Unit 3
Processes

Contents

- Threads
- Virtualization
- Clients
- Servers
- Code Migration

Introduction to Threads

- Process
  A software processor in whose context one or more threads may be executed. Executing a thread, means executing a series of instructions in the context of that thread.
  A process can be thought of as a program in execution

- Thread
  A minimal software processor in whose context a series of instructions can be executed. Saving a thread context implies stopping the current execution and saving all the data needed to continue the execution at a later stage
  A thread is the unit of execution within a process

Threads

- Thread is a light weight process created by a process
- Thread is a single sequence of execution within a process
- Thread has it own
  - Program counter that keeps track of which instruction to execute next
  - System registers which hold its current working variables
  - Stack which contains the execution history
- Processes are generally used to execute large, ‘heavyweight’ jobs such as working in word, while threads are used to carry out smaller or ‘lightweight’ jobs such as auto saving a word document
- A thread shares few information with its peer threads (having same input) like code segment, data segment and open files.

Lifecycle of a thread

There are various stages in the lifecycle of a thread. They are:

- New
  The lifecycle of a new thread starts in this state. It remains in this state till a program starts
- Runnable
  A thread becomes runnable after it starts. It is considered to be executing the task given to it.
- Waiting
  While waiting for another thread to perform a task, the currently running thread goes into the waiting state and then transitions back again after receiving a signal from the other thread
- Timed Waiting: A runnable thread enters into this state for a specific time interval and then transitions back when the time interval expires or the event the thread was waiting for occurs
- Terminated
  A thread enters into this state after completing its task.

Process & Thread

Similarities between Process & Thread

- Like processes, threads share CPU and only one thread is running at a time
- Like processes, threads within a process execute sequentially
- Like a traditional process, a thread can be in any one of several states: running, blocked, ready or terminated
- Like process, threads have Program Counter, Stack, Registers and State

Process Vs. Thread

Process	Thread
Process means a program is in execution	Thread means a segment of a process
The process is not Lightweight	Threads are Lightweight
The process takes more time to terminate	The thread takes less time to terminate
It takes more time for creation	It takes less time for creation
Individual processes are independent of each other	Threads are parts of a process and so are dependent
All the different processes are treated separately by the operating system	All user level peer threads are treated as a single task by the operating system
Communication between processes requires more time than between threads	Communication between threads requires less time than between processes

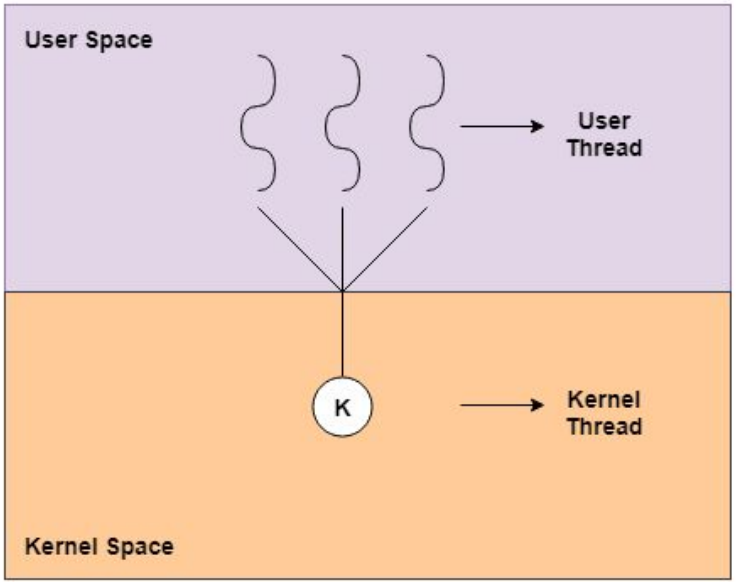
Types of Threads

There are different types of threads. They are:

- Kernel Level Thread
  This approach is to have the kernel be aware of threads and schedule them

- User Level Thread
  This approach is to construct a thread library that is executed entirely in user mode

Threads



User-Level Thread

- User-level threads are implemented by users and the kernel are not aware of the existence of the threads. It handles them as if they were single-threaded processes
- User-level threads are small and much faster than kernel level threads.
- They are represented by a program counter (PC), stack, registers and a small process control block
- There is no kernel involvement in synchronization for user-level threads

User-Level Thread

Advantages:

- User-level threads are easier and faster to create than kernel-level threads. They can also be more easily managed
- User-level threads can be run on any operating system
- There are no kernel mode privileges required for thread switching in user-level threads

User-Level Thread

Disadvantages:

- Multithreaded application in user-level threads cannot use multiprocessing to their advantages
- The entire process is blocked if one user-level thread performs blocking operation.

Kernel-Level Thread

- Kernel-level threads are handled by operating system directly and the thread management is done by the kernel.
- The context information for the process as well as the process threads is all managed by the kernel.
- Because of this, kernel-level threads are slower than the user-level threads.

Kernel-Level Thread

Advantages:

- Multiple threads of the same process can be scheduled on different processors in kernel-level threads
- The kernel routines can also be multithreaded
- If a kernel-level thread is blocked, another thread of the same process can be scheduled by the kernel

Kernel-Level Thread

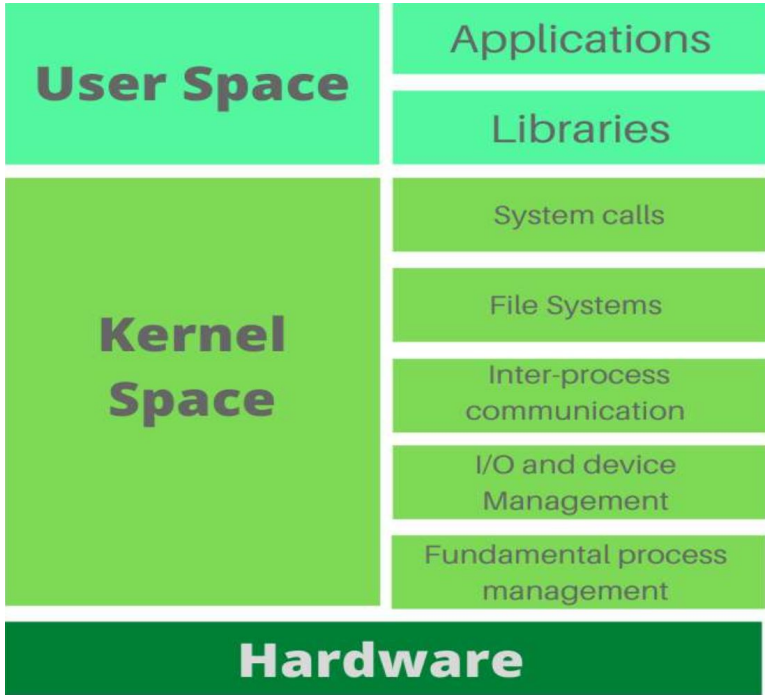
Disadvantages:

- A mode switch to kernel mode is required to transfer control from one thread to another in a process
- Kernel-level threads are slower to create as well as manage as compared to user-level threads.

