Distributed Systems

Processes

Chapter 3

Content

3.1 Threads
3.2 Virtualization
3.3 Clients
3.4 Servers
3.5 Code Migration

Process

- A Program does nothing unless its instructions are executed by a CPU.
- A program in execution is called a process.
- In order to accomplish its task, process needs the computer resources.
- There may exist more than one process in the system which may require the same resource at the same time.

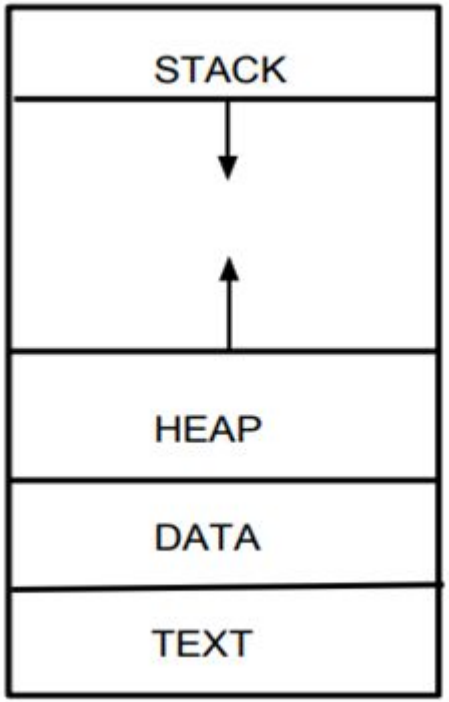
- Some resources may need to be executed by one process at one time to maintain the consistency otherwise the system can become inconsistent and deadlock may occur.

Child Process

- A process can initiate a sub-process , which is called child process.

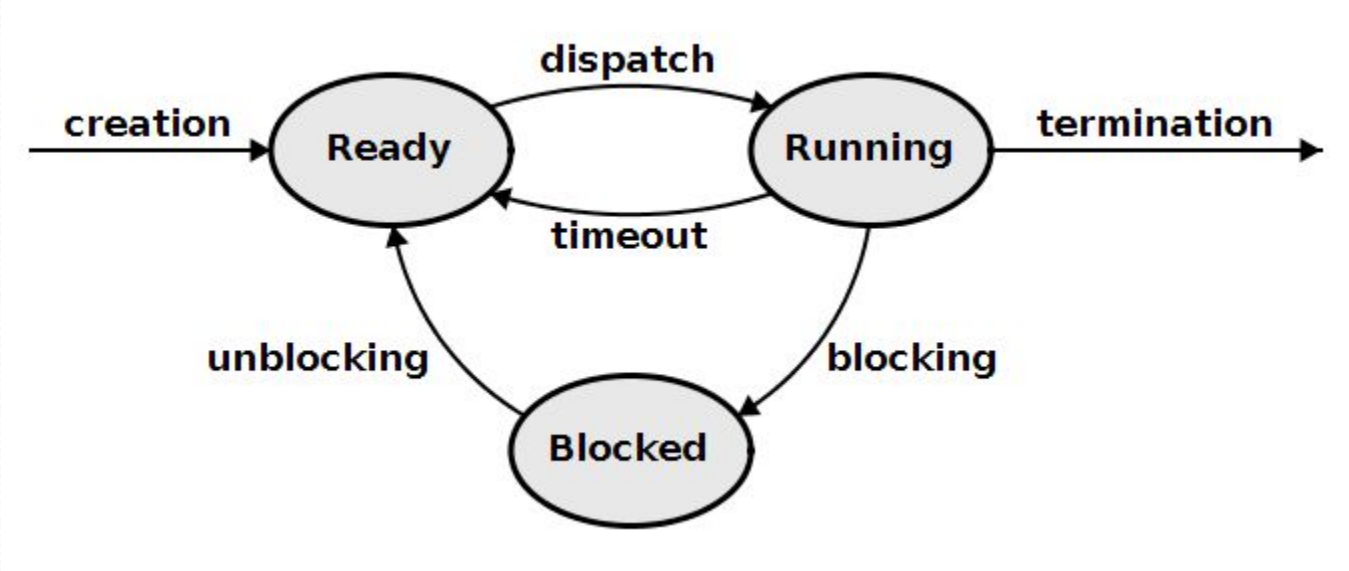
- A child process is replica of parent process and share resources of parent process , but cannot exist if parent is terminated.

Layout of process inside main memory



- Stack: This section contains local variable, function and returns address.
- Heap: This Section is used to provide dynamic memory whenever memory is required by the program during runtime It is provided form heap section.
- Text: This section contains the executable instruction, constants
- Data: This Section contains the global variables and static local variables.

Process Life Cycle



- The life-cycle of a process can be described by a state diagram which has states representing the execution status of the process at various times .
- A process is created .
- Whenever a process is created, it directly enters in the ready state, in which, it waits for the CPU to be assigned.
- One of the processes from the ready state will be chosen by the OS depending upon the scheduling algorithm.
- From the Running state, a process can make the transition to the block or wait state depending upon the scheduling algorithm or the intrinsic behavior of the process.
- A process terminates or ends.

