Unit - 4

STORAGE MANAGEMENT

Memory Management: Main Memory – Swapping, Contiguous Memory Allocation, Paging, Structure of the Page Tables, Segmentation. Virtual Memory: Demand paging, Page Replacement, Allocation, Thrashing; Allocating Kernel Memory, OS Examples.

✞ Memory Management

Memory: Memory is a large array of words or bytes, each with its own address.

There are two types of memory,

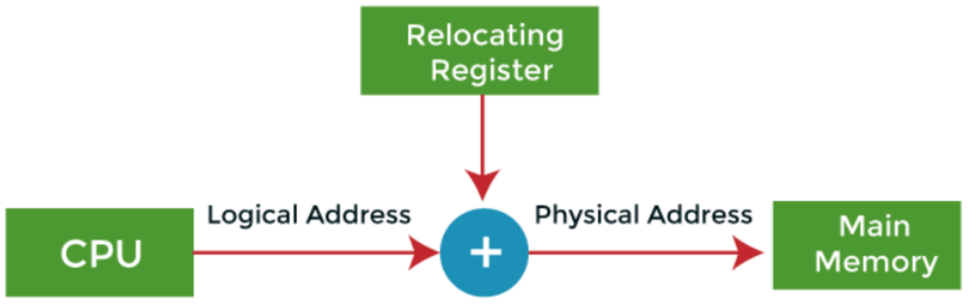
1. **Main Memory (RAM)**
2. **Secondary Memory (Hard Disk)**

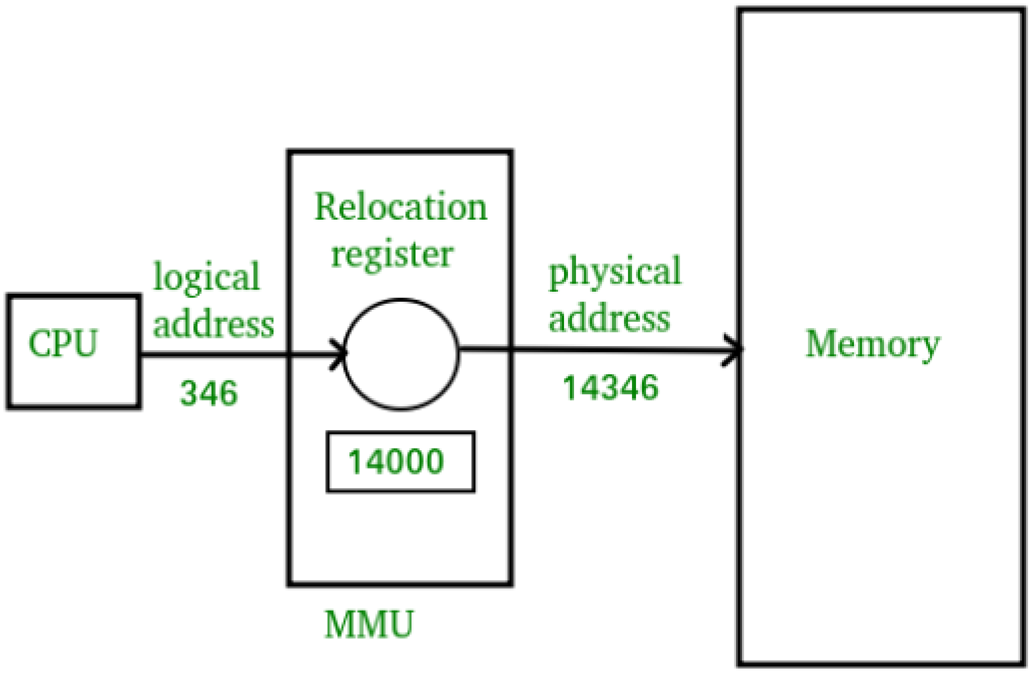
Main memory is central of all operation of computer system.
Main memory is volatile memory, which store data temporarily.

Memory Management:

Memory management is a service of O.S. which handles or manages primary(main) memory.
Memory management keeps track of each and every memory location either it is allocated to some process or it is free. It decides which process will get memory at what time and how much.
It tracks whenever some memory gets free or unallocated and correspondingly it updates the status.
Memory management provides protection by using two register, a base register and limit register.
The base register holds the smallest legal physical memory address and the limit register specifies the size of the range.
For example, if the base register holds 300000 and the limit register is 120000, then the program can legally access all addresses from 300000 through 411999.

