DEADLOCKS

When processes request a resource and if the resources are not available at that time the process enters into waiting state. Waiting process may not change its state because the resources they are requested are held by other process. This situation is called deadlock.

The situation where the process waiting for the resource i.e., not available is called deadlock.

System Model:-

A system may consist of finite number of resources and is distributed among number of processes. There resources are partitioned into several instances each with identical instances.

A process must request a resource before using it and it must release the resource after using it. It can request any number of resources to carry out a designated task. The amount of resource requested may not exceed the total number of resources available.

A process may utilize the resources in only the following sequences:-

1. Request:- If the request is not granted immediately then the requesting process must wait it can acquire the resources.
2. Use:- The process can operate on the resource.
3. Release:- The process releases the resource after using it.

Deadlock may involve different types of resources.

For eg:- Consider a system with one printer and one tape drive. If a process Pi currently holds a printer and a process Pj holds the tape drive. If process Pi request a tape drive and process Pj request a printer then a deadlock occurs.

Multithread programs are good candidates for deadlock because they compete for shared resources.

Deadlock Characterization:-

Necessary Conditions:-

A deadlock situation can occur if the following 4 conditions occur simultaneously in a system:-

1. Mutual Exclusion:- Only one process must hold the resource at a time. If any other process requests for the resource, the requesting process must be delayed until the resource has been released.

2. Hold and Wait:- A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by the other process.

3. No Preemption:- Resources can't be preempted i.e., only the process holding the resources must release it after the process has completed its task.

4. Circular Wait:- A set {P0,P1……..Pn} of waiting process must exist such that P0 is waiting for a resource i.e., held by P1, P1 is waiting for a resource i.e., held by P2. Pn-1 is waiting for resource held by process Pn and Pn is waiting for the resource i.e., held by P1. All the four conditions must hold for a deadlock to occur.

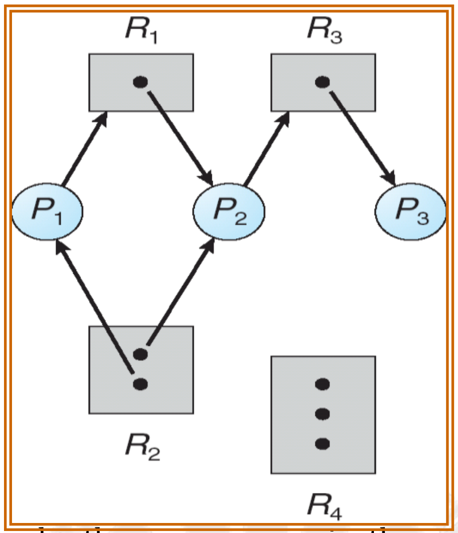
Resource Allocation Graph:-

Deadlocks are described by using a directed graph called system resource allocation graph. The graph consists of set of vertices (v) and set of edges (e).

The set of vertices (v) can be described into two different types of nodes P={P1,P2……..Pn} i.e., set consisting of all active processes and R={R1,R2……….Rn}i.e., set consisting of all resource types in the system.

A directed edge from process Pi to resource type Rj denoted by Pi → Ri indicates that Pi requested an instance of resource Rj and is waiting. This edge is called Request edge.

A directed edge Ri →Pj signifies that resource Rj is held by process Pi. This is called Assignment edge.



If the graph contain no cycle, then no process in the system is deadlock. If the graph contains a cycle then a deadlock may exist.

If each resource type has exactly one instance than a cycle implies that a deadlock has occurred. If each resource has several instances then a cycle do not necessarily implies that a deadlock has occurred.

Methods for Handling Deadlocks:-

There are three ways to deal with deadlock problem

- We can use a protocol to prevent deadlocks ensuring that the system will never enter into the deadlock state.
- We allow a system to enter into deadlock state, detect it and recover from it.

- We ignore the problem and pretend that the deadlock never occur in the system. This is used by most OS including UNIX.

To ensure that the deadlock never occur the system can use either deadlock avoidance or a deadlock prevention.

Deadlock prevention is a set of method for ensuring that at least one of the necessary conditions does not occur.
Deadlock avoidance requires the OS is given advance information about which resource a process will request and use during its lifetime.
If a system does not use either deadlock avoidance or deadlock prevention then a deadlock situation may occur. During this it can provide an algorithm that examines the state of the system to determine whether a deadlock has occurred and algorithm to recover from deadlock. Undetected deadlock will result in deterioration of the system performance.

<u>Deadlock Prevention</u>:-

For a deadlock to occur each of the four necessary conditions must hold. If at least one of

the there condition does not hold then we can prevent occurrence of deadlock.