Threads

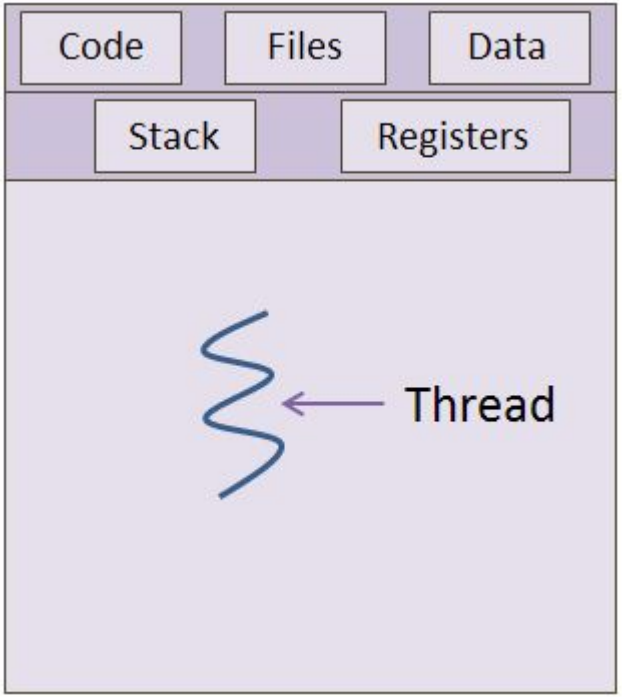
- A thread is the smallest unit of processing that can be performed in an OS.

- Think of MS Word application, which is a process that runs on computer. But an application can do more than one thing at a time, which means that a given process in an operating system can have one or more **threads**.

Advantages of Thread over Process

- *1. Responsiveness:* If the process is divided into multiple threads, if one thread completes its execution, then its output can be immediately returned.
- *2. Faster context switch:* Context switch time between threads is lower compared to process context switch. Process context switching requires more overhead from the CPU.

- 3. *Resource sharing:* Resources like code, data, and files can be shared among all threads within a process.
- 5. *Communication:* Communication between multiple threads is easier, as the threads shares common address space.

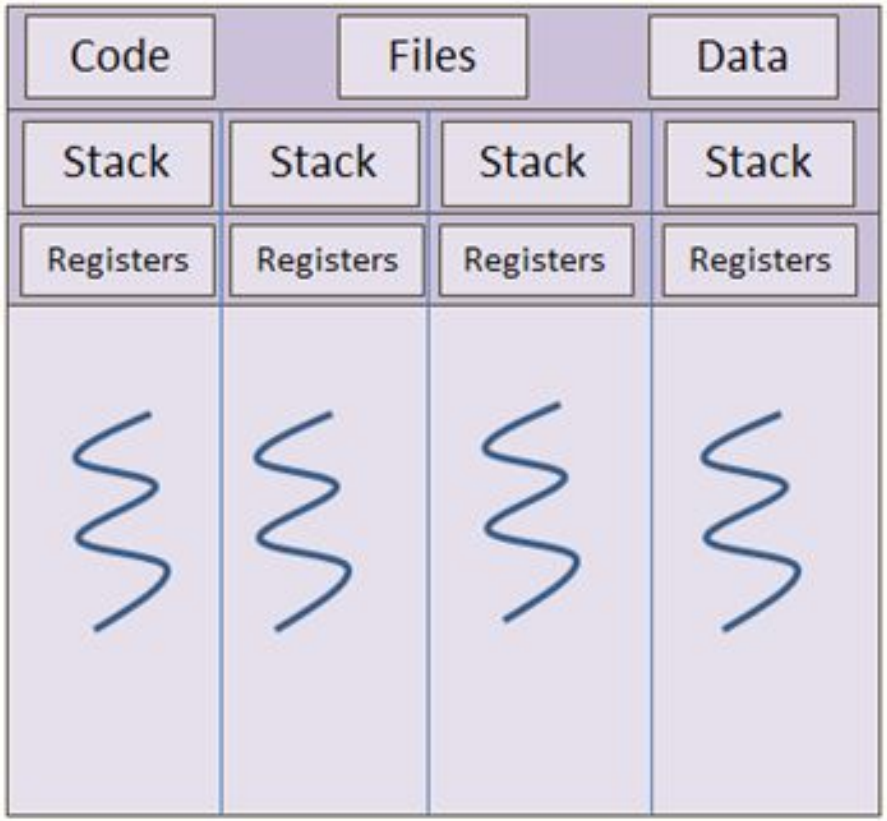

Code
Files
Data
Stack
Registers
Thread

Processes	Threads
When switching a process, operating system's resources are required	No OS resources are required for thread-switching
If a process is blocked, other processes waiting in the queue are also blocked	If a thread is blocked, another thread in the same process can still execute
Each process uses same code and has its own memory	All threads can share files and share child processes
An application having multiple processes will use more system resources	Processes using multiple threads use less system resources
Each process works on its own	Threads can access data of other threads

Multithreading

- **Multi-threading** is the ability of a process to execute multiple threads at the same time.
- Again, the MS Word example is appropriate in multi-threading scenarios.
- The process can check spelling, auto-save, and read files from the hard-drive, all while you are working on a document.

