MODULE 3
MEMORY MANAGEMENT

5.1 MEMORY MANAGEMENT STRATGIES

- Memory management is concerned with managing the primary memory.
- Memory consists of array of bytes or words each with their own address.
- The instructions are fetched from the memory by the cpu based on the value program counter.

Functions of memory management:

- Keeping track of status of each memory location.
- Determining the allocation policy.
- Memory allocation technique.
- De-allocation technique.

Address Binding:

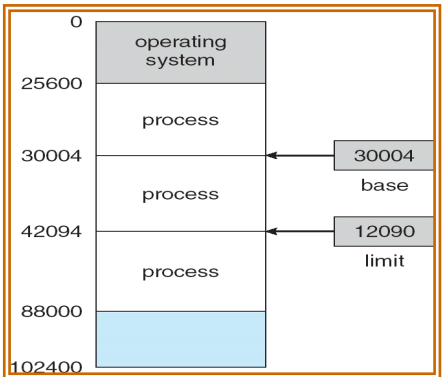
- Programs are stored on the secondary storage disks as binary executable files.
- When the programs are to be executed they are brought in to the main memory and placed within a process.
- The collection of processes on the disk waiting to enter the main memory forms the input queue. One of the processes which are to be executed is fetched from the queue and placed in the main memory.
- During the execution it fetches instruction and data from main memory. After the process terminates it returns back the memory space.
- During execution the process will go through different steps and in each step the address is represented in different ways.
- In source program the address is symbolic.
- The compiler converts the symbolic address to re-locatable address.
- The loader will convert this re-locatable address to absolute address.

Binding of instructions and data can be done at any step along the way:

1. Compile time:-If we know whether the process resides in memory then absolute code can be generated. If the static address changes then it is necessary to re-compile the code from the beginning.
2. Load time:-If the compiler doesn't know whether the process resides in memory then it generates the relocatable code. In this the binding is delayed until the load time.
3. Execution time:-If the process is moved during its execution from one memory segment to another then the binding is delayed until run time. Special hardware is used for this. Most of the general purpose operating system uses this method.

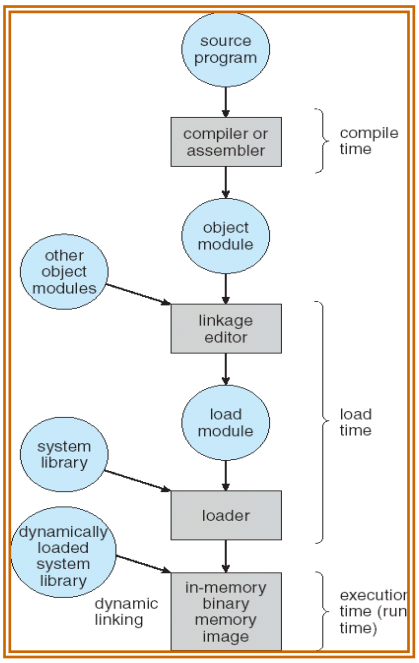
Base and Limit Registers

- A pair of **base** and **limit** registers define the logical address space



5.2 BACKGROUND

Logical versus physical address:
The address generated by the CPU is called logical address or virtual address.
The address seen by the memory unit i.e., the one loaded in to the memory register is called the physical address.
Compile time and load time address binding methods generate some logical and physical address.
The execution time addressing binding generate different logical and physical address.
Set of logical address space generated by the programs is the logical address space.
Set of physical address corresponding to these logical addresses is the physical address space.
The mapping of virtual address to physical address during run time is done by the Hardware device called memory management unit (MMU).
The base register is also called re-location register.
Value of the re-location register is added to every address generated by the user process at the time it is sent to memory.

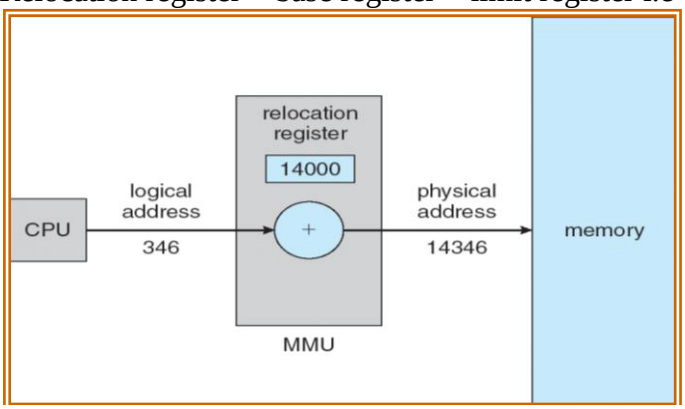


The above figure shows that dynamic re-location which implies mapping from virtual addresses space to physical address space and is performed by the hardware at run time.

Re-location is performed by the hardware and is invisible to the user dynamic relocation makes it possible to move a partially executed process from one area of memory to another without affecting.

Dynamic relocation using relocation register :

Relocation register = base register + limit register i.e combination of limit and base registers



Dynamic Loading:

For a process to be executed it should be loaded in to the physical memory. The size of the process is limited to the size of the physical memory.

Dynamic loading is used to obtain better memory utilization. x In dynamic loading the routine or procedure will not be loaded until it is called.

Whenever a routine is called, the calling routine first checks whether the called routine is already loaded or not. If it is not loaded it cause the loader to load the desired program in to the memory and updates the programs address table to indicate the change and control is passed to newly called routine.

Advantages: Gives better memory utilization.

Unused routine is never loaded.

Do not need special operating system support.

This method is useful when large amount of codes are needed to handle in frequently occurring cases.

Dynamic linking and Shared libraries:

Some operating system supports only the static linking.

In dynamic linking only the main program is loaded in to the memory. If the main program requests a procedure, the procedure is loaded and the link is established at the time of references. This linking is postponed until the execution time.

With dynamic linking a "stub" is used in the image of each library referenced routine. A "stub" is a piece of code which is used to indicate how to locate the appropriate memory resident library routine or how to load library if the routine is not already present.

When "stub" is executed it checks whether the routine is present is memory or not. If not it loads the routine in to the memory.