1. <u>Mutual Exclusion</u>:-
   This holds for non-sharable resources.

Eg:- A printer can be used by only one process at a time.
Mutual exclusion is not possible in sharable resources and thus they cannot be involved in deadlock. Read-only files are good examples for sharable resources. A process never waits for accessing a sharable resource. So we cannot prevent deadlock by denying the mutual exclusion condition in non-sharable resources.

2. <u>Hold and Wait</u>:-
   This condition can be eliminated by forcing a process to release all its resources

held by it when it request a resource i.e., not available.
One protocol can be used is that each process is allocated with all of its resources before its start execution.

Eg:- consider a process that copies the data from a tape drive to the disk, sorts the file and then prints the results to a printer. If all the resources are allocated at the beginning then the tape drive, disk files and printer are assigned to the process. The main problem with this is it leads to low resource utilization because it requires printer at the last and is allocated with it from the beginning so that no other process can use it.
Another protocol that can be used is to allow a process to request a resource when the process has none. i.e., the process is allocated with tape drive and disk file. It performs the required operation and releases both. Then the process once again request for disk file and the printer. Problem with this is starvation is possible.

3. <u>No Preemption</u>:-

To ensure that this condition never occurs the resources must be preempted. The following protocol can be used.
If a process is holding some resource and request another resource that cannot be immediately allocated to it, then all the resources currently held by the requesting process are preempted and added to the list of resources for which other processes may be waiting. The process will be restarted only when it regains the old resources and the new resources that it is requesting.
When a process request resources, we check whether they are available or not. If they are available we allocate them else we check that whether they are allocated to some other waiting process. If so we preempt the resources from the waiting process and allocate them to the requesting process. The requesting process must wait.

4. Circular Wait:-

The fourth and the final condition for deadlock is the circular wait condition. One way to ensure that this condition never, is to impose ordering on all resource types and each process requests resource in an increasing order.

Let R={R1,R2,………Rn} be the set of resource types. We assign each resource type with a unique integer value. This will allows us to compare two resources and determine whether one precedes the other in ordering.

Eg:-we can define a one to one function as follows :-

F(disk drive)=5   F(printer)=12 F(tape drive)=1

Deadlock can be prevented by using the following protocol:-

Each process can request the resource in increasing order. A process can request any number of instances of resource type say Ri and it can request instances of resource type Rj only F(Rj) > F(Ri).

Alternatively when a process requests an instance of resource type Rj, it has released any resource Ri such that F(Ri) >= F(Rj). If these two protocol are used then the circular wait can't hold.

Deadlock Avoidance:-

Deadlock prevention algorithm may lead to low device utilization and reduces system throughput.

Avoiding deadlocks requires additional information about how resources are to be requested. With the knowledge of the complete sequences of requests and releases we can decide for each requests whether or not the process should wait. For each requests it requires to check the resources currently available, resources that are currently allocated to each processes future requests and release of each process to decide
whether the current requests can be satisfied or must wait to avoid future possible deadlock.

A deadlock avoidance algorithm dynamically examines the resources allocation state to ensure that a circular wait condition never exists. The resource allocation state is defined by the number of available and allocated resources and the maximum demand of each process.

Safe State:-

A state is a safe state in which there exists at least one order in which all the process will run completely without resulting in a deadlock.

A system is in safe state if there exists a safe sequence.

A sequence of processes <P1,P2,………..Pn> is a safe sequence for the current allocation state if for each Pi the resources that Pi can request can be satisfied by the currently available resources.
If the resources that Pi requests are not currently available then Pi can obtain all of its needed resource to complete its designated task.

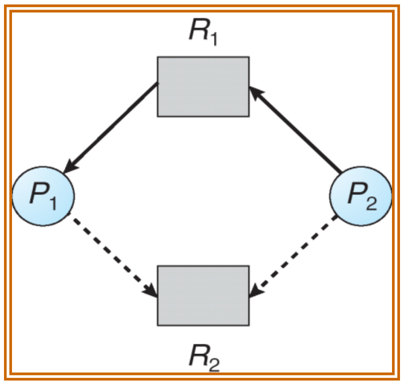
A safe state is not a deadlock state.

