UNIT- III

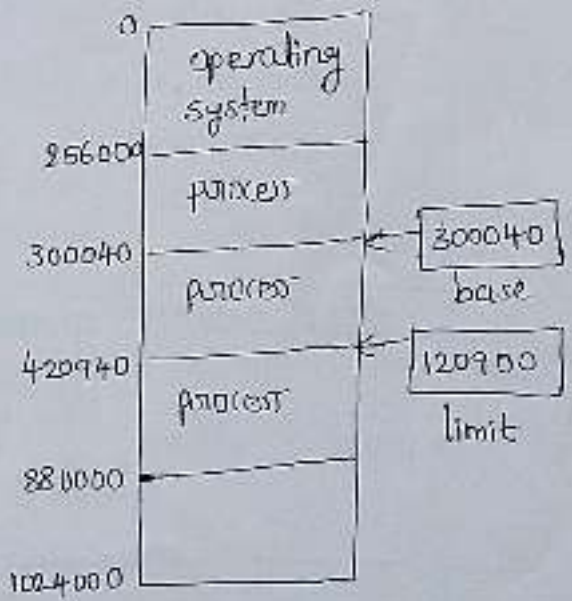
Memory Management and virtual memory - memory management strategies Background, swapping, contiguous memory allocation, segmentation, paging structure of page table, IA-32 segmentation, IA-32 paging.
Virtual memory management background, Demand paging, copy-on-write Page Replacement, page Replacement algorithms, Allocation of frames, Thrashing, virtual memory in windows.

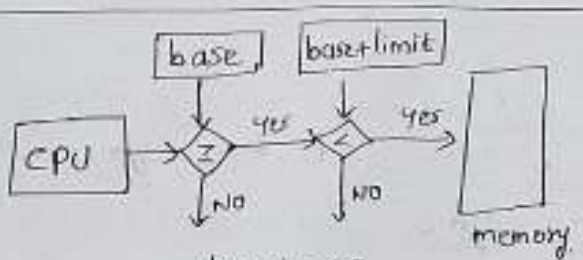
Introduction & Background:

→ Memory consists of a large array of words or bytes, each word has its own address.
→ CPU fetches the instruction from memory and execute these instructions

Basic Hardware:

→ Main memory and registers are only storage, CPU can access directly
→ Register access in one CPU clock, but main memory can take many cycles
→ cache sists between main memory and CPU registers.
→ A pair of base and limit registers define the logical address space.
→ CPU hardware compares every address generated in user mode with the registers.
→ A program executing in user mode attempt to access OS memory or other user's memory results in a trap to the OS, which treats the attempt as a fatal error.
→ This prevents a user program from accidentally modifying the code or data structure of either the OS or other users.





trap to OS
monitor - addressing error.

Address Binding:

→ A user program will go through several steps, before being executed.

→ address binding of instructions and data to memory addresses can happen at three different stages.

⇒ Compile time: if memory location known a priori, absolute code can be generated, must recompile code if starting location changes eg: DOS programs.

⇒ Load time: must generate relocatable code if memory location is not known at compile time.

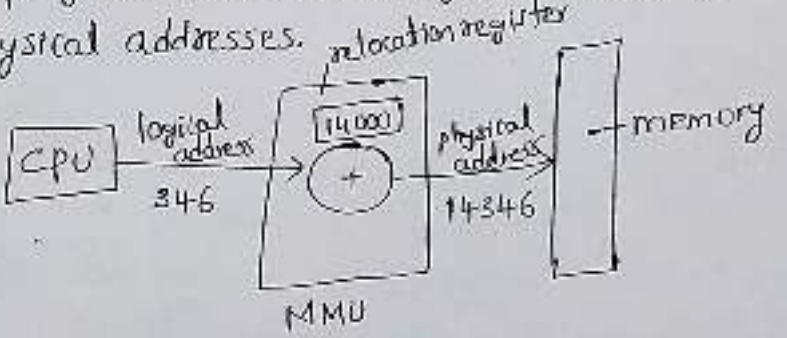
⇒ Execution time: Binding delayed until run time, if the process can be moved during its execution from one memory segment to another.

Logical versus physical address space

Logical address: generated by the CPU, also refered as virtual address.

physical address: address seen by the memory unit i.e, loaded to memory address register.

→ logical and physical addresses are the same in compile-time and load-time address-binding. logical and physical addresses differ in execution time address binding schema.
→ The run-time mapping from virtual to physical addresses is done by a hardware device called the memory management unit (mmu)
→ In mmu, the value in the relocation register is added to every address generated by a user process at the time it is sent to memory.
→ The user program deals with logical addresses, it never sees the real physical addresses.



