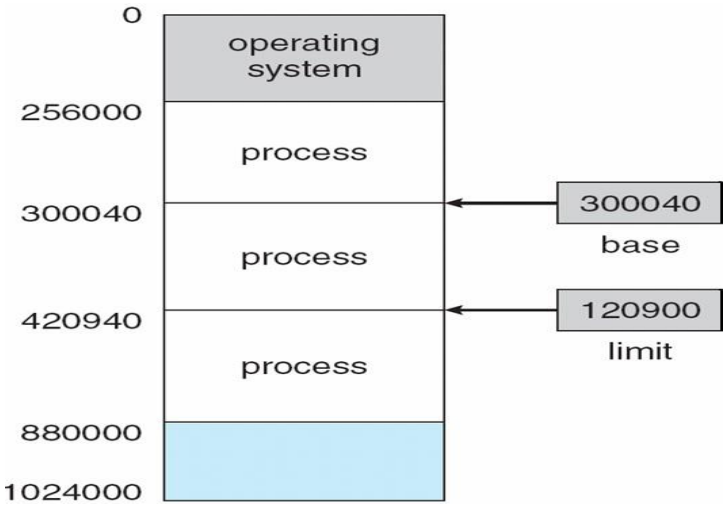
UNIT III

Memory Management

- To provide a detailed description of various ways of organizing memory hardware
- To discuss various memory-management techniques, including paging and segmentation
- To provide a detailed description of the Intel Pentium, which supports both pure segmentation and segmentation with paging
- Program must be brought (from disk) into memory and placed within a process for it to be run
- Main memory and registers are only storage CPU can access directly
- Register access in one CPU clock (or less)
- Main memory can take many cycles
- **Cache** sits between main memory and CPU registers
- Protection of memory required to ensure correct operation

Base and Limit Registers

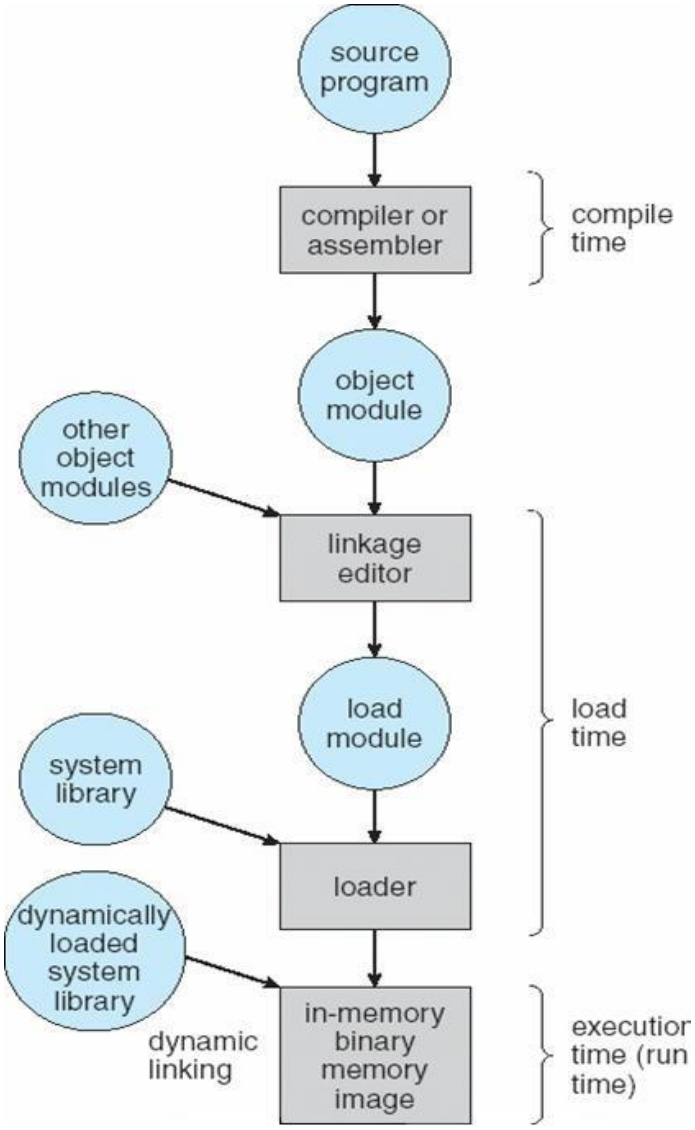
A pair of **base** and **limit** registers define the logical address space



Binding of Instructions and Data to Memory

- Address binding of instructions and data to memory addresses can happen at three different stages
- **Compile time**: If memory location known a priori, **absolute code** can be generated; must recompile code if starting location changes
- **Load time**: Must generate **relocatable code** if memory location is not known at compile time
- **Execution time**: Binding delayed until run time if the process can be moved during its execution from one memory segment to another. Need hardware support for address maps (e.g., base and limit registers)

Multistep Processing of a User Program



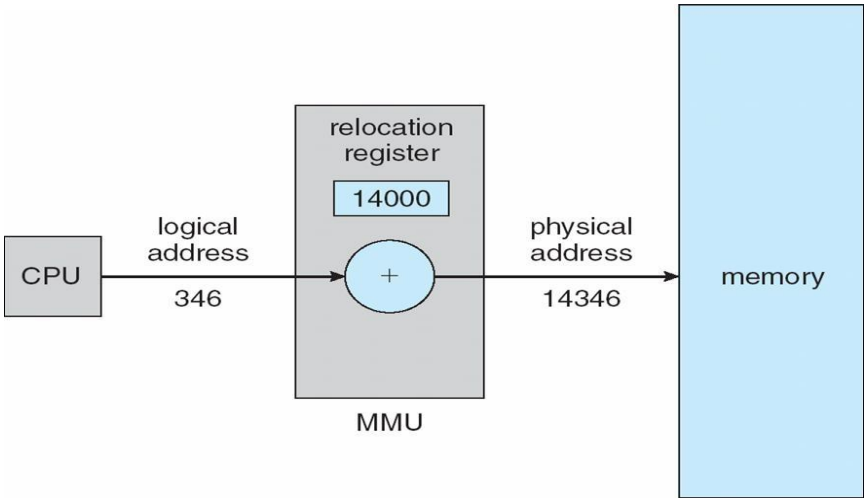
Logical vs. Physical Address Space

- The concept of a logical address space that is bound to a separate **physical address space** is central to proper memory management
- **Logical address** – generated by the CPU; also referred to as **virtual address**
- **Physical address** – address seen by the memory unit
- Logical and physical addresses are the same in compile-time and load-time address-binding schemes; logical (virtual) and physical addresses differ in execution-time address-binding scheme

Memory-Management Unit (MMU)

- Hardware device that maps virtual to physical address
- In MMU scheme, the value in the relocation register is added to every address generated by a user process at the time it is sent to memory
- The user program deals with *logical* addresses; it never sees the *real* physical addresses

Dynamic relocation using a relocation register



Dynamic Loading

- Routine is not loaded until it is called
- Better memory-space utilization; unused routine is never loaded
- Useful when large amounts of code are needed to handle infrequently occurring cases
- No special support from the operating system is required implemented through program design

Dynamic Linking

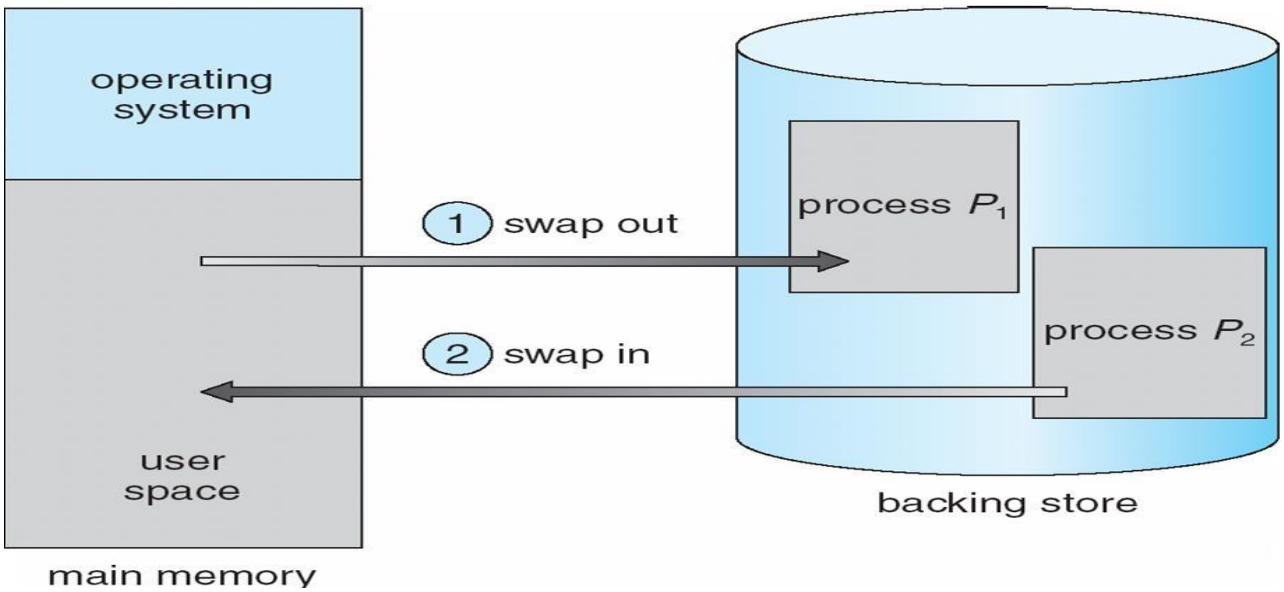
- Linking postponed until execution time
- Small piece of code, *stub*, used to locate the appropriate memory-resident library routine
- Stub replaces itself with the address of the routine, and executes the routine
- Operating system needed to check if routine is in processes' memory address
- Dynamic linking is particularly useful for libraries
- System also known as **shared libraries**

Swapping

A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued executionn**Backing store** – fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory imagesn**Roll out, roll in** – swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executednMajor part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swappednModified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)

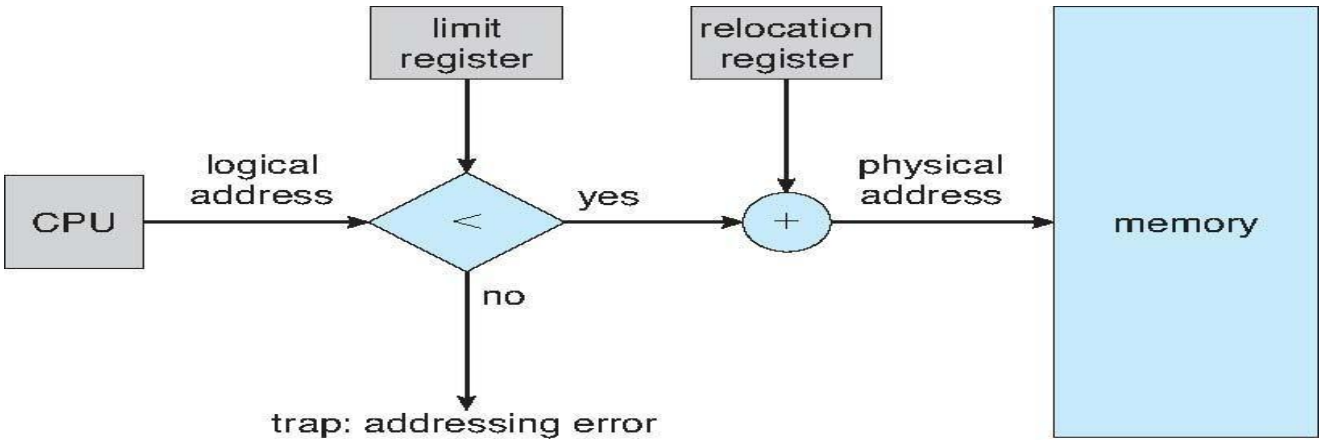
System maintains a **ready queue** of ready-to-run processes which have memory images on disk

Schematic View of Swapping



- Main memory usually into two partitions:
- Resident operating system, usually held in low memory with interrupt vector
- User processes then held in high memorynRelocation registers used to protect user processes from each other, and from changing operating-system code and data
- Base register contains value of smallest physical address
- Limit register contains range of logical addresses – each logical address must be less than the limit register
- MMU maps logical address *dynamically*

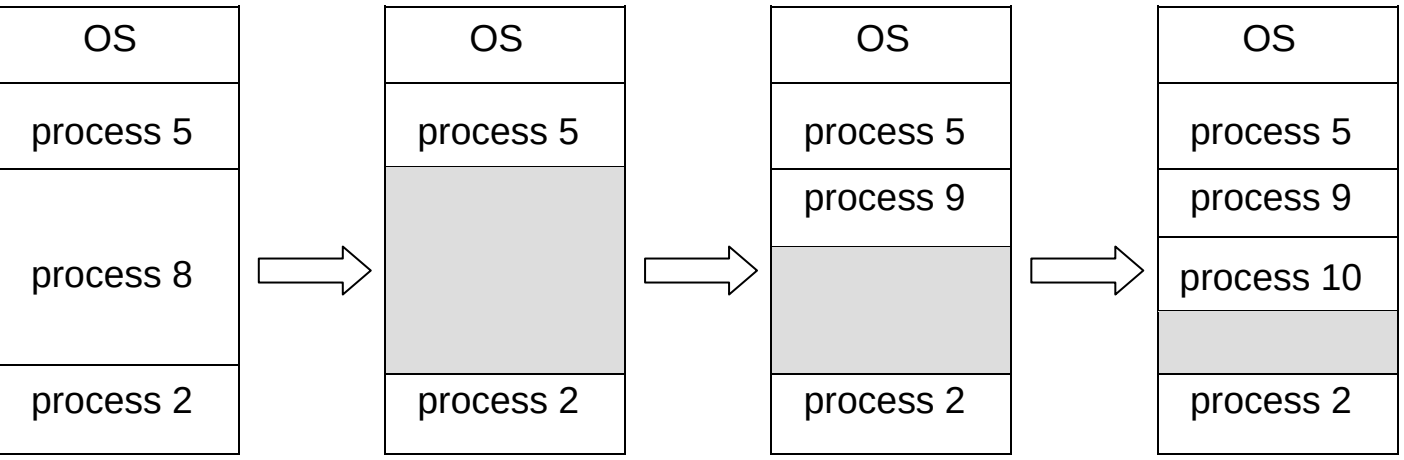
Hardware Support for Relocation and Limit Registers



