① Chapter # 1.

Introduction :-

Operating System is the most important system software. Os acts as an intermidiary between user of the compiler and compiler hardware. Some important task of Os are;

- Maintains operations of compiler.
- Store and retrieves files
- Schedule programs for execution.
  eg: More applications are executed.
- Coordinate the execution of program.

→ Compiler system
① Hardware
② Operating System
③ Application Programs.
④ Users.

→ The Hardware
① Central Processing Unit
② Memory
③ I/O devices
- Provides basic computing resources for system.

→ The Application Programs such as ① word processors
② spread sheets
③ compiless
④ Webbrowsers

- define ways in which these resources are used to solve user's compiling problems.

→ Operating System provides environment within which other programs can do useful work.

→ OS is a resource allocator (controls programs) Decides btw conflicting requests for efficient & fair use.

→ Kernel: Performs basic required functions. Runs at all times on the computer. (*)

→ Mobile OS includes not only a core kernel but also middleware (set of software frameworks that provides additional services to application developers).

→ OS is a control program that manages execution of user programs to prevent errors and improper use of computers.
Concerned with I/O devices.

→ Computer System Organization: -

- Initial program to run → bootstrap program
- Stored within ROM (or) EEPROM as Firmware
- Initializes all aspects of the system, from CPU registers to device controllers to memory contents
- Bootstrap program loads kernel into memory.
- Then Kernel provides services to the system and its users and starts execution.

- Interrupt: Occurence of an event from either the hardware or the software.
- Software execute interrupt by a "system call".
- Hardware Trigger interrupt by sending signal to CPU.

Interrupt $\xrightarrow{IV}$ Interrupt service routine → Interrupt Specific Handler

- Interrupt Vector table: Holds addresses of interrupt service routine.

(*) System Programs: Associated with OS but not the part of Kernel.

Application Programs: Include programs that are not associated with operation of the system.

Von-neumann Architecture (4)

→ Simple instruction execution cycle

Fetch, decode, execute.

→ Storage Structure : =

(CPU can load instructions only from memory)

* Program written in main memory (RAM)
  - DRAM.

Main memory is usually too small to store all needed programs and data permanently.

Main memory is volatile storage device (loses contents) → CPU can access directly.

* ROM (Read Only Memory)
  - EEPROM.

Static Programs
Non-volatile.
Instructions for booting / Bootstrap program.

* Secondary Storage → Extension of main memory
  - holds large data permanently
  eg: Magnetic disk

* Cache-memory
* CD-ROM
* Magnetic Tapes.

* Cache - Memory:
  - Intermidiate buffer for storage of data.
  - Copying information into faster storage system.
  - Hardware based.
  - Improved system's performance.

→ Computer System Architecture :-

* Single Processor Systems:
  Contains only one general purpose CPU, capable of executing general purpose instruction set. Can have other special purpose processors as well.

* Multi Processor Systems:
  Parallel systems / Multicore systems / Tightly coupled.
  Have two or more processors in close communication.
  ① Increased Throughput: By increasing number of processors, more work can be done in less time.
  ② Economy of scale: Multiprocessor systems can cost less than multiple single processor systems.
  ③ Increased Reliability: failure of one processor will not halt system, only slow it down.

* Asymmetric Multiprocessing : ◦- Multiple CPU with different architecture
◦- Each processor is assigned a specific task ◦- Own address space
◦- Boss - worker relationship. ◦- intensive Task
◦- Boss processor schedules and allocates work to the worker processors.

* Symmetric Multiprocessing : ◦- Multiple CPU with same architecture
◦- Each processor performs all tasks with an OS ◦- simple Task
◦- Each processor has its own set of registers
◦- Shared memory space
◦- Many processes can run simultaneously.

Uniform Memory Access (UMA):
◦- Access to any RAM from any CPU takes the same amount of time.

Non-Uniform Memory Access (NUMA):
- Some parts of memory may take longer time to access than other parts, creating performance penalty.

Multicore processors :- Multiple computing cores on a single chip → (less power & faster communication)
- More efficient than multiple chips with single core → (higher power & slower communication).

Dual core processors :- Two cores on a single chip. Each core has its own register set as well as its own local cache.

→ Clustered Systems :
- Group of system/servers for fast processing.
- Share storage.
- Closely linked through LAN
- High Availability service : Works even if one or more system fails
- High performance Computing environments.

* Asymmetric clustering :-
  - One machine is in hot-stand by mode while other is running application
  - Hot-stand by machine monitor the active server.
  - If Active-server fails, hot-stand by machine becomes an Active-server.

* Symmetric Clustering:
  - Two or more hosts are running applications and are monitoring each other.

→ Operating System Structure:

Multiprogramming
- Needed for efficiency.
- CPU selects & runs job via job scheduling

Time sharing (Multitasking)
- Interactive computing
- Response time is so fast
- Several jobs are ready to run at same time (job scheduling)
- Swapping moves processes in and out of memory to run

→ Operating System Operations:

Dual-mode operation.
* User mode → less privileged
* Kernel mode → high privileged

- System call → changes mode to kernel mode
- Returning from system call → Resets to user mode

- Mode-bit : Kernel (0) & user (1)

→ Process Management :

Program in execution is called Process.
(job / time-shared program)

- Program ⇒ Passive Entity
  ; Contents of file stored on disk
- Process ⇒ Active Entity.
  ; needs resources like memory, files, I/O devices

* Creating & deleting both user & system processes
* Suspending & resuming processes.
* Process synchronization
* Process communication
* Deadlock handling.

→ Memory Management:
- Keeping track of which part of memory are currently being used and by whom.
- Deciding which processes and data to move into and out of memory.
- Allocating & deallocating memory space as needed.

→ Storage Management:

* File-system Management:
- Creating & deleting files and directories
- Primitives to manipulate files & dirs.
- Mapping files onto secondary storage
- Backup files on stable storage media.

Secondary/
Mass-Storage Management:
- Free space management
- Disk scheduling
- Storage allocation

I/O Subsystems:
- Memory management of I/O.
- Caching (Storing parts of data in

faster storage for performance)

- Spooling: Overlapping of output of one job with input of another job.

→ Protection and Security:

- Protection:
  Access control is defined by OS to prevent data from being corrupt by the user.

- Security:
  Mechanism to protect from malware, viruses, identity theft.
  like Antivirus Program.

