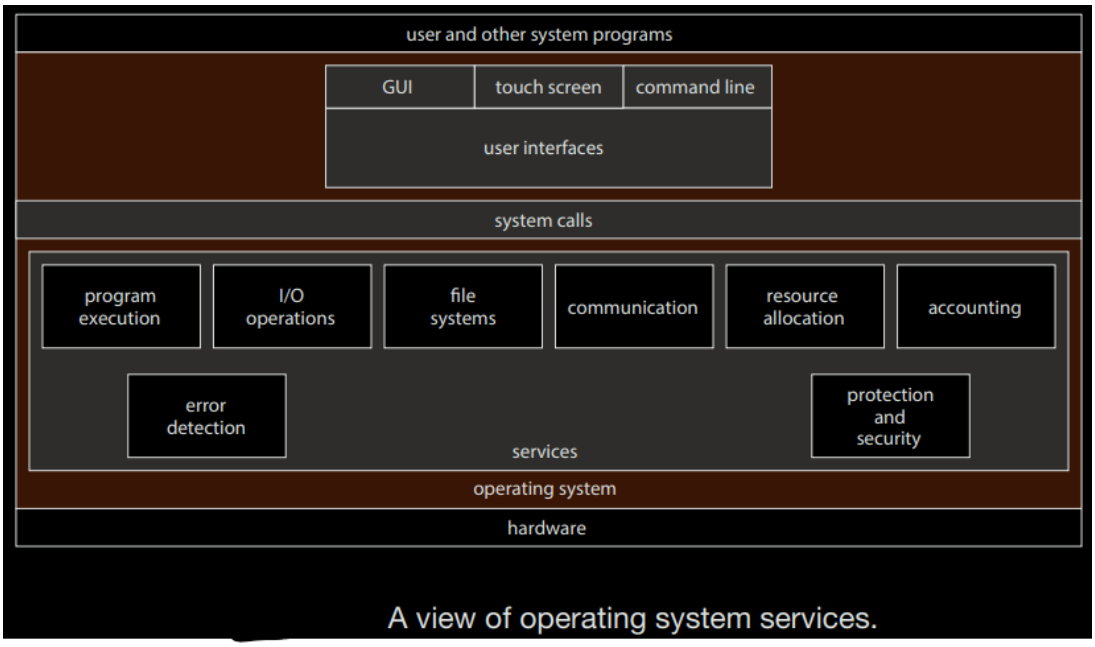
Preset:

Operating System: An Operating System (OS) is a collection of software that manages computer hardware, provides basis for application programs and acts as an intermediary between the user and computer.

Types of OS:

- Monolithic OS: the entire OS is working in kernel space and is alone in supervisor mode
- Modular OS: Part of the system core will be located in independent files called modules that can be added to the system at run time
- Micro OS: The kernel is broken down into separate processes, known as servers. Some of the servers run in kernel space and some run in user-space.

General Functions:
-Hide Hardware Complexity
-Resource Management
-Provide Isolation and Protection



A view of operating system services.

OS Elements:
-Abstractions:
-Process
-Thread
-File
-Socket
-Memory Page.
-Mechanisms:
-Create
-Schedule
-Open
-Write
-Allocate...
-Policies:

-Least-recently used (LRU)
-Earliest deadline first (EDF)

Interrupt: An interrupt is a signal or event generated by hardware or software to interrupt the normal flow of execution of a processor. They transfer control to the corresponding **interrupt service routine (ISR).**

Hardware Interrupts: These are generated by external hardware devices like a keyboard, mouse, timer, disk drives, network interface cards (NICs), etc., to signal that they need attention or service from the CPU.

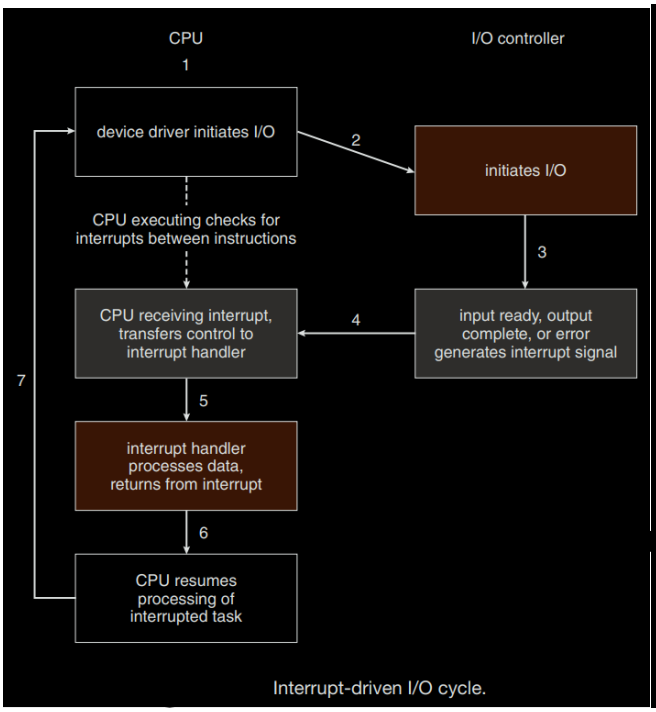
Software Interrupts: These are generated by software or programs to request services from the operating system. For example, system calls made by user programs, such as I/O operations or other OS services, generate software interrupts.

Mechanism:

- **Interrupt Vector Table:** A table of pointers to ISR routines for various devices, enabling rapid access to specific interrupt routines.
- Indexed by a unique number provided with the interrupt request.
- Stores addresses of ISRs in low memory.

Handling Process:

- **Interrupt Handling:** Hardware saves the state information of the interrupted process for restoration after servicing the interrupt.
- ISR modifies the processor state, saves the current state, and restores it before returning control.
- CPU detects interrupts via the interrupt-request line and jumps to the ISR using the interrupt number as an index in the interrupt vector.
- Interrupt handler determines the cause, processes, restores state, and returns to the interrupted task.
- Deferring Interrupt Handling: Critical processing allows deferral of interrupts.
- Efficient Dispatching: Need an efficient method to direct interrupts to the correct handler.
- Multilevel Interrupts: Distinguishing between high and low-priority interrupts is essential for different levels of urgency.

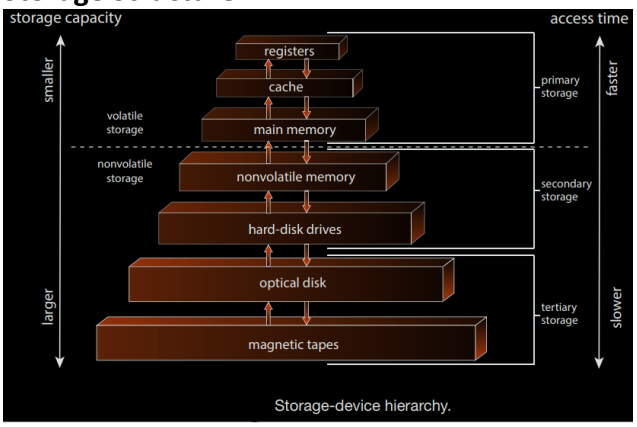


Interrupt-driven I/O cycle.

vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19–31	(Intel reserved, do not use)
32–255	maskable interrupts

Figure 1.5 Intel processor event-vector table.

Storage Structure:

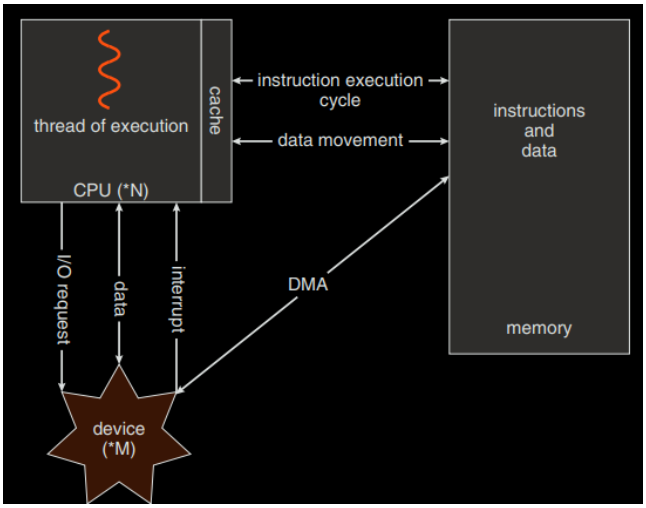


Storage-device hierarchy.