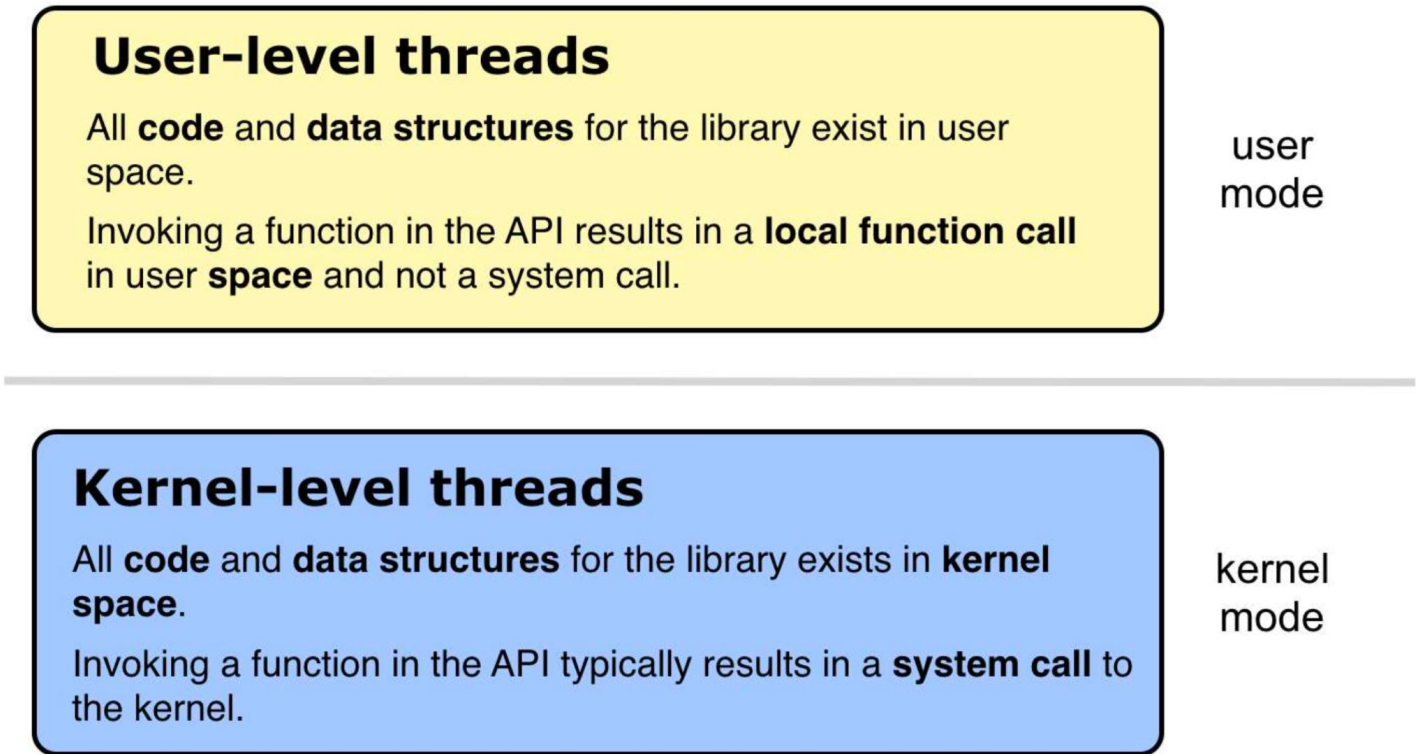
Kernel Mode vs User Mode

- **Kernel Space** – Executing code has unrestricted access to any of the memory address space and to any underlying hardware. It is reserved for the highest of trusted functions within a system. Kernel mode is generally reserved for the lowest- level, most trusted functions of the operating system. Due to the amount of access the kernel had, any instability within the kernels executing code can result in complete system failure.

- **User Space** – Executing code has limited access. API calls are used to the kernel to request memory and physical hardware access. Because of the restricted access, malfunctions within user mode are limited only to the system space they are operating within.



User-level and Kernel-level threads



Kernel Mode vs User Mode

User Level Thread	Kernel Level Thread
1. User level thread are managed by user level library	1. Kernel level thread are managed by OS
2. User level threads are typically faster	2. Kernel level thread are slower than user level
3. Context switching is faster	3. Context switching is slower
4. If once user level thread perform blocking operation then entire process gets blocked	4. If one Kernel level thread blocked, there is no effect on the others

Multithreading

- Multithreading is a phenomenon of executing multiple threads at the same time.
- As we have two types of thread i.e. **user-level thread** and **kernel-level thread**. So, for these threads to function together there must exist a relationship between them. This relation is established by using **Multithreading Models**.
- There are **three** common ways of establishing this relationship.
  1. Many-to-One Model
  2. One-to-One Model
  3. Many-to-Many Model

Threads in Distributed System

- Example: A spreadsheet program where the different cells are dependent, when a user changes the value in a single cell, such a modification can trigger a large series of computations:
  - If there is only a single thread of control, computation cannot proceed while the program is waiting for input. Likewise, it is not easy to provide input while dependencies are being calculated.
  - The easy solution is to have at least two threads of control: one for handling interaction with the user and one for updating the spreadsheet. In the meantime, a third thread could be used for backing up the spreadsheet to disk while the other two are doing their work.
- Used to express communication in the form of multiple logical connections at the same time.
- An important property of threads is that they can provide a convenient means of allowing blocking system calls without blocking the entire process in which the thread is running.
- A main contribution of threads in distributed systems is that they allow clients and servers to be constructed such that communication and local processing can overlap, resulting in a high level of performance.
- Attractive to use in distributed systems:
  1. Multithreaded client
  2. Multithreaded server

