Page No.
Date 6 8 22

Operating System →

* Resource management (Apps does not get bulky)
* Isolation & protection

* An OS is the piece of software that manages all resources of computer both hardware & software.

* PCB – Process Control Block (saves the data from unfinished process)
* Context switching → used in multitasking

* Types of OS →
  * Single process (MS-dos)
  * Batch process (Atlas)
  * multiprogramming (THE)
  * multitasking (CTSS)
  * Multi process (Windows)
  * Distributed (loosely coupled)
  * Real time

* Process → Program under execution
* Thread → light weight process (small subprocess of process)
* multithreading → Break process into different parts which are not depended on each other & execute together for fast execution. (parallelism in a process)

Multitasking	multithreading
→ Concept of more than 2 process	→ more than one thread in single process
→ Isolation & memory protection present.	→ does not present.
→ Process schedule	→ Thread process

Page No.
Date

* Components of OS →
  → User space → apps/software runs here
  → kernal → access to underlying hardware

* User space
  → Provide convinient environment to user.
    * GUI - Graphical user interface
    * CLI - Command line interface
  → interacts with kernal

* kernal →
  → Heart of OS. (interacts with hardware)
  → Function → * Process managment (creation, termination)
    * Process & thread schedule
    * Process synchronise (process communication)
  → * Memory managment
  → * file mangement (file create/delete)
    * directory managment
  → * Input/output management
    → spooling
    → Buffering
    → caching

* Types of kernal
  * Monolithic kernal (oldest) → Linux/unix, ms-dos
    → all kernal components are present in one space. (that's why fast communication)
    → kernal bulky
    → less reliable

* Micro kernal → L4 Linux, Symbian OS.
  → only memory & process managment is present
  → reliable / stable
  → performance less as compared to monolithic kernal

* How to communicate between user mode & kernal mode?

→

IPC - Interprocess communication.
① shared memory
② message passing

* Hybrid kernal → mac, windows 7 & above
→ Process, memory & I/O management present.
→ better speed than microkernal

* nano kernal
* Exo kernal

* System Calls → implemend in C language
To switch from user space to kernal space
SCI → System call interface

System call is a mechanism using which a user program can request a service from the kernal for which it does not have a permission to perform.

* Types of System Calls →
→ Process control
→ file management
→ Device management
→ information maintainance
→ Communication management

Page No.
Date

* What happens when you turn on PC.

① → Power on.
* Electric supply given

② → CPU loads BIOS or UEFI
* BIOS – Basic input output system
* UEFI – Unified extensible firmware interface
↖ upgraded version of BIOS
① CPU initialize & goes to chip bios

③ → BIOS or UEFI run tests & init hardware
① loads some setting from a memory area backed by CMOS battery.
* CMOS → Complementary metal oxide semiconductor
② BIOS program loads with setting
* POST – power on self test

④ → BIOS & UEFI hands off to boot device [bootloader]
* Bootloader → is a program which turns on/contains OS.
Bootloader is stored in →
(old system) * MBR – Master boot record. [disk – 0th index]
(new system) * EFI – Extensible firm interface. [partition in disk (C-drive)]

⑤ — Bootloader loads the full OS.
(file names) windows → bootmgr.exe
Mac → boot.efi
Linux → GRUB

Page No.
Date

* 32 bit Vs 64 bit OS.

32-bit contains - 32 bit register from 0 to 31 bit
64 bit contains - 64 bit register from 0 to 63 bit.

→ 32 bit
* 2^{32} unique address.
* Can support upto 4gb ram

→ 64 bit
* 2^{64} unique address
* 17179869184 GB ram support

* Advantages of 64 bit.
→ more unique adress (2^{64}).
→ Better resource usage (can put ram more than 4gb).
→ Performance fast.
64 bit can process 64 bit data or fetch instruction in 1 instruction cycle.
→ Compatibility → can run both 32 bits & 64 bit os
→ Better graphic performance → 8 byte graphic computation is possible

* Storage types →
→ Register → smallest unit of storage only bits are stored.
→ Cache → addition memory to store temperary memory/data.
→ main memory → RAM.
all important data is stored here.
process data is stored here.

primary memory	* Register * cache * main memory
secondary memory	* Electronic disk * Magnetic disk * Optical disk * Magnetic tapes.

→ Hard disks → all data is stored here which is (secondary memory) then processed & send data to main memory.