→ Kernel Data Structure:

* Lists, stacks and Queues:-

Lists: Collection of data in sequence

- Singly-linked list.
- Doubly-linked list.
- Circular-linked list

Stack : Sequentially ordered data struction that uses LIFO principle.

Queue: Sequentially ordered data structer that uses FIFO principle.

* Trees: Hierarchical data structure.
  - linux uses balanced bst for CPU scheduling algorithm

* Hash function & maps

→ Computing Environments:

* Traditional Computing Environment:
  - Portals provide web access to internal systems
  - Network computers (thin clients) are like Web terminals
    Used in Banks, call centres.
  - Mobile computer interconnect via wireless networks.

* Mobile - Computing Environment:
  - Handheld smartphones, tablets etc.
  - Extra features → more OS features like GPS.
  - Apple iOS and Google Android

* Distributed - Computing Environment:
  - Collection of seperate, possibly heterogeneous, systems network together
  - Network is communication path
    LAN — Intranet.
    WAN — Internet

* Client Server:
  - Centralized management
  - One server and multiple clients
  - Server responds requests generated by clients

* Peer - to - Peer: eg: Torrent
  - Not a centralized system
  - Distributed management & systems are connected.

Chapter # 3: Processes

→ Process Concept:

* Batch system: Groups jobs and executed together.
* Time-shared: User programs or tasks

"Process is a program in execution"

* Process includes:
  - Text section → Program code
  - Program Counter → Address of next instruction
  - Stack → temporary data (function parameters, return addresses & local variables)
  - Data section → Global variable
  - Heap → Dynamically allocated runtime memory.

* Program: Passive entity
  Executable file stored on disk.

* Process: Active entity.
  Executable file loaded into memory along with program counter and set of associated resources.

eg: An executable java program is executed within the java virtual machine.

* Process States:
As a process executes, it changes state.
- New: Process is being created → fork( ).
- Running: Instructions are being executed.
- Waiting: Waiting for some event to occur.
- Ready: Waiting to be assigned to a processor (scheduling algorithm) (FCFS, SJF,)
- Terminated: Finished execution.

* Process Control Block:
Process in OS is represented by PCB also called Task Control Block.

- Process State :- New, running, waiting, ready, halt.
- Program Counter :- Instruction Register.
- CPU registers:- Accumulator, index register, stack pointer, general-purpose registers. Contents of all process.
- CPU-scheduling information:-
Process priority, pointers to scheduling queues.
- Memory Management information:-
Value of base and limit registers, page tables, segment tables

- Accounting Information:
  Amount of CPU & real time used, time limits, account numbers, job or process numbers.
- I/O status information:
  List of I/O devices, list of open files etc.

* Thread:
- Smallest execution unit in a process is called thread.
- Process is a program that performs single thread of execution.
- Modern operating systems allow a process to have multiple threads of execution.
- Beneficial for multicore systems, where multiple threads can run in parallel.

→ Process Scheduling:
- Process schedular selects an available process for program execution on CPU.
- Objective of multiprogramming: Maximize CPU utilization. (Some process running all times)
- Objective of time sharing: Switch CPU among processes so frequently that users can interact with each program.

* Scheduling Queues:-

- Job queue:- Set of newly created processes
- Ready queue:- Processes in main memory, that are ready & are waiting to execute (linked list)
- Device queue:- Processes that are waiting for particular I/O devices

A new process is initially put in the ready queue. It waits there until it is selected for execution, or dispatched. Once the process is allocated the CPU and is executing, one of several events could occur.

- I/O request generated → I/O queue (waiting state to ready state)
- Creates child process → wait for child's termination
- Remove process forcibly/ → due to interrupt put back in ready queue.

* Schedulers:

Selects an available process for program execution.

1) Long Term scheduler / Job scheduler
2) Short Term scheduler / CPU scheduler

① Long Term Scheduler / Job scheduler:
- Selects which process to be brought into the ready queue from job queue.
- It is invoked infrequently
- long term scheduler can be slow.
- It controls the degree of multiprogramming. (capacity of a processor)

(Unix & Microsoft systems do not have long term scheduler)

② Short Term Scheduler / CPU scheduler.
- Selects which process should be executed next and allocates CPU (ready queue to running queue)
- It is invoked very frequently
- Short term scheduler must be fast.

Note: Long-term scheduler should selects a good process mix of I/O bound & CPU bound processes, for best performance, otherwise system will be unbalanced
- All I/O bound processes → ready queue will always be empty & short term will have little work to do
- All CPU bound processes → I/O waiting queue will always be empty & devices will go unused

* I/O bound processes: I/O intensive, spends more time with I/O rather than computation.
Short CPU burst time (execution)

* CPU bound processes: Computation intensive, spends more time doing compilation rather than I/O.
Very long CPU bursts.

* Medium-Term Scheduler: (intermidiate level)
Decreases degree of multiprogramming via swapping low priority process in ready queue and then reintroduce later into the memory and continued its execution where it left off.

* Context-Switching:
Switching CPU to another process, due to interrupt, requires to save current process state in its PCB and loads new process scheduled to run.

- Context-switch time is pure overhead, because the system does no useful work while switching.
- Context-switch time are highly dependent on hardware support.
- Simply requires changing the pointer to the current register set.

→ Operations on Processes:-

* Process Creation:
- Creating Process : Parent Process
- New Process : Child Process (baby chunna murna)
- Multiple childs : form tree of processes.
- Process Identifier (Pid) : Unique integer for each process in system, used as index.
- Init process (pid 1) : Root parent process.

Kthreadd
(create additional process that performs task on behalf of kernel).

sshd
(managing clients that connects to the system Secure shell.)

- Listing of processes → ps command.

eg: ps -el (list complete information of all processes currently active in system).

- Parent & children may or may not share all resources
- Execution:
  → Parent continues to execute concurrently with its children.
  → Parent waits until some or all of its children have terminated.

(Continued at pg # 32)

Chapter # 2 "Operating System Structures"

* An operating system provides an environment for the execution of the program.
* It provides certain services to programs and to the users to those programs.

→ Services provided by OS are helpful for user:

1. User Interface:
   - Every OS have user interface.
   - Command Line Interface (CLI):
     Uses text commands & a method for entering them.
   - Batch Interface:
     Commands & directives are entered into files & those files are executed.
   - Graphical User Interface (GUI):
     Window system interface with a pointing device and a keyboard.