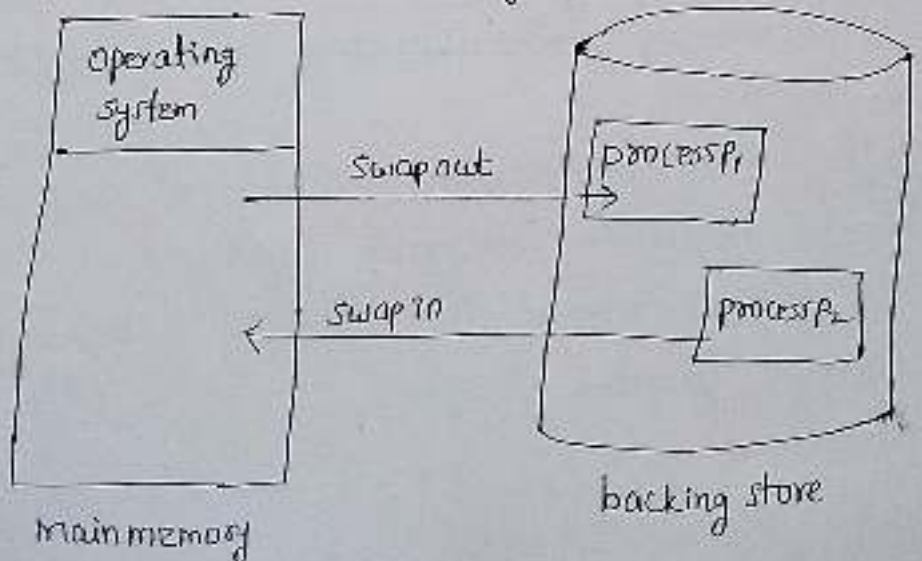
Dynamic loading:
→ All routines are kept on disk in a relocatable load formed and main program is loaded into memory and is executed.
→ Routine is not loaded until it is called
→ The relocatable linking loader is called to load the desired routine.
→ Better memory space utilization since unused routine is never loaded.
→ it is useful when large amounts of code are needed to handle frequently occurring cases.
→ No special support from the OS is required implemented through program design.

Dynamic Linking and Shared Libraries:

→ Linking postponed until execution time
→ Small piece of code, stub used to locate the appropriate memory resident library routine.
→ Stub replaces itself with the address of the routine, and executes the routine.
→ Dynamic linking is particularly useful for libraries. this system also known as Shared libraries.

Swapping:

→ A process needs to be in memory for execution but sometimes there is not enough main memory to hold all the currently active processes in a timesharing system.
→ A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution.
→ Similar to RR CPU scheduling algorithm, when a TQ expires the memory manager will swap out that process to swap another process into the memory space that has been freed.



→ backing store: it is a disk to store data that is present in memory

→ swapping can be done by priority based scheduling algorithm lower priority process is swapped out so higher priority process can be loaded and executed.

→ The swapped out process will be swapped back into the same memory space it occupied previously due to the restriction by the method of address binding.

→ The dispatcher swaps out a process in memory if there is no free memory region and swaps in the desired process from a ready queue.

→ The major part of swap time is transfer time. total transfer time is directly proportional to the amount of memory swapped.

Eg: user process is 100MB

Transfer rate of 50 MB per sec

Transfer rate = 100/50 = 2 sec [2 sec = 2*1000 m.c = 2000 ms]

swap time = transfer time + seek time (latency time)

seek time = 8 sec

swap time = 2000 + 8

= 2008 millisec

Total swap time = swapout + swap in

= 2008 + 2008

= 4016 millisec

Contiguous memory Allocation:

→ The memory is usually divided into two partitions: one for the resident operating system and one for the user processes.

→ contiguous memory allocation is a memory allocation technique

→ it allows to store the process only in a contiguous fashion. thus entire process has to be stored as a single entity at one place inside the memory. it is of two types

contiguous memory allocation Technique

- Static partitioning (or) fixed size partitioning
- Dynamic partitioning (or) variable size partitioning

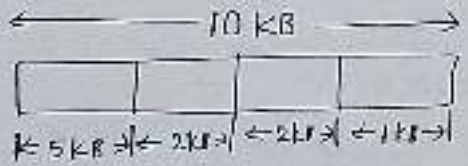
Static partitioning:

→ In static partitioning, main memory is pre divided into fixed size partitions.

→ The size of each partition is fixed and can not be changed

→ each partition is allowed to store only one process

Eg: 10 kB memory divided into fixed size partitions



→ These partitions are allocated to the processes as they arrive. The partition allocated to the arrived process depends on the algorithm followed.

Algorithm for partition allocation
- First fit Algorithm
- Best fit
- Worst fit

First fit:

→ It starts the searching the partitions serially from the starting.
→ When an empty partition that is big enough to store the process is found, it is allocated to the process.
→ obviously, the partition size has to be greater than or at least equal to the process size.

Best fit:

→ It first scans all the empty partitions.
→ next it allocate the smallest size partition to the process.
→ Best fit works best because space left after the allocation inside the partition is of very small size. internal fragmentation also least and search time is more

Worst fit:

→ it first scans all the empty partitions.
→ next it allocates the largest size partition to the process.
→ worst fit works worst, because space left after the allocation inside the partition is of very large size thus internal fragmentation is maximum.

Translating logical address into physical address:

→ cpu always generates a logical address. a physical address is needed to access the main memory.
→ The translation scheme uses two registers
  - Relocation Register
  - Limit Register

→ Relocation Register stores the base address or starting address of the process in the main memory.

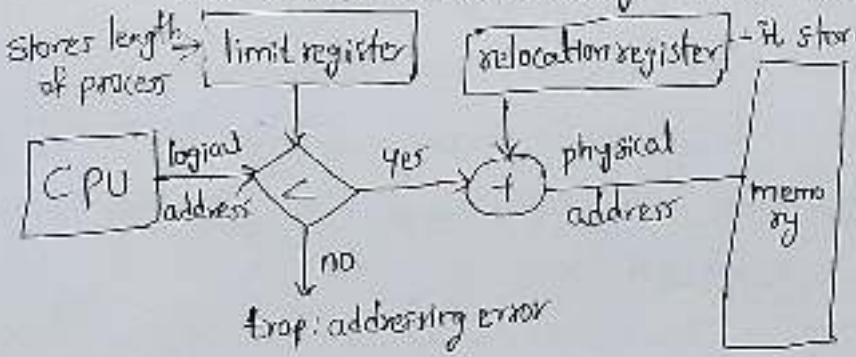
→ Limit register stores the size of or length of the process.