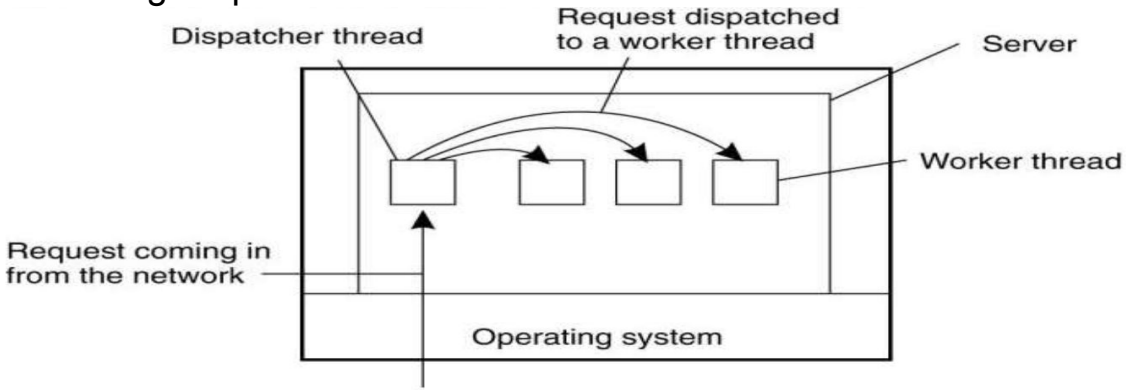
Multithreaded Client

- Can be used to hide delays/latencies in network communications, by initiating communication and immediately proceeding with something else.
- Example: web browsers such as IE are multi-threaded . A web browser can start up several threads: once the main HTML file has been fetched, separate threads can be activated to take care of fetching the other parts. Each thread sets up a separate connection to the server and pulls in the data. One for downloading the HTML source of the page, one each for images on the page, one each for animations/applets etc.
- Replicated web servers along with multi-threaded clients can result in shorter download times.

- Servers can be constructed in three ways:
  1. Single Threaded Process:
     - It gets a request, examines it, carries it out to completion before getting the next request.
     - The server is idle while waiting for disk read; i.e. system calls are blocking.
  2. Threads (Multithreaded Server)
  3. Finite State Machine:
     - If threads are not available, it gets a request ,examines it, tries to fulfill the request from cache , else sends a request to the file system; but instead of blocking it records the state of the current request and proceeds to the next request.

Multithreaded Server

- Multithreading not only simplifies server code considerably, but also makes it much easier to develop servers that exploit parallelism to attain high performance, even on uniprocessor systems. However, now that multiprocessor computers are widely available as general-purpose workstations, multithreading for parallelism is even more useful.



- Dispatcher Worker model also called as Boss worker model. In this model there is a server process which contain one dispatcher or boss thread and multiple worker threads. Dispatcher thread continuously check the status of the worker threads and assign job to the thread which is free or idle.

Multithreaded Server Model

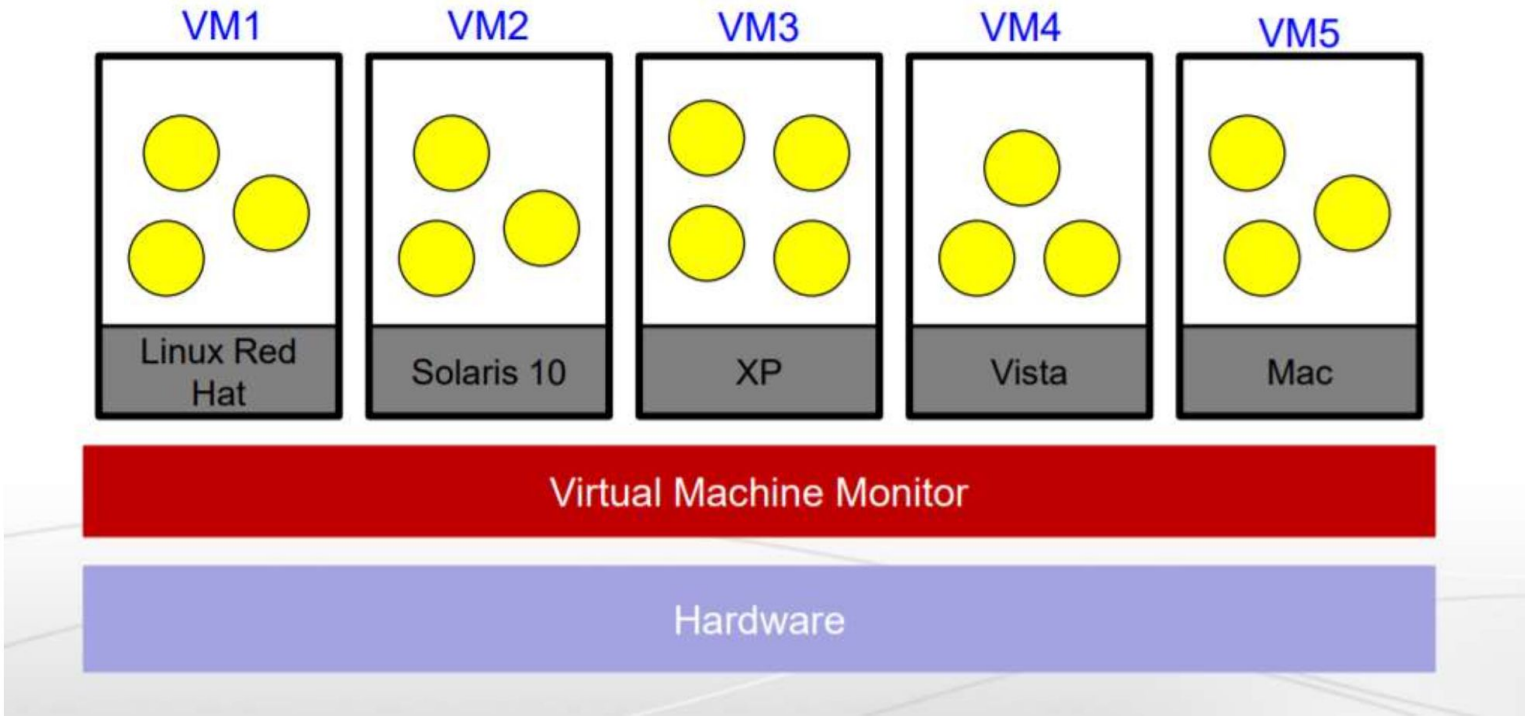
Models	**Characteristics**
Single Threaded Process	No Parallelism, blocking system calls
Threads	Parallelism, blocking system calls
Finite State Machine	Parallelism, No blocking system calls