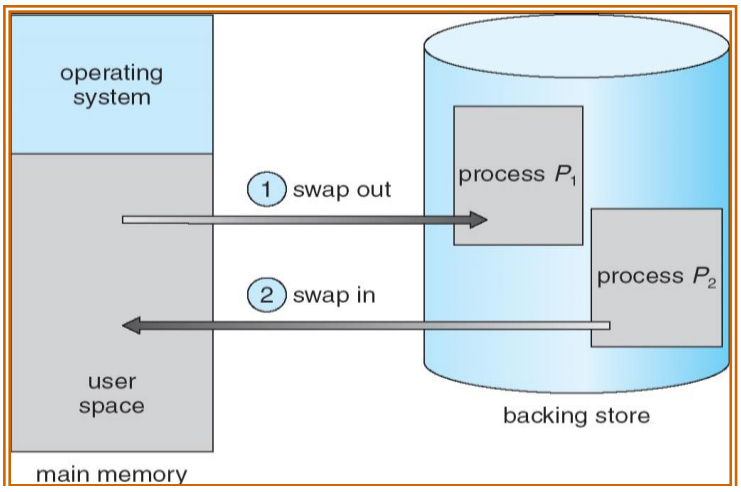
This feature can be used to update libraries i.e., library is replaced by a new version and all the programs can make use of this library.

More than one version of the library can be loaded in memory at a time and each program uses its version of the library. Only the program that are compiled with the new version are affected by the changes incorporated in it. Other programs linked before new version is installed will continue using older libraries this type of system is called "shared library".

5.3 SWAPPING

• Swapping is a technique of temporarily removing inactive programs from the memory of the system.
• A process can be swapped temporarily out of the memory to a backing store and then brought back in to the memory for continuing the execution. This process is called swapping.
Eg:-In a multi-programming environment with a round robin CPU scheduling whenever the time quantum expires then the process that has just finished is swapped out and a new process swaps in to the memory for execution.
• A variation of swap is priority based scheduling. When a low priority is executing and if a high priority process arrives then a low priority will be swapped out and high priority is allowed for execution. This process is also called as Roll out and Roll in.
• Normally the process which is swapped out will be swapped back to the same memory space that is occupied previously. This depends upon address binding.
• If the binding is done at load time, then the process is moved to same memory location.
• If the binding is done at run time, then the process is moved to different memory location. This is because the physical address is computed during run time.
• Swapping requires backing store and it should be large enough to accommodate the copies of all memory images.
• The system maintains a ready queue consisting of all the processes whose memory images are on the backing store or in memory that are ready to run.
• Swapping is constant by other factors:
x To swap a process, it should be completely idle.
x A process may be waiting for an i/o operation. If the i/o is asynchronously accessing the user memory for i/o buffers, then the process cannot be swapped.

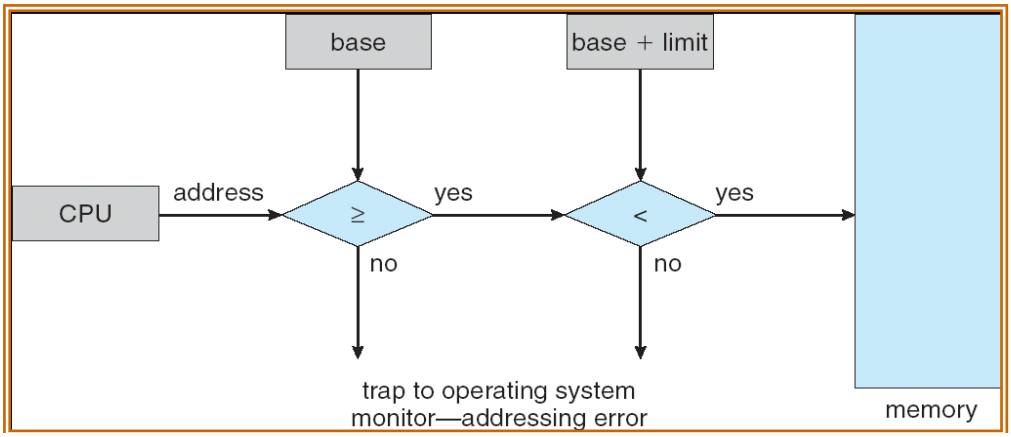
Schematic view of swapping :



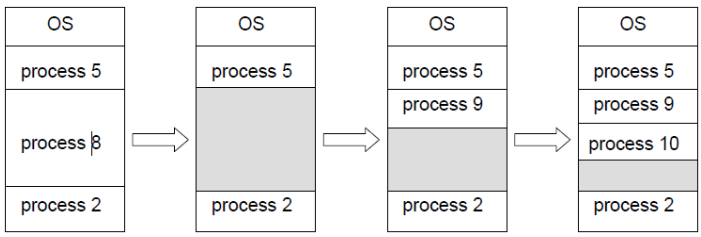
5.4 CONTIGUOUS MEMORY ALLOCATION

One of the simplest method for memory allocation is to divide memory in to several fixed partition.
Each partition contains exactly one process. The degree of multi-programming depends on the number of partitions.

Hardware address protection with base and limit registers :



In multiple partition method, when a partition is free, process is selected from the input queue and is loaded in to free partition of memory.
When process terminates, the memory partition becomes available for another process.
Batch OS uses the fixed size partition scheme.
The OS keeps a table indicating which part of the memory is free and is occupied.
When the process enters the system it will be loaded in to the input queue. The OS keeps track of the memory requirement of each process and the amount of memory available and determines which process to allocate the memory.
When a process requests, the OS searches for large hole for this process, hole is a large block of free memory available.
If the hole is too large it is split in to two. One part is allocated to the requesting process and other is returned to the set of holes.
OS maintains about a) allocated partitions b) free partitions(hole)



The set of holes are searched to determine which hole is best to allocate. There are three strategies to select a free hole:
• First bit:-Allocates first hole that is big enough. This algorithm scans memory from the beginning and selects the first available block that is large enough to hold the process.
• Best bit:-It chooses the hole i.e., closest in size to the request. It allocates the smallest hole i.e., big enough to hold the process.
• Worst fit:-It allocates the largest hole to the process request. It searches for the largest hole in the entire list.

