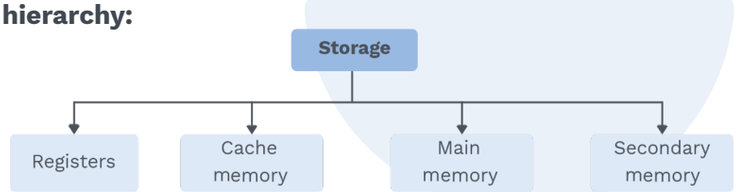
5 Memory Management

5.1 BASICS OF MEMORY MANAGEMENT

- The main purpose of a computer system is to execute programs. These programs, along with their data, must at least be partially present in the main memory during execution.
- To greatly enhance the utilisation of the CPU and its response time to users, many programs exist in the main memory. Thus various memory management methods are present to manage memory in different situations.
- Memory Management provides the functionality of allocating and de-allocating the main memory to the processes. It helps the operating system to keep a record of all the memory locations which are allocated to any process or are freely available.

Memory hierarchy:



Different places at which data is stored in a computer system in a hierarchical manner.

1) Registers:

- A CPU register is one of a small number of data storage areas found within the computer processor.
- Registers are a type of computer memory that is used to execute programs quickly and efficiently by storing data that is often used. The sole function of a register is to allow for quick retrieval of data for processing.

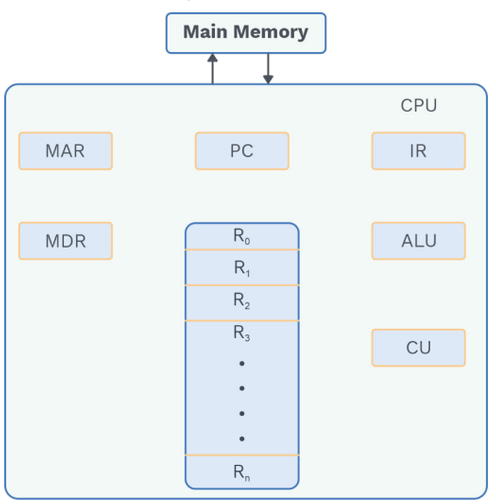
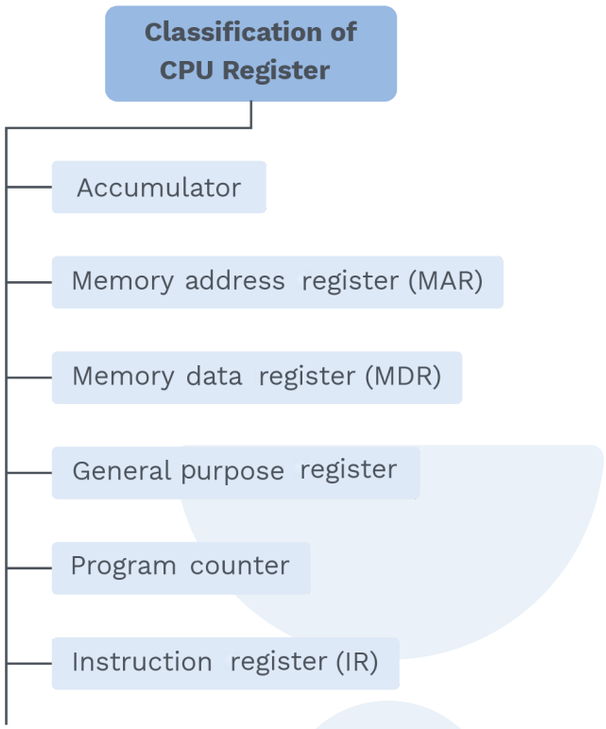


Fig. 5.1 Different Types of Register



- The CPU has direct access to registers; it can retrieve data from registers in a single clock cycle.

2) Cache memory:

- Cache memory is a volatile computer memory that saves frequently used applications and enables high-speed data access to a CPU.
- Cache memory is more expensive than main memory or disc memory, but it is less expensive than registers. It serves as a buffer between the processor and the main memory.
- Cache memory is used to lower the average time to access data from the main memory by storing copies of data from frequently accessed main memory locations. It is smaller and faster than the main memory.

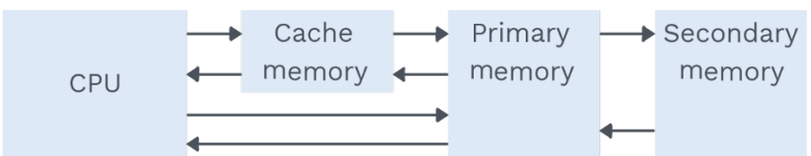


Fig. 5.2 Levels of memory hierarchy

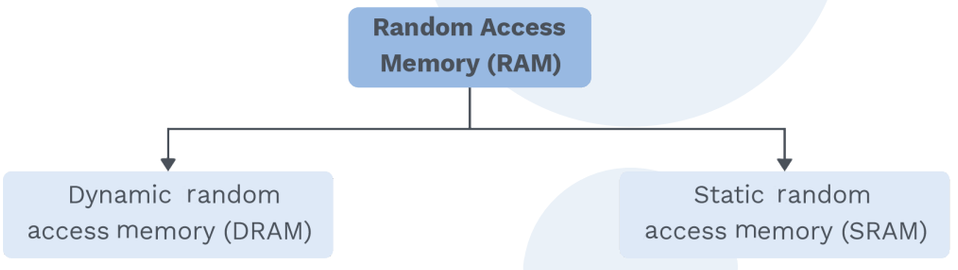
Note:

Data transfer is in words between CPU and cache memory and in block between cache and main memory.

3) Main memory:

It's also known as read–write memory, primary memory, or main memory.

- Because the data is lost when the power is switched off, it is a volatile memory.
- Random Access Memory is another name for it (RAM). It is the portion of the computer that contains the operating system, software applications, and other data for the processor. The term "random access" refers to the fact that the CPU can go to any portion of the main memory without having to proceed in sequence.



Note:

Data transfer between main memory and secondary memory is done by the operating system.

i) DRAM:

The most prevalent type of main memory in a computer is DRAM. In PCs, it is a common memory source. DRAM is continually recovering whatever data is currently stored in memory. It sends millions of pulses per second to the memory cells to refresh the data.

ii) SRAM:

It's a popular choice for embedded devices. SRAM data does not need to be refreshed on a regular basis. When the power is turned off, the information in this RAM stays as a static image until it is overwritten or destroyed. When not in use, it is less dense and more energy efficient.

4) Secondary memory:

- It is non–volatile and persistent computer memory that cannot be accessed directly by a computer.
- Primary memory has limited storage capacity and is volatile; this limitation is overcome by secondary memory.
- It is slower in data accessing. Typically main memory is at least six times faster than the secondary memory.

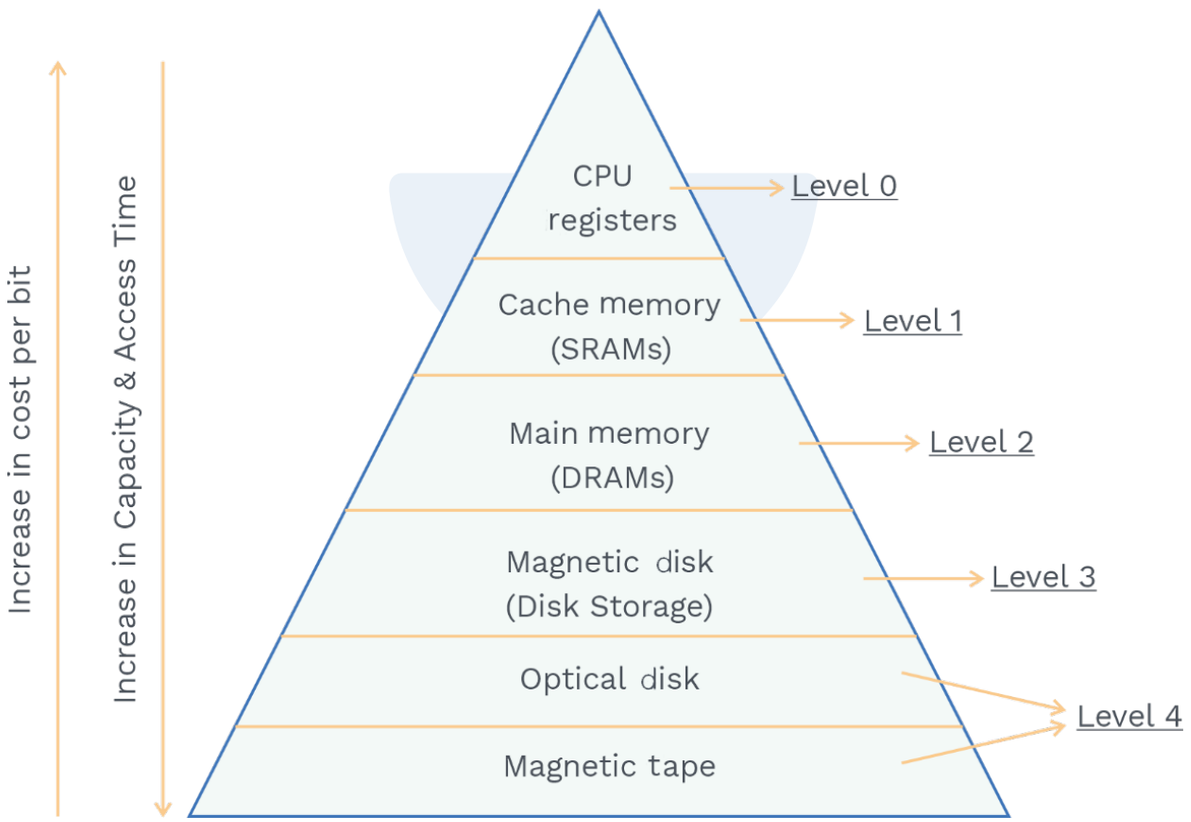


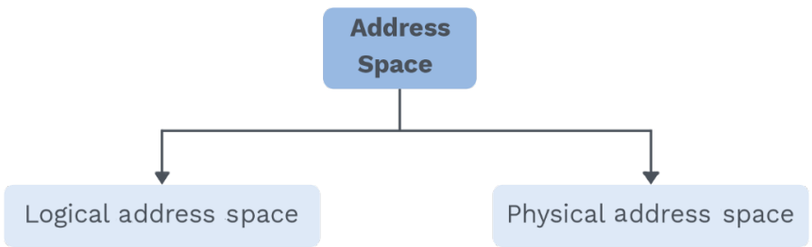
Fig. 5.3 Memory Hierarchy Design

Address space:

- An address space is the collection of all the addresses that are allocated to the program for its storage and execution. The memory locations in the address space can be accessed by the programs or the processes.

Classification of address space:

- Address space can be logical address space for storing the program or can be physical address space for executing the program.

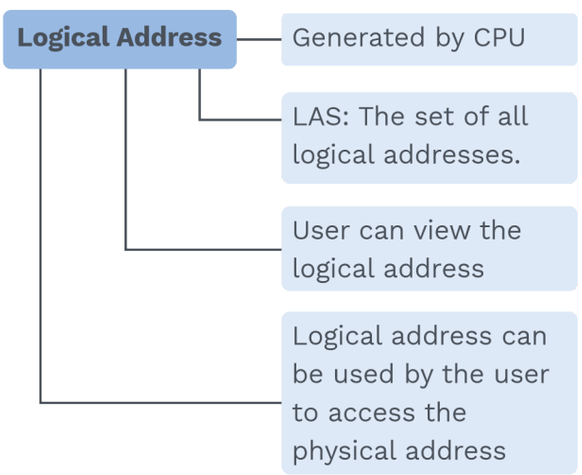


1) Logical address space:

- CPU generates logical addresses.
- It is used by the CPU to access the physical memory location as a reference.
- The set of all logical addresses generated from a program's point of view is known as the logical address space.

2) Physical address space:

- In memory, a physical address represents the physical location of requested data. The user never interacts with the physical address directly but can access it via its logical address.
- The logical address is generated by the user software.
- The hardware mechanism that converts a logical address to its physical address is the Memory-Management Unit (MMU).
- MMU must first map the logical address to the physical address.



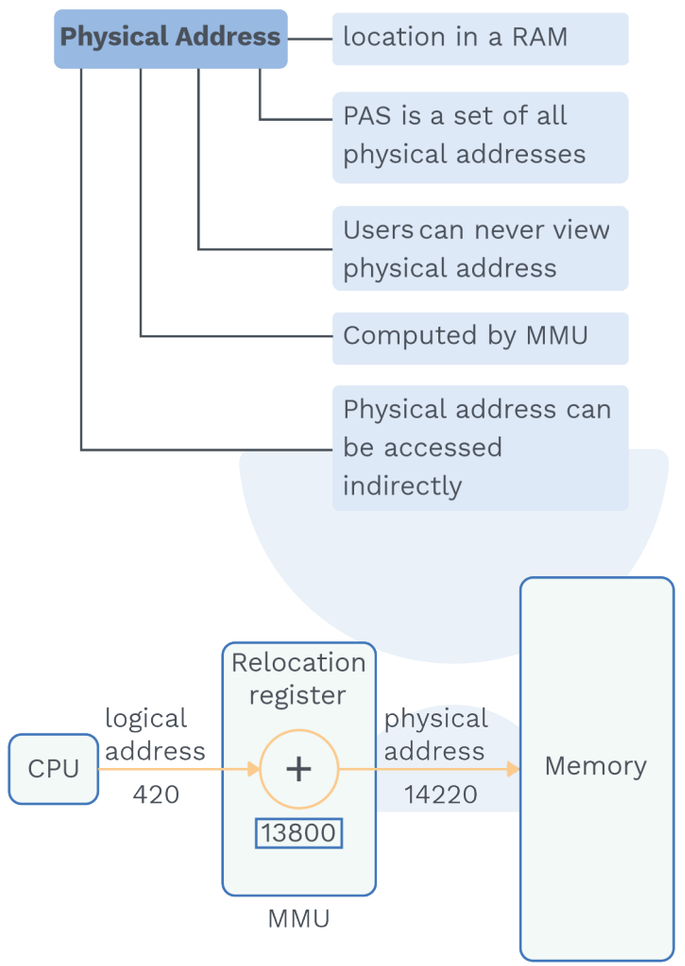


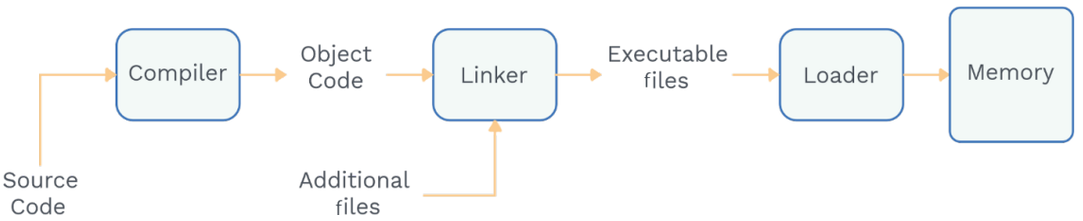
Fig. 5.4 Translation of Logical Address into Physical Address

Linker and loader:

Definition

The utility programs Linker and Loader play an important part in the execution of a program. Before being executed, a program's source code passes via the compiler, assembler, linker, and loader in that order.

Linking and loading:



1) Linking:

A program has multiple modules in it, which we compile together in order to generate a single executable code that can be loaded into the main memory for its execution. There are some codes which are reused, again and again; they are known as library functions. All different object files and library functions are combined into a single executable code; this process is known as linking. It can be done in two ways:

i) Static linking:

- Static linking is done during the compile time. In static linking, all the object modules and the library functions are combined in an executable file.
- Static linking is performed by special programs called linkers.
- Linker appends, all the library functions needed to execute the object file, which is generated by the compiler/assembler.

Advantage:

Programs in which all the modules and library functions are linked statically are faster than the programs in which all the modules and library functions are linked during run time.

Disadvantage:

- Executable file becomes very large, so loading time becomes more.
- If any library is replaced by a new version, all the programs must be re-compiled and re-linked to the new library version.

ii) Dynamic linking:

- Dynamic linking is a mechanism for linking a library into memory at runtime, retrieving variable and function addresses, executing the functions, and unloading the program from memory.
- It's frequently used to create software plugins. This method is used by Apache Web Server to load a dynamic shared object (*.dso) files at runtime.

Advantage:

- If any library is replaced by a new version, all the programs that reference the library will automatically use the new version without the need for recompilation.

Disadvantage:

- Program startup time is slower because during the execution, it checks all linked files if they are in main memory or not.

2) Loading:

- Initially executable code of every program is present in the secondary storage. But for the execution of the executable file, it must be present in the main memory.
- Bringing the code from secondary memory and storing it in the main memory is known as loading.
- It is done by the loader.
- Loader is a special program that takes the executable code present in the secondary memory, which is generated by the linker, and loads it into main memory. It can be done in two ways:

i) Static loading:

- Static loading refers to loading the entire executable file in the main memory before its execution starts.

Advantage:

- Program execution will be fast.

Disadvantage:

- Inefficient utilisation of main memory because even if the entire code need not be executed at once, loader still loads the whole code in main memory, thus occupying the main memory unnecessarily.
- Also, the size of the process which can be loaded into the main memory is limited by the size of the main memory. Thus, any process whose size is more than the size of the main memory cannot be executed.

ii) Dynamic loading:

- A routine in dynamic loading is not loaded until it is called.
- The main program is loaded into the memory and gets executed; other routines are kept on the disk in a relocatable format.
- When a routine needs to call another routine, the calling routine first checks whether the other routine has been loaded.

- If it is not, the relocatable linking loader is called to load the desired routine into the memory.
- This type of loading is useful when the large number of codes are needed to handle infrequently occurring cases, such as error routines.

Advantage:

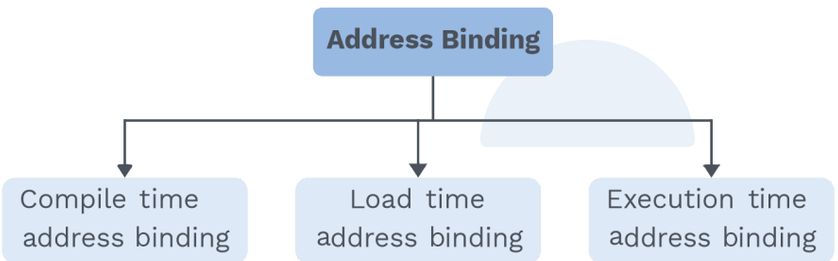
- Unused routine is never loaded into the main memory; thus main memory can be efficiently used.

Disadvantage:

- Program execution will be slower.

Address binding:

- Address Binding is the process of connecting application program instructions and data into physical memory locations.
- It refers to the process of converting one address space to another.
- The physical address relates to the location in the memory unit, but the logical address is generated by the CPU during execution.