- **Logical Address and Physical Address:**





An address generated by the CPU is a logical address whereas address actually available on memory unit is a physical address. Logical address is also known a Virtual address.

Virtual and physical addresses are the same in compile-time and load-time address-binding schemes. Virtual and physical addresses differ in execution-time address-binding scheme.

The set of all logical addresses generated by a program is referred to as a logical address space. The set of all physical addresses corresponding to these logical addresses is referred to as a physical address space.

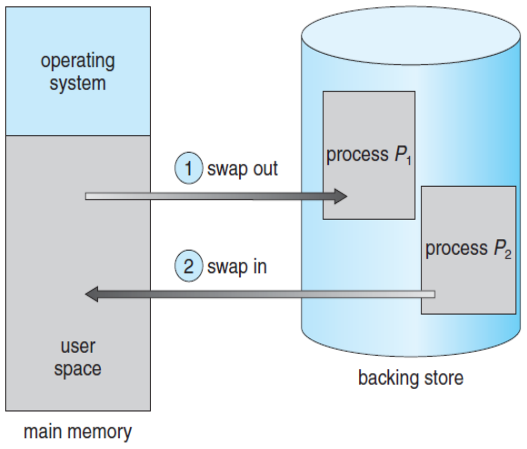
The run-time mapping from virtual to physical address is done by the memory management unit (MMU) which is a hardware device. MMU uses following mechanism to convert virtual address to physical address.

The value in the base register is added to every address generated by a user process which is treated as offset at the time it is sent to memory. For example, if the base register value is 10000, then an attempt by the user to use address location 100 will be dynamically reallocated to location 10100.

The user program deals with virtual addresses; it never sees the real physical addresses.

✞ Swapping

- A process must be in memory to be executed. A process, however, can be swapped temporarily out of memory to a backing store (disk) and then brought back into memory for continued execution.



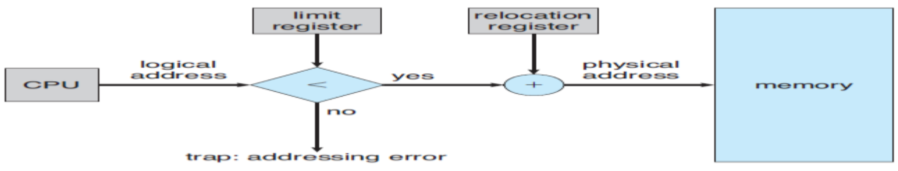
Contiguous Memory Allocation

- The memory is usually divided into two partitions:
  - one for the resident OS
  - one for the user processes.
- We can place the OS in either low memory or high memory (depends on the location of the interrupt vector).
- We usually want several user processes to reside in memory at the same time. We therefore need to consider how to allocate available memory to the processes that are in the input queue waiting to be brought into memory.
- In this contiguous memory allocation, each process is contained in a single contiguous section of memory.

Memory Mapping and Protection

- With relocation and limit registers, each logical address must be less than the limit register;
- The MMU maps the logical address dynamically by adding the value in the relocation register. This mapped address is sent to memory.

Hardware support for relocation and limit registers.



- When the CPU scheduler selects a process for execution, the dispatcher loads the relocation and limit registers with the correct values as part of the context switch.

- The relocation-register scheme provides an effective way to allow the OS size to change dynamically.

Memory Allocation

- One of the simplest methods for allocating memory is to divide memory into several fixed-sized partitions. Each partition may contain exactly one process.
- Thus, the degree of multiprogramming is bound by the number of partitions.

Multiple-partition method

  - When a partition is free, a process is selected from the input queue and is loaded into the free partition.
  - When the process terminates, the partition becomes available for another process.
  - This method is no longer in use.
- The method described next is a generalization of the fixed-partition scheme (called MVT); it is used primarily in batch environments.

