Operating System Concepts

SunBeam Institute of Information & Technology, Hinjwadi, Pune & Karad.

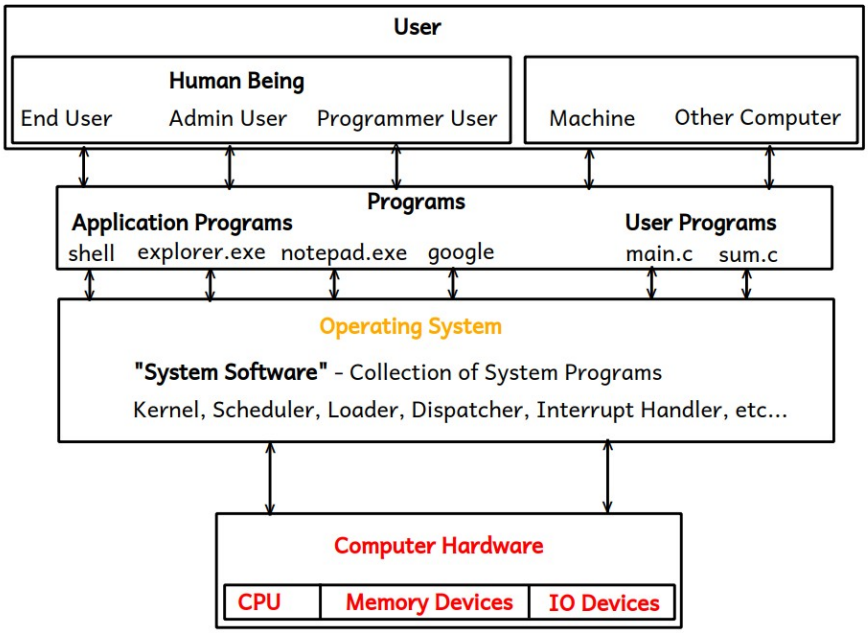
Q. Why there is a need of an OS?

- Computer is a **machine/hardware** does different tasks efficiently & accurately.

- Basic functions of computer:

1. **Data Storage**
2. **Data Processing**
3. **Data Movement**
4. **Control**

- As any user cannot communicates/interacts directly with computer hardware to do different tasks, hence there is need of some interface between user and hardware and to provide this interface is the job of an OS.



Q. What is a Software?

- Software is a collection of programs.

Q. What is a Program?

- Program is a **finite set of instructions written in any programming language** (either low level or high level programming language) **given to the machine to do specific task.**

- 3 types of programs are there:

1. "user programs": programs defined by the programmer user/developers
e.g. main.c, hello.java, addition.cpp etc....

2. "application programs": programs which comes with an OS / can be installed later
e.g. MS Office, Notepad, Compiler, IDE's, Google Chrome, Mozilla Firefox, Calculator, Games etc....

3. "System Programs": programs which are inbuilt in an OS/part of an OS.
e.g. Kernel, Loader, Scheduler, Memory Manager etc...

Q. What is an IDE (Integrated Software Developement) ?

- It is an **application software** i.e. collection of tools/application programs like **source code editor, preprocessor, compiler, linker, debugger** etc... required for **faster software developement.**

e.g. VS code editor, MS Visual Studio, Netbeans, Android Studio, Turbo C etc....

Source Code – Program written in any programming langauge.

1. "Editor": it is an application program used to write a source code.

e.g. notepad, vi editor, gedit etc...

2. "Preprocessor": it is an application program gets executes before compilation and does two jobs - **it executes all preprocessor directives** and **removes all comments from the source code.**

e.g. cpp

3. "Compiler": it is an application program which **converts high level programming language code into low level programming language code i.e. human understandable language code into the machine understandable language code.**

e.g. gcc, tc, visual c etc...

4. "Assembler": it is an application program which converts assembly language code into machine language code/object code.

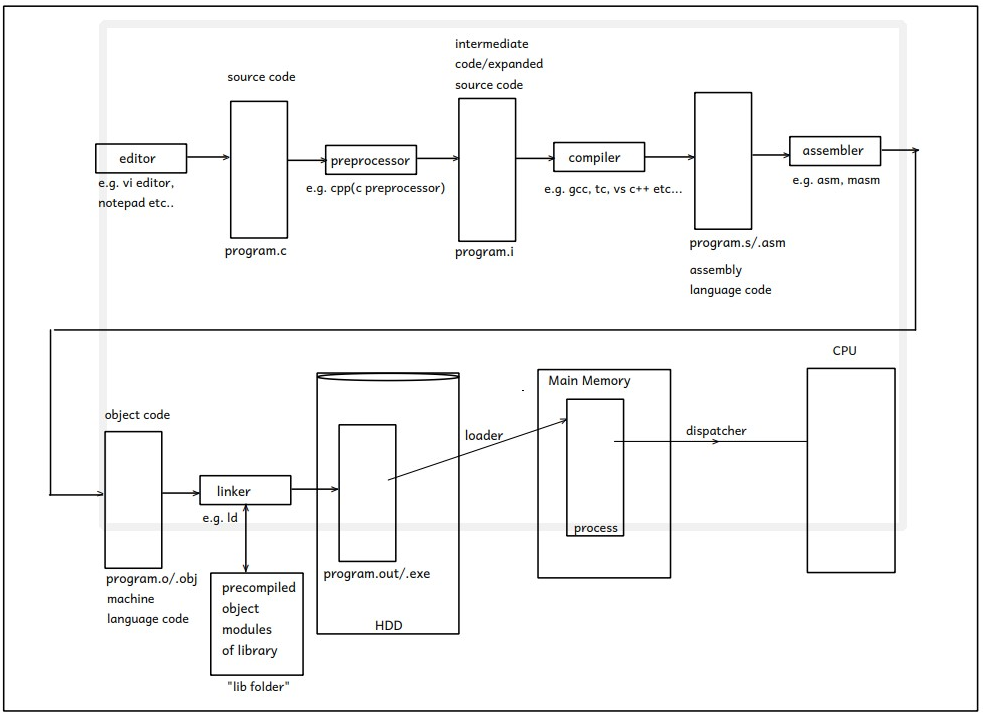
e.g. masm, tasm etc...

5. "Linker": it is system program which links object file/s of a program with precompiled object modules of library functions exists in a lib folder and creates final **single executable file.**

e.g. **ld**: link editor in Linux.

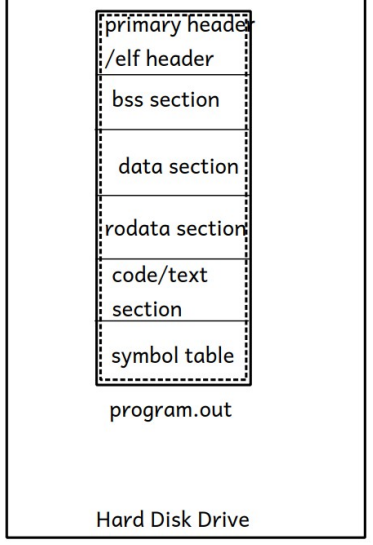
Loader : It is a system program (i.e. inbuilt program of an OS) which loads an executable file from **HDD** into the **main memory.**

Dispatcher : It is a system program (i.e. inbuilt program of an OS) which loads data & instructions of a program which is in the **main memory** onto **the CPU** (i.e. into the CPU registers).