* Cost of storage
Register → most ~~memory~~ expensive

* Access speed →
Register — most speedy access

* Storage size —
Register - smallest memory

* Volatility →
* In primary memory all data is erased when computer is shut off.

→×←

* Process Management
→ What is process — It is programme under execution

* Programme to Process →
① → load the program & static data to memory.
static data is data which we have assigned in our written program

② → Allocate Runtime stack →
stack is part of memory used to store local variable, function arguments & return values

③ → Allocate Heap →
part of memory used for dynamic allocation

④ → IO Related tasks →

eg → In unix system when process is generating 3 types of IO descriptors are made

① I/p ② O/p ③ error

which helps in creating the process.

⑤ → OS handoff control to main() in our function.

Final step in creating function.

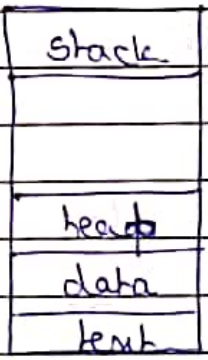
* → Architecture of Process

① Text → compiled code.

② data → Global variables & static data

③ Heap → dynamic memory location

④ stack → Static memory location



* Attributes of Process. →

PCB - Process control block

structure →

Process ID.
Program counter
process state
priority
Register
Open file. list.
open devices list.

* Types of register —

① Stack pointer register (SP)

② Base pointer register (BP)

③ Control Register (CR)

FD - file descriptor (present in open file list)

DD - Device descriptors (present in open device list)

Page No.
Date

* Process State (from generation to termination)

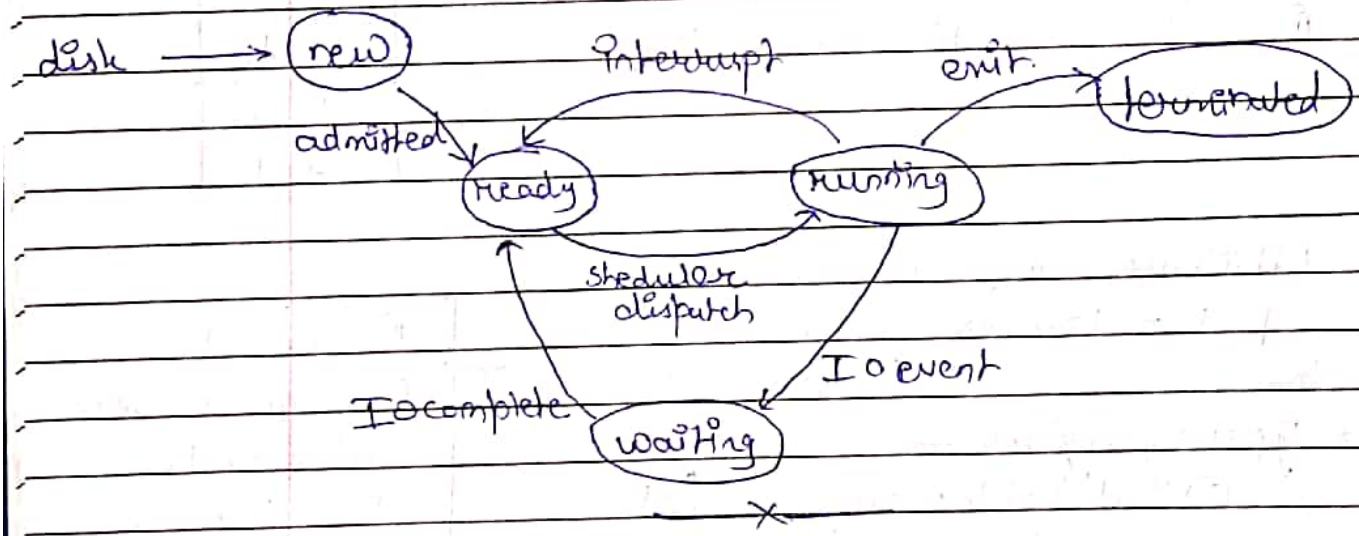
① → New → Process is being done/made

② → Ready → Process is in memory waiting for CPU
→ in ready queue

③ → Running State → CPU allocated

④ → Waiting State → waiting for I/O completing

⑤ → Terminated → Process finished



* different types of queue -

① → Job queue - process in new state

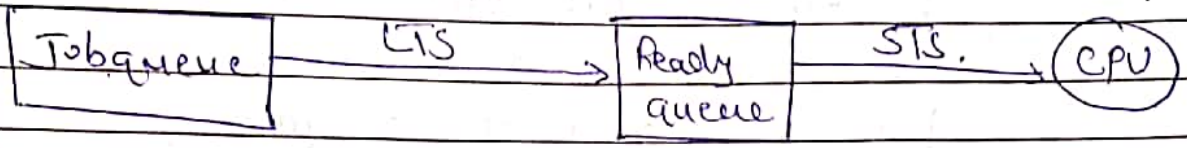
Job scheduler / long term scheduler (LTS) → from job queue to ready state. It has less frequency.