- Consider the following diagram. The threads share the same code, files, and data.
- This means that two ore more threads can run at the same time (auto-save, grammar check, spell-check, word-count, etc.).


Code
Files
Data
Stack
Stack
Stack
Stack
Registers
Registers
Registers
Registers

Benefits of Multithreading

- **Resource Sharing :**All the threads of a process share its resources such as memory, data, files etc.

- **Responsiveness :**Program responsiveness allows a program to run quickly. For example - A web browser with multithreading can use one thread for user contact and another for image loading at the same time.

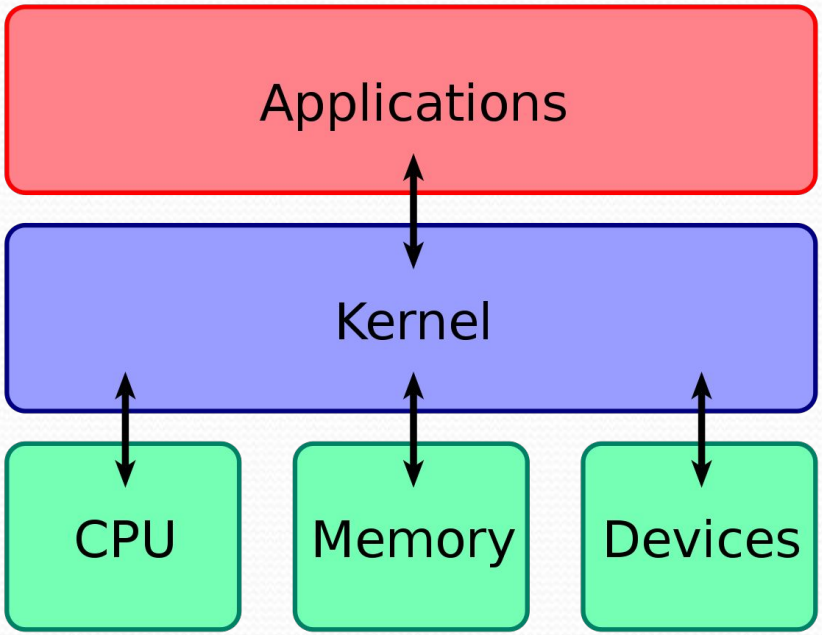
- **Utilization of Multiprocessor Architecture :** In a multiprocessor architecture, each thread can run on a different processor in parallel using multithreading.

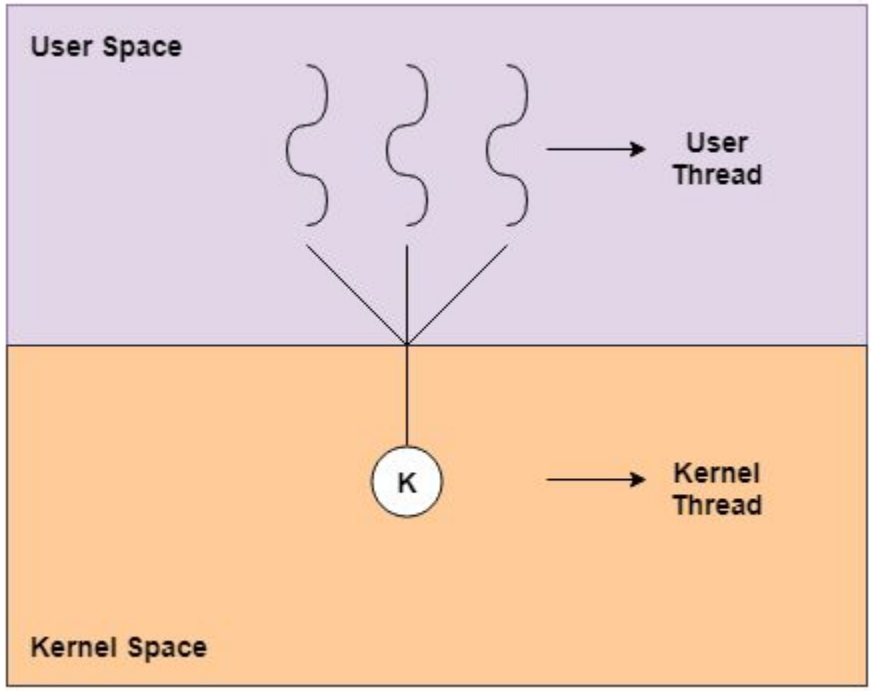
- **Economy :** It is more economical to use threads as they share the process resources.

Types of Thread

- The two main types of threads are user-level threads and kernel-level threads.
- A diagram that demonstrates these is as follows –

- *Note : The kernel is* ***the essential center of a computer operating system*** *(OS). It is the core that provides basic services for all other parts of the OS.*




User Space
User
Thread
K
Kernel
Thread
Kernel Space

User Level Thread

- The user-level threads are implemented by users and the kernel is not aware of the existence of these 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.
- Also, there is no kernel involvement in synchronization for user-level threads.
- examples: Java thread.

- 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.

Kernel Level Thread

Kernel-level threads are handled by the 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 user-level threads. Example: Window .