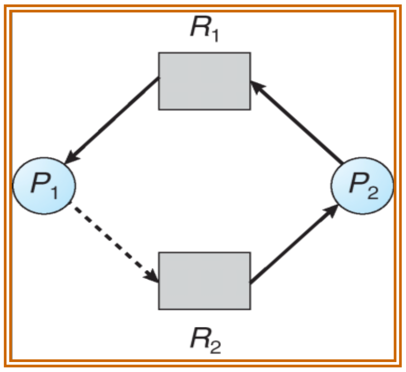
Whenever a process request a resource i.e., currently available, the system must decide whether resources can be allocated immediately or whether the process must wait. The request is granted only if the allocation leaves the system in safe state.
In this, if a process requests a resource i.e., currently available it must still have to wait. Thus resource utilization may be lower than it would be without a deadlock avoidance algorithm.

Resource Allocation Graph Algorithm:-

This algorithm is used only if we have one instance of a resource type. In addition to the request edge and the assignment edge a new edge called claim edge is used.
For eg:- A claim edge Pi Rj indicates that process Pi may request Rj in future. The claim edge is represented by a dotted line.

- When a process Pi requests the resource Rj, the claim edge is converted to a request edge.
- When resource Rj is released by process Pi, the assignment edge Rj Pi is replaced by the claim edge Pi Rj.
- When a process Pi requests resource Rj the request is granted only if converting the request edge Pi Rj to as assignment edge Rj Pi do not result in a cycle. Cycle detection algorithm is used to detect the cycle. If there are no cycles then the allocation of the resource to process leave the system in safe state





.**Banker's Algorithm**:-

This algorithm is applicable to the system with multiple instances of each resource types, but this is less efficient than the resource allocation graph algorithm.

When a new process enters the system it must declare the maximum number of resources that it may need. This number may not exceed the total number of resources in the system. The system must determine that whether the allocation of the resources will leave the system in a safe state or not. If it is so resources are allocated else it should wait until the process release enough resources.

Several data structures are used to implement the banker's algorithm. Let 'n' be the number of processes in the system and 'm' be the number of resources types. We need the following data structures:-

Available:- A vector of length m indicates the number of available resources. If Available[j]=k, then k instances of resource type Rj is available.
Max:- An n*m matrix defines the maximum demand of each process if Max[i,j]=k, then Pi may request at most k instances of resource type Rj.
Allocation:- An n*m matrix defines the number of resources of each type currently allocated to each process. If Allocation[i,j]=k, then Pi is currently k instances of resource type Rj.
Need:- An n*m matrix indicates the remaining resources need of each process. If Need[i,j]=k, then Pi may need k more instances of resource type Rj to compute its task. So Need[i,j]=Max[i,j]-Allocation[i]

Safety Algorithm:-

This algorithm is used to find out whether or not a system is in safe state or not.

```
Step 1. Let work and finish be two vectors of length M and N respectively.
        Initialize work = available and
        Finish[i]=false for i=1,2,3,…….n
Step 2. Find i such that both
                Finish[i]=false
                Need i <= work
        If no such i exist then go to step 4
Step 3. Work = work + Allocation
                Finish[i]=true
                Go to step 2
Step 4. If finish[i]=true for all i, then the system is in safe state.
```

This algorithm may require an order of m*n*n operation to decide whether a state is safe.

Resource Request Algorithm:-

Let Request(i) be the request vector of process Pi. If Request[(i)[j]=k, then process Pi wants K instances of the resource type Rj. When a request for resources is made by process Pi the following actions are taken.

If Request(i) <= Need(i) go to step 2 otherwise raise an error condition since the process has exceeded its maximum claim.

If Request(i) <= Available go to step 3 otherwise Pi must wait. Since the resources are not available.

If the system want to allocate the requested resources to process Pi then modify the state as follows.

Available = Available – Request(i)
Allocation(i) = Allocation(i) + Request(i)
Need(i) = Need(i) - Request(i)

If the resulting resource allocation state is safe, the transaction is complete and Pi is allocated its resources. If the new state is unsafe then Pi must wait for Request(i) and old resource allocation state is restored.

Deadlock Detection:-

If a system does not employ either deadlock prevention or a deadlock avoidance algorithm then a deadlock situation may occur. In this environment the system may provide