2. Program Execution:
   - Program loaded into the memory, executed and ended normally or abnormally (indicating error).

3. I/O Operations:

- A running program may require I/O, which may involve a file or an I/O device.
- For efficiency & protection user can not control I/O devices directly. Therefore OS must provide means to do I/O.

File-system Manipulation:

- Programs need to write and read files and directories. They also need to create, delete & search files.
- OS includes permission management based on files ownership.

Communication:

- These are processes which need to exchange information with other processes.
- Communication can be implemented via shared memory.
  i.e Two or more processes read and write to a shared memory of a section.
- It can also be implemented via message passing.
  i.e Packets of information in predefined

formats are moved between processes by the OS.

6. Error detection:
- OS needs to be constantly aware of all errors.
- Error may occur in the CPU, memory, I/O devices or in user programs.
- For each error, OS should take an appropriate action.

1. Resource Allocation:
- When multiple jobs are running concurrently, OS decides between the conflicting requests for efficient and fair use and allocate resources to each program.

8. Accounting:
- Keeps the track of which user uses and how much and what kind of computer resources.

9. Protection:
- OS ensures that the access to the system resources is controlled, to

prevent data from being corrupt by the user

10 Security:
- Security requires authentication of a user usually by means of a password, to protect from malware, viruses and theft of services

→ User & OS Interface:

- Command line interface
- Graphical User interface
- Touch screens

* Command-line Interface:
- CLI allows direct command entry.
- Multiple command interpreters are known as "Shells"

eg: Bourne shell, Bourne Again Shell, C Shell, Korn shell etc.
- Fetches commands from user and execute it.

* Graphical User Interface:
- GUI is a user friendly desktop interface.
- GUI uses input/output devices.
- Icons represents files and programs
- Various actions are performed by mouse click or by keyboard.

* Touch screens Interface:
- Actions are performed by gestures of fingers.
- Virtual keyboards for entering text.

→ System Calls:
- Programming interface to the services provided by OS.
- Written in high-level language.
- Thousands of system calls are execute per second.

* Application Programming Interface:
- (API) specifies a set of functions available to an application programmer including parameters and return values.

eg:
1. Windows API for Windows Systems.

2. POSIX API for POSIX-based systems.
3. JAVA API for JVM.

- A programmer accesses API via library of code provided by an OS.

* System Call's Parameters:
There are three ways
1. Pass parameters in register.
2. Stored in a block or a table in memory
3. By using stack.

* Types of System Calls:

1. Process Control:
- end / abort
- load, execute.
- create process, terminate process. fork(), exit()
- get process attributes, set process attributes.
- wait for time. wait()
- wait event, signal event.
- allocate and free memory.

2. File Management:
- create file, delete file
- open, close
- read, write, reposition
- get file attributes / set file attributes

open()
read(), write()

3. Device Management:
- request / release device
- read, write, reposition
- get / set device attributes
- attach / detach devices

read(), write()

4. Information Maintenance:
- get / set time / date
- get / set system data
- get / set process / file / device attributes

getpid(), alarm(), sleep()

5. Communication:
- create, delete communication connection
- send, receive messages
- transfer status information
- attach or detach remote devices

Pipe(), mmap(), shmget()

→ Operating System Structure:

① Simple Structure:
- MS-DOS does not have well defined structure
- Interface and levels of functionalities are not seperated and it was not divided into subsystems.
- No distinction between user and kernel mode.
- Allow direct access to hardware.
- No dual mode and no hardware protection can cause entire system to crash.

② Layered Approach:
- OS divides in layers.
- Each layer can be developed independently and can be debugged easily.
- Problem is with the order of layer, as no layer can call services of higher layer.
- It is inefficient because it has to access higher layer to lower layer each time to reach hardware.

③ Microkernels:

- Divided into two modes
  - * Kernel mode: Have all privileged instruction
  - * User mode: Less privileged or system applications
- Basic process and memory management.
- Enhances security & protection due to dual mode.
- System expansion is easier. Any program can be installed without launching kernel

④ Modules:

- Object-Oriented Technique.
- Small core kernel and a set of modules which can be linked dynamically.
- Similar to layers but with clearly defined tasks and interfaces.
- Message passing to kernel is not required, since modules are free to contact eachother.

⑤ Hybrid:

→ Operating System Debugging:

- Finding & fixing errors or bugs
- Log Files : Error information
- Core dump : Generated on failure of an application (process memory).
- Crash dump: Generated on failure of OS (kernel memory).

→ SYSGEN program obtains information concerning the specific configuration of the hardware system.

→ System Programs:
Provides environment to execute and develop program.
Associated with OS but not a part of Kernel

- File management
- Status information
- File modification
- Programming-language support
- Program loading & execution
- Communication
- Background Services
- Application Programs

Chapter #3 Continued....

- Address space :
  - → Child process is a duplicate of parent process
  - → Child process has new program loaded into it
- fork( ) → creates child.
  - return 0 ⇒ for child.
  - return > 0 ⇒ for parent.
  - return < 0 ⇒ for error.
- exec( ) → loads binary file into a memory and starts its execution.
- wait( ) → parent waits for Termination of its child.

* Process Termination:

- exit( ) → asks OS to delete process after its execution.
- All resources are deallocated.
- Parent may terminate execution of child process
  - → child has exceeded allocated resources.
  - → Task assigned to child is not longer required.
  - → The parent is exiting and OS does

not allow a child to continue if its parent terminates.

- Cascading Termination: If a process terminates, then all its child process must also be terminated.
- Orphan Process: If a parent process has finished execution and exited, but at the same time if the child process remains unexecuted, the child is called Orphan Process.
- Zombie Process: If a child process terminates but parent does not call wait( ) system call yet.

→ Inter process Communication:

* Independent process: If process cannot affect or be affected by other processes executing in system.

* Cooperating process: If it can affect or be affected by other processes executing in system.

* Process Cooperation Reasons:

1. Information Sharing:
   Environment to provide access to certain piece of information.

2. Computation Speedup:
- Multiple processing cores requires
- Tasks break into sub tasks, each of which will executing in parallel to other to run faster.

3. Modularity:
- Dividing functions into seperate processes or threads.

4. Convenience:
- Multiple tasks can be performed on same time

* Cooperating requires IPC mechanism to exchange data & information.

