Haroon Rashid

Reg# 16549

Semester: 6th

Final Term Paper: Operating System Concepts

Submitted to: Sir DAUD KHAN

1. **Differentiate between a process and thread with example.**

Answer: **Process:**

A process is the execution of a program that allows you to perform the appropriate actions specified in a program. It can be defined as an execution unit where a program runs. The OS helps you to create, schedule, and terminates the processes which is used by CPU. The other processes created by the main process are called child process.

A process operation can be easily controlled with the help of PCB (Process Control Block). You can consider it as the brain of the process, which contains all the crucial information related to processing like process id, priority, state, and contents CPU register, etc.

Example

if you open up two browser windows then you have two processes, even though they are running the same program.

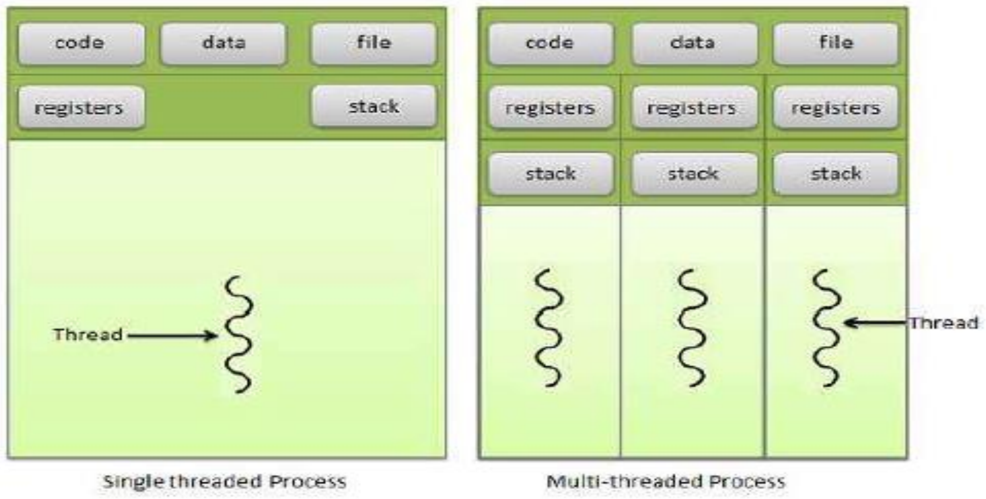
Thread: -

Thread is an execution unit that is part of a process. A process can have multiple threads, all executing at the same time. It is a unit of execution in concurrent programming. A thread is lightweight and can be managed independently by a scheduler. It helps you to improve the application performance using parallelism.

Multiple threads share information like data, code, files, etc. We can implement threads in three different ways:

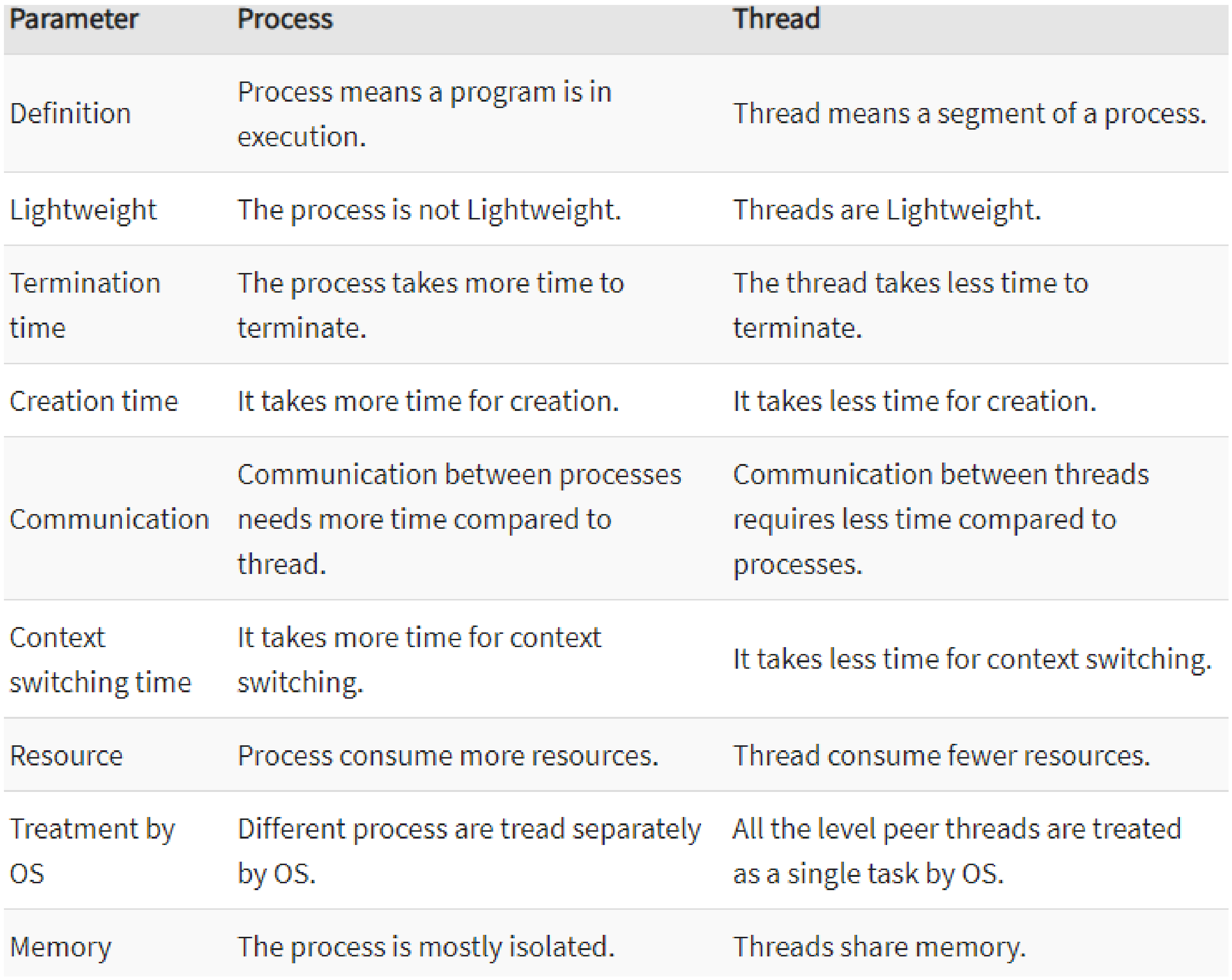
1. Kernel-level threads
2. User-level threads
3. Hybrid threads

Each thread belongs to exactly one process and no thread can exist outside a process. Each thread represents a separate flow of control. Threads have been successfully used in implementing network servers and web server. They also provide a suitable foundation for parallel execution of applications on shared memory multiprocessors. Following figure shows the working of the single and multithreaded processes.



KEY DIFFERENCE

- Process means a program is in execution, whereas thread means a segment of a process.
- A Process is not Lightweight, whereas Threads are Lightweight.
- A Process takes more time to terminate, and the thread takes less time to terminate.
- Process takes more time for creation, whereas Thread takes less time for creation.
- Process likely takes more time for context switching whereas as Threads takes less time for context switching.
- A Process is mostly isolated, whereas Threads share memory.
- Process does not share data, and Threads share data with each other.



Parameter	Process	Thread
Definition	Process means a program is in execution.	Thread means a segment of a process.
Lightweight	The process is not Lightweight.	Threads are Lightweight.
Termination time	The process takes more time to terminate.	The thread takes less time to terminate.
Creation time	It takes more time for creation.	It takes less time for creation.
Communication	Communication between processes needs more time compared to thread.	Communication between threads requires less time compared to processes.
Context switching time	It takes more time for context switching.	It takes less time for context switching.
Resource	Process consume more resources.	Thread consume fewer resources.
Treatment by OS	Different process are tread separately by OS.	All the level peer threads are treated as a single task by OS.
Memory	The process is mostly isolated.	Threads share memory.

Q2: List and discuss few types of thread.