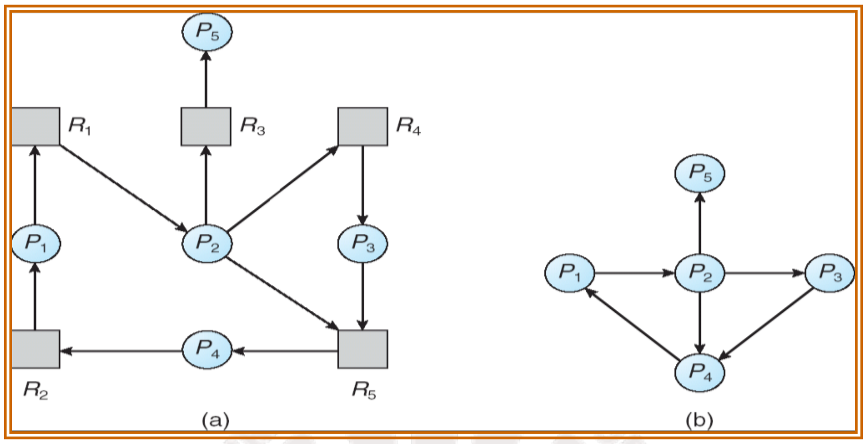
- An algorithm that examines the state of the system to determine whether a deadlock has occurred.
- An algorithm to recover from the deadlock.

Single Instances of each Resource Type:-

If all the resources have only a single instance then we can define deadlock detection algorithm that uses a variant of resource allocation graph called a wait for graph. This graph is obtained by removing the nodes of type resources and removing appropriate edges.

An edge from Pi to Pj in wait for graph implies that Pi is waiting for Pj to release a resource that Pi needs.

An edge from Pi to Pj exists in wait for graph if and only if the corresponding resource allocation graph contains the edges Pi →Rq and Rq→ Pj. Deadlock exists within the system if and only if there is a cycle. To detect deadlock the system needs an algorithm that searches for cycle in a graph.



Several Instances of a Resource Types:-

The wait for graph is applicable to only a single instance of a resource type. The following algorithm applies if there are several instances of a resource type. The following data structures are used:-

Available:- Is a vector of length m indicating the number of available resources of each type.
Allocation:- Is an m*n matrix which defines the number of resources of each type currently allocated to each process.
Request:- Is an m*n matrix indicating the current request of each process.
If request[i,j]=k then Pi is requesting k more instances of resources type Rj.

```
Step 1. let work and finish be vectors of length m and n respectively.
Initialize Work = available/expression.
For i=0,1,2……….n if allocation(i)!=0 then Finish[i]=0 else Finish[i]=true

Step 2. Find an index(i) such that both
                    Finish[i] = false
                    Request(i)<=work
              If no such I exist go to step 4.
Step 3. Work = work + Allocation(i)
             Finish[i] = true
              Go to step 2.
Step 4. If Finish[i] = false for some i where m>=i>=1.
```

When a system is in a deadlock state. This algorithm needs an order of m*n square operations to detect whether the system is in deadlock state or not.

Recovery From Deadlock

- Abort all deadlocked processes.
- Abort one process at a time until the deadlock cycle is eliminated.
  In which order should we choose to abort?
  - Priority of the process.
  - How long process has computed, and how much longer to completion.
  - Resources the process has used.
  - Resources process needs to complete.
  - How many processes will need to be terminated.
  - Is process interactive or batch?.

Resource Preemption

- Selecting a victim – minimize cost.
- Rollback – return to some safe state, restart process for that state.
- Starvation – same process may always be picked as victim, include number of rollback in cost factor.

MEMORY MANAGEMENT

Background:

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:-

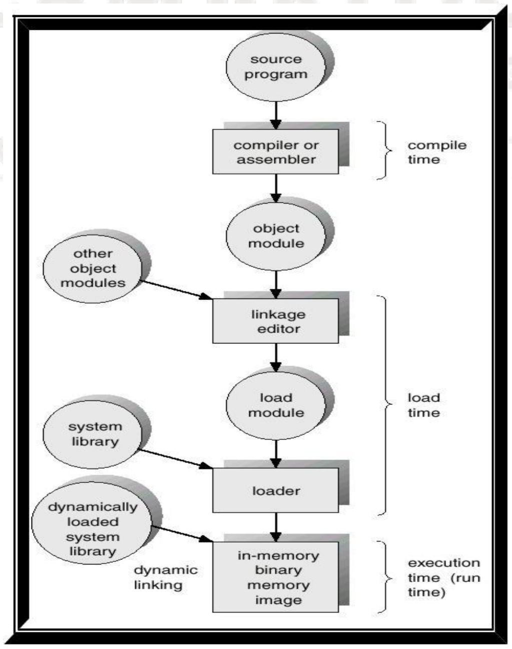
- Keeping track of status of each memory location..
- Determining the allocation policy. Memory allocation technique. De-allocation technique.

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 re-locatable 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.