Operating System Concepts

Structure of an executable file
ELF file format in Linux



1. **primary header/exe header:** it contains information which is required to starts an execution of the program.
   e.g. - addr of an entry point function --> main() function
   - **magic number:** it is constant number generated by the compiler which is file format specific.
   - magic number in Linux starts with ELF in its eq **hexadecimal format**.
   - info about remaining sections.
2. **bss(block started by symbol) section:** it contains uninitialized global & static vars
3. **data section:** it contains initialized global & static vars
4. **rodata (readonly data) section:** it contains string literals and constants.
5. **code/text section:** it contains an executable instructions
6. **symbol table:** it contains info about functions and its vars in a tabular format.

Q. What is an Operating System?

- An OS is a **system software** (i.e. collection of system programs) which **acts as an interface between user and hardware.**

- An OS also **acts as an interface between programs(user & application programs) and hardware.**

- An OS allocates resources like **main memory, CPU time, i/o devices access** etc... to all running programs, hence it is also called as a **resource allocator.**

- An OS controls an execution of all programs, it also controls hardware devices which are connected to the computer system and hence it is also called as a **control program.**

Q. What is an Operating System?

- An OS manages limited available resources among all running programs, hence it is also called as a **resource manager.**

- An OS is a software (i.e. collection of thousands of system programs and utitlity programs in a binary format) comes either in CD/DVD/PD, has following 3 main components:

1. Kernel: It is a **core program/part of an OS** which runs continuosly into the main memory does **basic minimal functionalities** of it.

e.g. Linux: **vmlinuz**, Windows: **ntoskrnl.exe / win32krnl.dll**

2. Utility Programs - Softwares: e.g. disk manager, windows firewall, anti-virus software etc...

3. Application Programs - Softwares: e.g. google chrome, shell, notepad, msoffice etc...

 # Functions of an OS:

Basic minimal functionalities/Kernel functionalities – must be supported by any OS – compulsory functionalities :

1. Process Management
2. Memory Management
3. Hardware Abstraction
4. CPU Scheduling
5. File & IO Management

Extra utility functionalities/optional:

6. Protection & Security
7. User Interfacing
8. Networking

 # UNIX Operating System:

- UNIX: UNICS – **Uniplexed Information & Computing Services/System.**

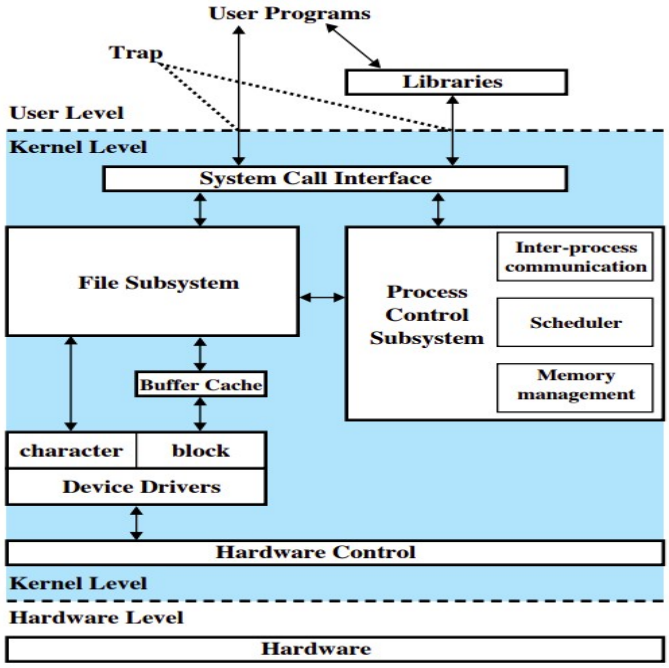
- UNIX was developed at **AT&T Bell Labs** in US, in the decade of 1970's by **Ken Thompson, Denies Ritchie** and team.

- It was first run on a machine **DEC-PDP-7** (Digital Equipment Corporation – Programmable Data Processing-7).

- UNIX is the first **multi-user, multi-programming** & **multi-tasking** operating system.

- UNIX was specially designed for developers by developers

- System architecture design of UNIX is followed by all modern OS's like Windows, Linux, MAC OS X, Android etc..., and hence UNIX is referred as **mother of all modern operating systems.**



- Kernel acts as an interface between programs and hardware.
- Operating System has subsystems like **System Call Interface Block, File Subsystem Block, Process Control Subsystem Block (which contains IPC, Memory Management & CPU Scheduling), Device Driver, Hardware Control/Hardware Abstraction Layer.**
- There are two major subsystems:

1. Process Control Subsystem
2. File Subsystem

- In UNIX, whatever is that can be stored is considered as a file and whatever is active is reffered as a process.
- **File has space & Process has life.**

- From UNIX point of view all devices are considered as a file

- In UNIX, devices are catagorises into two catagories:

1. Character Devices: Devices from which data gets transfered character by character --> character special device file

e.g. keyboard, mouse, printer, monitor etc...

2. Block Devices: Devices from which data gets transfered block by block --> block special device file

e.g. all storage devices.

- **Device Driver:**It is a program/set of programs enable one or more hardware devices to communicate with the computer's operating system.

- Hardware Control Layer/Block does communication with control logic block i.e. controller of a hardware.

 # System Calls: are the functions defined in a C, C++ & Assembly languages, which provides interface of services made available by the kernel for the user (programmer user).

- If programmers want to use kernel services in their programs, it can be called directly through system calls or indirectly through set of libary functions provided by that programming language.

- There are 6 catagories of system calls:

1. Process Control System Calls: e.g. fork(), _exit(), wait() etc...

2. File Operations System Calls: e.g. open(), read(), write(), close() etc...

3. Device Control System Calls: e.g. open(), read(), write(), ioctl() etc...

4. Accounting Information System Calls: e.g. getpid(), getppid(), stat() etc...

5. Protection & Security System Calls: e.g. chmod(), chown() etc...

6. Inter Process Communication System Calls: e.g. pipe(), signal(), msgget() etc...

- In UNIX 64 system calls are there.

- In Linux more than 300 system calls are there

- In Windows more than 3000 system calls are there

- When system call gets called the CPU switched from user defined code to system defined code, and hence system calls are also called as **software interrupts/trap.**

 # Dual Mode Operation:

- System runs in two modes:

1. System Mode
2. User Mode

1. System Mode:

- When the CPU executes system defined code instructions, system runs in a system mode.