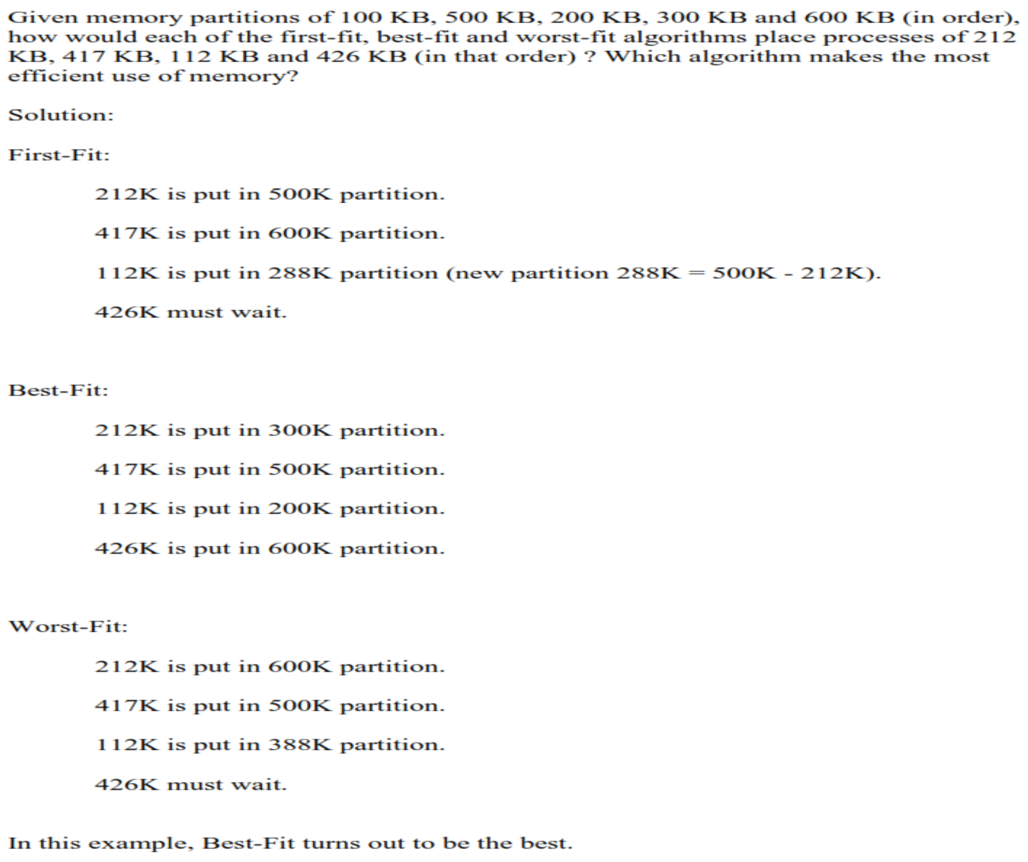
Fixed-partition scheme

  - The OS keeps a table indicating which parts of memory are available and which are occupied.
  - Initially, all memory is available for user processes and is considered one large block of available memory, a **hole.**
  - When a process arrives and needs memory, we search for a hole large enough for this process.
  - If we find one, we allocate only as much memory as is needed, keeping the rest available to satisfy future requests.
- At any given time, we have a list of available block sizes and the input queue. The OS can order the input queue according to a scheduling algorithm.
- When a process terminates, it releases its block of memory, which is then placed back in the set of holes. If the new hole is adjacent to other holes, these adjacent holes are merged to form one larger hole.
- This procedure is a particular instance of the general **dynamic storage-allocation problem**, which concerns how to satisfy a request of size from a list of free holes. There are many solutions to this problem.

  - **First fit**. **Allocate the first hole that is big enough**. Searching can start either at the beginning of the set of holes or where the previous first-fit search ended. We can stop searching as soon as we find a free hole that is large enough.

  - **Best fit**. **Allocate the smallest hole that is big enough**. We must search the entire list, unless the list is ordered by size. This strategy produces the smallest leftover hole.

  - **Worst fit**. **Allocate the largest hole**. Again, we must search the entire list, unless it is sorted by size. This strategy produces the largest leftover hole, which may be more useful than the smaller leftover hole from a best-fit approach.


Given memory partitions of 100 KB, 500 KB, 200 KB, 300 KB and 600 KB (in order), how would each of the first-fit, best-fit and worst-fit algorithms place processes of 212 KB, 417 KB, 112 KB and 426 KB (in that order) ? Which algorithm makes the most efficient use of memory?

Solution:

First-Fit:

212K is put in 500K partition.

417K is put in 600K partition.

112K is put in 288K partition (new partition 288K = 500K - 212K).