- Multiple-partition allocation
- Hole – block of available memory; holes of various size are scattered throughout memory
- When a process arrives, it is allocated memory from a hole large enough to accommodate it

- Operating system maintains information about:

- a) allocated partitions b) free partitions (hole)



Dynamic Storage-Allocation Problem

- **First-fit**: Allocate the *first* hole that is big enough
- **Best-fit**: Allocate the *smallest* hole that is big enough; must search entire list, unless ordered by size Produces the smallest leftover hole
- **Worst-fit**: Allocate the *largest* hole; must also search entire list
- Produces the largest leftover hole
- First-fit and best-fit better than worst-fit in terms of speed and storage utilization

Fragmentation

- **External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous
- **Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used
- Reduce external fragmentation by **compaction**
- Shuffle memory contents to place all free memory together in one large block
- Compaction is possible *only* if relocation is dynamic, and is done at execution time.
- I/O problem
    - Latch job in memory while it is involved in I/O
    - Do I/O only into OS buffers

Paging

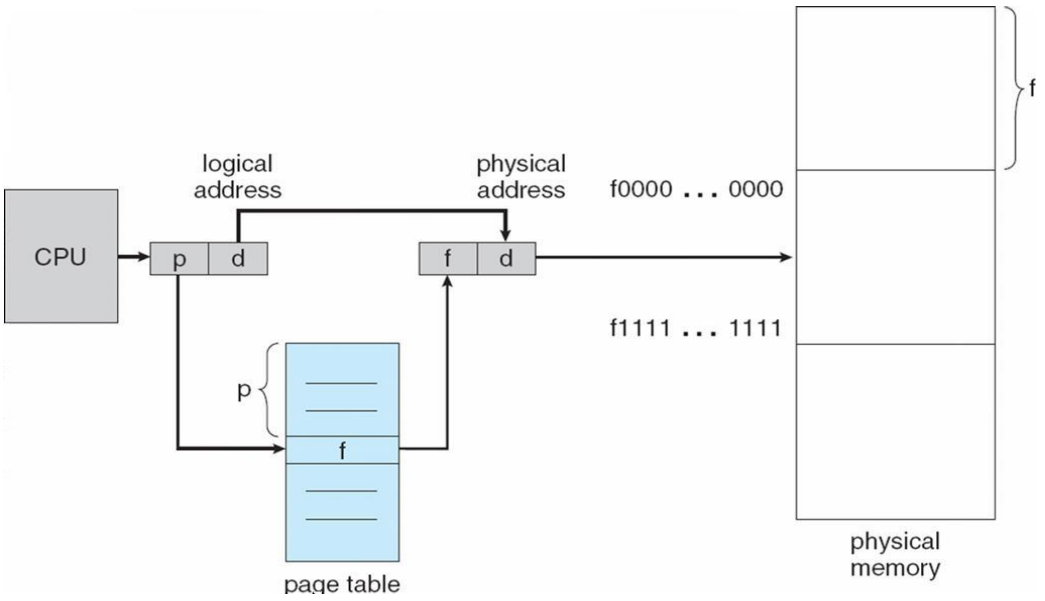
- Logical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available
- Divide physical memory into fixed-sized blocks called **frames** (size is power of 2, between 512 bytes and 8,192 bytes)
- Divide logical memory into blocks of same size called **pages**nKeep track of all free frames

- To run a program of size ***n*** pages, need to find *n* free frames and load program
- Set up a page table to translate logical to physical addresses
- Internal fragmentation

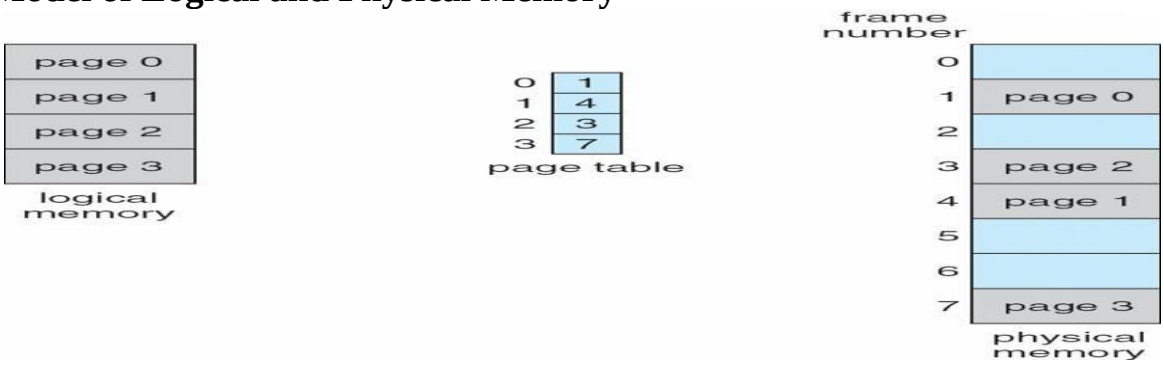
Address Translation Scheme

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

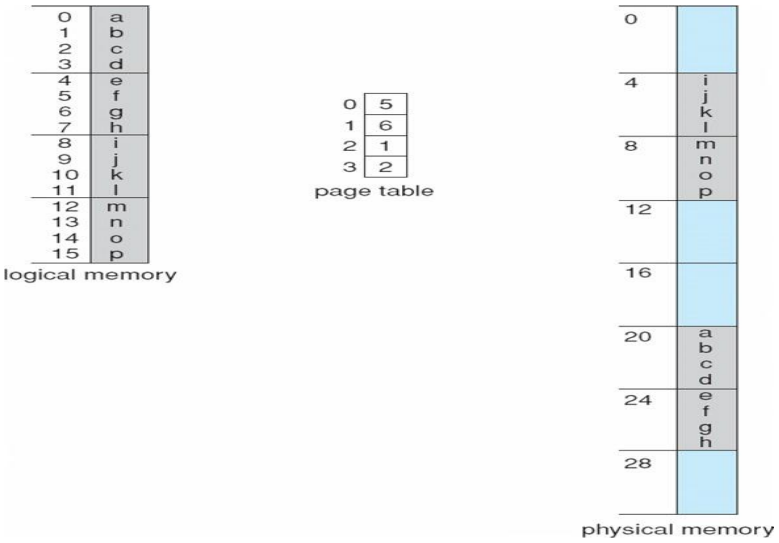
Paging Hardware



Paging Model of Logical and Physical Memory

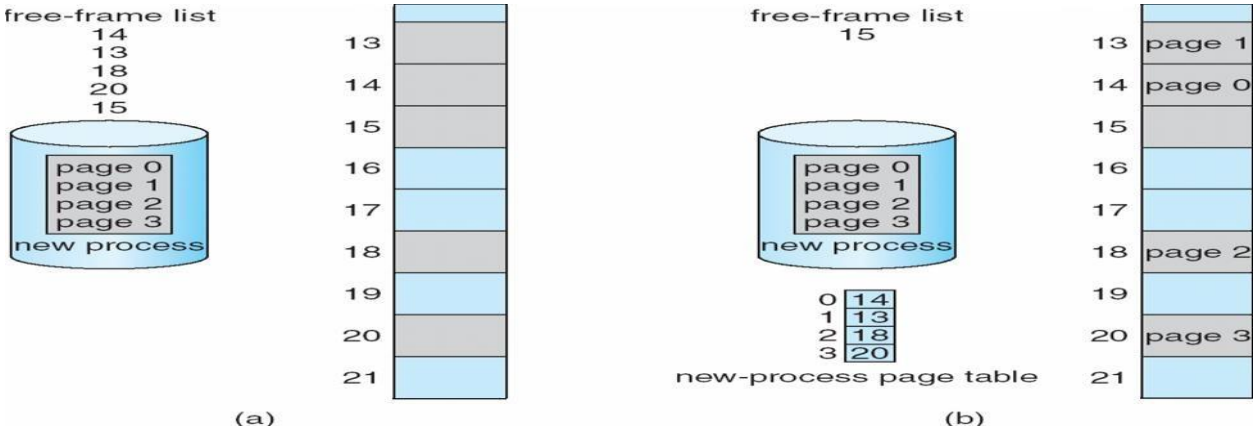


Paging Example



32-byte memory and 4-byte pages

Free Frames



Implementation of Page Table

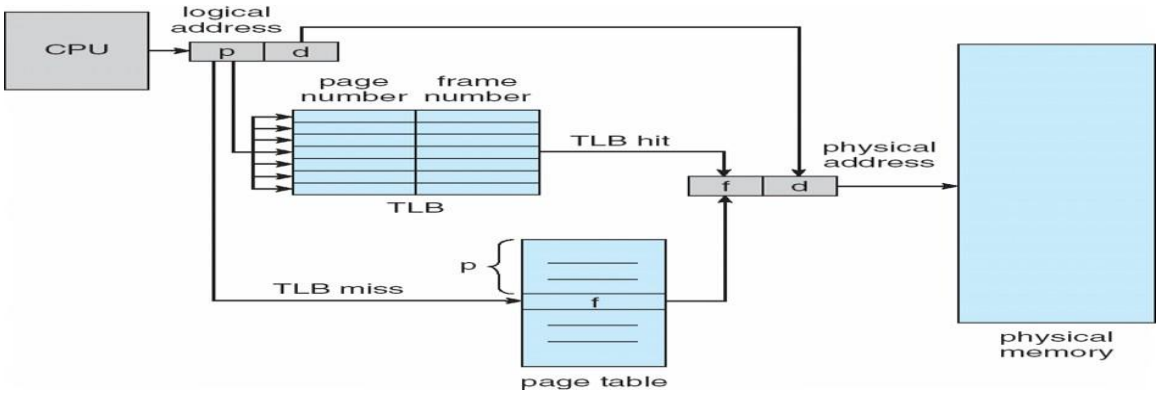
- Page table is kept in main memory
- **Page-table base register (PTBR)** points to the page table
- **Page-table length register (PRLR)** indicates size of the page table
- In this scheme every data/instruction access requires two memory accesses. One for the page table and one for the data/instruction.
- The two memory access problem can be solved by the use of a special fast-lookup hardware cache called **associative memory** or **translation look-aside buffers (TLBs)**
- Some TLBs store **address-space identifiers (ASIDs)** in each TLB entry – uniquely identifies each process to provide address-space protection for that process

Associative Memory

- Associative memory – parallel search
- Address translation (p, d)
- If p is in associative register, get frame # out
- Otherwise get frame # from page table in memory

Page #	Frame #

Paging Hardware With TLB



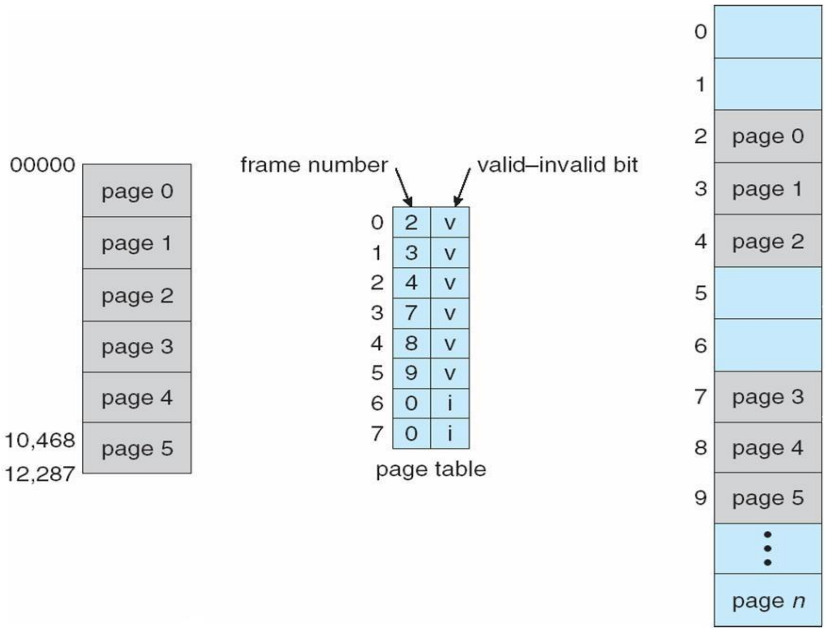
Effective Access Time

- Associative Lookup = e 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 = an **Effective Access Time** (EAT)

$$EAT = (1 + e)\, a + (2 + e)(1 - a)$$
$$= 2 + e - a$$

Memory Protection

- Memory protection implemented by associating protection bit with each frame
- **Valid-invalid** bit attached to each entry in the page table:
- "valid" indicates that the associated page is in the process' logical address space, and is thus a legal page
- "invalid" indicates that the page is not in the process' logical address space
- **Valid (v) or Invalid (i) Bit In A Page Table**