- Blocking system calls make programming easier and parallelism improves performance.
- The single-threaded server retains the ease and simplicity of blocking system calls, but gives up performance.
- The finite-state machine approach achieves high performance through parallelism, and uses non blocking calls, thus is hard to program.

Virtualization

- Virtualization is the creation of a virtual (rather than actual) version of something, such as a hardware platform, resources, operating system, a storage device or network resources.
- It is the process of running a virtual instance of a computer system in a layer abstracted from the actual hardware. Most commonly, it refers to running multiple operating systems on a computer system simultaneously. To the applications running on top of the virtualized machine, it can appear as if they are on their own dedicated machine, where the operating system, libraries, and other programs are unique to the guest virtualized system and unconnected to the host operating system which sits below it.
- Virtualization refers to the practice of breaking down the physical infrastructure of computing and networking resources into smaller, reusable and more flexible units. These units are then presented to the users as though each was a discrete one.

Virtualization



Virtualization

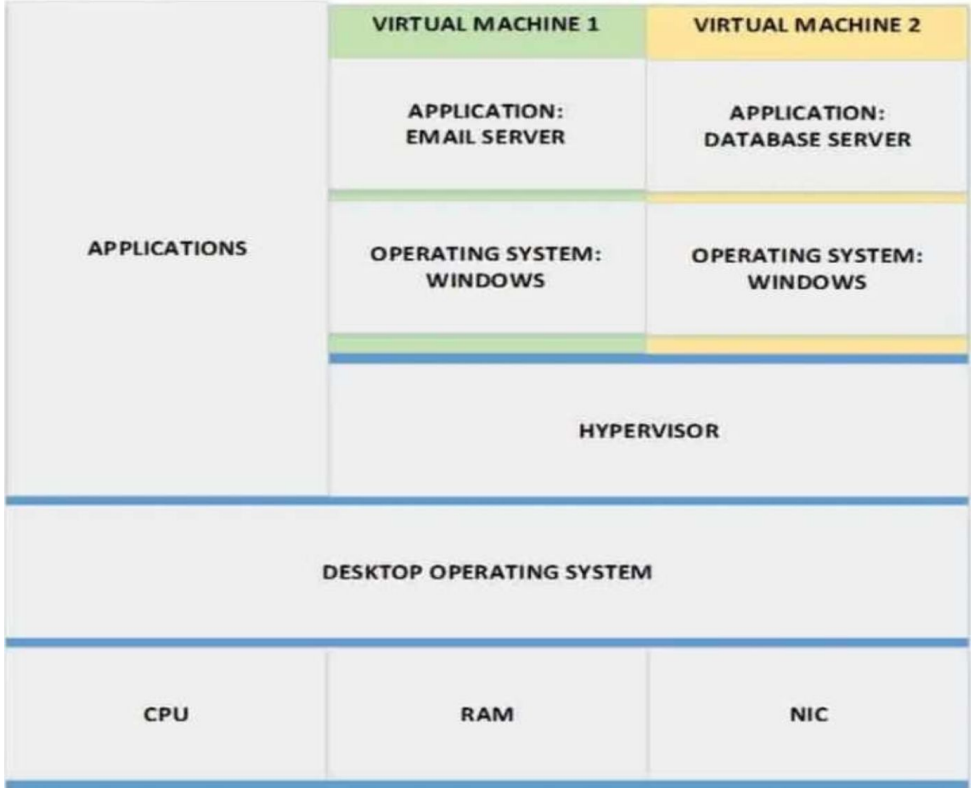
Role of Virtualization

- Helps with scalability and better utilization of hardware resources.
- Allows legacy software to run on expensive mainframe hardware.
- Runs multiple different operating systems at the same time.
- Provides a high degree of portability and flexibility.

Different Types of Virtualization:

- Storage Virtualization: Composition of multiple network storage device into what appears to be a single storage unit.
- Server Virtualization: The partitioning a physical server into small virtual servers.
- Network Function Virtualization: Decoupling of network functions from proprietary hardware appliances and running them as software in virtual machines

Server Virtualization

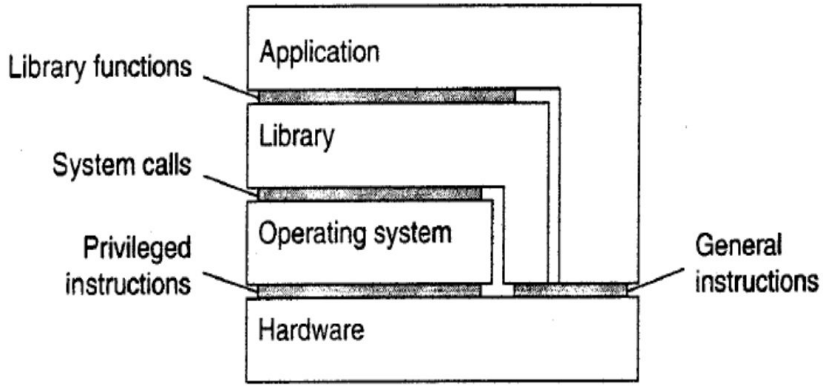


Virtual Machine

- A virtual machine (VM) is a software implementation of a computer machine. It creates an isolated duplication of a real computer and allows us to perform operations as we perform operations on a real computer.
- A VM is a software program that not only exhibits the behavior of a separate computer but is also capable of performing tasks such as running applications and programs like a separate computer.