Level	1	2	3	4	5
Name	registers	cache	main memory	solid-state disk	magnetic disk
Typical size	< 1 KB	< 16MB	< 64GB	< 1 TB	< 10 TB
Implementation technology	custom memory with multiple ports CMOS	on-chip or off-chip CMOS SRAM	CMOS SRAM	flash memory	magnetic disk
Access time (ns)	0.25-0.5	0.5-25	80-250	25,000-50,000	5,000,000
Bandwidth (MB/sec)	20,000-100,000	5,000-10,000	1,000-5,000	500	20-150
Managed by	compiler	hardware	operating system	operating system	operating system
Backed by	cache	main memory	disk	disk	disk or tape

Direct Memory Access (DMA):

- Interrupt-driven I/O is suitable for small data movements but causes high overhead in bulk data transfers
- DMA resolves this issue by allowing the device controller to directly transfer entire blocks of data between devices and main memory.
- DMA setup involves configuring buffers, pointers, and counters for I/O devices.
- Only one interrupt per block is generated upon completion, reducing CPU intervention and enhancing efficiency.
- While the device controller manages data transfers via DMA, the CPU is free to perform other tasks, optimizing system resource utilization.



Computer System Organization:

Single-Processor Systems:

- Historically, most systems had a single processor containing one CPU with a single processing core.
- Special-purpose processors (e.g., disk, keyboard, graphics controllers) ran limited instruction sets and were managed by the OS.
- Few contemporary systems strictly fall under the category of single-processor systems due to diverse architectures.

Multiprocessor Systems:

- Modern systems, from mobile devices to servers, predominantly use multiprocessor systems.
- These systems have two or more processors, each with a single-core CPU, sharing resources like memory, peripheral devices, and a bus.
- Increased processors lead to improved throughput but with diminishing returns due to overhead and shared resource contention.
- Symmetric multiprocessing (SMP) is common, where peer CPUs perform tasks independently, yet inefficiencies can occur due to varied workloads.

Evolution to Multicore Systems:

- The definition of multiprocessor has expanded to include multicore systems where multiple cores reside on a single chip.
- Multicore systems enhance efficiency by enabling faster on-chip communication and lower power consumption compared to single-core chips.
- Operating systems need to efficiently manage multiple cores, and application programs should leverage this architecture for optimal performance.

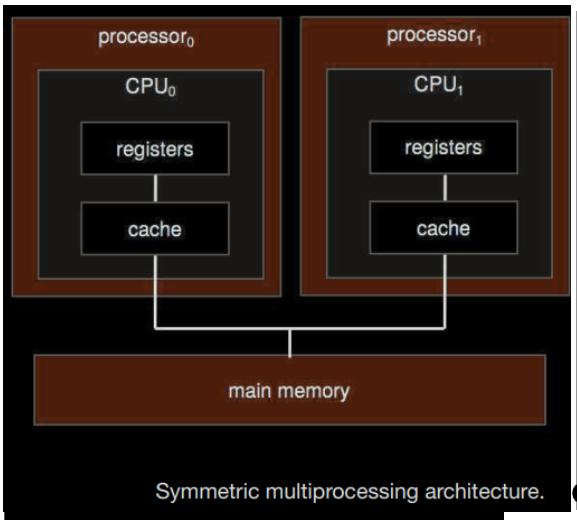
Non-Uniform Memory Access (NUMA):

- NUMA systems offer local memory to CPUs with faster access but may face latency issues when accessing remote memory.
- CPU scheduling and memory management are crucial in NUMA systems to mitigate performance penalties.

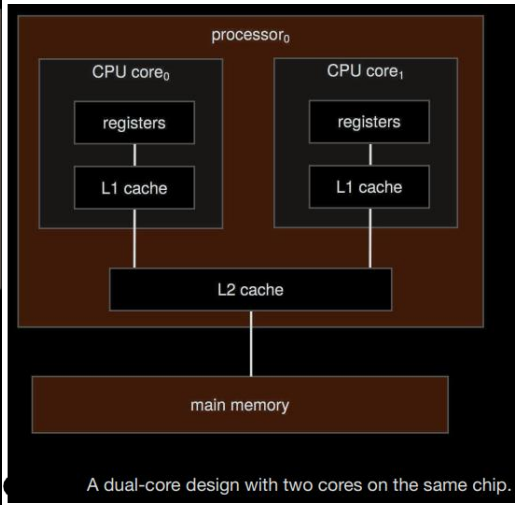
Clustered Systems:

- Clusters, composed of multiple CPUs or nodes, offer high availability and reliability by sharing storage and running applications redundantly.
- They can scale performance through parallelization of applications across multiple computers in the cluster.
- Rapid advancements in cluster technology support thousands of systems, distributed over miles, facilitated by storage-area networks (SANs) for increased performance and reliability.
- SANs enable shared access to applications and data among multiple hosts, boosting performance and reliability significantly.

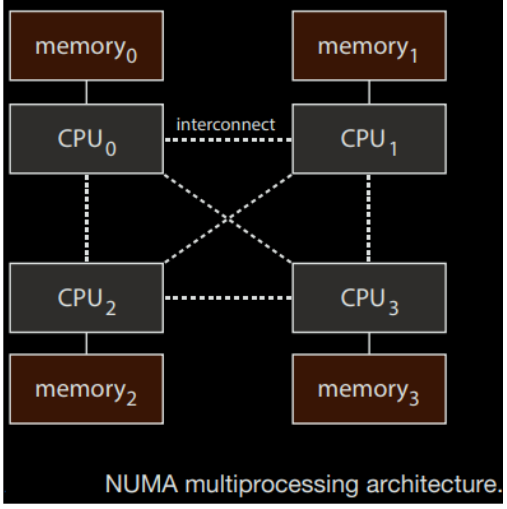
Blade servers are systems in which multiple processor boards, I/O boards, and networking boards are placed in the same chassis. The difference between these and traditional multiprocessor systems is that each blade processor board boots independently and runs its own operating system.



Symmetric multiprocessing architecture.



A dual-core design with two cores on the same chip.



NUMA multiprocessing architecture.

OS Operations:

- **Multiprogramming** achieves concurrency by time-sharing a single processor among multiple programs.
- **Multiprocessing** achieves concurrency by running multiple processes simultaneously on multiple processors or cores.
- **Dual Mode:** Ensuring system integrity by distinguishing between user mode and kernel mode. There is a mode bit that differentiates these two modes User mode (1) and kernel mode (0). Mode Transition occurs during system calls, traps, or interrupts.
- Intel has up to 3 modes in certain processors while ARM can have up to 7 modes for granular control. Virtualization machine manager (VMM) uses these modes.
- Timers: Prevent user programs from causing system hang-ups by using a timer mechanism. Jiffies Variable: Represents the number of timer interrupts since system boot.

System Calls:

- System calls provide a standardized interface for user-level processes to request services from the operating system.
- Boundary Crossing: They facilitate the transition from user mode to kernel mode, enabling user programs to access privileged OS functionalities.
- Types of System Calls: They cover a wide range of functionalities, including file management, process control, device management, memory management, and networking.
- Invocation Methods: System calls can be invoked through various mechanisms, such as trap instructions, software interrupts, or specialized instructions (e.g., syscall instruction).

- Request Handling: When a system call is invoked, control transfers to a specific location in the interrupt vector, leading to a service routine in the operating system.
- Parameters and Results: System calls require specific parameters passed via registers, stacks, or memory. The OS verifies the parameters, executes the request, and returns results or error codes.

	Windows	Unix
Process control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File management	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device management	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communications	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shm_open() mmap()
Protection	SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

Linux Boot Process:

Step 1 - When we turn on the power, BIOS (Basic Input/Output System) or UEFI (Unified Extensible Firmware Interface) firmware is loaded from non-volatile memory, and executes POST (Power On Self Test).

Step 2 - BIOS/UEFI detects the devices connected to the system, including CPU, RAM, and storage.

Step 3 - Choose a booting device to boot the OS from. This can be the hard drive, the network server, or CD ROM.