Logical versus physical address:-

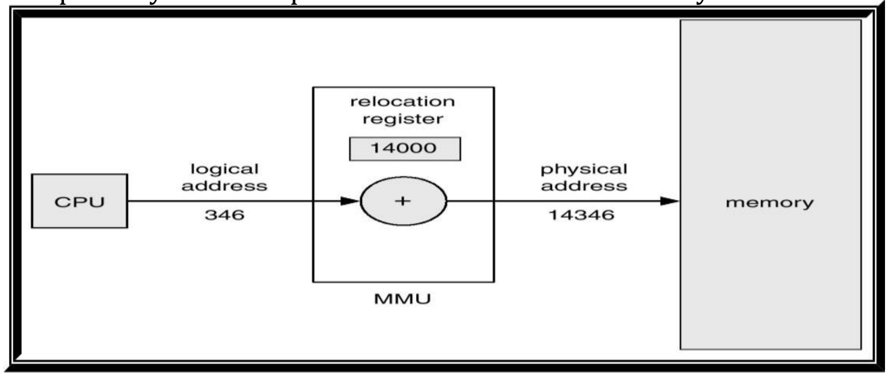
The address generated by the CPU is called logical address or virtual address.

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.
User program never sees the real physical address.

The figure below shows the dynamic relation. Dynamic relation implies mapping from virtual address space to physical address space and is performed at run time usually with some hardware assistance.

Relocation 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 re-location using a re-location registers 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 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.

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.

Advantage:-

- 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.

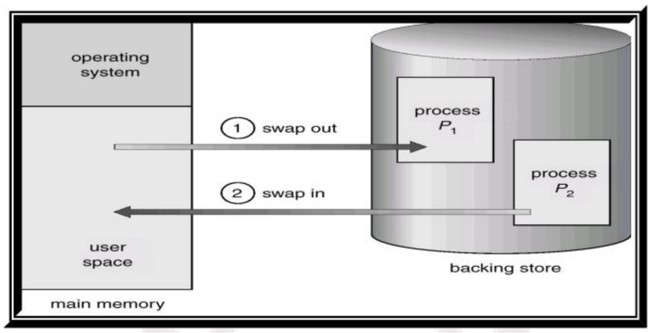
In dynamic linking only the main program is loaded in to the memory. If the main program requests a procedure, the procedure is loaded and the link is established at the time of references. This linking is postponed until the execution time.

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".

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:-

- To swap a process, it should be completely idle.
- 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.

Contiguous Memory Allocation:-

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. 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.

The set of holes are searched to determine which hole is best to allocate. There are three strategies to select a free hole:-
First fit:- 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 fit:- 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.

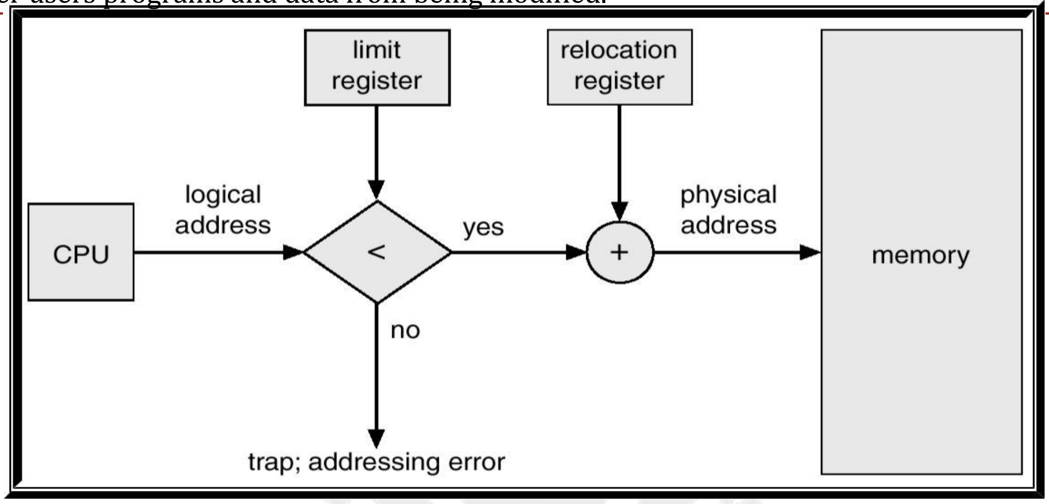
All these algorithms suffer from fragmentation.

Memory mapping and 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.

Re-location 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.

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 re-location 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.