→ case1: [generated address >= Limit]

If address is found to be greater than or equal to the limit, a trap is generated.

→ case 2: [generated address < Limit]

The address must always lie in the range [0, limit -1]



Disadvantages of static partition:

→ It suffers from both internal fragmentation and external fragmentation.

→ It utilizes memory inefficiently.

→ There is a limitation on the size of process since process with size greater than the size of largest partition can't be stored and executed.

Fragmentation:

It may be of two types

1. internal fragmentation
2. External fragmentation

Internal fragmentation:

→ It occurs when the space is left inside the partition after allocating the partition to a process.
→ This space is called as internally fragmented space.
→ and this space cant allocated to any other process
→ This is because only static partitioning allows to store only one process in each partition.
→

External fragmentation:

→ it occurs when the total amount of empty space required to store the process is available in the main memory.
→ But because the space is not contiguous so the process can not be stored.

Segmentation: [non-contiguous Memory allocation]

→ Segmentation is a memory management technique in which, the memory is divided into the variable size parts.
→ Each part is known as segment which can be allocated to a process.
→ The details about each segment are stored in a table called as segment table. Segment table is stored in one of the segments
→ and segment table contains following information.

Base: it is base address of the segment
Limit: it is the length of the segment

→ it supports user view of memory. a logical address space is a collection of segments.

10

Translation of Logical address into physical address by segment table:

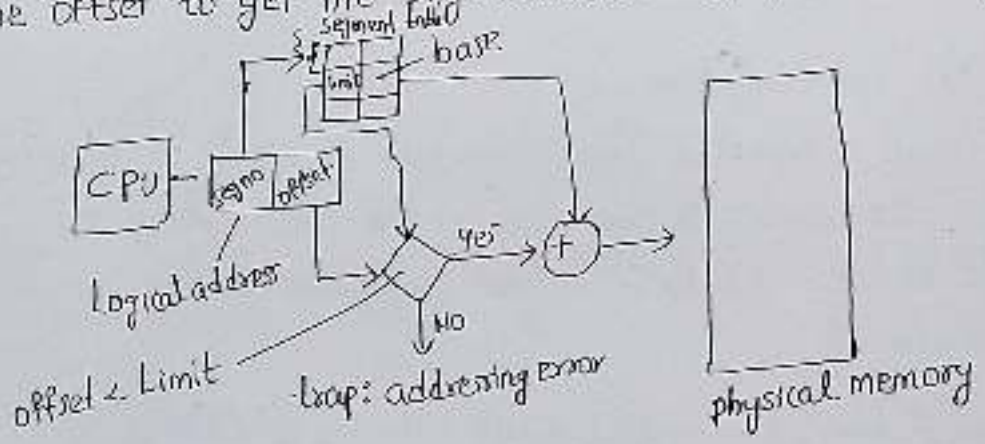
→ CPU generates a logical address which contains two parts

< segment-number, offset >

→ The segment number is mapped to the segment table. the limit of segment compared with the offset

→ If the offset is less than the limit then the address is valid, otherwise it throws an error as the address is invalid

→ In case of valid address, the base address of the segment is added to the offset to get the physical address of main memory.



Eg:

seg.no	Base	length
1	2300	14
2	90	100
3	1327	580

which of the following logical address will produce trap addressing error.

1. 1, 11 [segment no, offset]
2. 2, 100
3. 3, 425

→ segment offset must always lie in the range [0, limit-1]

→ for option 1: offset = 11, segment no = 1

The segment must lie [0, 14-1] = [0, 13]

so offset address lies between 0 to 13, no trap will be produc

physical address = 2300 + 11 [Base + offset]

= 2311

→ for option 2: [0, 99], but offset address is 100. so trap will be occur.

Paging:

→ Paging is a non-contiguous memory allocation technique.

→ To avoid external fragmentation, it allows to store parts of a single process in a non-contiguous fashion. thus, different parts of the same process can be stored at different places in the main memory.

→ Paging is a fixed size partitioning scheme. in paging, secondary memory and main memory are divided into equal fixed size partitions.

→ The partitions of secondary memory or logical memory are called as pages. the partitions of main memory or physical memory are called as frames.

→ Each process is divided into parts where size of each part is same as page size.

→ The page size is defined by the hardware. the size of the page is power of '2', varying between 512 byter and 16MB per page.

→ CPU generates a logical address consisting of two parts

1. Page Number
2. page offset

→ Page Number specifies the specific page of the process from which CPU wants to read the data.

→ Page offset specifies the specific word on the page that CPU wants to read.

Note: Logical address generated by the CPU is represented in bits

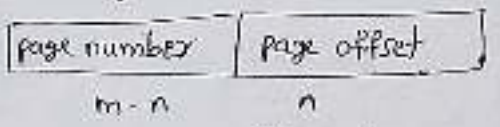
(LAS) Logical address space generated by a program is represented in words or bytes.