Problems with Processes

- Creating and managing processes is generally regarded as an *expensive* task .
- Making sure all the processes peacefully co-exist on the system is not easy .

Thread Model

- All modern OS support Kernel Level Thread, allowing kernel to perform multiple simultaneous task.
- Hence , user thread must be mapped to kernel thread.

- Three models:
  - Many to One
  - One to One
  - Many to Many

Many to One

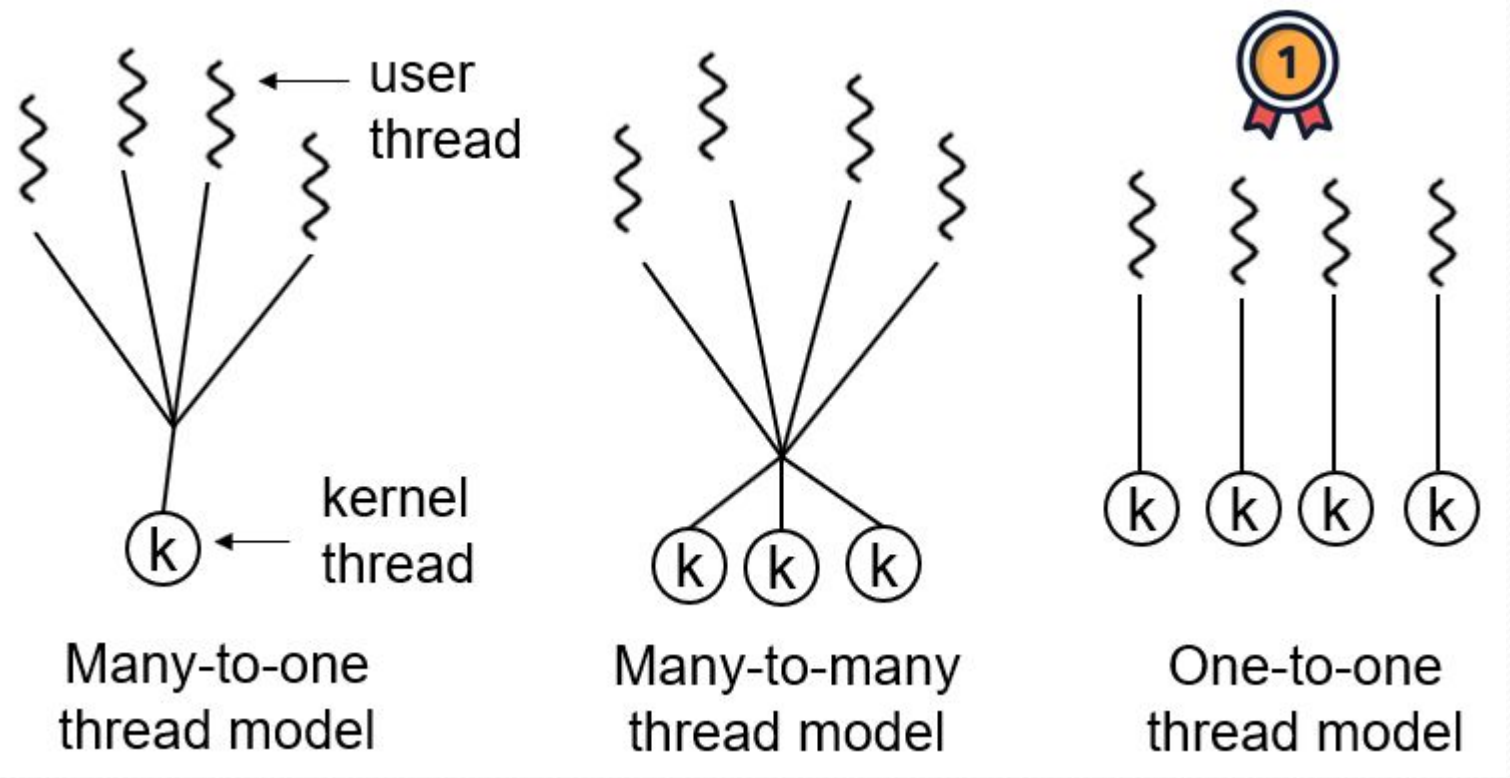
- Maps many user level thread to one kernel level thread

One to One

- Maps one user level thread to one kernel level thread

Many to Many

- Maps many user level thread to many kernel level thread


user
thread
kernel
thread
k
Many-to-one
thread model
k
k
k
Many-to-many
thread model
1
k
k
k
k
One-to-one
thread model

Students Work

- Example of each type of model

Why Mapping from User to kernel?

- All the user threads that are created by a process are executed over the same kernel level thread appointed to the whole process.
- Whenever it's the turn for the specified process to execute on the CPU it's kernel thread is scheduled onto the CPU and hence the process is executed.
- The user threads, since all of them are controlled by the creating process itself, they are to be mapped onto the appointed kernel thread one by one and therefore executed.

- We can think of this whole process as creating a great new product, maybe an electronic gadget or something. If that product is to be sold it has to be sold under a brand name and that brand or company needs to be registered to the government and further that company has to follow the rules and regulations imposed by the government to sell the desired products via shops in the market.