• First fit and best fit are the most popular algorithms for dynamic memory allocation. First fit is generally faster. Best fit searches for the entire list to find the smallest hole i.e., large enough. Worst fit reduces the rate of production of smallest holes.
• All these algorithms suffer from fragmentation.

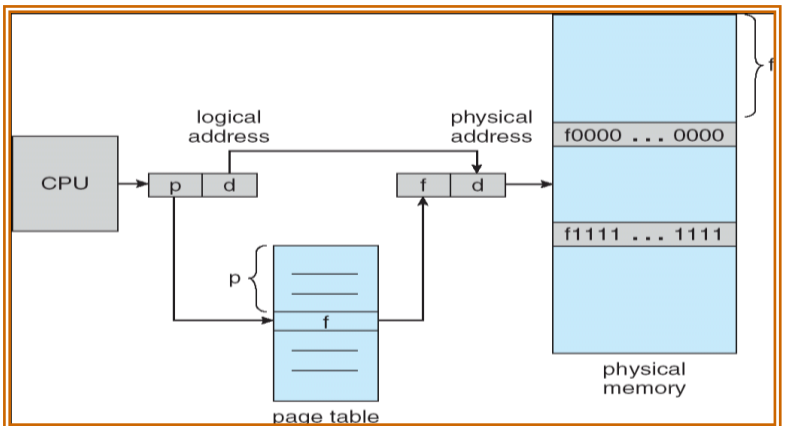
Memory Protection:
Memory protection means protecting the OS from user process and protecting process from one another.

Memory protection is provided by using a re-location register, with a limit register.
Relocation register contains the values of smallest physical address and limit register contains range of logical addresses. (Re-location = 100040 and limit = 74600).
The logical address must be less than the limit register, the MMU maps the logical address dynamically by adding the value in re-location register.
When the CPU scheduler selects a process for execution, the dispatcher loads the re-location and limit register with correct values as a part of context switch. x Since every address generated by the CPU is checked against these register we can protect the OS and other users programs and data from being modified.

Fragmentation:
• Memory fragmentation can be of two types: Internal Fragmentation & External Fragmentation
• In Internal Fragmentation there is wasted space internal to a portion due to the fact that block of data loaded is smaller than the partition. *Eg*:-If there is a block of 50kb and if the process requests 40kb and if the block is allocated to the process then there will be 10kb of memory left.
• External Fragmentation exists when there is enough memory space exists to satisfy the request, but it not contiguous i.e., storage is fragmented in to large number of small holes.
• External Fragmentation may be either minor or a major problem.
• One solution for over-coming external fragmentation is compaction. The goal is to move all the free memory together to form a large block. Compaction is not possible always. If the relocation is static and is done at load time then compaction is not possible. Compaction is possible if the re-location is dynamic and done at execution time.
• Another possible solution to the external fragmentation problem is to permit the logical address space of a process to be non-contiguous, thus allowing the process to be allocated physical memory whenever the latter is available.

5.5 PAGING AND STRUCTURE OF PAGE TABLE

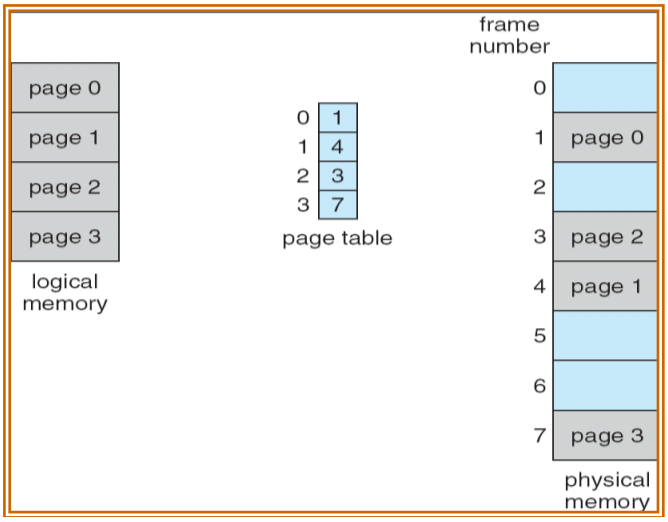
Paging is a memory management scheme that permits the physical address space of a process to be non-contiguous. Support for paging is handled by hardware.
It is used to avoid external fragmentation.
Paging avoids the considerable problem of fitting the varying sized memory chunks on to the backing store.
When some code or date residing in main memory need to be swapped out, space must be found on backing store.



Basic Method:

Physical memory is broken in to fixed sized blocks called frames (f).
Logical memory is broken in to blocks of same size called pages (p).

Paging model of logical and physical memory :



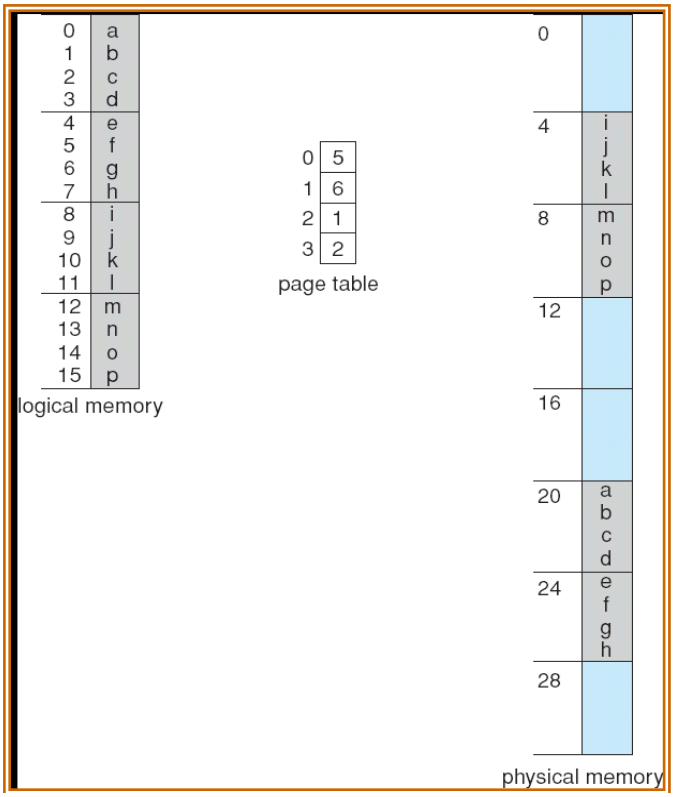
When a process is to be executed its pages are loaded in to available frames from backing store. The blocking store is also divided in to fixed-sized blocks of same size as memory frames.