Shared Pages

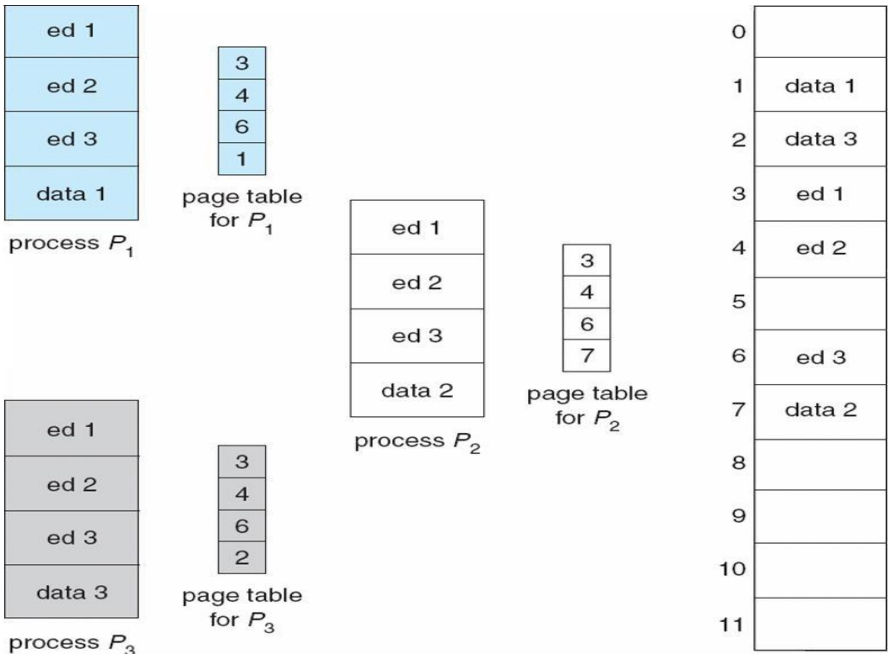
Shared code

- One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, window systems).
- Shared code must appear in same location in the logical address space of all processes

Private code and data

- Each process keeps a separate copy of the code and data
- The pages for the private code and data can appear anywhere in the logical address space

Shared Pages Example



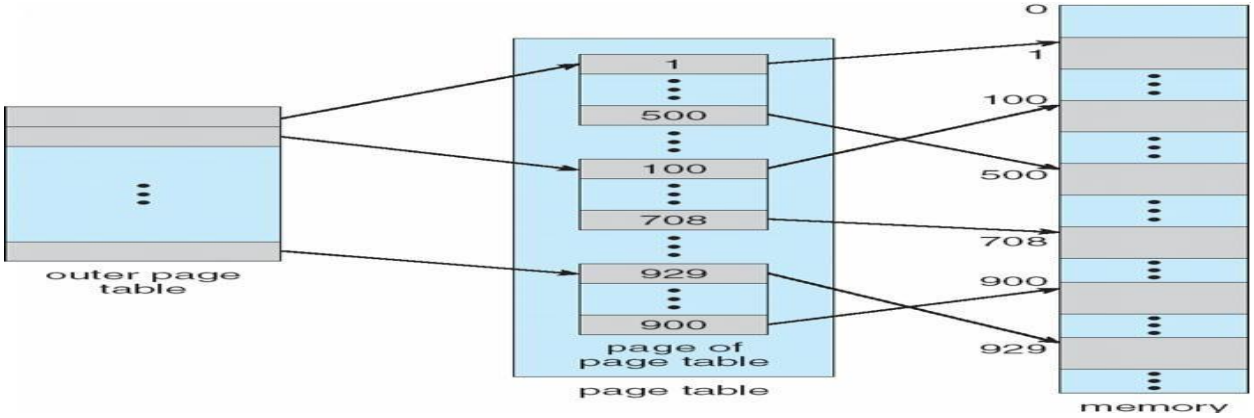
Structure of the Page Table

- Hierarchical Paging
- Hashed Page Tables
- Inverted Page Tables

Hierarchical Page Tables

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

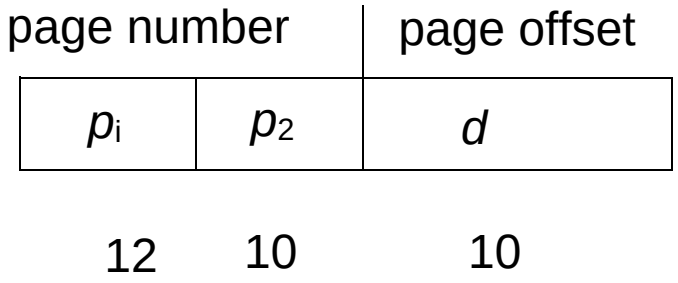
Two-Level Page-Table Scheme



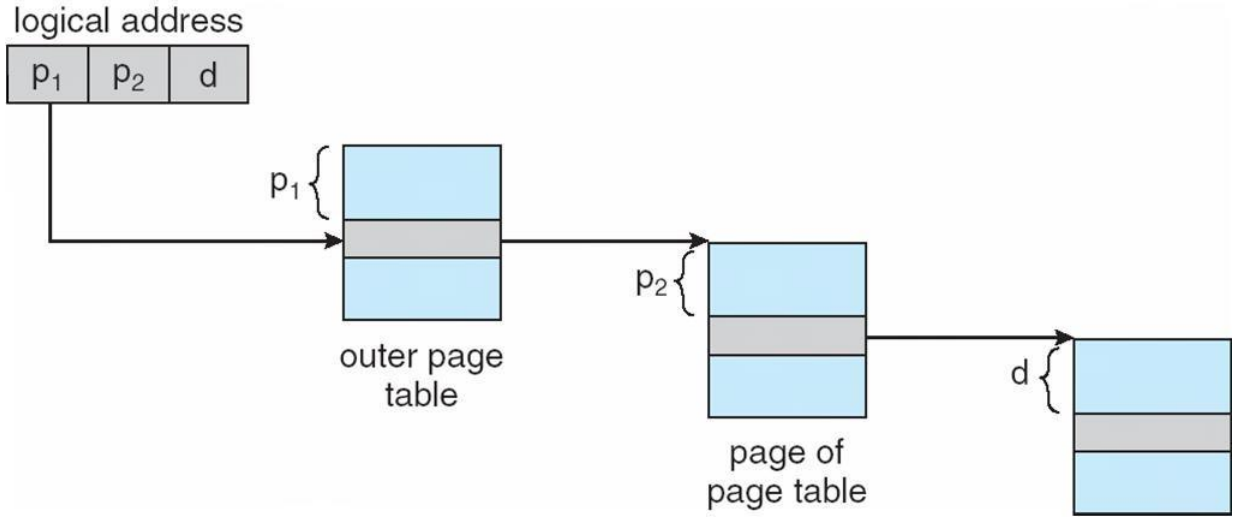
Two-Level Paging Example

- 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:

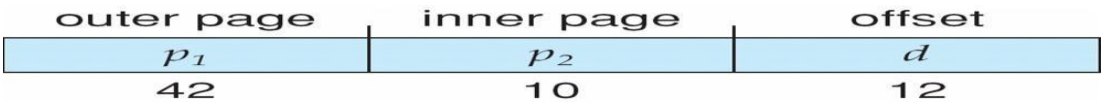
  where *pi* is an index into the outer page table, and *p2* is the displacement within the page of the outer page table

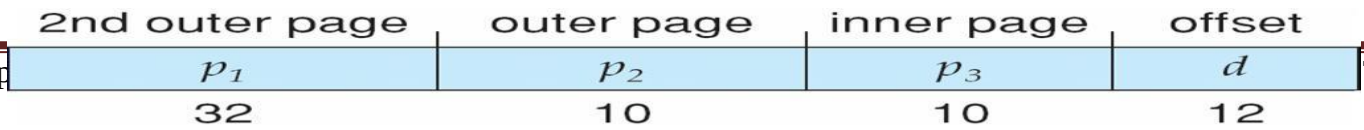


Address-Translation Scheme



Three-level Paging Scheme

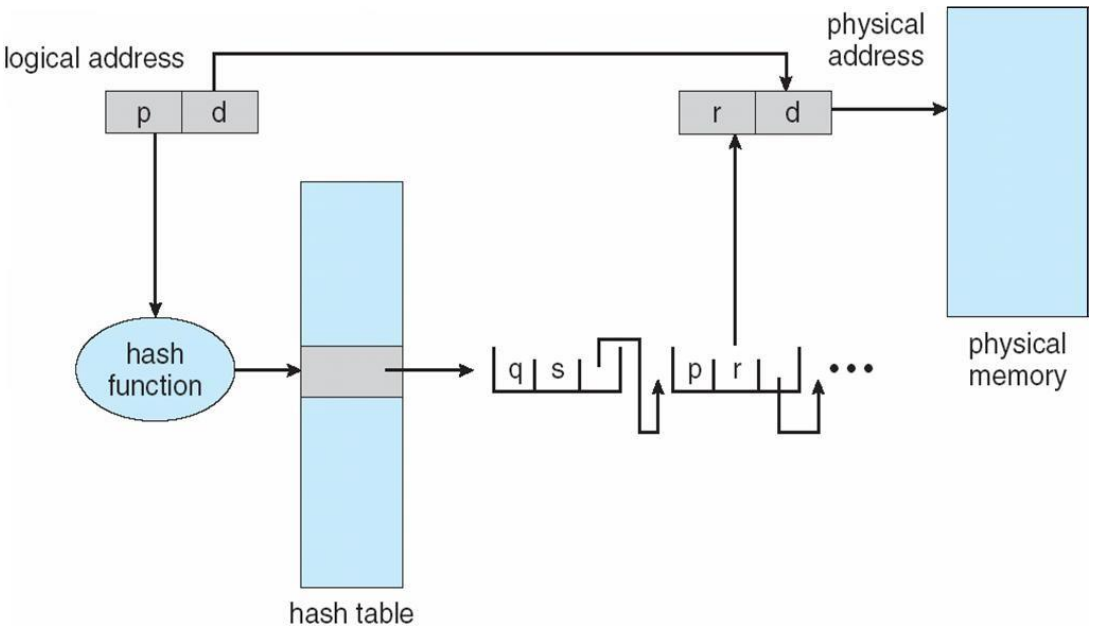




Hashed Page Tables

- Common in address spaces > 32 bits
- The virtual page number is hashed into a page table
- This page table contains a chain of elements hashing to the same location
- Virtual page numbers are compared in this chain searching for a match
- If a match is found, the corresponding physical frame is extracted

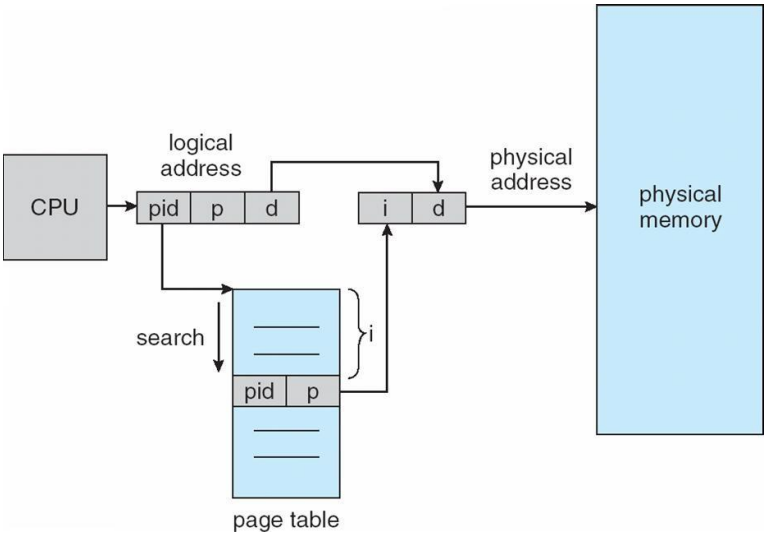
Hashed Page Table



Inverted Page Table

- One entry for each real page of memory
- Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page
- Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs
- Use hash table to limit the search to one — or at most a few — page-table entries