(PA) physical address available in main memory is represented in bits

(PAS) physical address space: represented in words or bytes. the set of all physical addresses corresponing to the logical addresses.

→ The size of the logical address space is 2^m, and page size is 2^n, then the high-order bits m-n represent the page number, and the n low-order bits represent page offset.

Eg:



$1K = 2^{10}$
$1M = 2^{20}$
$\because 1G = 2^{30}$

→ If LA (Logical address) = 31 bit, LAS = 2^{31} words
$= 2^{30} \cdot 2^1 = 2 \times 1G$
$= 2G$ words

→ LAS = 128 M words
$2^7 \times 2^{20} = 2^{27}$ words, the LA = 27 bits

→ PA = 32 bit, the PAS = 2^{32} words, PAS = $2^{20} \cdot 2^2$
$= 4 \times 1G = 4G$ words

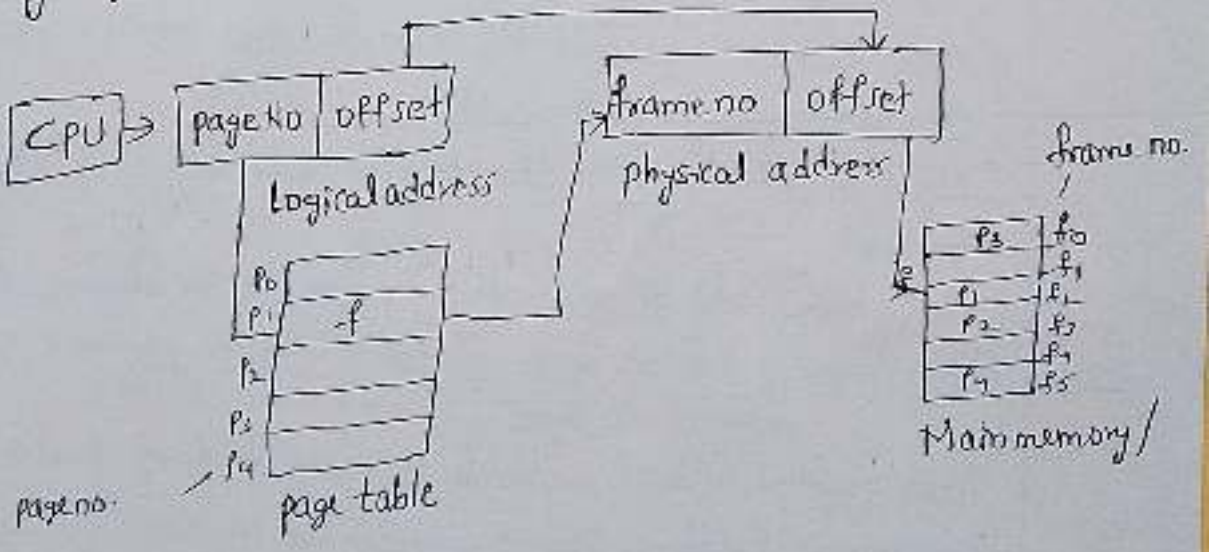
→ PAS = 16M words
$2^4 \times 2^{20}$ words, then PA = 24 bits

→ no of frames = Physical address space / frame size

→ no of pages = Logical address space / page size

→ frame offset = page offset.

Translating logical address into physical address

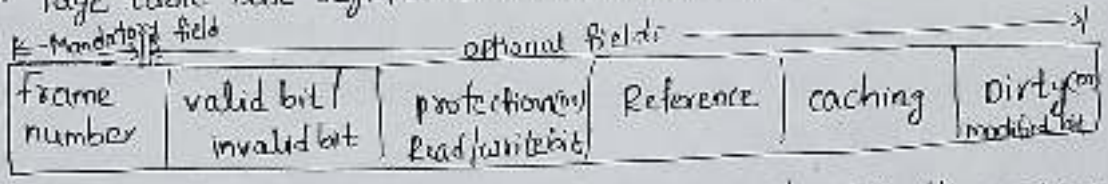


Page table:

→ page table maps the page number referenced by the cpu to the frame number where that page is stored.

→ Number of entries in page table = Number of pages in which the process is divided.

→ Page table base register (PTBR) contains the base address of page table.



Frame no: specifies the frame where the page is stored in the main memory

valid/invalid bit: if bit is present in main memory it specify by 1 otherwise set to 0

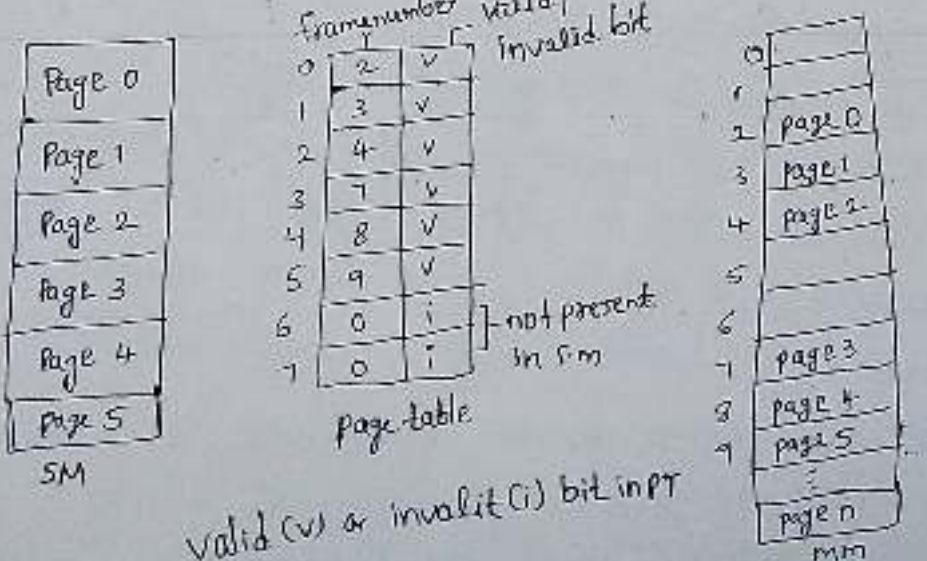
protection: To perform only read operation set bit as 0, if bit is set to 1 then both read and write operations are allowed