Logical address generated by the CPU is divided in to two parts: page number (p) and page offset (d).

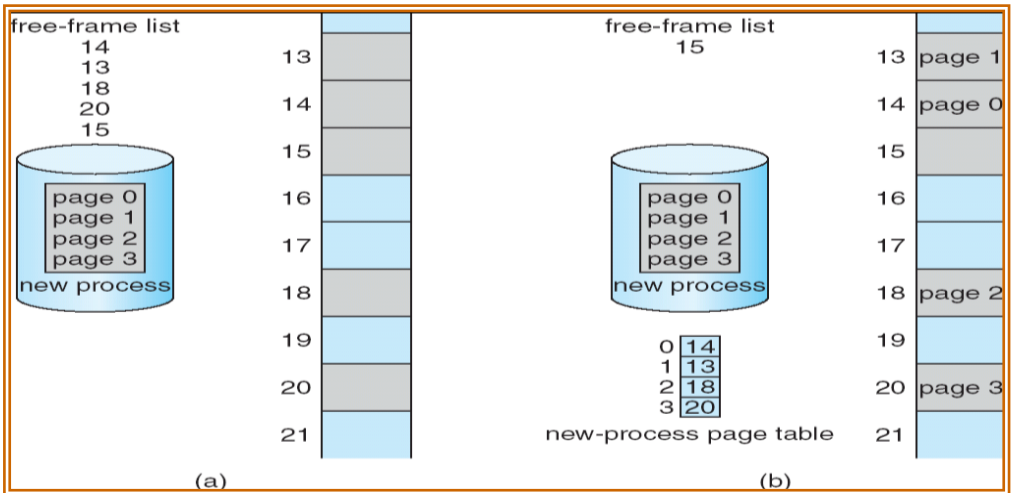
page number	page offset
p	*d*
m - n	*n*

The page number (p) is used as index to the page table. The page table contains base address of each page in physical memory. This base address is combined with the page offset to define the physical memory i.e.,sent to the memory unit.
The page size is defined by the hardware. The size of a power of 2, varying between 512 bytes and 10Mb per page.
If the size of logical address space is 2^m address unit and page size is 2^n, then high order m-n designates the page number and n low order bits represents page offset.
Eg:-To show how to map logical memory in to physical memory consider a page size of 4 bytes and physical memory of 32 bytes (8 pages).
a. Logical address 0 is page 0 and offset 0. Page 0 is in frame 5. The logical address 0 maps to physical address 20. [(5*4) + 0].
b. Logical address 3 is page 0 and offset 3 maps to physical address 23 [(5*4) + 3].
c. Logical address 4 is page 1 and offset 0 and page 1 is mapped to frame 6. So logical address 4 maps to physical address 24 [(6*4) + 0].
d. Logical address 13 is page 3 and offset 1 and page 3 is mapped to frame 2. So logical address 13 maps to physical address 9 [(2*4) + 1].

Fig. Paging example for a 32 byte memory with 4-byte pages:



- When a process arrives in the system to be executed , its size , expressed in form of pages is examined.
- Each page in a process needs one frame.
- If the process requires n pages, at least n frames must be available in memory.
- The first page of the process is loaded into one of the allocated frames.
- Frame number is put into page table for this process.
- Similarly next page also is loaded and page table is updated.



Hardware Support for Paging:

The hardware implementation of the page table can be done in several ways:

1. The simplest method is that the page table is implemented as a set of dedicated registers. These registers must be built with very high speed logic for making paging address translation. Every accessed memory must go through paging map. The use of registers for page table is satisfactory if the page table is small.
2. If the page table is large then the use of registers is not visible. So the page table is kept in the main memory and a page table base register [PTBR] points to the page table. Changing the page table requires only one register which reduces the context switching type. The problem with this approach is the time required to access memory location. To access a location [i] first we have to index the page table using PTBR offset. It gives the frame number which is combined with the page offset to produce the actual address. Thus we need two memory accesses for a byte.
3. The only solution is to use special, fast, lookup hardware cache called translation look aside buffer [TLB] or associative register. TLB is built with associative register with high speed memory. Each register contains two paths a key and a value.

When an associative register is presented with an item, it is compared with all the key values, if found the corresponding value field is return and searching is fast.

TLB is used with the page table as follows:x TLB contains only few page table entries.

When a logical address is generated by the CPU, its page number along with the frame number is added to TLB. If the page number is found its frame memory is used to access the actual memory.

If the page number is not in the TLB (TLB miss) the memory reference to the page table is made. When the frame number is obtained use can use it to access the memory.

If the TLB is full of entries the OS must select anyone for replacement.

Each time a new page table is selected the TLB must be flushed [erased] to ensure that next executing process do not use wrong information.

The percentage of time that a page number is found in the TLB is called HIT ratio.