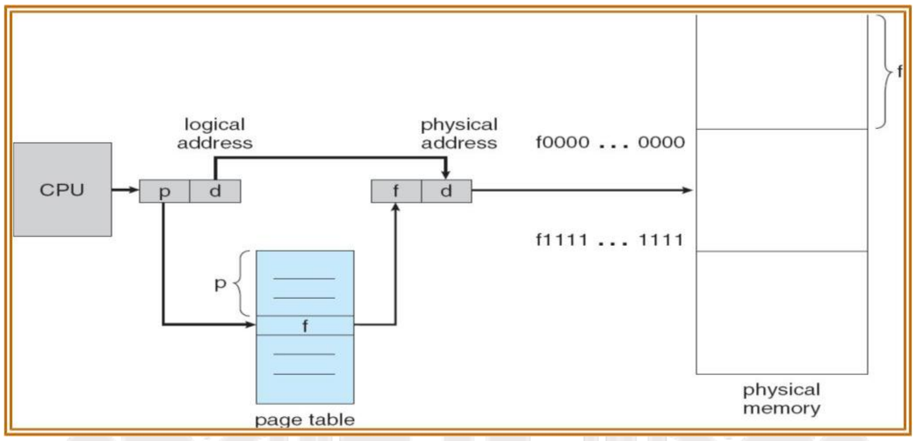
Paging:-
Paging is a memory management scheme that permits the physical address space of a process to be non-contiguous. Support for paging is handled by hardware. It is used to avoid external fragmentation.
Paging avoids the considerable problem of fitting the varying sized memory chunks on to the backing 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).
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.

The following figure shows paging hardware:-

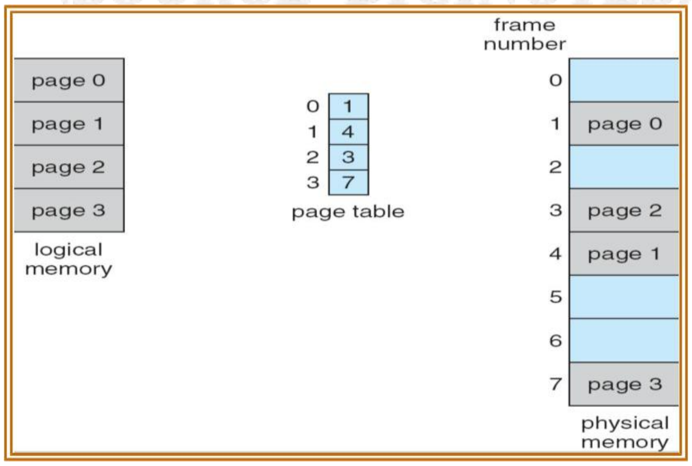


Logical address generated by the CPU is divided in to two parts: page number (p) and page offset (d).

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].

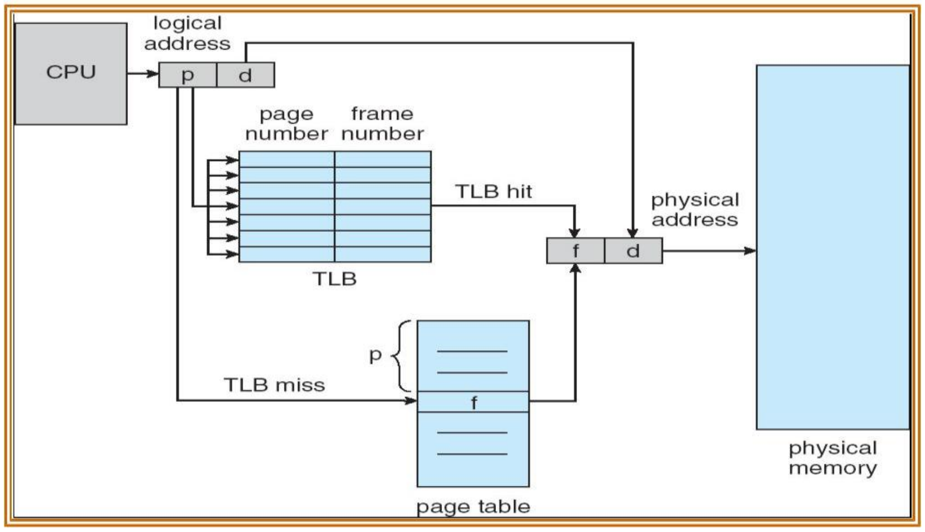
c. Logical address 4 is page 1 and offset 0 and page 1 is mapped to frame 6. So logical address 4 maps to physical address 24 [(6*4) + 0].
d. Logical address 13 is page 3 and offset 1 and page 3 is mapped to frame 2. So logical address 13 maps to physical address 9 [(2*4) + 1].