- Shared memory: (Faster)
  Two or more processes read and write to a shared memory of a section. (shared region).
- Message passing:
  Packets of information are moved between processes by the OS.
  Easy to exchange smaller amounts of data.

- Shared-Memory :
  → Region of shared memory
  * Producer-Consumer problem:
    Producer : Produces information (client)
    Consumer : Consumes information (server).
  * Buffer of items in a shared region, filled by producers and emptied by consumers.
  * Types of buffers:
    → Unbounded buffer
      No limit on size of buffer
    → Bounded buffer
      Fixed buffer size.

- Message-Passing:
  → Mechanism to allow processes to communicate and synchronize their actions without sharing the same address space
  → Distributed environment
  → Facilities : Send or receive.
  → logical implementation of link :
    - Direct / indirect communication.
    - Synchronous / Asynchronous communication.
    - Automatic / Explicit buffering.

* Direct Communication:
  - Processes must name each other explicitly.
  - Communication link:
    → links are established automatically.
    → link is associated with one pair of communicating process.
    → Each pair have exactly one link.
    → link may be unidirectional, but is usually bidirectional.

* Indirect Communication:
  - Messages are send and received from mailboxes, each having unique id.
  - Communication link:
    → links established if process share a common mailbox.
    → link may be associated with many programs.
    → Each pair may have several links.
    → Links may be unidirectional a bidirectional.

* Synchronization:
Message passing may be either blocking or non-blocking

Blocking
* Synchronous
* Blocking send:
Sender block until message received
* Blocking receive:
Receiver block until message available

Non-blocking
* Asynchronous
* Non-blocking send:
Sender send message
* Non-blocking receive:
Receiver receives a valid message

* Buffering:
Queue of messages attached to the link.
① Zero-capacity: 0 message in waiting queue
② Bounded capacity: n-message in waiting queue
③ Unbounded capacity: infinite message can wait in a queue

* Pipes:
Conduit allowing two processes to communicate.

o- Ordinary Pipes:
→ Producer-consumer style.
→ Producer writes to one end.
→ Consumer reads from other end.
→ Unidirectional.
→ Parent-child relationship

o- Named Pipes:
→ More powerful than ordinary pipes.
→ Bidirectional.
→ No parent-child relationship.
→ Used by both Unix & Windows System.

Threads :-

Thread: Thread is a basic unit of CPU utilization (OR) Smallest execution unit. It comprises of thread ID, a program counter, a register set and a stack.

Single Threaded	Multithreaded
* Able to service only one client at a time. & a client might have to wait for long time for its request to be serviced.	* Enhance processing capabilities on multicore systems. * Performs CPU intensive tasks in parallel across multiple computing cores.

→ Remote Process Call (RPC):

Thread plays important role in RPC systems. RPCs provide communication mechanism to allow interprocess communication. RPC servers are multithreaded.

→ Benefits of Mullithreading:

① Responsiveness:
Mullithreading is an interactive application. If a time consuming operation is performed on a single thread, application remains responsive.

② Resource sharing:
Process share resources → shared memory / Message passing
Threads share memory & resources of the process to which they belong by default.

③ Economy:
Since thread share resources by default it is more economical to create and context-switch threads.

④ Scalability:
Mullithreading benefits more to multiprocessing architecture, where threads may be running in parallel on different processing cores

→ Multicore Programming :-

* Multiple computing cores on a single chip, whether across CPU chips or within CPU chips called Multicore or multiprocessor systems.

* Multithreaded programming → more efficient use of multicore systems and improved concurrency.
  i.e: threads can run in parallel.

Parallelism

Data Parallelism

* Data divides accross parallel computing nodes but single task is performed

Eg:
1. Sorting
2. Word count
3. Searching

* More speed b/c of one execution-thread
* Parallelism is proportional to input data size.

Task Parallelism

* Data divides across parallel computing nodes with multiple divided Tasks

Eg
1. Matrix mulliplication (x,+)
2. Gaming
3. Banking System.

* Less speed as each will execute a different thread.
* Parallelism is proportional to no. of independent tasks

→ Chalenges for programmer for better utilization of Multicore programming.

① Dividing tasks: Concurrent tasks are seperated to run on different processors.

② Balance: Balance work distributed to each processor.

③ Data Splitting: Data accessed and manipulated by the tasks must be divided to run on seperate cores.

④ Data Dependency: Programmers must ensure that the execution of the dependent tasks must be synchronized.

⑤ Testing & Debugging: Testing & debugging programs running in parallel is difficult.

→ Amdah'ls Law:

" Identifies speedup in latency of the execution tasks at fixed work load by adding additional computing cores to an application. "

$$\text{speedup} \leq \frac{1}{S + (1-S)/N}$$

S = sesial execution portion.

N = processing cores.

→ divide

* Fork - Join Model:
  - If problem is small, solve directly using sequential algorithm.
  - Else divide tasks into subtasks on multiple parallel thread, having one master thread.

→ Multithreading Models:

① User thread:
  - Threads provided by the user level.
  - User threads are supported above the kernel
  - User threads are managed without kernel support

② Kernel thread:
  - Threads provided at the kernel level
  - Kernel threads are supported and managed directly by the OS.

* Multithreading Models establish the relationships between the user thread & kernel thread.

① Many-to-One Model:

- Many User level threads mapped to single kernel level thread.
- Only one thread can access the kernel at a time
- Multiple threads are unable to run in parallel on multicore systems.
- Entire process will block, if a thread makes a blocking system call.

Disadvantage: inability to take advantage of multiple processing cores.

② One-to-One Model:

- Each User level thread maps to Kernel level thread.
- More concurrent because process can run even if single thread makes a blocking system call.
- Multiple threads can run in parallel on multicore systems.

Disadvantage:

- Creating a user thread requires creating the corresponding kernel thread.
- Overhead of creating kernel threads burdens performance of application.

(3) Many - to - Many Model:

- Multiplexes many user-level threads to equal or smaller number of kernel-level threads.
- Multiple user threads can be created as per requirement.
- Multiple kernel threads corresponding to user thread can run in parallel
- Kernel can schedule another user thread for execution even if one thread makes a blocking system call.

→ Examples:

- Green thread (For Solaris systems & early versions of JAVA) used Many-to-One model.
- Linux along with WOS implement One-to-One model
- Solaris OS supported Two-level Model (Many-to-Many Model).

