

OPERATING SYSTEM

why operating system?

OS does the resource management like the distribution between the memory, gpu, cpu, and disk.

OS act as interface.

OS also writes the memory and resource management code(arbitration), by this OS reduces the bulkiness of the app.(abstraction)

Memory protection and isolation is another thing of the OS.

Access to the computer hardware.

What is OS?

The OS is a piece of software that provide the above things to the user.

Goals of OS:

1. Maximum CPU utilization.
2. To avoid process starvation.
3. High priority job execution.

Types of OS:

1. **Single process OS:** only one job at a time, very old type of OS, no maximum CPU utilization, there will be process starvation. High priority job many at times cannot be executed. Ex: MS Docs.
2. **Batch processing OS:** there are N user and each of them had submitted their jobs, now there is an operator that sorts the jobs and bring them in batches and each batch executes sequentially now in this also there is no max. CPU utilization, there is process starvation and also there is no high priority job execution. Ex atlas.
3. **Multi- programming OS:** when there are multiple jobs and one job goes in the i/o state the other job is taken by the cpu and state of the first job is being saved in the process control block (PCB). This complete process is called context switching. Ex THE
4. **Multi-tasking OS:** there is single cpu and there is context switching but the only diff between the multi programming and the multi tasking is that there is time sharing. Ex: CTSS.
5. **Multi-processing OS:** there is context switching, there is time sharing the only diff is that there is more than one CPU. Ex : windows.
6. **Distributed OS:** also called loosely coupled, there are one OS and multiple CPU and memory connected over the world and whenever they want they can execute the task to any oof the cpu.

7. **RTOS:** real-time OS, we use this where we want very high accuracy and very negligible chances of error, ex: air traffic control, industrial application etc.

What is program?

Program is an executable file that contains certain set of instructions written to complete the specific job. It is compiled code, ready to execute. Stored in disk.

What is process?

Program under execution.

What is thread?

The light weight process is called thread, the minute process that can be independently executed.

What is multi-threading?

A single task is broken down into multiple threads (smaller tasks) and they executes asynchronously or simultaneously so the task gets completes in lesser time. For multi-threading we need multiple CPU's more than one. So that each sub task is executed by different CPU. Used to achieve parallelism. Ex: multiple tabs in a browser, text editor

Difference between multi-tasking and multi-threading?

Multi-tasking: more than 1 process concept, there is isolation and memory protection, processes are scheduled.

Multi-threading: more than one thread, there is no memory protection or isolation, threads are scheduled., more the number of cores then more we can divide into threads.

Difference between the thread context switching and the process context switching?

OS saves current state of thread and switches to another thread of same process, does not include the switching of memory space address, fast switching, CPU cache state is reserved.

OS saves current state of process and switches to another process, includes switching of memory space address, slow switching, CPU cache state is flushed.

Components of OS:

1. **User space:** the space where convenient apps are run is called user space, no hardware access, provides convenient space to user. Ex. GUI(graphical user interface), CLI(command line interface).

2. **Kernel:** kernel has the access to underlying hardware. This is also the heart of the OS,

User space --→ kernel ----→ hardware

Functions of kernel:

1. **Process management** (creation, termination)
scheduling of process or thread.
Process communication.
Process synchronization.
2. **Memory management** (allocate, deallocate, free space management)
3. **File management** (create, manage, delete), directory management.
4. **I/O management** (pen drive, other external devices). (spooling, buffering, caching).

Types of kernels:

1. **Monolithic kernel:** process mgmt., memory mgmt., I/O mgmt., file mgmt. all is present within the same kernel, oldest type of kernel.
Adv: the communication is fast as they are present in the same kernel.
Dis adv.: kernel become bulky and less reliable, ex. Linux
2. **Micro kernel:** the process management and memory management is within the kernel and file and I/O management is with the user space.
Adv: less bulky and more reliable, more stable.
Dis adv.: the communication is little slow. There is overhead. Ex. L4 linux etc.

How is the communication between user mode and kernel mode?

It is through inter-process communication:

Using shared memory.

Message passing

3. **Hybrid kernel:** combined approach, file management is under user space and other three inside kernel space. Ex: MacOS, windows.
4. **Nano kernel:** not very imp.
5. **Exo kernel:** not very imp.

System calls in the OS:

The only way to switch from user mode to kernel mode is through system calls. There is an interface between user space and the kernel space named SCI (system call interface).

Types of system calls:

1. Process control.
2. File management.
3. Device management.
4. Information maintenance.
5. Communication management.

Types of System Calls:

- 1) Process Control
 - a. end, abort
 - b. load, execute
 - c. create process, terminate process
 - d. get process attributes, set process attributes
 - e. wait for time
 - f. wait event, signal event
 - g. allocate and free memory
- 2) File Management
 - a. create file, delete file
 - b. open, close
 - c. read, write, reposition
 - d. get file attributes, set file attributes
- 3) Device Management
 - a. request device, release device
 - b. read, write, reposition
 - c. get device attributes, set device attributes
 - d. logically attach or detach devices
- 4) Information maintenance
 - a. get time or date, set time or date
 - b. get system data, set system data
 - c. get process, file, or device attributes
 - d. set process, file, or device attributes

Examples of Windows & Unix System calls:

Category	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Management	CreateFile() ReadFile() WriteFile() CloseHandle() SetFileSecurity() InitializeSecurityDescriptor() SetSecurityDescriptorGroup()	open () read () write () close () chmod() umask() chown()
Device Management	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Management	GetCurrentProcessID() SetTimer() Sleep()	getpid () alarm () sleep ()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe () shmget () mmap()

How OS boots up?

Five steps of the booting up:

1. **Turning ON the power supply**, as soon as the power supply turned ON the motherboard and other components gets the power supply.
2. **CPU loads BIOS or UEFI** (advance version of BIOS).
3. BIOS or UEFI runs some tests and init the hardware, loads some setting from a memory area backed by **CMOS battery**. Programs load with the settings then it runs **POST (power on self-test)** to check the essential hardware components.
4. BIOS or the UEFI hands off to boot devices (disk, pen drive, HDD/SSD), then boot loader runs the actual program that loads the OS, this is present in the MBR (master boot record), present in old devices, or the EFI (partition in the disk).
5. Boot loader loads the full OS.

32 bits vs 64 bits?

1. Addressable space (32bits – $2^{32}-1$, 64bits – $2^{64}-1$)
2. Resource usages better in 64bits.
3. Calculation much fast in 64bits.
4. 64bits CPU can run both 32 and 64bit OS, while 32bit can run 32bit OS.
5. Better graphic performance.

Comparison between different storages used in computers?

1. **Resister** (smallest memory and closest to CPU).
2. **Cache** (memory that stores temporary data that are more frequently used)
3. **Main memory** (contains the program under execution)

The above the three are called **primary memory**.

4. Electronic disk
5. Magnetic disk
6. Optical disk
7. Magnetic tapes

These 4 are called secondary memory.