Reference: if the page has been referenced recently, then this bit is set to 1 otherwise set to 0. [used in LRU page replacement policy]

caching: whenever fresh or new data is required cache is disabled by setting bit as 1 otherwise set to 0

Dirty bit: if the page has been modified, then this bit is set to 1 otherwise set to 0. Dirty bit helps to avoid unnecessary writes.

Eg:



Valid (v) or invalid (i) bit in PT

→ In above example frame number 0 - indicate the page 6 and page 7 but those pages are not present in secondary memory

Page fault:

→ When a page referenced by the CPU is not found in the main memory it is called as a page fault.

→ When a page fault occurs, the required page has to be fetched from the secondary memory into the main memory.

→ Disadvantage of paging is, it increase the effective access time due to increased number of memory accesses i.e one memory access is get the frame number from the page table, another memory access is get the word from the page.

→ To reduce the effective access time we use TLB

Translation Lookaside Buffer:

→ Being a hardware, the access time of TLB is very less as compared to the main memory.

→ TLB consist of page Number and frame number.

In paging schema using TLB, the logical address generated by the CPU is translated into the physical address using following steps

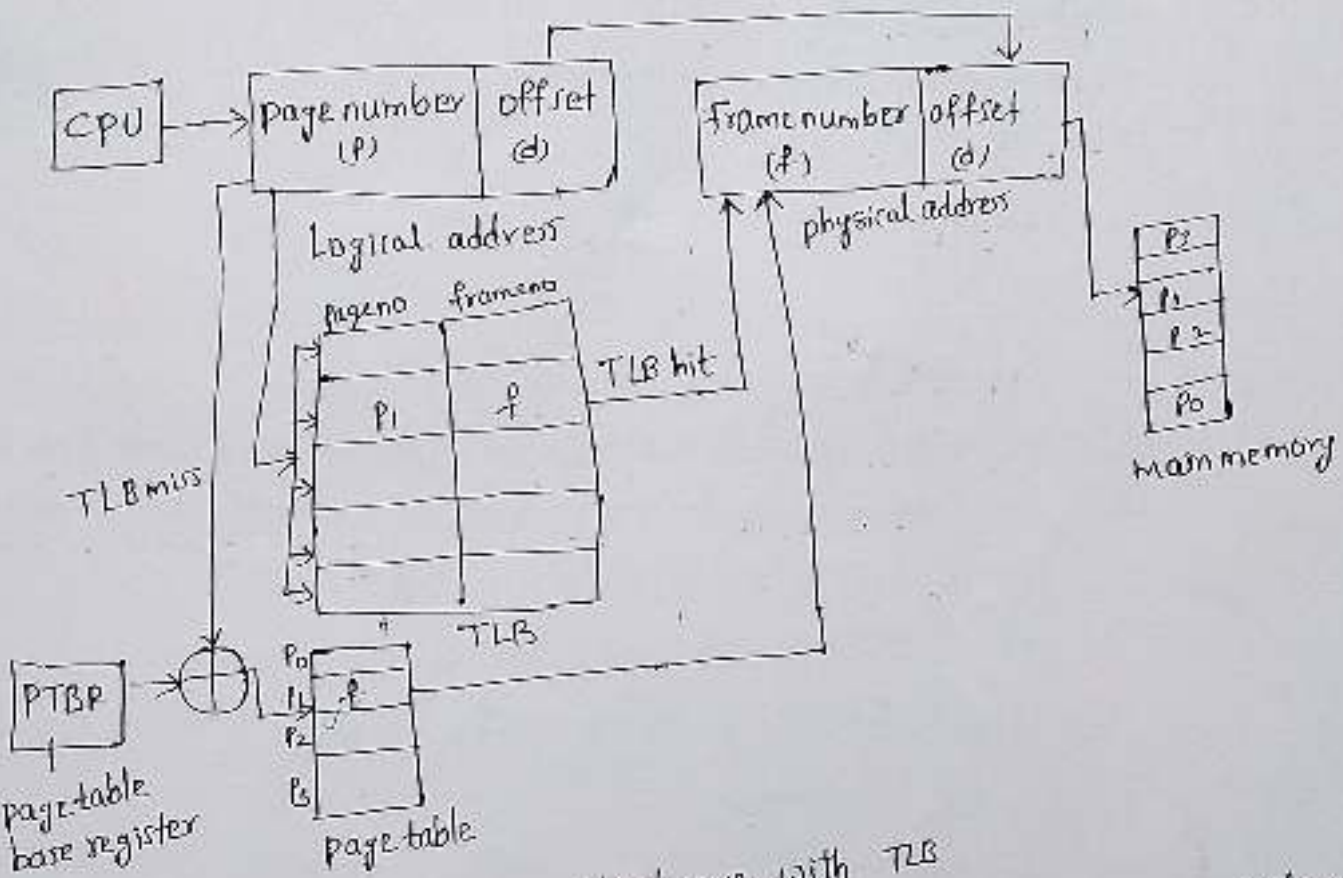
→ TLB is checked to see if it contains an entry for the reference page number. the referenced page number is compared with the TLB entries all at once.