② → Ready queue → Process in ready state

CPU scheduler / short term scheduler (STS) → from ready state to running state according to priority. (dispatch to CPU from ready)
It has very high frequency

* dispatcher → The module of OS that gives control to CPU for process
* Degree of multiprogramming → how many processes can be present in ready queue at same time
* handled by job scheduler (LTS)

① → Waiting queue → Process in waiting state



* Medium term scheduler (MTS) → (performs swapping)

* swapping → when there are too many processes in ready queue some processes are sent into secondary (swapped out) storage for saving the space after the process is terminated which were already in ready queue the process which were in secondary storage comes back to ready queue (swapped in) this process is called swapping

* Context switching → It is pure overhead as per user prespective. Speed is dependent on speed of memory.

Context switching is the process in which we stop the ongoing process for some reason may be because of I/O operation or time is up for the given process & we switch to other process after saving the content of previous process.

* init is first process in linux present on 0th index PID is 1.
* To check exit status we call wait().

Page No.
Date

* Orphan Process →
when a parent process is killed before child process the child process is called as orphan process. The PPID of this type of process is 1. The PPID of first process of OS.

* Zombie Process → (Defunct process)
When a parent process is waiting for child process to exit but child process exit before the time & parent is still waiting then the child process is called as zombie process for that particular time. At this time process is exited resources are released but entry in process table is still there

* Process table entries
→ windows → 512 entries on 32 bit
256 entries on 64 bit
→ linux → 32768 entries factory default
4194304 max
→ mac → 1064 or 2088

* Reaping of Zombie process
when parent process comes & reads the exit status of zombie process

---

* Process scheduling algorithm → (CPU scheduling)
It tells which process to pick from the ready queue & give to cpu.

→ 2 types of scheduling
① * non-preemptive scheduling
② * Preemptive scheduling

* Starvation → when CPU is not getting available for certain process.

Page No.
Date

① → Non-preemptive scheduling
It will leave the process only when the process terminates or goes to wait state.
* more process starvation, less CPU utilization, less overhead

② → Preemptive scheduling.
It will leave the process when it terminates or goes to wait state but also when time quantum expires.
* CPU utilization more, more overhead, less process starvation.

* → Goals of CPU scheduling algorithm.
① maximum CPU utilization
② minimum turn around time (TAT) (TAT = CT − AT)
→ time between ready queue to exit status.
③ minimum wait time (WT = TAT − BT)
④ minimum response time.
→ time between ready queue to getting CPU to work
⑤ Maximum throughput.
→ no of process complete during per unit time.

* Arrival time (AT) → process arrived in ready queue.
* Burst time (BT) → Time required for process to execute
* Completion Time (CT) → Time taken till process is terminated

CPU

* Scheduling Algorithms.

IMP

① → FCFS (First come first serve) (Simplest)

It has convoy effect → If the process as longer BT it will have major effect on average WT of different process called convoy effect.

Convoy effect is a situation where many processes who need to use a resource for a short time are blocked by one process holding resource for long time. This cause poor resource management

② (nearly impossible to implement because it is very difficult to find actual BT of process.)

SJF (Shortest Job first) (Non-preemptive version)

* process with least BT will get CPU.
* convoy effect present.

Criteria → Arrival time + least burst time

② SJF (Preemptive version)

* convoy effect not present.

③ * Priority Scheduling

→ Assign priority to each process in ready queue

→ Criteria → highest priority time

* non preemptive	* Preemptive
* low overhead	* high overhead
* Convoy effect present	* Convoy effect present

Biggest Drawback → It has indefinate waiting or extreme starvation

In RR algo the overheads depends upon the time quantum

* Solution of indefite waiting
Ageing → Gradually increasing priority of jobs so that low priority job will get chance to do job.

