Unit V

Part A (Memory Management)

1. Swapping
2. Contiguous Memory Allocation
3. Paging
4. Structure of the Page Table
5. Segmentation

Virtual Memory Management :

6. Demand Paging
7. Page-Replacement Algorithms
8. Thrashing.

Part B (File-System Interface)

1. File Concept
2. Access Methods
3. Directory structure
4. File-System mounting
5. Files Sharing
6. Protection
7. File-System Structure
8. File-System implementation
9. Allocation Methods
10. Free-Space Management
11. Disk Structure
12. Disk Scheduling

****

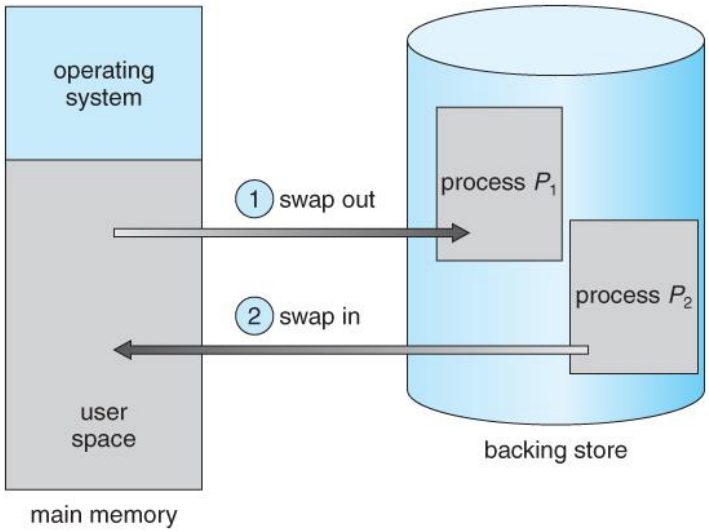
1. Swapping

A process must be loaded into memory in order to execute.
If there is not enough memory available to keep all running processes in memory at the same time, then some processes who are not currently using the CPU may have their memory swapped out to a fast local disk called the ***backing store.***

Standard Swapping

- If compile-time or load-time address binding is used, then processes must be swapped back into the same memory location from which they were swapped out. If execution time binding is used, then the processes can be swapped back into any available location.
- Swapping is a very slow process compared to other operations.
- To reduce swapping transfer overhead, it is desired to transfer as little information as possible, which requires that the system know how much memory a process ***is*** using, as opposed to how much it ***might*** use. Programmers can help with this by freeing up dynamic memory that they are no longer using.

- It is important to swap processes out of memory only when they are idle, or only when there are no pending I/O operations.
- Most modern OS use Paging, because Swapping is too slow.

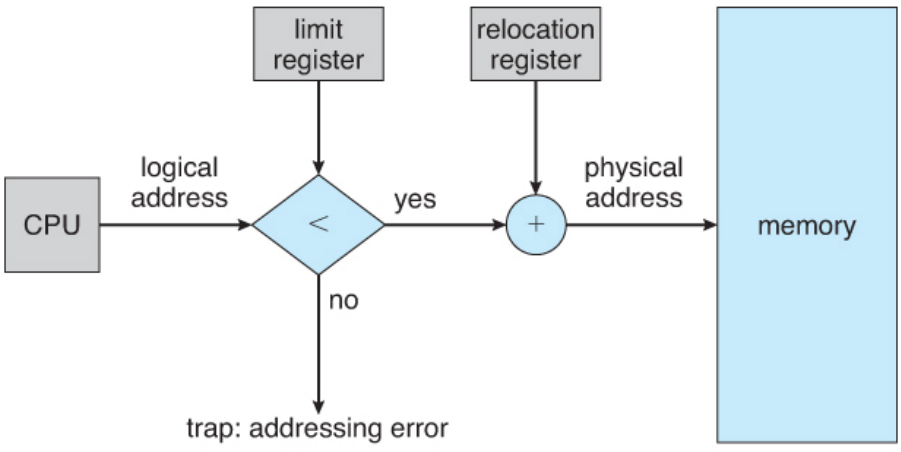


Swapping of two processes using a disk as a backing store

2. Contiguous Memory Allocation

One approach to memory management is to load each process into a contiguous space. The operating system is allocated space first, usually at either low or high memory locations, and then the remaining available memory is allocated to processes as needed.

Memory Allocation



Hardware support for relocation and limit registers

One method of allocating contiguous memory is to divide all available memory into equal sized partitions, and to assign each process to their own partition. This restricts both the number of simultaneous processes and the maximum size of each process, and is no longer used.