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.

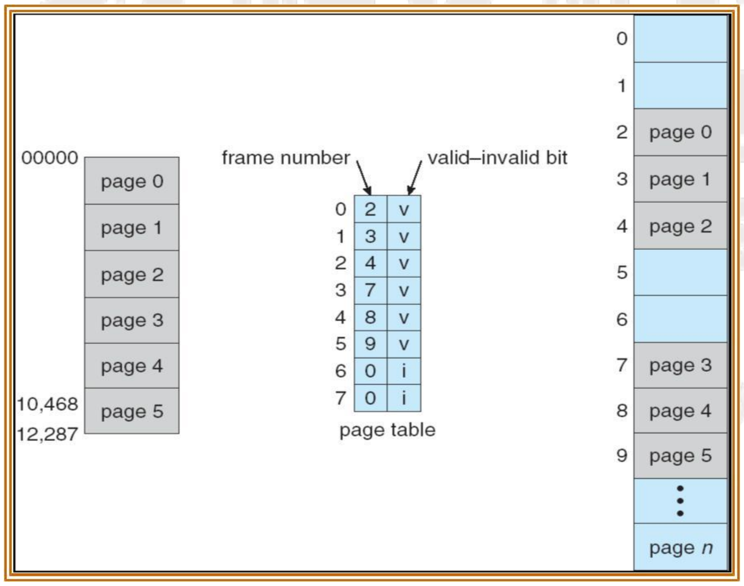


When an associative register is presented with an item, it is compared with all the key values, if found the corresponding value field is return and searching is fast. TLB is used with the page table as follows:-
TLB contains only few page table entries.
When a logical address is generated by the CPU, 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.

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.