- System mode is also reffered as kernel mode/monitor mode/supervisor mode/priviledged mode.

2. User Mode:

- When the CPU executes user defined code instructions, system runs in a user mode.

- User mode is also reffered as non-priviledged mode.

- Throughout execution, the CPU keeps switch between kernel mode and user mode

Dual Mode Operation:

- Throughout an execution of any program, the CPU keeps switcesh in between kernel mode and user mode and hence system runs in two modes, it is reffered as **dual mode operation.**

- To differentiate between user mode and kernel mode one bit is there onto the CPU which is maintained by an OS, called as **mode bit**, by which the CPU identifies wheather currently executing instruction is of either system defined code instruction/s or user defined code instruction/s.

- In Kernel mode value of **mode bit = 0**, whereas

- In User mode **mode bit = 1.**

Process Management

- When we say an OS does process management it means an OS is responsible for **process creation**, **to provide environment for an execution of a process**, **resource allocation, scheduling, resources management, inter process comminication, process coordination**, and **terminate the process.**

Q. What is a Program?

- **User view:** Program is a finite set of instructions written in any programming language given to the machine to do specific task.

- **System view:** Program is an **executable file** in HDD which divided logically into sections like **exe header, bss section, data section, rodata section, code section, symbol table.**

Q. What is a Process?

User view:

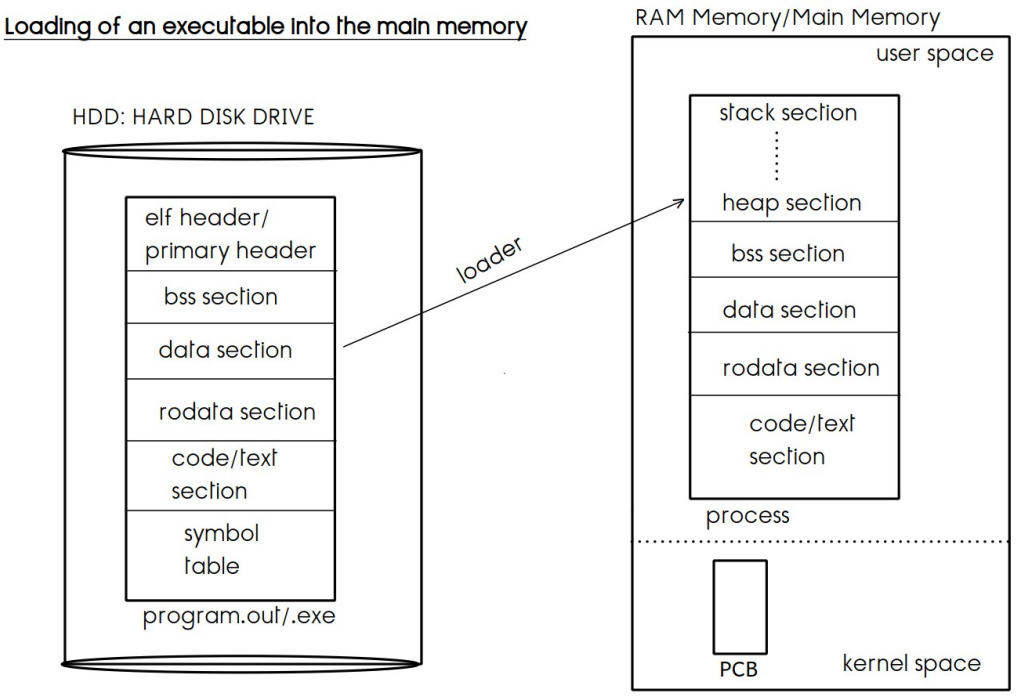
- Program in execution is called as a process.
- Running program is called as a process.
- When a program gets loaded into the main memory it is reffered as a process.
- Running instance of a program is reffered as a process.

System view:

- Process is a program loaded into the main memory which has got PCB into the main memory inside kernel space and program itself into the main memory inside user space has got **bss section, rodata section, code section**, and two new sections gets added for the process:

stack section: contains function activation records of called functions.

heap section: dynamically allocated memory



- As a kernel, core program of an a OS runs continuosly into the main memory, **part of the main memory which is occupied by the kernel reffered as kernel space and whichever part is left is reffered as an user space**, so main memory is divided logically into two parts: **kernel space & user space.**

- User programs gets loaded into the user space only.

- When we execute a program or upon submission of a process very first one structure gets created into the main memory inside kernel space by an OS in which all the information which is required to control an execution of that process can be kept, this structure is reffered as a **PCB: Process Control Block**, is also called as a **Process Descriptor.**

- Per process one PCB gets created and PCB remains inside the main memory throughout an execution of a program, upon exit PCB gets destroyed from the main memory.

- PCB mainly contains: PID, PPID, PC, CPU sched information, memory management information, information about resources allocated for that process, execution context etc...

 # Process States:

- Throughout execution, process goes through different states out of which at a time it can be only in a one state.

- States of the process:

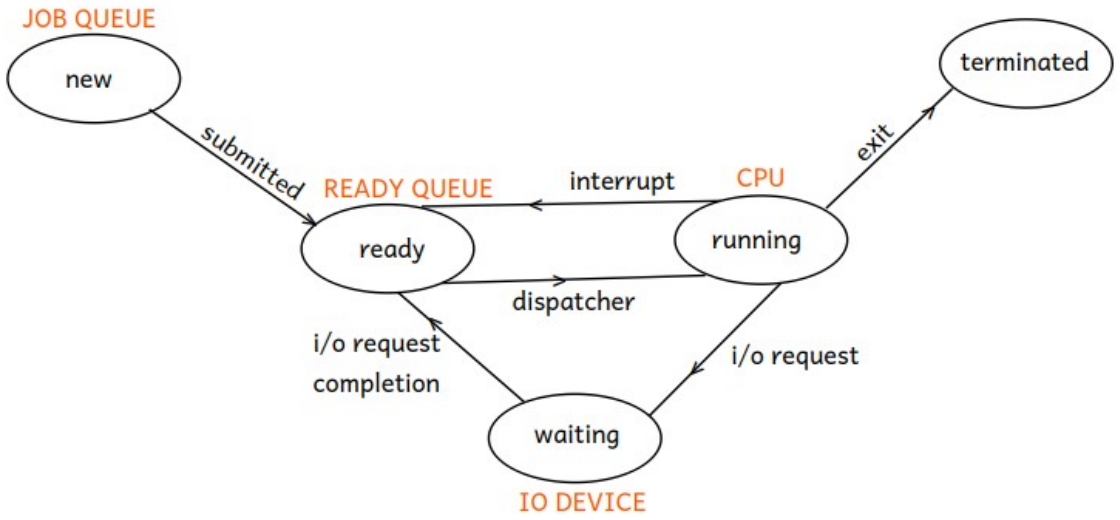
1. New state: upon submission or when a PCB for a process gets created into the main memory process is in a new state.