An alternate approach is to keep a list of unused memory blocks, There are many different strategies for finding the "best" allocation of memory to processes, they are,
1. First fit - Search the list of free memory until one is found that is big enough to satisfy the request, and assign a portion to that process.
2. Best fit - Allocate the *smallest Memory* that is big enough to satisfy the request.
3. Worst fit - Allocate the largest Memory available, thereby increasing the likelihood that the remaining portion will be usable for satisfying future requests.

Fragmentation

External fragmentation means that the available memory is broken up into lots of little pieces, none of which is big enough to satisfy the next memory requirement, although the sum total could.

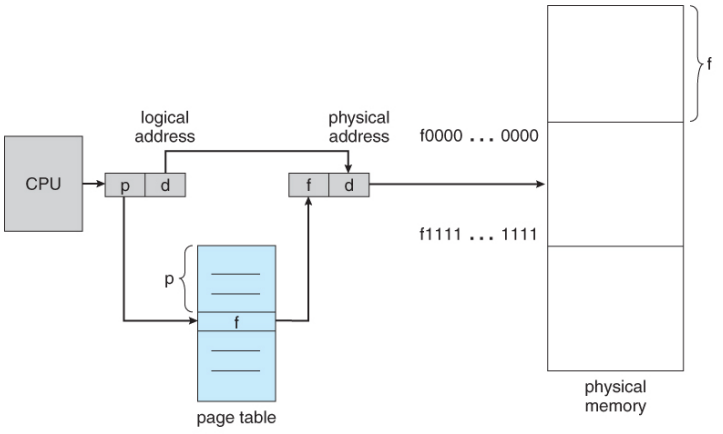
Internal fragmentation also occurs, with all memory allocation strategies. This is caused by the fact that memory is allocated in blocks of a fixed size, whereas the actual memory needed will rarely be that exact size.

3. Paging

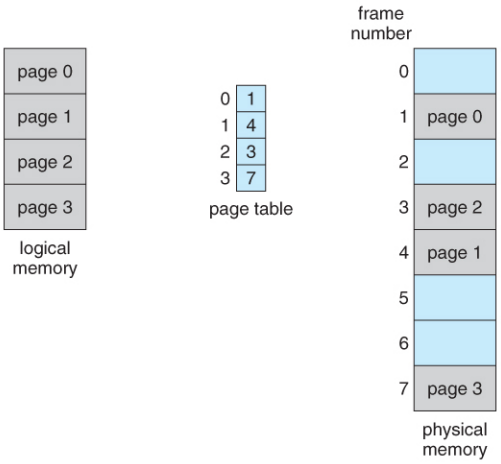
Paging is a memory management scheme that allows processes physical memory to be discontinuous, and which eliminates problems with fragmentation by allocating memory in equal sized blocks known as ***pages***.

Basic Method

- The basic idea behind paging is to divide physical memory into a number of equal sized blocks called ***frames***, and to divide a programs logical memory space into blocks of the same size called ***pages.***
- Any page (from any process) can be placed into any available frame.
- The ***page table*** is used to look up what frame a particular page is stored in at the moment.



Paging hardware

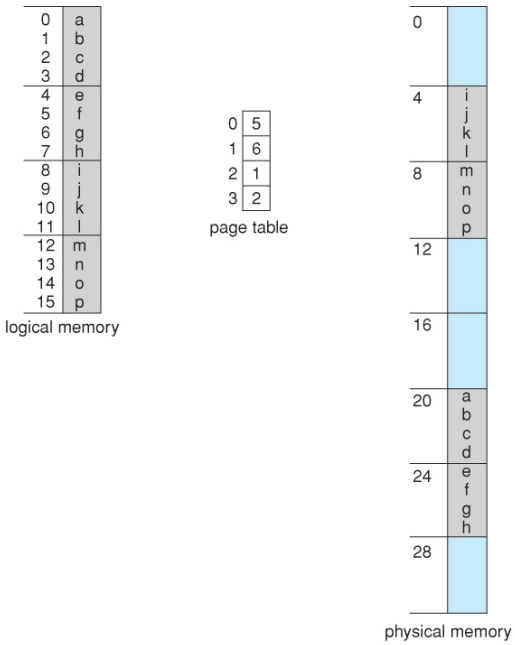


Paging model of logical and physical memory

- A logical address consists of two parts: A page number in which the address resides, and an offset from the beginning of that page. (The number of bits in the page number limits how many pages a single process can address. The number of bits in the offset determines the maximum size of each page, and should correspond to the system frame size.)
- The page table maps the page number to a frame number, to yield a physical address which also has two parts: The frame number and the offset within that frame. The number of bits in the frame number determines how many frames the system can address, and the number of bits in the offset determines the size of each frame.
- Page numbers, frame numbers, and frame sizes are determined by the architecture, but are typically powers of two, allowing addresses to be split at a certain number of bits.
- The number of bits in the page number and the number of bits in the frame number do not have to be identical.