Answer: Types of Thread

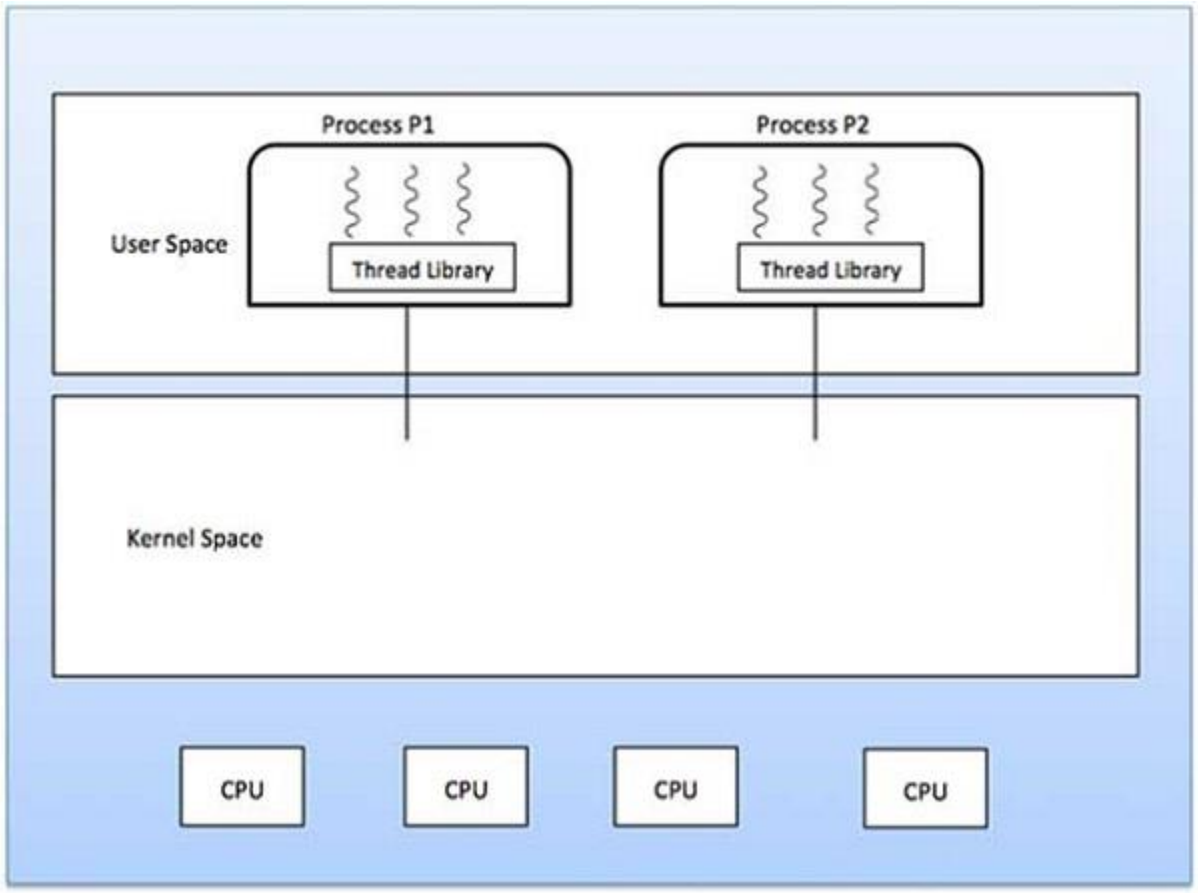
Threads are implemented in following two ways −

- **User Level Threads** − User managed threads.

- **Kernel Level Threads −** Operating System managed threads acting on kernel, an operating system core.

User Level Threads

In this case, the thread management kernel is not aware of the existence of threads. The thread library contains code for creating and destroying threads, for passing message and data between threads, for scheduling thread execution and for saving and restoring thread contexts. The application starts with a single thread.



Advantages

- Thread switching does not require Kernel mode privileges.
- User level thread can run on any operating system.
- Scheduling can be application specific in the user level thread.
- User level threads are fast to create and manage.

Disadvantages

- In a typical operating system, most system calls are blocking.
- Multithreaded application cannot take advantage of multiprocessing.