1) Compile time address binding:

- If you know where the process will live in memory during compile time, an absolute address is generated, i.e. the physical address is given to the program's executable fire during compilation.
- Address binding is the responsibility of the compiler.
- It will be completed before the application is loaded into memory.
- To achieve compile–time address binding, the compiler must communicate with an OS memory management.

2) Load time address binding:

- If the location of the process is unknown at compile-time, a relocatable address will be produced. The relocatable address is converted to an absolute address by the loader.
- The loader generates an absolute address by adding the process's base address in the main memory to all logical addresses.
- It will be completed after the application has been loaded into memory.
- The OS memory manager, or loader, will handle this form of address binding.

Note:

If the process's base address changes during load time, address binding, the process must be reloaded.

3) Execution time/dynamic address binding:

- The CPU executes instructions of a process which are stored in memory.
- If a process can be relocated from one memory location to another during execution, this is employed.
- Even after the program has been loaded into memory, the address binding will be delayed. Until the program is finished, the application program will keep modifying the memory locations.
- The CPU does this type of address binding during program execution.
- It's compatible with dynamic absolute addresses.

Note:

Hardware support for address mappings, such as base and limit registers are required for dynamic address binding.

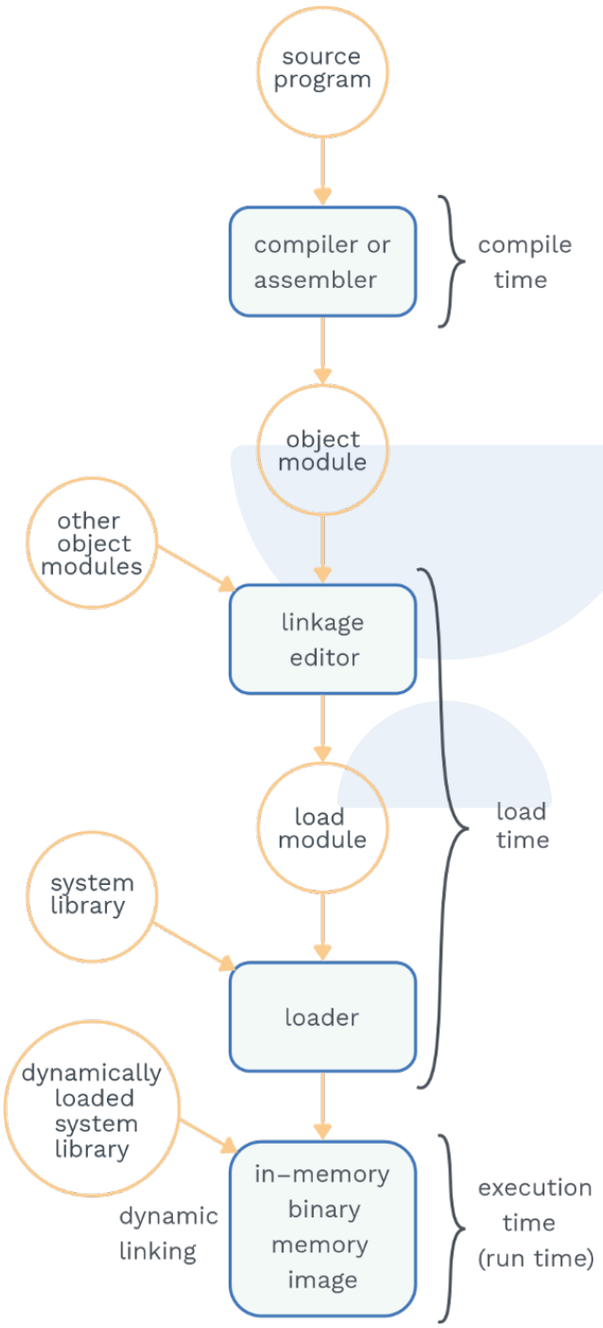


Fig. 5.5 Multistep Processing of a User Program

Swapping:

- A process must be in memory to be executed however, it can be temporarily moved to a backing store/secondary memory/nonvolatile storage, and then returned to main memory for further execution.
- Swapping allows the entire physical address space of all processes to surpass the system's actual physical (RAM) memory.
- Swapping is another technique for increasing the degree of multiprogramming in a system.

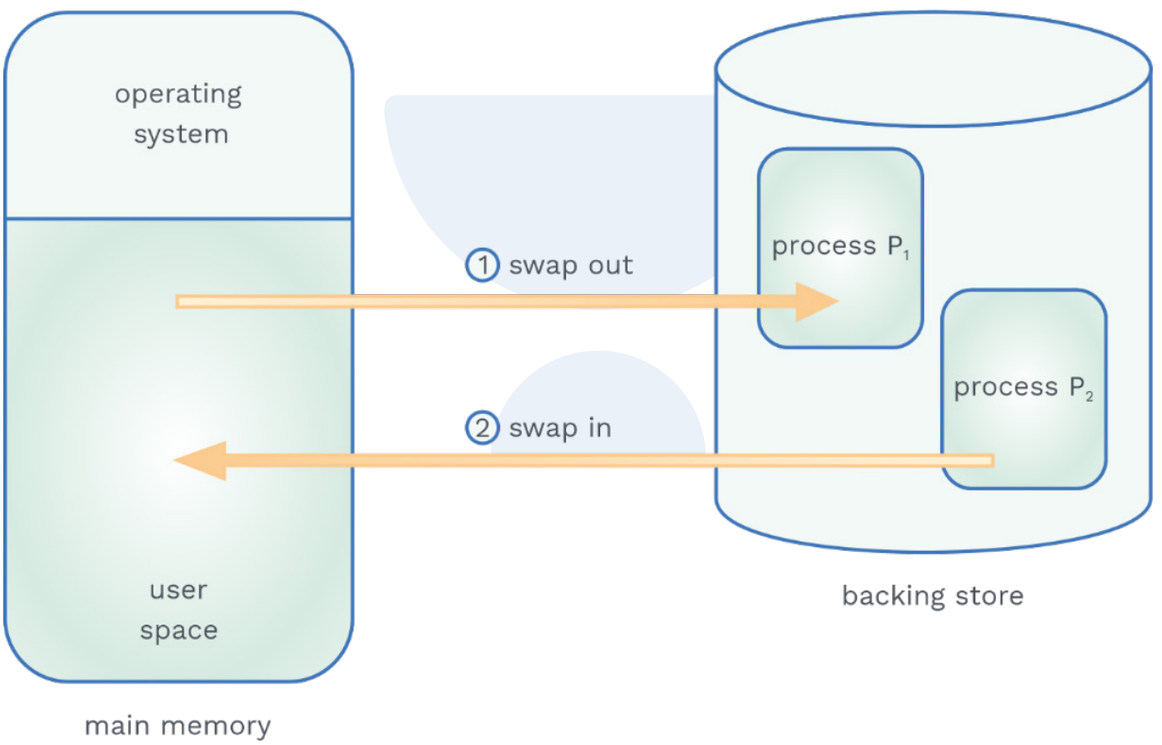
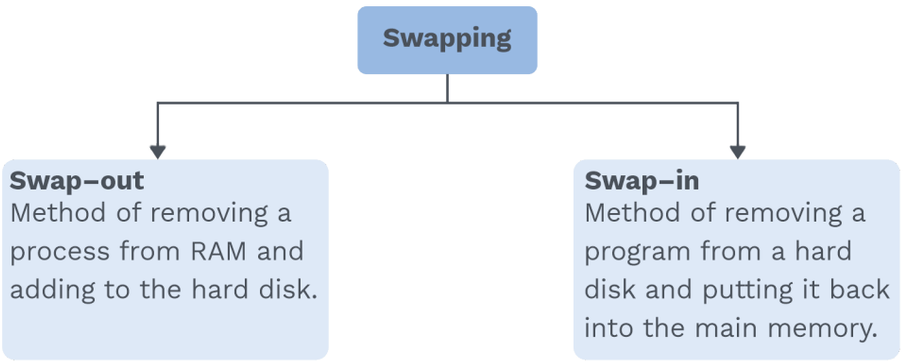


Fig. 5.6 Swapping of Two Processes Using a Disk as a Backing Store

Definition

Swapping is a memory management technique that allows any the process to be temporarily switched from main memory to secondary memory to free up main memory for other processes.

The concept of swapping has divided into two parts:



SOLVED EXAMPLES

Q1 **Suppose a user process of size 4096 KB is stored on a standard HDD, where swapping has a transfer rate of 1 MBPS. Calculate how long (in milliseconds) will it take to transfer the process from RAM to HDD?**

Sol: **Range: 4000-4000**

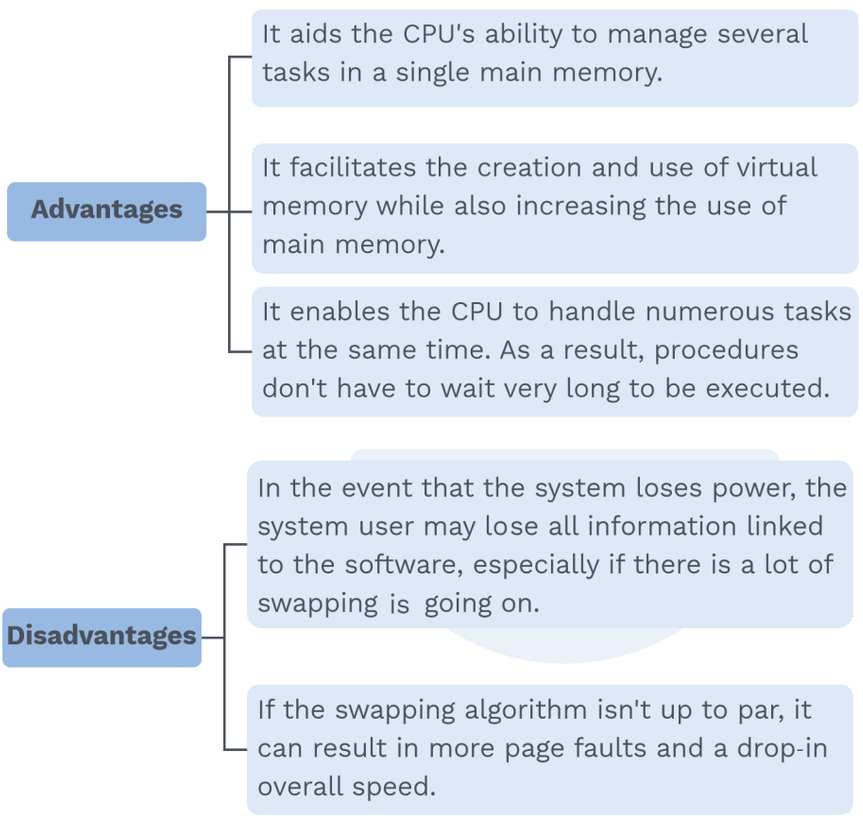
User process size = 4096 KB

Data Transfer Rate = 1 MBPS = 1024 KBPS

$$\text{Time} = \frac{\text{Process Size}}{\text{Transfer Rate}} = \frac{4096\,\text{KB}}{1024\,\text{KBPS}}$$

Time = 4 seconds

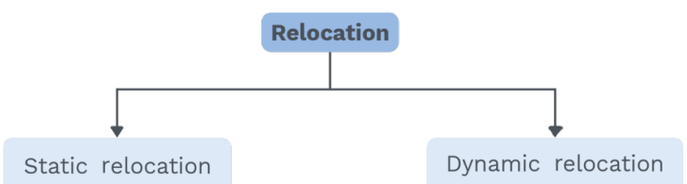
Time = 4000 milliseconds



Objectives of memory management system:

1) Relocation:

- The ability to shift processes around in memory without having any impact on their execution.
- Memory is managed by the operating system rather than the programmer, and processes can be moved into the memory.
- Memory Management (MM) is responsible for translating logical addresses to physical addresses.



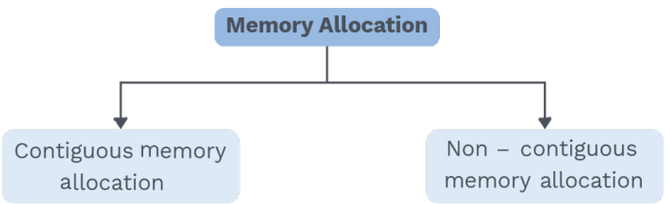
Static relocation:
Before or during the loading of the process into memory, the program must be relocated.
Relocator must be executed again if the program is not loaded into the same address space in memory.

Dynamic relocation:
In memory, processes can freely move about. At runtime, a virtual–to–physical address space mapping is performed.

2) Protection:
- To prevent data and instructions from being over–written.
- For data and instructions security purposes, OS is protected from user processes and on the same note, user processes are protected from other user processes.
- The reason behind this is many of the languages compute address of memory during runtime.

3) Sharing:
- Different processes may need to run the same operation or even access the same data at times.
- Different processes must access the same memory address when they signal or wait for the same semaphore.

Memory allocation:



Contiguous memory allocation:
The RAM is frequently partitioned into two parts: one for the operating the system, and the other for user processes.

Note:
The operating system can be installed in either a low-memory or a high-memory location. It is determined by the Interrupt Vector's location.

Definitions

It is basically a method in which a single contiguous part of memory is allocated to a process.

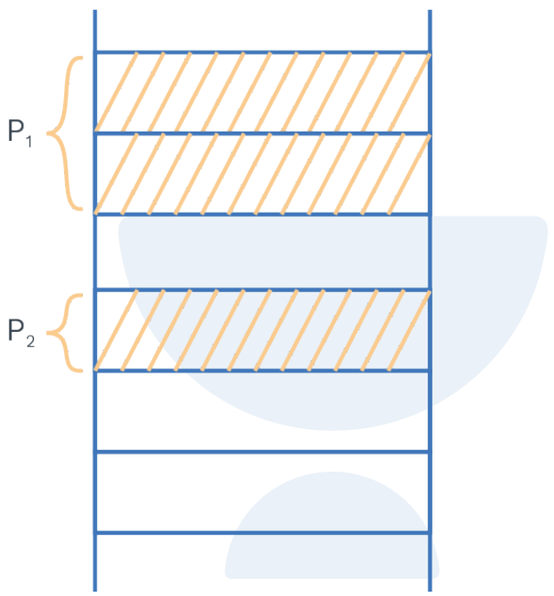


Fig. 5.7 Contiguous Memory Allocation

- Unused memory space is allocated to the same location in contiguous memory.
- Processes are faster in execution.
- Processes are easier for the OS to control.
- Address translation is minimum while executing a process.
- It suffers from both Internal and external fragmentation.

Grey Matter Alert!

Internal fragmentation:
When memory blocks associated with a process have unused space within the block, it is called internal fragmentation.

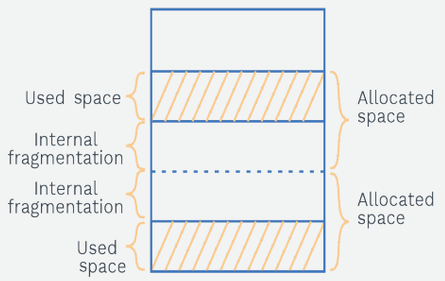


Fig. 5.8 Internal Fragmentation

External fragmentation:
When there is enough space within the memory to satisfy a method's memory request, but the process' memory request cannot be completed because the memory is spread in a non-contiguous manner, several approaches, such as compaction, are applied to solve external fragmentation.

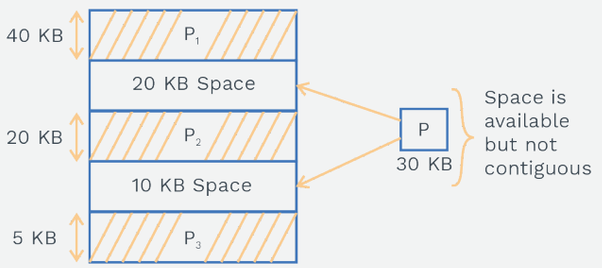
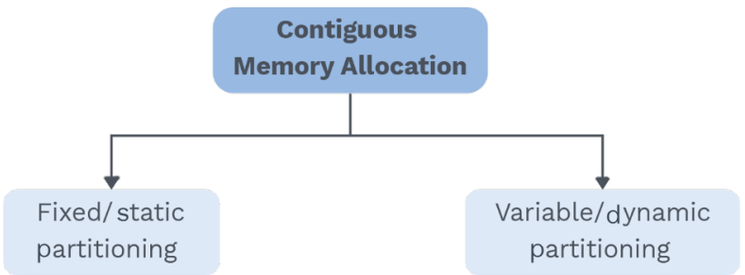


Fig. 5.9 External Fragmentation

In the above figure P1, P2 and P3 have been allocated in RAM, leaving holes of 20KB and 10KB which are not contiguous. When a process P of 30KB request for space in memory; it cannot be allocated because available space is not contiguous.



Fixed partitioning:

- This is the simplest technique used to store more than one process at a time in the main memory.
- In this partitioning, a number of non-overlapping partitions in physical memory is fixed, but the size of each partition may or may not be the same.
- In this, only one process is allowed per partition.

Note:

The operating system is always installed in the first partition, with the remaining partitions being utilised to store user processes.

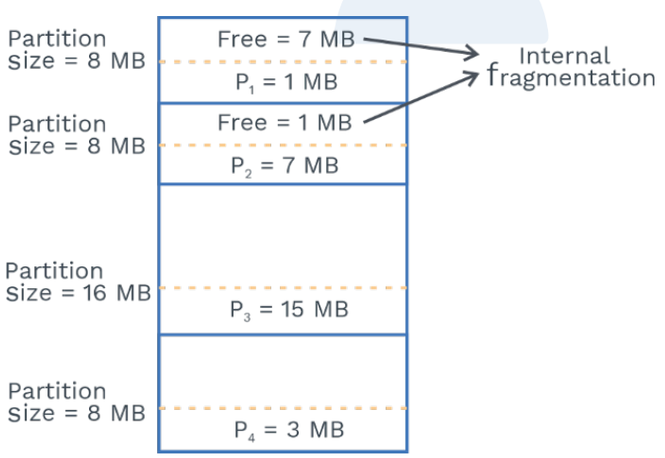


Fig. 5.10 Fixed Size Partition

Advantages:

1) Implementation is easy

2) Fixed partitioning processing necessitates less CPU resources, and minimal OS overhead.

3) Fixed partitioning does not suffer from external fragmentation.

Disadvantages:

1) Internal Fragmentation is suffered in the fixed partition.

2) Because partitions are made before execution, it limits the degree of multiprogramming. For example, if there are 'x' partitions in RAM and 'y' is the number of processes, then the 'y' = 'x' requirement must be met. In fixed partitioning, processes that are larger than partitions (in terms of quantity) are invalid.

3) It restricts the size of the process since it cannot support processes larger than the partition size.

4) Spanning a process into two partitions is not possible.

Variable partitioning:

Initially, the whole user space of memory is free, and partitions are created as needed after the arrival of processes.

- The total number of partitions is not fixed and is determined by the number of incoming processes and the amount of RAM available.
- The operating system takes up the first partition, and the remaining space is dynamically partitioned.