→ Thread Libraries: -

A thread library provides the programmer with an API for creating and managing threads

- Three main thread libraries:

① POSIX Pthreads:

- May be provided either as user-level or kernel-level library. (Linux & Unix)

② Windows Thread:

Kernel level library on Windows System (Windows)

③ JAVA Thread API:

Allows threads to be created & managed directly in JAVA programs. (JVM).

→ Implicit Threading:

To address the difficulties of multicore programming transfer the creation & management of threading from application developers to compilers and run-time libraries. It is called Implicit threading.

Alternative approaches of designing multithreaded programs for better utilization of multicore processors through implicit threading.

① Thread Pool ② Open MP ③ Grand Central Dispatch.

① Thread Pools:

- Create a number of threads at process startup and place them into a pool, where they wait for work. When a server receives a request, it passes the request to the available thread for service. Once service is completed, thread returns to pool and wait for more work.

Benefits:

* Servicing a request with an existing thread is faster than waiting to create a thread.
* Limits the number of threads. Good for systems that cannot support a large number of concurrent threads.

- Creation of threads depends on processor's specifications.

② Open MP:

- A set of compiler directives as well as an API for programs written in C, C++ or FORTRAN that provides support for parallel programming in shared memory environment.

③ GCD:

- Is combination of extensions to C, an API, and a run-time library that allows application developers to identify sections of code to run in parallel.

→ Signal Handling:

- To notify a process that a particular event has occured.

Signal generated → delivered to process → handled.

- Signal handling is easy in single-threaded program, as signal are delivered to the process.
- However, delivering signals is complicated in multi-threaded programs.
  - Deliver to the thread to which the signal applies.
  - Deliver to every thread in process.
  - Deliver to certain thread in process.
  - Assign a specific thread to receive all signals for the process.

CPU Scheduling :-

→ CPU - I/O Burst Cycle:

- Process execution consist of a cycle of CPU execution & I/O execution.
- CPU burst is followed by I/O burst

→ Processes:

- CPU bound: Require most of time on CPU
- I/O bound: Require most of time on I/O devices.

CPU Scheduling

Non-Preemptive	Preemptive
- Rigid	- Flexible
- Can not interrupt b/w execution.	- Can interrupt b/w execution
- CPU allocated till process terminates or switches to waiting state	- CPU allocated for limited time
- No context switching hence no overhead of switching.	- Overhead of switching processes and maintaining ready queue.

* Burst time / Execution time / Running time : Time process requires for running on the CPU.

* Waiting time : Time spend by a process in ready state waiting for CPU.

* Arrival Time : When a process enters ready state.

* Exit Time : When a process completes execution and exit from the system.

* Turn Around Time : Total time spend by a process in the system.

$$T.A.T = E.T - A.T = B.T + W.T$$

* Response time : Time between which a process enters ready queue and get scheduled on the CPU for the first time.

→ Criteria for Scheduling Algorithm :

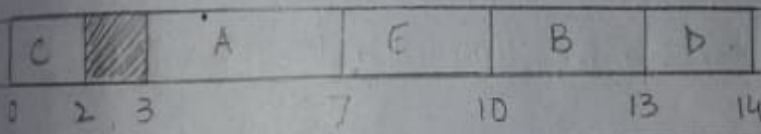
- Avg waiting time → Min
- Avg response time → Min
- CPU utilization → Max
- Throughput → Max
  (No. of process executing per unit time)

→ First Come First Serve Algorithm:

- No starvation
- Can serve from convoy effect
- Implemented using queue
- Non-premptive

Pid	A.T	B.T	C.T - A.T	W.T = TAT - BT
A	3	4	7-3 = 4	0
B	5	3	13-5 = 8	5
C	0	2	2-0 = 2	0
D	5	1	14-5 = 9	8
E	4	3	10-4 = 6	3

Gannit Chart:



Avg. TAT = 5.8

Avg. W.T = 3.2

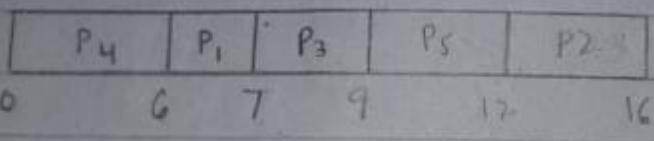
* Convoy effect: Smaller process have to wait for long time for bigger process to release CPU.

* Starvation: If process has to wait because processor is biased

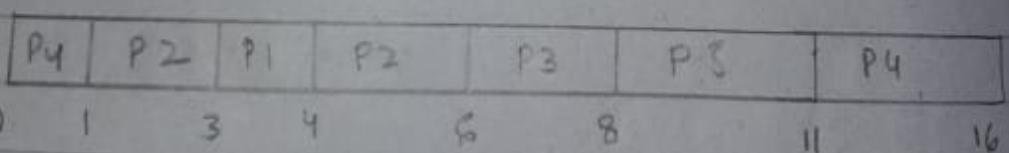
→ Shortest - Job - First (Non - preemptive) :

• - CPU is assigned to the process having smallest burst time.

Pid	A.T	B.T	T.A.T	W.T
P_1	3	1	7-3 = 4	3
P_2	1	4	16-1 = 15	11
P_3	4	2	9-4 = 5	3
P_4	0	6	6-0 = 6	0
P_5	2	3	12-2 = 10	7



→ Shortest - Remaining Time First (Preemptive) :



TAT	W.T
4 - 3 = 1	0
6 - 1 = 5	1
8 - 4 = 4	2
16 - 0 = 16	10
11 - 2 = 9	6

• - Gives minimal avg. wt.

• - Can not be implemented.

• - Standard algo.

- Processes starved.

→ Priority Algorithm:

- CPU is allocated to the processes with the highest priority.

Non-preemptive: (suppose highest number have higher priority)

Pid	AT	BT	Priority	TAT	WT
P1	0	4	2	4 - 0 = 4	0
P2	1	3	3	15 - 1 = 14	11
P3	2	1	4	12 - 2 = 10	9
P4	3	5	5	9 - 3 = 6	1
P5	4	2	5	11 - 4 = 7	5

P1	P4	P5	P3	P2
0	4	9	11	12	15

Preemptive:

P1	P2	P3	P4	P5	P2	P1
0	1	2	3	8	10	12	15

:- low priority processes suffer from starvation.