Comparison:

Cost: resisters are highly costly, made of rare earth metals.

Next comes the cache and then main memory.

Access speed: resister then cache then main memory and then secondary one.

Storage size: resister then cache then main memory.

Volatility: main memory data is flushed off as the computer is turned off but the secondary memory remains saved.

How operating system created a process?

There are total of 5 steps:

1. Load the program and the static data to memory, used for initialization.
2. Allocate runtime stack.
3. Allocate heap, part of memory used for dynamic allocation.
4. I/O task input handle, output handle and error handle.
5. OS hands off the control to main.

Architecture of process:

Stack ----- heap----data----text.

Stack overflow-> assign base case.

Memory overflow->deallocate unnecessary objects.

Attributes of process:

To study this, we need to focus upon PCB (process control block).

1. Process ID: unique ID to the process.
2. Program counter: keeps the track on which process currently we are.
3. Process state: to check the state of the program, new, wait, RUN.....
4. Priority: priority of the job.
5. Register: save the registers when there is context switching.
6. Open file/ list:
7. Open devices list.

What are different process states in OS?

New state: program to process begin done.

Ready state: process is in memory or in ready queue

Running state: CPU is being allocated and the process is running

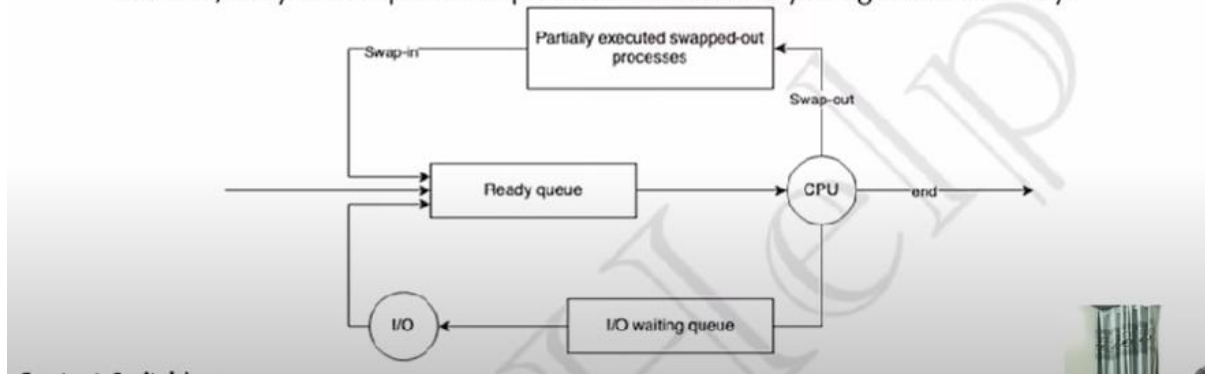
Waiting state: while running sometimes there is I/O call/ instruction so the process goes in waiting state and waits for the I/O to complete.

Termination state: the process is complete and there is no more the existence of the process.

Job scheduler (long term scheduler) handles degree of multi programming and CPU scheduler (short term scheduler) handles ready to running state.

Medium term scheduler:

- f. Swapping is a mechanism in which a process can be swapped temporarily out of main memory (or move) to secondary storage (disk) and make that memory available to other processes. At some later time, the system swaps back the process from the secondary storage to main memory.



Context switching in OS: Context switching in an operating system refers to the process of storing the state of a currently running process or thread so that it can be resumed later and restoring the state of another process or thread to start or resume its execution. This allows multiple processes to share a single CPU and gives the illusion that they are running simultaneously (multitasking).

Orphan process: the process that does not have parent or we can say that parent process has been terminated, and it is still running, first process of OS is init.

Zombie process: the process that has been terminated or its execution is complete but it still has an entry in the process table. It usually occurs for the child process, once the zombie process is eliminated the process is removed from the table, this one is called reaping.

The two types of CPU scheduling?

1. **Non-Preemptive scheduling:** the process will not leave the CPU until it is terminated or it is sent to the wait state. Process starvation is more in this.
2. **Pre-emptive:** the process will leave the CPU when it is terminated, sent to wait state or time quantum for that process expires. More CPU utilization. More overhead.

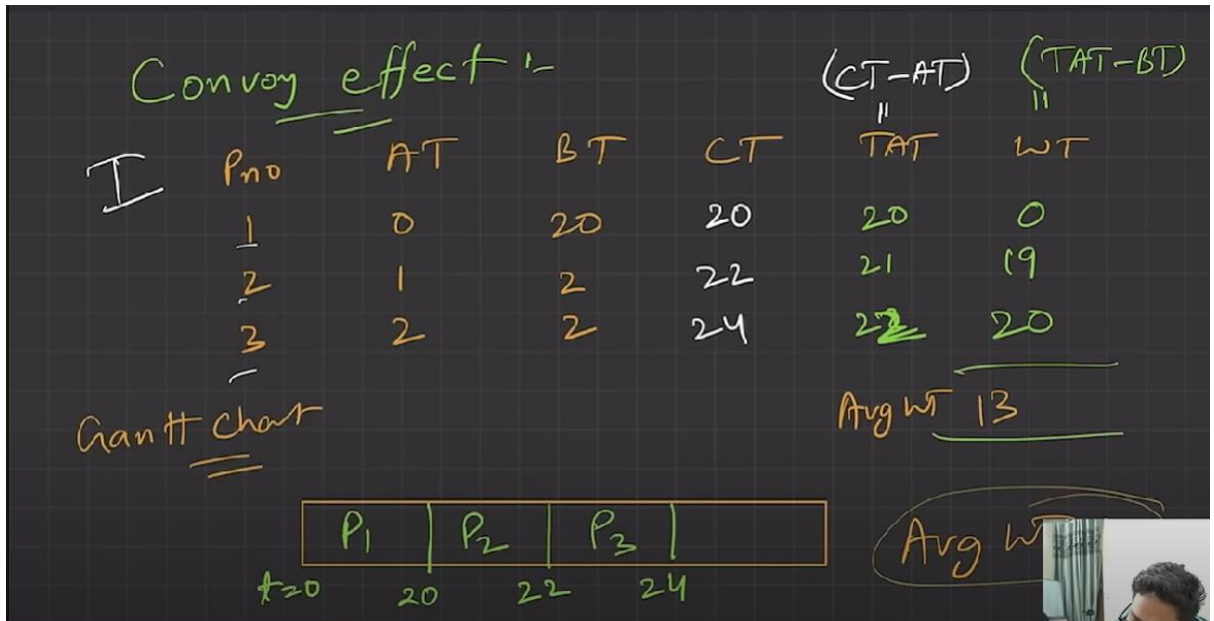
Goals of the CPU scheduling:

1. More CPU utilization.
2. Minimum turn-around time. ($TAT = CT - AT$)
3. Min. wait time. ($WT = TAT - BT$)
4. Min. response time.
5. Max. throughput (max. process executed per unit time).