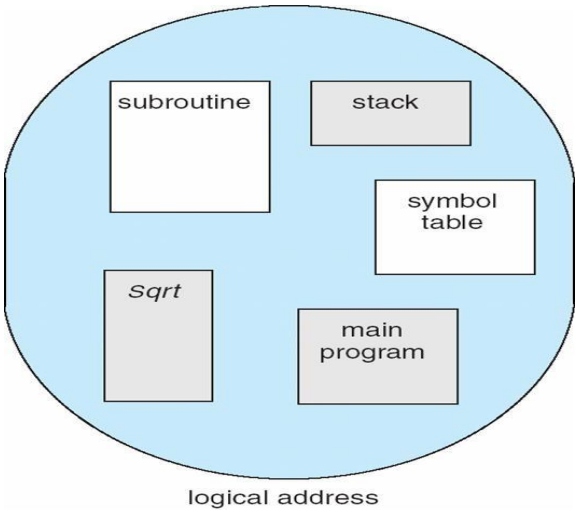
Inverted Page Table Architecture



Segmentation

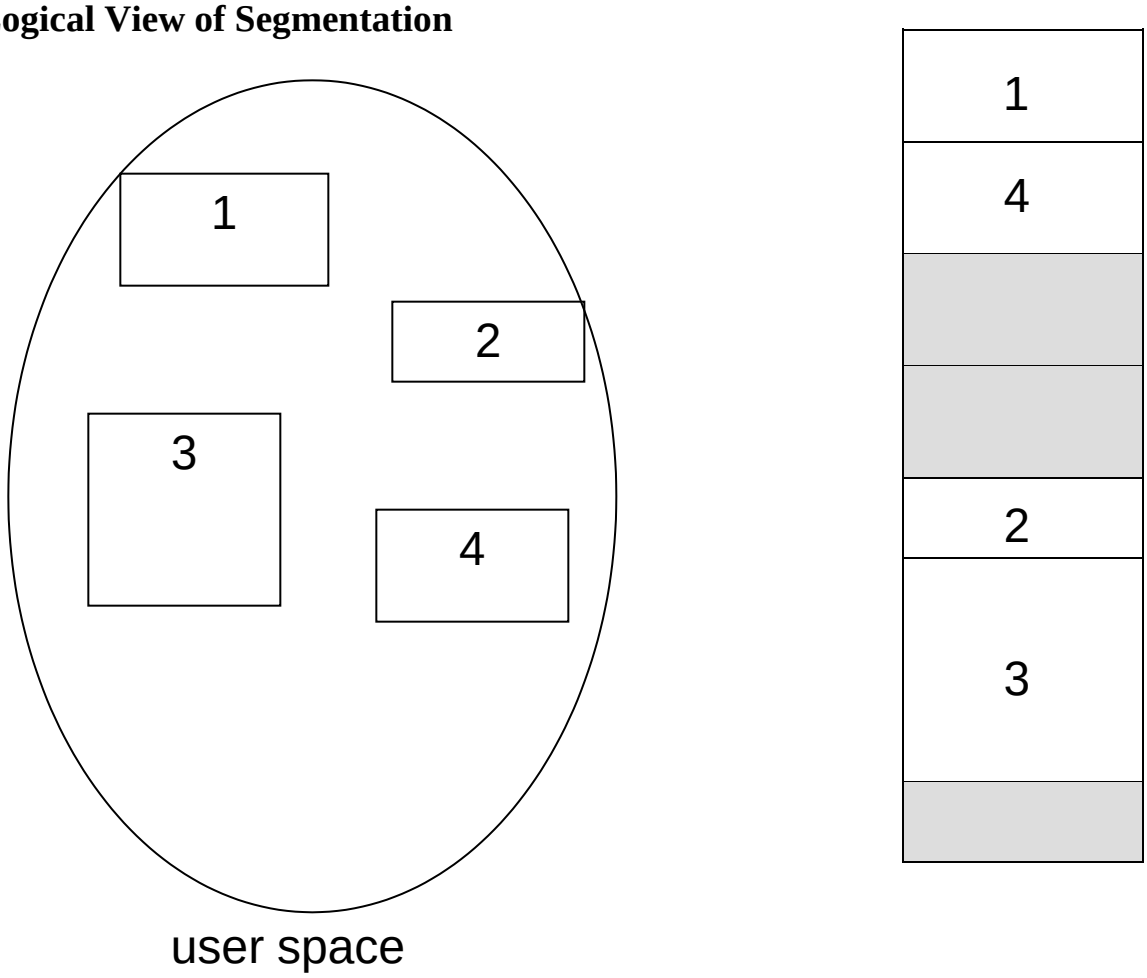
Memory-management scheme that supports user view of memory

- A program is a collection of segments
- A segment is a logical unit such as:
- main program
- procedure function
- method
- object
- local variables, global variables
- common block
- stack
- symbol table
- arrays



User's View of a Program

Logical View of Segmentation



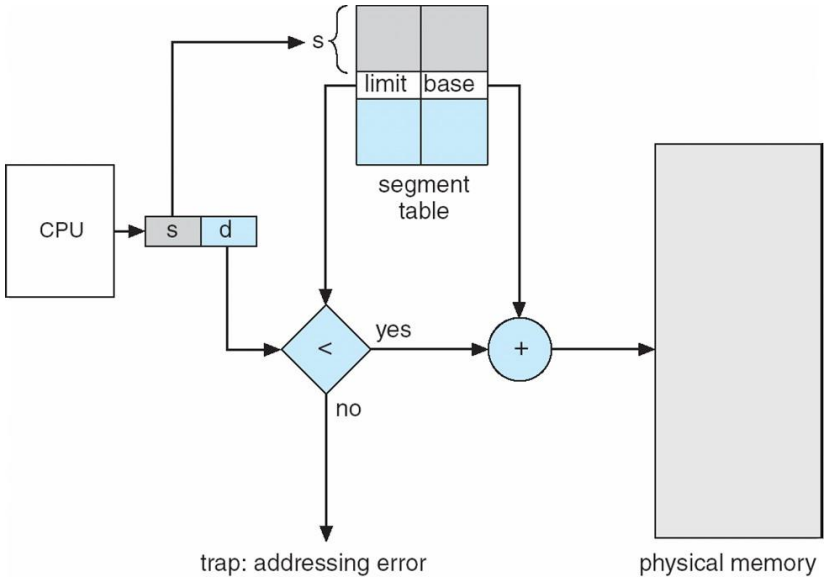
Segmentation Architecture

- 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;

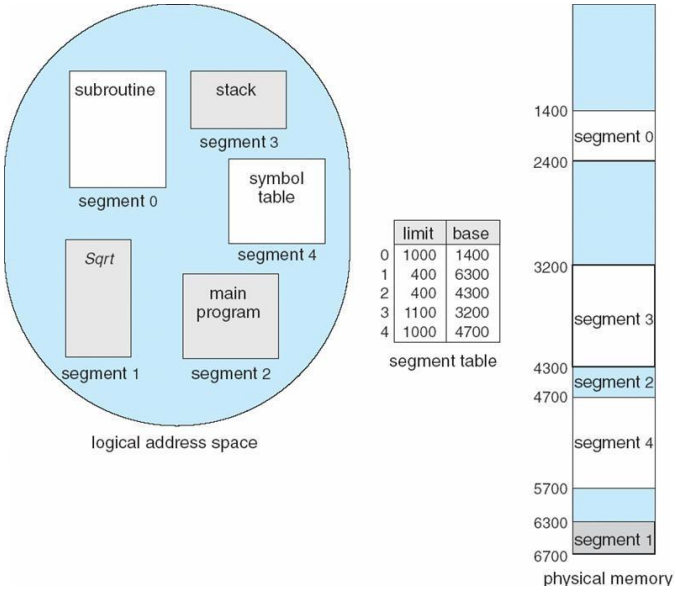
segment number ***s*** is legal if ***s*** < **STLR**

- Protection
- With each entry in segment table associate:
- □validation bit = 0 Þ illegal segment
- □read/write/execute privileges
- Protection bits associated with segments; code sharing occurs at segment level
- Since segments vary in length, memory allocation is a dynamic storage-allocation problem
- A segmentation example is shown in the following diagram

Segmentation Hardware



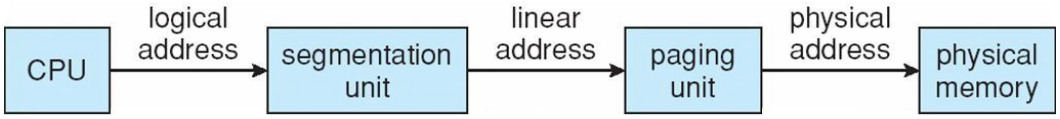
Example of Segmentation

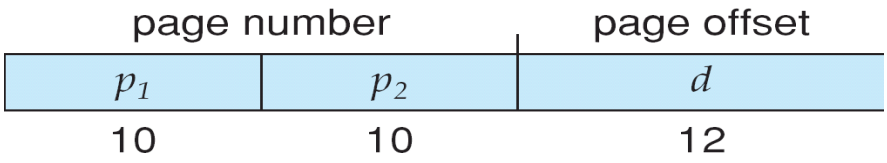


Example: The Intel Pentium

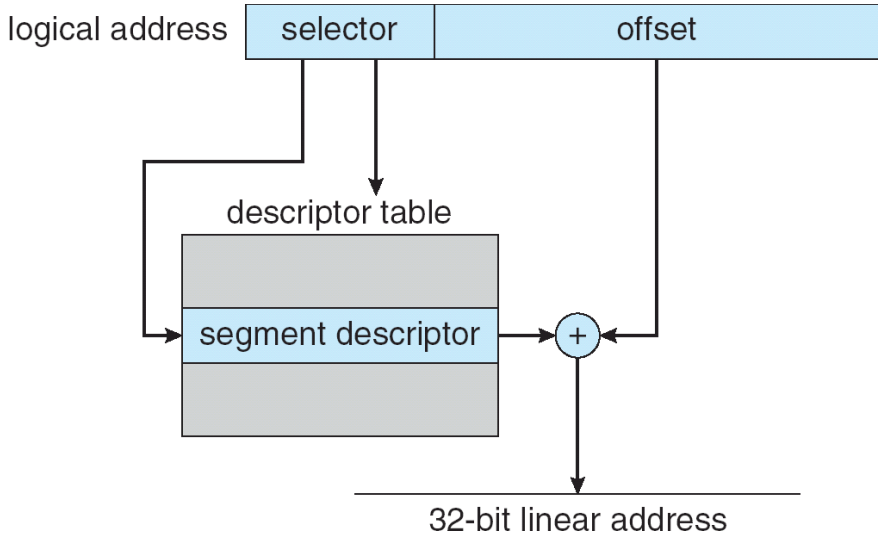
- Supports both segmentation and segmentation with paging
- CPU generates logical address
- Given to segmentation unit
  - Which produces linear addresses
- Linear address given to paging unit
  - Which generates physical address in main memory
  - Paging units form equivalent of MMU

Logical to Physical Address Translation in Pentium

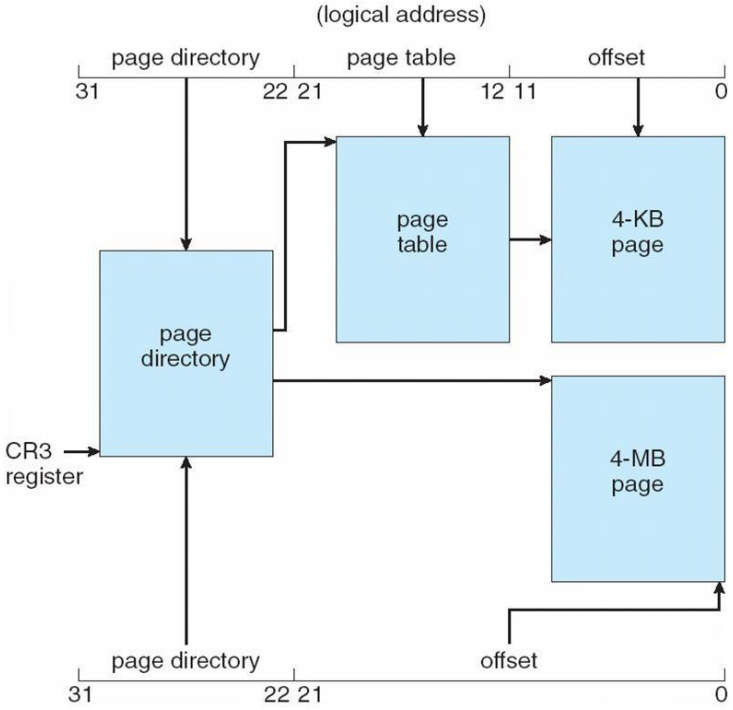




Intel Pentium Segmentation

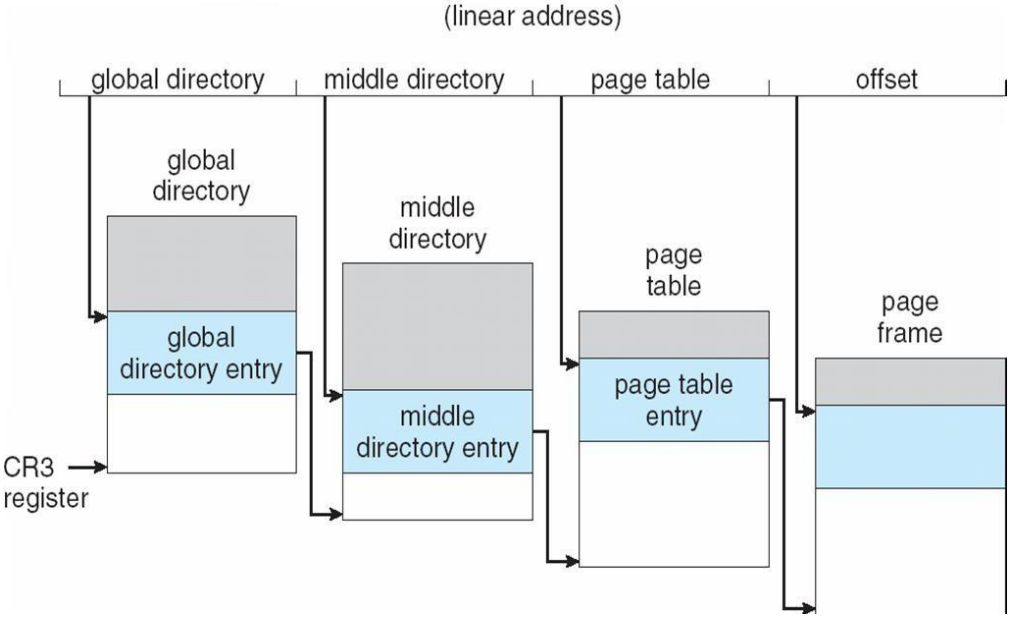


Pentium Paging Architecture



Linear Address in Linux

Three-level Paging in Linux



VIRTUAL MEMORY

Virtual Memory

- Virtual memory is a technique that allows the execution of process that may not be completely in memory. The main visible advantage of this scheme is that programs can be larger than physical memory.