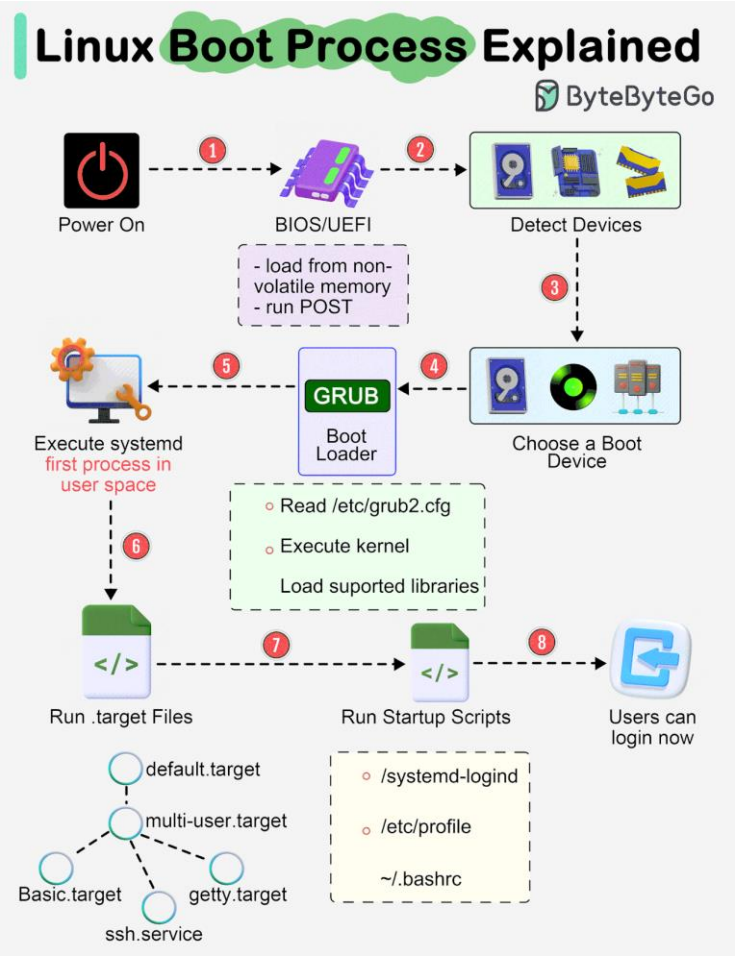
Step 4 - BIOS/UEFI runs the boot loader (GRUB), which provides a menu to choose the OS or the kernel functions.

Step 5 - After the kernel is ready, we now switch to the user space. The kernel starts up systemd as the first user-space process, which manages the processes and services, probes all remaining hardware, mounts filesystems, and runs a desktop environment.

Step 6 - systemd activates the default. target unit by default when the system boots. Other analysis units are executed as well.

Step 7 - The system runs a set of startup scripts and configure the environment.

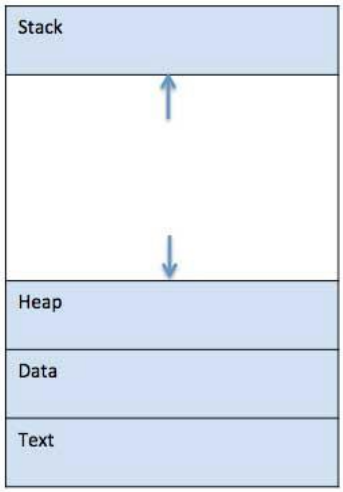
Step 8 - The users are presented with a login window. The system is now ready.



Processes and Process Management:

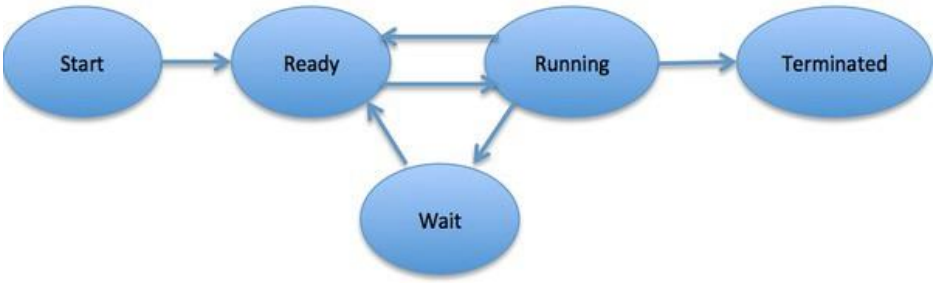
A process is basically a program in execution. The execution of a process must progress in a sequential fashion. To put it in simple terms, we write our computer programs in a text file and when we execute this program, it becomes a process which performs all the tasks mentioned in the program.

When a program is loaded into the memory and it becomes a process, it can be divided into four sections – stack, heap, text and data. The following image shows a simplified layout of a process inside main memory



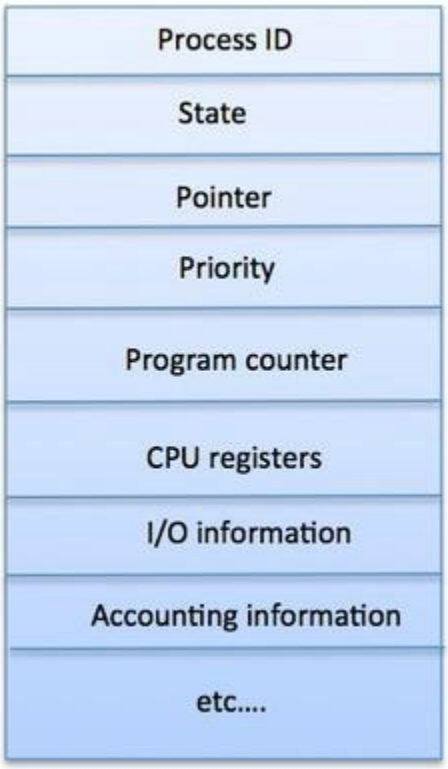
- Stack: The process Stack contains the temporary data such as method/function parameters, return address and local variables.
- Heap: This is dynamically allocated memory to a process during its run time.
- Text: This includes the current activity represented by the value of Program Counter and the contents of the processor's registers.
- Data: This section contains the global and static variables.

When a process executes, it passes through different states. These stages may differ in different operating systems, and the names of these states are also not standardized. In general, a process can have one of the following five states at a time:



- Start: This is the initial state when a process is first started/created.
- Ready: The process is waiting to be assigned to a processor. Ready processes are waiting to have the processor allocated to them by the operating system so that they can run. Process may come into this state after Start state or while running it by but interrupted by the scheduler to assign CPU to some other process.
- Running: Once the process has been assigned to a processor by the OS scheduler, the process state is set to running and the processor executes its instructions.
- Waiting: Process moves into the waiting state if it needs to wait for a resource, such as waiting for user input, or waiting for a file to become available.
- Terminated or Exit: Once the process finishes its execution, or it is terminated by the operating system, it is moved to the terminated state where it waits to be removed from main memory.

A Process Control Block is a data structure maintained by the Operating System for every process. The PCB is identified by an integer process ID (PID).



- Process State: The current state of the process i.e., whether it is ready, running, waiting, or whatever.
- Process Privileges: This is required to allow/disallow access to system resources.
- Process ID: Unique identification for each of the process in the operating system.
- Pointer: A pointer to parent process.
- Program Counter: Program Counter is a pointer to the address of the next instruction to be executed for this process.
- CPU Registers: Various CPU registers where process need to be stored for execution for running state.
- CPU Scheduling Information: Process priority and other scheduling information which is required to schedule the process.
- Memory Management Information: This includes the information of page table, memory limits, Segment table depending on memory used by the operating system.
- Accounting Information: This includes the amount of CPU used for process execution, time limits, execution ID etc.
- IO Status Information: This includes a list of I/O devices allocated to the process.

Zombie Process: Also known as defunct process, is basically a process that is terminated or completed but the PCB is not cleaned up from main memory as it still has an entry in the process table to report to its parent process.

Cascading Termination: It is a process termination in which if the parent process is existing or terminating then the children process will also get terminated.

Deadlocks:

- Deadlock is generally a situation where a set of processes are blocked as each process is holding resources and waits to acquire resources held by another process.