→ if there is a TLB hit, TLB contains an entry for the referenced page number. in this case, TLB entry is used to get the corresponding frame number for the referenced page number.

→ if TLB does not contain an entry for the referenced page number, A TLB miss occurs. in this case, page table is used to get the referenced page number. Then, TLB is updated with the page number and frame number for future references.

→ After the frame number is obtained, it is combined with the page offset to generate the physical address.

→ Then, physical address is used to read the required word from the main memory.



Paging Hardware with TLB

→ To identify process TLB stors (ASIDs) address space identifiers in each TLB entry

→ Unlike page table, there exists only one TLB in the system

→ advantages of TLB is, it reduces the effective access time, only one memory access is required when TLB hit occurs.

→ The percentage of times that a particular page number is found in the TLB is called the hit ratio.

Effective Access Time:- Hit ratio of TLB × (Access time of TLB + Access time of mainmemory) + miss ratio of TLB × (Access time of TLB + 2 × Access time of main memory).

Eg: A paging scheme uses a TLB. the effective memory access takes 160 ns and a main memory access takes 100ns. what is the TLB Access time if the TLB hit ratio is 60% and there is no page fault.

16

Sol:

Effective access time = 160 ns

Main memory access time = 100 ns

TLB hit ratio 60% = 0.6

TLB miss ratio = 1 − TLB hit ratio

= 1 − 0.6

= 0.4

Let TLB access time = T ns

EAT = Hit ratio of TLB × (Access time of TLB + Access time of MM) +
Miss ratio of TLB × (Access time of TLB + 2 × Access time of

$160 = 0.6 * (T + 100) + 0.4 \times (T + 2 \times 100)$

$160 = 0.6 \times T + 0.6 \times 100 + 0.4 \times T + 0.4 \times 200$

$160 = 0.6 \times T + 60 + 0.4 \times T + 80$

$160 = T + 140$

$160 - 140 = T$

$T = 20$ ns //

Eg2: A paging scheme uses a TLB. A TLB access takes 10 ns and a main memory access takes 50 ns. What is EAT if the TLB hit ratio is 90% and there is no page fault.

A:

TLB access time = 10 ns

Main memory access time = 50 ns

TLB hit ratio = 90% = 0.9

TLB miss ratio = 1 − TLB hit ratio ⇒ 1 − 0.9 = 0.1

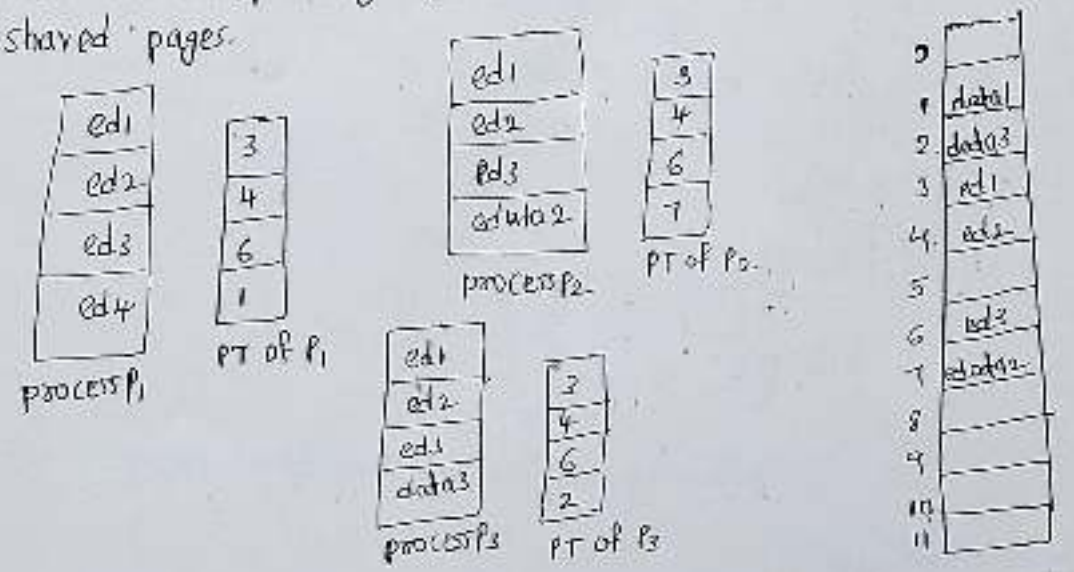
EAT = $0.9 \times (10 + 50) + 0.1 (10 + 2 \times 50)$

$= 0.9 \times 60 + 0.1 \times 110$

$= 54 + 11$

EAT = 65 ns //

→ The main advantage of paging is the possibility of sharing common code. some operating systems implement shared memory using shared pages.