Associative Memory

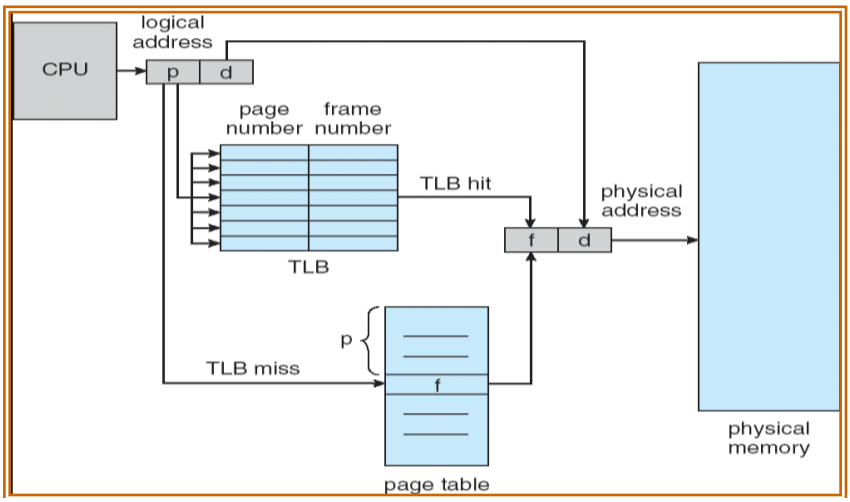
- Associative memory – parallel search

Page #	Frame #

Address translation (p, d)

- If p is in associative register, get frame # out
- Otherwise get frame # from page table in memory

Paging hardware with TLB :



Effective Access Time

- Associative Lookup = ε time unit
- Assume memory cycle time is 1 microsecond
- **Hit ratio** – percentage of times that a page number is found in the associative registers; ratio related to number of associative registers
- Hit ratio = α
- **Effective Access Time** (EAT) is success rate and failure rate.
- **Success rate is time taken to access TLB and main memory (TLB hit).**
- Failure

  EAT = (1 + ε) α + (2 + ε)(1 – α)

  = 2 + ε – α

Eg. 80% hit ratio , Main memory access=120ns , Page table access time = 120ns , TLB access = 20ns

E.A.T= 0.8 * 120 + 0.2 * 220

= 140ns.

Access time is slowed down by 40%.

Protection:

Memory protection in paged environment is done by protection bits that are associated with each frame these bits are kept in page table.

One bit can define a page to be read-write or read-only.

To find the correct frame number every reference to the memory should go through page table. At the same time physical address is computed.

The protection bits can be checked to verify that no writers are made to read-only page.

Any attempt to write in to read-only page causes a hardware trap to the OS.

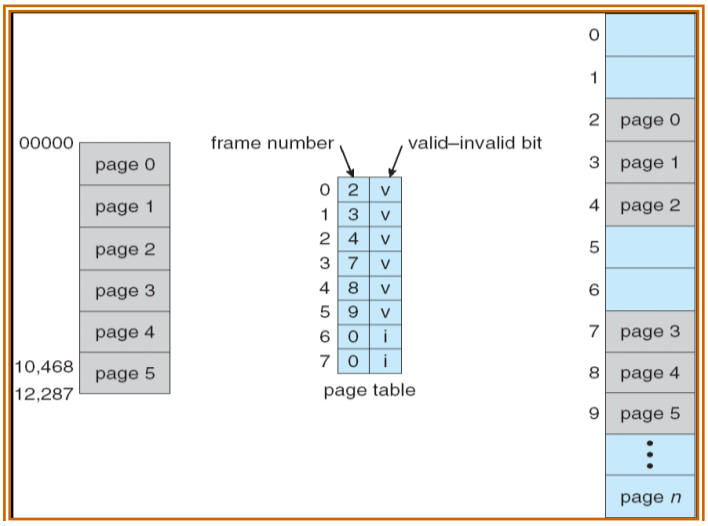
This approach can be used to provide protection to read-only, read-write or execute-only pages.

One more bit is generally added to each entry in the page table: a valid-invalid bit.

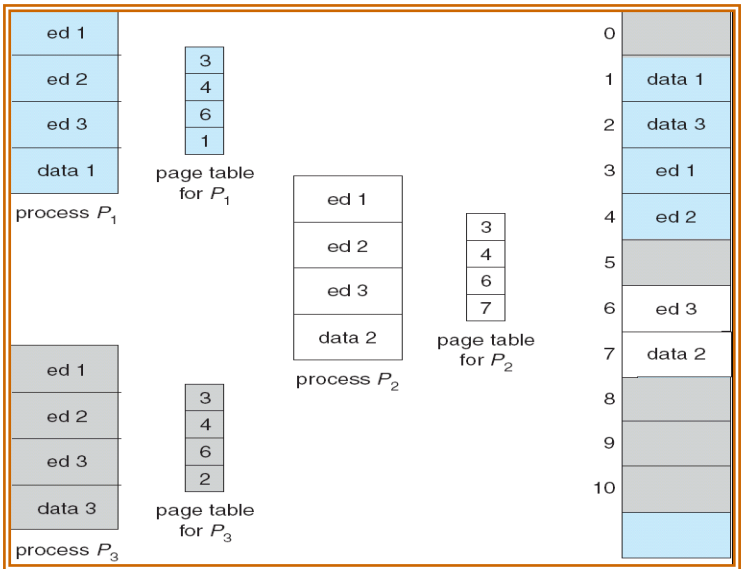
A valid bit indicates that associated page is in the processes logical address space and thus it is a legal or valid page.

If the bit is invalid, it indicates the page is not in the processes logical addressed space and illegal. Illegal addresses are trapped by using the valid-invalid bit.

The OS sets this bit for each page to allow or disallow accesses to that page.