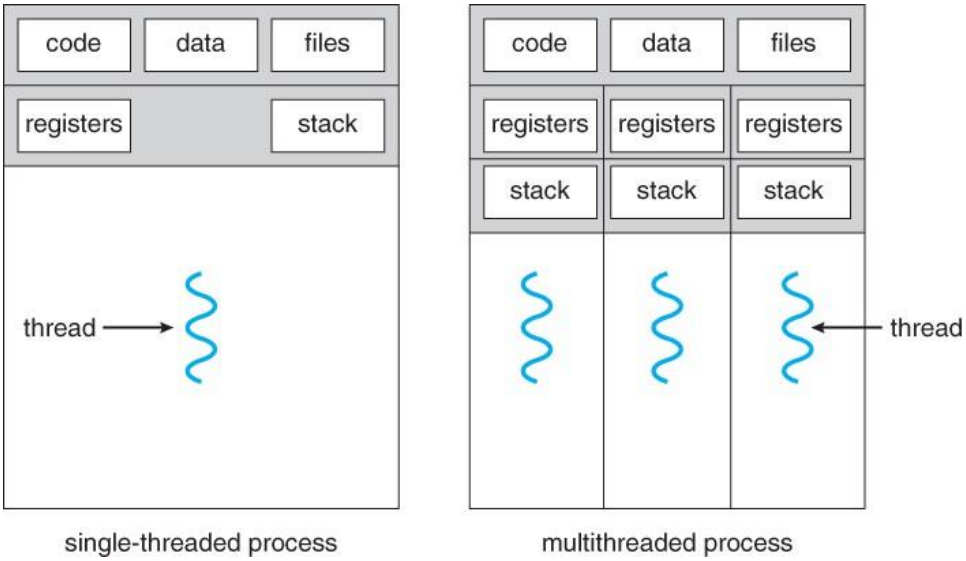
- In this situation, two or more processes simply try to execute simultaneously and wait for each to finish their execution because they are dependent on each other. We can see a hand problem in our system whenever a deadlock occurs in a program. It is one of the common problems you can see in multiprocessing.
- Necessary conditions for Deadlock
    - Mutual Exclusion
    - Hold and Wait
    - No Pre-emption
    - Circular Wait or Resource Wait

Threads & Concurrency:

- A thread is a flow of execution through the process code, with its own program counter that keeps track of which instruction to execute next, system registers which hold its current working variables, and a stack which contains the execution history.
- A thread shares with its peer threads few information like code segment, data segment and open files. When one thread alters a code segment memory item, all other threads see that.
- A thread is also called a lightweight process. Threads provide a way to improve application performance through parallelism. Threads represent a software approach to improving performance of operating system by reducing the overhead thread is equivalent to a classical process.
- 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.

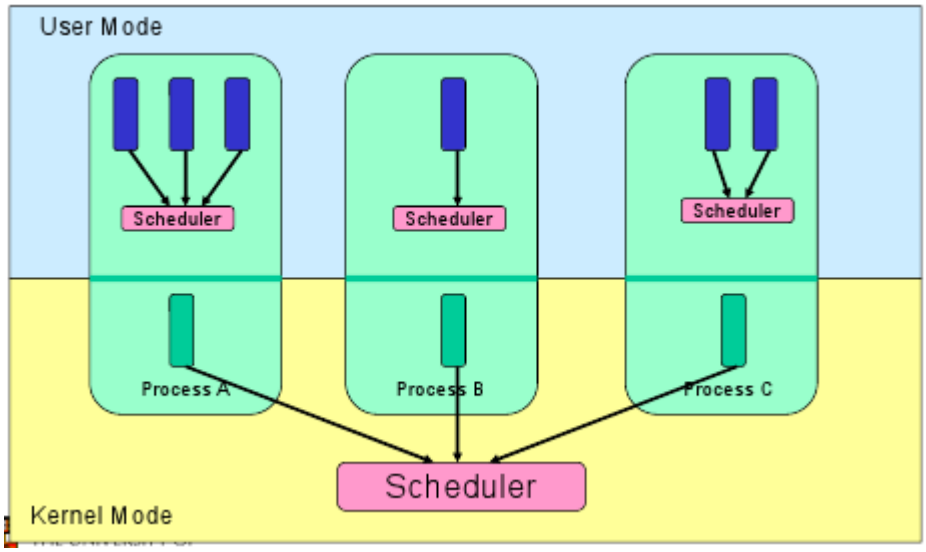
Advantages of Thread:

- Threads minimize the context switching time.
- Use of threads provides concurrency within a process.
- Efficient communication.
- It is more economical to create and context switch threads.
- Threads allow utilization of multiprocessor architectures to a greater scale and efficiency.



Threads are implemented in the following 2 ways:

1. **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.

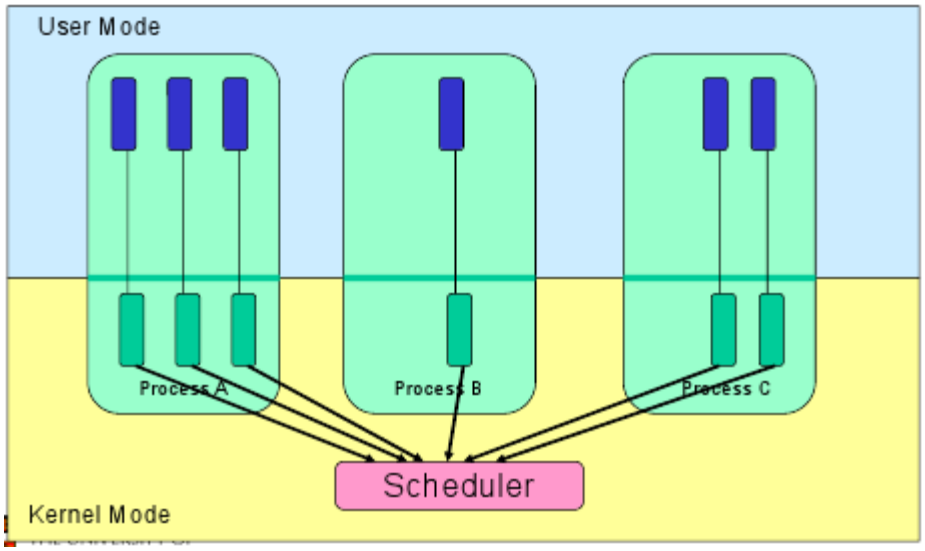


2. **Kernel Level Threads:**

   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.



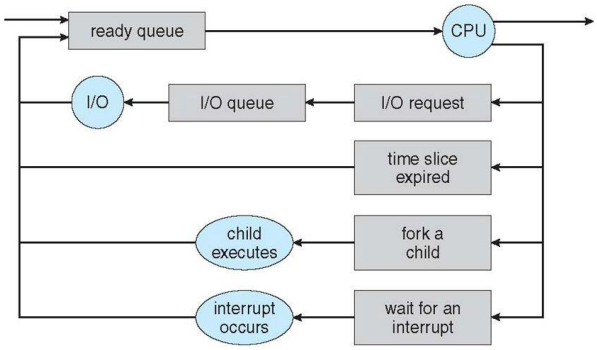
Scheduling:

- The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.
- Process scheduling is an essential part of a Multiprogramming operating systems. Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.
- The OS maintains all Process Control Blocks (PCBs) in Process Scheduling Queues. The OS maintains a separate queue for each of the process states and PCBs of all processes in the same execution state are placed in the same queue. When the state of a process is changed, its PCB is unlinked from its current queue and moved to its new state queue.
- The Operating System maintains the following important process scheduling queues:
    - Job queue − This queue keeps all the processes in the system.
    - Ready queue − This queue keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue.
    - Device queues − The processes which are blocked due to unavailability of an I/O device constitute this queue.

- The OS can use different policies to manage each queue (FIFO, Round Robin, Priority, etc.). The OS scheduler determines how to move processes between the ready and run queues which can only have one entry per processor core on the system; in the above diagram, it has been merged with the CPU.
- Two-state process model refers to running and non-running states:
    - Running: When a new process is created, it enters into the system as in the running state.
    - Not Running: Processes that are not running are kept in queue, waiting for their turn to execute. Each entry in the queue is a pointer to a particular process. Queue is implemented by using linked list. Use of dispatcher is as follows. When a process is interrupted, that process is transferred in the waiting queue. If the process has completed or aborted, the process is discarded. In either case, the dispatcher then selects a process from the queue to execute.