Kernel Level Threads

In this case, thread management is done by the Kernel. There is no thread management code in the application area. Kernel threads are supported directly by the operating system. Any application can be programmed to be multithreaded. All of the threads within an application are supported within a single process.

The Kernel maintains context information for the process as a whole and for individuals threads within the process. Scheduling by the Kernel is done on a thread basis. The Kernel performs thread creation, scheduling and management in Kernel space. Kernel threads are generally slower to create and manage than the user threads.

Advantages

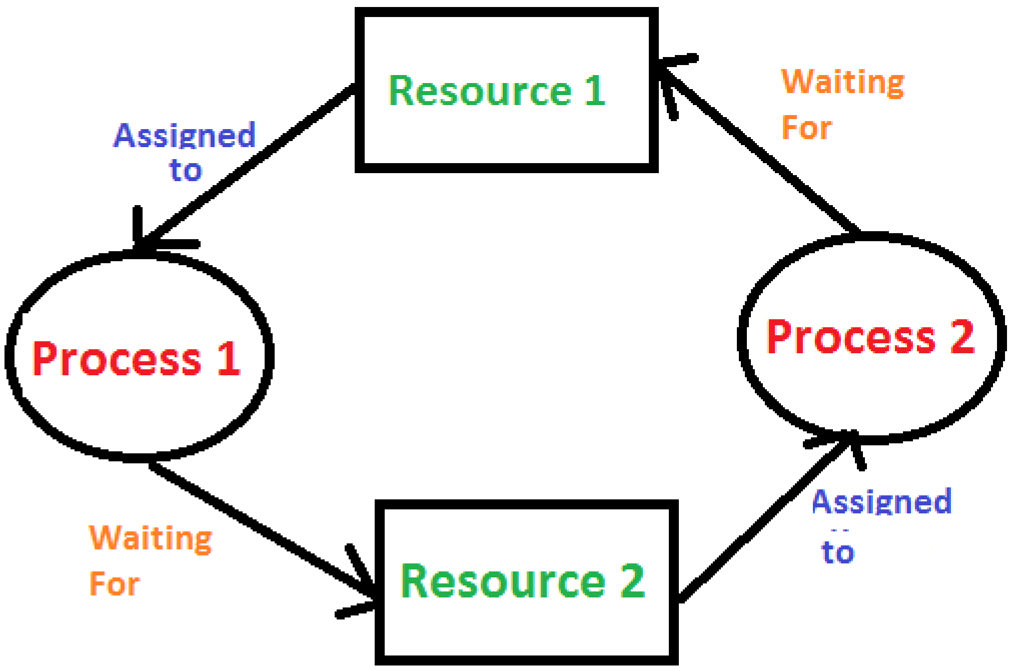
- Kernel can simultaneously schedule multiple threads from the same process on multiple processes.
- If one thread in a process is blocked, the Kernel can schedule another thread of the same process.
- Kernel routines themselves can be multithreaded.

Disadvantages

- Kernel threads are generally slower to create and manage than the user threads.
- Transfer of control from one thread to another within the same process requires a mode switch to the Kernel.

Q3: What is a deadlock? In what situations it occurs in an OS.

Answer: ***Deadlock*** is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process. Consider an example when two trains are coming toward each other on same track and there is only one track, none of the trains can move once they are in front of each other. Similar situation occurs in operating systems when there are two or more processes hold some resources and wait for resources held by other(s). For example, in the below diagram, Process 1 is holding Resource 1 and waiting for resource 2 which is acquired by process 2, and process 2 is waiting for resource 1.



Deadlock can arise if following four conditions hold simultaneously (Necessary Conditions)

Mutual Exclusive :- When a Single Process is used by two or more Processes, Means a Single Resource if used for performing the two or more activities as a Shared Based. But this is will Also Create a Problem because when a Second user Request for the System Resource which is being used by the use.