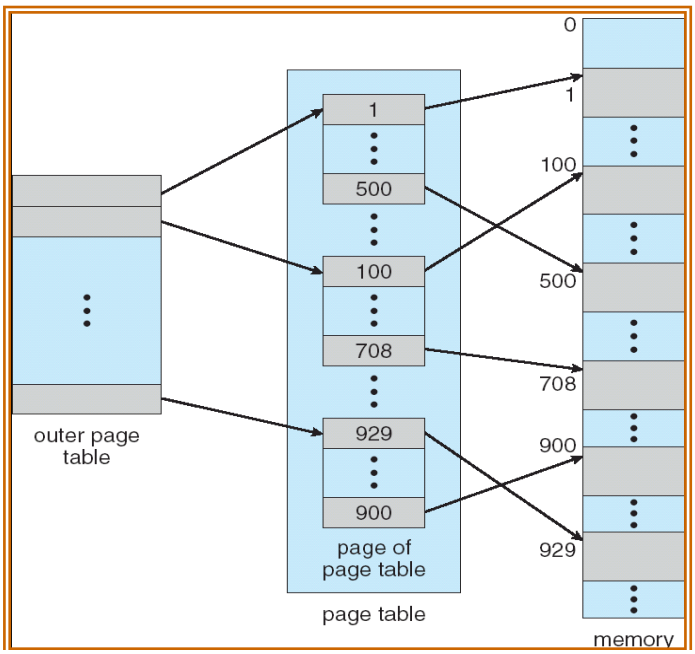
Shared Pages:
Another advantage of paging is the possibility of sharing common code. This is useful in timesharing environment. *Eg*:-Consider a system with 40 users, each executing a text editor. If the text editor is of 150k and data space is 50k, we need 8000k for 40 users. If the code is reentrant it can be shared. Consider the following figure
• If the code is reentrant then it never changes during execution. Thus two or more processes can execute same code at the same time. Each process has its own copy of registers and the data of two processes will vary.
• Only one copy of the editor is kept in physical memory. Each users page table maps to same physical copy of editor but date pages are mapped to different frames.
• So to support 40 users we need only one copy of editor (150k) plus 40 copies of 50k of data space i.e., only 2150k instead of 8000k
Fig : Text editor is shared among multiple users.
Instead of allocating frames for duplicate copies of same editor program , copy is shared.
Data space is dedicated for each user, which could be edited .
Editor process is divided into 3 pages.
If process size is 150KB, page size is 50KB each and data is 50KB.
Hence, 3 frames are allocated for 3 pages of editor process.
Total space allocated for 3 user processes are (150)+(3*150)=600KB.



Structure of the Page Table
a. **Hierarchical paging:**
• Recent computer system support a large logical address apace from 2^32 to 2^64. In this system the page table becomes large. So it is very difficult to allocate contiguous main memory for page table. One simple solution to this problem is to divide page table in to smaller pieces. There are several ways to accomplish this division.
• One way is to use two-level paging algorithm in which the page table itself is also paged. *Eg*:-In a 32 bit machine with page size of 4kb. A logical address is divided in to a page number consisting of 20 bits and a page offset of 12 bit. The page table is further divided since the page table is paged, the page number is further divided in to 10 bit page number and a 10 bit offset. So the logical address is

- Break up the logical address space into multiple page tables
- A simple technique is a two-level page table

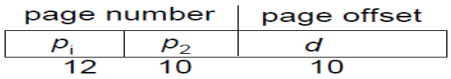
2 level paging scheme:



Two-Level Paging Example

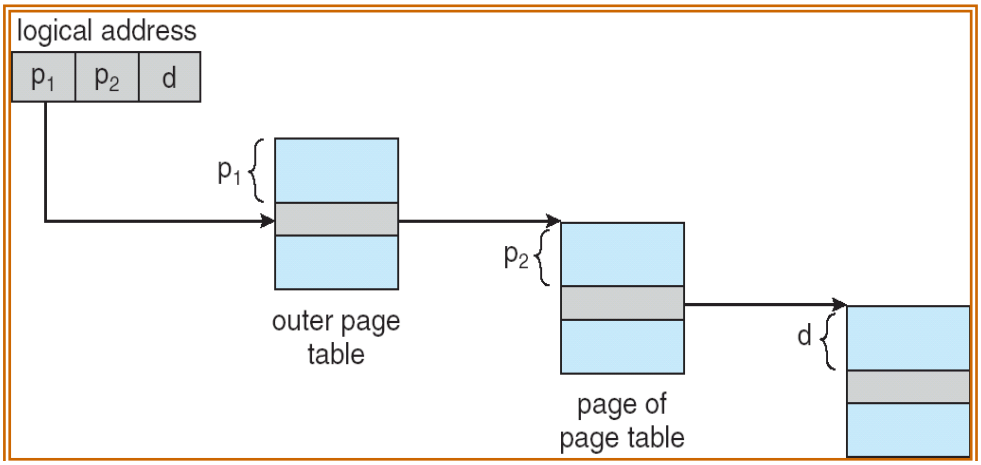
A logical address (on 32-bit machine with 1K page size) is divided into:

- a page number consisting of 22 bits
- a page offset consisting of 10 bits
- Since the page table is paged, the page number is further divided into:
- a 12-bit page number
- a 10-bit page offset
- Thus, a logical address is as follows:



where ***pi*** is an index into the outer page table, and ***p2*** is the displacement within the page of the outer page table

Address-Translation Scheme



3-level paging scheme :

2nd outer page	outer page	inner page	offset
p_1	p_2	p_3	d
32	10	10	12

outer page	inner page	offset
p_1	p_2	d
42	10	12

b. **Hashed page table:**

• Hashed page table handles the address space larger than 32 bit. The virtual page number is used as hashed value. Linked list is used in the hash table which contains a list of elements that hash to the same location.

• Each element in the hash table contains the following three fields:

Virtual page number Mapped page frame value Pointer to the next element in the linked list

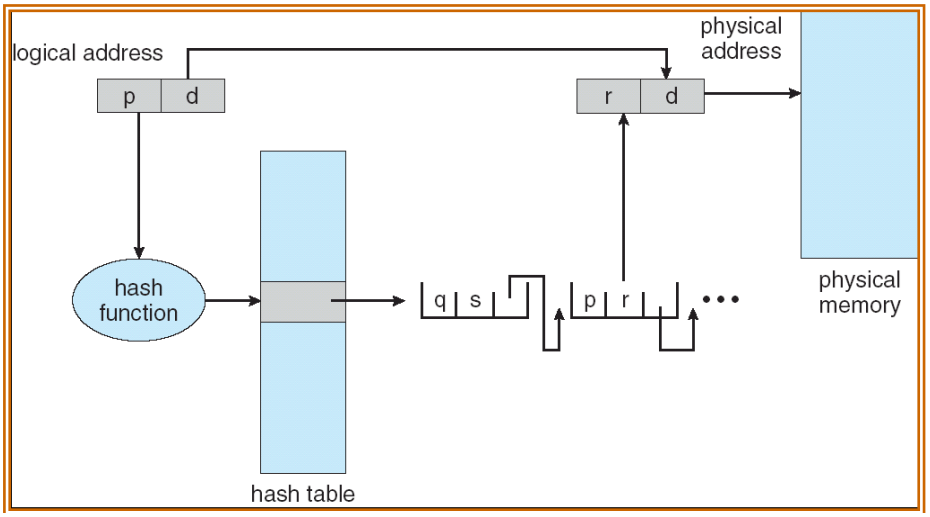
Working:

Virtual page number is taken from virtual address.