- Here I am referring to user threads as the product, kernel as government, process as company and shops as kernel threads.

Important Implications

- Two Important Implications:
  1. Threaded applications often run faster than non-threaded applications .
  2. Threaded applications are harder to develop .

Virtualization

- It is the process of creating a virtual version of something like computer hardware, storage or network.

- It was initially developed during the mainframe era.

- It involves using specialized software to create a virtual or software-created version of a computing resource rather than the actual version of the same resource.

- With the help of Virtualization, multiple operating systems and applications can run on same machine and its same hardware at the same time, increasing the utilization and flexibility of hardware.

- In other words, one of the main cost effective, hardware reducing, and energy saving techniques used by cloud providers is virtualization.

- Virtualization allows to share a single physical instance of a resource or an application among multiple customers and organizations at one time.

- The machine on which the virtual machine is going to be built is known as Host Machine and that virtual machine is referred as a Guest Machine.

BENEFITS OF VIRTUALIZATION

1. More flexible and efficient allocation of resources.
2. Enhance development productivity.
3. It lowers the cost of IT infrastructure.
4. Remote access and rapid scalability.
5. High availability and disaster recovery.
6. Pay per use of the IT infrastructure on demand.
7. Enables running multiple operating systems.

Disadvantage of Virtualization

- It can have a high cost of implementation.
- It creates a security risk.
- It creates an availability issue.
- It creates a scalability issue.
- It takes time.

TYPES OF VIRTUALIZATION

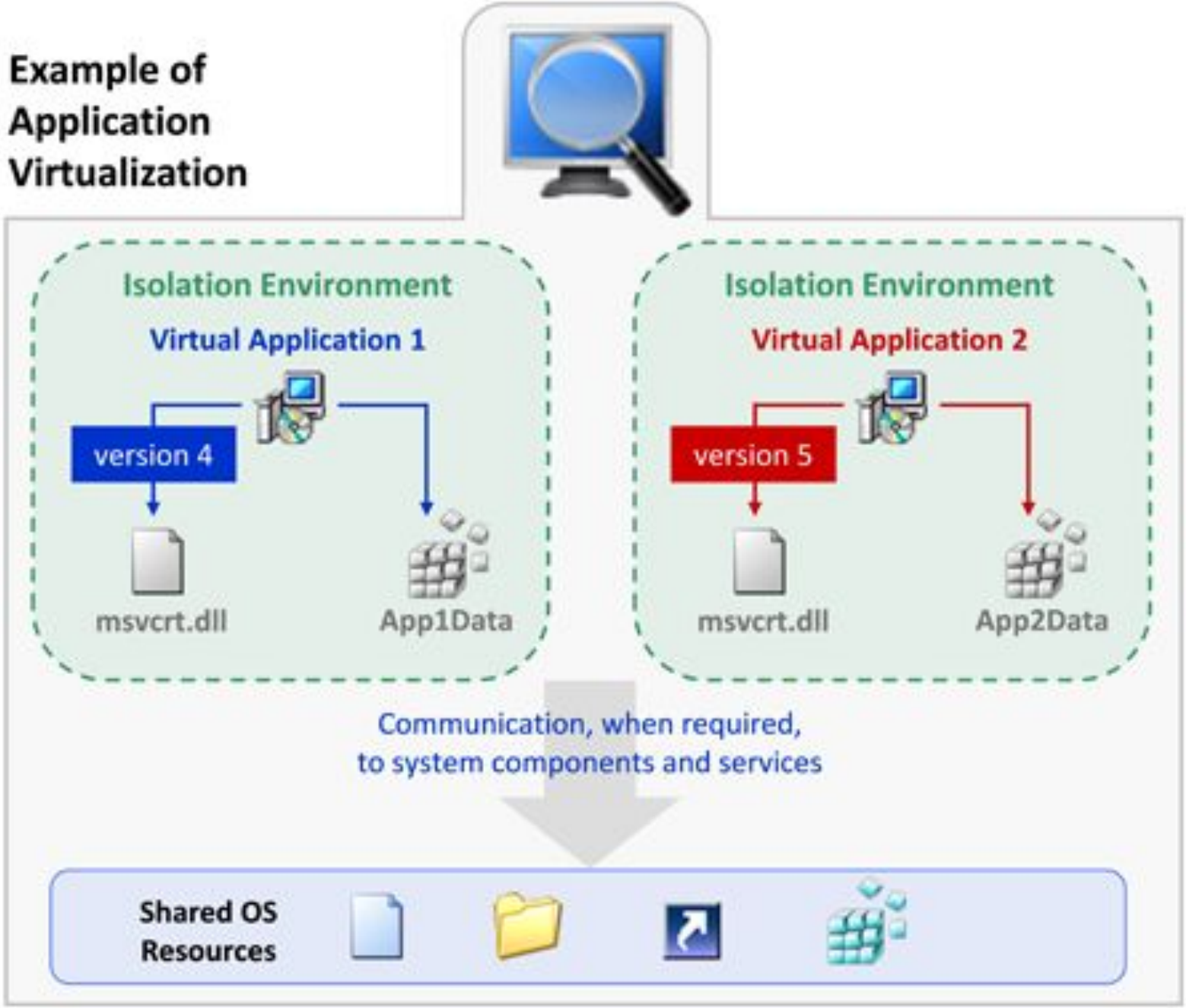
- 1.Application Virtualization.
- 2.Network Virtualization.
- 3.Desktop Virtualization.
- 4.Storage Virtualization.
- 5.Server Virtualization. 6.Data virtualization.