Architectures of Virtual Machine

- To understand the differences in virtualization, it is important to realize that computer systems generally offer four different types of interfaces, at four different levels:
- An interface between the hardware and softwar consisting of machine instructions
  - Can be invoked by any program.
- An interface between the hardware and softwar consisting of machine instructions
  - can be invoked only by privileged programs, such as an operating system.
- An interface consisting of system calls as offered by an operating system.
- An interface consisting of library calls,
  - known as an application programming interface (API).



The essence of virtualization is to mimic the behavior of these interfaces.

Types of Virtual Machine

- Virtualization can take place in two different ways:
- First, we can build a runtime system that essentially provides an abstract instruction set that is to be used for executing applications. (Process Virtual Machine or Application Virtual Machine)
- An alternative approach toward virtualization is to provide a system that is essentially implemented as a layer completely shielding the original hardware, but offering the complete instruction set of that same (or other hardware) as an interface. Crucial is the fact that this interface can be offered *simultaneously* to different programs. As a result, it is now possible to have multiple, and different operating systems run independently and concurrently on the same platform. The layer is generally referred to as a **virtual machine monitor** (VMM). (Virtual Machine Monitor or System Virtual Machine)

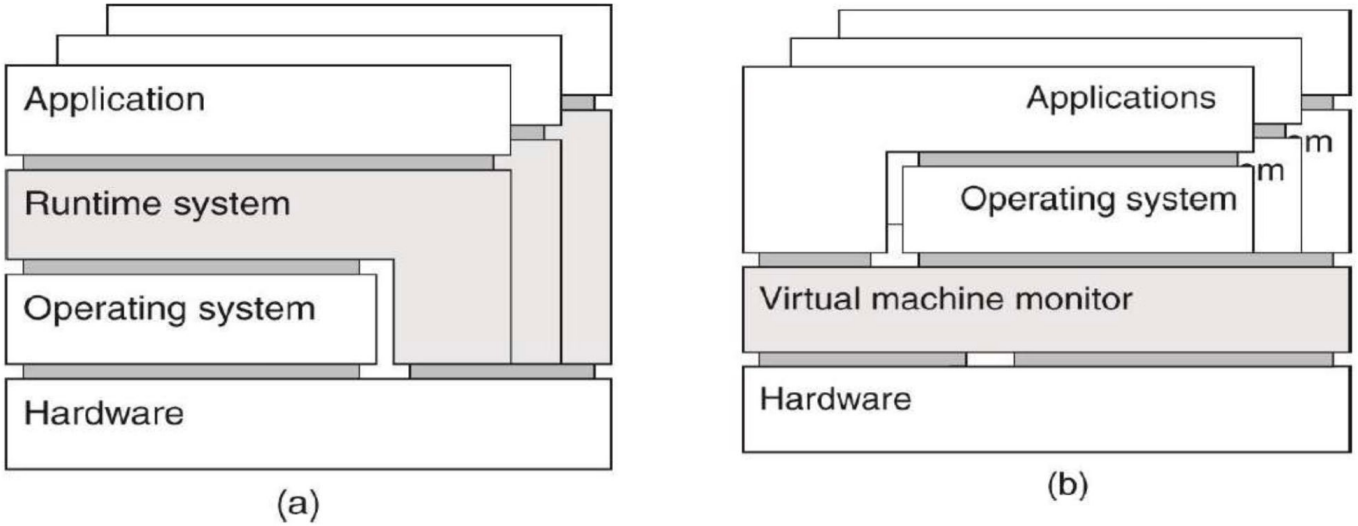
Process Virtual Machine:

- Virtualization is done essentially only for executing a single process (program). An abstract instruction set that is to be used for executing applications.
- For example: Java runtime,.

Virtual Machine Monitor:

- Composed of the host OS and the virtualization software.
- A layer completely shielding the original hardware but offering the complete instruction set of that same (or other hardware) as an interface.
- Provides a further decoupling between hardware and software allowing moving complete environment from one machine to another.
- Makes it possible to have multiple instances of different operating systems run simultaneously and concurrently on the same platform.
- Examples: Vmware, VirtualBox, Xen, VirtualPC, Parallels etc.

Architectures of Virtual Machine



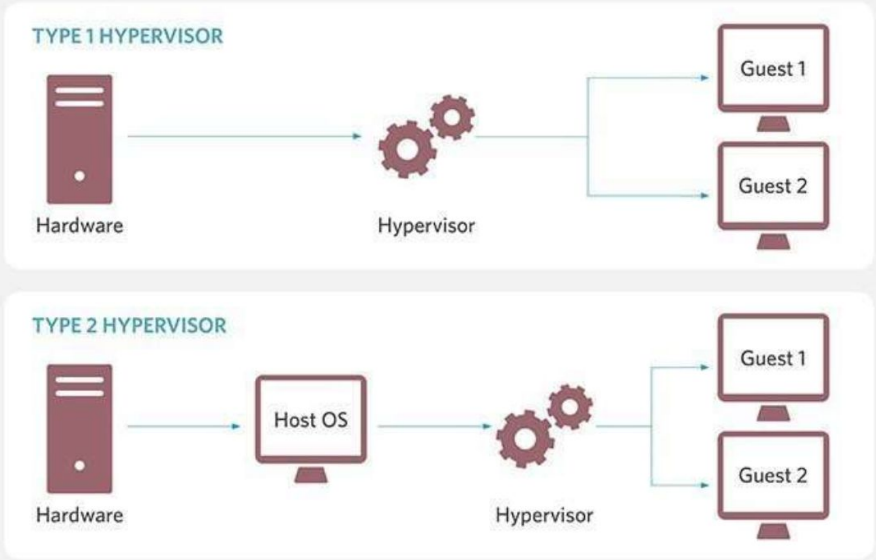
A process virtual machine, with multiple instances of (application, runtime) combinations. (a)

(b) A virtual machine monitor, with multiple instances of (applications, OS) combinations.