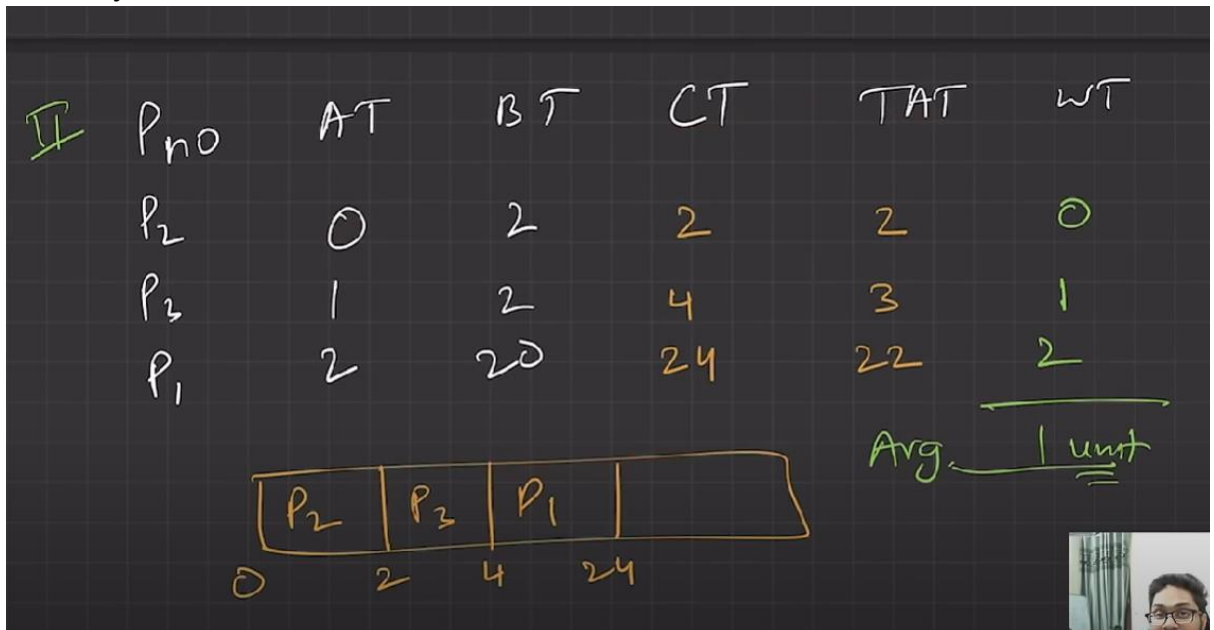
Types of Scheduling algorithms:

1. **First come first serve (FCFS):** the first process that will come in the ready queue, it will be assigned CPU and so on.

One of the drawbacks of this is convoy effect which is calculated using gantt chart.



2. **Shortest job first:**



So, convoy effect states that if a job has higher BT then it will have the impact on the avg. waiting time of the of all the job.

3. **Non- Preemptive Shortest job first (SJF):** the process with the least burst time is allocated CPU first,

P	AT	BT	CT	TAT	WT
P ₁	0	8	8	8	0
P ₂	1	4	12	11	7
P ₃	2	9	26	24	15
P ₄	3	5	17	14	9
					<u>AvgWT 7.75s.</u>

Non-pre-emptive: [P₁ | P₂ | P₄ | P₃ |]
 t=0 8 12 17 26

Criteria: AT + BT

Pre-emptive version:

* Preemptive SJF ✓

P	AT	BT	CT	TAT	WT
P ₁	0	8 7	17	17	9
✓ P ₂	1	4	5	4	0
P ₃	2	9	26	24	15
P ₄	3	5	10	7	2
					<u>AvgWT: 6.5 units.</u>

[P₁ | P₂ | P₄ | P₁ | P₃ |]
 t=0 1 5 10 17 26

Now the question comes will there be convoy effect in this case, so the answer is NO as when any other job having burst time less than the executing job that will be stopped. So, less starvation in this case.

4. **Priority Scheduling:** we assign priority to each job,

P	Prints	AT	BT	CT	TAT	WT
1	2	0	4	4	4	0
2	4	1	2	25	24	22
3	6	2	3	23	22	19
4	10	3	5	9	6	1
5	8	4	1	20	16	15
6	12	5	4	13	8	4
7	9	6	6	19	13	7
Avg. 9.714						

P ₁	P ₄	P ₆	P ₇	P ₅	P ₂	P ₃
4	9	13	19	20	23	25

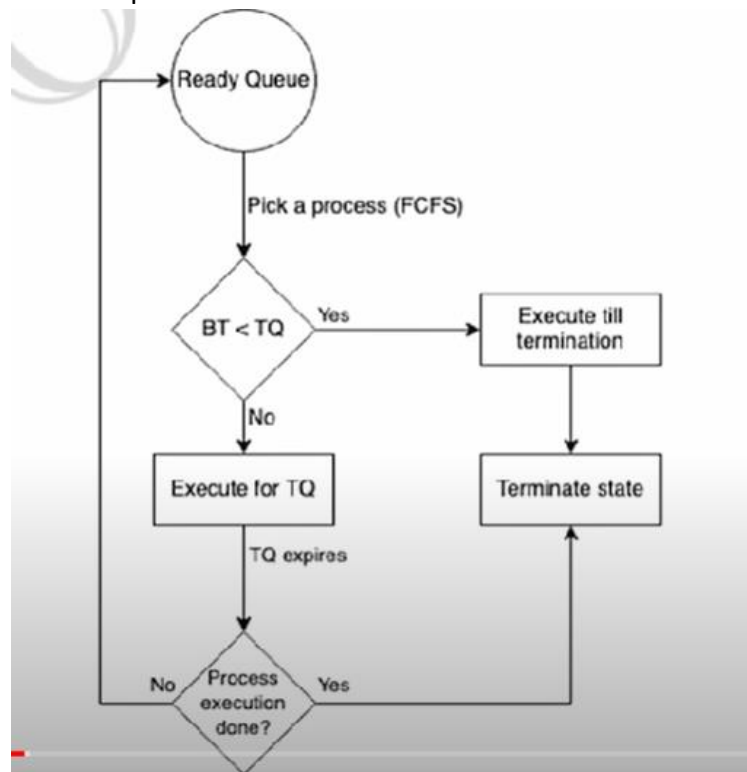
$\sum = 20$

Pre-emptive priority Scheduling: A lot of overhead.

Biggest drawback of both pre-emptive and non-Preemptive priority scheduling is indefinite waiting or extreme starvation.

Solution: gradually increase the priority of the job, as the age of the job increases.

5. **Round robin:** most popular and the pre-emptive version of FCFS. Its criteria is Arrival time and time quantum. Design time sharing, easy to implement, multi-tasking best suited. More the time quantum less the overhead and vice-versa.



6.

Multi-level queue scheduling and comparison between the CPU scheduling algo?