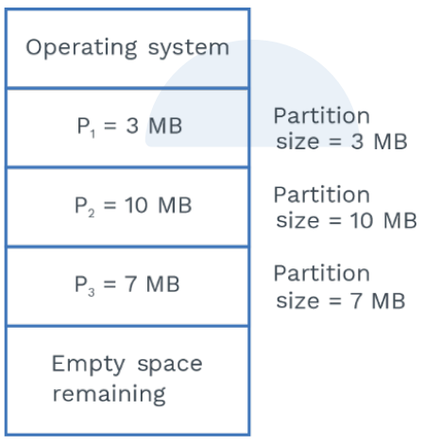


Fig. 5.11 Variable Partitioning

Advantages:

1) There is no internal fragmentation because the main memory space is allocated based on the needs of the process. There will be no unused partition space remaining.

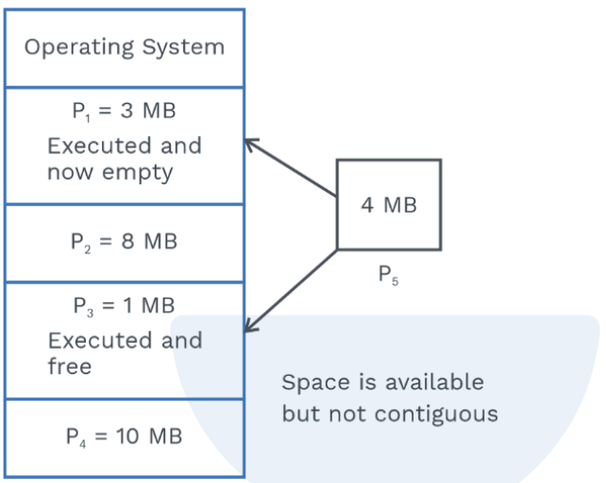
2) There are no restrictions on the amount of multiprogramming that can be done.

3) There is no limit on the size of the process requesting RAM.

Disadvantages:

1) Implementation is complex as the allocation of memory is at runtime.

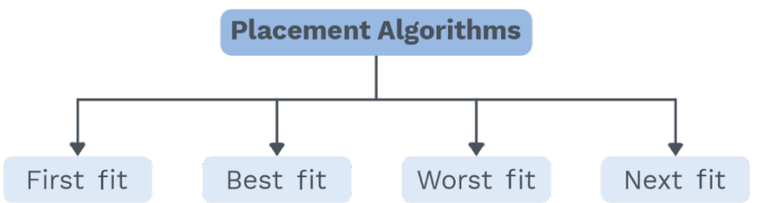
2) It suffers from external fragmentation.



In the above figure P_1, P_2, P_3 and P_4 has been allocated in RAM. Process P_1 and Process P_3 have been executed and freed up space of 3MB and 1MB, which are not contiguous. When process P_5 of 4MB requests for space in memory, but it cannot be allocated because available space is not contiguous.

Allocation policies:

When there are multiple partitions available to accommodate a process, a partition must be chosen using one of the partition allocation policies.



1) First fit:

It assigns the first available free partition (hole) that is large enough. It begins scanning the memory from the beginning and selects the first large enough block available. Among the several allocation policies, it is the quickest.

2) Best fit:

- It allocates the smallest sufficient free block that can hold the process.
- It gives the worst performance overall in terms of allocation time, as every time the smallest hole is found by searching in the whole memory.
- It will give least internal fragmentation.
- Compaction is done here more often.
- Best fit is an efficient allocation policy for fixed partitioning.

3) Worst fit:

- It allocates the largest block among all available blocks that is big enough.
- It is the opposite of best fit allocation.
- Worst fit is efficient allocation policy for variable partitioning as it will create holes of larger size so that other small processes can be placed in those memory holes. .

4) Next fit:

Next fit will start searching from last allocated partition; rest is the same as that of the first fit.

Rack Your Brain

Is Best–fit really the best allocation technique among others in fixed partitioning?

Grey Matter Alert!

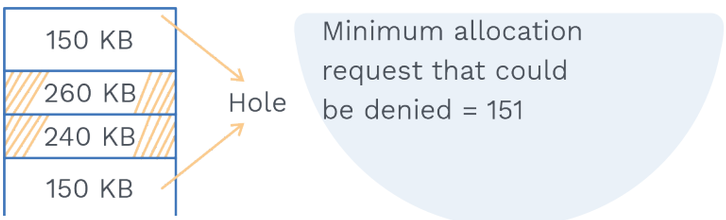
Compaction:
Compaction or shuffling memory contents is a solution for external fragmentation; all free memory is gathered into one single block. Moving should be dynamic in order to make compaction possible. Using a paging or segmentation technique, external fragmentation can be resolved.

SOLVED EXAMPLES

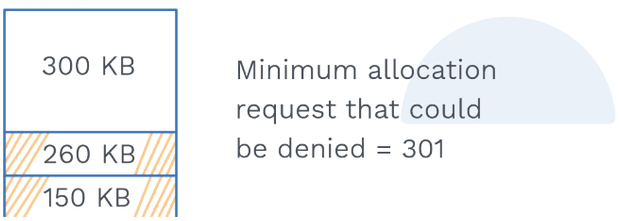
Q2 **A memory of size 800 KB is managed using variable partitioning with no compaction method. Memory currently has the following partitions:**
Partition I – 260 KB
Partition II – 240 KB
What is the smallest allocation request size (in KB) that could be denied?

Sol: **Range: 101–101**

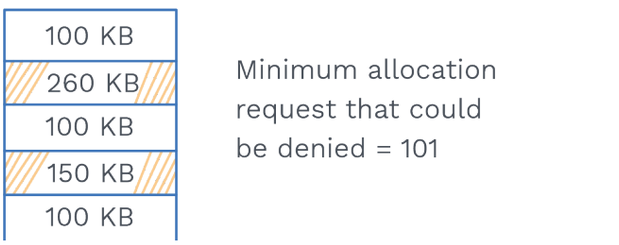
Case 1:



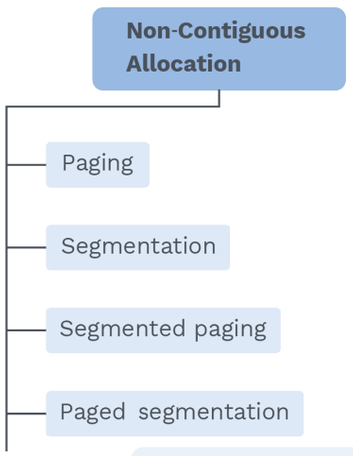
Case 2:



Case 3:



Non–contiguous allocation:
In contrast to contiguous allocation, it is a mechanism that allocates memory space in different locations to the process according to its needs.
All of the available vacant space is dispersed across.
This memory allocation strategy helps to prevent memory waste, which eventually leads to internal and external fragmentation.

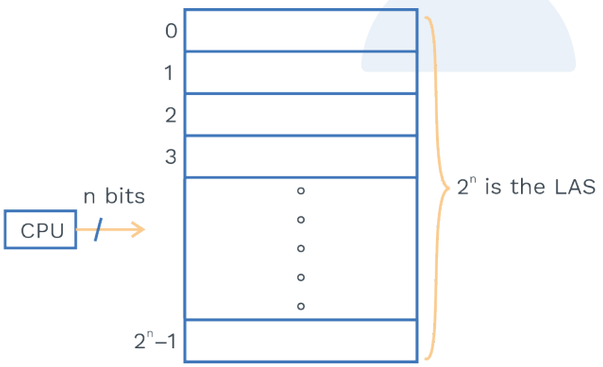


Paging:

- Paging is a storage mechanism used in operating systems to retrieve processes from secondary storage into main memory in the form of pages.

Logical address space (LAS):

It is the collection of logical addresses



LAS = 2^{LA}

Logical Address (LA) = $\log_2$LAS

- It is generated by the CPU and is also referred to as a virtual address.
- It is a reference to a memory location that is independent of the data that is currently assigned to that location.

Translation is required to convert logical to a physical address.

Some terminologies:

Address space:

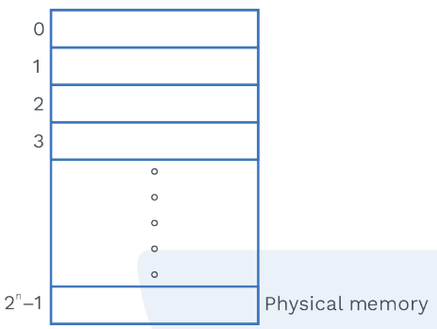
A set of words/memory locations/addresses are called address space.

Two types:

1) Physical Address Space (PAS)

2) Logical Address Space (LAS)

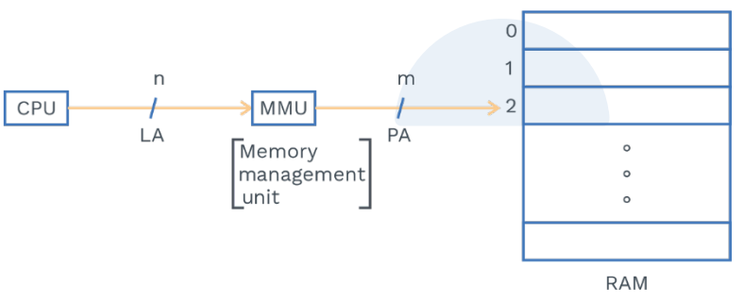
Physical address space:



PAS = 2^{PA}

PA (Physical Address) = ($\log_2$ PAS)

PA is the number of bits to uniquely identify each frame of physical memory



Example: Consider a byte addressable memory, where

LAS = 16 KB = 2^{14}B = LA: 14 bits

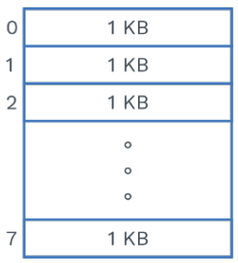
PAS = 4 KB = 2^{12} B = PA: 12 bits

Non-Contiguous memory allocation has mainly four steps.

1) Organisation of LAS:

- It is the view of the process from CPU perspective.
- Pages are equisised.
- Pages are nothing but logical division of long processes into smaller blocks.

Example: Consider the following logical address space(LAS), which is divided into equal size pages:



Size of page = 1KB
LAS: 8 KB = 2^{13} B

So number of pages (P) = $\frac{\text{LAS}}{\text{Page size}} = \frac{2^{13}\text{B}}{2^{10}\text{B}} \Rightarrow 8$

Now to represent eight pages, we need minimum of 3 bits, and 10 bits to identify each byte in pages, assuming memory is byte addressable.
Number of page bits (P) depends on number of pages in LAS(N).
$N = 2^P$
$P = \log_2(N)$

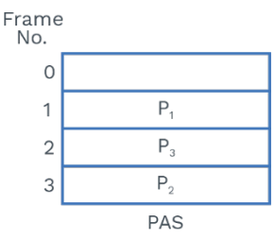
So LA = | 3 P | 10 d | (13)

P : Page Number
d : Page offset

2) Organisation of PAS:

- Physical memory is divided equisised into many logical parts known as frames.
- One frame accommodates one page of a process.
- Size of frame is equal to page size.

Example:



Size of PAS = 4 KB = 2^{12}B
PA = 12 bits
Frame Size = 1 KB

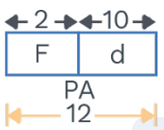
$\Rightarrow$ Number of frames (M) = $\frac{\text{PAS}}{\text{Frame size}}$

$$= \frac{2^{12}B}{2^{10}B} = 4$$

To represent 4 frames, we need atleast 2 bits and 10 bits for page offset.
The number of frames bits (F) depends on number of the frames in PAS(M).

$$M = 2^F$$

$$F = \log_2(M)$$



F : Frame number
d : Frame offset

3) Memory management unit – organisation:

- Each process is associated with its own page table, which are kept in the main memory (overhead).
- Page table is organized into a sequence of entries where entries are equal to the number of entries in logical address space (page table size).
- Page table entry (PTE) contains the essentially frame number of PAS in which the page referred is present.
- PTE denoted by 'e' is measured in bytes. The minimum size of PTE is 1 byte.
- PTS (Page Table Size) is given by

$$\boxed{PTS = N * e \text{ Bytes}}$$

N = No. of pages in LAS

e = Page table entry

- Page table of a process is not accessible to other processes.

4) Address translation:

- The LA (logical address) is generated by 'CPU' divided into a page number and offset, used as an index into the page table and then to frame number.

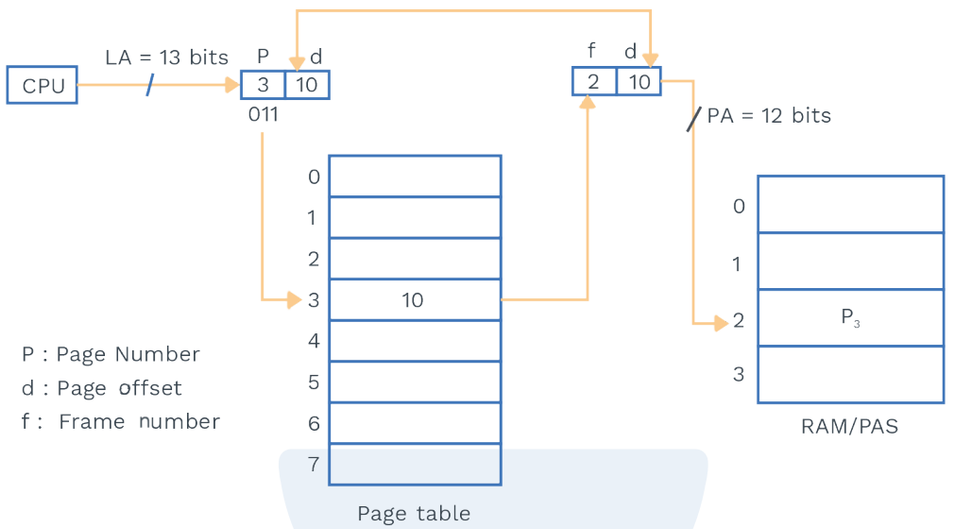


Fig. 5.12 Concept of Paging

SOLVED EXAMPLES

Q3 **Consider the following details of a byte addressable memory:**
LA = 29 bits
PA = 21 bits
Page size = 4 KB
If the page table entry size is 2 bytes.
What is the page table size(in KB)?

Sol: **Range: 256-256**

Let 1 W = 1 Byte

LAS = 2^{29} B

PAS = 2^{21} B

PS (Page Size) = 2^{12} B

$$\text{Number of pages} = \frac{\text{LAS}}{\text{PS}} = \frac{2^{29}}{2^{12}} = 2^{17} \text{ pages}$$

LA = | P | d | P : 17, d : 12

PA = | f | d | f : 9, d : 12

$\Rightarrow$ e = 2B

Page Table Size = $2^{17} \times 2B$

= 2^{18} B

= 256 KB

Q4 **Suppose LAS = PAS = 2^{16} bytes in a paging scheme. Consider page table entry size be 4 Bytes. What should be the page size (in bytes) of a process so that page table fits in exactly one page?**

Sol: **Range: 512-512**

Given process size is 2^{16}B

Let page size be 2^KB

So, number of pages = $\frac{\text{Process Size (LAS)}}{\text{Page Size}} = \frac{2^{16}}{2^K}$

$= 2^{16-K}$

Page table size $= 2^{16-K} \times 4B$

$= 2^{18-K}$ B

Now we have to fit this page table in one page.

So, $2^{18-K} = 2^K$

$\Rightarrow 18-K = K$

$\Rightarrow 18-2K = 0$

$\Rightarrow K = 9$

So page size is 2^9B = 512B

Q5 **Consider a system with byte addressable memory, a virtual address space of 36 bits and an 8-KB page size. Size of the main memory is 512 MB. What is the approximate size of the page table (in MB) when PTE contains 2 valid bit, 2 modified bit and 2 reference bit?**

Sol: Range: 24-24

Virtual address = 36 bit

So VAS = 2^{36} bit

Number of pages = $\frac{2^{36}}{2^{13}} = 2^{23}$ Number of frames = $\frac{2^{29}}{2^{13}} = 2^{16}$

Page table entry contains frame number(16 bits; along with that, it also has 2 valid bits, 2 modified bits and 2 dirty bits.