426K must wait.

Best-Fit:

212K is put in 300K partition.

417K is put in 500K partition.

112K is put in 200K partition.

426K is put in 600K partition.

Worst-Fit:

212K is put in 600K partition.

417K is put in 500K partition.

112K is put in 388K partition.

426K must wait.

In this example, Best-Fit turns out to be the best.

Fragmentation

- Both the first-fit and best-fit strategies for memory allocation suffer from external fragmentation.
- **External fragmentation exists when there is enough total memory space to satisfy a request, but the available spaces are not contiguous**; storage is fragmented into a large number of small holes.
- **Internal Fragmentation Memory block assigned to process is bigger than requested.** The difference between these two numbers is internal fragmentation; memory that is internal to a partition but is not being used.
- The general approach to avoiding this problem is to break the physical memory into fixed-sized blocks and allocate memory in units based on block size.

• **One solution to the problem of external fragmentation is compaction.** The goal is to shuffle the memory contents so as to place all free memory together in one large block.

•The simplest compaction algorithm is to move all processes toward one end of memory; all holes move in the other direction, producing one large hole of available memory. This scheme can be expensive.

•Another possible solution to the external-fragmentation problem is to permit the logical address space of the processes to be non-contiguous, thus allowing a process to be allocated physical memory wherever the latter is available.

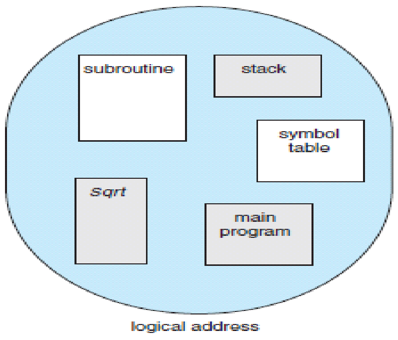
Segmentation

- An important aspect of memory management that became unavoidable with paging is the separation of the user's view of memory and the actual physical memory.
- The user's view of memory is not the same as the actual physical memory. The user's view is mapped onto physical memory.

Basic Method

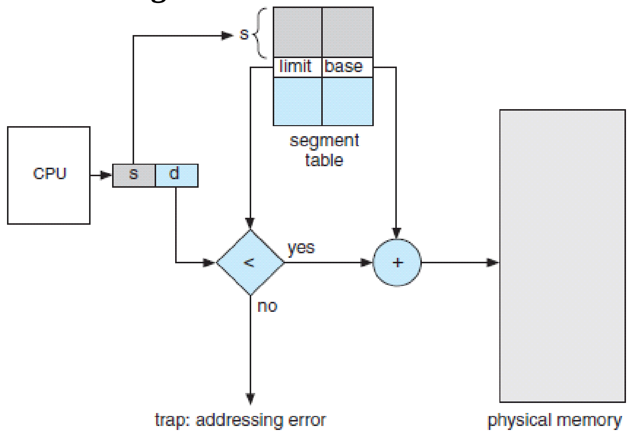
- Users prefer to view memory as a collection of variable-sized segments, with no necessary ordering among segments

Programmer's view of memory

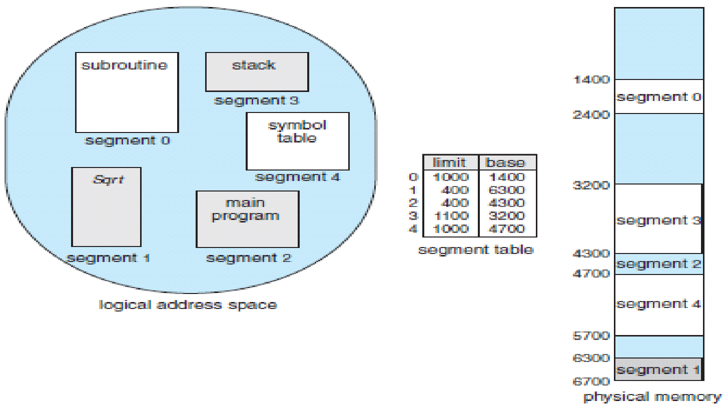


- Consider how you think of a program when you are writing it. You think of it as a main program with a set of methods, procedures, or functions.
- Segmentation is a memory-management scheme that supports this user view of memory. A logical address space is a collection of segments.
- Each segment has a name and a length. The addresses specify both the segment name and the offset within the segment.
- The user therefore specifies each address by two quantities:
    - a segment name
    - an offset
- For simplicity of implementation, segments are numbered and are referred to by a segment number, rather than by a segment name.
- Thus, a logical address consists of a two tuple: • <segment- number, offset>