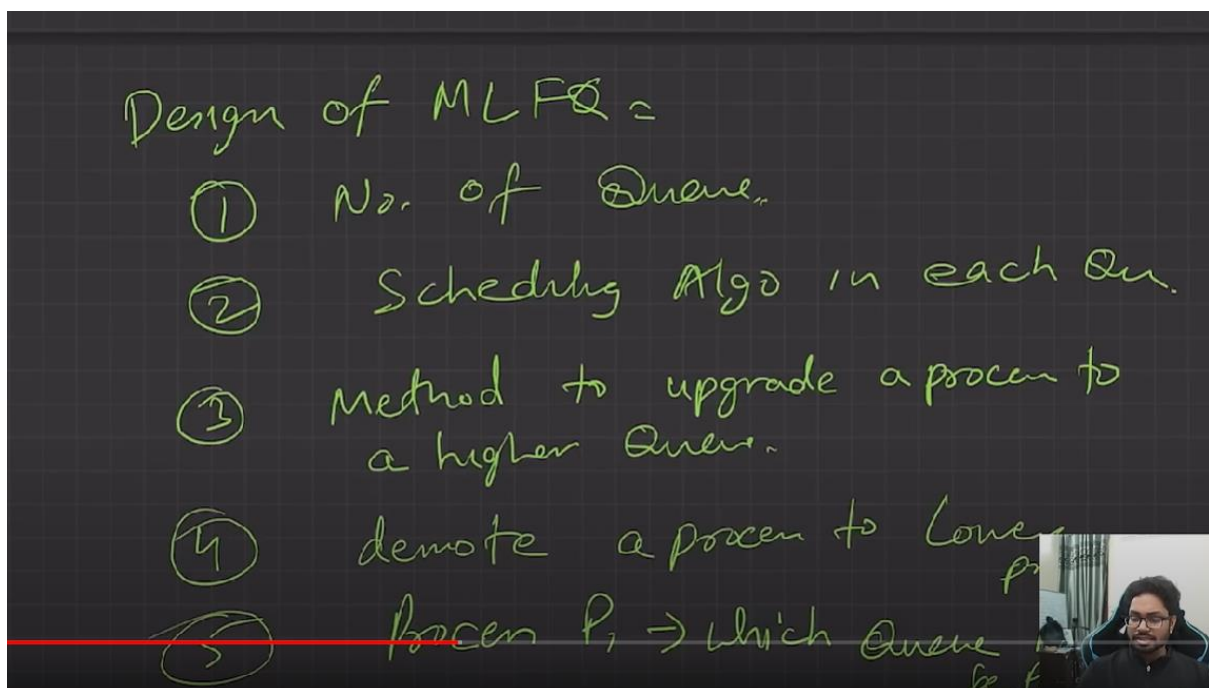
System process: created by OS. **FIRST PRIORITY**

Interactive process: requires user input. Also called foreground process. **SECOND PRIORITY.**

Batch process: these runs in the background, also called background processes. **LEAST PRIORITY.**

So, in multi-level queue scheduling convoy effect is present.

Multi-level feedback queue scheduling: multiple sub-queues, there is allowed to move in the inter-queue. Separate process based on BT, more the BT lower the priority, I/O bound and interactive process are kept in higher priority queue. Aging method is used to prevent starvation. Flexible and configurable OS design.



Comparison of all the algo we have studied till now:

3. Comparison:

	FCFS	SJF	PSJF	Priority	P-Priority	RR	MLQ	MLFQ
Design	Simple	Complex	Complex	Complex	Complex	Simple	Complex	Complex
Preemption	No	No	Yes	No	Yes	Yes	Yes	Yes
Convoy effect	Yes	Yes	No	Yes	Yes	No	Yes	Yes
Overhead	No	No	Yes	No	Yes	Yes	Yes	Yes

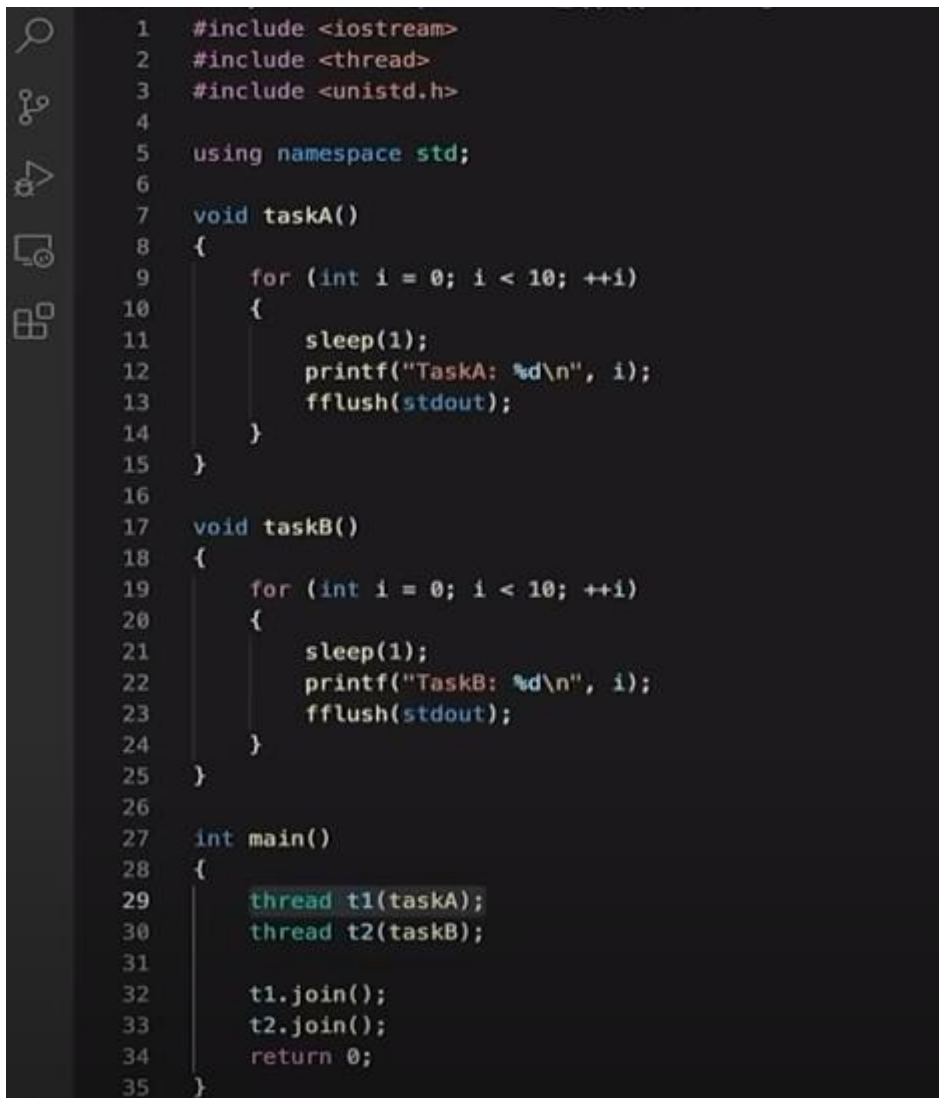
What is concurrency?