2. Ready state: after submission, if process is in the main memory and waiting for the CPU time, it is in a ready state.

3. Running state: if currently the CPU is executing any process then state of that process is considered as a running state.

4. Waiting state: if a process is requesting for any i/o device then state of that process is considered as a waiting state.

5. Terminated state: upon exit, process goes into terminated state and its PCB gets destroyed from the main memory.



PROCESS STATE DIAGRAM

1. multi-programming: system in which more than one processes can be submitted/an execution of more than one programs can be started at a time.

- **degree of multi-programming:** no. of programs that can be submitted into the system at a time.

2. multi-tasking: system in which, the CPU can execute more than one programs simultaneously/concurrently, the speed at which it executes multiple programs simultaneosuly it seems that it executes multiple programs at a time.

At a time the CPU can execute only one program

- **time-sharing:** system in which CPU time gets shared among all running programs.

3. multi-threading: system in which it seems that the CPU can execute more than one threads which are of either same process or are of different processes simultaneously/concurrently.

4. multi-processor: system can run on a machine in which more than one CPU's are connected in a closed circuit.

5. multi-user: system in which multiple users can loggedin at a time.

 # Process States:

- To keep track on all running programs, an OS maintains few **data structures** reffered as **kernel data structures:**

1. Job queue: it contains list of PCB's of all submitted processes.

2. Ready queue: it contains list of PCB's of processes which are in the main memory and waiting for the CPU time.

3. Waiting queue: it contains list of PCB's of processes which are requesting for that particular device.

1. Job Scheduler/Long Term Scheduler: it is a system program which selects/schedules jobs/processes from job queue to load them onto the ready queue.

2. CPU Scheduler/Short Term Scheduler: it is a system program which selects/schedules job/process from ready queue to load it onto the CPU.

- Dispatcher: it is a system program which loads a process onto the CPU which is scheduled by the CPU scheduler, and the time required for the dispatcher to stops an execution of one process and to starts an execution of another process is reffered as **dispatcher latency.**

 # Context Switch:

- As during context-switch, the CPU gets switched from an execution context of one process onto an execution context of another process, and hence it is reffered as "context-switch".

- context-switch = state-save + state-restore

- **state-save** of suspended process can be done i.e. an execution context of suspended process gets saved into its PCB.

- state-restore of a process which is scheduled by the CPU scheduler can be done by the dispacther, dispather copies an execution context of process scheduled by the cpu scheduler from its PCB and restore it onto the CPU registers.

- When a high priority process arrived into the ready queue, low priority process gets suspended by means of sending an interrupt, and control of the CPU gets allocated to the high priority process, and its execution gets completed first, then low priority process can be resumed back, i.e. the CPU starts executing suspended process from the point at which it was suspended and onwards.

- CPU Scheduler gets called in the following four cases:

Case-1: Running -> Terminated

Case-2: Running -> Waiting

Case-3: Running -> Ready

Case-4: Waiting -> Ready

- There are two types of CPU scheduling:

1. Non-preemptive: under non-preemptive cpu scheduling, process releases the control of the CPU by its own i.e. voluntarily.

e.g. in above case 1 & case 2

2. Preemptive: under preemptive cpu scheduling, control of the CPU taken away forcefully from the process.

e.g. in above case 3 & 4.

- Following algorithms used for CPU Scheduling:

1. FCFS (First Come First Served) CPU Scheduling

2. SJF (Shortest Job First) CPU Scheduling

3. Round Robin CPU Scheduling

4. Priority CPU Scheduling

- Multiple algorithms are there for CPU scheduling, so there is need to decide which algorithm is best suited at specific situation and which algorithm is an efficient one, to decide this there are certain criterias called as **scheduling criterias**: **cpu utilization, throughput, waiting time, response time and turn-around-time.**

 # CPU Scheduling Criterias:

1. CPU Utilization: one need to select such an algorithm in which utilization of the CPU must be as **maximum** as a possible.

2. Throughput: total work done per unit time. One need to select such an algorithm in which throughput must be as **maximum** possible.

3. Waiting Time: it is the toal amount of time spent by the process into the ready queue for waiting to get control of the CPU from its time of submission. One need to select such an algorithm in which waiting time must be as **minimum** as possible.

4. Response Time: it is a time required for the process to get first response from the CPU from its time of submission. One need to select such an algorithm in which response time must be as **minimum** as possible.

5. Turn-Around-Time: it is the total amount of time required for the process to complete its execution from its time of submission.

One need to select such an algorithm in which turn-around-time must be as **minimum** as possible.

- Turn-around-time is the sum of periods spent by the process into ready queue for waiting and onto the CPU for execution from its time of submission.

Execution Time: it is the total amount of time spent by the process onto the CPU to complete its execution.

CPU Burst Time: total no. of CPU cycles required for the process to complete its execution.

- **Turn-Around-Time = Waiting Time + Execution Time.**

1. FCFS (First Come First Served) CPU Scheduling

- In this algorithm, process which is arrived first into the ready queue gets the control of the CPU first i.e. control of the CPU gets allocated for processes as per their order of an arrival into the ready queue.

- This algorithm is simple to impelement and can be implemented by using fifo queue.

- It is a non-preemptive scheduling algorithm.

 # Convoy effect: due to an arrival of longer processes before shorter processes, shorter processes has to wait for longer time and their average waiting time gets increases, which results into an increase in an average turn-around-time and hence overall system performance gets down.

2. SJF(Shortest Job First) CPU Scheduling:

- In this algorithm, process which is having minimum CPU burst time gets the control of the CPU first.

- SJF algorithm ensures minimum waiting time.

- Under non-preemptive SJF, algorithm fails if the submission time of processes are not same, and hence it can be implemented as preemptive as well.

- Non-preemptive SJF is also called as **SNTF(Shortest-Next-Time-First).**

- Preemptive SJF is also called as **SRTF(Shortest-Remaining-Time-First).**

Starvation: in this algorithm, as shorter processes has got higher priority, process which is having larger CPU burst time may gets blocked i.e. control of the CPU will never gets allocated for it, such situation is called as **starvation/indefinite blocking.**

3. Round Robin Scheduling Algorithm

- In this algorithm, before allocating the CPU for processes, some fixed **time slice** or **time quantum** gets decided in advanced, and at any given time control of the CPU may remains allocated with any process maximum for that decided time-slice, once the given time slice is finished of that process, it gets suspended and control of the CPU will be allocated to the next process again for maximum that decided time slice and so on..., each process gets control of the CPU in a round robin manner i.e. cpu gets shared among processes equally.

- If any process completes its execution before allocated time slice then control of the CPU will be released by that process and CPU gets allocated to the next process as soon as it is completed for effective utilization of the CPU.

- **There is no starvation** in RR Scheduling algorithm.

- This algorithm is **purely preemptive.**

- This algorithm ensures **minimum response time.**

- If time slice is minimum then there will be extra overhead onto the CPU due to frequent context-switch.