Segmentation Hardware



Example of segmentation



	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

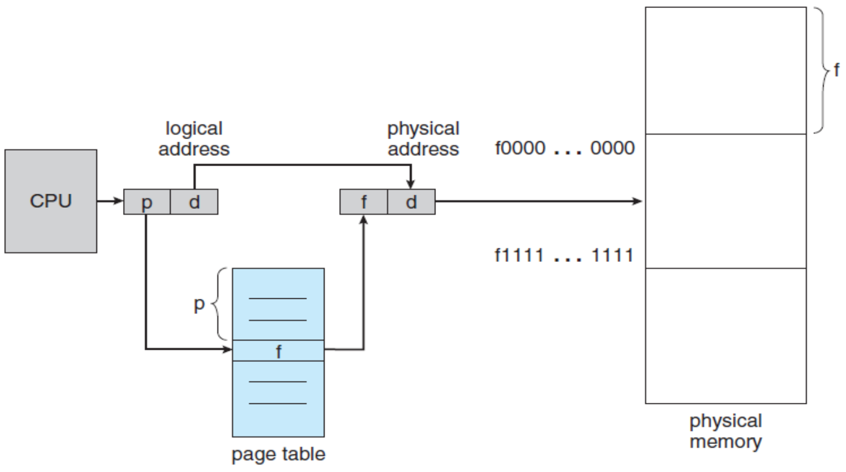
Paging

- Paging is a memory-management scheme that permits the physical address space of a process to be non-contiguous.

Basic Method

- The basic method for implementing paging involves
  - **breaking physical memory into fixed-sized blocks called frames**
  - **breaking logical memory into blocks of the same size called pages.**
- When a process is to be executed, its pages are loaded into any available memory frames from the backing store.
- The backing store is divided into fixed-sized blocks that are of the same size as the memory frames.

Paging Hardware

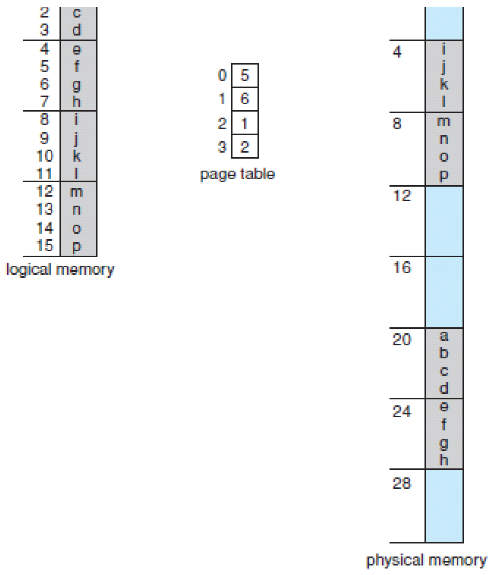


Address Translation Scheme

- Address generated by CPU is divided into:
  - **Page number (*p*)** – used as an index into a **page table** which contains **base address** of each page in physical memory
  - **Page offset (*d*)** – combined with base address to define the physical memory address that is sent to the memory unit