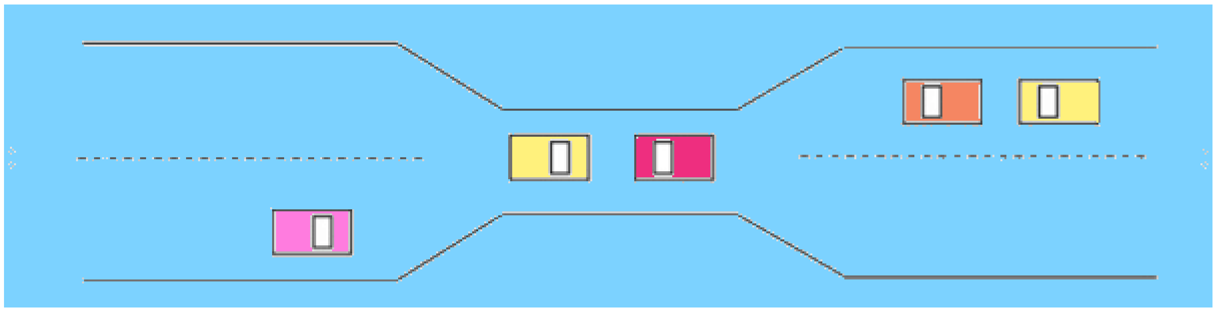
HOLD and Wait: - A Single Process may need two or more System Resources. And suppose if a Process have a Single Resource, and he is waiting the Second Resource. Then Process can't Leave the first Resource and waiting for the Second Resource. So that there will also be the Condition of Deadlock.

No Preemption: - if there is no Rule to use the System Resources. Means if all the System Resources are not allocated in the Manner of Scheduling. Then this will also create a Problem for a Deadlock because there is no surety that a Process will Release the System Resources after the Completion. **So, Preemption and Transaction Rollback prevents Deadlock situation**.

Circular Wait:- When two or More Requests are Waiting For a Long Period of Time and no one can Access the Resource from the System Resources , Then this is called as Circular Wait For Example if two or more users Request for a Printer, at a Same Time , they Request to Print a Page. Then they will be on the Circular Wait. Means System Will Display Busy Sign of Sign.

Example of Deadlock

- A real-world example would be traffic, which is going only in one direction.
- Here, a bridge is considered a resource.
- So, when Deadlock happens, it can be easily resolved if one car backs up (Preempt resources and rollback).
- Several cars may have to be backed up if a deadlock situation occurs.
- So starvation is possible.



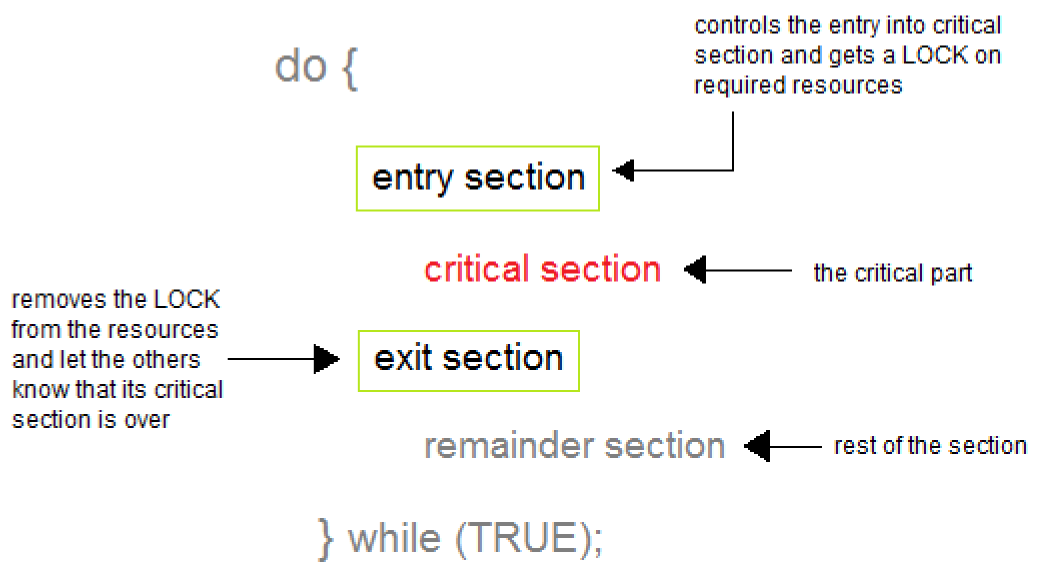
In what situations it occurs in an OS.