④ Round Robin Algorithem (RR) (most popular)
* Lowest starvation, less starvation.
* (preemptive version of FCFS)

* Criteria = AT + Time quantum.
* designed for time sharing which is used in multitasking os.
* easy implementation.
* lot of context switching (more overheads)

* Real life scheduling Algo
① multi level queue scheduling (MLQ) (not used now)
* System process — created by os.
(foreground) * Interactive process) → user input required
(background) * batch process → no i/p or o/p.

→ System process creates SP queue who works on RR algo
→ Interactive process creates IP queue who works on RR algo.
→ batch process creates BP queue who works on F-CFS algo.

* Priority given to SP first IP second & BP third.
* Average waiting time is more.
* Convoy effect present [Indefinate waiting possible]
* Starvation will happen.
* Interqueue movement is not present.

(MLFQ)

⑥ → Multilevel feedback queue (used nowadays)
* multiple subqueue
* interqueue movement is allowed.
* Seperate process (process which take light BT will be shifted to lower queue. Process which take more I/O will be moved to upper queue
based on →
* Ageing is used to increase priority of lower queue processes.
* flexible. (configurable).

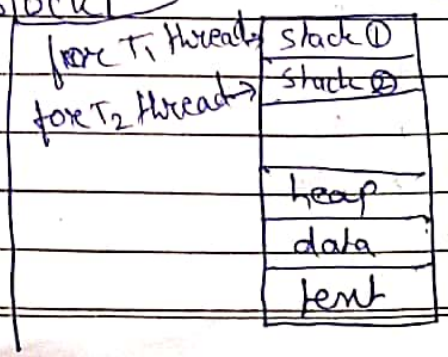
Design of MLFQ →
① → no. of queues.
② → scheduling algo in each queue
③ → method to upgrade a process to a higher queue
④ → Demote a process
⑤ → which queue to push the process

→

* Concurrency → executing multiple instruction at same time.
can be acheived by breaking a process into multiple threads.

* Thread →
→ TCB – Thread control block
similar to PCB.

how memory looks when multithreading is occuring



* Benifit of multithreading →
① Responsiveness increase (interactive)
② Resource sharing efficient because address space is same, fast content switching.
③ Economy ~~~~ increase.
④ Thread better utilizes multicore CPU.

* Critical section → section of code where multiple thread comes together & works.

* Race condition → If some data loss happens because (critical section problem) of some content switching then it is called race condition

—×—

* Solution to Race condition.

① → In C++ it is possible to make atomic variable so that the process will not break into different threads in the middle of that particular integer process.

② → Mutual Exclution [using locks (mutex)]
In all languages there is function to create locks (mutex) which does not let 2 processes to simultaneously work on critical section.

~~② → Peterson Solution~~

→ ③ Single flag
mutual exclution is done but progress is not happening all the time

* only used for 2 Threads

④ → Peterson Solution →
flag[2] → array of 2 indicate if thread is ready to enter the C.S. If flag[i] is true implies that thread is ready.
turn → Turns whose turn is to enter the C.S.

* both mutual exclution & progress is present.

Page No.
Date

Race condition ?

Q → Sol' of critical section should have 3 condition
→ (1) Mutual exclusion
(2) Progress should happen (any thread can go at any time)
(3) Bonded waiting (should not happen indefinate waiting)

* Disadvantages of locks → (mutex)
(1) Contention → If some thread enter C.S. & that thread gets dead for some reason then other threads will go to infinite wait.
(2) Dead locks →
(3) Debugging issue
(4) Starvation (Busy waiting) → high priority threads may starve because of low priority thread.

(5) → Using Conditional Variable →

(6) → use semaphores
* It makes the number of resources count (count > 0) when all resources are full it will not allow other threads to enter C.S.
* whatever we keep the value of semaphore - only that no. of threads will work on C.S at a time
* If semaphore value is 1 then it is called binary semaphore.
* If semaphore > 1 it is called counting semaphore.

IPC

* Classical Problems of Synchronization.

* ① Producer - Consumer Problem. (Bounded buffer problem)

Here there are 2 threads Producer & Consumer both threads work. Producer putter the data into buffer (C.S) & consumer takes that data from the buffer (C.S.).

* Problems → ① Only one thread should work at a time (basically race condition should not occure) so that we do not lose any data. (synchronize between producer & consumer).

② Producer must not insert when buffer is full

③ Consumer must not remove data when buffer is ~~not~~ full.