Virtual Machine Monitor

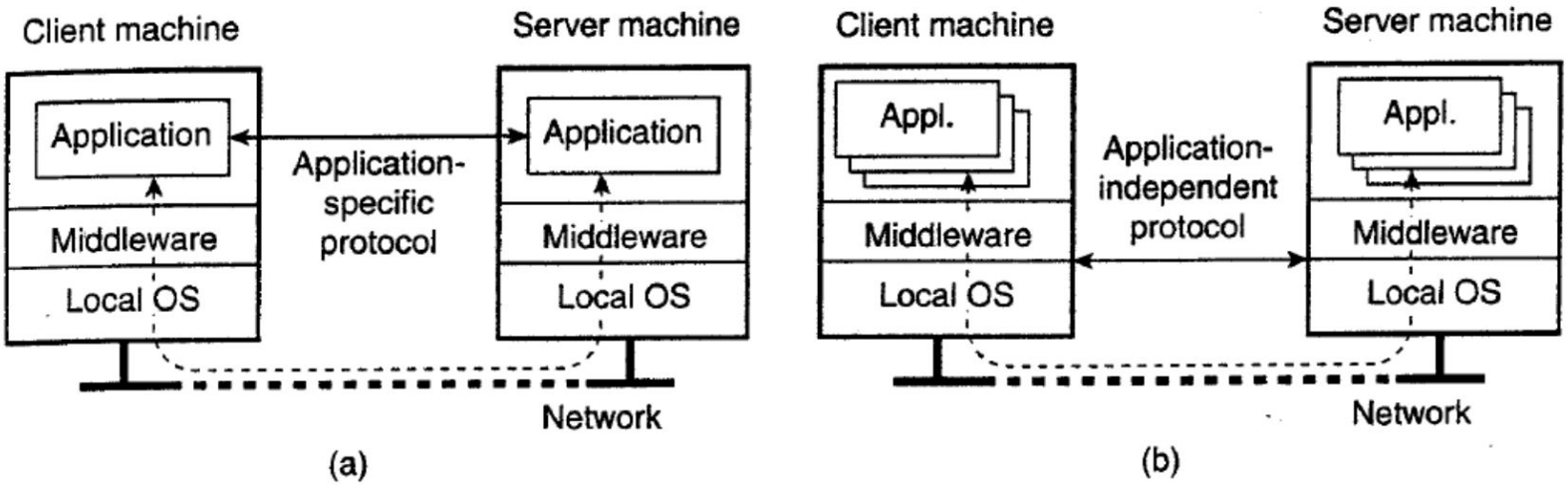
- A Virtual Machine Monitor (VMM) is a software program that enables the creation, management and governance of virtual machines (VM) and manages the operation of a virtualized environment on top of a physical host machine. VMM is also known as Virtual Machine Manager and Hypervisor.
- The hardware on which hypervisor is installed is labeled as the **host machine**. The virtual resources, created and managed by the hypervisor, are known as **virtual or guest machines**.
- A Type 1 Hypervisor runs directly on the system hardware and acts as the operating system. Its other job is to give out access to the underlying hardware to the different virtual machines.
- The most popular Type 1 Hypervisors are VMware ESXI, Microsoft Hyper-V, Red Hat KVM, Oracle VMServer, and Citrix XenServer. In Cloud environments, CitrixXen Server is very popular because it’s the cheapest option to use for the Cloud service providers.
- A Type 2 Hypervisor runs on top of a normal host operating system.
- Popular Type 2 Hypervisors include VMware Workstation and VMware Player for PC, and VMware Fusion for Mac. We also have VirtualBox, QEMU, and Parallels.

Hypervisor types



Clients

- A major task of client machines is to provide the means for users to interact with remote servers. There are roughly two ways in which this interaction can be supported.
- First, for each remote service the client machine will have a separate counterpart that can contact the service over the network. In this case, an application-level protocol will handle the synchronization, as shown in Fig 3.8(a). (Business Logic with in client application)
- A second solution is to provide direct access to remote services by only offering a convenient user interface. Effectively, this means that the client machine is used only as a terminal with no need for local storage, leading to an application neutral solution as in Fig (b).



(a) A networked application with its own protocol. (b) A general solution to allow access to remote applications.

Thick and Thin Clients

- Client thickness is a term which refers to the amount of processing and storage that it does in comparison to the server it is connected to.
- A thick client is a computer/application in client server architecture or networks that typically provides broad functionality independent of the central server. The name is contrasted to thin client, which describes a computer heavily dependent on server's applications.
- A thick client still requires at least periodic connection to a network or central server, but is often characterized by the ability to perform many functions without that connection. In contrast, a thin client generally does as little processing as possible and relies on accessing the server each time input data needs to be processed or validated.

Thick Client: Majority of processing and storage done on client.

Thin Client: Very little processing and storage done on client.

Server Design Issues

- A server is a process implementing a specific service on behalf of a collection of clients. In essence, each server is organized in the same way: it waits for an incoming request from a client and subsequently ensures that the request is taken care of, after which it waits for the next incoming request.

1. Iterative vs Concurrent Server:
   There are several ways to organize servers. In the case of an iterative server, the server itself handles the request and, if necessary, returns a response to the requesting client. A concurrent server does not handle the request itself, but passes it to a separate thread or another process, after which it immediately waits for the next incoming request. A multithreaded server is an example of a concurrent server. An alternative implementation of a concurrent server is to fork a new process for each new incoming request. The thread or process that handles the request is responsible for returning a response to the requesting client.

Iterative vs concurrent

- Iterative server handles one request at a time
- If another request arrives while the server is busy handling an existing request, the new request has to wait.
  - Iterative servers are easier to design, implement and maintain
  - Iterative server is not good for a service with a long “request processing time”.
- Concurrent server handles multiple requests concurrently
  - It can decrease the response time
  - It can achieve high performance on a server with multiple processors
  - It can achieve high performance by overlapping processing and I/O.

