:

- **- in these systems all data must be translated from the source processor format to the destination processor format (external data representation),**
- **- a standard representation is used for the transport of data,**
- **- each processor should convert data to and from the standard format,**

ADVANTAGES of PROCESS MIGRATION

1. reducing average response time of processes

- **- process migration may reduce the average response time of the processes of overloaded nodes,**

2. speeding up individual tasks

- **- migrating processes to different nodes and executing them concurrently,**
- **- migrating a process to a faster processor,**

3. gaining higher throughput

- **- capabilities of processors are better used as a result of load balancing among processors,**

4. utilising resources effectively

- **- processes can be migrated to the most suitable node to utilise the system resources in the most efficient manner,**
- **- there are some resources that cannot be accessed remotely, process migration facilitates the use of such resources,**

5. reducing communication load

- **- migrating a process closer to the resources, it requires, may reduce communication load,**

6. improving system reliability

- **- migration of a process to a node whose reliability is higher,**
- **- migration of a copy of a critical process to some other node and to execute both the original and copied processes concurrently on different nodes,**
- **- after a fault in a node a process, running on it, can be migrated to an**

another node to continue the execution before the node completely fails,

7. improving system security

- **- a process may be migrated and run on a node that is not directly accessible to general users,**