1. Application Virtualization:

- Application virtualization helps a user to have remote access of an application from a server.
- **Application virtualization** software allows users to access and use an application from a separate computer than the one on which the application is installed.

- Example of this would be a user who needs to run two different versions of the same software.

Example of Application Virtualization



2. Network Virtualization:

- **Network virtualization** is the transformation of a network that was once hardware-dependent into a network that is software-based.
- Network virtualization provides a facility to create and provision virtual networks—logical switches, routers, firewalls, load balancer.

One example of network virtualization is virtual LAN (VLAN). A VLAN is a subsection of a local area network (LAN) created with software that combines network devices into one group, regardless of physical location.

3. Desktop Virtualization:

- **Desktop virtualization** is simply the concept of replacing traditional physical desktop environments with remotely controlled computing environments

- Desktop virtualization allows the users' OS to be remotely stored on a server in the data centre.

- It allows the user to access their desktop virtually, from any location by a different machine.
- Users who want specific operating systems other than Windows Server will need to have a virtual desktop.
- Main benefits of desktop virtualization are user mobility, portability, easy management of software installation, updates, and patches.

4. Storage Virtualization:

- Storage virtualization is the **pooling of physical storage from multiple storage devices** into what appears to be a single storage device. Storage virtualization is an array of servers that are managed by a virtual storage system.
- The servers aren't aware of exactly where their data is stored, and instead function more like worker bees in a hive.
- It makes managing storage from multiple sources to be managed and utilized as a single repository.

5. Server Virtualization:

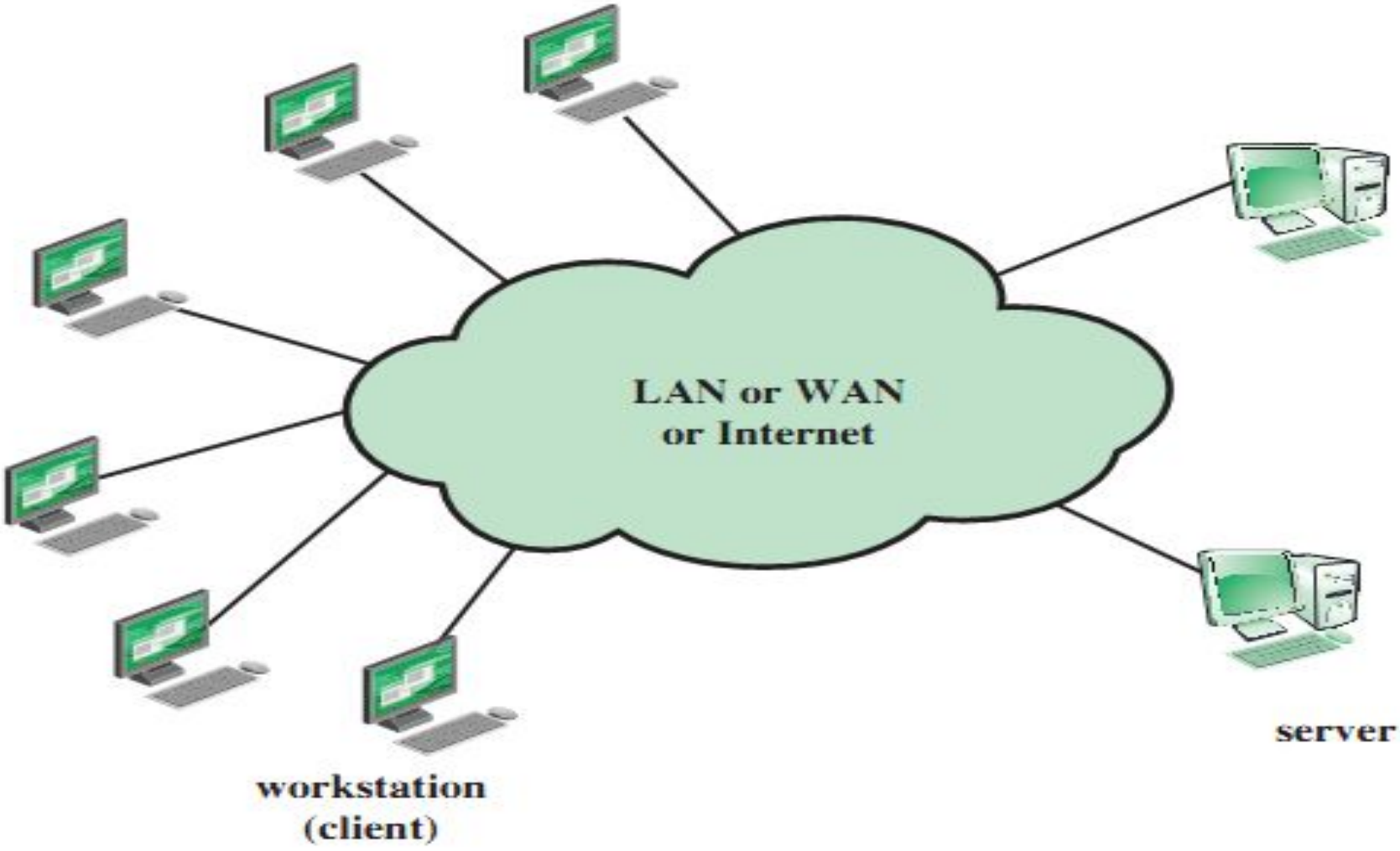
- Server virtualization is the process of **dividing a physical server into multiple unique and isolated virtual servers** by means of a software application.
- Here, the central-server(physical server) is divided into multiple different virtual servers by changing the identity number, processors.
- So, each system can operate its own operating systems in isolate manner.