In an operating system, a deadlock occurs when a process or thread enters a waiting state because a requested system resource is held by another waiting process, which in turn is waiting for another resource held by another waiting process. If a process is unable to change its state indefinitely because the resources requested by it are being used by another waiting process, then the system is said to be in a deadlock.

Q4: Discuss a solution to the critical-section problem must satisfy the three requirements:

Answer: Critical Section Problem

A Critical Section is a code segment that accesses shared variables and has to be executed as an atomic action. It means that in a group of cooperating processes, at a given point of time, only one process must be executing its critical section. If any other process also wants to execute its critical section, it must wait until the first one finishes.


do {
controls the entry into critical section and gets a LOCK on required resources
entry section
critical section
the critical part
removes the LOCK from the resources and let the others know that its critical section is over
exit section
remainder section
rest of the section
} while (TRUE);

Solution to Critical Section Problem

A solution to the critical section problem must satisfy the following three conditions:

1. **Mutual Exclusion:** If process Pi is executing in its critical section, then no other processes can be executing in their critical sections.

2. **Progress:** If no process is executing in its critical section, and there are some processes that want to enter into their own critical sections, then the decision of whom to enter must not be postponed indefinitely.

3 Bounded Waiting: There exist a bound on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

Q5: Differentiate between dynamic loading and dynamic linking with example.

Answer:

Dynamic loading

Dynamic loading is a mechanism by which a computer program can, at run time, load a library into memory, retrieve the addresses of functions and variables contained in the library, execute those functions or access those variables, and unload the library from memory.

All the programs are loaded in the main memory for execution. Sometimes complete program is loaded into the memory, but sometimes a certain part or routine of the program is loaded into the main memory only when it is called by the program, this mechanism is called **Dynamic Loading**, this enhance the performance.

Dynamic linking refers to resolving symbols - associating their names with addresses or offsets - after compile time.

The reason it's hard to make a distinction is that the two are often done together without recognizing the subtle distinctions around the parts I put in bold.

Perhaps the clearest way to explain is to go through what the different combinations would mean in practice.

- **Dynamic loading, static linking.**

The executable has an address/offset table generated at compile time, but the actual code/data aren't loaded into memory at process start. This is not the way things tend to work in most systems nowadays, but it would describe some old-fashioned overlay systems. I'd also be utterly unsurprised if some current embedded systems work this way too. In either case, the goal is to give the programmer control over memory use while also avoiding the overhead of linking at runtime.

- **Static loading, dynamic linking.**

This is how dynamic libraries specified at compile time usually work. The executable contains a reference to the dynamic/shared library, but the symbol table is missing or incomplete. Both loading and linking occur at process start, which is considered "dynamic" for linking but not for loading.

- **Dynamic loading, dynamic linking.**

This is what happens when you call dlopen or its equivalent on other systems. The object file is loaded dynamically under program control (i.e. after start), and symbols both in the calling program and in the library are resolved based on the process's possibly-unique memory layout at that time.

• **Static loading, static linking.**

Everything is resolved at compile time. At process start everything is loaded into memory immediately and no extra resolution (linking) is necessary. In the abstract it's not necessary for the loading to occur from a single file, but I don't think the actual formats or implementations (at least those I'm familiar with) can do multi-file loading without dynamic linking.

Dynamic Linking

Dynamic linking consists of compiling and linking code into a form that is loadable by programs at run time as well as link time. The ability to load them at run time is what distinguishes them from ordinary object files. Various operating systems have different names for such loadable code:

• Dynamic Linking Loader is a general re-locatable loader.

• Allowing the programmer multiple procedure segments and multiple data segments and giving programmer complete freedom in referencing data or instruction contained in other segments.

• The assembler must give the loader the following information with each procedure or data segment.