Operating System Concepts

4. Priority Scheduling

- In this algorithm, **process which is having highest priority gets control of the CPU first,** each process is having priority in its PCB.
- priority for a process can be decided by two ways:

1. internally – priority for process can be decided by an OS depends on no. of resources required for it.

2. externally – priority for process can be decided by the user depends on requirement.

- **Minimum priority value indicates highest priority.**

- This algorithm is **purely preemptive.**

- **Due to the very low priority process may gets blocked into the ready queue** and control of the CPU will never gets allocated for such a process, this situation is reffered as a **starvation** or **indifinite blocking.**

- **Ageing:** it is a technique in which, **an OS gradually increments priority of blocked process**, i.e. priority of blocked process gets incremented after some fixed time interval by an OS, so that priority of blocked process becomes sufficient enough to get control of the CPU, and starvation can be avoided.

Inter Process Communication

- Processes running into the system can be divided into two catagories:

1. Independent Processes:

- Process which do not shares data (i.e. resources) with any other process reffered as an **independent process**. **OR** Process which **do not affects** or **not gets affected by any other process** reffered as an independent process.

2. Co-operative Processes:

- Process which shares data (i.e. resources) with any other process reffered as co-operative process. **OR** Process which affects or gets affected by any other process reffered as co-operative process.

Q. Why there is need of an IPC?

As concurrently executing co-operative processes shares common resources, so there are quite chances to occur conflictions between them and to avoid this conflictions there is a need of communication takes place between them.

What is an Inter Process Communication?

- An IPC is one of the important **service made available by the kernel, by using which co-operative processes can communicates with each other.**

- Inter process communication takes place only between **co-operative processes.**

- Any process cannot directly communicates with any other process, hence there is a need of some medium, to provide this medium is the job of an OS/Kernel.

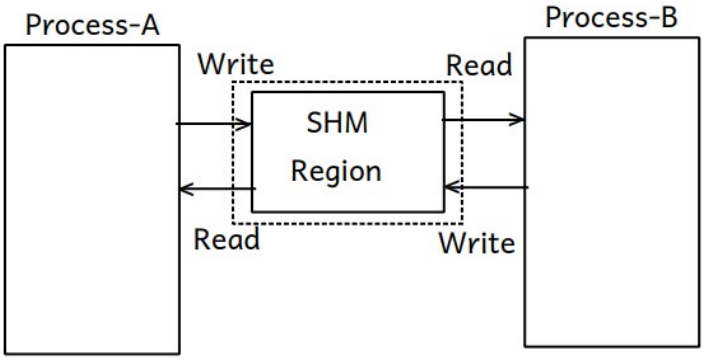
- There are two **techniques** by which IPC can be done/there are two **IPC Models:**

1. Shared MeMemory Model: under this technique, processes can communicates with each other by means of reading and writing data into the **shared memory region** (i.e. it is a region/portion of the main memory) which is provided by an OS temporarily on request of processes want to communicates.

2. Message Passing Model: under this technique, processes can communicates with each other by means of sending messages.

- Any process cannot directly send message to any other process.

- Shared Memory Model is faster than Message Passing Model.



SHARED MEMORY MODEL

Inter Process Communication

2. Message Passing Model: there are further different IPC techniques under message passing model.

i. Pipe:

- By using Pipe mechanism **one process can send message to another process**, vice-versa is not possible and hence it is a **unidirectional communication technique.**

- In this IPC mechanism, from one end i.e. from **write end** one process can writes data into the pipe, wheras from another end i.e. from **read end**, another process can read data from it, and communication takes place.

- There are two types of pipes:

1. unnamed pipe: in this type of pipe mechanism, only **related processes** can communicates by using **pipe (|) command.**

2. named pipe: in this type of pipe mechanism, **related as well as non-related processes** can communicates by using **pipe() system call.**

- By using Pipe o**nly processes which are running in the same system can communicates**,

Inter Process Communication

ii. Message Queue:

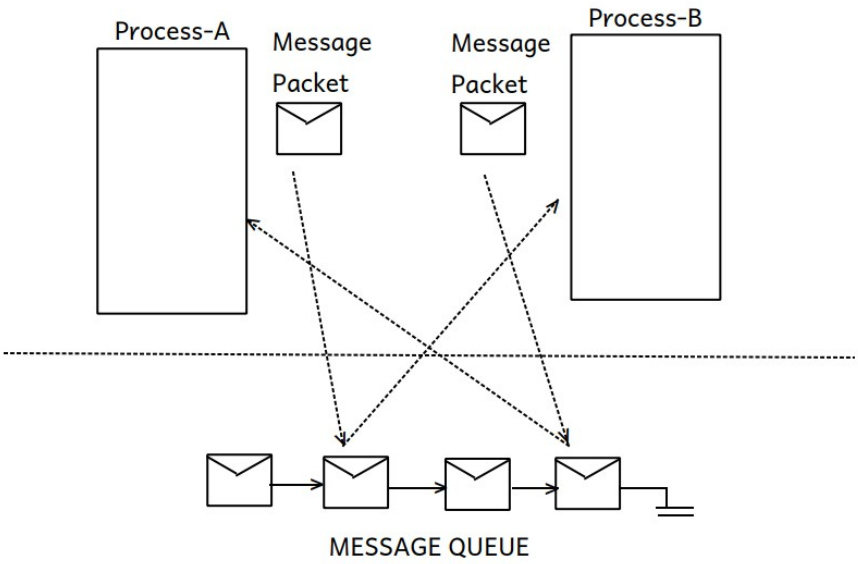
- By using message queue technique, processes can communicates by means of sending as well as recieving **message packets** to each other via **message queue** provided by the kernel, and hence it is a **bidirectional communication.**

- **Message Packet: Message Header(Information about the message) + Actual Message.**

- Internally an OS maintains message queue in which message packets sent by one process are submitted and can be sent to reciever process and vice-versa.

- By using message queue technique, only processes which are running in the same system can communicates.

Inter Process Communication



Inter Process Communication

iii. Signals:

- Processes communicates by means of sending signals as well.
- One process can send signal to another process through an OS.
- An OS sends signal to any process but any other process cannot sends signal to an OS.

Example: When we shutdown the system, an OS sends **SIGTERM** signal to all processes, due to which processes gets **terminated normally**, but few processes can handle SIGTERM i.e. even after recieving this signal from an OS they continues execution, to such processes an OS sends **SIGKILL** signal due to which processes gets **terminated forcefully.**

e.g. SIGSTOP, SIGCONT, SIGSEGV etc...

- By using signal ipc technique, only processes which are running in the same system can communicates.

 # Process Coordination / Process Synchronization

Why Process Co-ordination/Synchronization?