2. Stateless vs Stateful Server:

- A stateless server does not remember anything from one request to another. For example, a HTTP server is stateless.
- Stateful servers maintains information about its clients. For example, FTP Server

3. Where clients contact a server?

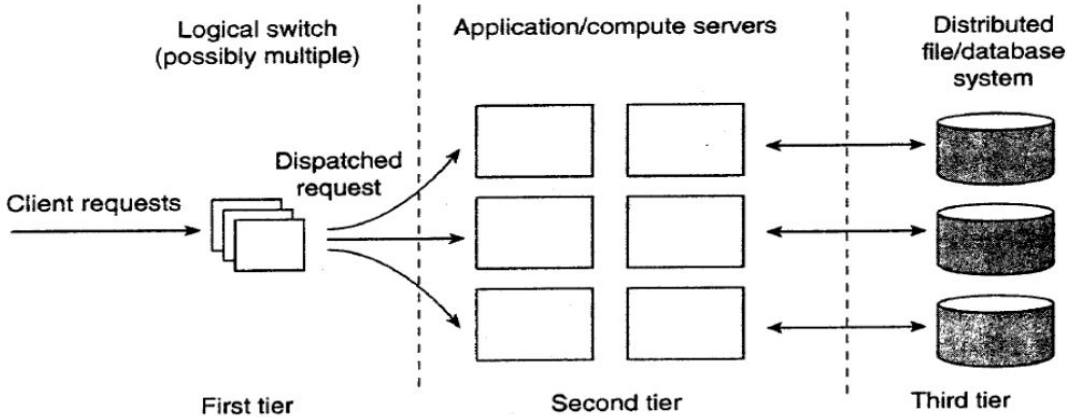
- In all cases, clients send requests to an end point, also called a port, at the machine where the server is running.
- Each server listens to a specific end point:
- Servers that handle Internet FTP requests always listen to TCP **port 21**. An HTTP server for the www listen to TCP port 80.

4. How to handle communication interrupts?

- One approach is abruptly exit the client application (which will automatically break the connection to the server), immediately restart it, and pretend nothing happened. The server will eventually tear down the old connection, thinking the client has probably crashed.
- Another approach is use out-of-band data. Example: to cancel the upload of a huge file.
- Server listens to separate endpoint, which has higher priority, while also listening to the normal endpoint (with lower priority).
- Send urgent data on the same connection. Can be done with TCP, where the server gets a signal (SIGURG) on receiving urgent data.

Server Clusters

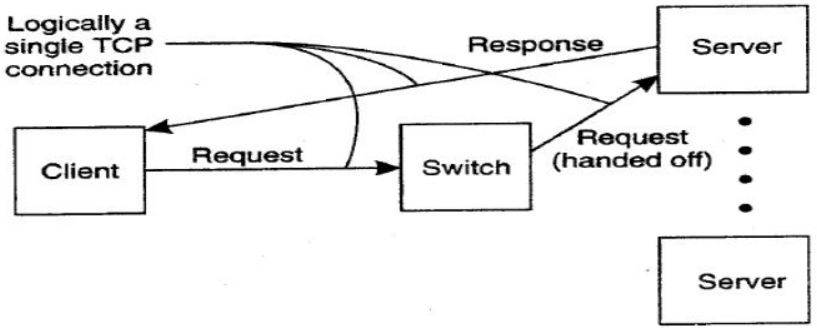
- A server cluster is nothing else but a collection of machines connected through a network, where each machine runs one or more servers.
- In most cases, a server cluster is logically organized into three tiers, as shown in Fig. The first tier consists of a (logical) switch through which client requests are routed.



- As in any multitiered client-server architecture, many server clusters also contain servers dedicated to application processing. In cluster computing, these are typically servers running on high-performance hardware dedicated to delivering compute power.
- The third tier, which consists of data-processing servers, notably file and database servers.
- Not all server clusters will follow this strict separation. It is frequently the case that each machine is equipped with its own local storage, often integrating application and data processing in a single server leading to a two tiered architecture.
- An important design goal for server clusters is to hide the fact that there are multiple servers. In other words, client applications running on remote machines should have no need to know anything about the internal organization of the cluster.
- This access transparency is invariably offered by means of a single access point, in turn implemented through some kind of hardware switch such as a dedicated machine.
- The switch forms the entry point for the server cluster, offering a single network address. For scalability and availability, a server cluster may have multiple access points, where each access point is then realized by a separate dedicated machine.

TCP handoff

- A standard way of accessing a server cluster is to set up a TCP connection over which application-level requests are then sent as part of a session. A session ends by tearing down the connection. In the case of transport-layer switches, the switch accepts incoming TCP connection requests, and hands off such connections to one of the servers
- When the switch receives a TCP connection request, it subsequently identifies the best server for handling that request, and forwards the request packet to that server. The server, in turn, will send an acknowledgment back to the requesting client, but inserting the switch's IP address as the source field of the header of the IP packet carrying the TCP segment.
- This spoofing is necessary for the client to continue executing the TCP protocol: it is expecting an answer back from the switch, not from some arbitrary server it is has never heard of before.



Code Migration

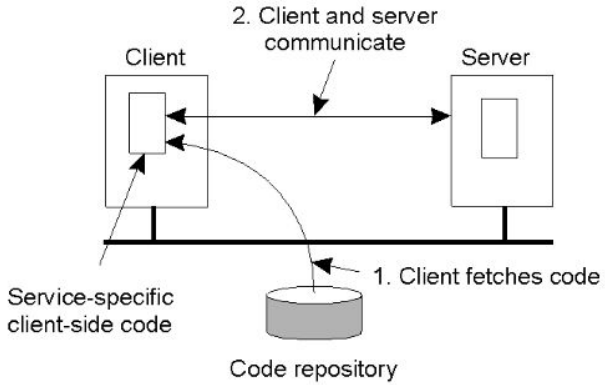
- So far, we have been mainly concerned with distributed systems in which communication is limited to passing data. However, there are situations in which passing programs, sometimes even while they are being executed, simplifies the design of a distributed system.
- Code Migration (mobility) can be defined informally as the capability to dynamically change the bindings between code fragments and the location where they are executed.
- Code migration in the broadest sense deals with moving programs between machines, with the intention to have those programs be executed at the target.

Reasons for Code Migration:

- Improve computing performance by moving processes from heavily-loaded machines to lightly loaded machines. - E.g. Java applets
- Improve communication times by shipping code to systems where large data sets reside. Otherwise, the network may be swamped with the transfer of data from the server to the client. In this case, code migration is based on the assumption that it generally makes sense to process data close to where those data reside. - E.g. a client ships code to a database server or vice versa. This same reason can be used for migrating parts of the server to the client.

Reason for Code Migration

- Flexibility to dynamically configure distributed systems.



The principle of dynamically configuring a client to communicate to a server. The client first fetches the necessary software, and then invokes the server . The important advantage of this model of dynamically downloading client side software is that clients need not have all the software preinstalled to talk to servers. Instead, the software can be moved in as necessary, and likewise, discarded when no longer needed.