Process Scheduling Queues

A process migrates among the queues throughout its life:



Context Switching:

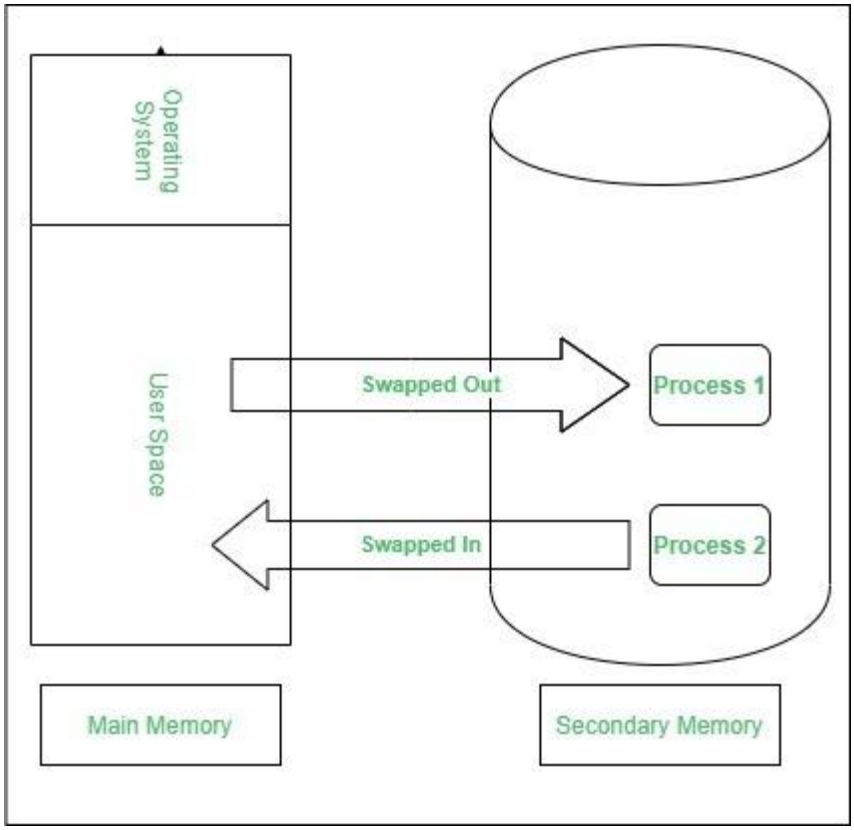
- A context switch is the mechanism to store and restore the state or context of a CPU in Process Control block so that a process execution can be resumed from the same point at a later time. Using this technique, a context switcher enables multiple processes to share a single CPU. Context switching is an essential part of a multitasking operating system features.
- When the scheduler switches the CPU from executing one process to execute another, the state from the current running process is stored into the process control block. After this, the state for the process to run next is loaded from its own PCB and used to set the PC, registers, etc. At that point, the second process can start executing.
- Context switches are computationally intensive since register and memory state must be saved and restored. To avoid the amount of context switching time, some hardware systems employ two or more sets of processor registers. When the process is switched, the following information is stored for later use: Program Counter, Scheduling Information, Base and Limit Register Value, Currently Used Register, Changed State, I/O State Information, and Accounting Information.

Memory Management

- Main Memory: The main memory is central to the operation of a modern computer. Main Memory is a large array of words or bytes, ranging in size from hundreds of thousands to billions. Main memory is a repository of rapidly available information shared by the CPU and I/O devices. Main memory is the place where programs and information are kept when the processor is effectively utilizing them. Main memory is associated with the processor, so moving instructions and information into and out of the processor is extremely fast. Main memory is also known as RAM(Random Access Memory). This memory is a volatile memory.
- Memory Management: In a multiprogramming computer, the operating system resides in a part of memory and the rest is used by multiple processes. The task of subdividing the memory among different processes is called memory management. Memory management is a method in the operating system to manage operations between main memory and disk during process execution. The main aim of memory management is to achieve efficient utilization of memory.
- Why Memory Management is required:
    - Allocate and de-allocate memory before and after process execution.
    - To keep track of used memory space by processes.
    - To minimize fragmentation issues.
    - To proper utilization of main memory.
    - To maintain data integrity while executing of process.

- **Logical Address space:** An address generated by the CPU is known as “Logical Address”. It is also known as a Virtual address. Logical address space can be defined as the size of the process. A logical address can be changed. A logical address is generated so that a user program never directly access the physical memory and the process donot occupies memory which is acquired by another process thus corrupting that process.
- **Physical Address space:** An address seen by the memory unit (i.e the one loaded into the memory address register of the memory) is commonly known as a “Physical Address”. A Physical address is also known as a Real address. The set of all physical addresses corresponding to these logical addresses is known as Physical address space. A physical address is computed by MMU. The run-time mapping from virtual to physical addresses is done by a hardware device Memory Management Unit(MMU). The physical address always remains constant.
- Virtual and physical addresses are the same in compile-time and load-time address-binding schemes. Virtual and physical addresses differ in execution-time address-binding scheme.
- The process address space is the set of logical addresses that a process references in its code. For example, when 32-bit addressing is in use, addresses can range from 0 to 0x7fffffff; that is, 2^{31} possible numbers, for a total theoretical size of 2 gigabytes.
- The operating system takes care of mapping the logical addresses to physical addresses at the time of memory allocation to the program. There are three types of addresses used in a program before and after memory is allocated:
    - Symbolic addresses: The addresses used in a source code. The variable names, constants, and instruction labels are the basic elements of the symbolic address space.
    - Relative addresses: At the time of compilation, a compiler converts symbolic addresses into relative addresses.
    - Physical addresses: The loader generates these addresses at the time when a program is loaded into main memory.
- **Loading:** To load a process into the main memory is done by a loader. There are two different types of loading:
    - Static loading:- loading the entire program into a fixed address. It requires more memory space.
    - Dynamic loading:- The entire program and all data of a process must be in physical memory for the process to execute. So, the size of a process is limited to the size of physical memory. To gain proper memory utilization, dynamic loading is used. In dynamic loading, a routine is not loaded until it is called. All routines are residing on disk in a relocatable load format. One of the advantages of dynamic loading is that unused routine is never loaded. This loading is useful when a large amount of code is needed to handle it efficiently.
- **Linking:** To perform a linking task a linker is used. A linker is a program that takes one or more object files generated by a compiler and combines them into a single executable file.
    - Static linking: In static linking, the linker combines all necessary program modules into a single executable program. So there is no runtime dependency. Some operating systems support only static linking, in which system language libraries are treated like any other object module.
    - Dynamic linking: The basic concept of dynamic linking is similar to dynamic loading. In dynamic linking, “Stub” is included for each appropriate library routine reference. A stub is a small piece of code. When the stub is executed, it checks whether the needed routine is already in memory or not. If not available then the program loads the routine into memory.
- Swapping: When a process is executed it must have resided in memory. Swapping is a process of swap a process temporarily into a secondary memory from the main memory, which is fast as compared to secondary memory. A swapping allows more processes to be run and can be fit into memory at one time. The main part of swapping is transferred time and the total time directly proportional to the amount of memory swapped. Swapping is also known as roll-out, roll in, because if a higher priority process arrives and wants service, the memory manager can swap out the lower priority process and then load and

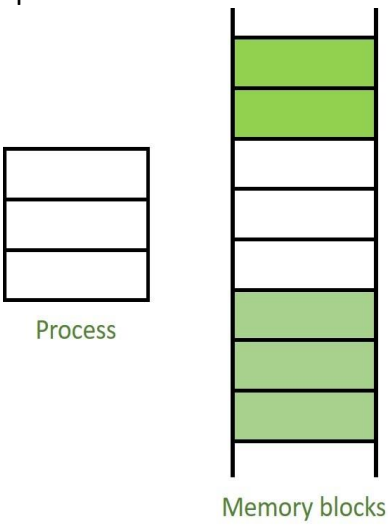
execute the higher priority process. After finishing higher priority work, the lower priority process swapped back in memory and continued to the execution process.



Memory Allocation:

1. Contiguous Memory Allocation :

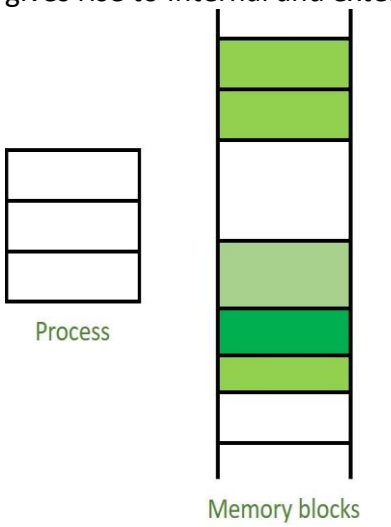
- Contiguous memory allocation is basically a method in which a single contiguous section/part of memory is allocated to a process or file needing it. Because of this all the available memory space resides at the same place together, which means that the freely/unused available memory partitions are not distributed in a random fashion here and there across the whole memory space.