- Virtual memory is the separation of user logical memory from physical memory this separation allows an extremely large virtual memory to be provided for programmers when only a smaller physical memory is available (Fig).
- Following are the situations, when entire program is not required to load fully.

1. User written error handling routines are used only when an error occurs in the data or computation.
2. Certain options and features of a program may be used rarely.
3. Many tables are assigned a fixed amount of address space even though only a small amount of the table is actually used.

- The ability to execute a program that is only partially in memory would counter many benefits.

1. Less number of I/O would be needed to load or swap each user program into memory.
2. A program would no longer be constrained by the amount of physical memory that is available.
3. Each user program could take less physical memory, more programs could be run the same time, with a corresponding increase in CPU utilization and throughput.

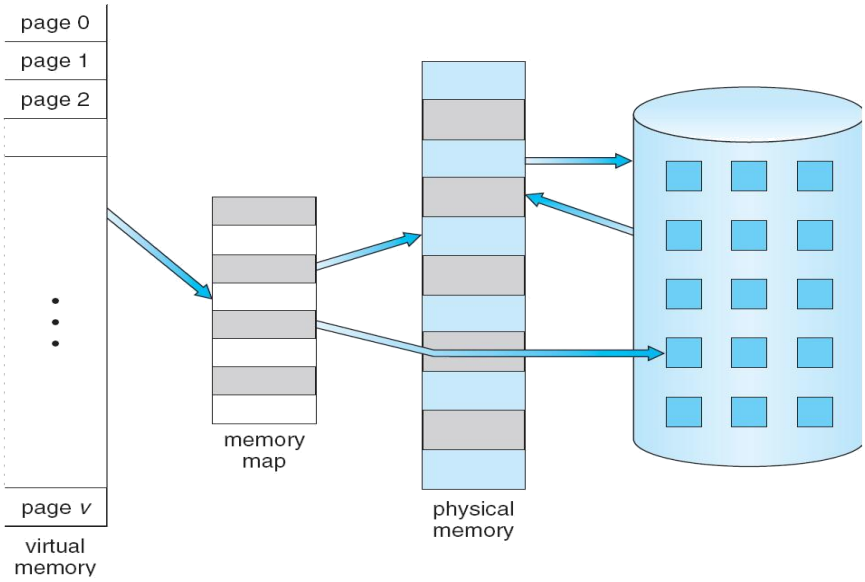


Fig. Diagram showing virtual memory that is larger than physical memory.

Virtual memory is commonly implemented by demand paging. It can also be implemented in a segmentation system. Demand segmentation can also be used to provide virtual memory.

Demand Paging

A demand paging is similar to a paging system with swapping(Fig 5.2). When we want to execute a process, we swap it into memory. Rather than swapping the entire process into memory.

When a process is to be swapped in, the pager guesses which pages will be used before the process is swapped out again Instead of swapping in a whole process, the pager brings only those necessary pages into memory. Thus, it avoids reading into memory pages that will not be used in anyway, decreasing the swap time and the amount of physical memory needed.

Hardware support is required to distinguish between those pages that are in memory and those pages that are on the disk using the valid-invalid bit scheme. Where valid and invalid pages can be checked checking the bit and marking a page will have no effect if the process never attempts to access the pages. While the process executes and accesses pages that are memory resident, execution proceeds normally.

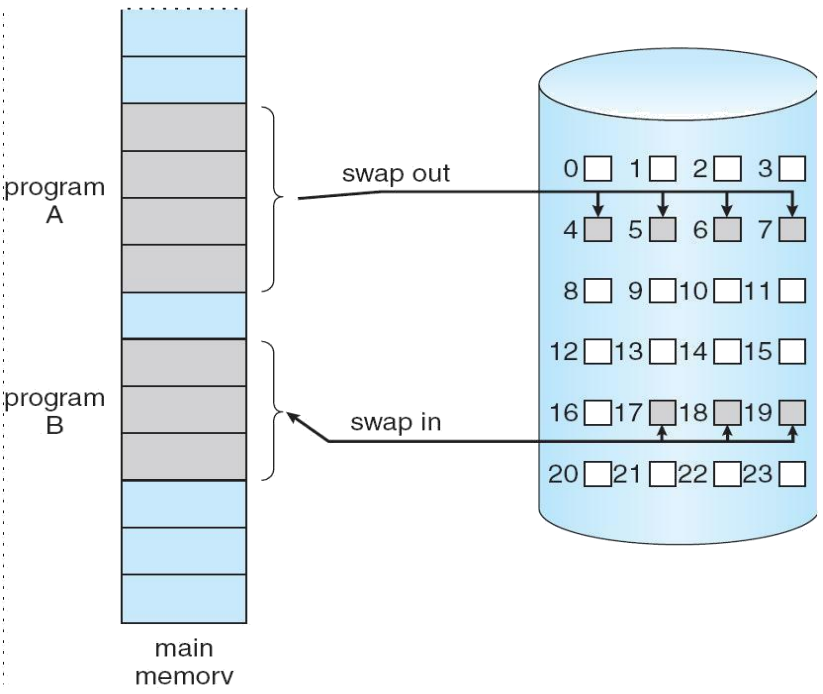


Fig. Transfer of a paged memory to continuous disk space

Access to a page marked invalid causes a page-fault trap. This trap is the result of the operating system's failure to bring the desired page into memory. But page fault can be handled as following (Fig 5.3):

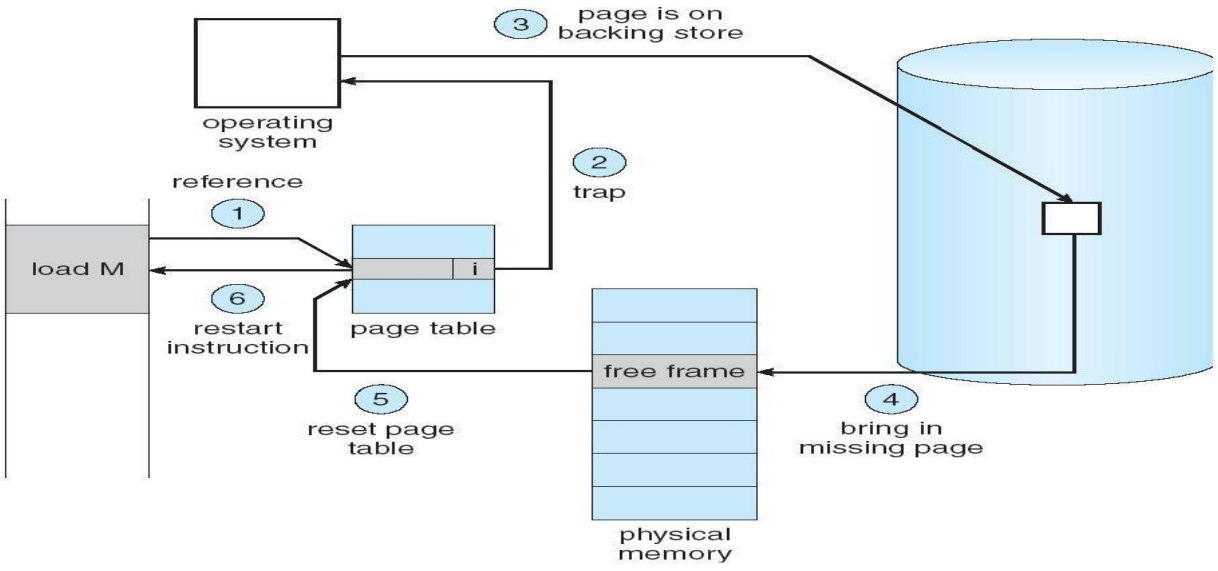


Fig. Steps in handling a page fault

1. We check an internal table for this process to determine whether the reference was a valid or invalid memory access.
2. If the reference was invalid, we terminate the process. If .it was valid, but we have not yet brought in that page, we now page in the latter.
3. We find a free frame.

4. We schedule a disk operation to read the desired page into the newly allocated frame.

5. When the disk read is complete, we modify the internal table kept with the process and the page table to indicate that the page is now in memory.

6. We restart the instruction that was interrupted by the illegal address trap. The process can now access the page as though it had always been memory.

Therefore, the operating system reads the desired page into memory and restarts the process as though the page had always been in memory.

The page replacement is used to make the frame free if they are not in used. If no frame is free then other process is called in.

Advantages of Demand Paging:

1. Large virtual memory.
2. More efficient use of memory.
3. Unconstrained multiprogramming. There is no limit on degree of multiprogramming.

Disadvantages of Demand Paging:

4. Number of tables and amount of processor over head for handling page interrupts are greater than in the case of the simple paged management techniques.
5. due to the lack of an explicit constraints on a jobs address space size.

Page Replacement Algorithm

There are many different page replacement algorithms. We evaluate an algorithm by running it on a particular string of memory reference and computing the number of page faults. The string of memory references is called reference string. Reference strings are generated artificially or by tracing a given system and recording the address of each memory reference. The latter choice produces a large number of data.

1. For a given page size we need to consider only the page number, not the entire address.

2. if we have a reference to a page p, then any immediately following references to page p will never cause a page fault. Page p will be in memory after the
first reference; the immediately following references will not fault.

Eg:- consider the address sequence

0100, 0432, 0101, 0612, 0102, 0103, 0104, 0101, 0611, 0102, 0103, 0104, 0101, 0610, 0102,

0103, 0104, 0104, 0101, 0609, 0102, 0105
and reduce to 1, 4, 1, 6,1, 6, 1, 6, 1, 6, 1

To determine the number of page faults for a particular reference string and page replacement algorithm, we also need to know the number of page frames available. As the number of frames available increase, the number of page faults will decrease.

FIFO Algorithm

The simplest page-replacement algorithm is a FIFO algorithm. A FIFO replacement algorithm associates with each page the time when that page was brought into memory. When a page must be replaced, the oldest page is chosen. We can create a FIFO queue to hold all pages in memory.

The first three references (7, 0, 1) cause page faults, and are brought into these empty eg. 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1 and consider 3 frames. This replacement means that the next reference to 0 will fault. Page 1 is then replaced by page 0.

Optimal Algorithm

An optimal page-replacement algorithm has the lowest page-fault rate of all algorithms. An optimal page-replacement algorithm exists, and has been called OPT or MIN. It is simply

Replace the page that will not be used for the longest
period of time.

Now consider the same string with 3 empty frames.

The reference to page 2 replaces page 7, because 7 will not be used until reference 15, whereas page 0 will be used at 5, and page 1 at 14. The reference to page 3 replaces page 1, as page 1 will be the last of the three pages in memory to be referenced again. Optimal replacement is much better than a FIFO.

The optimal page-replacement algorithm is difficult to implement, because it requires future knowledge of the reference string.

LRU Algorithm

The FIFO algorithm uses the time when a page was brought into memory; the OPT algorithm uses the time when a page is to be used. In LRU replace the page that has not been used for the longest period of time.

LRU replacement associates with each page the time of that page's last use. When a page must be replaced, LRU chooses that page that has not been used for the longest period of time.

Let S^R be the reverse of a reference string S, then the page-fault rate for the OPT algorithm on S is the same as the page-fault rate for the OPT algorithm on S^R.

LRU Approximation Algorithms

Some systems provide no hardware support, and other page-replacement algorithm. Many systems provide some help, however, in the form of a reference bit. The reference bit for a page is set, by the hardware, whenever that page is referenced. Reference bits are associated with each entry in the page table Initially, all bits are cleared (to 0) by the operating system. As a user process executes, the bit associated with each page referenced is set (to 1) by the hardware.

Additional-Reference-Bits Algorithm

The operating system shifts the reference bit for each page into the high-order or of its 5-bit byte, shifting the other bits right 1 bit, discarding the low-order bit.

These 5-bit shift registers contain the history of page use for the last eight time periods. If the shift register contains 00000000, then the page has not been

used for eight time periods; a page that is used at least once each period would have a shift register value of 11111111.

Second-Chance Algorithm

The basic algorithm of second-chance replacement is a FIFO replacement algorithm. When a page gets a second chance, its reference bit is cleared and its arrival e is reset to the current time.

Enhanced Second-Chance Algorithm

The second-chance algorithm described above can be enhanced by considering troth the reference bit and the modify bit as an ordered pair.

1. (0,0) neither recently used nor modified best page to replace.
2. (0,1) not recently used but modified not quite as good, because the page will need to be written out before replacement.
3. (1,0) recently used but clean probably will be used again soon.
4. (1,1) recently used and modified probably will be used again, and write out will be needed before replacing it

Counting Algorithms

There are many other algorithms that can be used for page replacement.

• **LFU Algorithm:** The least frequently used (LFU) page-replacement algorithm requires that the page with the smallest count be replaced. This algorithm suffers from the situation in which a page is used heavily during the initial phase of a process, but then is never used again.

• **MFU Algorithm:** The most frequently used (MFU) page-replacement algorithm is based on the argument that the page with the smallest count was probably just brought in and has yet to be used.

Page Buffering Algorithm

When a page fault occurs, a victim frame is chosen as before. However, the desired page is read into a free frame from the pool before the victim is written out.

This procedure allows the process to restart as soon as possible, without waiting for the victim page to be written out. When the victim is later written out, its frame is added to the free-frame pool.

When the FIFO replacement algorithm mistakenly replaces a page mistakenly replaces a page that is still in active use, that page is quickly retrieved from the free-frame buffer, and no I/O is necessary. The free-frame buffer provides protection against the relatively poor, but simple, FIFO replacement algorithm.