- **6. Data virtualization:**
- This is the kind of virtualization in which the data is collected from various sources and managed that at a single place without knowing more about the technical information like how data is collected, stored & formatted then arranged that data logically.
- Many big giant companies are providing their services like Oracle, IBM, At scale, Cdata, etc.

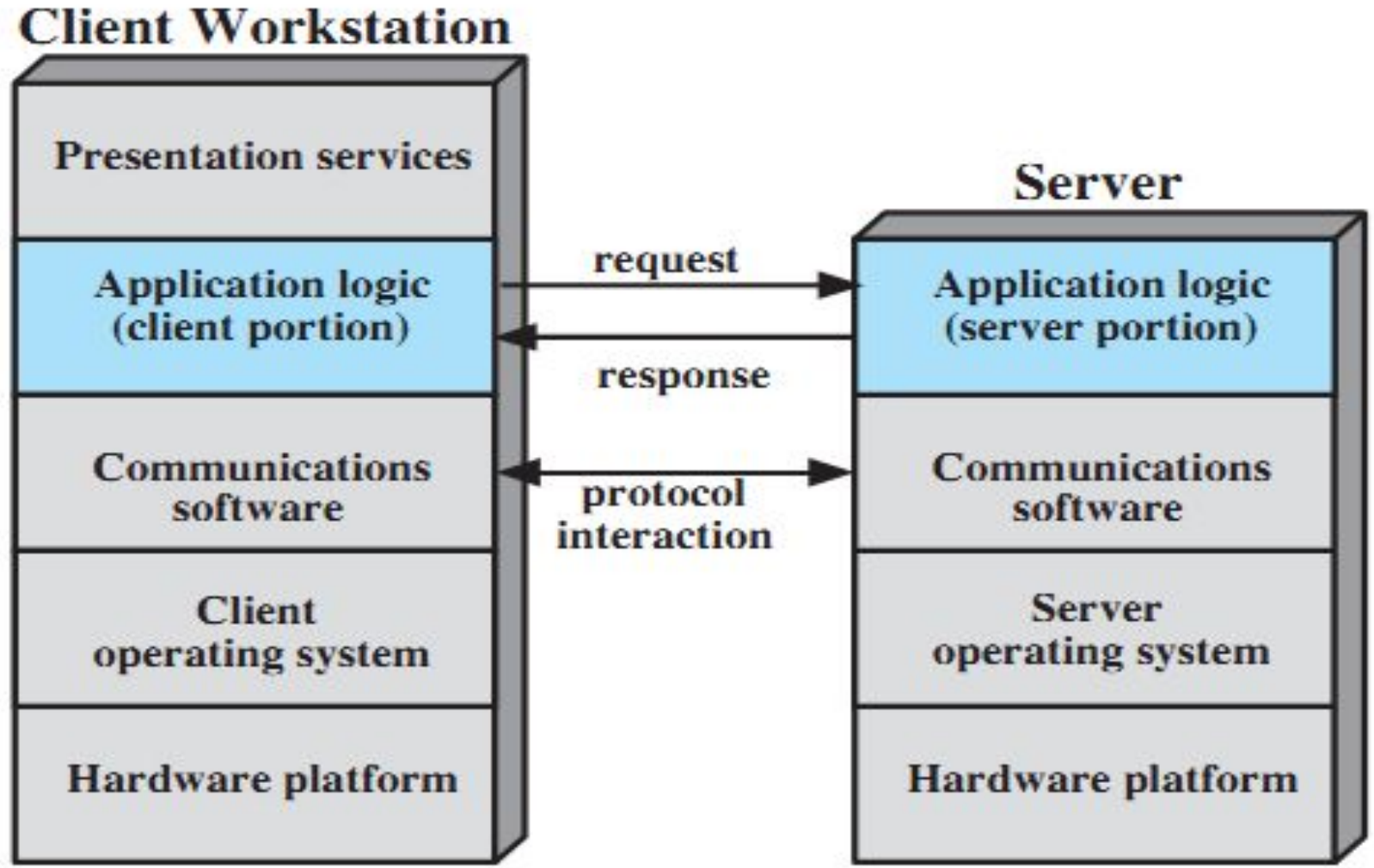
Clients

- What's a client?
- *Definition*: "A program which **interacts** with a human user and a remote server."
- Typically, the user interacts with the client via a GUI.
- Of course, there's more to clients than simply providing a UI. Remember the multi-tiered levels of the Client/Server architecture from earlier

Generic Client/Server Environment



Generic Client/Server Architecture



Servers

- What's a server?
- *Definition*: "A process that implements a specific service **on behalf of** a collection of clients".
- Typically, servers are organized to do one of two things:
  1. Wait
  2. Service

... wait ... service ... wait ... service ... wait ...