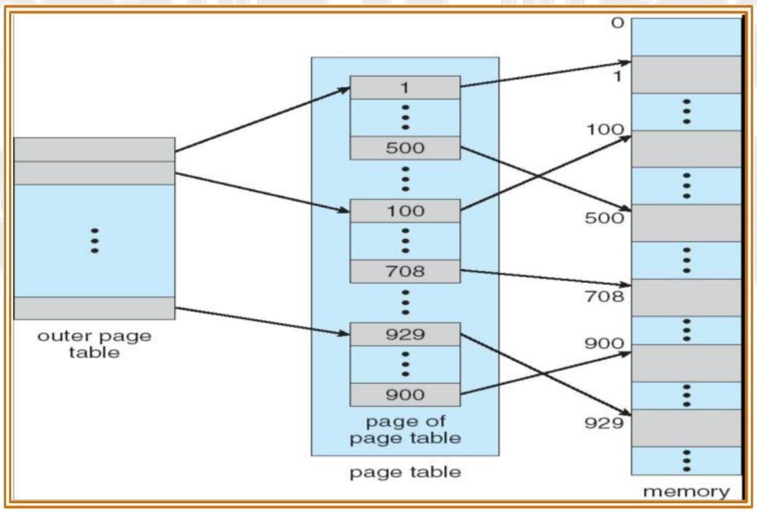
Structure of the Page Table:-

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

Page number		page offset
P1	P2	d
10	10	12



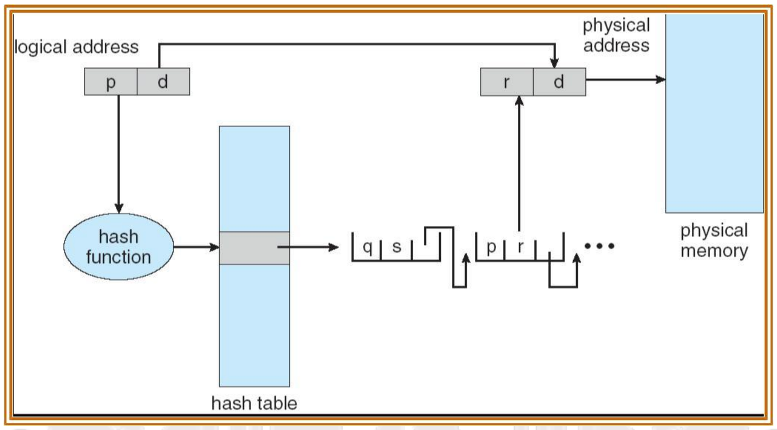
Hashed page table:-

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:-

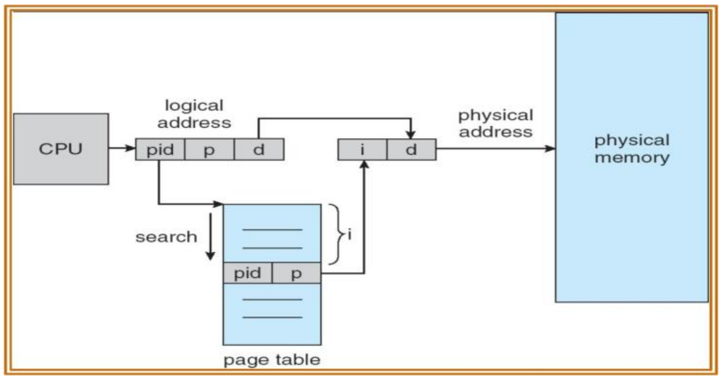
1) Virtual page number
2) Mapped page frame value
3) Pointer to the next element in the linked list

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.



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:-

i. Eliminates fragmentation.
ii. Support high degree of multiprogramming.
iii. Increases memory and processor utilization.
iv. Compaction overhead required for the re-locatable partition scheme is also eliminated.

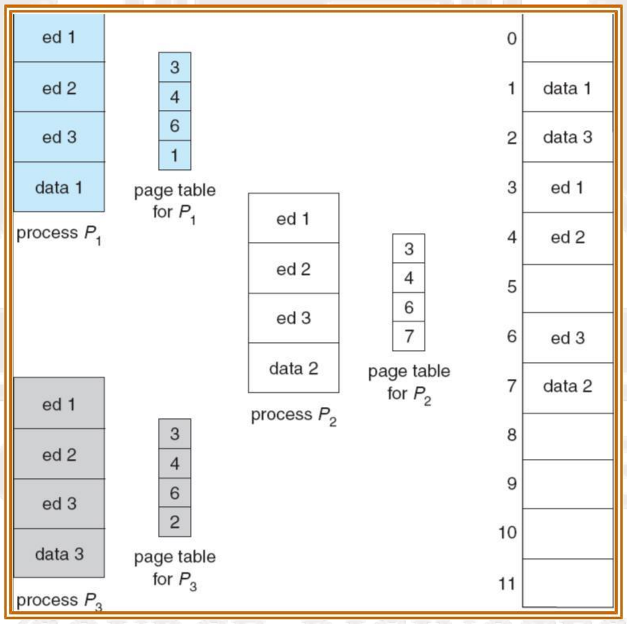
Disadvantage:-

- Page address mapping hardware increases the cost of the computer.
- Memory must be used to store the various tables like page tables, memory map table etc.
- 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.

Shared Pages:-

Another advantage of paging is the possibility of sharing common code. This is useful in time-sharing 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.

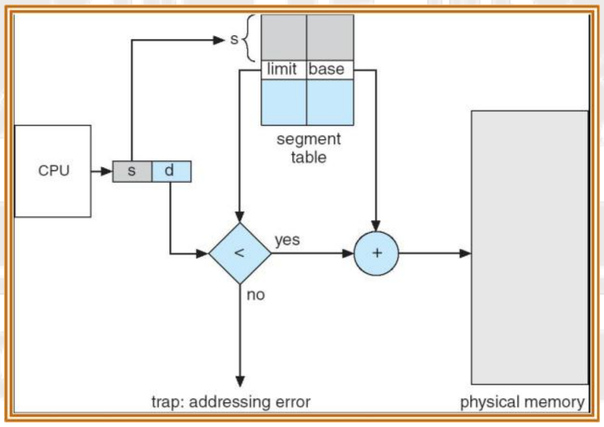
Segmentation:-

Basic method:-

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.
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.
The users specifies address by using two quantities: a segment name and an offset.
For simplicity the segments are numbered and referred by a segment number. So the logical address consists of <segment number, offset>.

Hardware support:-

We must define an implementation to map 2D user defined address in to 1D physical address.
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.



The use of segment table is shown in the above figure:-

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

Protection and Sharing:-

A particular advantage of segmentation is the association of protection with the segments.

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.

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.
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.
We can also share parts of a program.

Advantages:-

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

Differences between segmentation and paging:-

Segmentation:-
Program is divided in to variable sized segments.
User is responsible for dividing the program in to segments.
Segmentation is slower than paging.
Visible to user.
Eliminates internal fragmentation.
Suffers from external fragmentation.
Process or user segment number, offset to calculate absolute address.

Paging:-
Programs are divided in to fixed size pages.
Division is performed by the OS.
Paging is faster than segmentation.
Invisible to user.
Suffers from internal fragmentation.
No external fragmentation.
Process or user page number, offset to calculate absolute address.