This executes multiple instructions at the same time. this is done using breaking the process into threads and following the instruction.

For ex: in MS word (text editor, spell checker, text formatter) all three are different threads executes simultaneously. This is done using thread scheduling algo. Address space remains same, and context switching is very fast. But in single CPU there is no advantage we get.

Benefits of multi-threading:

1. Responsiveness, more interactive.
2. Resource sharing is much more efficient.
3. Threads are economical.
4. Threads better utilize multi core CPU.



```
1  #include <iostream>
2  #include <thread>
3  #include <unistd.h>
4
5  using namespace std;
6
7  void taskA()
8  {
9      for (int i = 0; i < 10; ++i)
10     {
11         sleep(1);
12         printf("TaskA: %d\n", i);
13         fflush(stdout);
14     }
15 }
16
17 void taskB()
18 {
19     for (int i = 0; i < 10; ++i)
20     {
21         sleep(1);
22         printf("TaskB: %d\n", i);
23         fflush(stdout);
24     }
25 }
26
27 int main()
28 {
29     thread t1(taskA);
30     thread t2(taskB);
31
32     t1.join();
33     t2.join();
34     return 0;
35 }
```

Critical section problem: The critical section problem in operating systems refers to the challenge of ensuring that multiple processes or threads can access shared resources or data in a way that prevents conflicts and ensures data integrity. The critical section is a part of the code where a process accesses shared resources. The problem arises because simultaneous access to shared resources can lead to data inconsistency or corruption.

Ways to prevent Critical section:

1. **Mutual exclusion:** Ensuring that only one process can be in the critical section at a time.
2. **Progress:** If no process is in the critical section, and there are some processes that wish to enter the critical section, then one of these processes must be allowed to enter the critical section.
3. **Bounded Waiting:** There should be a limit 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.
4. **Peterson's Algorithm:** A classical software-based solution for two processes that ensures mutual exclusion, progress, and bounded waiting.
5. **Lock Variables:** Simple variables used to control access to the critical section.

Conditional variables and semaphore to synchronise thread:

Semaphore: A more complex synchronization tool that can be used to solve the critical section problem for multiple processes. Semaphores use two atomic operations, **wait()** and **signal()**.

Producer and consumer problem and its solution:

Also called as bounded buffer problem,

There are producer thread and consumer thread, there should be sync between producer and consumer, producer must not insert data when the buffer is full, also the consumer must not pick/remove data when the buffer is empty.

Reader writer problem and its solution:

In this there are two threads reader thread and writer threads. Note that there are one writer and multiple reader.

If there are multiple writers then race problem will occur and data will become inconsistency. Also, at the time when writer is writing the reader cannot read at that moment.

Solution:

Mutex: using binary semaphore, to ensure mutual exclusion, when read count is updated, no two threads modify read count the same time.

Write: this one also binary semaphore, common for both reader and writer.

Read count: integer initialized with 0.

The dining philosopher's problem:

1. We have **5 philosophers**.
2. They spend their life just being in **two states**:
 - a. Thinking
 - b. Eating
3. They sit on a circular table surrounded by 5 chairs (1 each), in the center of table is a bowl of noodles, and the table is laid with 5 single forks.
4. **Thinking state:** When a ph. Thinks, he doesn't interact with others.
5. **Eating state:** When a ph. Gets hungry, he tries to pick up the 2 forks adjacent to him (Left and Right). He can pick one fork at a time.
6. One can't pick up a fork if it is already taken.
7. When ph. Has both forks at the same time, he eats without releasing forks.
8. Solution can be given using semaphores.
 - a. Each fork is a binary semaphore.
 - b. A ph. Calls wait() operation to acquire a fork.
 - c. Release fork by calling signal().
 - d. **Semaphore fork[5]{1};**
9. Although the semaphore solution makes sure that no two neighbors are eating simultaneously but it could still create **Deadlock**.
10. Suppose that all 5 ph. Become hungry at the same time and each picks up their left fork, then All fork semaphores would be 0.
11. When each ph. Tries to grab his right fork, he will be waiting for ever (Deadlock)
12. We must use **some methods to avoid Deadlock and make the solution work**
 - a. Allow at most 4 ph. To be sitting simultaneously.
 - b. Allow a ph. To pick up his fork only if both forks are available and to do this, he must pick them up in a critical section (atomically).
 - c. **Odd-even rule.**
an odd ph. Picks up first his left fork and then his right fork, whereas an even ph. Picks up his right fork then his left fork.
13. Hence, only semaphores are not enough to solve this problem.
We must add some enhancement rules to make deadlock free solution.

What is a deadlock?

1. In Multi-programming environment, we have several processes competing for finite number of resources
2. Process requests a **resource (R)**, if R is not available (taken by other process), process enters in a waiting state. Sometimes that waiting process is never able to change its state because the resource, it has requested is busy (forever), called **DEADLOCK (DL)**
3. Two or more processes are waiting on some resource's availability, which will never be available as it is also busy with some other process. The Processes are said to be in **Deadlock**.
4. DL is a bug present in the process/thread synchronization method.
5. In DL, processes never finish executing, and the system resources are tied up, preventing other jobs from starting.
6. **Example of resources:** Memory space, CPU cycles, files, locks, sockets, IO devices etc.
7. Single resource can have multiple instances of that. E.g., CPU is a resource, and a system can have 2 CPUs.
8. How a process/thread utilize a resource?
 - a. Request: Request the R, if R is free Lock it, else wait till it is available.
 - b. Use
 - c. Release: Release resource instance and make it available for other processes



Necessary condition for deadlock:

1. Mutual exclusion.
2. Hold and wait.
3. No-preemption.
4. Circular wait.

Method to avoid deadlock:

1. Deadlock prevention:
 12. **Deadlock Prevention:** by ensuring at least one of the necessary conditions cannot hold.
 - a. **Mutual exclusion**
 - i. Use locks (mutual exclusion) only for non-sharable resource.
 - ii. Sharable resources like Read-Only files can be accessed by multiple processes/threads.
 - iii. However, we can't prevent DLs by denying the mutual-exclusion condition, because some resources are intrinsically non-sharable.
 - b. **Hold & Wait**
 - i. To ensure H&W condition never occurs in the system, we must guarantee that, whenever a process requests a resource, it doesn't hold any other resource.
 - ii. Protocol (A) can be, each process has to request and be allocated all its resources before its execution.
 - iii. Protocol (B) can be, allow a process to request resources only when it has none. It can request any additional resources after it must have released all the resources that it is currently allocated.

c. **No preemption**

- i. If a process is holding some resources and request another resource that cannot be immediately allocated to it, then all the resources the process is currently holding are preempted. The process will restart only when it can regain its old resources, as well as the new one that it is requesting. (Live Lock may occur).
- ii. If a process requests some resources, we first check whether they are available. If yes, we allocate them. If not, we check whether they are allocated to some other process that is waiting for additional resources. If so, preempt the desired resource from waiting process and allocate them to the requesting process.

d. **Circular wait**

- i. To ensure that this condition never holds is to impose a proper ordering of resource allocation.
- ii. P1 and P2 both require R1 and R1, locking on these resources should be like, both try to lock R1 then R2. By this way which ever process first locks R1 will get R2.