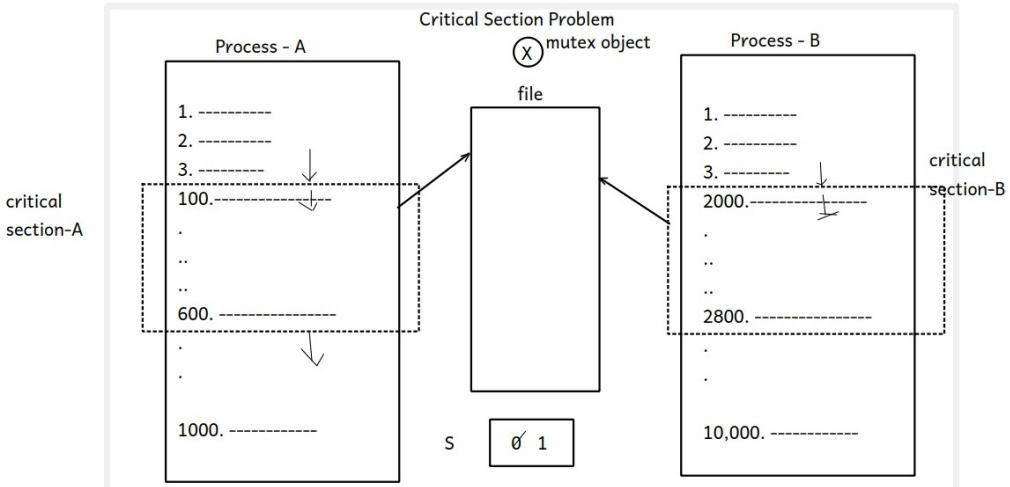
- If concurrently executing co-operative processes are accessing common resources, then conflictions may takes place, which may results into the problem of **data incosistency**, and hence to avoid this problem co-ordination / synchronization between these processes is required.

Race Condition: if two or more processes are trying to access same resource at a time, race condition may occures, and **data inconsistency** problem may takes place due to race condition.

- Race condition can be avoided by an OS by

1. deciding order of allocation of resource for processes, and
2. whichever changes did by the last accessed process onto the resource remains final changes.

Critical Section Problem:



"data incosistency" problem occure in above case only when both the sections of process A & B are runniing at a same time, and hence these sections are reffered as critical section, and hence data inconsistency problem may occure when two or more processes are running in their critical sections at a same time, and this problem is also reffered as "critical section problem".

Synchronization Tools:

1. Semaphore: there are two types of semaphore

i. Binary semaphore: can be used when at a time resource can be acquired by only one process.

- It is an integer variable having either value is 0 or 1.

ii. Counting / Classic semaphore: can be used when at a time resource can be acquired by more than one processe

2. Mutex Object: can be used when at a time resource can be acquired by only one process.

- Mutex object has two states: **locked** & **unlocked**, and at a time it can be only in a one state either locked or unlocked.

- Semaphore uses **signaling mechanism**, whereas mutex object uses **locking and unlocking mechanism.**

Deadlock:

- There are four **neccessary and sufficient conditions to occure deadlock / characteristics of deadlock:**

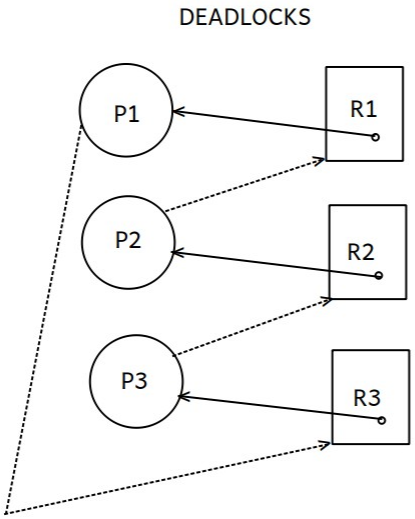
1. Mutual Exclusion: at a time resource can be acquired by only one process.

2. No Preemption: control of the resource cannot be taken away forcefully from any process.

3. Hold & Wait: every process is holding one resource and waiting for the resource which is held by another process.

4. Circular Wait: if process P1 is holding resource and waiting for the resource held by another process P2, and process P2 is also holding one resource and waiting for the resource held by process P1.

Deadlock: Resource Allocation Graph



- Three deadlock handling methods are there:

1. Deadlock Prevention: deadlock can be prevented by discarding any one condition out of four neccessary and sufficient conditions.

2. Deadlock Detection & Avoidance: before allocating resources for processes all input can be given to deadlock detection algorithm in advanced and if there are chances to occure deadlock then it can be avoided by doing neccessary changes.

- There are two deadlock detection & avoidance algorithms:

1. Resource Allocation Graph Algorithm

2. Banker's Algorithm

3. Deadlock Recovery:

- System can be recovered from the deadlock by two ways:

1. Process termination: in this method randomly any one process out of processes causes deadlock gets selected and **terminated forcefully** to recover system from deadlock.

- Process which gets terminated forcefully in this method is reffered as a **victim process.**

2. Resource preemption: in this method control of the resource taken away forcefully from a process to recover system from deadlock.

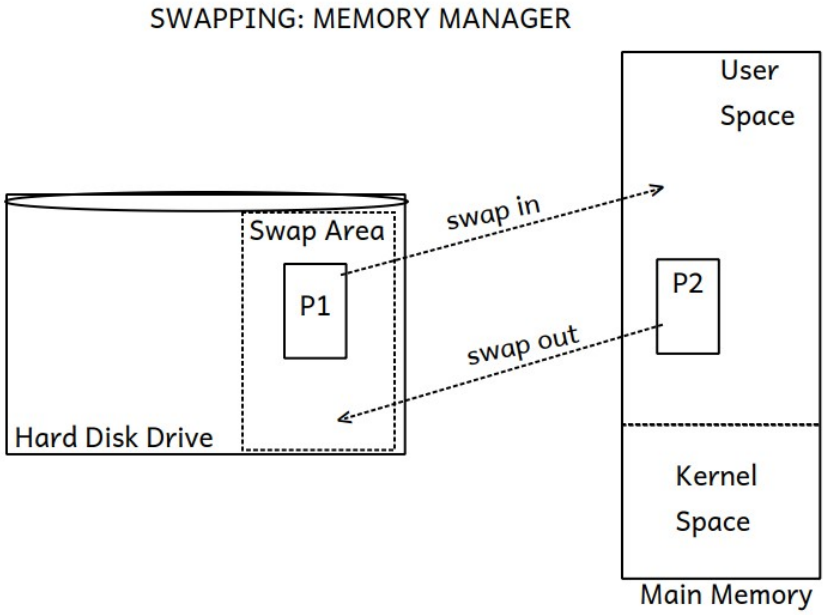
 # Memory Management

Q. Why there is a need of memory (main memory)management?

- As main memory is must for an execution of any program and it is a limited memory, hence an OS manages main memory.

- To achieve maximum CPU utilization, an OS must support multi-tasking, and to support multi-tasking multiple porcesses must be submitted into the system at a time i.e. it must support multi-programming, but as main memory is limited to support multi-programming an OS has to do memory management to complete an execution of all submitted processes.