Contiguous Memory Allocation

- The main memory is a combination of two main portions- one for the operating system and other for the user program. We can implement/achieve contiguous memory allocation by dividing the memory partitions into fixed size partitions.

- **2. Non-Contiguous Memory Allocation :**
  - Non-Contiguous memory allocation is basically a method on the contrary to contiguous allocation method, allocates the memory space present in different locations to the process as per it's requirements. As all the available memory space is in a distributed pattern so the freely available memory space is also scattered here and there.
  - This technique of memory allocation helps to reduce the wastage of memory, which eventually gives rise to Internal and external fragmentation.
  -

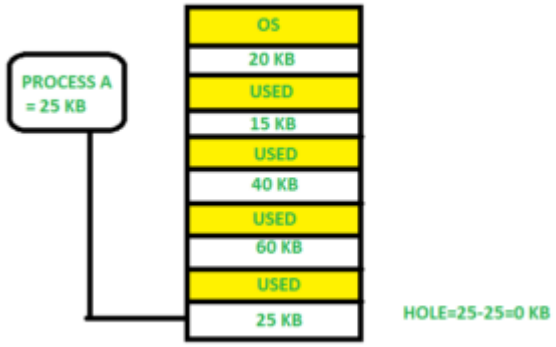


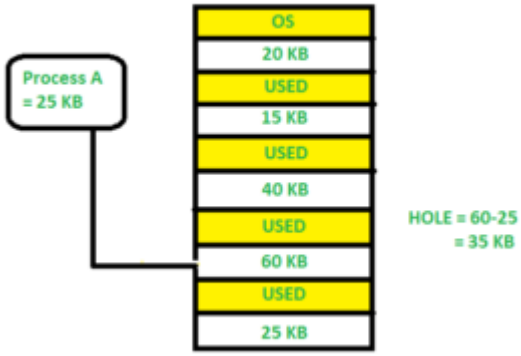
Noncontiguous Memory Allocation

Memory allocation:

- To gain proper memory utilization, memory allocation must be allocated efficient manner. One of the simplest methods for allocating memory is to divide memory into several fixed-sized partitions and each partition contains exactly one process. Thus, the degree of multiprogramming is obtained by the number of partitions.
- Multiple partition allocation: In this method, a process is selected from the input queue and loaded into the free partition. When the process terminates, the partition becomes available for other processes.
- **Fixed partition allocation:** In this method, the operating system maintains a table that indicates which parts of memory are available and which are occupied by processes. Initially, all memory is available for user processes and is considered one large block of available memory. This available memory is known as "Hole". When the process arrives and needs memory, we search for a hole that is large enough to store this process. If the requirement fulfills then we allocate memory to process, otherwise keeping the rest available to satisfy future requests. While allocating a memory sometimes dynamic storage allocation problems occur, which concerns how to satisfy a request of size n from a list of free holes. There are some solutions to this problem:
- First Fit:

- 

- Best Fit:
- 

- Worst Fit:
- 

Fragmentation:

- A Fragmentation is defined as when the process is loaded and removed after execution from memory, it creates a small free hole. These holes can not be assigned to new processes because holes are not combined or do not fulfill the memory requirement of the process. To achieve a degree of multiprogramming, we must reduce the waste of memory or fragmentation problem. In operating system two types of fragmentation:
- Internal fragmentation:

- Internal fragmentation occurs when memory blocks are allocated to the process more than their requested size. Due to this some unused space is leftover and creates an internal fragmentation problem.
- Example: Suppose there is a fixed partitioning is used for memory allocation and the different size of block 3MB, 6MB, and 7MB space in memory. Now a new process p4 of size 2MB comes and demand for the block of memory. It gets a memory block of 3MB but 1MB block memory is a waste, and it can not be allocated to other processes too. This is called internal fragmentation.
- External fragmentation:
- In external fragmentation, we have a free memory block, but we can not assign it to process because blocks are not contiguous.
- Example: Suppose (consider above example) three process p1, p2, p3 comes with size 2MB, 4MB, and 7MB respectively. Now they get memory blocks of size 3MB, 6MB, and 7MB allocated respectively. After allocating process p1 process and p2 process left 1MB and 2MB. Suppose a new process p4 comes and demands a 3MB block of memory, which is available, but we can not assign it because free memory space is not contiguous. This is called external fragmentation.
- Both the first fit and best-fit systems for memory allocation affected by external fragmentation. To overcome the external fragmentation problem Compaction is used. In the compaction technique, all free memory space combines and makes one large block. So, this space can be used by other processes effectively.
- Another possible solution to the external fragmentation is to allow the logical address space of the processes to be non-contiguous, thus permit a process to be allocated physical memory where ever the latter is available.

Paging:

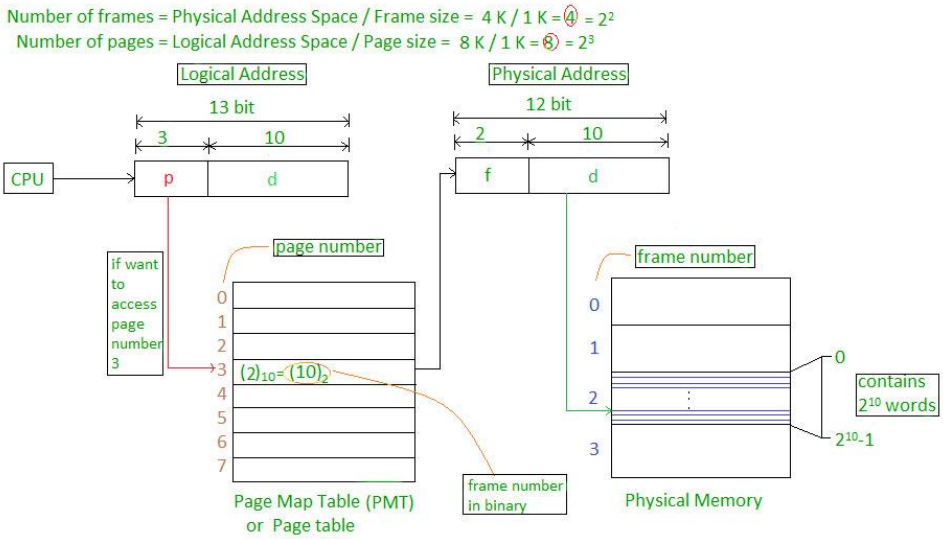
- Paging is a memory management scheme that eliminates the need for contiguous allocation of physical memory. This scheme permits the physical address space of a process to be non-contiguous.
    - Logical Address or Virtual Address (represented in bits): An address generated by the CPU
    - Logical Address Space or Virtual Address Space (represented in words or bytes): The set of all logical addresses generated by a program
    - Physical Address (represented in bits): An address actually available on a memory unit
    - Physical Address Space (represented in words or bytes): The set of all physical addresses corresponding to the logical addresses

- Example:
- If Logical Address = 31 bits, then Logical Address Space = 231 words = 2 G words (1 G = 230)
- If Logical Address Space = 128 M words = 27 * 220 words, then Logical Address = log2 227 = 27 bits
- If Physical Address = 22 bits, then Physical Address Space = 222 words = 4 M words (1 M = 220)
- If Physical Address Space = 16 M words = 24 * 220 words, then Physical Address = log2 224 = 24 bits