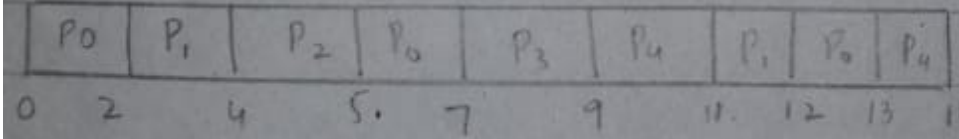
→ Round-Robin Algorithm:

- Fifo variant.
- Assigns equal time slots to each process called Time Quantum.
- Best avg. response time.
- Preemptive algo.

Pid	AT	BT	TAT	W.T
P_0	0	5		
P_1	1	3		
P_2	2	1		
P_3	3	2		
P_4	4	3		

Time Quantum = 2

P_0 P_1 P_2 P_0 P_3 P_4 P_1 P_0 P_4



- Time quantum small → overhead because of context switching.
- Time quantum large → FIFO

* CPU must ensure that memory access generated in user mode is between base and limit for that user

Memory Management : 1 (55)

* Physical Address : Actual address.
* Logical Address : Address generated by CPU.
* Base Register : Holds starting physical memory address
* Limit Register : Specifies the size of the range

* Address Binding :
  - The processes on the disk that are waiting to be brought into memory for execution, form the input queue
  - Mapping of addresses from one address space to another is called Address Binding.

Source Program —(symbolic addr)→ Compiler —(relocatable addr)→ Loader/Linker —(absolute addr)→ Exe.

* Logical Versus Physical Address Space :
  ↓ memory-address register

- At compile time & load-time address binding methods generate identical logical and physical addresses.
- However, at execution time addresses are different.
- Now logical address is refered as Virtual Address.

- Memory Management Unit: Run-time mapping from virtual to physical addresses is done by a hardware device called MMU.
- Base register is now called a relocation register.
- User program deals only with logical addresses.

* Dynamic Loading:
- Process and data must be brought into physical memory for its execution.
- For better memory utilization.
- Routine is loaded only when it is needed.
- Large sized program can be executed on a small memory.
- Loading is postponded until execution.

* Dynamic Linking and Shared Libraries:

- Dynamically linking libraries are system libraries that are linked to user program at the time of execution.

- Dynamic Linking is similar to dynamic loading. Linking is postponded until execution.

- Static linking libraries are system libraries combined by loader into the binary program image.

- Stub:
Holds addresses of library used during execution.
(OR) Small piece of code that locates appropriate library routine.

[If library is not already located in memory, then it loads library into the memory and replaces with it self or else uses already loaded library).

- Shared Libraries:
→ A version of library is loaded into memory, each program uses its version info to decide which copy of the library to use.
→ Version with minor changes retain the same version number while version with major changes increments the version.

→ Programs compiled with new version are affected by incompatible change
→ Programs linked before new version will continue using older version.
→ This system is called Shared Libraries.

* Swapping:

A process can be swapped temporarily out of memory to a backing store and then brought back into memory for continued execution.

- Standard Swapping:
  → Involves moving processes between main memory and backing store.
  → System maintains a ready queue consisting of all processes whose memory images are on the backing store or in the memory and are ready to run.
  → Then dispatcher schedules the process next in the ready queue and ~~loa~~ already loaded into the memory or else it swaps previous process with already

loaded process / ~~mem~~ in the queue.

- Backing Store:
  - → A fast disk.
  - → large enough to accomodate copies of all memory images for all users and it must provide direct access to these memory images.

→ Swapping is normally disabled because if swapping is enabled then low priority processes will starve.

→ Swapping is enabled, if more than threshold amount of memory is allocated.

* Contiguous Allocation:

→ Main memory must support both OS and user processes.

→ Allocate main memory efficiently.

→ Two Partitions
  One for resident OS
  One for user program.

→ OS is placed in lower memory because of interrupt vector.

→ Each process is placed in a single contiguous section of a memory.

- Memory Protections:

→ Issue of memory protection can be resolved by using relocation register and limit register.

→ Relocation Register : value of smallest physical address.

→ Limit Register : range of logical addresses.

- Memory Allocations:

① Multiple Partition method:

→ Memory is divided into several fixed size partitions

→ Each partition may contain exactly one process.

→ Degree of multiprogramming is bound by the no. of partitions.

→ When partition is free, process from input queue is loaded into free partition.

→ When process terminates, partition

becomes available for another process.

(2) Variable Partition Method:
→ OS keeps Table indicating available and occupied pairs of memory.
→ Hole: Available block of memory for user processes.
→ Memory contains set of holes of various sizes.
→ Processes are assigned to holes large enough to accomodate them.
→ Process exiting frees its partition & adjacent free partitions combined.

- Dynamic Storage Allocation:

→ Concerned with allocation of free holes to size-n process.
(1) First Fit: First hole that is big enough
(2) Best Fit: Smallest hole that is big enough for process.
(3) Worst Fit: Largest hole

o – Fragmentation:

① External Fragmentation:
When there is enough memory space to satisfy a request but available spaces are not contiguous.

② Internal Fragmentation:
When allocated memory is slightly larger than requested memory, this size difference is fragment internal to a partition.

Solution of External Fragmentation:

① Compaction: Shuffle memory contents and place all free memory in one large block.
Only possible while dynamic relocation

② Permit the logical address bit space to be non-contiguous, thus allowing processes to be allocated physical memory wherever such memory is available.

* Segmentation:
  - User's view of program.
  - Dividing a program in variable length segments, can be non contiguous
  eg: S_1 → main program
  S_2 → function 1
  S_3 → stack
  S_4 → object
  etc.
  - Logical address space is the collection of segments.
  <segment no., offset>.
  - Segment table holds base address and limits of segments.

* Paging:
  - Divides physical memory into fixed sized blocks called frames.
  - Divides logical memory into blocks of same size called pages.

Virtual Memory Management

* Virtual memory is a technique that allows the execution of the processes that are not completely in the memory.
* One advantage of this scheme is that programs can be larger than physical memory.
* Virtual memory can share files
* Processes can implement shared memory.

→ Demand Paging :-

* Load pages only when they are demanded is called demand paging.
* Pager swaps the pages into the memory only when it is required.

* Valid / Invalid bit scheme :-

If bit is set

Valid (v) : Page is legal & resides in the memory

Invalid (i) : Page is invalid or it resides on the disk

* Page fault : -

If a process tries to access page that was not loaded into the memory or marked invalid, then it causes page fault.