Page table entry size = 16 + 2 + 2 + 2 bits

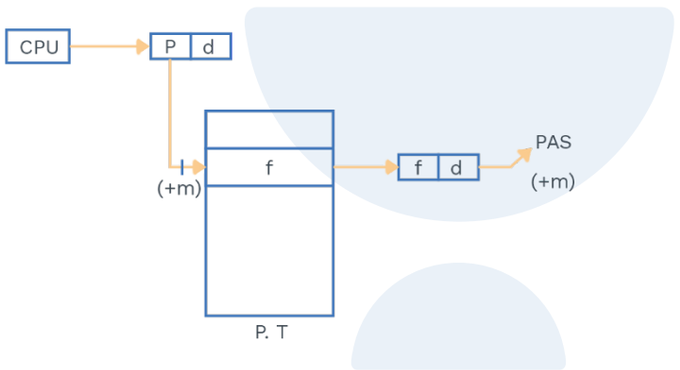
= 22 bits = 24 bits (Since PTE size must be in multiple of bytes)
Page table size = Number of pages × PTE size

$$= 2^{23} \times \frac{24}{8} \text{ B}$$

$$= 24 \times 2^{20} \text{ B}$$
$$= 24 \text{ MB}$$

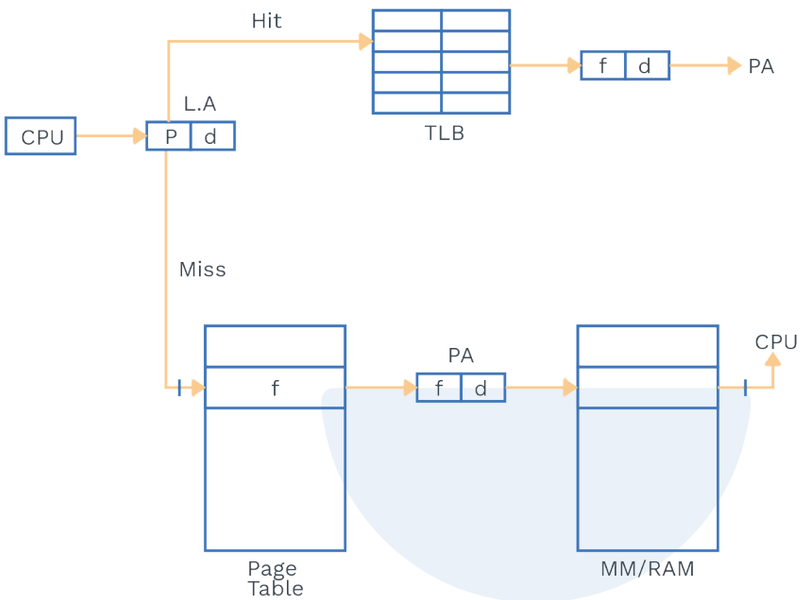
Performance of simple paging:
Two issues with paging technique

1) Time (temporal) issue:



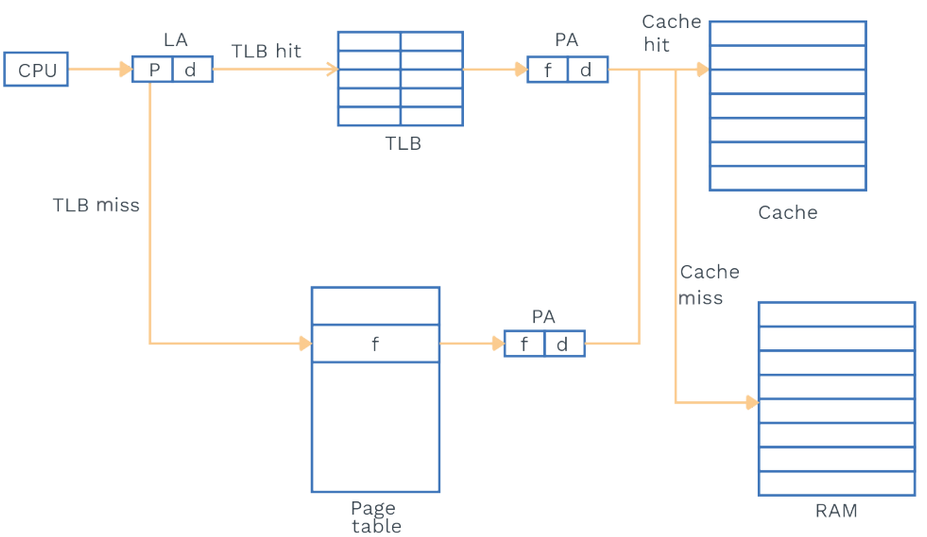
Set the memory access time to 'm' milli seconds.
Effective memory access time (EMAT)
= m(P.T.) + m(word)
= 2m
This EMA includes 1) Page table access time
2) Physical memory access time
So, to reduce EMAT we can use paging with TLB (Translation lookaside buffer)

Paging with TLB:



Let TLB hit ratio be 'x' ; $0 \leq x \leq 1$
TLB miss ratio = (1 – x)
TLB access time 'c': (c << m)
EMAT = x(c + m) + (1 – x) (c + 2m)

Paging with TLB and Cache:



Let TLB hit ratio be 'x': $0 \leq x \leq 1$
TLB miss ratio: (1 – x)
TLB access time be 'c': (c << m)
Cache hit ratio be 'y': $0 \leq y \leq 1$
Cache miss ratio: (1 – y)
Cache access time: K
Memory access time: m
Let all the pages are in main memory.
$EMAT_{(TLB+Cache)}$: x (c + y (K)+ (1–y) (K+m)) + (1–x) (c + m + y(K) + (1–y) (K+m))

SOLVED EXAMPLES

Q6 **Consider a system with a memory access time 280 ns. Assuming a TLB with access time 40 ns is used with the memory. What should be the TLB hit ratio (up to two decimal place) to reduce effective memory access time to 180 ns?**

Sol: Range: 1-1
C = 40ns
$EMAT_{(SP)}$ = 280 ns = 2 × m ⇒ m = 140 ns
$EMAT_{(SP + TLB)}$ = 180 ns = x(40+140) + (1 – x) (40 + 280)

x = 1 ⇒ 100 percent TLB hit when hit ratio is 1.

2) Spatial issue:

- It aims to reduces Page table size overhead.
- Reducing the page table size by increasing page size but, it leads to more internal fragmentation.
- Decreasing the page size will increase the number of pages thus increases Page table size.

Optimal page size:
An optimal page size should minimize both page table size and internal fragmentation.
Let logical address space = S bytes
Page size (PS) = P bytes

Page table entry = e bytes

$\Rightarrow$ So, number of pages in LAS $= \frac{S}{P}$

$\Rightarrow$ page table size (PTS) $= \frac{S}{P} * e$ byte

$\Rightarrow$ We take internal fragmentation due to the wasted memory in the last page of process is

IF (avg) $= \frac{PS}{2} = \frac{P}{2}$

Total overhead = PTS + IF $= \frac{S}{P}(e) + \frac{P}{2}$

We minimize total overhea by taking the first derivative with respect to 'P' and solving it:

$\Rightarrow \frac{d}{dP}\left(\frac{S}{P}(e) + \frac{P}{2}\right) = 0$

$\Rightarrow -\frac{S}{P^2}(e) + \frac{1}{2} = 0$

$\Rightarrow P^2 = 2Se$

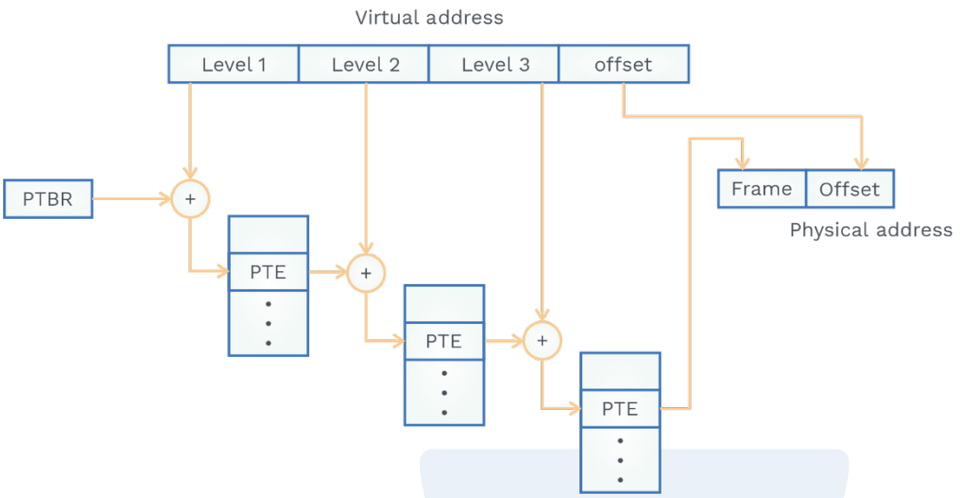
$\Rightarrow P = \sqrt{2Se}$

Conclusion is, $\sqrt{2Se}$ is the optimal page size to have minimum overhead of page table and internal fragmentation.

Multilevel paging:

Multilevel paging is a hierarchical paging technique that consists of two or more levels of page tables.

- This method is to avoid memory overhead of page tables. (i.e. larger page table size is not desirable)
- Multilevel paging refers to paging on a page table, in which the address space of the page table is partitioned into pages of equal-sized.
- The frames of PAS accommodate pages of page tables.
- Pages of the page table are accessed in PAS through another page table known as the outer page table.
- Page table base register stores the address of the level 1 (outermost) page table (PTBR).



1) PTBR value + level 1 offset present in virtual address created by CPU equals reference to PTE in level 1 page table.
2) Base address (present in level 1 PTE) + level 2 offset equals reference to PTE in level 2 page table (present in V.A.)
3) Base address in level 2 PTE + level 3 page offset equals reference to PTE in level 3 page table.
4) PTE is the address of the actual page frame (present in level 3).

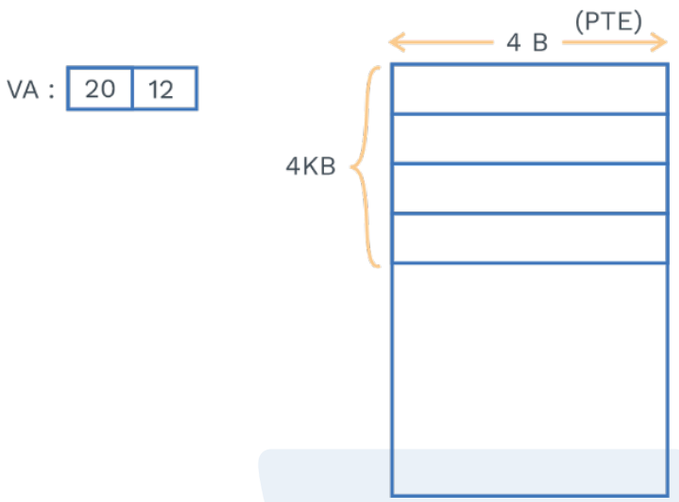
Example: Suppose we have a multilevel paging scheme where LAS = 4 GB. Page size = 4 KB. How many level page table exists. Assume PTE(e) = 4 B.

Sol: Number of pages $= \frac{\text{LAS}}{\text{PS}}$

$= \frac{2^{32}}{2^{12}}$

$\Rightarrow 2^{20}$

So to represent 2^{20} we need 20 bits atleast and 12 bits for page offset.



Number of pages in the innermost page table = 2^{20} pages

Page table size of innermost page table

= Number of pages × PTE size

$2^{20} \times 4B = 2^{22}B$

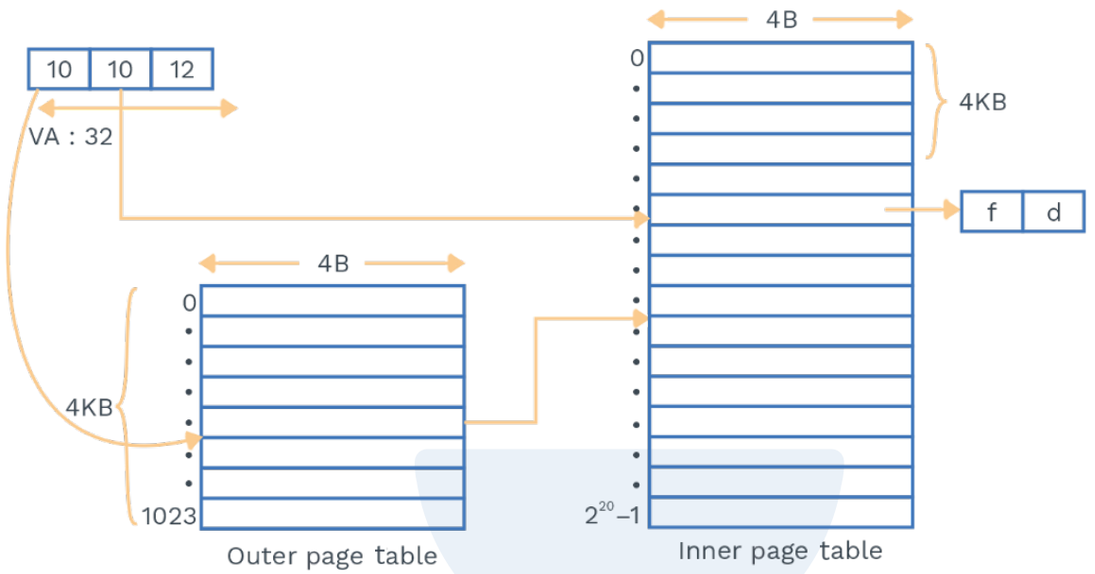
Which is greater than page size.

So more level of paging

Number of pages in next level page table

$$= \frac{\text{Page table size at inner level}}{\text{page size}}$$

$$= \frac{2^{22}B}{2^{12}B} = 2^{10}$$

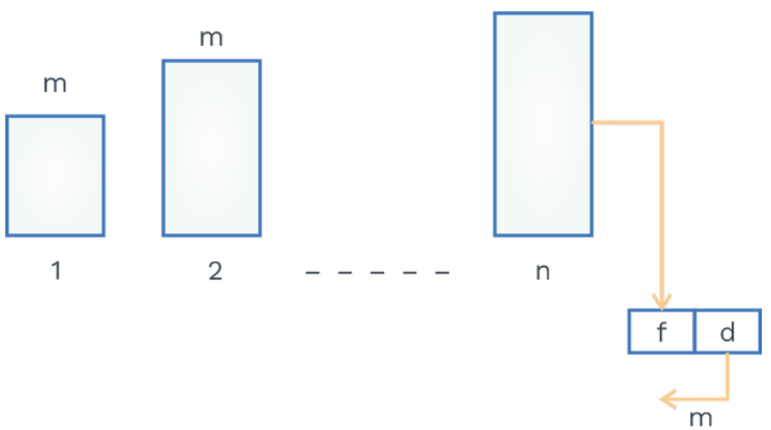


The outer page table has 2^{10} entries that point to the pages of the inner page table.
$= 2^{10} \times 4B = 2^{12}$ which is equal to page size, so no more page table is created.
So two levels of page table exist.

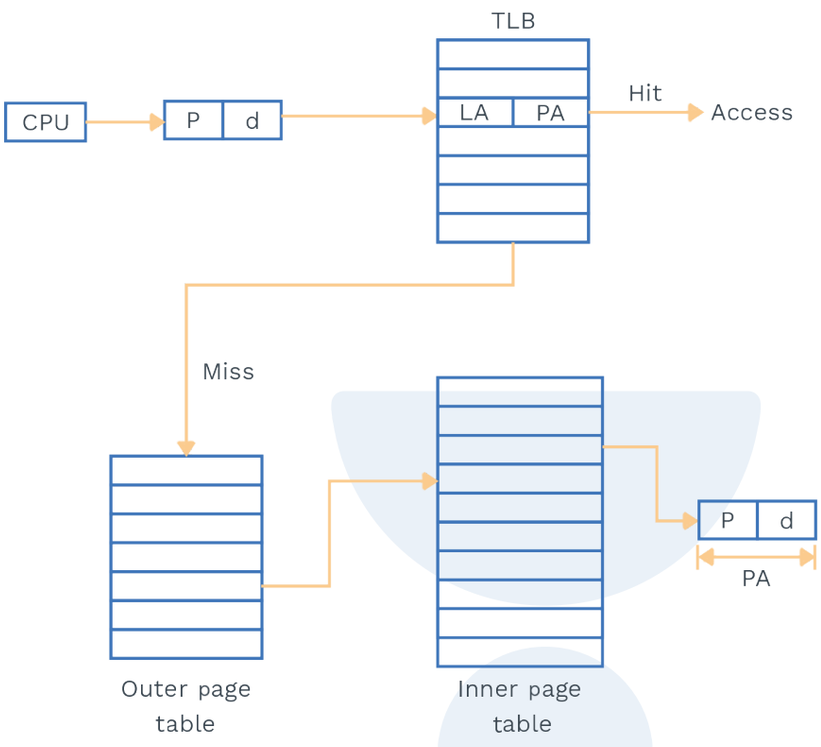
Performance of multi–level paging:

- Effective memory access time of 2–level paging

$(EMAT_{2LP}) = 3 * m$ ns
$\therefore$ m = memory access time
Generated formula for n–level paging
$EMAT_{'n'LP} = (n+1)\ m$ ns

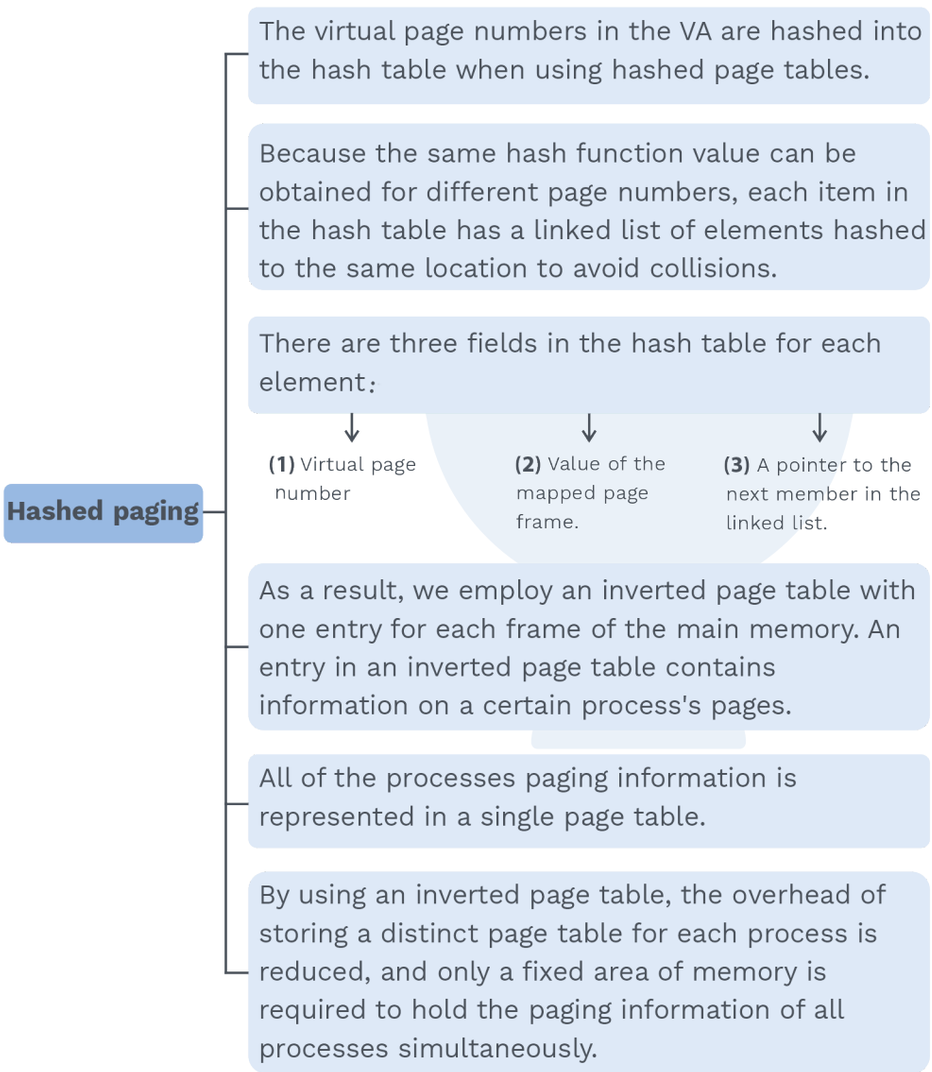


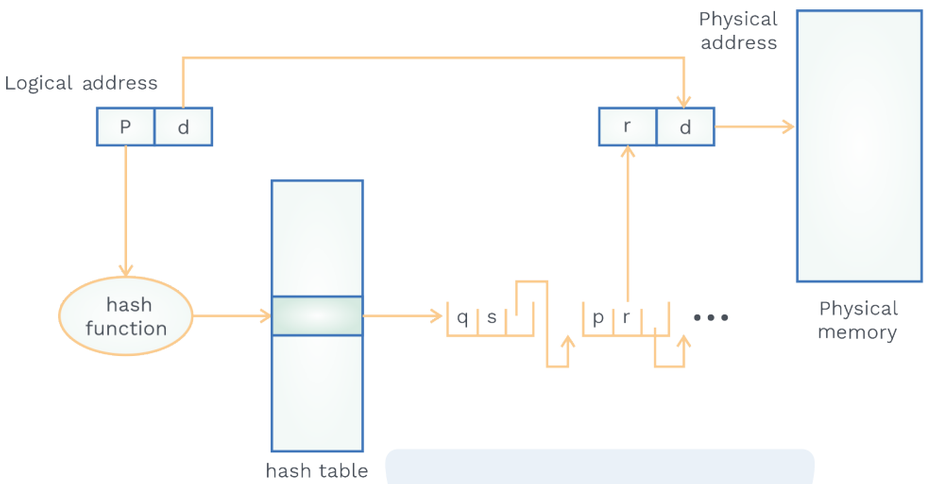
Multilevel paging with TLB:



Let TLB hit rate = x
TLB access time = c
memory access time = m
$EMAT_{2LP+TLB}$ = x (c + m) + (1 – x) (c + m + m + m)
= x (c + m) + (1 – x) (c + 3m)
Generalize formula if n level page table is present
$EMAT_{n\ LP+TLB}$ = x (c + m) + (1 – x) (c + (n + 1) m)

Paging with hashing (using data structures):





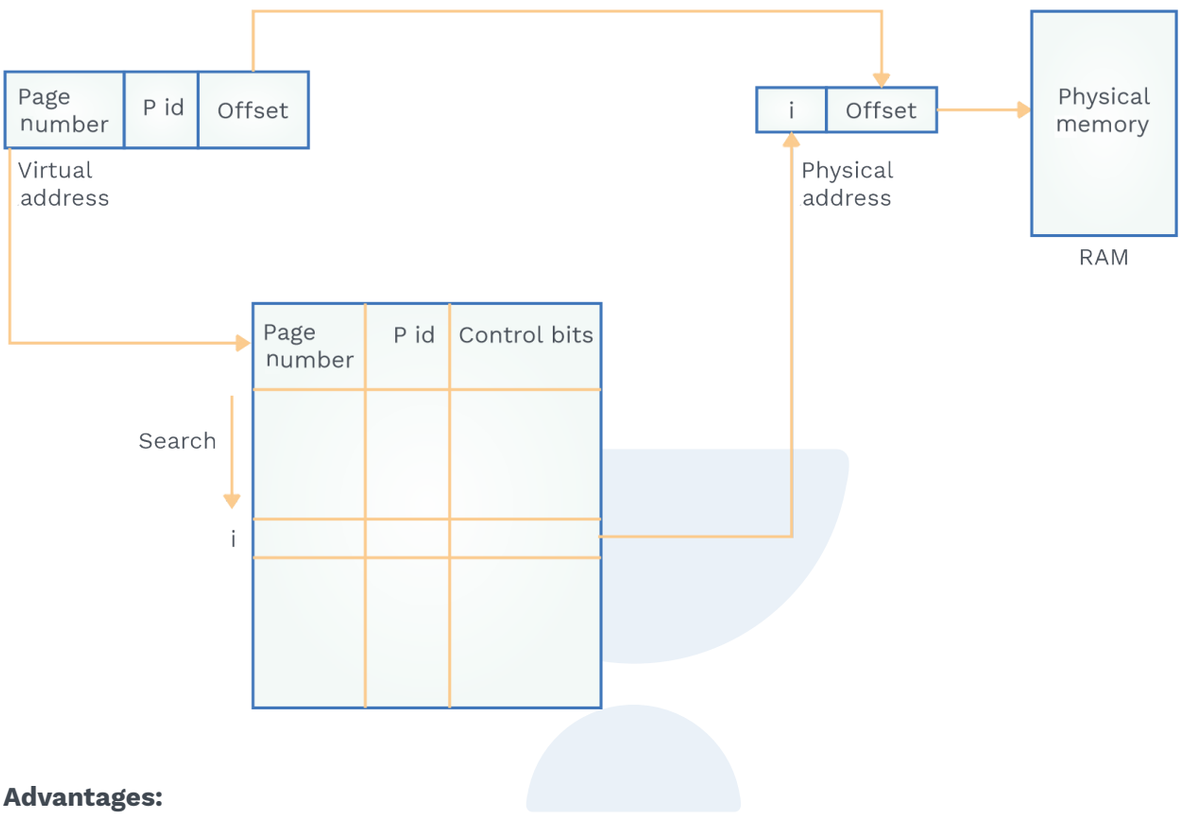
Inverted page table entry

Page number	Process-id	Control bits	Chained pointer
		It includes valid bit, dirty bit, reference bits, protection, etc.	It may be possible sometimes that two or more processes share a part of RAM. In this case, two or more logical pages map to same page table entry then chaining pointer is used to map the details of these logical pages to the root page table.

Working:

1) The fields are contained in the virtual address generated by the CPU, and each page table item contains other pertinent information.

2) When a memory reference occurs, the virtual address is matched by the memory-mapping unit, and an inverted page table is searched for a match with the appropriate frame number acquired from the page table index; if no match is discovered, a segmentation fault is issued.



Advantages:

- Reduced memory space

Disadvantages:

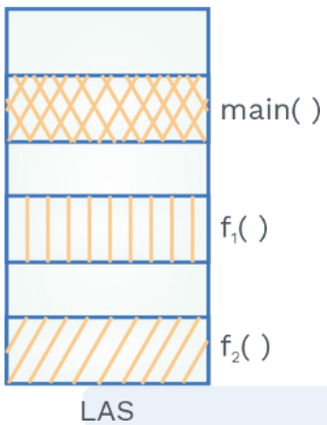
- As a result of the longer search time, the inverted page table is sorted by frame number, but the memory lookup is done with the virtual address in mind.
- Implementing shared memory is tough.

Segmentation:

There are certain limitations to simple paging, which are overcome by segmented paging.

- Paging does not preserve the user's view of memory allocation to the program.
- As per user view, the program is divided into logical parts representing functions, and procedures known as segments.
- Those segments which may be of different sizes are stored in PAS.
- The words of the segments are accessed in PAS through a segment table.
- Virtual address consists of segment number and offset.

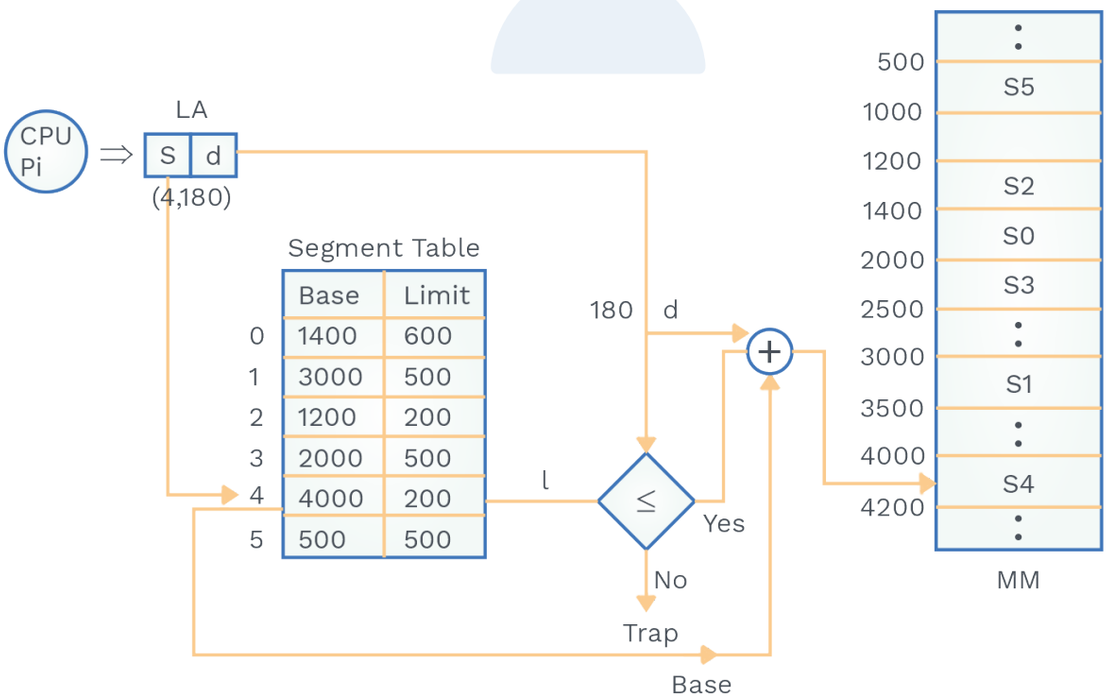
Organisation of LAS in segmentation:



- The whole program is divided into unequal size segments.

Organisation of PAS in segmentation:

- Variable sized partitioning.
- No concept of frames.
- Partition allocation methods such as best fit, Worst fit are used.



Example:

Logical Address	Physical Address
2 | 120	1200 + 120 = 1320 (Valid)
1 | 350	2000 + 350 = 2350 (Valid)
1 | 510	3000 + 510 = 3510 (Invalid)
4 | 180	4000 + 180 = 4180 (Valid)

- Logical address is divided into two parts.

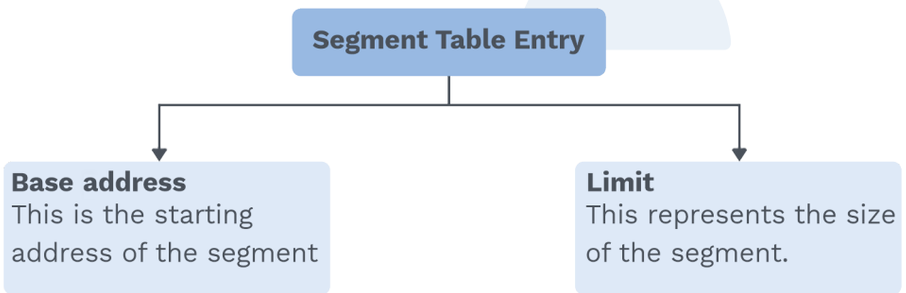
1) Segment number: It represents the segment, CPU requested the word from.

2) Offset: The address of a word displaced from the starting address of the first word of segment.

- Physical address is divided into two parts.

1) Segment starting address

2) Offset



- MMU checks the validity of the memory reference by calculating the absolute address by adding the base address and offset, and if the offset is greater than the limit specified in segment table, then a trap is generated, and MMU invalidates the memory request.

Segmentation v/s paging:

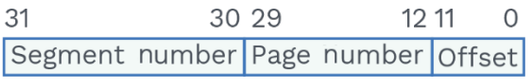
Segmentation	Paging
Non contiguous memory allocation	Non contiguous memory allocation
Variable sized segment	Fixed sized pages
Slower	Faster
It suffers from external fragmentation	It suffers from internal fragmentation
For segmentation, compiler is responsible	For paging, OS is accountable
OS maintains a list of holes in main memory	OS maintains a free frame list.
The segment size is given by user	Page size is determined by hardware
Segment table is used	Page table is used

EMAT in segmentation:
Let memory access time be 'm'.
EMAT (Effective memory access time of segmentation) = 2m
EMAT (Segmentation + TLB) = x (C + m) + (1 – x) (C + 2m)
x = TLB hit ratio
C = TLB access time
m = memory access time

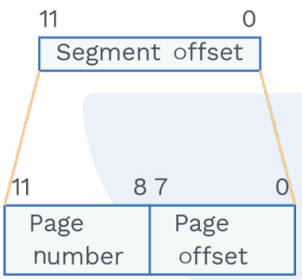
Issues with segmentation:
- External fragmentation occurs.
- Segment table slows down the translation process.
- External fragmentation can be overcome by segmented paging.

Segmented paging:
Segmented paging uses a dual concept of segmentation (partition of a process from user's view) and paging.
- The program is divided into variable-size segments, and each segment is further get divided into fixed-size pages.
- Pages are smaller than segments, and each segment has its own page table, implying that each program has several page tables.

- Segment number, page number, and page offset are used to represent virtual addresses.



- The logical address is made up of segment number and segment offset from the perspective of the programmer, but from the perspective of the operating system, segment offset is made up of page number and page offset for a page within its defined segment.



Example:

Let a segment be of 64 KB, and the virtual address generated by CPU is of 32 bits. Segmented paging is applied with page size (PS) = 1 KB.

$$\text{Number of pages} = \frac{\text{Segment size}}{\text{Page size}} = \frac{64\,\text{KB}}{1\,\text{KB}} = 64$$

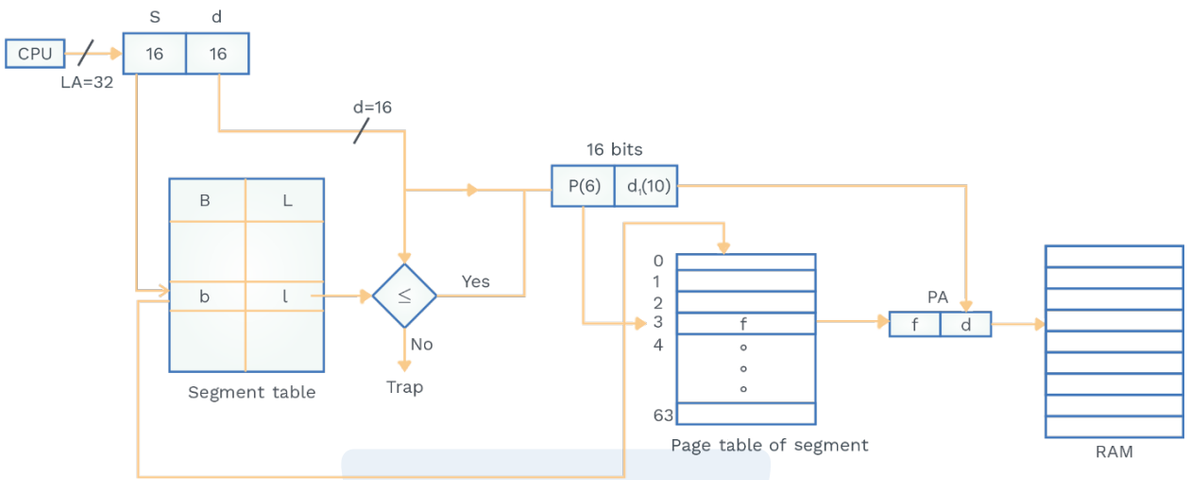


Fig. 5.12 Segmented Paging

Advantages of segmented paging:

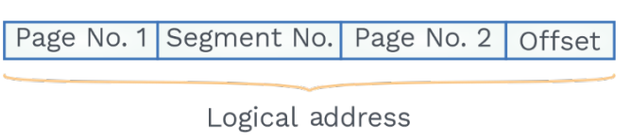
- The page table size is minimised because pages are only present for segment data, lowering memory requirements.
- It provides a user view as well as the benefits of paging.
- When compared to simple segmentation, it reduces external fragmentation.
- Because virtual memory (described later in this chapter) does not require the complete segment to be swapped out, the transition to virtual memory is simple.

Disadvantages of segmented paging:

- Internal fragmentation will still exist in paging.
- External fragmentation happens due to variable widths of page tables and segment tables in today's system, which necessitates the employment of additional hardware.

Paged segmentation:

- To avoid external fragmentation, we employ paging on the segment table, also known as Paged Segmentation.
- The logical address created by the CPU in paged segmentation will now contain of



S: Segment number
P: Page number
d: Offset
x: Starting address of the page of segment table
b: Base address of page table of segment
l: Limit or size of the segment

- Even with segmented paging, the page table can have a number of invalid pages. The issue with large page tables can be resolved by using multi-level paging.

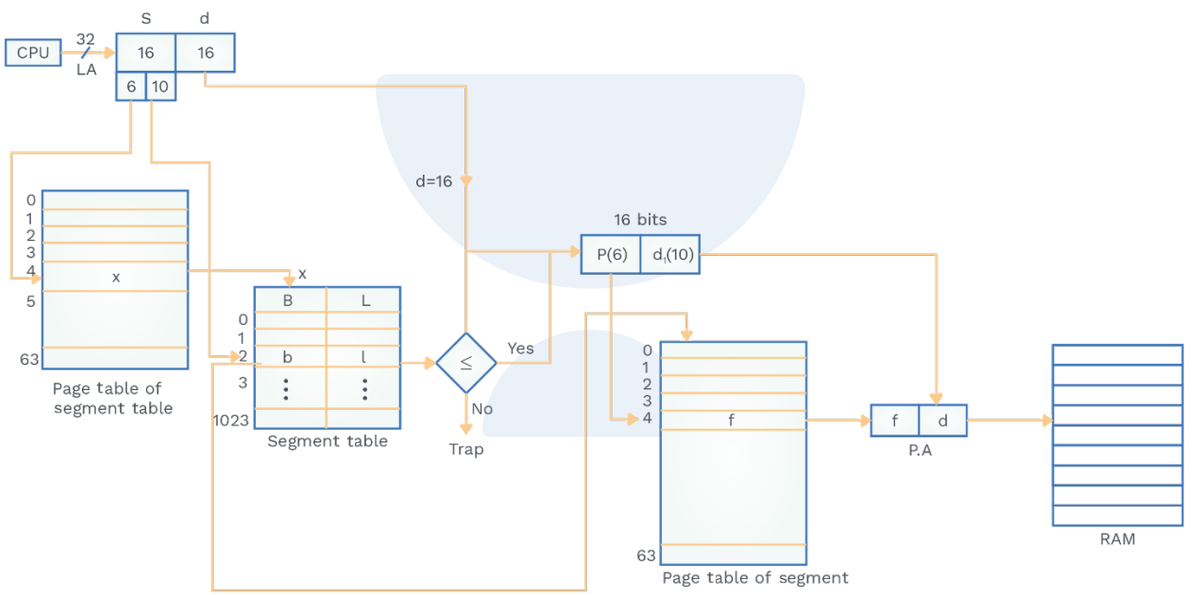


Fig. 5.13 Paged Segmentation

Advantages of paged segmentation:

- Page table size will be reduced.
- There is no external fragmentation, and memory requirements are reduced because the number of pages is limited to the segment size.
- Swapping out the full segment in virtual memory is not required.
- Like segmented paging, the size of the page table is reduced.
- No external fragmentation
- The entire segment need not be swapped out in virtual memory.

Disadvantages of paged segmentation:

- Internal fragmentation still exists.
- Hardware is required, which is more complex than segmented paging.
- Extra layer of paging at the first stage at segment table adds to the delay in access of memory.

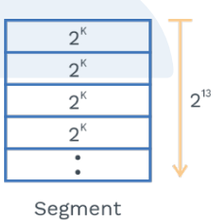
SOLVED EXAMPLES

Q7 **Suppose LAS = PAS = 2^{16} B, LAS is divided into 8 equisized segments and paging on each segment is applied. Let page table entry size be 2 byte. What should be the page size (in bytes) of a segment so that page table of segment fits in exactly one page?**

Sol: Range: 128-128

Size of segment is 2^{13} B. Let page size be 2^K B.