- Memory space of one process should gets protected from another process.



 # Swapping:

- Swap area: it is a portion of the **hard disk drive** (keep reserved while installation of an OS) can be used by an OS as an extension of the main memory in which inactive running programs can be kept temporarily and as per request processes can be swapped in and swapped out between swap area and the main memory.

- In Linux swap area can be maintained in the form of **swap partition,** whereas in Windows swap area can be maintained in the form of **swap files.**

- Conventionally size of the swap area should be doubles the size of the main memory, i.e. if the size of main memory is 2 GB then size of swap area should be 4 GB, if the size of main memory is 4 GB then size of swap area should be 8 GB and so on.

Operating System Concepts

 # Swapping:

- Swapping done by the system program of an OS named as **Memory Manager**, it swap ins active running programs into the main memory from swap area and swap outs inactive running programs from the main memory and keep them temporarily into the swap area.

- there are two variants of **swapping: swap in & swap out.**

 # Swapping:

- Swapping done by the program of an OS named as **Memory Manager**, it swapin active programs into the main memory from swap area and swapout inactive running programs from the main memory and keep them temporarily into the swap area.

- there are two variants of swapping: swapin & swapout.

- Addresses generated by compiler (i.e. compiler + linker) are reffered as **logical addresses.**

- Addresses which can be see by the process when it is in the main memory reffered as **physical addresses.**

- MMU (Memory Management Unit): wich is a hardware unit **converts logical address into physical address.**

- MMU is a hardware contains **adder circuit, comparator circuit, base register and limit register.** Values of base register and limit registers gets change during context-switch, and memory space of one process gets protected from another process.

- CPU always executes program in its logical memory space.

Memory Allocation:

- When a process is requesting for the main memory, there are two methods by which memory gets allocated for any process

1. Contiguos Memory Allocation
2. Non-contiguos Memory Allocation

1. Contiguos Memory Allocation:

Under this method, process can complete its execution only if memory gets allocated for it in a contiguos manner.

- There are two methods by which memory gets allocated for process under contiguos memory allocation method.

1. Fixed Size Partitioning
2. Variable Size Partitioning

1. Fixed Size Partitioning:

- In this method, physical memory i.e. main memory (user space) is divided into fixed number of partitions and size of each partition is remains fixed.

- If any process is requesting for the memory it can be loaded into main memory in any free partition in which it can be fit.

 # Advantages:

- This method is simple to implement

 # Disadvantages:

- Internal fragmentation: if memory remains unused which is internal to the partition.

- **Degree of multi-programming** is limited to the number of partitions in the main memory.

- **Maximum size of a process is limited to max size partition** in the main memory.

- To overcome limitations/disadvantages of fixed size partitioning method, variable/dynamic size partitioning method has been designed.

2. Variable/Dynamic Size Partitioning:

- In this method, initially whole user space i.e. physical memory is considered as a single free partition, and processes gets loaded into the main memory as they request for it.

- Size of partition and number of partitions are not fixed in advanced, it gets decided dynamically.

 # Advantages:

- there are very less chances of internal fragmentation

- Degree of multi-programming is not limited/fixed

- Size of the process is not also limited, any size process may get load into the main memory.

 # Disadvantages:

- External fragmentation: due to loading and removing of processes into and from the main memory, main memory is fragmented i.e. gets devided into used partitions and free partitions.

2. Variable/Dynamic Size Partitioning:

- External fragmentation: due to loading and removing of processes into and from the main memory, main memory gets fragmented i.e. it gets divided into used partitions and free partitions.

In such case, if any new process is requesting for the memory and even if the requested size of memory is available, but due to unavaialability of the memory in a contuguos manner request of that process cannot be accepted, this problem is reffered as an **external fragmentation.**

- External fragmentation is the biggest problem under contiguos memory allocation, and hence there are two solutions on this problem:

1. Compaction

2. Non-contiguos Memory Allocation

1. Compaction: shuffling of main memory can be done in such a way that all used partitions can be shifted to one side and all free partitions can be shifted to other side and contiguos large free partition will be made available for the new processes.

- Compaction is practically not feasible as there is need to do recalculations of addresses every time.

2. Non-contiguos Memory Allocation:

- Under this method, process can complete its execution even if memory gets allocated for it in a non-contiguos manner, and it can be achieved by two memory management techniques:

1. Segmentation

2. Paging

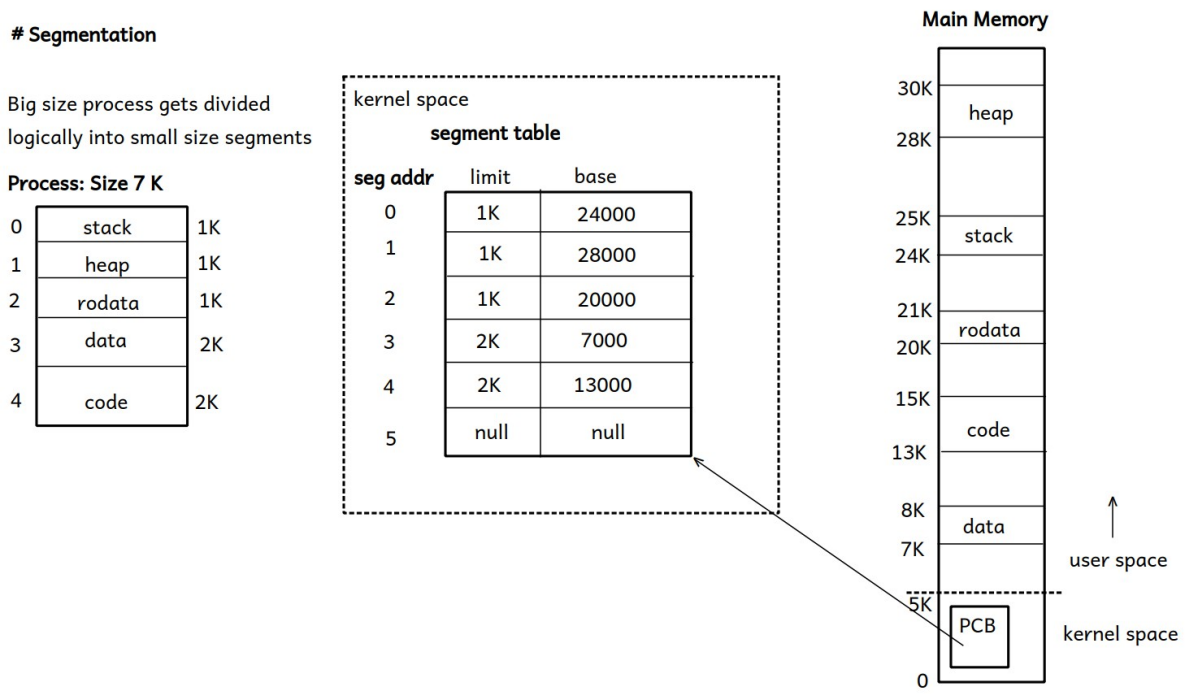
- So by using segmentation & paging techniques, process can complete its execution even after memory gets allocated for it in a **non-contiguos manner.**

1. Segmentation

- In this technique, process in its logical memory is divided into small size segments like **stack segment, heap segment, data segment, bss segment, rodata segment, code segment etc**..., and when process is requesting for memory it is not requesting memory contiguosly for whole process, memory gets allocated contiguosly only for small size segments, and segments of one process may get load into the memory randomly at any locations, i.e. for a process memory gets allocated in a **non-contiguos manner**, and then only an execution of a process can be completed.