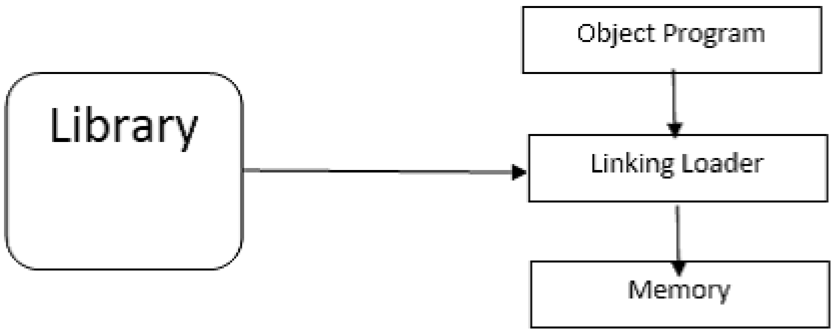
• Dynamic linking defers much of the linking process until a program starts running. It provides a variety of benefits that are hard to get otherwise:

• Dynamically linked shared libraries are easier to create than static linked shared libraries.

• Dynamically linked shared libraries are easier to update than static linked shared libraries.

• The semantics of dynamically linked shared libraries can be much closer to those of unshared libraries.

• Dynamic linking permits a program to load and unload routines at runtime, a facility that can otherwise be very difficult to provide.

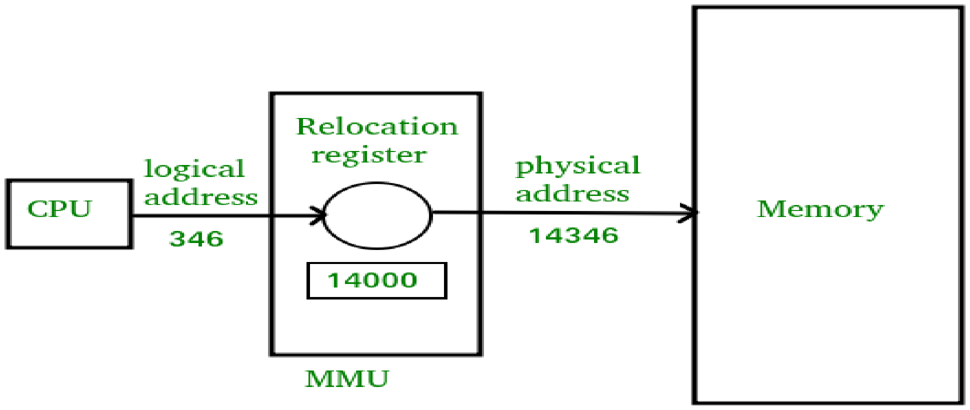


Q6: Write your understanding about logical Vs Physical address space?

Answer: Logical and Physical Address in Operating System

Logical Address: - Logical Address is generated by CPU while a program is running. The logical address is virtual address as it does not exist physically, therefore, it is also known as Virtual Address. This address is used as a reference to access the physical memory location by CPU. The term Logical Address Space is used for the set of all logical addresses generated by a program's perspective.
The hardware device called Memory-Management Unit is used for mapping logical address to its corresponding physical address.

Physical Address: - Physical Address identifies a physical location of required data in a memory. The user never directly deals with the physical address but can access by its corresponding logical address. The user program generates the logical address and thinks that the program is running in this logical address but the program needs physical memory for its execution, therefore, the logical address must be mapped to the physical address by MMU before they are used. The term Physical Address Space is used for all physical addresses corresponding to the logical addresses in a Logical address space.



Key Differences Between Logical and Physical Address in OS

1. The basic difference between Logical and physical address is that Logical address is generated by CPU in perspective of a program. On the other hand, the physical address is a location that exists in the memory unit.
2. The set of all logical addresses generated by CPU for a program is called Logical Address Space. However, the set of all physical address mapped to corresponding logical addresses is referred as Physical Address Space.
3. The logical address is also called virtual address as the logical address does not exist physically in the memory unit. The physical address is a location in the memory unit that can be accessed physically.
4. Identical logical address and physical address are generated by Compile-time and Load time address binding methods.
5. The logical and physical address generated while run-time address binding method differs from each other.
6. The logical address is generated by the CPU while program is running whereas, the physical address is computed by the MMU (Memory Management Unit).