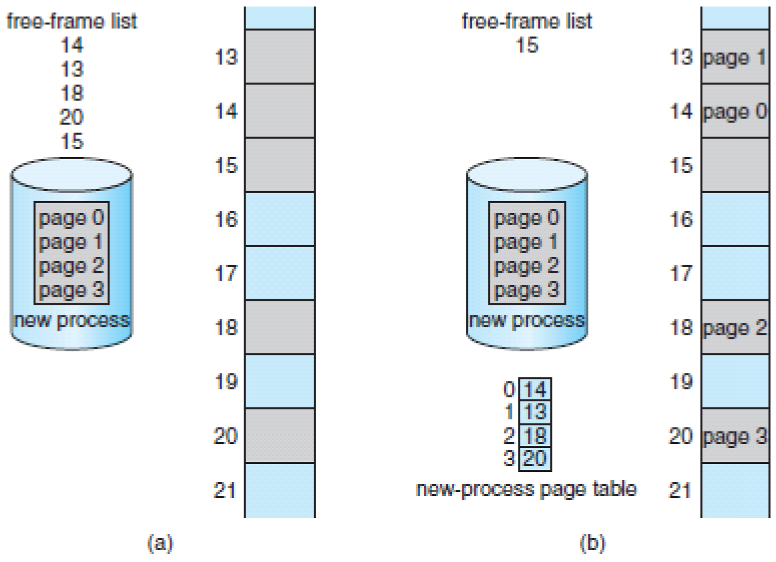
page number	page offset
p	d
m -n	n

  - For given logical address space 2^m and page size 2^n



Using a page size of 4 bytes and a physical memory of 32 bytes (8 pages). It is shown that how the user's view of memory can be mapped into physical memory.

- Logical address 0 is page O, offset O. Indexing into the page table, we find that page 0 is in frame 5. Thus, logical address 0 maps to physical address 20 (= (5 x 4) + 0).
- Logical address 3 (page 0, offset 3) maps to physical address 23 (= (5 x 4) + 3).
- Logical address 4 is page 1, offset 0; according to the page table, page 1 is mapped to frame 6. Thus, logical address 4 maps to physical address 24 (= (6 x 4) + 0).
- Logical address 13 maps to physical address 9.



Hardware Support

1. **Register Method :**

 In most operating system page table are implemented in a set of CPU registers. This method is very faster, but it support only in small page table.

2. **TLB (Translation look-aside buffer) (Associative Registers)**

 TLB is collection of associative registers. This is very high-speed cache memory.

 Each associative register store the page no and frame no.

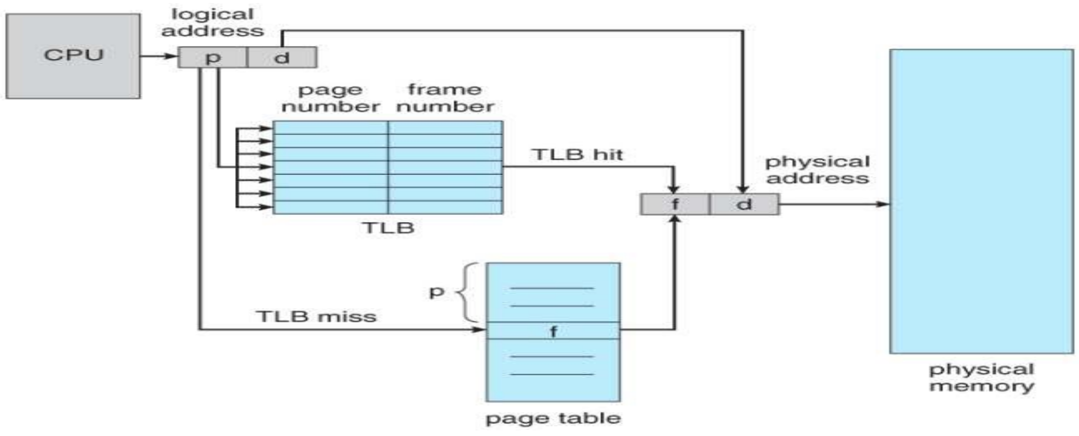
 Instead to check whole page table of process, TLB will search by page number, if page is found in TLB than frame No. of this frame will combine with displacement and physical address will created.

- If page is not found in TLB, then page table of process will search, and frame number will combine with displacement and physical address will be generated and also one entry will be created into TLB.
- If page found into TLB, is called TLB hit and if page is not found in TLB, it is called TLB miss.
- Hit Ratio: The percentage of time page is found in TLB is called Hit

ratio. Pages from TLB are removed by some page replacement algorithms.

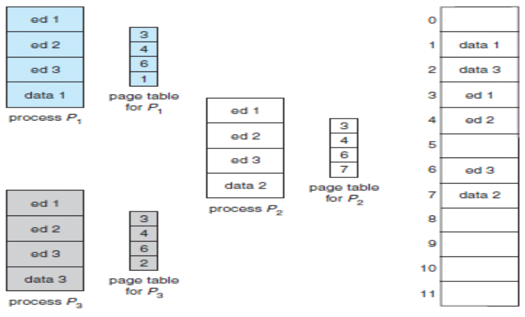
3.**Page table is kept in main memory**

- **Page-table base register** (**PTBR**) points to the page table
- **Page-table length register** (**PTLR**) indicates size of the page table
- In this scheme every data/instruction access requires two memory accesses
  - One for the page table and one for the data / instruction
    - **Following figure indicate the address mapping in TLB.**



Shared Pages

- An advantage of paging is the possibility of sharing common code. This consideration is particularly important in a time-sharing environment
  - **Reentrant code** is non-self-modifying code; it never changes during execution. Thus, two or more processes can execute the same code at the same time.