Sharing of code in a paging environment

Structure of page table:

The most common techniques used for structuring the page table are

- Hierarchical paging (or multilevel paging
- Hashed page tables
- Inverted page tables

Hierarchical paging / multilevel paging:

→ If the page table might be too big to fit in a contiguous space, so we may have a hierarchy with several levels.

→ so, we break up the logical address space into multiple page tables.

In this technique we use

1. Two level page table
2. Three level page table.

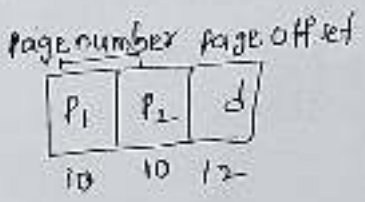
Two level page table:

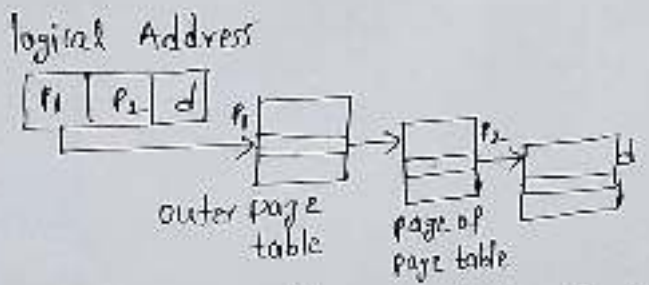
→ A logical Address (on 32-bit machine with 4K page size) is divided into a page number consisting of 20 bits
a page offset consisting of 12 bits

[∵ $4K = 2^2 \times 2^{10}$
$= 2^{12}$
= 12 bits of
32 - 12 = 20]

→ Since the page table is paged, the page number is further divided into
- A 10-bit page number
- A 10-bit page offset

Then, logical Address follows

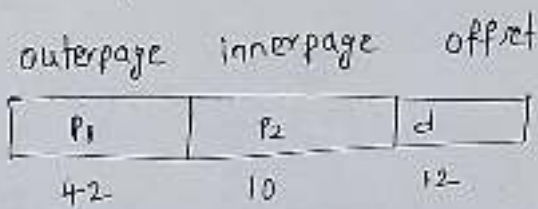


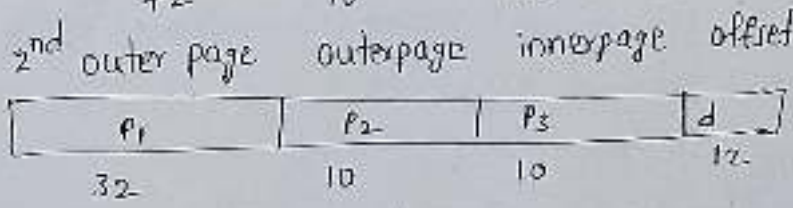


Address translation for a two level paging

Three level paging:

→ A logical address (on 64-bit machine with 4k page size) is divided into



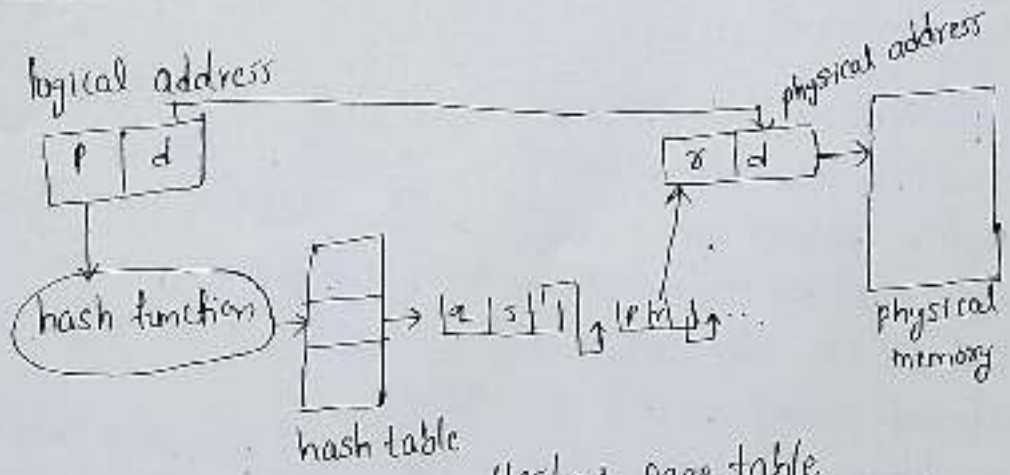


Hashed page tables:

→ it is common approach used when address space is > 32 bits

→ The virtual page number is hashed into a page table this page table contains a chain of elements or linked list elements hashing to the same location.