Solving leetcode problem on concurrency.

Print order – leetcode – 1114:

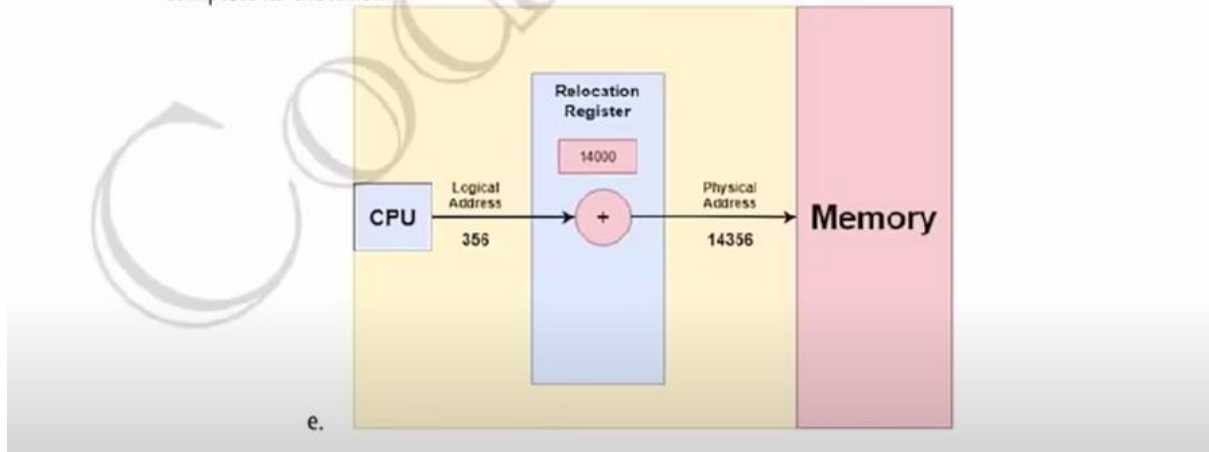
Fizz Buzz Multi-threaded – leetcode – 1195:

Dining's philosophers' problem – leetcode – 1226:

Memory management in OS:

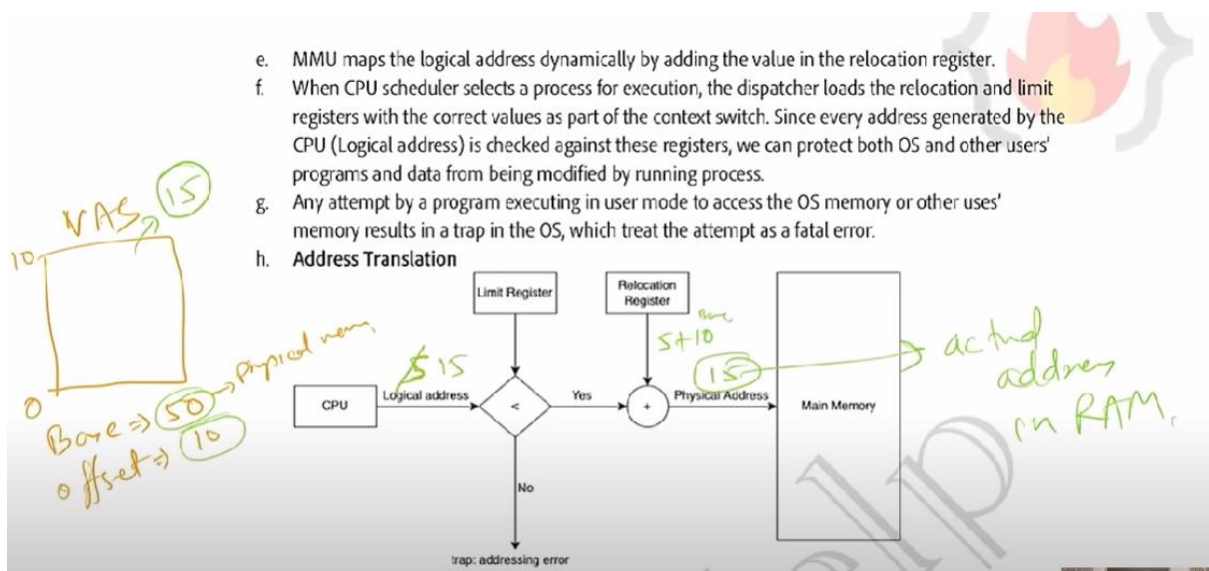
1. In Multi-programming environment, we have multiple processes in the main memory (Ready Queue) to keep the CPU utilization high and to make computer responsive to the users.
2. To realize this increase in performance, however, we must keep several processes in the memory; that is, we must **share** the main memory. As a result, we must **manage** main memory for all the different processes.
3. **Logical versus Physical Address Space**
 - a. **Logical Address**
 - i. An address generated by the CPU.
 - ii. The logical address is basically the address of an instruction or data used by a process.
 - iii. User can access logical address of the process.
 - iv. User has indirect access to the physical address through logical address.
 - v. Logical address does not exist physically. Hence, aka, **Virtual address**.
 - vi. The set of all logical addresses that are generated by any program is referred to as Logical Address Space.
 - vii. **Range: 0 to max.**
 - b. **Physical Address**
 - i. An address loaded into the memory-address register of the physical memory.
 - ii. User can never access the physical address of the Program.
 - iii. The physical address is in the memory unit. It's a location in the main memory physically.
 - iv. A physical address can be accessed by a user indirectly but not directly.
 - v. The set of all physical addresses corresponding to the Logical addresses is commonly known as Physical Address Space.
 - vi. It is computed by the **Memory Management Unit (MMU)**.

- vii. Range: $(R + 0)$ to $(R + \text{max})$, for a base value R .
- c. The runtime mapping from virtual to physical address is done by a hardware device called the **memory-management unit (MMU)**.
- d. The user's program mainly generates the logical address, and the user thinks that the program is running in this logical address, but the program mainly needs physical memory in order to complete its execution.



4. How OS manages the isolation and protect? (Memory Mapping and Protection)

- a. OS provides this Virtual Address Space (VAS) concept.
- b. To separate memory space, we need the ability to determine the range of legal addresses that the process may access and to ensure that the process can access only these legal addresses.
- c. The relocation register contains value of smallest physical address (Base address $[R]$); the limit register contains the range of logical addresses (e.g., relocation = 100040 & limit = 74600).
- d. Each logical address must be less than the limit register.