Disk scheduling is done by operating systems to schedule I/O requests arriving for the disk. Disk scheduling is also known as I/O scheduling.

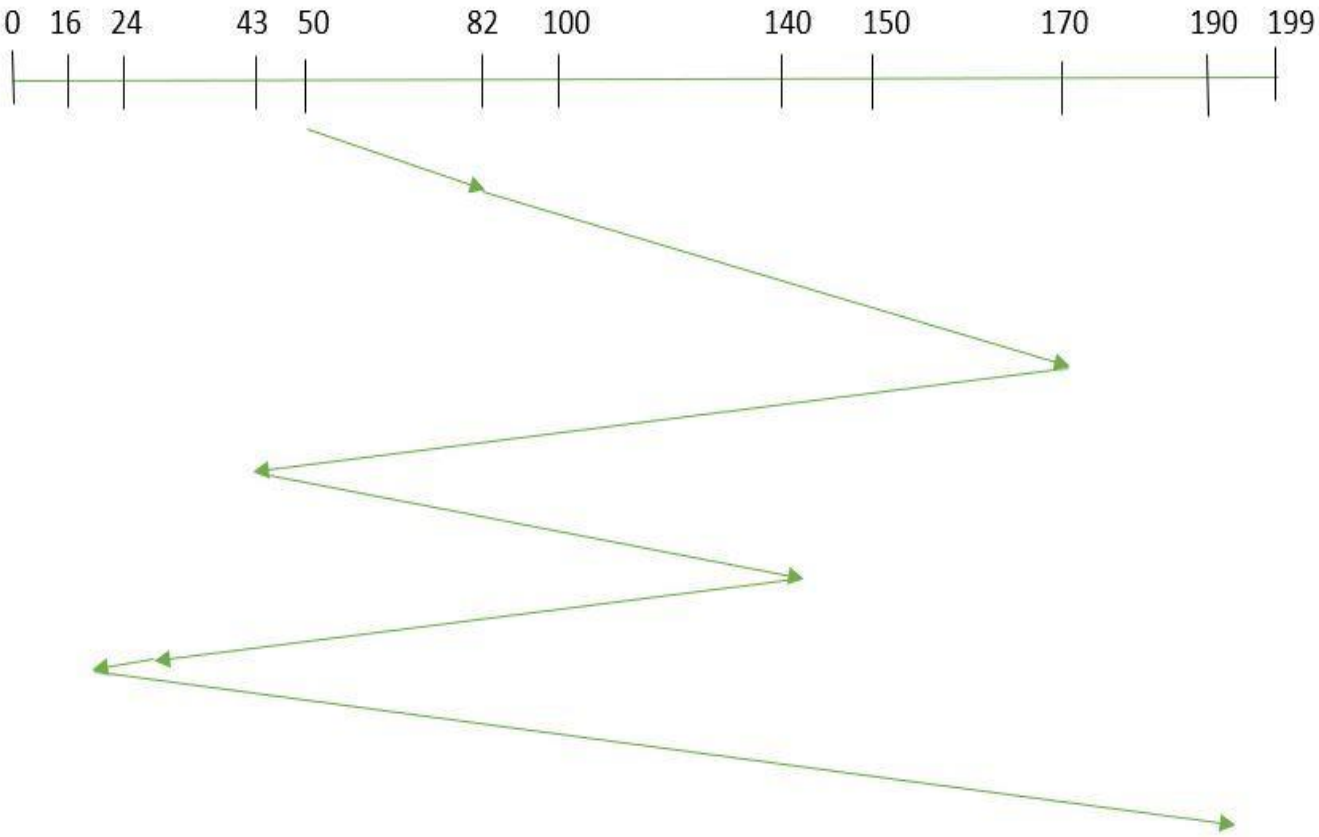
The purpose of disk scheduling algorithms is to reduce the total seek time.

Disk scheduling is important because:

- Multiple I/O requests may arrive by different processes and only one I/O request can be served at a time by the disk controller. Thus other I/O requests need to wait in the waiting queue and need to be scheduled.
- Two or more request may be far from each other so can result in greater disk arm movement.
- Hard drives are one of the slowest parts of the computer system and thus need to be accessed in an efficient manner
- .

1. **FCFS:** FCFS is the simplest of all the Disk Scheduling Algorithms. In FCFS, the requests are addressed in the order they arrive in the disk queue.Let us understand this with the help of an example.

   Example:
   Suppose the order of request is- (82,170,43,140,24,16,190)
   And current position of Read/Write head is : 50



So, total seek time:
=(82-50)+(170-82)+(170-43)+(140-43)+(140-24)+(24-16)+(190-16)
=642

Advantages:

- Every request gets a fair chance
- No indefinite postponement

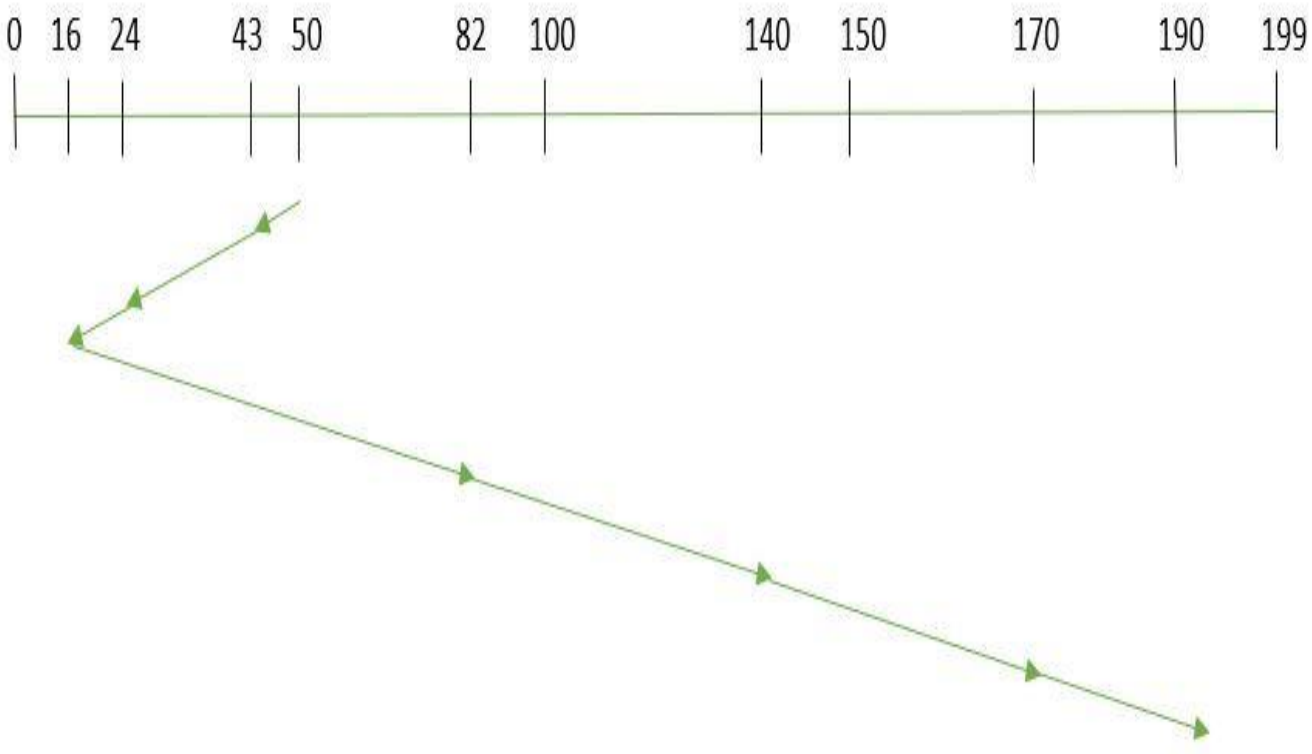
Disadvantages:

- Does not try to optimize seek time
- May not provide the best possible service

2. **SSTF:** In SSTF (Shortest Seek Time First), requests having shortest seek time are executed first. So, the seek time of every request is calculated in advance in the queue and then they are scheduled according to their calculated seek time. As a result, the request near the disk arm will get executed first. SSTF is certainly an improvement over FCFS as it decreases the average response time and increases the throughput of system.Let us understand this with the help of an example.

   Example:

Suppose the order of request is- (82,170,43,140,24,16,190)
And current position of Read/Write head is : 50



So, total seek time:

=(50-43)+(43-24)+(24-16)+(82-16)+(140-82)+(170-40)+(190-170)
=208

Advantages:

- Average Response Time decreases
- Throughput increases

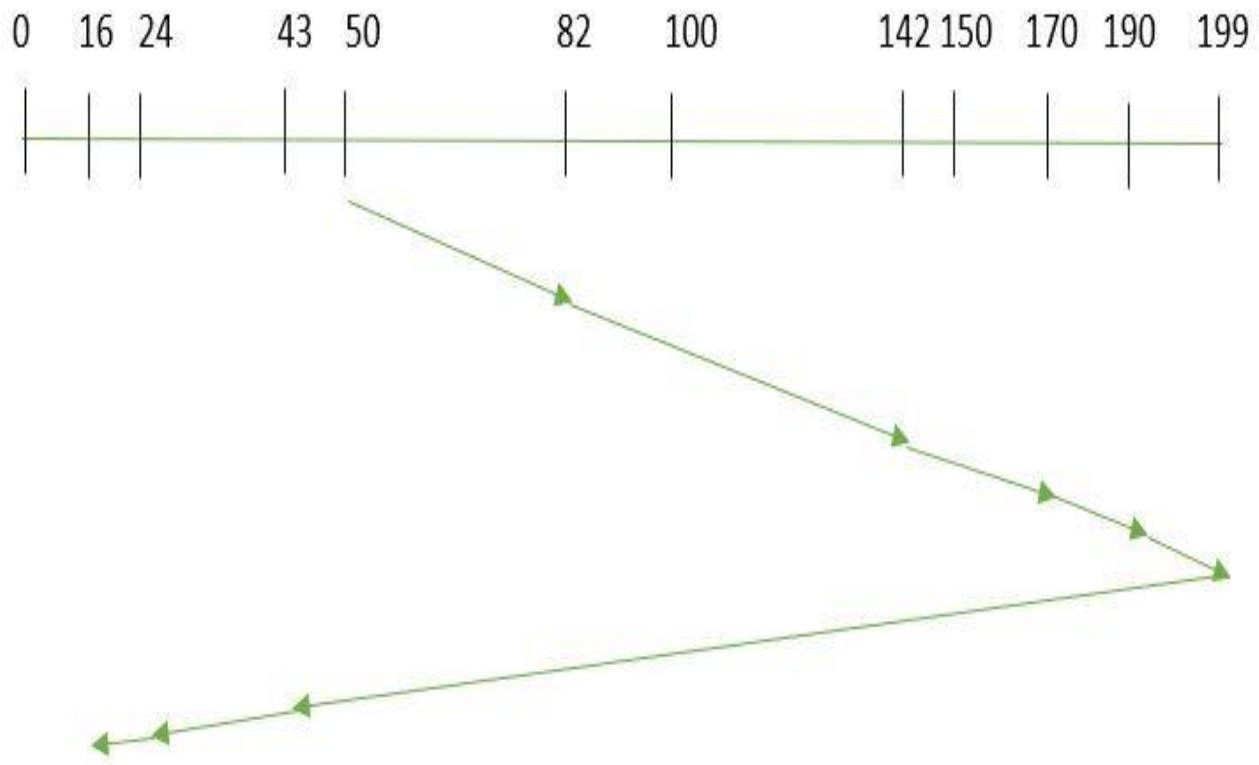
Disadvantages:

- Overhead to calculate seek time in advance
- Can cause Starvation for a request if it has higher seek time as compared to incoming requests
- High variance of response time as SSTF favours only some requests

3. **SCAN:** In SCAN algorithm the disk arm moves into a particular direction and services the requests coming in its path and after reaching the end of disk, it reverses its direction and again services the request arriving in its path. So, this algorithm works as an elevator and hence also known as **elevator algorithm.** As a result, the requests at the midrange are serviced more and

those arriving behind the disk arm will have to wait.