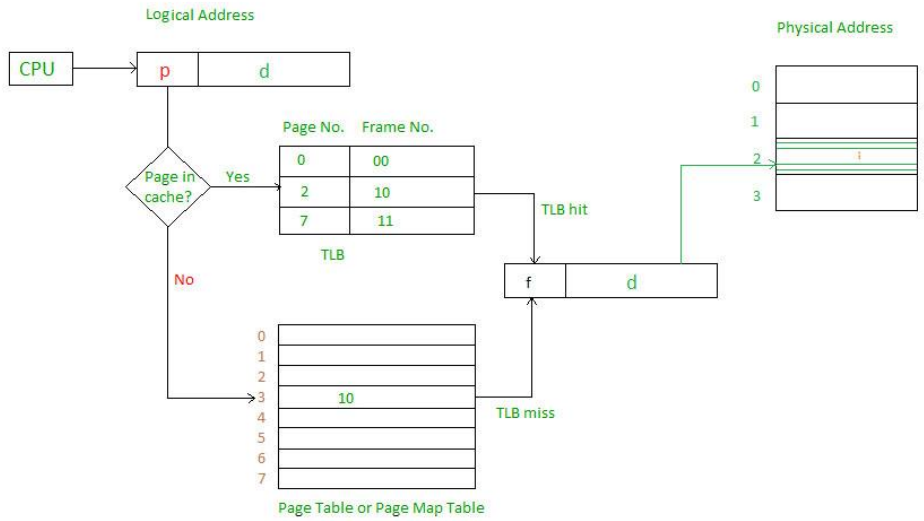
- The mapping from virtual to physical address is done by the memory management unit (MMU) which is a hardware device and this mapping is known as the paging technique.
    - The Physical Address Space is conceptually divided into several fixed-size blocks, called frames.
    - The Logical Address Space is also split into fixed-size blocks, called pages.
    - Page Size = Frame Size

- The address generated by the CPU is divided into
    - Page number(p): Number of bits required to represent the pages in Logical Address Space or Page number

- Page offset(d): Number of bits required to represent a particular word in a page or page size of Logical Address Space or word number of a page or page offset.
- Physical Address is divided into
- Frame number(f): Number of bits required to represent the frame of Physical Address Space or Frame number frame
- Frame offset(d): Number of bits required to represent a particular word in a frame or frame size of Physical Address Space or word number of a frame or frame offset.

- 

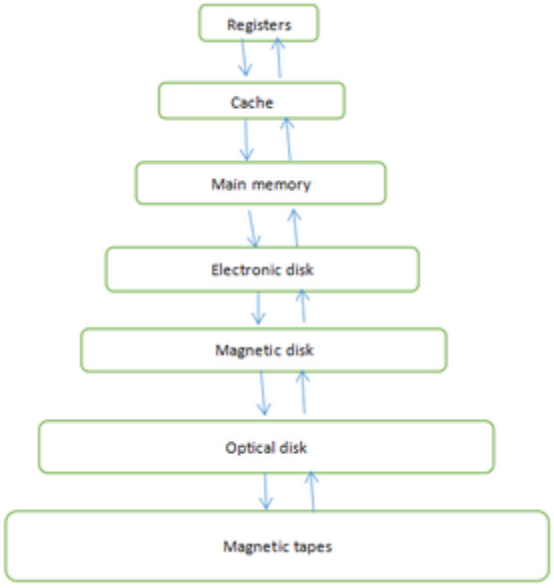
- The hardware implementation of the page table can be done by using dedicated registers. But the usage of register for the page table is satisfactory only if the page table is small. If the page table contains a large number of entries then we can use TLB(translation Look-aside buffer), a special, small, fast look-up hardware cache.
  - The TLB is an associative, high-speed memory.
  - Each entry in TLB consists of two parts: a tag and a value.
  - When this memory is used, then an item is compared with all tags simultaneously. If the item is found, then the corresponding value is returned.

- 

Demand Paging:

Demand paging is a method that loads pages into memory on demand. This method is mostly used in virtual memory. In this, a page is only brought into memory when a location on that particular page is referenced during execution. The following steps are generally followed:

- Attempt to access the page.
- If the page is valid (in memory) then continue processing instructions as normal.
- If a page is invalid then a page-fault trap occurs.
- Check if the memory reference is a valid reference to a location on secondary memory. If not, the process is terminated (illegal memory access). Otherwise, we have to page in the required page.
- Schedule disk operation to read the desired page into main memory.
- Restart the instruction that was interrupted by the operating system trap.
- Thrashing: Situation where swapping and paging are utilizing more CPU than a process's execution
- Segmentation: a Memory Management technique that divides processes into modules and parts of different sizes. These parts and modules are known as segments that can be allocated to process.



Interprocess Communication:

- A process can be of two types:
  - Independent process.
  - Co-operating process.
- An independent process is not affected by the execution of other processes while a co-operating process can be affected by other executing processes. Though one can think that those processes, which are running independently, will execute very efficiently, in reality, there are many situations when co-operative nature can be utilized for increasing computational speed, convenience, and modularity. Inter-process communication (IPC) is a mechanism that allows processes to communicate with each other and synchronize their actions.
- The communication between these processes can be seen as a method of co-operation between them. Processes can communicate with each other through both:
  - **Shared Memory Method:**
    - There are two processes: Producer and Consumer. Producer produces some item and Consumer consumes that item. The two processes shares a common space or memory

location known as buffer where the item produced by Producer is stored and from where the Consumer consumes the item if needed.

- There are two version of this problem: first one is known as unbounded buffer problem in which Producer can keep on producing items and there is no limit on size of buffer, the second one is known as bounded buffer problem in which producer can produce up to a certain amount of item and after that it starts waiting for consumer to consume it.

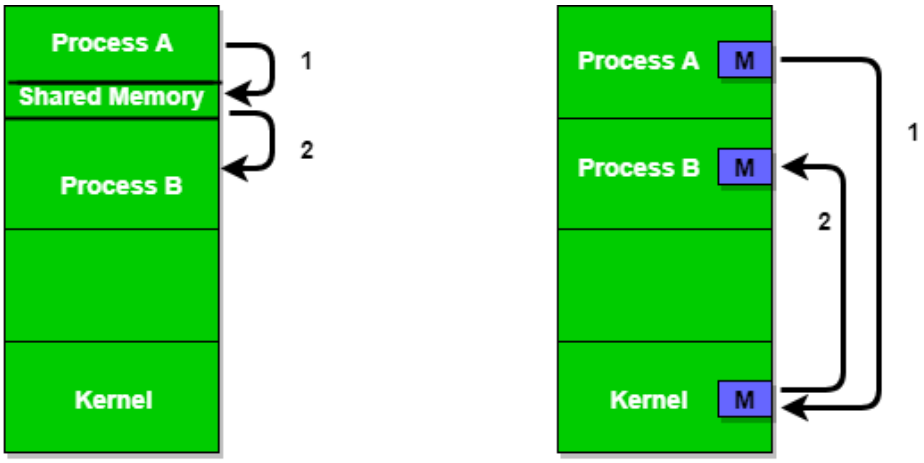
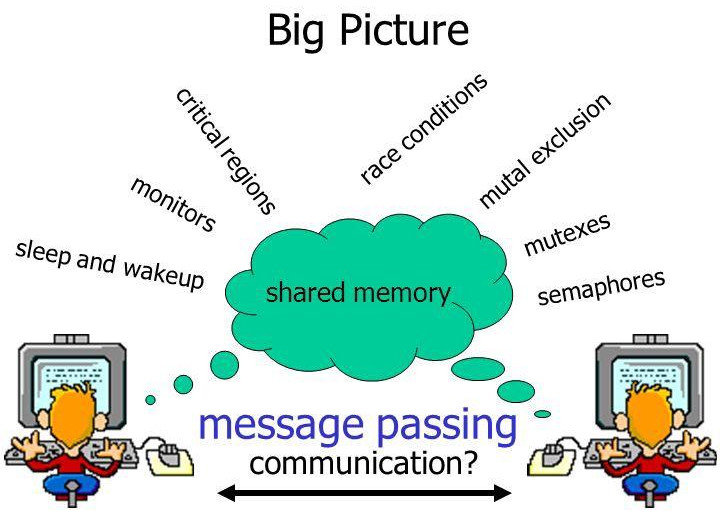


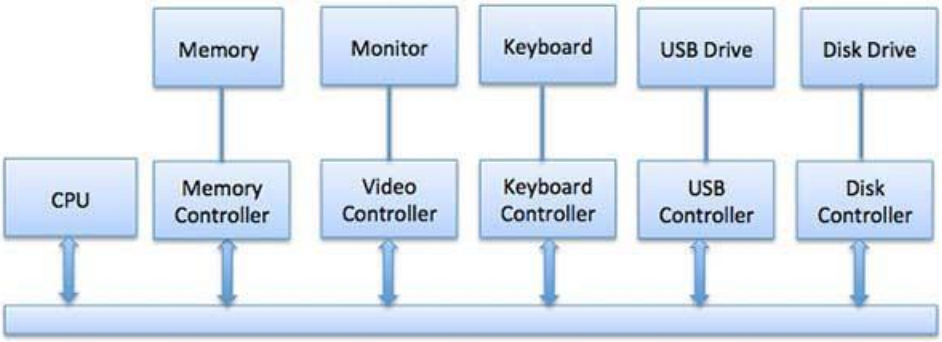
Figure 1 - Shared Memory and Message Passing