Number of pages in segment $= \frac{2^{13}}{2^K} = 2^{13-K}$



Size of page table = Number of pages × size of PTE

$= 2^{13-K} \times 2B$

$= 2^{14-K}$ B

Now we have to fit this page table in one page.

$$2^{14-K} = 2^K$$

$$14-K = K$$

$$K = 7$$

So page size = 2^7 B = 128B

Q8 **Consider a system with a virtual address of 38 bits that uses 2–level paging. The page directory is indexed with the most significant 10 bits, whereas the page table is indexed with the next 16 bits. In both levels, each entry is 4 bytes long. What is the maximum number of page tables a process can have?**

Sol: Range: 1025-1025
At the first, the level there is a page directory, and each entry in the page directory points to a page table. So, 10 bits are given for the page directory.
Then it points to 2^{10} = 1024 page tables.
So total no. of page tables = 1 + 1024 = 1025

Q9 **Consider a system with TLB support and two–level paging. TLB access takes 20 nanoseconds, while main memory access takes 80 nanoseconds. TLB has references for 130 pages out of 400 pages references. How long (in ms) does it take to access memory effectively?**

Sol: Range: 208 to 208
TLB hit ratio (x) = $\frac{130}{400}$ = 0.325

EMAT = x (C + m) + (1 – x) (C + 3m) ns

C : TLB access time
m : Memory access time

= 0.325 (20 + 80) + 0.675 (20 + 3 * 80)
= 32.5 + 175.5 ns
= 208 ns

Q10 **Consider a paging system with 1 KB pages and 32–bit virtual addresses. Each page table entry necessitates the use of 32 bits. The page table size should be limited to one page. When multilevel paging is used, how many levels are needed?**

Sol: Range: 3-3

Page table size of 1'st level (inner most) = $\frac{2^{32}B}{2^{10}B} \times 4B = 2^{24}$ B

2^{24} B is greater than 1 KB (2^{10} B)

So, page table size of 2nd level = $\frac{2^{24}B}{2^{10}B} \times 4B$

$= 2^{16}$ B

2^{16} B is also greater than 1 KB.

Page table size of 3rd level (Outer most) = $\frac{2^{16}B}{2^{10}B} \times 4B$

$= 2^{8}B$

2^{8} B is lesser than 2^{10} B.

So, 3 level of page table required.

Q11 **Consider a system implementing paging, and in which average process size is 16 MB, and each page table entry size is 16 B. The optimal size of page (in KB, rounded off to 1 decimal digit) to minimize the total overhead due to page table and internal fragmentation is (in KB)?**

Sol: Range: 22.5 to 22.6

Assume page size = x bytes

Process overhead = Page table overhead

+ Overhead due to internal fragmentation

Page table overhead = Number of pages × PTE

$$= \left[\frac{\text{Process Size}}{\text{Page Size}}\right] \times 16\,B$$

$$= \frac{16\,MB}{xB} \times 16\,B$$

Average overhead due to internal fragmentation.

$$= \frac{0 + x}{2} = \frac{x}{2}$$

So, total overhead of paging

$$= \frac{256\,MB}{x} + \frac{x}{2}$$

For minimizing overhead, differentiation with respect to 'x', then

$$-\frac{256\,MB}{x^2}+\frac{1}{2}=0$$

$$x^2 = 2 \times 256\ MB$$

$$x = \sqrt{512\,MB}$$

$$x = 22.6\ KB$$

Q12 **Consider a system with physical address of 40–bit, 16 K frame and page table contains 64 K entries. The number of bits in virtual address will be?**

Sol: Range: 42-42

64K = 16bit = number of pages in page table

It is given 16K frames = 2^{14} frames

= 14 bits

So 26 bits will be the page/frame offset.

Virtual address = No. of pages + Page offset bits

= 42 bits

Q13 **Consider a system with 2-level paging and a TLB with a hit rate of 80%, given TLB access time is 2ns. Find effective memory access time (in ns) if the data cache hit rate is 75% and cache access time is 1 ns, and main memory access time is 100 ns. (Rounded off to nearest integer)**

Sol: Range: 67-67

TLB hit rate= 0.80, TLB miss rate = 0.20, TLB access time = 2 ns

Cache hit rate = 0.75, Cache miss rate = 0.25, Cache access time = 1 ns

Accessing main memory needs 100 ns.

Effective memory access time
= .80 [2 + .75 (1) + 0.25 (1+100)] + 0.20 [2 + 200 + 0.80(1) + 0.20 (1+100)]
= .80 [2 + 0.75 + 0.25 (101)] + 0.20 [2 + 200 + 0.80 + 0.20 (101)]
= 0.80 [28] + 44.6
= 67 ns

Q14 **Consider a system implementing virtual memory of address size 32–bits, having 30–bits of physical address, and a page size of 4 KB. Both physical address space and virtual address space are byte-addressable. The system uses an inverted page table, where each entry has a page number and process identifier (PID) of size 12 bits. Find the size of the inverted page table in Mega Bytes.**

Sol: Range: 1-1
Given,
Physical address Size: 30–bits
Virtual address size: 32–bits
Page size: 4 KB
Process ID size: 12 bits

Physical address space (PAS) = 2^{30} bytes

$$\text{Number of frames} = \frac{\text{PAS}}{\text{Page Size}} = \frac{2^{30}\text{ bytes}}{4\text{KB}}$$

$$= \frac{2^{30}\text{ Bytes}}{2^{12}\text{ Bytes}} = 2^{18}$$

Number of pages = LAS/page size

$$= \frac{2^{32}}{2^{12}} = 2^{20}$$

Number of bits to denote page number
= $\log_2 2^{20}$ = 20 bits
Page table entry size = Page number bits + PID bits
= 20 + 12 = 32 bits = 4 bytes
So, size of inverted page table = Number of frames × e
= $2^{18} \times 2^2$ bytes
= 2^{20} bytes = 1 MB

Q15 **Consider a byte addressable virtual memory system with 34-bits of virtual address. Each page table entry is 4 bytes long. What is the minimum page size (in kilo bytes) required for a three-level paging scheme? (Consider each page table to fit into exactly in one frame).**

Sol: Range: 1-1

Given:

Virtual address = 34 bits, page table entry, page table entry (PTE) = 4 B

The size of the virtual address space (VAS) = 2^{34} B

$$\text{Size of a page table} = \left(\frac{\text{VAS}}{\text{Page size}}\right) * \text{PTE}$$

Let the page size is 2n bytes.

So,

$$\text{Size of page table at first level of paging} = \left(\frac{2^{34}}{2^{n}}\right) * 2^{2}$$

$$\text{Size of page table at second level of paging} = \left(\frac{2^{34}}{2^{2n}}\right) * 4^{2}$$

$$\text{Size of page table at third level of paging} = \left(\frac{2^{34}}{2^{3n}}\right) * 4^{3}$$

Given, every page table should fit into one page,

$$2^{n} \geq \left(\frac{2^{34}}{2^{3n}}\right) * 4^{3}$$

$$\Rightarrow 2^{4n} \geq 2^{40}$$

$$\Rightarrow 4^{*n} \geq 40$$

$\Rightarrow n \geq 10$ i.e., n = 10 (for minimum page size)

Page size = 2^{n} bytes = 2^{10} bytes = 1 KB

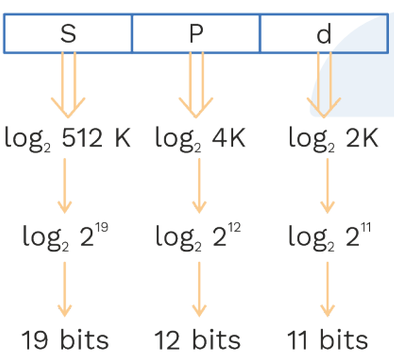
Q16 **Consider a system where the logical address space is divided into 512 K segments. Each segment is divided into 4K equal-sized pages. Each page stores 2 KB information. What are the storage capacities of logical and physical address spaces, respectively? (Assume byte–addressable memory and page table entry size is 2 B).**

a) 4 TB, 128 MB
b) 2 TB, 512 MB
c) 4 TB, 256 MB
d) 8 TB, 128 MB

Sol: **Option: a)**

Logical address structure :	S	P	d

S: The number of bits required to represent all segments of LAS.
P: The number of bits required to represent all pages in one segment.
d: The number of bits required to represent page size of the segment.



- Length of logical address = (19+12+11) bits = 42 bits
- Size of logical address space = 2^{42} B = 4 TB

Physical address structure:	F	d

F: The number of bits required to represent all frames uniquely in physical address space.
d: The frame size of physical address space.
Since, page table entry size is 2B = 16 bits.
- F = 16 bits as page table entry contains frame number.

In segmented paging:
Frame size = Page size of segment
d = 11 bits
The size of physical address = (16+11) bits = 27 bits
Size of physical address space = 2^{27} B = 128 MB

Previous Years' Question

Q: In a system with 32-bit virtual addresses and 1 KB page size, use of one–level page tables for virtual to physical address translation is not practical because of

a) The large amount of internal fragmentation
b) The large amount of external fragmentation
c) The large memory overhead in maintaining page tables
d) The large computation overhead in the translation process

Sol: c) **(GATE CSE-2003)**

Virtual memory:

Definition

Virtual memory is a technique that lets non-completely in-memory processes to be executed.

Several memory management solutions aim to keep many processes in memory at the same time to allow multiprogramming, but they require the full process to be in memory before it can run.

- Virtual memory alleviates programmers' concerns about memory and storage constraints.
- It enables processes to easily share files and implement shared memory.

There are numerous advantages to being able to run a program that is only partially in memory.

1) At any point in time, the size of a program can be larger than the available memory.

2) More programs could be run if each program required less physical memory runs at the same time, resulting in a rise in CPU utilisation and memory usage throughput.
3) Because less I/O is required to load or swap user programs into memory, each user program can be loaded or swapped faster. The user software would run more quickly.

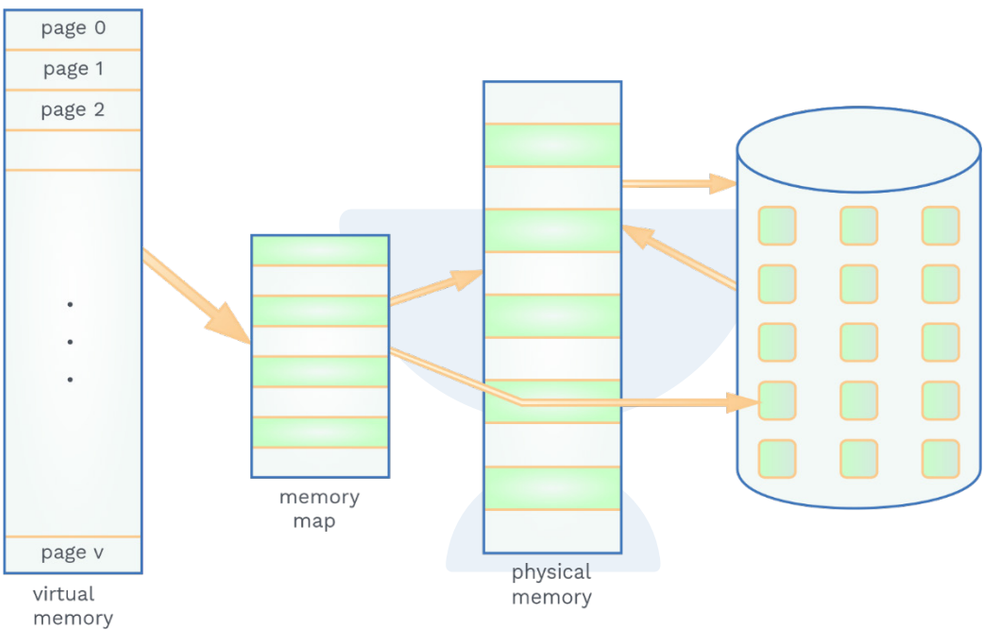


Fig. 5.14 Diagram Showing Virtual Memory that is Larger than Physical Memory

- Virtual memory, in addition to separating logical and physical memory, allows two or more processes to share files and memory via page sharing.

Advantages

- System libraries can be shared by many processes using a virtual address space mapping of the shared item.
- Virtual memory enables one process to generate a memory space that can be shared with another. Although the virtual address space of processes sharing this region is considered part of their virtual address space, the physical pages of memory is shared.
- Using the fork() system call, pages can be shared during process creation, speeding up the process creation process.

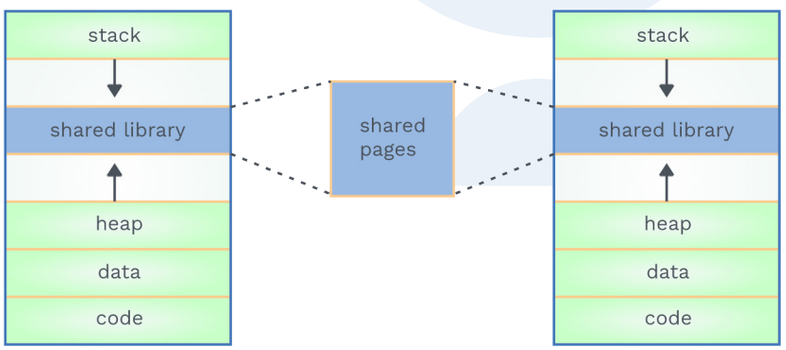


Fig. 5.15 Shared Library Using Virtual Memory

- Within a process, all memory references are logical addresses that are dynamically transformed into physical addresses at run time. This means that during the execution of a process, it can be swapped in and out of main memory, occupying different locations in main memory at different moments.
- A process can be divided down into multiple pieces, and these fragments do not need to be kept in the main memory throughout execution. This is possible because of a mix of dynamic run–time address translation and the use of a page or segment table.

Demand paging:

Demand paging is the process of loading a page into memory on demand (when a page fault occurs).

- However, determining the pages to be kept in the main memory and the pages to be kept in the secondary memory is quite a challenge since to predict when a process will require a specific page at a specific time is unknown.
- It recommends that all frame pages be kept in secondary memory until they are needed, and that no page be loaded into the main memory until it is needed.

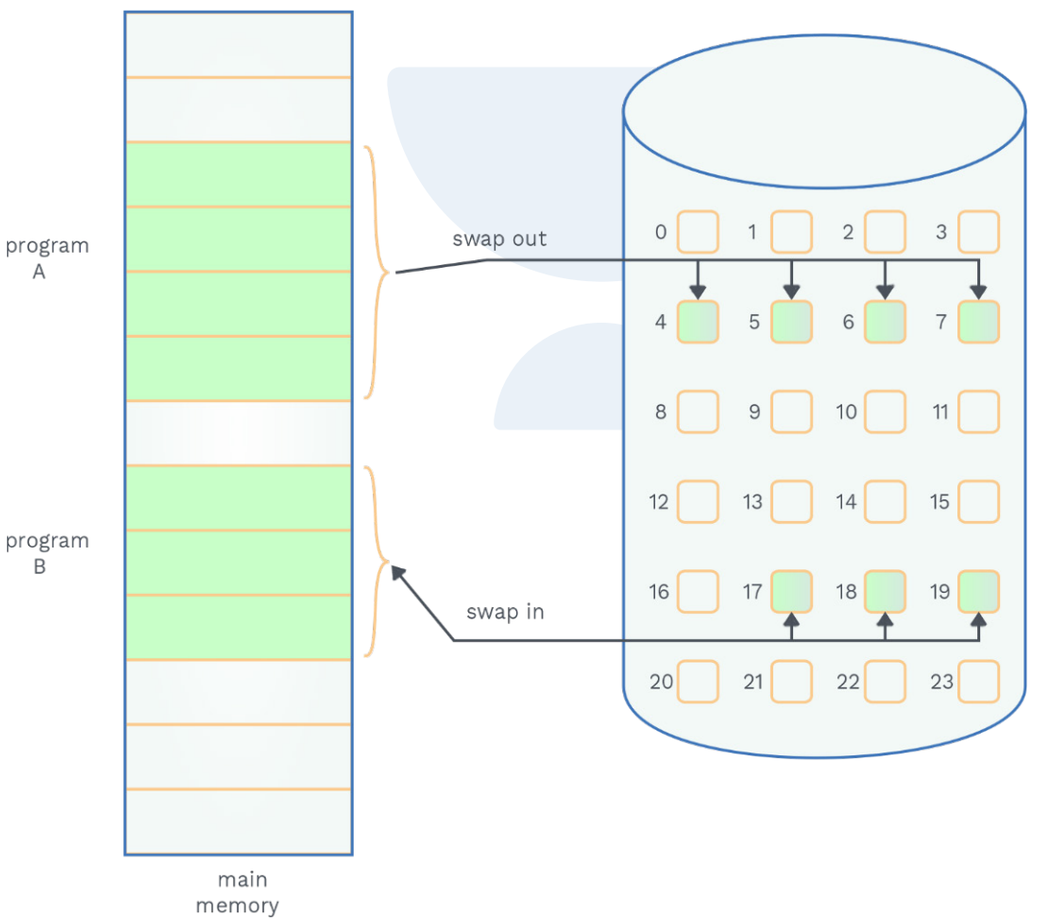


Fig. 5.16 Transfer of a Paged Memory to Contiguous Disk Space

- The operating system guesses which pages will be used when a process is swapped in before it is swapped out again. Instead of storing the entire process, the operating system saves only those pages.