Example:
Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move **“towards the larger value”.**



Therefore, the seek time is calculated as:

=(199-50)+(199-16)
=332

Advantages:

- High throughput
- Low variance of response time
- Average response time

Disadvantages:

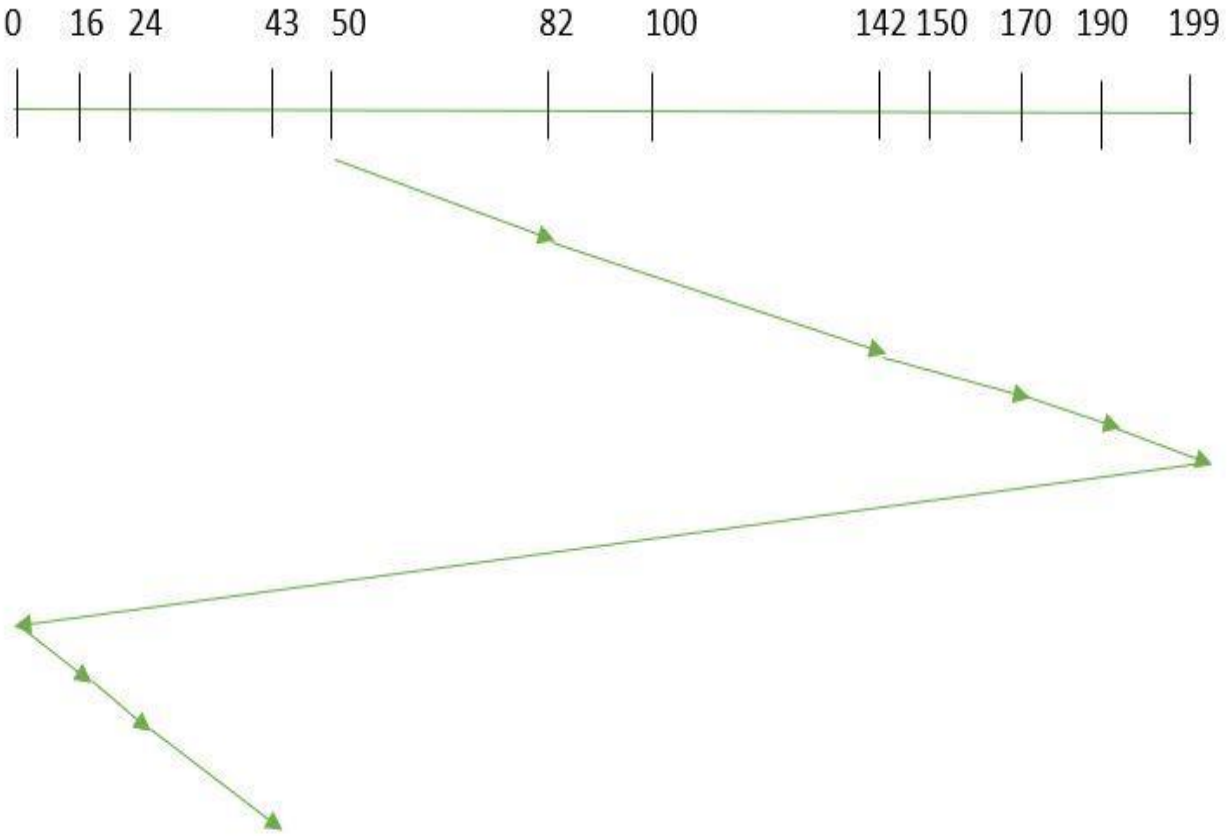
- Long waiting time for requests for locations just visited by disk arm

4. **CSCAN**: In SCAN algorithm, the disk arm again scans the path that has been scanned, after reversing its direction. So, it may be possible that too many requests are waiting at the other end or there may be zero or few requests pending at the scanned area.

These situations are avoided in *CSCAN* algorithm in which the disk arm instead of reversing its direction goes to the other end of the disk and starts servicing the requests from there. So, the disk arm moves in a circular fashion and this algorithm is also similar to SCAN algorithm and hence it is known as C-SCAN (Circular SCAN).

Example:

Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move **"towards the larger value".**



Seek time is calculated as:

=(199-50)+(199-0)+(43-0)
=391

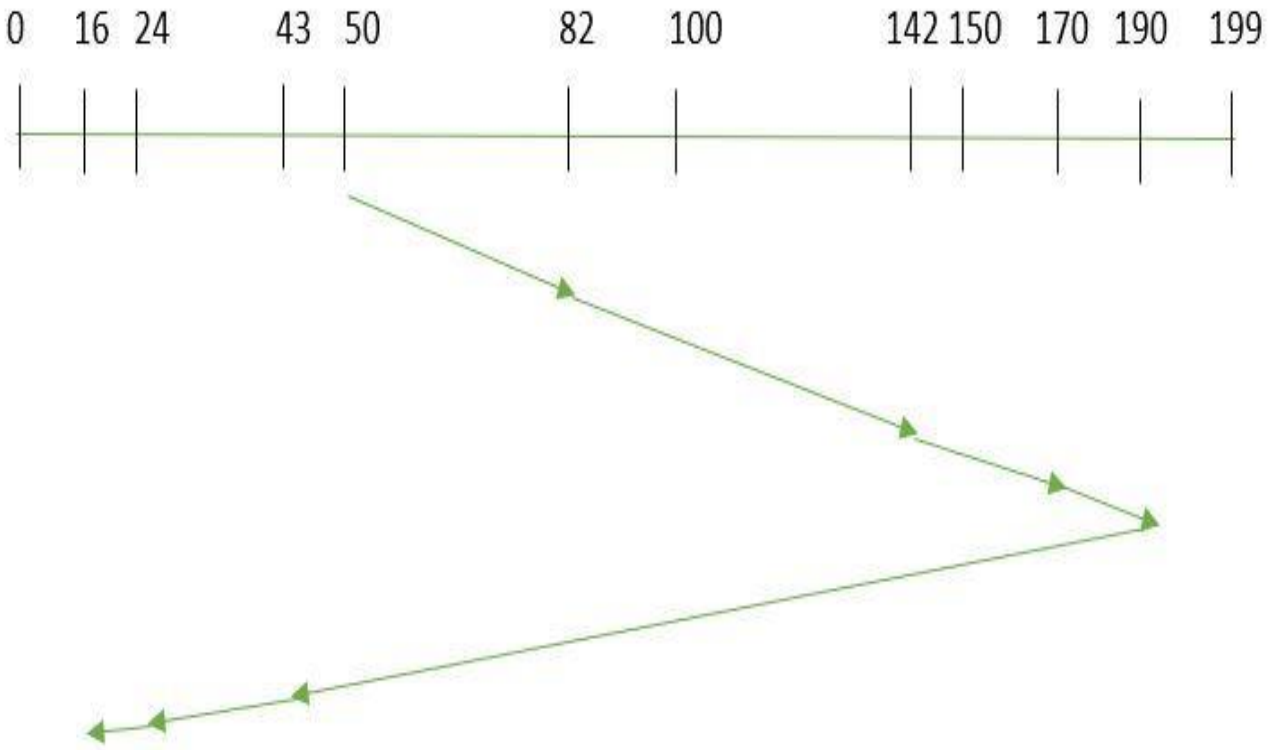
Advantages:

- Provides more uniform wait time compared to SCAN

5. **LOOK:** It is similar to the SCAN disk scheduling algorithm except for the difference that the disk arm in spite of going to the end of the disk goes only to the last request to be serviced in front of

the head and then reverses its direction from there only. Thus it prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.

Example:
Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move **"towards the larger value".**



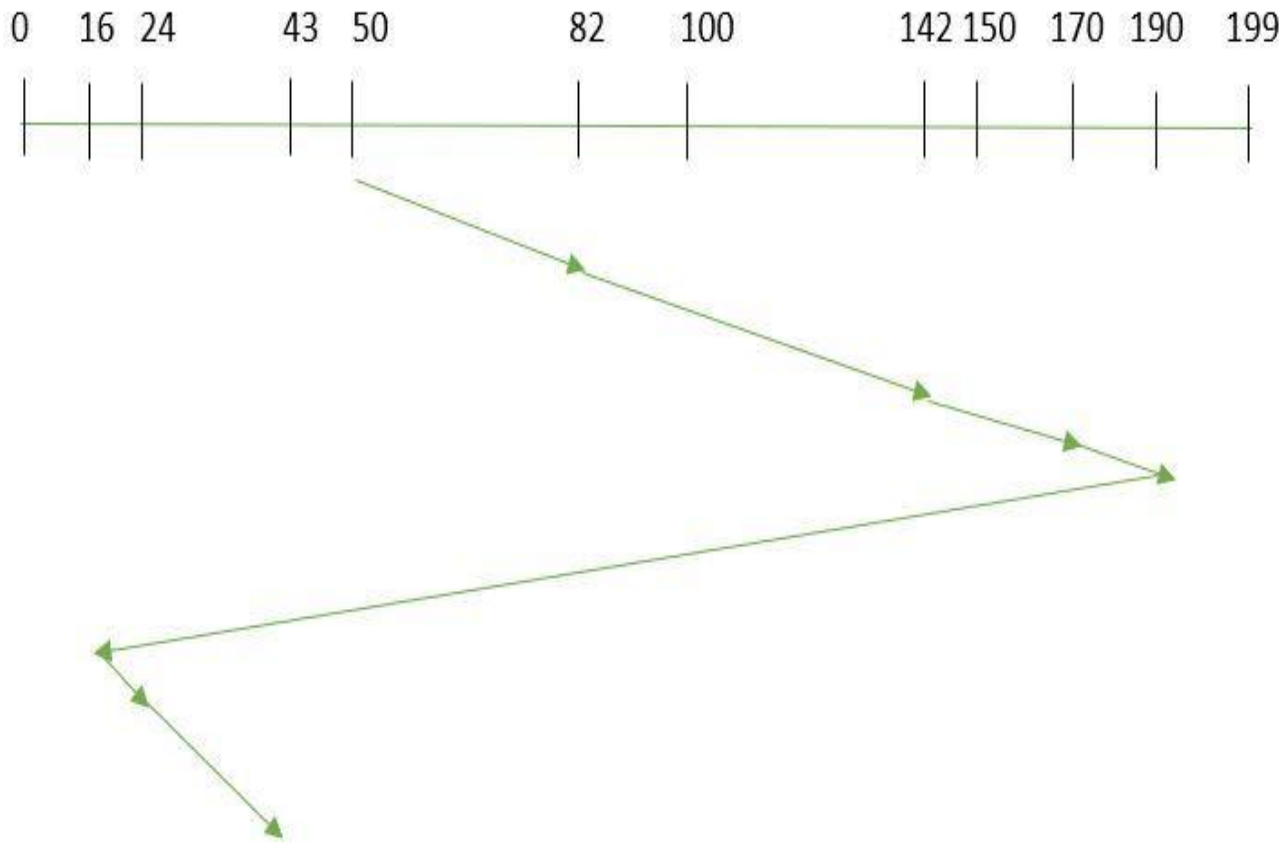
So, the seek time is calculated as:

=(190-50)+(190-16)
=314

6. **CLOOK:** As LOOK is similar to SCAN algorithm, in similar way, CLOOK is similar to CSCAN disk scheduling algorithm. In CLOOK, the disk arm in spite of going to the end goes only to the last request to be serviced in front of the head and then from there goes to the other end's last request. Thus, it also prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.

   Example:

Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move **“towards the larger value”**



So, the seek time is calculated as:

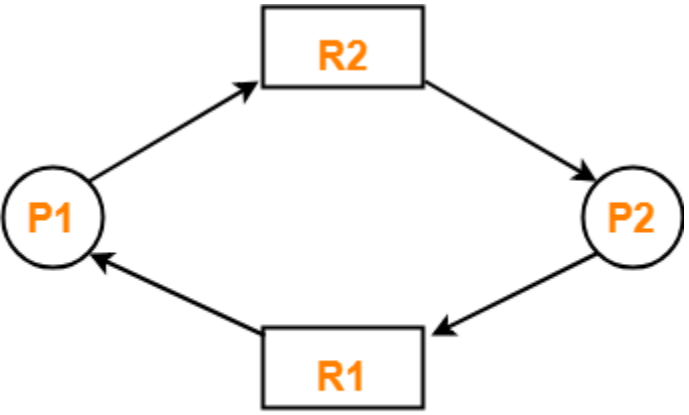
=(190-50)+(190-16)+(43-16)
=341

Deadlock in OS-

Deadlock is a situation where-

The execution of two or more processes is blocked because each process holds some resource and waits for another resource held by some other process.

Example-



Example of a deadlock

Here

Process P1 holds resource R1 and waits for resource R2 which is held by process P2.
Process P2 holds resource R2 and waits for resource R1 which is held by process P1.
None of the two processes can complete and release their resource.
Thus, both the processes keep waiting infinitely.

Conditions For Deadlock-

There are following 4 necessary conditions for the occurrence of deadlock-

. Mutual Exclusion
. Hold and Wait
. No preemption
. Circular wait

1. Mutual Exclusion-

By this condition,

There must exist at least one resource in the system which can be used by only one process at a time.
If there exists no such resource, then deadlock will never occur.
Printer is an example of a resource that can be used by only one process at a time.

2. Hold and Wait-

By this condition,

There must exist a process which holds some resource and waits for another resource held by some other process.

3. No Preemption-

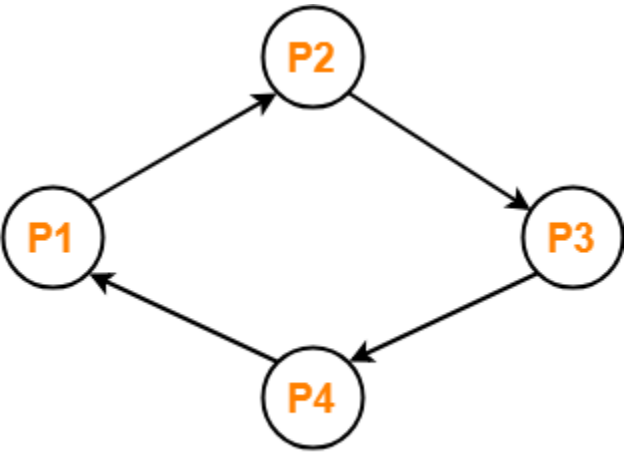
By this condition,

Once the resource has been allocated to the process, it can not be preempted.
It means resource can not be snatched forcefully from one process and given to the other process.
The process must release the resource voluntarily by itself.

4. Circular Wait-

By this condition,

All the processes must wait for the resource in a cyclic manner where the last process waits for the resource held by the first process.



Circular Wait

Here,

Process P1 waits for a resource held by process P2.
Process P2 waits for a resource held by process P3.
Process P3 waits for a resource held by process P4.
Process P4 waits for a resource held by process P1.