- **Message Passing Method:**
  - In this method, processes communicate with each other without using any kind of shared memory. If two processes p1 and p2 want to communicate with each other, they proceed as follows:
  - Establish a communication link (if a link already exists, no need to establish it again.)
  - Start exchanging messages using basic primitives. We need at least two primitives:
    – send(message, destination) or send(message)
    – receive(message, host) or receive(message)



I/O Management:

- An I/O system is required to take an application I/O request and send it to the physical device, then take whatever response comes back from the device and send it to the application. I/O devices can be divided into two categories:
  - Block devices — A block device is one with which the driver communicates by sending entire blocks of data. For example, hard disks, USB cameras, Disk-On-Key etc.
  - Character Devices — A character device is one with which the driver communicates by sending and receiving single characters (bytes, octets). For example, serial ports, parallel ports, sounds cards etc.



- The CPU must have a way to pass information to and from an I/O device. There are three approaches available to communicate with the CPU and Device.
  - Special Instruction I/O
    - This uses CPU instructions that are specifically made for controlling I/O devices. These instructions typically allow data to be sent to an I/O device or read from an I/O device.
  - Memory-mapped I/O
    - When using memory-mapped I/O, the same address space is shared by memory and I/O devices. The device is connected directly to certain main memory locations so that I/O device can transfer block of data to/from memory without going through CPU.

        - While using memory mapped IO, OS allocates buffer in memory and informs I/O device to use that buffer to send data to the CPU. I/O device operates asynchronously with CPU, interrupts CPU when finished.
        - The advantage to this method is that every instruction which can access memory can be used to manipulate an I/O device. Memory mapped IO is used for most high-speed I/O devices like disks, communication interfaces.
    - **Direct Memory Access:**
        - Slow devices like keyboards will generate an interrupt to the main CPU after each byte is transferred. If a fast device such as a disk generated an interrupt for each byte, the operating system would spend most of its time handling these interrupts. So a typical computer uses direct memory access (DMA) hardware to reduce this overhead.
        - Direct Memory Access (DMA) means CPU grants I/O module authority to read from or write to memory without involvement. DMA module itself controls exchange of data between main memory and the I/O device. CPU is only involved at the beginning and end of the transfer and interrupted only after entire block has been transferred.
        - Direct Memory Access needs a special hardware called DMA controller (DMAC) that manages the data transfers and arbitrates access to the system bus. The controllers are programmed with source and destination pointers (where to read/write the data), counters to track the number of transferred bytes, and settings, which includes I/O and memory types, interrupts and states for the CPU cycles.

Sockets:

- Sockets are endpoints in IPC that are basically a combination of an IP address and a port number.
- Types of Sockets:
    - Stream Sockets
    - Datagram Sockets
    - Sequenced Packet Sockets
    - Raw Sockets

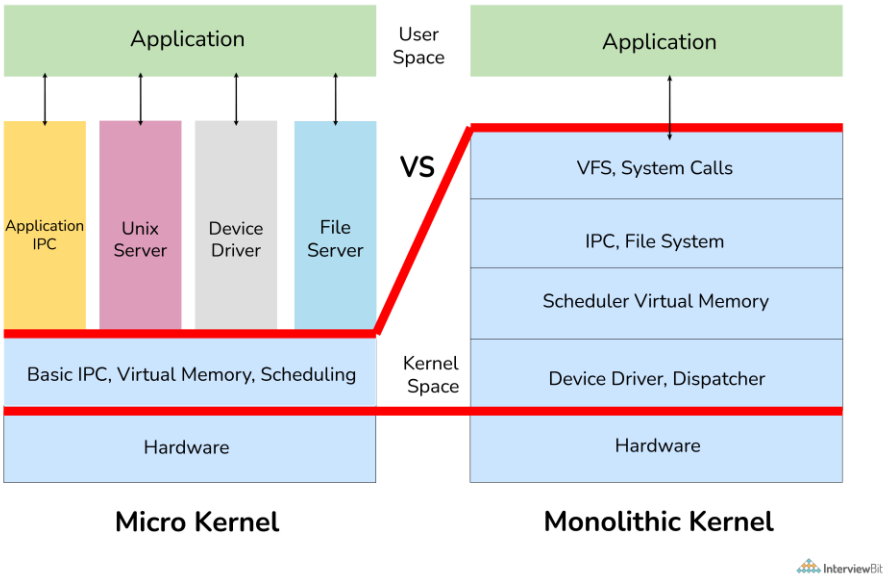
Semaphores: Semaphore is a signaling mechanism. It only holds one positive integer value. It is simply used to solve the problem or issue of critical sections in the synchronization process by using two atomic operations - wait() and signal().

Binary Semaphore	Mutex
It allows various process threads to get the finite instance of the resource until resources are available.	It allows various process threads to get single shared resource only one at a time.
Its functions are based upon signaling mechanisms.	Its functions are based upon a locking mechanism.
Binary semaphores are much faster as compared to Mutex.	Mutex is slower as compared to binary semaphores.
It is basically an integer.	It is basically an object.

Kernel:

- It is basically a computer program usually considered as a central component . Responsible for handling, managing and controlling all operations of computer systems and hardware. Acts as an interface between user applications and hardware.
- Functions:
    - It is responsible for managing all computer resources such as CPU, memory, files, processes, etc.

    - It facilitates or initiates the interaction between components of hardware and software.
    - It manages RAM memory so that all running processes and programs can work effectively and efficiently.
    - It also controls and manages all primary tasks of the OS as well as manages access and use of various peripherals connected to the computer.
    - It schedules the work done by the CPU so that the work of each user is executed as efficiently as possible.
- Types:
    - Monolithic Kernel
    - MicroKernel
    - Hybrid Kernel
    - Nano Kernel
    - Exo Kernel



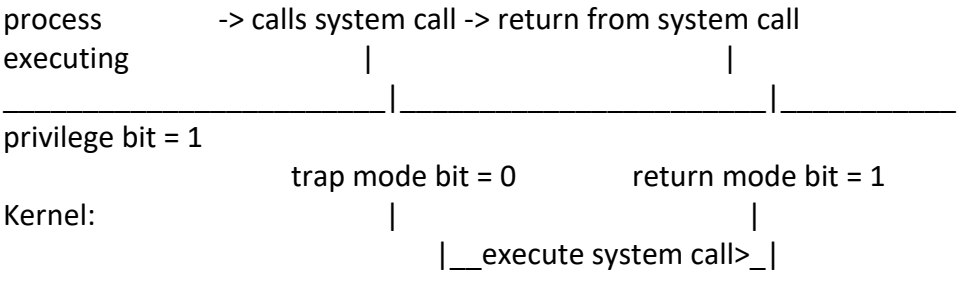
User/Kernel Boundary:
Unprivileged Mode - user level
Privileged Mode - Kernel Level

-Difference is maintained generally using a privilege bit which is set when it is in kernel mode
-Trap: If an user tries to perform a kernel operation, the op is interrupted and this condition is called a Trap
-System Call: The interaction between the applications of the Operating System can be via the system call interface.
- Signals: Signal is a mechanism for the os to pass notifications into the applications

---

System Calls
user process (privilege bit = 0)

user



To make a system call, an application must:
-write arguments
-save relevant data at well-defined location
-make system call

Types:
-Synchronous Mode: Waits until the sys call completes
-Asynchronous Mode:

Cache States:
A cache would be considered hot if an application is accessing the cache when it contains the data/addresses it needs.

A cache would be considered cold if an application is accessing the cache when it does not contain the data/addresses it needs -- forcing it to retrieve data/addresses from main memory

---

OS Services and their responsibilities:
- Block Device Driver: Responsible for access to a block device like disk
- Memory Manager: For Allocating the underlying physical memory
- Scheduler: Controlling the access to CPU/CPUs