Sharing of code in a paging environment

o Each process has its own copy of registers and data storage to hold the data for the process's execution. The data for two different processes will, of course, be different.

o Only one copy of the editor need be kept in physical memory. Each user's page table maps onto the same physical copy of the editor, but data pages are mapped onto different frames.

Structure of the Page Table

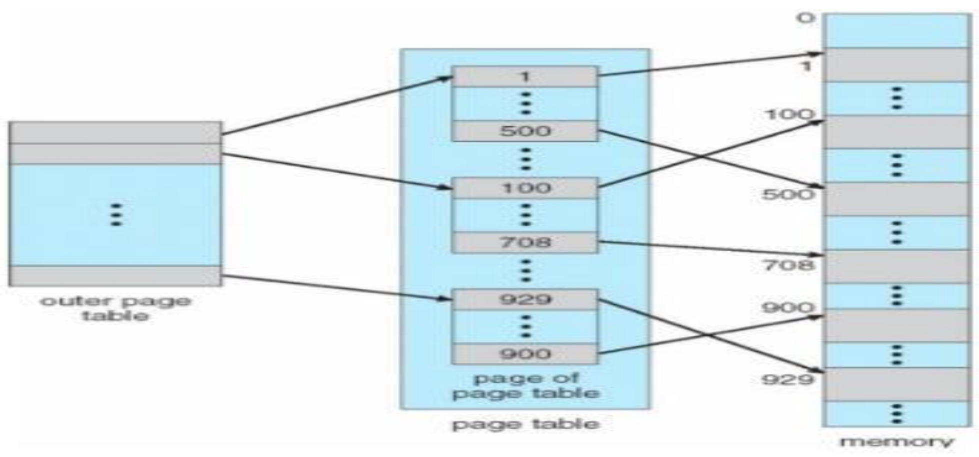
- Hierarchical Paging
- Hashed Page Tables
- Inverted Page Tables

Multilevel Paging(Hierarchical Paging)

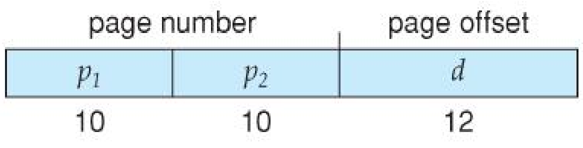
Some time for larger process, entries in page table are also increased. so, to overcome the problem of large page size, page table is divided into small piece and page table itself is also paged.

Here page table will divide into smaller page tables.

In following figure, two-page tables are there. One is outer page table which is index for inner page table.



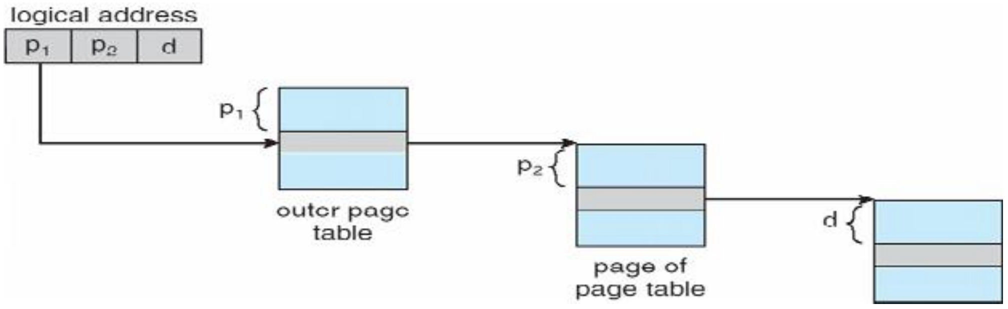
Here logical address consists of 3 parts.



Here logical address will be of is 32 bits . 12 bits for displacement.

where p1 is an index into the outer page table, and p2 is the displacement within the page of the outer page table.