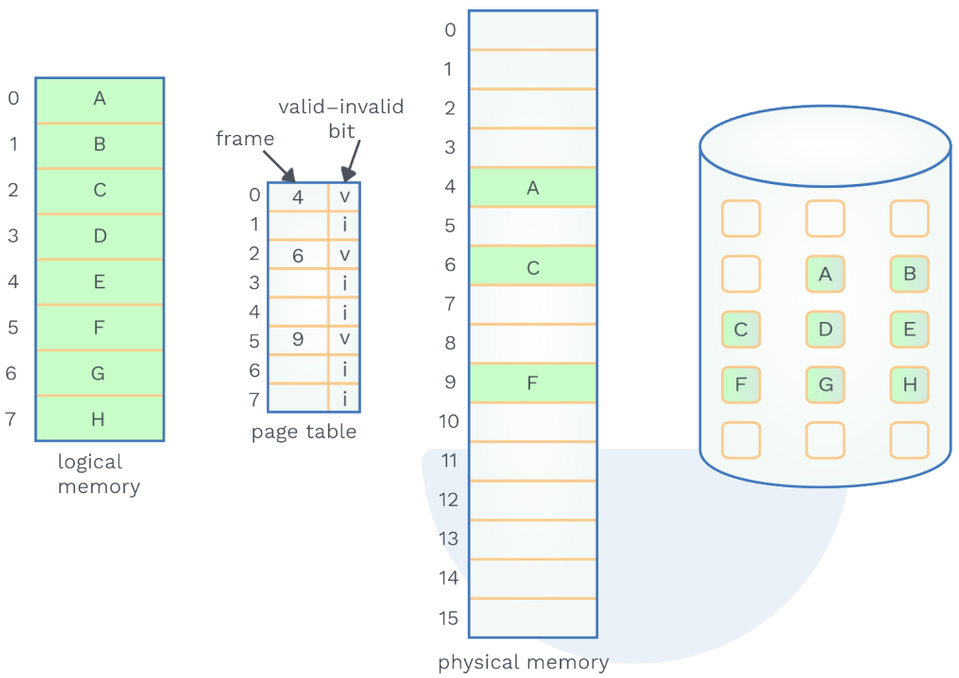


Fig. 5.17 Page Table When Some Pages are Not in Main Memory

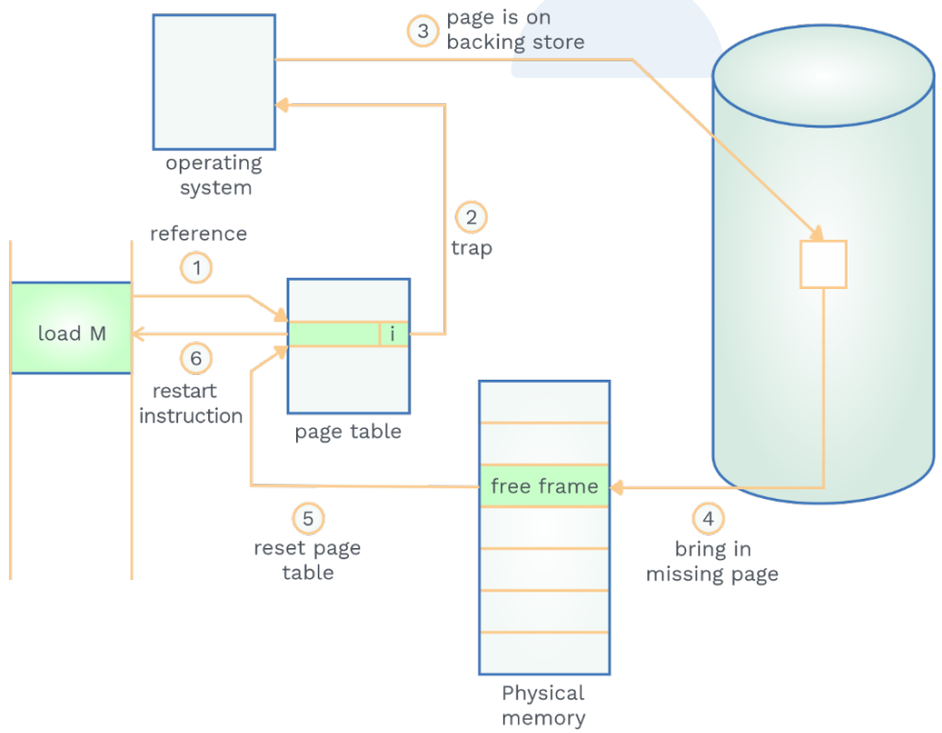


Fig. 5.18 Steps in Handling a Page Fault

- When a program accesses a page that hasn't been loaded into memory, it causes a page fault. While translating the address through the paging hardware.
- It will check the valid–invalid bit in the page table, and if the invalid bit is set, it will produce an error.
- Trap to the operating system, after which a page fault is handled.

The following is the technique for dealing with a page fault:

Technique for dealing with a page fault

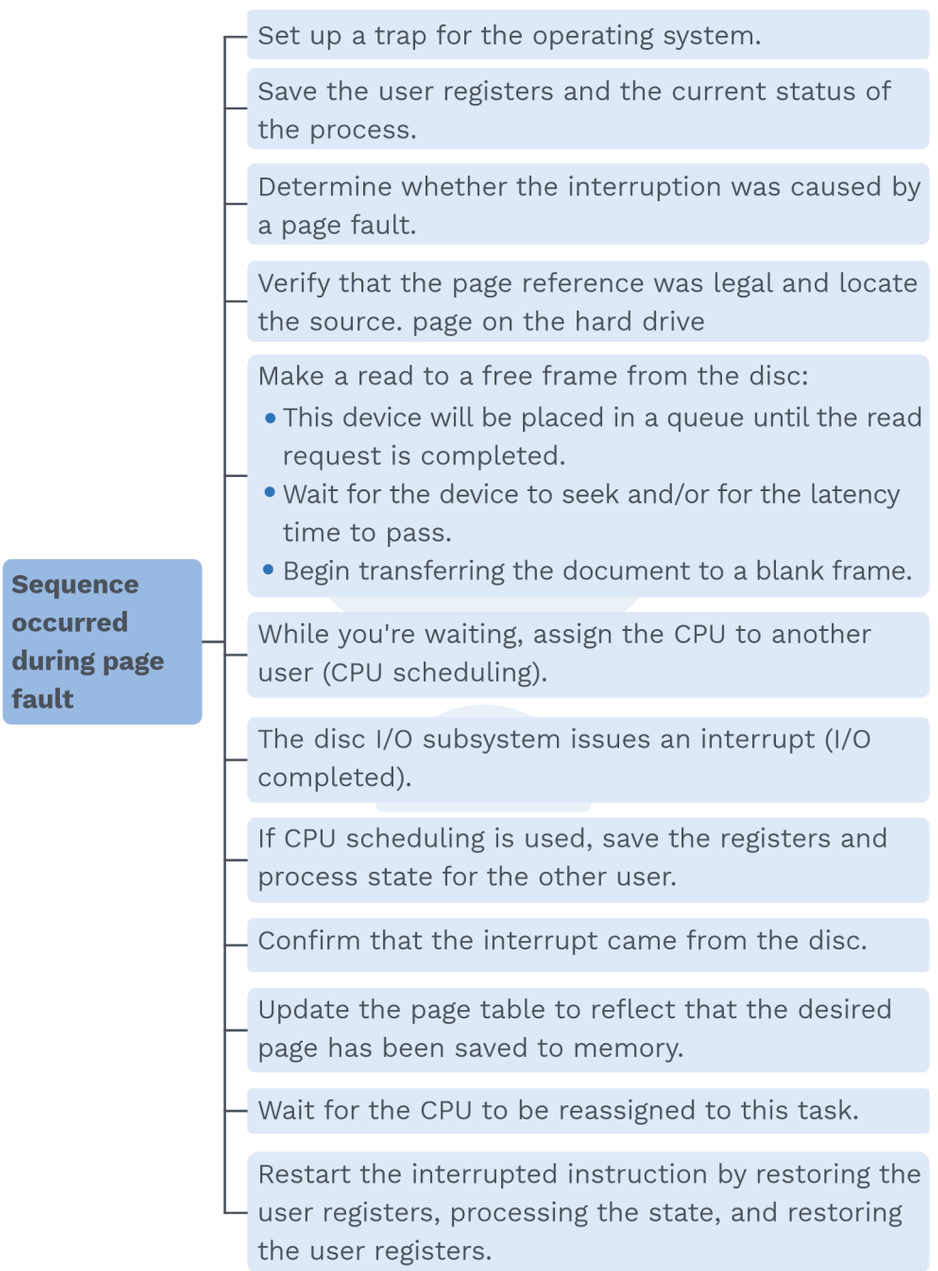
- It looks for this in an internal table (within the process control block). procedure for determining whether a reference was legitimate or not, access to memory
- We stop the process if the reference is invalid. If it was legitimate, but we hadn't yet imported that page, we did so now.
- From the list, we select a free frame.
- A disc operation is scheduled to read the desired page into the newly allocated frame.
- Once the disc read is complete, we update the process' internal table as well as the page table to reflect that the page has been loaded into memory.
- The instruction that was halted by the trap is restarted. The page can now be accessed as if it had always been in memory by the process.

Performance of demand paging:

p = Probability of page fault ($0 \le p \le 1$)

$$\text{EMAT} = (1-p)m + p \times (\text{page fault service time})$$

m = memory access time



- In every scenario, all of these actions are required. However, three components of the page fault service time slows down the system's overall performance.
  1) Service the page–interrupt the fault.
  2) Read the text on the page.
  3) Begin the process all over again.

SOLVED EXAMPLES

Q17 **Consider a system using demand paging architecture that takes 10 ms to serve a page fault the main memory access time is 3 ms. The maximum acceptable page fault rate (in percentage rounded off up to 2 decimal digits) to get the effective memory access not to be more than 3.70 ms is?**

Sol: Range: 10.00-10.00

Let P be the page fault rate.

EMAT $\geq$ P (page fault service time) + (1 – P) (m)

$$3.70 \geq P\,(10) + (1 - P) \times 3$$

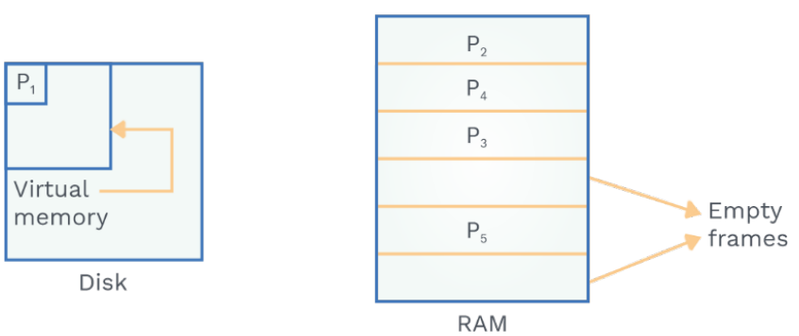
$$.70 \geq 7P$$

$$P = \frac{.7}{7} = 10\%$$

Page fault:

Two cases arises in paging when page fault occurs

Case 1: Empty frame available in physical memory.

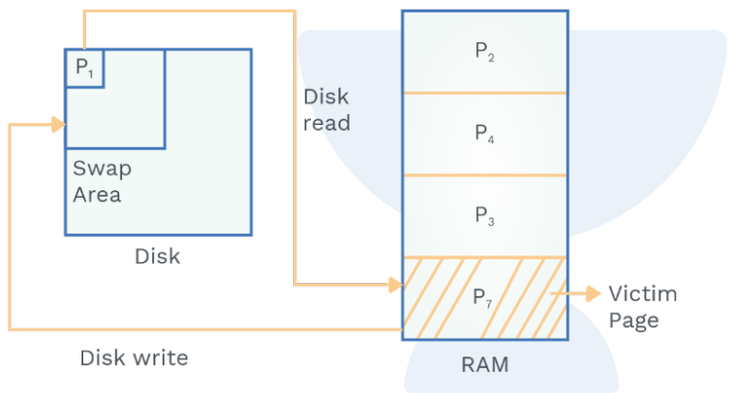


Here P_1 page will move from the swap area on the disk to one of the free frames, and the process will continue executing after page fault.

Case 2: No empty frame in physical memory

Here when a page fault occurs, and process requested page is brought into RAM from the swap area of the disk, there is not a single frame free to

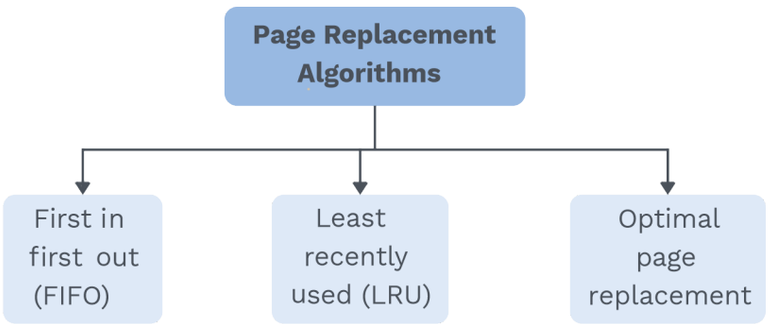
accommodate the page. So some page replacement algorithm is used that selects a victim page of some process that swaps out, and the desired page occupies its space. Victim pages are checked whether they are dirty pages or clean pages. A dirty page is a page when some modification of data has happened. Since it was last brought in memory. A clean page is when no data is altered on the page. A dirty page needs a write back to the disk when it becomes a victim page for some page replacement algorithm. It is a costly operation for the execution of the process as disk read/write are slower significantly in terms of physical memory read/write, whereas clean page don't need a write back to disk. So only disk reads happen in which desired page overwrites the victim page.



Grey Matter Alert!

- Whether the victim page is a dirty page or a clean page can be identified by dirty bit in page table of a process.
- If the dirty bit is set by hardware then it means the page is modified, and if it is replaced, it must be written back to the disk.
- Many systems keep a small list of clean pages that are available immediately for a replacement.

Page replacement algorithms:



1) First in first out (FIFO):

- This is the most basic page replacement method. The operating system maintains a queue for all pages in memory, with the oldest page at the front of the list to be replaced. When a page has to be replaced, the first page in the queue is chosen for replacement.
- On one side, the page replaced may be an initialization part that was utilised a long time ago and is no longer needed, while on the other hand, it may contain a widely used variable that was initialised early and is in constant use.

SOLVED EXAMPLES

Q18 **Consider the page reference strings 1, 7, 0, 7, 5, 6, 3 and the process's allocation of three - page frames. Determine the total number of page errors. Assume you're using the FIFO page replacement technique.**

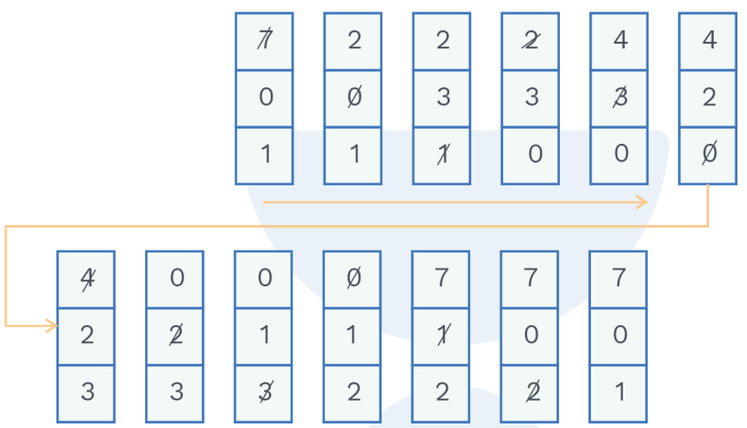
Sol: Range: 6–6

1	7	0	7	5	6	3
		0	0	0	0	3
	7	7	7	7	6	6
1	1	1	1	5	5	5
Miss	Miss	Miss	Hit	Miss	Miss	Miss

Total page fault = 6

Q19 **Given a page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1. Determine the total number of page errors, when the system allocates three frames to the process and employs the FIFO page replacement technique to replace the pages.**

Sol: **Range: 15-15**



so total fault in string are

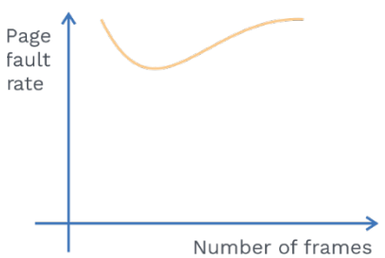
7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

Pages with square box were page fault so total number of page faults are 15.

Note:

If we increase the number of pages frame from 3 to 4. Then the total number of page fault will be 10.

Increasing the number of page frames can sometimes result in an increase in the number of page faults.



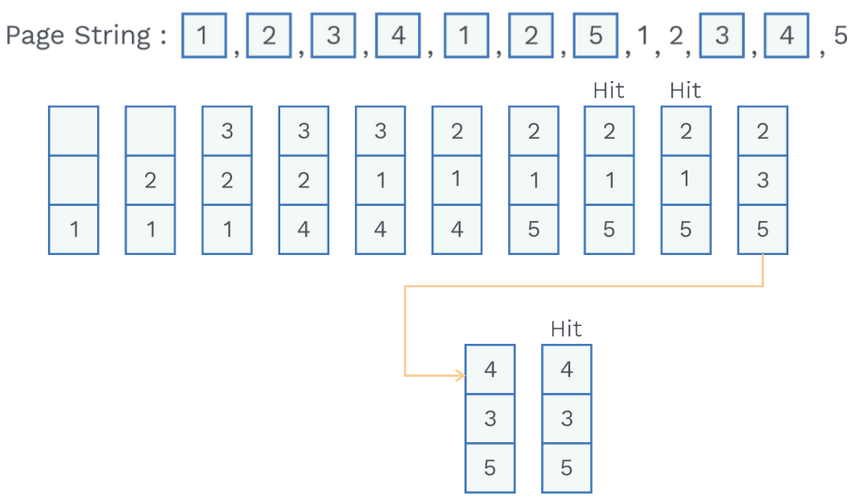
Belady's anomaly:
The phenomenon known as Belady's anomaly, it occurs when the number of page frames for a given memory access pattern increases, resulting in an increase in the frequency of page faults. Belady's anomaly affects FIFO policies and other non-stack based algorithms.

Why does this happen?

- The other two algorithms, Optimal and LRU, are widely utilised, but they never suffer from Belady's anomaly since they are stack-based.
- A stack-based algorithm is one that can display a set of pages in a stack.
- Memory for N frames is always a subset of the total number of pages in the in the total number of N + 1 frames of memory.
- The set of pages in RAM in an LRU policy would be the n most recently referenced pages. If the number of frames grows, these n pages will continue to be the most recently referenced and will remain in memory.
- So, only the FIFO page replacement policy suffers from Belady's anomaly.

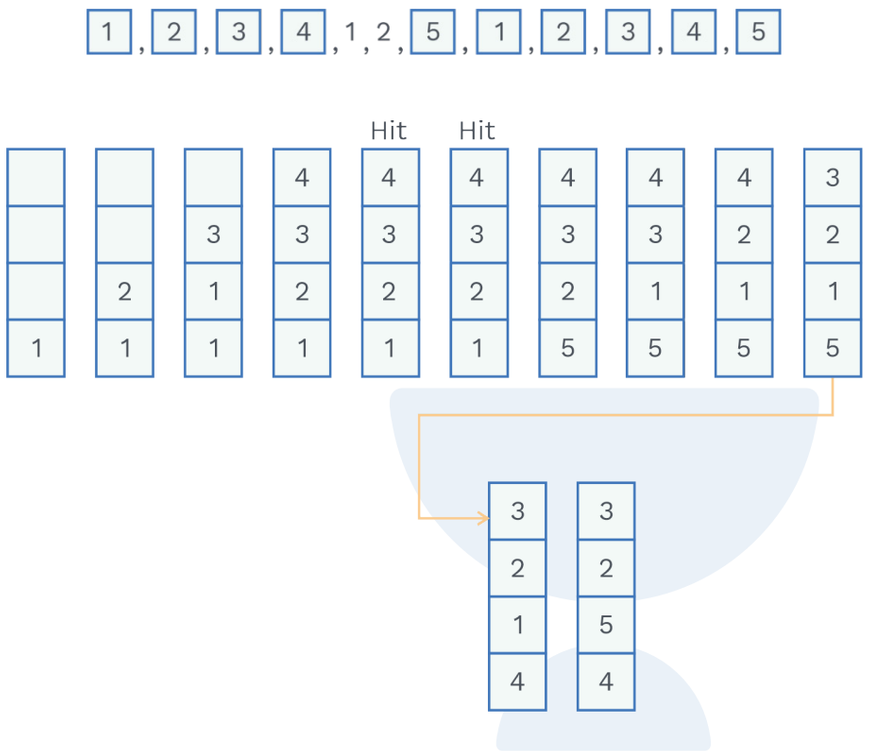
Example:
Consider the following page reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5. When the number of page frames assigned is 3 and 4, calculate the number of page faults.