Memory allocation in physical memory:

1. Contiguous memory location: each process is contained in a single block of contiguous blocks.

For this we can do fixed partitioning (ram divided into partitions of equal size), in this we internal + external fragmentation. In this we have low degree of multi-programming.

2. Dynamic partitioning: in this technique, partition size is not declared initially, declared at the time process loading.

Advantages: no internal fragmentation. No limit on the size of the process. Better degree of multi-programming.

Disadvantage: external fragmentation.

How operating system manages free space:

This is done using defragmentation or compaction. There will be no effect on the user process as user works on logic memory and OS works on physical memory. There is only one issue in this and that is efficiency is decreased a bit as defragmentation is itself a time-consuming process. Linked list data structure is used to keep the track of the free space.

There are various algorithms which are implemented by the OS in order to find out the holes in the linked list and allocate them to the process:

1. First fit: we allocate the first hole that is big enough. For ex: the process is of 90kb and the linked list is:
50->100->90->200->50->null
So, we allocate the 90/100 as it is the first big space. Fast and easy to implement.
2. Next fit: now let suppose after the process p1 = 90kb is allocated the space and next process p2 = 10kb comes then it will be allocated not from the starting but just after the process p1 as 10kb of space is left in that 10/100.
3. Best fit: the process is allocated to the space that is just bigger than the process so it leaves the least amount of the space. In the above example we allocate 90kb of linked list space to p1. Least fragmentation internal, slow. Creates major external fragmentation.
4. Worst fit: we allocate the process the largest hole in case of 90kb of process we allocate 200 of the linked list holes. Lesser external fragmentation. slow

What is paging?

The logical address space is divided into pages and the physical address space(RAM) is divided into frames this is called paging and this is done using page table.



Paging is a memory management scheme used in operating systems to manage how data is stored and retrieved from physical memory (RAM). Here's a simpler explanation:

Concept of Paging

1. Dividing Memory into Pages:

- **Logical Memory** (the memory addresses used by programs) is divided into fixed-size blocks called **pages**.
- **Physical Memory** (the actual RAM) is also divided into fixed-size blocks called **frames**.
- Pages and frames are usually the same size, for example, 4KB.

2. Storing Pages in Frames:

- When a program needs to run, its pages are loaded into any available frames in physical memory.
- The operating system keeps track of which page is in which frame using a **page table**.

3. Address Translation:

- When a program wants to access a memory location, it uses a logical address (page number and offset).
- The operating system translates this logical address into a physical address (frame number and offset) using the page table.

Why Use Paging?

1. Efficient Use of Memory:

- Paging allows the physical memory to be used more efficiently because it eliminates the need for contiguous allocation of memory. Programs can be loaded into any available frames.

2. Isolation and Security:

- Each program operates in its own set of pages, which helps in isolating programs from each other and enhances security.

3. Easier Memory Management:

- It simplifies memory management since all pages are of fixed size, avoiding problems like fragmentation (wasted memory).

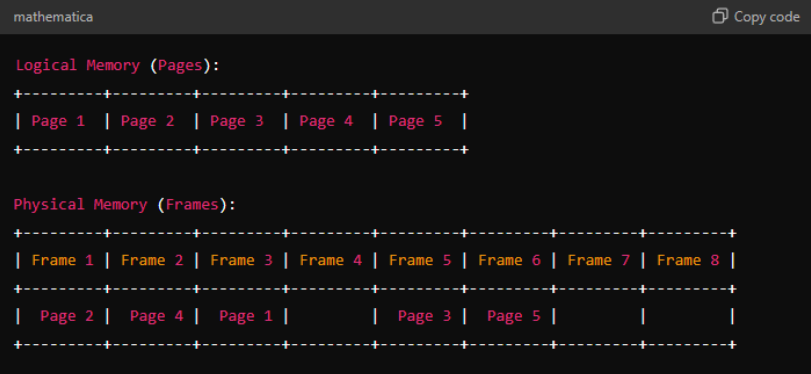
Example

- Suppose a program has a logical address space divided into 5 pages (each 4KB in size).
- The physical memory has 8 frames (each 4KB in size).
- When the program runs, its pages could be loaded into any 5 of the 8 available frames.

If the program wants to access the 2nd byte of its 3rd page:

1. The logical address might be something like (Page 3, Offset 2).
2. The operating system uses the page table to find that Page 3 is currently in Frame 5.
3. The physical address is then (Frame 5, Offset 2).

Diagram



In this diagram:

- Page 1 is in Frame 3.
- Page 2 is in Frame 1.
- Page 3 is in Frame 5.
- Page 4 is in Frame 2.
- Page 5 is in Frame 6.

What is segmentation?

This handles not the fixed partitioning of the logical address space but variable partitioning of the logical address space. For ex: a function of code should not get divided into two parts but they should remain intact.

10. Advantages:

- No internal fragmentation.
- One segment has a contiguous allocation, hence efficient working within segment.
- The size of segment table is generally less than the size of page table.
- It results in a more efficient system because the compiler keeps the same type of functions in one segment.

11. Disadvantages:

- External fragmentation.
- The different size of segment is not good that the time of swapping.

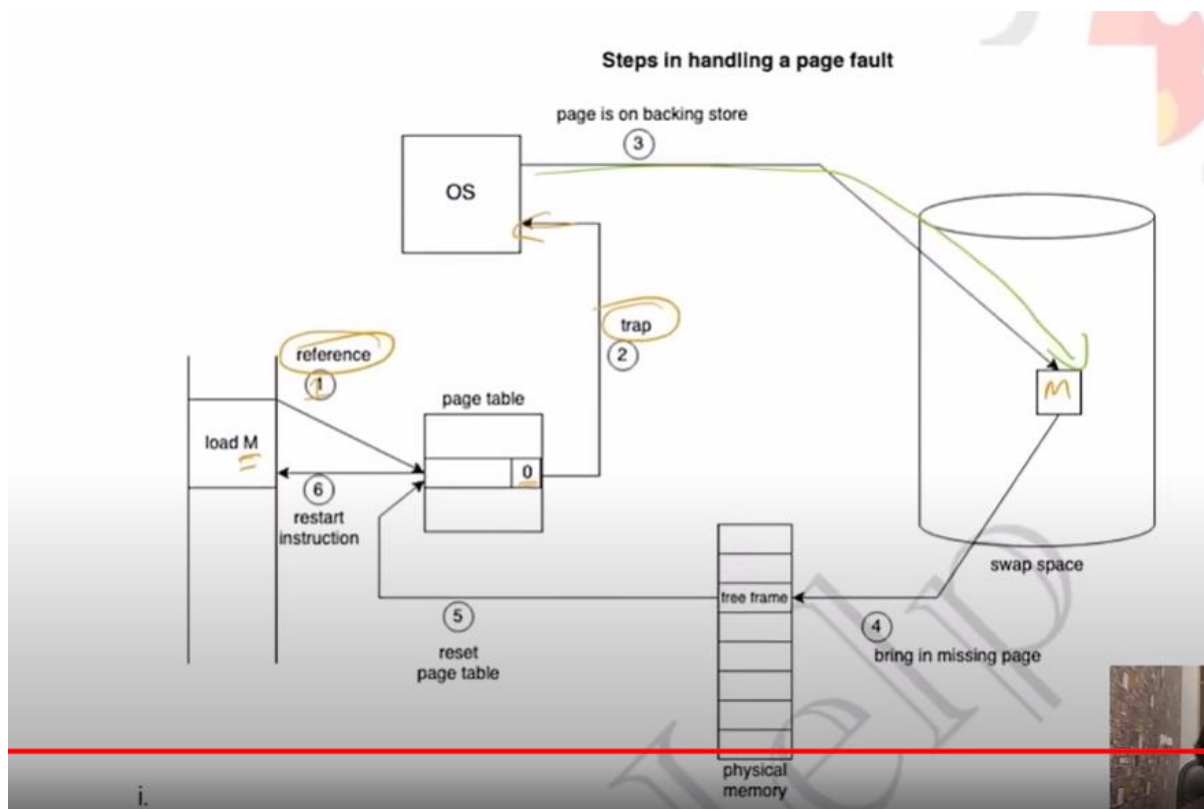
What is virtual memory?