④ Bounde buffer problem

* Solution →

① mutex → Binary semaphore is used to aquire lock on buffer.

② empty variable → initial value is 'n' which is no. of slots in buffer.
It is used to track empty slots in buffer

③ full variable → Tracks filled slots.
Initial value is '0'.

IPC

* ② Read - write Problem → (CS)

here one data base is present. the writer thread want to write onto the database & reader thread wants to read data from database.

* Problems

① If reader > 1 (not issue).

③ If writer > 1 or if 1 writer thread & some other thread (Reader / writer)
* Race condition may occure
* data inconsistancy will happen
So basically when 1 writer is working on C.S. then no other thread should work on C.S.

* Solution — (semaphore Sol(1))
① use Mutex → * use to ensure mutual exclution when read count variable (RC) is updated.
* not two threads would modify read count at same time.

② use WRT variable. → (binary semaphore)
* Common for both read write threads

③ Read count (RC) → (integer)
Tracks how many readers are reading into C.S.

* ③ → Dining Philosopher Problem →
Problem * Here 5 people sitting on diving table they need 2 spoons to eat but only 5 are available so we have to synchronize the system in such a way that no deadlock or any breakage happens

* Solution → ① use semaphores
make each spoon - semaphores [binary semphore]
rule ② use 4 people instead of 5.
rule ③ Allow people only to pick spoon when 2 spoons are available by making those commands as Critical section.

rule (4) odd person will pick left spoon first & even will pick right spoon first.

---

* **Deadlock** → Two or more process are waiting for some resource but that resource will never be available because it is busy.
→ **Resources** → memory space, CPU instances, files, locks, I/O devices.

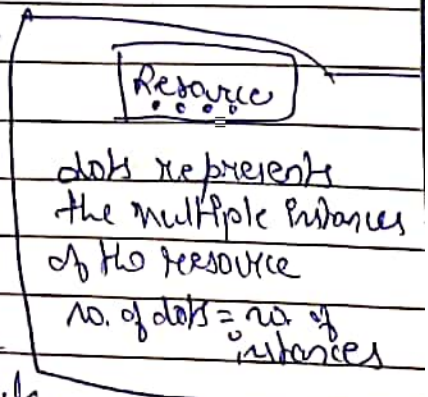
* **How process/thread utilize resource.**
step (1) → Request (lock)
step (2) → use
step (3) → Release (unlock)

* **Necessary Condition for deadlock** (should simultaneously happen)

(1) Mutual exclusion
(2) Hold & wait. (process is holding 1 resource & waiting for another resource)
(3) No Preemption (no process gets resource when one process is using the resource.)
(4) Circular wait. (similar to hold & wait condition)

* **Resource Allocation Graph (RAG)** →
(1) Vertex → (1) Process Vertex
(2) Resource vertex
(2) edges → (1) Assign
(2) Request

By definition → (1) If in RAG no cycle – no deadlock
(2) If cycle present – may be deadlock



* Methods to Handle Deadlock →

(1) Prevent or avoid Deadlock
(2) allow system to go in deadlock detect if true then recover
(3) Ostrich Algorithm (Deadlock ignore)
(put everything on programmer OS will be out)

(1) * Deadlock Prevention ~~or avoid technique~~

(1) Minimum use of mutual execution
→ only use locks on non shareable resources
make them critical section

(2) Hold & wait → (should not happen)
protocol (A) process should get all the resource before execution.
Protocol (B) Allow process to request resource only when it has none. It can request any any additional resource after it must have released all the resources that is currently allocated

(3) Avoid no Preemption →

(4) Avoid Circular wait →

* Avoid deadlock →

(1) Current state (known) → (1) no. of processes
(2) max need of resources of each process with currently allocated amount of resources
(3) max amount of each resource.