Migration Models

- A process consists of three segments: code segment, resource segment, execution segment.

  Code Segment

  - Set of instructions being executed

  Resource Segment

  - References to external resources
  - Files, printers, devices, et cetera

  Execution Segment

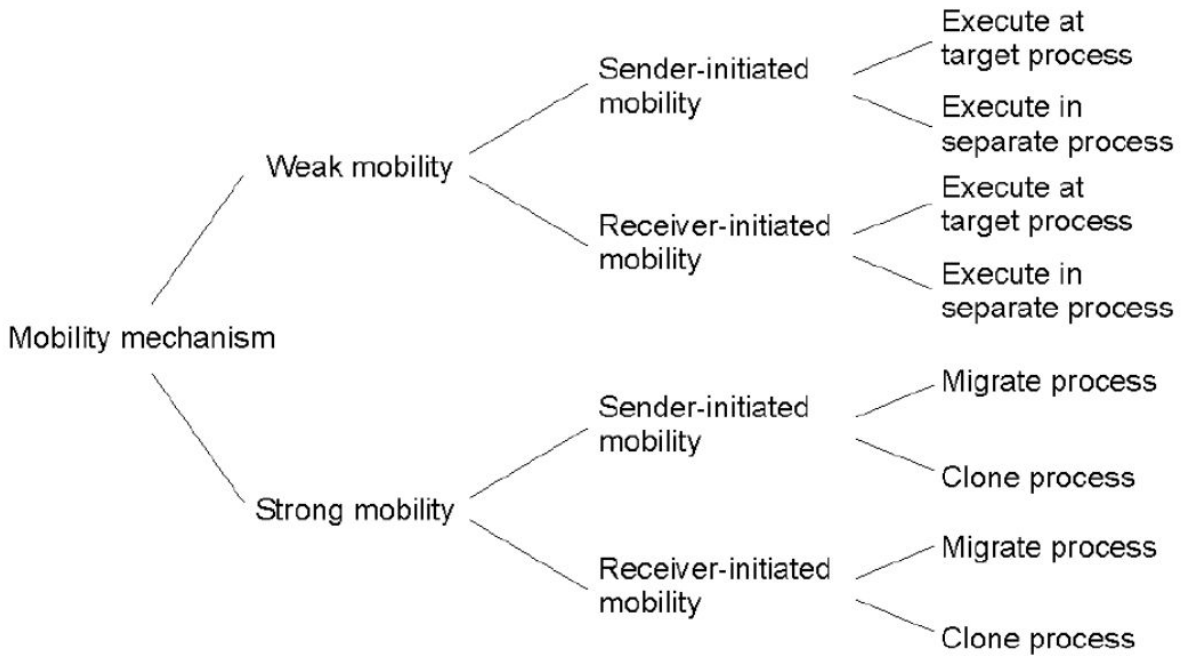
  - Process state, private data, stack , and instruction pointer

- Migration (or mobility) can be **weak** or **strong**:
  Weak migration: only the code segment can be transferred. Transferred program always starts at one of the predefined starting positions. E.g. java applets.
  Strong migration: code and execution segments can both be transferred.
- Also, migration can be **sender-initiated** or **receiver-initiated**:
  Sender initiated: uploading code to a server. Ex: Client sending a query to database server.
  Receiver-initiated: downloading code from a server by a client. Ex: client downloading Java applets from a server.

- Strong code migration (mobility) is best characterized by two terms:
  - Migration
    is when an execution unit is suspended, transferred from one computational environment to another, and resumes execution.
  - Remote Cloning
    is when a copy of the running execution unit is copied to another computational environment while continuing execution on the original executional environment.

- In the case of weak mobility, it also makes a difference if the migrated code is executed by the target process, or whether a separate process is started.

The various alternatives for code migration are summarized:



Resource Migration:

- What often makes code migration so difficult is that the resource segment cannot always be simply transferred along with the other segments without being changed. For example, suppose a process holds a reference to a specific TCP port through which it was communicating with other (remote) processes. Such a reference is held in its resource segment. When the process moves to another location, it will have to give up the port and request a new one at the destination.
- In other cases, transferring a reference need not be a problem. For example, a reference to a file by means of an absolute URL will remain valid irrespective of the machine where the process that holds the URL resides.

Process to Resource Binding:

1. By Identifier:
    - The strongest binding is when a process refers to a resource by its identifier. In this case, the process requires precisely the referenced resource, and nothing else. An example of such a binding by identifier is when a process uses a VRL to refer to a specific Web site or when it refers to an FrP server by means of that server's Internet address.
2. By Value:
    - A weaker form of process-to-resource binding is when only the value of a resource is needed. In that case, the execution of the process would not be affected if another resource would provide that same value. Not the specific files, but their content is important for the proper execution of the process.

3. By Type:

- The weakest form of binding is when a process indicates it needs only a resource of a specific type. This binding by type is exemplified by references to local devices, such as monitors, printers, and so on.

Resource to Machine Binding:

1. Unattached Resources:
- Unattached resources can be easily moved between different machines, and are typically (data) files associated only with the program that is to be migrated.
2. Fastened Resources:
- Moving or copying a fastened resource may be possible, but only at relatively high costs. Typical examples of fastened resources are local databases and complete Web sites. Although such resources are, in theory, not dependent on their current machine, it is often infeasible to move them to another environment.
3. Fixed Resources:
- Fixed resources are intimately bound to a specific machine or environment and cannot be moved. Fixed resources are often local devices. Another example of a fixed resource is a local communication end point

Combining three types of process-to-resource bindings, and three types of resource-to-machine bindings, leads to nine combinations that we need to consider when migrating code. These nine combinations are shown in Fig.

Process-to-Resource Binding	Resource-To-Machine Binding		
	Unattached	Fastened	Fixed
By identifier	MV (or GR)	GR (or MV)	GR
By value	CP (or MV, GR)	GR (or CP)	GR
By type	RB (or MV, CP)	RB (or GR, CP)	RB (or GR)

- Actions to be taken with respect to the references to local resources when migrating code to another machine
  - **GR:** Establish a global system-wide reference
  - **MV:** Move the resource
  - **CP:** Copy the value of the resource
  - **RB:** Rebind the process to a locally available resource