→ Each element consists of three fields
1. virtual page number
2. The value of the mapped page frame
3. A pointer to the next element in the linked list



Hashed page table

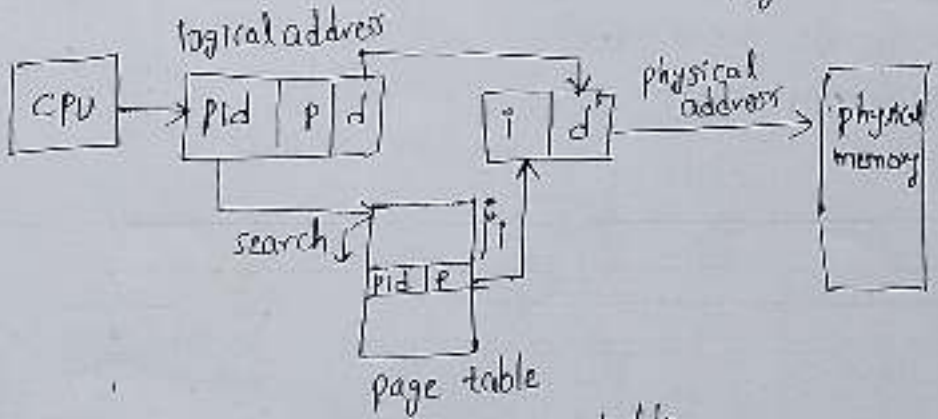
Inverted page tables:

→ The inverted page table combines a page table and a frame table into one data structure

→ One entry for each virtual page number & real page of memory.

→ Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page.

→ Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs.



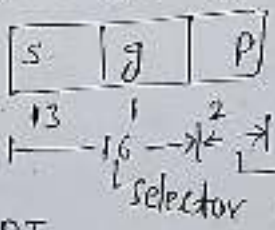
Inverted page table

→ < process-id, page-number, offset >

IA-32 Segmentation:

→ Intel pentium architecture allows a segment to be 4 GB large, and maximum number of segments per process is 16K.
→ The logical address of a process is divided into two partitions
→ the first partition consist of up to 8k segments that are private to that process. the second partition consist of 8k segments that are shared among all the processes.
→ First partition information kept in the local descriptor table (LDT)
→ Second partition information kept in the global descriptor table (GDT)
→ Each entry in LDT or GDT consist of 8-byte segment descriptor.

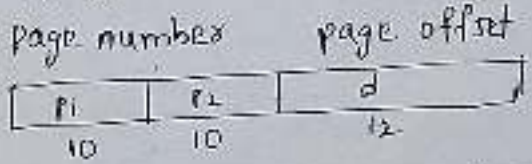
logical address represented as

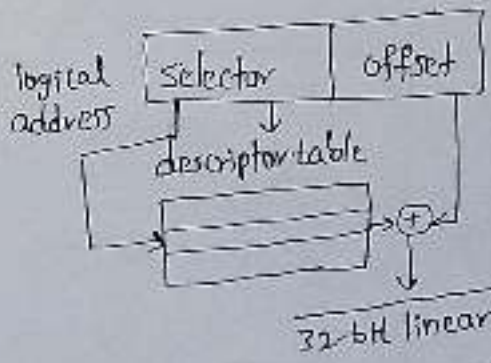


s- segment number
g - segment in the GDT or LDT
p - protection

IA-32 paging:

→ The pentium architecture allows a page size of either 4KB or 4 MB for 4-KB pages pentium uses a two-level paging scheme in which the division of the 32-bit linear address follows





Intel pentium segmentati

Copy - on - Write (COW):

→ fork() system call used for creating child process, again if you call the fork() system call it will create the duplicate of its parents and creating parent address space for child but it is unnecessary.

→ Instead of this we use a technique COW (copy-on-write)

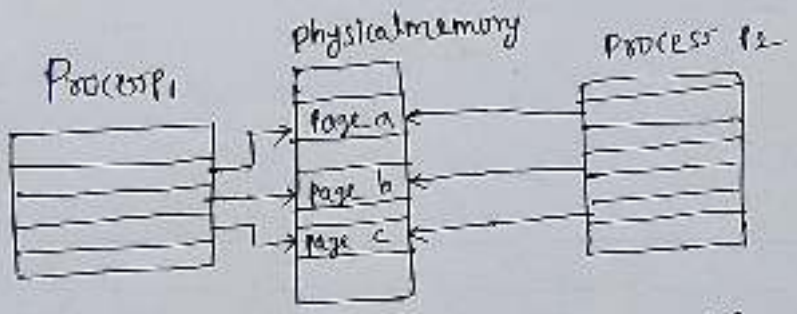
→ COW allows the parent and child processes initially to share the same pages these pages marked as COW.

→ By using these technique, only modified pages are copied and all unmodified pages can be shared by the parent and child processes.

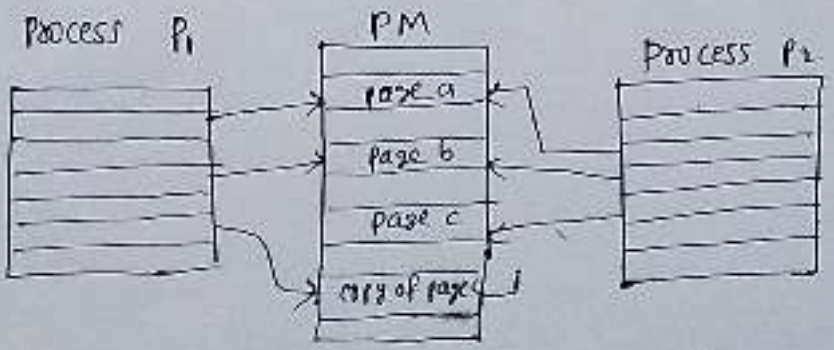
→ This technique is common in operating systems, including windows XP, Linux, & Solaris.

→ copied page can be allocated to free page

→ OS allocate these pages using a technique known as 'zero-fill-on-demand'



Before process P_1 modifies page c



After process P_1 modifies page c