Multilevel paging is reducing the memory overhead.



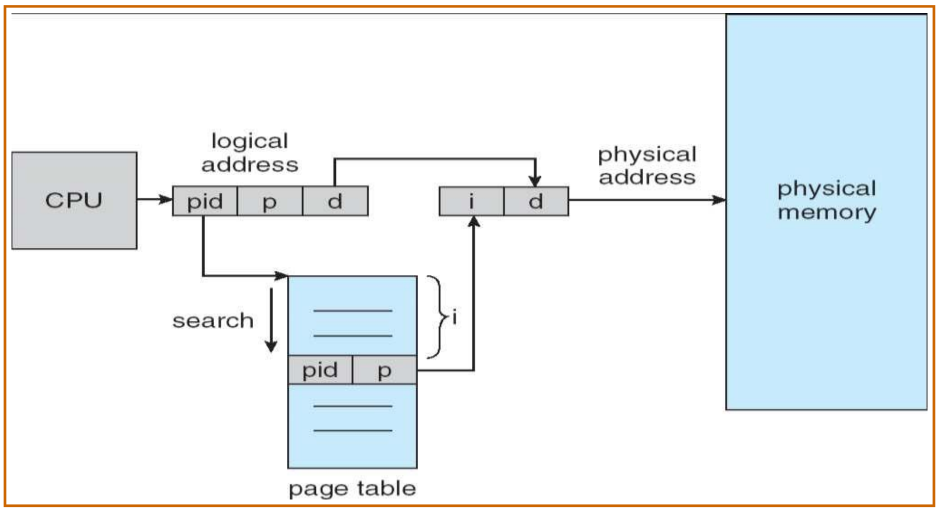
INVERTED PAGE TABLE (IPT):

Inverted page table is used to reduce the problem of memory overhead, when page table size is larger. Every process has a page table.

In inverted page table, **one entry is created for each frame of main memory. Instead of searching page table of every process, every page are stored in one-page table. This page table contains process-id and page number.**

Only one-page table for all process.

Here logical address contains process-id , page number and displacement. Inverted page table will be searched according to page number and process –id. Here entry number will combine with displacement and physical address will be generated.



Hashed Page Tables

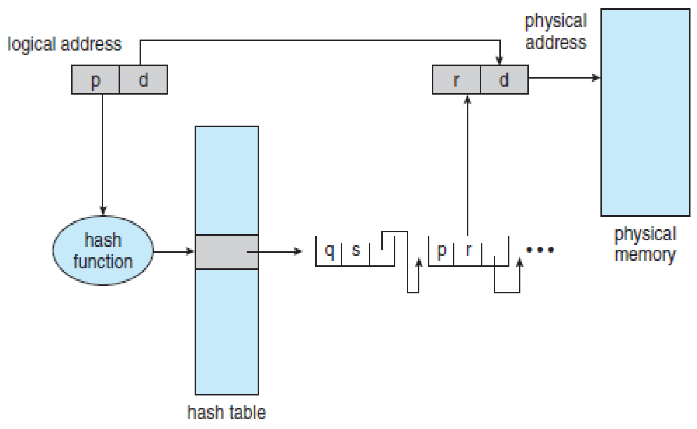
- A common approach for handling address spaces larger than 32 bits is to use a **hashed page table**, with the hash value being the virtual page number.

- Each entry in the hash table contains a linked list of elements that hash to the same location (to handle collisions).
- Each element consists of three fields:
    - the virtual page number
    - the value of the mapped page frame
    - a pointer to the next element in the linked list.

Algorithm:

- The virtual page number in the virtual address is hashed into the hash table.
- The virtual page number is compared with field 1 in the first element in the linked list.
- If there is a match, the corresponding page frame (field 2) is used to form the desired physical address.
- If there is no match, subsequent entries in the linked list are searched for a matching virtual page number.

This scheme is shown below:



- A variation of this scheme that is useful for 64-bit address spaces has been proposed.
- This variation uses **clustered page tables**, which are similar to hashed page tables except that each entry in the hash table refers to several pages (such as 16) rather than a single page. **Therefore, a single page-table entry can store the mappings for multiple physical-page frames.**

- Clustered page tables are particularly useful for sparse address spaces, where memory references are noncontiguous and scattered throughout the address space.