* Safe state → A state is safe if a system can allocate resources to each process (upto its max) in some order & still avoid deadlock

* Bankers Algorithm to avoid deadlock →

To find if the given state is safe state or unsafe state.

eg →

Process	allocated	max need	available (total − total already)	Remaining need (max − allocated)
	A B C	A B C	A B C	A B C
P_1	0 1 0	7 5 3	3 3 2	7 4 3
P_2	2 0 0	3 2 2	5 3 2	1 2 2
P_3	3 0 2	9 0 2	7 4 3	6 0 0
P_4	2 1 1	4 2 2	7 4 5	2 1 1
P_5	0 0 2	5 3 3	7 5 5	5 3 1

Total already 7 2 5

Sequence → $P_2 \rightarrow P_4 \rightarrow P_5 \rightarrow P_1 \rightarrow P_3$

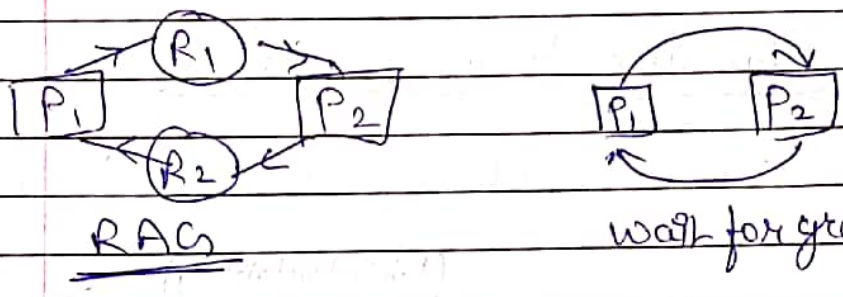
total resource
A = 10
B = 5
C = 7.

* Deadlock Detection — run after sometime interval

system → algorithm → Deadlock → no
↓ yes
Recovery

method (1) → Single instance of each resource
(used only for single instance)
wait for graph is used

after using wait for graph creates cycle is formed then deadlock is occured



RAG wait for graph

method (2) → multiple instances.
* used Bankers algorithm
if safe sequence present DL not present
if not present Dead lock present

* Deadlock Recovery →

method (1) → Process termination
* → abort all deadlock cycle processes
* → Remove one process at a time until deadlock is eliminated.

MMU – Translates logical address into physical address space

Page No.
Date

method ② Resource Preemption —
preempt some resources from processes & give resources to other processes until deadlock cycle is broken. [according to the priority of process]

* MMU → Memory mangement unit
** Memory Management →
logical address → address which user uses.
virtual address space (VAS) → address in actual memory
Base – Physical address in memory
offset → Range address of a process

* Allocation methods in Physical space (memory)
→ ① Contiguous memory allocation (not used nowadays)
allocating memory space continuous addresses
* each process is contained in a single continuous block

method ① → Fixed partitioning
The main memory is divided into partition of equal or different sizes

method ② → Dynamic Partitioning
Partition size is not declared initially it is declared at the time of process loading

→ ② Non-contiguous memory allocation
→ small partition of memory which is not useful.
* External fragmentation → Because of contiguous memory allocation some partition is already present in memory, & even free memory present we can not assign a process which need less memory is called as external fragmentation

* Defragmentation / Compaction →
assigning all the free memory partitions in a single contiguous memory space.
efficiency decreases of CPU-time consuming process

* Algorithms to satisfy a request of an n size from a list of free holes.

(1) First fit (fast, simple, easy to implement)
we allocate the first hole which is fit enough to contain the process

(2) Next fit (enhancement of first fit)
same as first fit but after one process we do not iterate the free list from start we iterate from the ~~past~~ previous point where the free list was saved

(3) Best fit (least internal fragmentation, slow, major external fragmentation)
~~it will find the best memory part which can contain the process~~ find smallest hole which is big enough to contain the process

(4) Worst fit (lesser external fragmentation, slow)
we will allocate the largest hole in free list

* Non-Contiguous memory allocation

* Paging (no external fragmentation)

Page size = frame size (fixed equal)

page → we break process into small partitions in order to allocate it into ram in non contiguous way.

frame size → Making small equal partitions in the ram of fixed size.

Process of dividing process into pages & ram into small frames is called as paging

Q * How MMU converts logical memory address into physical?

Ans → * Paging process is taken into work with the help of table data structure which contains information about which page is allocated to which frame.

* every process has its own different page table

* Page table Base Register (PTBR)
used by OS to keep track of different page table of different processes used for context switching

Q * Why paging is slow & how do we make it fast
Since there are too many memory references to access desired memory location in physical memory to make it fast we use TLB.

Translation look-aside buffer (TLB). (one type of cache register)

* It is a hardware support whose cost is even more than ram access time is very less than main memory

ASID - Address space identifier → used to differentiate between different process table entry.

* Problems in Paging → * less efficient (in a case of cpu breaks one function into multiple pages)
* more overhead

To solve this problem we use segmentation.