- Virtual memory** is a technique that allows the execution of processes that are not completely in the memory. It provides user an illusion of having a very big main memory. This is done by treating a part of secondary memory as the main memory. (Swap-space)
- Advantage** of this is, programs can be larger than physical memory.
- It is required that instructions must be in physical memory to be executed. But it limits the size of a program to the size of physical memory. In fact, in many cases, the entire program is not needed at the same time. So, we want an ability to execute a program that is only partially in memory would give many benefits:
 - A program would no longer be constrained by the amount of physical memory that is available.
 - Because each user program could take less physical memory, more programs could be run at the same time, with a corresponding **increase in CPU utilization and throughput**.
 - Running a program that is not entirely in memory would benefit **both the system and the user**.

4. Programmer is provided very large virtual memory when only a smaller physical memory is available.
5. **Demand Paging** is a popular method of **virtual memory management**.
6. In demand paging, the pages of a process which are least used, get stored in the secondary memory.
7. A page is copied to the main memory when its demand is made, or **page fault** occurs. There are various **page replacement algorithms** which are used to determine the pages which will be replaced.
8. Rather than swapping the entire process into memory, we use **Lazy Swapper**. A lazy swapper never swaps a page into memory unless that page will be needed.
9. We are viewing a process as a sequence of pages, rather than one large contiguous address space, using the term **Swapper** is **technically incorrect**. A swapper manipulates entire processes, whereas a **Pager** is concerned with individual pages of a process.

10. How Demand Paging works?

- a. When a process is to be swapped-in, the pager guesses which pages will be used.
- b. Instead of swapping in a whole process, the pager brings only those pages into memory. This, it avoids reading **into memory pages that will not be used anyway**.
- c. Above way, OS decreases the swap time and the amount of physical memory needed.
- d. The **valid-invalid bit scheme in the page table** is used to distinguish between pages that are in memory and that are on the disk.
 - i. Valid-invalid bit 1 means, the associated page is both legal and in memory.
 - ii. Valid-invalid bit 0 means, the page either is not valid (not in the LAS of the process) or is valid but is currently on the disk.
- e.
- f. If a process never attempts to access some invalid bit page, the process will be executed **successfully without even the need pages present in the swap space**.
- g. What happens if the process tries to access a page that was not brought into memory, access to a page marked invalid causes **page fault**. Paging hardware noticing invalid bit for a demanded page will cause a **trap to the OS**.
- h. **The procedure to handle the page fault:**
 - i. Check an internal table (in PCB of the process) to determine whether the reference was valid or an invalid memory access.
 - ii. If ref. was invalid process throws exception:
If ref. is valid, pager will swap-in the page.
 - iii. We find a free frame (from free-frame list)
 - iv. Schedule a disk operation to read the desired page into the newly allocated frame.
 - v. When disk read is complete, we modify the page table that, the page is now in memory.
 - vi. Restart the instruction that was interrupted by the trap. The process can now access the page as through it had always been in memory.



Study more about demand paging and pure demand paging.

Page replacement algorithm:

Page fault, in this RAM is divided into multiple pages or frames and these frames are allocated to different processes, now what happens not all the pages of the process are loaded but only the required ones are allocated the frame and whenever we requires other pages we just swap it with what we don't want at the moment, the page that we want to swap is present in swap area. This complete process is called page fault and the time taken is called page fault service time.

3. The **page replacement algorithm** decides which memory page is to be replaced. Some allocated page is swapped out from the frame and new page is swapped into the freed frame.
4. **Types of Page Replacement Algorithm:** (AIM is to have minimum page faults)
 - a. **FIFO**
 - i. Allocate frame to the page as it comes into the memory by **replacing the oldest page**.
 - ii. Easy to implement.
 - iii. Performance is **not** always good
 1. The page replaced may be an initialization module that was used long time ago (**Good replacement candidate**)
 2. The page may contain a heavily used variable that was initialized early and is in content use. (**Will again cause page fault**)
 - iv. **Belady's anomaly** is present.

Second method is, **Optimal page replacement:**

1. Best algorithm.'
2. Page fault minimum.
3. Almost impossible to implement.

- b. **Optimal** page replacement
 - i. Find if a page that is never referenced in future. If such a page exists, replace this page with new page.
If no such page exists, find a page that is **referenced farthest in future**. Replace this page with new page.
 - ii. **Lowest** page fault rate among any algorithm.
 - iii. Difficult to implement as **OS requires future knowledge of reference string** which is kind of impossible. (Similar to SJF scheduling)

Now, the third algorithm is, **LRU Cache**:

- c. **Least-recently used (LRU)**
 - i. We can use recent past as an approximation of the near future then we can replace the page that has not been used for the longest period.
 - ii. Can be implemented by two ways
 - 1. **Counters**
 - a. Associate time field with each page table entry.
 - b. Replace the page with smallest time value.
 - 2. **Stack**
 - a. Keep a stack of page number.
 - b. Whenever page is referenced, it is removed from the stack & put on the top.
 - c. By this, most recently used is always on the top, & least recently used is always on the bottom.
 - d. As entries might be removed from the middle of the stack, so Doubly linked list can be used.

What is thrashing?

- 1. **Thrashing**
 - a. If the process doesn't have the number of frames it needs to support pages in active use, it will quickly page-fault. At this point, it must replace some page. However, since all its pages are in active use, it must a page that will be needed again right away. Consequently, it quickly faults again, and again, and again, replacing pages that it must bring back in immediately.
 - b. This **high paging activity** is called **Thrashing**.
 - c. A system is Thrashing when it **spends more time servicing the page faults than executing processes**.