* Page fault handling:

① When page is needed check reference from internal table.
→ Invalid then abort
→ Valid but not in memory then page it in.
② Find free frame and schedule disk operation to read desired page into newly allocated frame.
③ Modify page table to indicate that page is loaded.
④ Restart the instruction that was interrupted by trap (page fault).

* Pure demand paging:
→ starts executing a process with no pages in memory.
→ When OS sets IP to first instruction of process, which is on non-memory resident page it causes page fault.
→ Continues execution & continues causing page fault until every page needs is in the memory.
→ Called Pure demand paging.

* Hardware Support for Demand Paging:
① Page Table: Marks an entry of page invalid through valid/invalid bit.
② Secondary Memory: Holds pages that are not in the memory called swap device.

* Performance of Demand-Paging:

EAT = (1 - P) ma + P * fault time

where

P = probability of page fault.

Ex: ma = 200ns , fault time = 8ns

$P = \frac{1}{1000} = 0.001$

EAT = (1 - 0.001) 200 + 0.001 * 8
= 8199.8 ns.

→ Copy-On-Write:
* Allows parent & child to share the same pages in the memory.
* If either process modifies a shared page, only then page is copied.
* Pool of free pages will allocate requested free-page for copying.

→ Page - Replacement:

* Required when there is no free frame.

① Basic Page Replacement:
* Find desired page on the disk.
* Find free frame.
  - If there is free frame, use it.
  - If there is no free frame, use page-replacement algorithm to select a victim frame
  - Change page & frame tables accordingly
* Read desired page into newly freed frame
* Restart the user process.

* Modify bit / Dirty bit is used to reduce overhead of page transfers (page in - page out). Only writes modified pages to the disk.

② FIFO. (First in first out)
③ Optimal Page Replacement. (Jo page dor hai usko replace kra)
④ Least Recently Used Page Replacement
(Least used page ko replace kra excluding page hit wala page.

5) Second - Chance Algorithm: (Fifo Variant)
* Set reference bit.
* Reference bit of page hit is 1.

a) 2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2.

2	2	2*	2	2	2*	2	2				
	3	3	3	5	5	5	5*				
			1	1	1	4	4				
2	2*	2	2*								
5	5	5*	5								
3	3	3	3								

Page hit = 6
Page fault = 6.

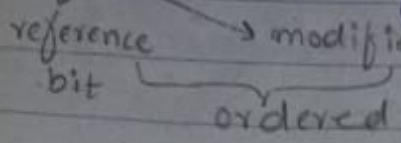
b) 2, 4, 5, 2, 2, 3, 5, 1, 2, 3, 1, 2, 3, 5, 6, 7, 8

2	2	2	2*	2*	2	2	1	1	1	1*	1	1	1	6	6	6
4	4	4	4	3	3	3	2	2	2	2	2*	2	2	2	7	7
	5	5	5	5	5*	5	5	3	3	3	3	3*	3	3	3	8

Page hit = 6 Page fault = 11

⑥ Enhanced Second Chance Algorithm:

(0, 0) → neither recently used nor modified.
(0, 1) → not recently used but modified.
(1, 0) → recently used but clean.
(1, 1) → recently used & modified.



* Give preference to those pages that are modified, to reduce the no. of I/O.

⑦ Counting Based Page Replacement:

* Keep a counter of the no of references that have been made to each page.

(a) Least Frequently Used:

* Page with the smallest count be replaced.
* Problem: Page is used heavily during the initial phase of the process but then is never used again. Such page has a large count and remains in memory even if not needed.

(b) Most Frequently Used: (Bar bar reference wala replace krnge)
Page with smallest count was probably just brought in and has yet to be used.

(c) Page-Buffering Algorithms:-
* Keep a pool of free frames.
* Keep a list of modified (COW) pages
* Keep free frame contents intact & note what is in it.

→ Allocation of Frames:

* Minimum no. of frames:
- Must allocate atleast a minimum number of frames.
- No. of frames allocated to each process decreases, page fault rate increases.
- Minimum number of frames per process is defined by the computer architecture.
- Maximum number of frames is defined by the amount available in physical memory.

* Allocation Algorithm:

① Equal Allocation:

- Allocate each process an equal no. of frame.

$$\text{Allocation} = \frac{\text{No. of frames}}{\text{No. of process}}$$

- Left over frames can be used as free frame buffer pool.

② Proportional Allocation:

- Allocate available memory to each process according to its size.

$$S = \sum s_i$$

$$a_i = \frac{s_i}{S} \times m$$

where S = virtual memory size for all process.
s_i = size of virtual memory for process P_i.
m = total no. of frames available.
a_i = frames allocated to process P_i.

eg 62 frames, P_1 = 10 pages, P_2 = 127 pages

$$a_1 = \frac{10}{(10+127)} \times 62 \approx 4$$

$a_2 = \frac{127}{(10+127)} \times 62 \approx 57.$

* Global v/s Local Allocation:

① Global Allocation: Allows a process to select a replacement frame from set of all frames, even if that frame is allocated to some other process. Cannot control its own page fault.

② Local Allocation: Allows a process to select a replacement frame from only its own set of allocated frames.

* Non-Uniform Memory Access:-
Systems in which some parts of memory may take longer time to access than other parts, creating performance penalty are called NUMA systems.

→ Thrashing :=.
- Page in page out on the basis of demand.
- A process is thrashing if is spending more time paging than executing.

* Locality - Model:
As a process, executes, it moves from Locality to locality. A Locality is a set of pages that are actively used together.

* Working Set Model:
- Working set Model sets a fixed size window for locality models.
- Accuracy of Working set depends on the selection of Δ
If Δ is too small, it will not encompass entire locality.
If Δ is too large, it may overlap several localities.
If Δ is infinite, then it is the set of pages used during the process execution.

Demanded frames = $D = \sum WSS_i$
Total no. of frames
If $D > (m) \Rightarrow$ thrashing occurs

- Decreases thrashing, increases the degree of multiprogramming & optimizes CPU utilization.

* Page - Fault Frequency:-
- It is a direct measure to control thrashing that is page fault rate.
- Establishes upper and lower bound for the desired page fault rate.
- If the actual page-fault rate exceeds the upper limit, then allocate process more frames.
- If the actual page-fault rate decreases the lower limit, then remove a frame from process.