* Segmentation →

advantages → * no internal fragmentation
* one segment has a contiguous allocation hence efficiency is more
* The size of segment table is less than page table
* result is more efficient because compiler keeps the same type of function in one segment.

disadvantage → * External fragmentation present
* The different size of segments is not good at the time of swapping

VVIMP

* Virtual memory Management →

Virtual Memory (swap-space)

It is a part of secondary memory which we are treating as a main memory. It provides an illusion to user of having very big main memory.

In this situation only needed pages is stored in RAM & the remaining pages is stored in virtual memory which is also in ready to use state. Because of which we can save lot of RAM memory & can run multiple programs or single program of very big size more than RAM without interruption.

→ Popular method of Virtual memory management.

* Demand Paging

It is the process in which the page which is least needed get stored in secondary memory. & page is copied to main memory when demand is made or when page fault occures

Rather than swapping the complete process into the main memory we use lazy swapper. which never swaps page into memory unless it is needed.

Since we are not swapping the complete process in/out of the memory instead of swapper we use pager.

So process is done by lazy pager.

* Pure Demand Paging

It is a process when we start process with no page in main memory & storing all pages in swapspace. & taking pages into main memory as they are needed. Do not bring page into memory unless it is required

Since at the start no page is stored in main memory it is little slow process.

* Locality of Reference →

Instead of pure demand paging we use locality of Reference. to bring out reasonable performance from demand paging.

* Virtual memory →

* Advantage → Degree of multiprogramming increase, big programs can run.
* disadvantage → It is slower as swapping takes time.

Thrashing may occure.

Page No.
Date

* Page Replacement Algorithms →

Page fault → when page is not present in main memory at the time it is needed then page fault occures

Page fault service time is a price for page fault.

① FIFO → First in first out
- * easy to implement
- * Performance is not always good
- * Belady's anomaly is present

as no. of frames increase page fault should decrease but sometimes even if frames increase page fault also increase this condition is known as belady's anomaly.

Best but impossible to implement

② Optimal Page Replacement
- * Impossible to implement
- * Idle algorithm

→ here we find a page which is never going to reference in the future & we replace that page.

→ If no such page is present we find the page which is referenced farthest in future. & replace it.

IMP ③ Least Recently used (LRU) →

→ here we find the ~~oldest~~ oldest page which is present in main memory & replace it with needed page.

→ here we relies on past information & assume that oldest page will not be used again so we replace it.

Page No.
Date

* Implement LRU.

① use of Counter. →

→ use counter for each page table entry.

→ Replace page with smallest ~~time~~ counter value.

But here counter can ~~be~~ overflow.

② Stack based. (use doubly linked list)

→ we keep track of page number.

→ whenever page is refered ~~it~~ it is removed & put on top of stack.

→ Because of this the lease recently used will get on the bottom which will be replaced.

④ → Counting based page algo. → keep counter each page

① least frequently used (LFU)

Page with smallest count is removed/replaced

② most frequently used (MFU)

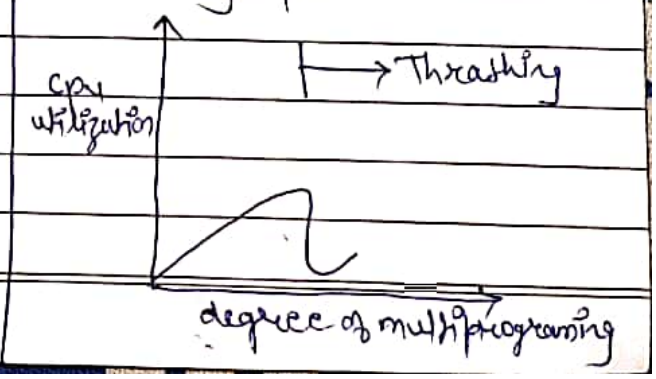
page with highest count is removed/replaced.

* → Thrashing →

replacing pages again & again that it must bring back immediately. ~~It is called thrashing~~ this high paging activity is called as Thrashing.

→ A system is thrashing when it spends more time servicing the page fault than executing process.

At the time of thrashing cpu is not doing any useful work it is spending more time in page swapping.



* Avoid Thrashing →

① Working Set Model

→ It works on concept of locality model.

→ process will only fault when it moves to some new locality. But if the allocated frames are lesser than the size of current locality then the process is bound to thrash.

② Page fault Frequency →

→ here we try to control page fault rate.

→ when page fault is high then we increase the no. of frames & vice-versa.

—X—