Servers: Iterative and Concurrent

- *Iterative*: server handles request, then returns results to the client; any new client requests *must wait* for previous request to complete (also useful to think of this type of server as *sequential*).

- *Concurrent*: server does not handle the request itself; a separate thread or sub-process handles the request and returns any results to the client; the server is then free to immediately service the next client (i.e., there's no waiting, as service requests are processed in *parallel*).

Server “States”- Stateful & Stateless

- A Stateful server remember client data (state) from one request to the next. Stateful servers, do store session state. They may, therefore, keep track of which clients have opened which files, current read and write pointers for files, which files have been locked by which clients, etc.

- A Stateless server keeps no state information.Stateless file servers do not store any session state.

- **Programming :**
- Stateful server is harder to code.
- Stateless server is straightforward to code.
- **Crash recovery :**
- Stateful servers have difficult crash recovery due to loss of information.
- Stateless servers can easily recover from failure because there is no state that must be restored.

Code Migration

- Traditionally, **code migration in distributed systems** took place in the form of process **migration** in which an entire process was moved from one machine to another.

- The basic idea is that overall **system** performance can be improved if processes are moved from heavily-loaded to lightly-loaded machines.

- Process migration **can be used to** load balancing, to achieve overall better utilization of a distributed system by ensuring that the computational load is appropriately shared among the processors.

- **Code migration** in the broadest sense deals with moving programs between machines, with the intention to have those programs be executed at the target.

Process and Code Migration

- Under certain circumstances, in addition to the usual passing of data, *passing code* (even while it is executing) can greatly simplify the design of a DS.
- However, code migration can be inefficient and very costly.
- So, why migrate code?

Reasons for Migrating Code

- Why? Biggest single reason: **better performance**.
- The big idea is to move a compute-intensive task from a *heavily loaded* machine to a *lightly loaded* machine
  "on demand" and "as required".

Code Migration Examples

- *Moving (part of) a client to a server* – processing data close to where the data resides. It is often too expensive to transport an entire database to a client for processing, so move the client to the data.

- *Moving (part of) a server to a client* – The use of local error-checking (using JavaScript) on Web forms is a good example of this type of processing. Error-check the data close to the user, not at the server.

SOME SOLVED QUESTIONS

USER LEVEL VS KERNEL LEVEL

User level thread	Kernel level thread
User thread are implemented by users.	kernel threads are implemented by OS.
OS doesn't recognize user level threads.	Kernel threads are recognized by OS.
Implementation of User threads is easy.	Implementation of Kernel thread is complicated.
Context switch time is less.	Context switch time is more.
Example : Java thread, POSIX threads.	Example : Window Solaris.

- WHY ULT: 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.
- WHY KLT: If one kernel thread perform blocking operation then another thread can continue execution.

- **Actions taken by a kernel to context-switch between kernel-level threads are-**

  Context switching between kernel threads typically requires saving the value of the CPU registers from the thread being switched out and restoring the CPU registers of the new thread being scheduled.

What resources are used when a thread created? How do they differ from those when a process is created?

- When a thread is created the threads does not require any new resources .The thread shares the resources like memory of the process to which they belong to.
- Where as if a new process creation is very heavyweight because it always requires new address space to be created .

- **The actions taken by a thread library to context switch between user-level threads.**
- Answer: In general, context switching between user threads involves taking a user thread of its Light Weight Process and replacing it with another thread. This act typically involves saving and restoring the state of the registers.

- **Under what circumstances does a multithreaded solution using multiple kernel threads provide better performance than a single-threaded solution on a single-processor system?**

- When a kernel thread suffers a page fault, another kernel thread can be switched in to use the interleaving time in a useful manner. A single-threaded process, on the other hand, will not be capable of performing useful work when a page fault takes place. Therefore, in scenarios where a program might suffer from frequent page faults or has to wait for other system events, a multi-threaded solution would perform better even on a single-processor system.

- **Can a multithreaded solution using multiple user-level threads achieve better performance on a multiprocessor system than on a single processor system? Explain.**

- A multithreaded system comprising of multiple user-level
  threads cannot make use of the different processors in a multiprocessor system simultaneously. The operating system sees only a single process and will not schedule the different threads of the process on separate processors. Consequently, there is no performance benefit associated with executing multiple user-level threads on a multiprocessor system.

- **Provide two programming examples in which multithreading does not provide better performance than a single-threaded solution.**

- Any kind of sequential program is not a good candidate to be threaded. An example of this is a program that calculates an individual tax return. (2) Another example is a "shell" program such as the C-shell or Korn shell. Such a program must closely monitor its own working space such as open files, environment variables, and current working directory.