Important Note-

All these 4 conditions must hold simultaneously for the occurrence of deadlock.
If any of these conditions fail, then the system can be ensured deadlock free

System Protection in Operating System

Last Updated : 21 Aug, 2019

Protection refers to a mechanism which controls the access of programs, processes, or users to the resources defined by a computer system. We can take protection as a helper to multi programming operating system, so that many users might safely share a common logical name space such as directory or files.

Need of Protection:

- To prevent the access of unauthorized users and
- To ensure that each active programs or processes in the system uses resources only as the stated policy,
- To improve reliability by detecting latent errors.

Role of Protection:

The role of protection is to provide a mechanism that implement policies which defines the uses of resources in the computer system.Some policies are defined at the time of design of the system, some are designed by management of the system and some are defined by the users of the system to protect their own files and programs.

Every application has different policies for use of the resources and they may change over time so protection of the system is not only concern of the designer of the operating system. Application programmer should also design the protection mechanism to protect their system against misuse.

Policy is different from mechanism. Mechanisms determine how something will be done and policies determine what will be done.Policies are changed over time and place to place. Separation of mechanism and policy is important for the flexibility of the system

SECURITY IN OS

Security refers to providing a protection system to computer system resources such as CPU, memory, disk, software programs and most importantly data/information stored in the computer system. If a computer program is run by an unauthorized user, then he/she may cause severe damage to computer or data stored in it. So a computer system must be protected against unauthorized access, malicious access to system memory, viruses, worms etc.

It is the responsibility of the Operating System to create a protection system which ensures that a user who is running a particular program is authentic. Operating Systems generally identifies/authenticates users using following three ways −

- **Username / Password** − User need to enter a registered username and password with Operating system to login into the system.
- **User card/key** − User need to punch card in card slot, or enter key generated by key generator in option provided by operating system to login into the system.
- **User attribute - fingerprint/ eye retina pattern/ signature** − User need to pass his/her attribute via designated input device used by operating system to login into the system.

One Time passwords

One-time passwords provide additional security along with normal authentication. In One-Time Password system, a unique password is required every time user tries to login into the system. Once a one-time password is used, then it cannot be used again. One-time password are implemented in various ways.

- **Random numbers** − Users are provided cards having numbers printed along with corresponding alphabets. System asks for numbers corresponding to few alphabets randomly chosen.
- **Secret key** − User are provided a hardware device which can create a secret id mapped with user id. System asks for such secret id which is to be generated every time prior to login.
- **Network password** − Some commercial applications send one-time passwords to user on registered mobiles or emils which prior to login.

ACCESS CONTROL

Access control for an operating system determines how the operating system implements accesses to system resources by satisfying the security objectives of integrity, availability, and secrecy. Such a mechanism authorizes subjects (e.g., processes and users) to perform certain operations (e.g., read, write) on objects and resources of the OS (e.g., files, sockets).

Operating system is the main software between the users and the computing system resources that manage tasks and programs. Resources may include files, sockets, CPU, memory, disks, timers, network, etc. System administrators, developers, regular users, guests, and even hackers could be the users of the computing system. Information security is an important issue for an operating system due to its...

Access control

Access control is about ensuring that authorized users can do what they are permitted to do … and no more than that. In the real world, we rely on keys, badges, guards, and policy rules for enforcing access control. In the computer world, we achieve access control through a combination of hardware, operating systems, web servers, databases, and other multi-access software, and polices.

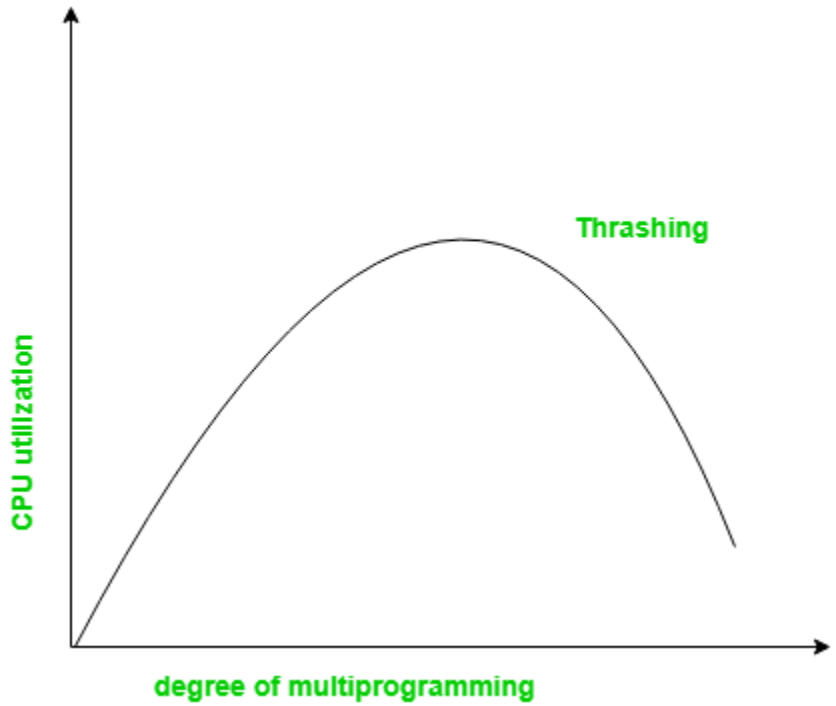
Access control and the operating system

In its most basic sense, an operating system is responsible for controlling access to system resources. These resources include the processor, memory, files and devices, and the network. The operating system needs to be protected from applications. If not, an operation may, accidentally or maliciously, destroy the integrity of the operating system. Applications also need to be protected from each other so that one application cannot read another's memory (enforce **confidentiality**) or modify them (enforce **integrity**). We also need to make sure that the operating system always stays in control. A process

should not be able to take over the computer and keep the operating system or other processes from running.

THRASHING:

Thrashing is a condition or a situation when the system is spending a major portion of its time in servicing the page faults, but the actual processing done is very negligible.



The basic concept involved is that if a process is allocated too few frames, then there will be too many and too frequent page faults. As a result, no useful work would be done by the CPU and the CPU utilisation would fall drastically. The long-term scheduler would then try to improve the CPU utilisation by loading some more processes into the memory thereby increasing the degree of multiprogramming. This would result in a further decrease in the CPU utilization triggering a chained reaction of higher page faults followed by an increase in the degree of multiprogramming, called Thrashing.

What is Demand Paging

Definition: Demand paging is a process of swapping in the **Virtual Memory** system. In this process, all data is not moved from hard drive to **main memory** because while using this demand paging, when some programs are getting demand then data will be transferred. But, if required data is already existed into memory then not need to copy of data. The demand paging system is done with swapping from auxiliary storage to **primary memory**, so it is known as "Lazy Evaluation".

Diagram of Demand Paging

Some **Page Replacement Algorithms** are used in the **demand paging concept** to replace different pages – such as FIFO, LIFO, Optimal Algorithm, LRU Page, and Random Replacement Page Replacement Algorithms.

How Does Demand Paging Working

Demand paging system is totally depend on the page table implementation because page table helps to maps logical memory to physical memory. Bitwise operators are implemented in the page table to indication that pages are ok or not (valid or invalid). All valid pages are existed into **primary memory**, and other side invalid pages are existed into **secondary memory**.

Now all process go ahead to access all pages, then some things will be happened. Such as –

Attempt to currently access page.

If page is ok (Valid), then all processing instructions work as normal.

If, anyone page is found as invalid, then page- fault issue is arise.

Now memory reference is determined that valid reference is existed on the auxiliary memory or not. If not existed, then process is terminated, otherwise needed pages are paged in.

Now disk operations are implemented to fetch the required page into **primary memory**.

Example of Demand Paging

Memory Access Time = 200 nanoseconds

Average Page Fault Service Time = 8 milliseconds

EAT = (1-p)*200+p(8 milliseconds)
= (1-p)*200+p*8000000

= 200+p*7999800

EAT means direct proportional to the page fault rate.

Advantages of Demand Paging

Memory can be utilized with better efficiently.
We have to right for scaling of **virtual memory**.
If, any program is larger to physical memory then It helps to run this **program**.
No need of compaction.
Easy to share all pages
Partition management is more simply.
It is more useful in **time sharing system**.
It has no any limitations on level of **multi-programming**.
Discards external fragmentation
Easy to swap all pages
It allows pre-paging concept.
Easy to swap out page least likely to be used
All pages can be mapped appropriately.
All data can be collected over PM

Disadvantages of Demand Paging

It has more probability of internal fragmentation.
Its memory access time is longer.
Page Table Length Register (PTLR) has limit for **virtual memory**
Page map table is needed additional memory and **registers**.

Pure Demand Paging

In this section, we will discuss about **difference between demand paging and pure demand paging**.

In the initially stage, if anyone page is not loaded into **primary memory** then page faults are occurred, and then paged are loaded with on demanding of the process. This process is known as “Pure Demand Paging”.

Pure demand paging is not able to load single page into **main memory**, otherwise it will be fired the Page-Fault.

In pure demand paging, when process will be started up then all memory is swapped from auxiliary memory to primary memory.

Demand Segmentation in OS

Operating system implements demand segmentation like demand paging. While using "Demand Paging", if it to get lack of **hardware resources**, then OS implements the demand segmentation.

Segment table keeps all information related to demand segmentation such as valid bit because on the behalf of valid bit can be specified that segment has existed in the physical memory or not. If, anytime physical memory is not capable to store their segments then to get segment fault, and then it try to fetch required segments from physical memory. Similar to **page fault**.

Demand segmentation helps to decrease the number of **page faults** in the paging system.

Cryptography and its Types

Last Updated : 08 Jan, 2020

Cryptography is technique of securing information and communications through use of codes so that only those person for whom the information is intended can understand it and process it. Thus preventing unauthorized access to information. The prefix "crypt" means "hidden" and suffix graphy means "writing".

In Cryptography the techniques which are use to protect information are obtained from mathematical concepts and a set of rule based calculations known as algorithms to convert messages in ways that make it hard to decode it. These algorithms are used for cryptographic key generation, digital signing, verification to protect data privacy, web browsing on internet and to protect confidential transactions such as credit card and debit card transactions.

Techniques used For Cryptography:

In today's age of computers cryptography is often associated with the process where an ordinary plain text is converted to cipher text which is the text made such that intended receiver of the text can only decode it and hence this process is known as encryption. The process of conversion of cipher text to plain text this is known as decryption.

Features Of Cryptography are as follows:

1. **Confidentiality:**
   Information can only be accessed by the person for whom it is intended and no other person except him can access it.
2. **Integrity:**
   Information cannot be modified in storage or transition between sender and intended receiver without any addition to information being detected.
3. **Non-repudiation:**
   The creator/sender of information cannot deny his or her intention to send information at later stage.

4. **Authentication:**
   The identities of sender and receiver are confirmed. As well as destination/origin of information is confirmed.

Types Of Cryptography:

In general there are three types Of cryptography:

1. **Symmetric Key Cryptography:**
   It is an encryption system where the sender and receiver of message use a single common key to encrypt and decrypt messages. Symmetric Key Systems are faster and simpler but the problem is that sender and receiver have to somehow exchange key in a secure manner. The most popular symmetric key cryptography system is Data Encryption System(DES).
2. **Hash Functions:**
   There is no usage of any key in this algorithm. A hash value with fixed length is calculated as per the plain text which makes it impossible for contents of plain text to be recovered. Many operating systems use hash functions to encrypt passwords.
3. **Asymmetric Key Cryptography:**
   Under this system a pair of keys is used to encrypt and decrypt information. A public key is used for encryption and a private key is used for decryption. Public key and Private Key are different. Even if the public key is known by everyone the intended receiver can only decode it because he alone knows the private key.

FCFS Scheduling

First come first serve (FCFS) scheduling algorithm simply schedules the jobs according to their arrival time. The job which comes first in the ready queue will get the CPU first. The lesser the arrival time of the job, the sooner will the job get the CPU. FCFS scheduling may cause the problem of starvation if the burst time of the first process is the longest among all the jobs.

Advantages of FCFS

- Simple
- Easy
- First come, First serv

Disadvantages of FCFS

1. The scheduling method is non preemptive, the process will run to the completion.
2. Due to the non-preemptive nature of the algorithm, the problem of starvation may occur.

3. Although it is easy to implement, but it is poor in performance since the average waiting time is higher as compare to other scheduling algorithms.

Example

Let's take an example of The FCFS scheduling algorithm. In the Following schedule, there are 5 processes with process ID **P0, P1, P2, P3 and P4**. P0 arrives at time 0, P1 at time 1, P2 at time 2, P3 arrives at time 3 and Process P4 arrives at time 4 in the ready queue. The processes and their respective Arrival and Burst time are given in the following table.

The Turnaround time and the waiting time are calculated by using the following formula.

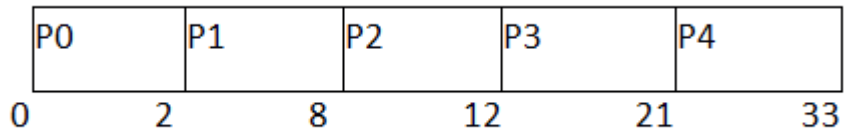
Turn Around Time = Completion Time - Arrival Time

Waiting Time = Turnaround time - Burst Time

The average waiting Time is determined by summing the respective waiting time of all the processes and divided the sum by the total number of processes.

Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time
0	0	2	2	2	0
1	1	6	8	7	1
2	2	4	12	8	4
3	3	9	21	18	9
4	4	12	33	29	17

Avg Waiting Time=31/5



(Gantt chart)