→ Memory Mapped files:
- Memory mapping a file, allows a part of virtual address space to be logically associated with a file.
- Multiple processes may be allowed to map the same file concurrently, to allow sharing of data.
- It supports copy on write functionality.
- Memory mapped files serve as the region of shared memory between the communicating processes.

→ Allocating Kernel Memory :-

Kernel memory is allocated from a free memory pool

① Buddy System Allocater:

* Buddy System allocates memory from a fixed-size segment consisting of physically contiguous pages.
* Memory is allocated using a power-of-2 allocater.

$256 = 2^8$

$128 = 2^7$ $\quad$ $128 = 2^7$

$64 = 2^6$ $\quad$ $64 = 2^6$

$32 = 2^5$ $\quad$ $32 = 2^5$

→ fulfils request of 21 Kb

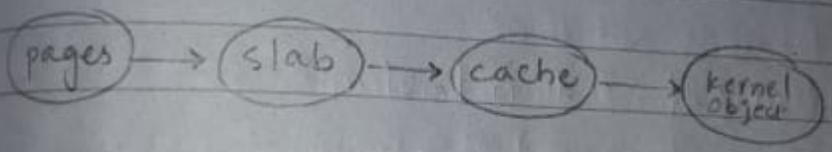
* Kernel request 21 KB from memory segment of 256 KB.

* Advantage: Adjacent buddies can be quickly combined to form larger segments using technique called Coalescing.

* Drawback: Causes fragmentation.

③ Slab Allocater:

* Slab is made up of one or more physically contiguous pages
* Cache consists of one or more slabs
* This algorithm uses caches to store kernel objects. When cache is created, the number of objects - initially marked as free, are allocated to cache. The number of objects in the slab cache depends on the size of the associated slab.



* Advantage:
  Prevents internal fragmentation.
  Memory requests can be satisfied quickly.

→ Other Consideration:

② Page size
③ TLB Reach
④ I/O interlock

} from slides

① Prepaging:

* To reduce high level initial page fault.
* Prepaging brings all the pages required into the memory at one time.

:- Process Synchronization :-

* Cooperating Process:
One that can affect or be affected by other processes executing the system.

* Race condition:
When several processes access and manipulate the same data concurrently and the outcome of the execution depends on the particular order in which the access takes place is called a race condition.

→ Critical - Section Problem:

Section of code in which only one process is allowed to be executed is called Critical section.

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

② Progress:
If no process is executing in its critical section and some processes wish to enter their critical sections, then only those processes that are not executing in their remainder sections can participate in deciding which will enter its critical section next and this selection cannot be postponed indefinitely.

③ Bounded waiting:
There exists a bound, or 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 & before that request is granted.

* We cannot make an assumption concerning the relative speed of the n-processes.

* Approches to handle critical section:

① Preemptive Kernels:
Allows a process to be preempted while it is running in Kernel mode.
Suitable for real time programming.

② Nonpreemptive kernels:
Doesn't allow a process running in kernel mode to be preempted.
Free from race condition.

→ Peterson's Solution:-
* Not guaranteed to work on modern architecture.
* It provides a good algorithmic description of solving the critical-section problem.

* Peterson's solution is restricted to two processes that alternate execution between their critical sections and remainder sections
* It has two data items
① int turn; (indicate whose turn is to enter its critical section)
② boolean flag[2]; (indicate if a process is ready to enter its critical section).

→ Synchronization Hardware:-

* Race conditions are prevented by requiring that critical regions be protected by locks. That is, process must acquire a lock before entering a critical region & releases the lock when it exits the critical region.

* Solution of Critical Section Problem in Uniprocessor:
If we could prevent interrupt from occuring while a shared variable was being modified. Then no other instructions would be run, so no unexpected modification could be made to the shared variable. This approach is taken by nonpreemptive kernels.

* Two instruction for multiprocessor system:

① TestAndSet( ):
TestAndSet( ) instruction executes automatically, thus if two TestAndSet( ) instructions are executed, they will be executed sequentially in some arbitrary order. We can achieve mutual exclusion by declaring boolean variable lock.

② Swap( ) :

Swap( ) instruction also executes automatically. It operates on the contents of two words. Mutual exclusion can be achieved by declaring a global variable lock and each process have a local boolean variable key.

- Above mentioned algorithms of two instructions only satisfies mutual exclusion.

→ Semaphores:

(Don't works on busy waiting)

A semaphore S is an integer variable that, apart from initialization, is accessed only through two standard atomic operations: wait( ) and signal( ).

* wait( ) → to test. (entry section)

```
wait(S)
{ while (S<=0)
      ;     // no operation.
   S--;
}
```

* signal( ) → to increment (exit section).

```
signal(S){
    S++;
}
```

① Counting Semaphores:
* Value of counting semaphore can range over an unrestricted domain.
* It is used to control access to a given resource.

② Binary Semaphores:
* Value of binary semaphore can range only between 0 and 1.
* Binary semaphores are also called mutex locks, as they are locks that provide mutual exclusion.

* Spinlock Semaphore: (mutex lock)
It works on busy waiting. Busy waiting wastes CPU cycles that some other process might be able to use productively.
Spinlocks are useful when locks are expected to be held for short time.

* Modified Semaphore operation:

- Overcomes the busy waiting

- When process executes wait( ) operation & finds that the semaphore value is not positive, it must wait. Process must be blocked & placed in a waiting queue. Then control of CPU is transferred to another process to be executed.
- A process that is blocked should be restarted when process executes a signal( ) operation.

- If a semaphore value is negative, its magnitude is the number of processes waiting on that semaphore.

* Deadlocks And Starvation:.

Deadlock: A situation where each of a process is waiting for an event from other process. Since all are waiting, none can provide an event being waited for creates a deadlock.

Conditions for deadlock:

① At least one resource must be held is in non-shareable mode.
② There exists a process holding a resource and waiting for another process.
③ Resources cannot be preempted
④ Processes are in circular waiting.

* Priority Inversion:

A scheduling challenge arises when a higher priority process needs to read or modify kernel data that are currently being accessed by lower priority process or a chain of lower priority processes. Since kernel data are typically protected with a lock, the higher priority process would have to wait for lower priority one to finish with the resource.

→ Classical Problems of Synchronization:

① Bounded-buffer Problem.
② Readers-Writers Problem.
③ Dining Philosophers Problem.