Sol:
Case 1: M = 3 (Page frames)
First, let the page frame number be 3.



Number of page faults = 9.

Case 2: M = 4 (Page frames)



The total number of the page faults = 10.

Hence, for the given reference string number of page frames are four but the number of page faults is 10.

2) Least recently used (LRU):

In the LRU page replacement policy, if all the frames are in use and a page is asked by the executing process, than the frame to be chosen for replacement is the one which has not been accessed least recently. The LRU page replacement policy is effective. The main issue is figuring out how to replace LRU frames. An LRU page–replacement method needs hardware support.

Implementation:

A stack of page numbers is kept here. When a page is referenced, it is moved to the top of the stack from the bottom. As a result, the most recently used page always appears at the top of the stack, during the least recently used page appears at the bottom.

It's better to use a double linked list with a head and tail pointer to implement the aforesaid strategy. It is an expensive process to remove a page and place it on top of the stack; the tail pointer refers to the bottom of the stack, which is the LRU page.
LRU replacement, unlike FIFO, is not affected by Belady's anomaly.

SOLVED EXAMPLES

Q20 **Given a page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3 for a process. The system allocates four frames to the process. Find number of page faults if LRU policy is used.**

Sol: **Range: 6-6**
Page string: 7 0 1 2 0 3 0 4 2 3 0 3 2 3

			2	2	2	2	2	2
		1	1	1	1	1	4	4
	0	0	0	0	0	0	0	0
7	7	7	7	7	3	3	3	3
Miss	Miss	Miss	Miss	Hit	Miss	Hit	Miss	Hit

Total page fault = 6

3) Optimal page replacement (OPT):
The optimal page replacement policy has the lowest page fault rate among all the page replacement algorithms. Belady's anomaly is never a problem for it.
Pages that will not be used for the greatest period of time in the future are chosen for replacement using this technique.
Example:
Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, with four-page frames. Determine the total number of page errors. The best policy is used.

Sol:

7	0	1	2	0	3	0	4	2	3	0	3	2

			2	2	2	2	2	2
		1	1	1	1	1	4	4
	0	0	0	0	0	0	0	0
7	7	7	7	7	3	3	3	3
Miss	Miss	Miss	Miss	Hit	Miss	Hit	Miss	Hit

Total page faults = 6

SOLVED EXAMPLES

Q21 **Consider a system which implements an optimal page replacement policy for page replacement.**
Given a page reference string 2, 5, 7, 4, 5, 7, 3, 4, 6, 3, 4, 2, and three frames empty initially. Find the number of page faults.

Sol: **Range: 7-7**

In optimal page replacement policy, when a page fault occurs, the page which is going to be used last in the given page reference string is chosen for replacement.

(2)	(5)	(7)	(4)	5	7	(3)	4	(6)	3	4	(2)
		7	7	7	7	3	3	3	3	3	3
	5	5	5	5	5	5	5	6	6	6	6
2	2	2	4	4	4	4	4	4	4	4	2

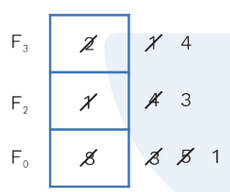
Page number encircled or resulted in page faults, so the total number of page faults is 7.

Q22 **For the reference strings 8, 1, 2, 3, 1, 4, 1, 5, 3, 4, 1, 4, 3, find the sum of page faults that occurred using FIFO, optimal and least recently page replacement algorithms with 3 -page frames. Given initially all frames are empty.**

a) 26
b) 27
c) 28
d) 32

Sol: **Option: a)**

FIFO :

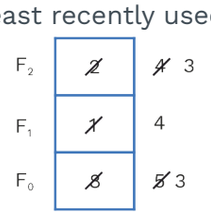


Total number of page fault
= 3 + 1 + 1 + 1 + 3 + 1 = 10

Optimal page replacement algorithm:

Total number of page fault
= 3 + 1 + 1 + 1 + 1 = 7

Least recently used:



Total number of page fault
= 3 + 1 + 1 + 1 + 1 + 1 + 1 = 9

The sum of page faults occurred using FIFO, optimal and least recently page replacement algorithms with 3 page frames = 10 + 7 + 9 = 26

Q23 **Which of the following is/are TRUE with respect to LRU page replacement strategy?**

a) LRU is useful for processes that exhibit spatial locality.

b) LRU incurs the overhead of maintaining an ordered list of pages and reordering that list.

c) LRU is useful for processes that exhibit temporal locality.

d) LRU incurs the overhead of maintaining a list of pages, and several ordered subsets of the list.

Sol: **Options: b), c)**

a) FALSE, LRU is useful for processes exhibiting temporal locality.

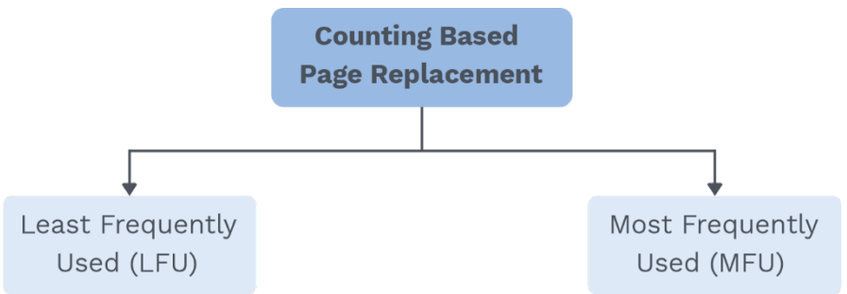
b) TRUE, LRU incurs the overhead of maintaining an ordered list of pages and reordering that list.

c) TRUE, LRU is useful for processes that exhibit temporal locality.

d) FALSE

Counting based page replacement:

Apart from FIFO, LRU, OPT page replacement algorithms, there are few more algorithms exists that can be used.



1) Least frequently used (LFU):

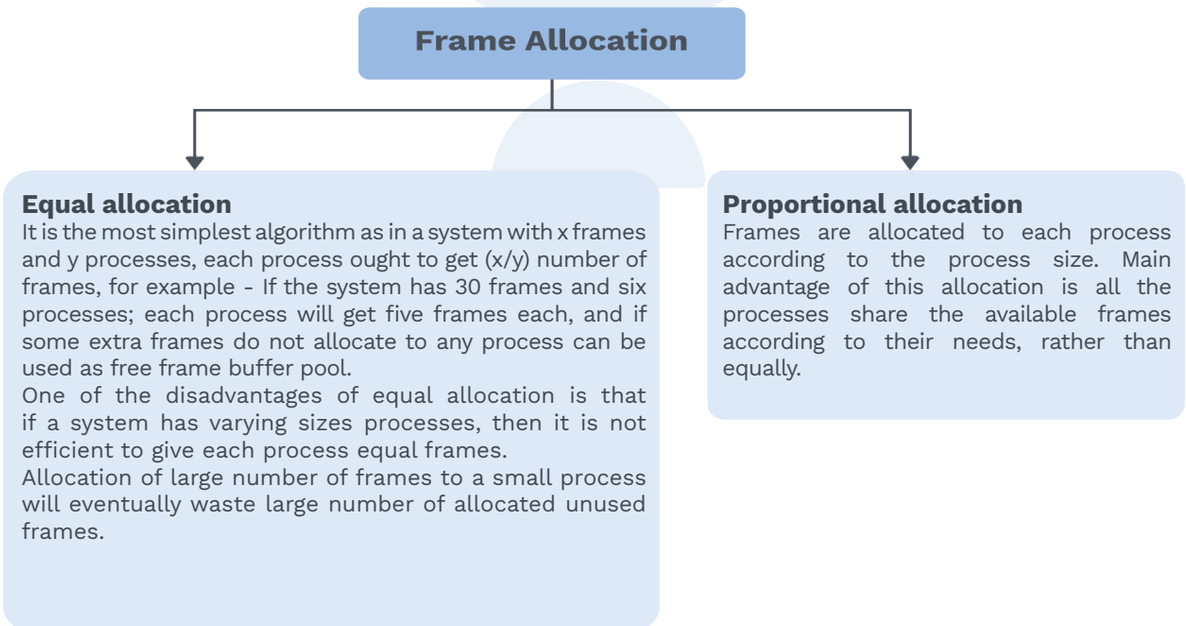
- It is necessary to replace the page with the lowest count. This choice is that a frequently visited page should have a high number of references.
- The issue arises when a page is frequently used during the first part of a procedure but is seldom used again.
- If it was widely used, it will have a large count and will remain in RAM even if it is no longer needed.

2) **Most frequently used (MFU):**
It's the page replacement method, which assumes that the page with the lowest count was probably freshly added and hasn't been used yet. It's the polar opposite of the least-often-used page replacement algorithm.

Allocation of frames:
Page replacement techniques and frame allocation algorithms are required when using demand paging. If you have numerous processes in physical memory, frame allocation methods are employed in the system to get the number of page frames that need to be allotted to each process.
Various challenges that the solutions for frame allocation encounter. You can't use more than the entire amount of frames available.

- Each process should be given at least a certain amount of frames.

Frame allocation algorithms:



Thrashing:
We must suspend a process execution set if the number of frames allocated to it falls below the minimum number required by the computer

architecture, as it will page fault fast. At this stage, it must take the place of one or more pages.
As a result, page faults recur in the same manner. Swap–in and Swap–out of pages will enhance disc activity while decreasing CPU usage. This is referred to as thrashing.

Cause for thrashing:

- The act of thrashing has a harmful effect on the system's performance. When CPU utilisation is low; the OS increases the degree of multiprogramming and employs a global page replacement algorithm that replaces pages independent of their order.
- To change pages in and out, these faulting processes must employ the paging device.
- CPU utilisation will be decreased when a lot of pages are queued up.
- The degree of multiprogramming will be increased. It indicates that more new processes will try to start by stealing resources from existing ones. Resulting in more page faults and a longer paging device queue.
- As a result of the foregoing scenario, system throughput plummets, and the fault rate skyrockets, and effective memory access time skyrockets while no work is being done.

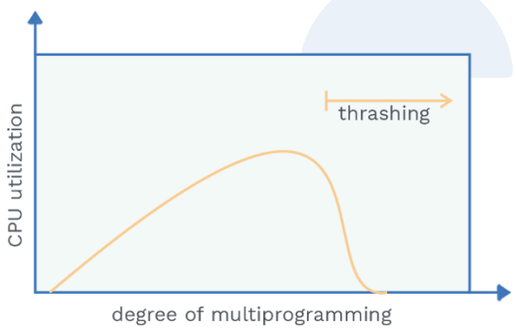
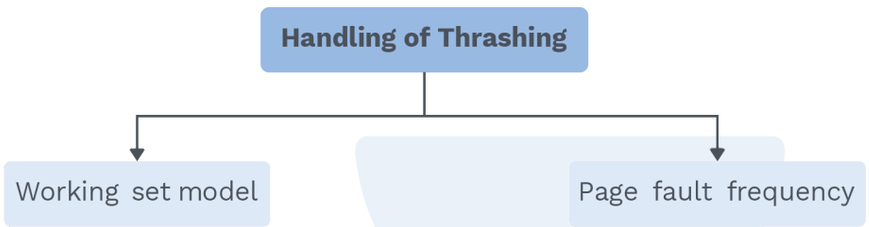


Fig. 5.19 Thrashing

- We can reduce the consequences of thrashing by using the local replacement technique, but it won't fully eliminate it because processes will still queue for the paging device most of the time. We must provide a process with as many frames as it requires to avoid thrashing. There are numerous methods for determining how many frames an operation requires. Working set strategy, for example, begins with determining how many frames a process truly employs. This method establishes a process execution locality model.

- **Locality model:**

We can reduce the consequences of thrashing by using the local replacement technique, but it won't fully eliminate it because processes will still queue for the paging device most of the time. We must provide a process with as many frames as it requires to avoid thrashing. There are numerous methods for determining how many frames an operation requires. Working set strategy, for example, begins with determining how many frames a process truly employs. This method establishes a process execution locality model.



1) Working set model:

- The basic idea of is that the process will fail only when it'll move to new location when enough number of frames are provided to store its current location locality.
- However, if the assigned frames are smaller than the current frame size, the process is doomed to thrash due to its proximity.
- The most frequently used pages would always become part of the working set.
- The working set's accuracy is determined by the size of parameter. If is too large, the working set may overlap, and if it is too small, the locality may not be completely covered.

Page reference table:

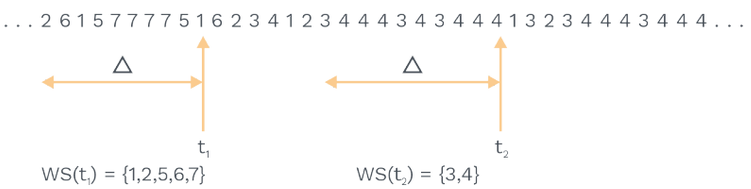


Fig. 5.20: Working Set Model

- Keeping track of the working set is a challenge with the working set model.
- The working set window moves about; a new reference appears at one end of each moving reference, while the oldest one fades away.

2) Page fault frequency:

- It's a more straightforward approach to dealing with thrashing.
- It reduces page fault, even if and page fault rate is high, it consider the process as failed.
- There aren't enough frames. Because a low rate of page faults indicates when there are two or more frames in a process, a precise upper and a lower limit should be set.
- Set the page fault that you want.

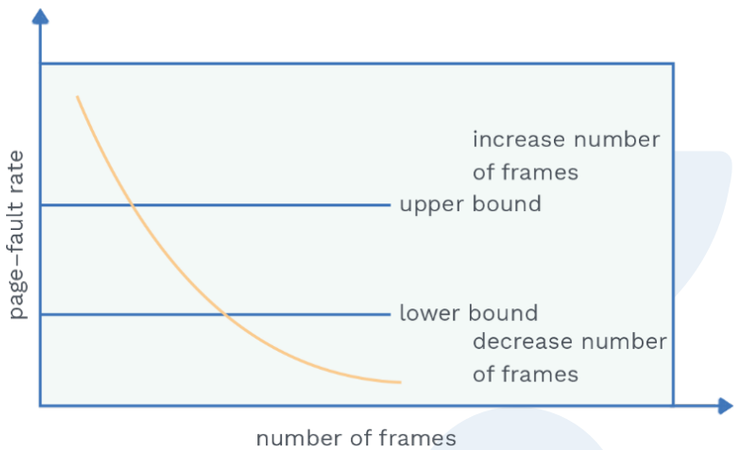


Fig. 5.21 Page–Fault Frequency

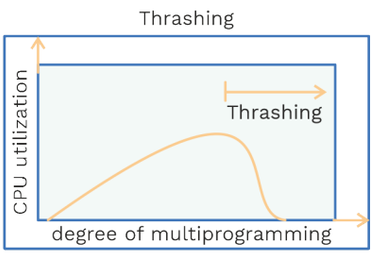
- If, in any case page fault rate < lower limit, then frames will be deleted, otherwise the process will be provided many numbers of frames.
- If there are no free frames and the page fault is high, some processes can be halted, and frames allotted to them can be reallocated.

SOLVED EXAMPLES

Q24 **Which of the following is TRUE about Thrashing?**
a) Thrashing always decreases degree of multiprogramming
b) Thrashing reduces page I/O
c) Thrashing implies excessive page I/O
d) None

Sol: **Option: c)**

In thrashing, maximum CPU time is wasted in I/O and page faults.



If degree of multiprogramming (DOM) gets increased beyond a certain limit, thrashing occurs. Thrashing doesn't decrease the DOM.

Q25 **Which of the following options is/are true with respect to thrashing?**
a) Thrashing decreases the CPU utilisation
b) Thrashing is about spending more time in paging than executing
c) The working set model in memory management is used to implement the concept of thrashing
d) Thrashing leads to high page fault rate

Sol: **Options: a), b), d)**
c) False. The working set model in memory management is used to implement the concept of the "principle of locality".
All other options are true.

IMPORTANT FORMULAE

Organization of the (LAS) logical address space:

If address size of LAS is n bits, and word size is of 1 byte

LAS = 2^n bytes

Logical address (LA) = $\log_2$ (LAS) bits = $\log_2$ (2^n) bits = n bits

Given, the size of each page is K bytes

$$\text{Number of pages in LAS} = \frac{\text{LAS}}{\text{Page size}}$$

Given, the size of each entry of page table as e bytes

$$\text{Then, the size of the Page Table} = \frac{\text{LAS}}{\text{Page size}} * \text{e Bytes}$$

Optimal page size to minimize page table size and internal fragmentation is given as

$$\sqrt{2 * \text{LAS} * \text{e}}$$

Organization of physical address space (PAS):

If address size of PAS is n bits, and word size is of 1 byte

PAS = 2^n bytes

Physical address (PA) = $\log_2$ (PAS) bits = $\log_2(2^n)$ bits = n bits

Given, frame size = K bytes

$$\text{Number of frames in PAS} = \frac{\text{PAS}}{\text{Frame size}}$$

Performance of multi–level paging:

Given main memory access takes m ns.

The average memory access time with n–level paging = (n+1) m ns

Performance of multilevel paging with TLB:

Given, TLB hitting rate = x, Time to access TLB is c

Memory access time = m

Average memory access time with n–level paging and a TLB

= x (c + m) + (1 – x) (c + (n + 1) m)

Performance of segmentation:

Given, main memory access time be 'm'.

Effective memory access time of segmentation (EMAT) = 2m

x = TLB hit ratio

c = TLB access time

m = memory access time

The average memory access time using segmentation and TLB is given as,

x (c + m) + (1 – x) (c + 2m)

concept of the "principle of locality".
All other options are true.

Chapter Summary

- Basics of memory management: – **1)** Registers
  2) Cache
  3) RAM memory
  4) Secondary memory
- Address spaces: – **1)** Logical address space
  2) Physical address space
- Linking: – It is used to generate a single executable code which is obtained by combining the different object files and library functions generated by the compiler.
- Loading: – It is used for bringing the executable code from secondary memory and loading it into the main memory for its execution.
- Swapping: – Movement of process between physical and secondary memory.
- Contiguous memory allocation: – **1)** Fixed partitioning
  2) Variable partitioning
- Allocation policies: – **1)** Best fit
  2) First fit
  3) Worst fit
  4) Next fit
- Non contiguous memory allocation: – **1)** Paging
  2) Segmentation
  3) Segmented paging
  4) Paged segmentation
- Virtual memory: – **1)** Demand paging
  2) Page replacement algo
  - FIFO
  - LRU
  - OPT

- Frame allocation algorithms – **3)** Belady's anomaly
  1) Equal allocation
  2) Proportional allocation
  3) Local allocation
  4) Global allocation
- Thrashing – If number of frames allocated to the process are lesser than what process