Virtual page number is hashed in to hash table.

Virtual page number is compared with the first element of linked list.

Both the values are matched, that value is (page frame) used for calculating the physical address. If not match then entire linked list is searched for matching virtual page number.



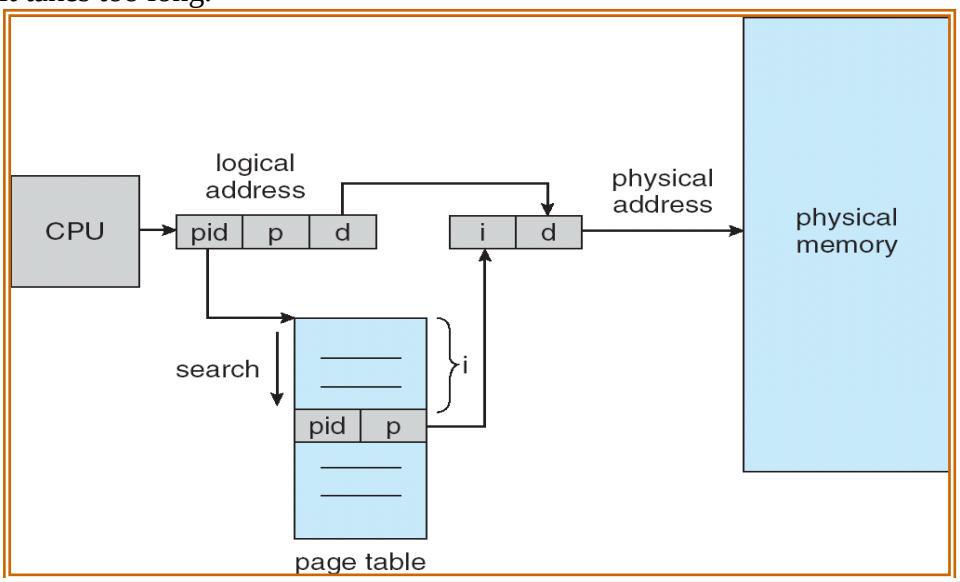
Clustered pages are similar to hash table but one difference is that each entity in the hash table refer to several pages.

c. **Inverted Page Tables:**

Since the address spaces have grown to 64 bits, the traditional page tables become a problem. Even with two level page tables, the table can be too large to handle. An inverted page table has only entry for each page in memory. Each entry consisted of virtual address of the page stored in that read-only location with information about the process that owns that page.

Each virtual address in the Inverted page table consists of triple <process-id , page number , offset >. The inverted page table entry is a pair <process-id , page number>. When a memory reference is made, the part of virtual address i.e., <process-id , page number> is presented in

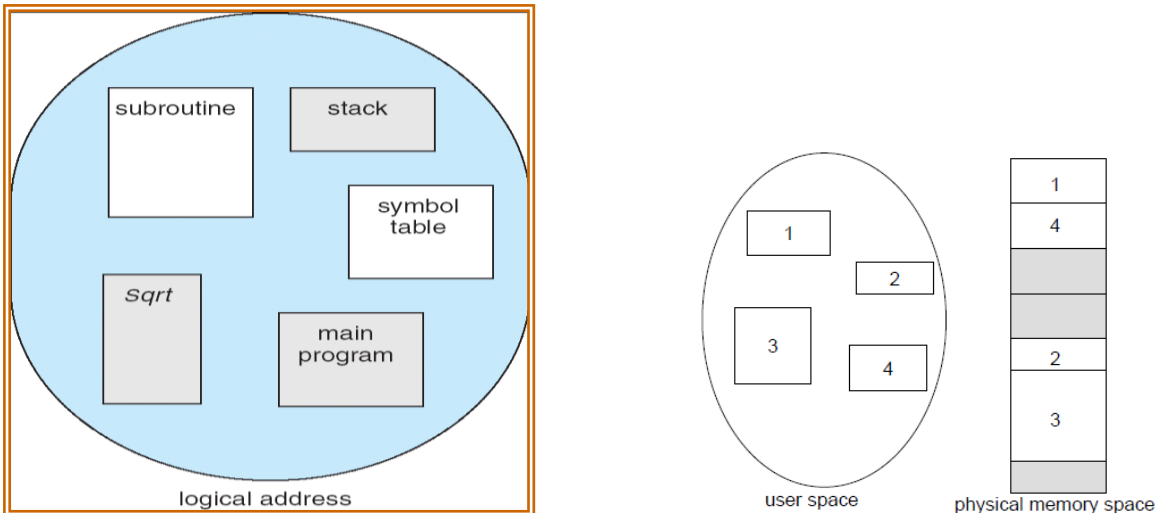
to memory sub-system. The inverted page table is searched for a match. If a match is found at entry I then the physical address <i , offset> is generated.
If no match is found then an illegal address access has been attempted. This scheme decreases the amount of memory needed to store each page table, it increases the amount of time needed to search the table when a page reference occurs. If the whole table is to be searched it takes too long.



Advantage:
x Eliminates fragmentation.
x Support high degree of multiprogramming.
x Increases memory and processor utilization.
x Compaction overhead required for the re-locatable partition scheme is also eliminated.
Disadvantage:
x Page address mapping hardware increases the cost of the computer.
x Memory must be used to store the various tables like page tables, memory map table etc.
x Some memory will still be unused if the number of available block is not sufficient for the address space of the jobs to be run.

5.6 SEGMENTATION

Basic method:
x Most users do not think memory as a linear array of bytes rather the users thinks memory as a collection of variable sized segments which are dedicated to a particular use such as code, data, stack, heap etc.
x A logical address is a collection of segments. Each segment has a name and length. The address specifies both the segment name and the offset within the segments.
x The users specifies address by using two quantities: a segment name and an offset. x For simplicity the segments are numbered and referred by a segment number. So the logical address consists of <segment number, offset>.
Hardware support:
x We must define an implementation to map 2D user defined address in to 1D physical address.
x This mapping is affected by a segment table. Each entry in the segment table has a segment base and segment limit. The segment base contains the starting physical address where the segment resides and limit specifies the length of the segment.