- As segments of a one process gets loaded randomly into the main memory, and in a system thousands processes are running at a time, hence to keep track on all the segments of each process, an OS maintains **one table per process** reffered as a **segmentation table** in which information about all the segments of that process can be kept.

- Using segmentation an **external fragmentation can be reduced but cannot be completely avoided => to overcome this limitation paging technique has been designed.**



2. Paging

- In this technique, **physical memory** (i.e. user space of a main memory) is divided into fixed size of blocks reffered as **frames**, and process's **logical memory** space is divided into same size of blocks reffered as **pages**, whereas maximum size of page must be equal to size of frame, i.e. if e.g. size of frame = 4K, then maximum size of each page must be 4K, size of page may be less than 4K.

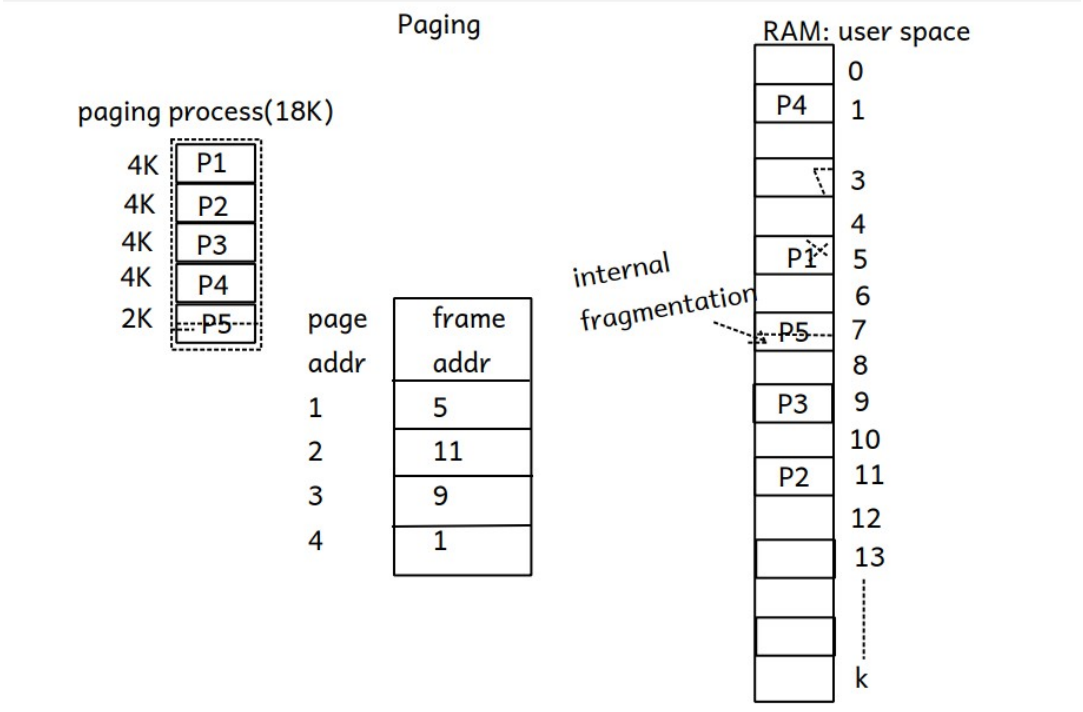
- As process is divided into pages, so when it is requesting for memory, pages of one process may gets loaded into the main memory at any free frames, and for a process memory gets allocated in a non-contiguos manner.

- As pages of a one process gets loaded randomly into the main memory, and in a system thousands processes are running at a time, so to keep track on all the pages of each process, an OS maintains one table per process reffered as a **page table** in which information about all the pages of that process can be kept.

- There is no external fragmentation in paging.

- Internal fragmentation may exists in paging when the size of page is less than size of frame.

Operating System Concepts



Virtual Memory Management:

- As we seen an OS does memory management for completing an execution of multiple submitted processes at once.

- An OS also able to complete an execution of such a process having size bigger than size of main memory itself, and to achieve this an OS manages swap area memory as well with main memory and hence it is reffered as **virtual memory management.**

- As an OS manages such a memory which is physically not a main memory and hence it is reffered as virtual memory management.

- Virtual memory management can be implemented by using **Paging + Swapping.**

- In this technique for a process to complete its execution it is not mandatory it must exists wholely in a main memory, even its part is their into the main memory then execution of such process can be completed part by part.

- Big size process is divided into pages and when a process is requesting for memory few pages gets loaded into the main memory and few pages can be kept into the swap area, and as per the request pages of that process can swappedin and swapped out between main memory and swap area.

Operating System Concepts

Virtual Memory Management:

- Demand Paging: any page of a process gets loaded into the main memory only after requesting by that process i.e. on demand and hence reffered as **demand paging**, page which is never requested never gets loaded into the main memory and hence it also called as **pure demand paging**.

- If a process is requesting for any page and if that page is not exists in the main memory, then it is referred as **page fault**.

- As the size of process may be bigger than size of main memory itself, and in a system multiple processes are running at a time, hence no. of pages are more than no. of frames, so there are quite good chances that all frames becomes full, and in this case if any process is requesting for a page which does not exists in the main memory at that time there is need to remove any one page from the main memory so that into that free frame requested page will get loaded.

- So there is need to decide which page should get removed and requested page gets replaced in that frame, to do this there are certain algorithms referred as **page replacement algorithms.**

Page Replacement Algorithms:

1. FIFO Page Replacement: page which was inserted first gets replaced by the requested page.

2. Optimal Page Replacement: page which will not get used in a near future gets replaced by the requested page

3. LRU(Least Recently Used) Page Replacement: least recently used page gets replaced by the requested page.

4. LFU(Least Frequently Used) Page Replacement: least frequetly used page gets replaced by the requested page.

5. MFU(Most Frequently Used) Page Replacement: most frequetly used page gets replaced by the requested page.

- Algorithms 1, 2 & 3 uses **stack based** approach, whereas algorithms 4 & 5 uses counting based approach.

- Conceptually Optimal Page Replacement is the most efficient page replacement algorithm, as no. of page faults in this algorithm are very less, but as its practical implementation is not feasible hence LRU is the most efficient algorithm (implementationwise).