Logical view of segmentation

The use of segment table is shown in the above figure:
x Logical address consists of two parts: segment number‘s’ and an offset‘d’ to that segment.
x The segment number is used as an index to segment table.
x The offset ’d’ must bi in between 0 and limit, if not an error is reported to OS.
x If legal the offset is added to the base to generate the actual physical address.
x The segment table is an array of base limit register pairs.

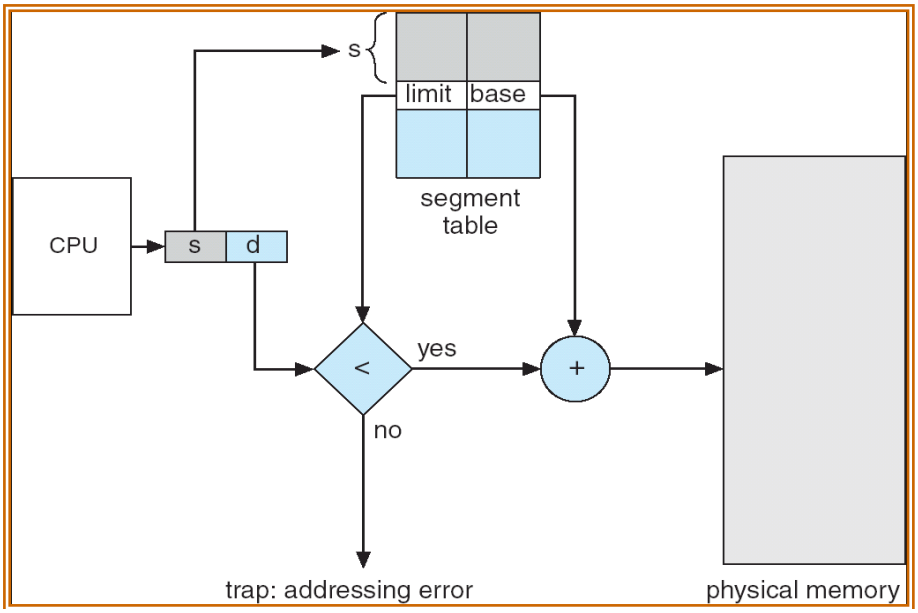
Segmentation Architecture

- Logical address consists of a two tuple:

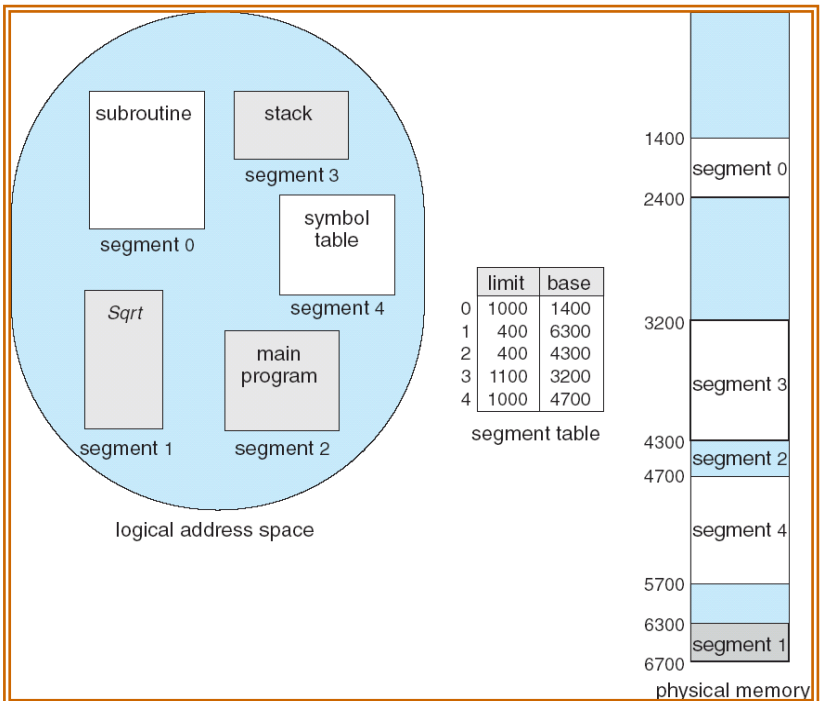
<segment-number, offset>

- **Segment table** – maps two-dimensional physical addresses; each table entry has:
- **base** – contains the starting physical address where the
- segments reside in memory
- **limit** – specifies the length of the segment
- **Segment-table base register (STBR)** points to the segment table’s location in memory
- **Segment-table length register (STLR)** indicates number of segments used by a program;
- segment number ***s*** is legal if ***s*** < **STLR**

Protection and Sharing:
x A particular advantage of segmentation is the association of protection with the segments.
x The memory mapping hardware will check the protection bits associated with each segment table entry to prevent illegal access to memory like attempts to write in to read-only segment.
x Another advantage of segmentation involves the sharing of code or data. Each process has a segment table associated with it. Segments are shared when the entries in the segment tables of two different processes points to same physical location.
x Sharing occurs at the segment table. Any information can be shared at the segment level. Several segments can be shared so a program consisting of several segments can be shared.
x We can also share parts of a program.



Eg. of Segmentation :



Advantages:

x Eliminates fragmentation.
x Provides virtual growth.
x Allows dynamic segment growth.
x Assist dynamic linking.
x Segmentation is visible.

Differences between segmentation and paging:-

Segmentation:

x Program is divided in to variable sized segments.
x User is responsible for dividing the program in to segments.
x Segmentation is slower than paging.

x Visible to user.
x Eliminates internal fragmentation.
x Suffers from external fragmentation.
x Process or user segment number, offset to calculate absolute address.

Paging:
x Programs are divided in to fixed size pages.
x Division is performed by the OS.
x Paging is faster than segmentation.
x Invisible to user.
x Suffers from internal fragmentation.
xNo external fragmentation.
x Process or user page number, offset to calculate absolute address.