Example :

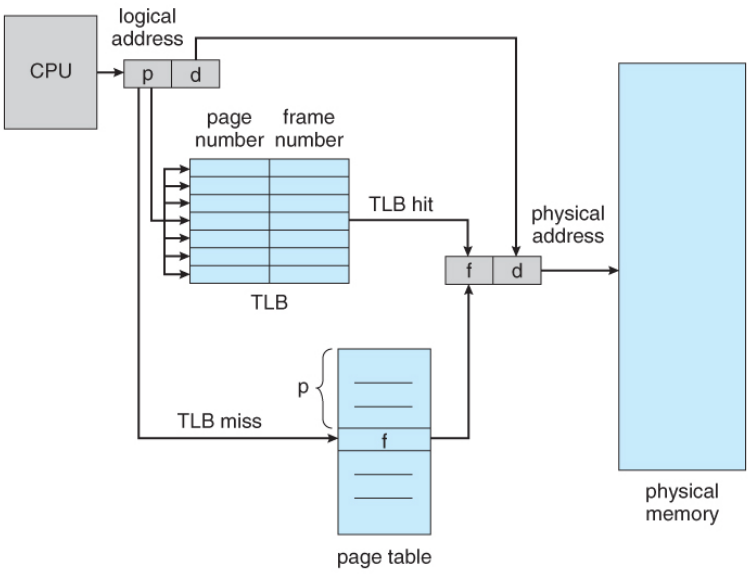
a process has 16 bytes of logical memory, mapped in 4 byte pages into 32 bytes of physical memory.



Paging example for a 32-byte memory with 4-byte pages

Hardware Support

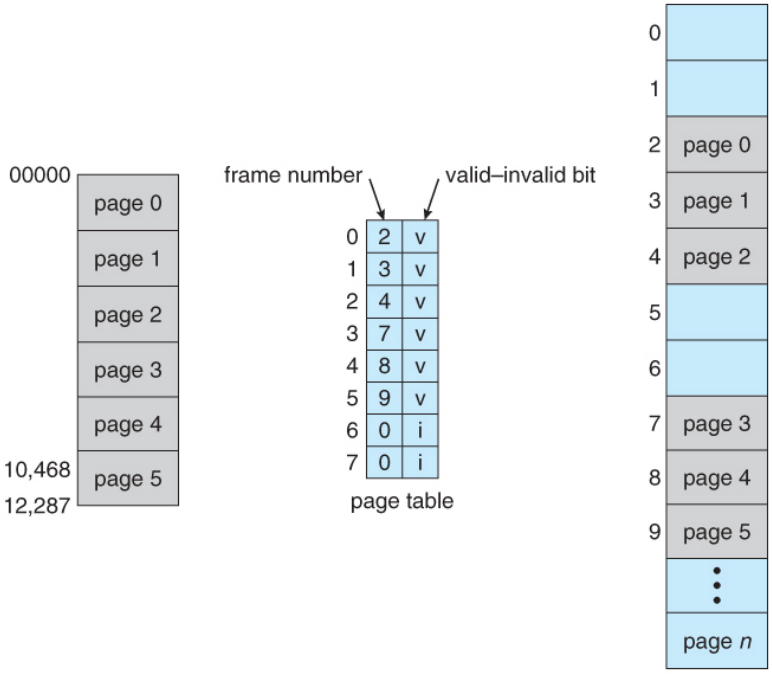
- Page lookups must be done for every memory reference, and whenever a process gets swapped in or out of the CPU, its page table must be swapped in and out too, along with the instruction registers, etc. It is therefore appropriate to provide hardware support for this operation, in order to make it as fast as possible and to make process switches as fast as possible also.
- An alternate option is to store the page table in main memory, and to use a single register (called the ***page-table base register, PTBR***) to record where in memory the page table is located.
    - Process switching is fast, because only the single register needs to be changed.
    - However memory access just got half as fast, because every memory access now requires ***two*** memory accesses - One to fetch the frame number from memory and then another one to access the desired memory location.
    - The solution to this problem is to use a very special high-speed memory device called the ***translation look-aside buffer, TLB.***
        - The benefit of the TLB is that it can search an entire table for a key value in parallel, and if it is found anywhere in the table, then the corresponding lookup value is returned.



Paging hardware with TLB

Protection

- The page table can also help to protect processes from accessing memory that they shouldn't, or their own memory in ways that they shouldn't.
- A bit or bits can be added to the page table to classify a page as read-write, read-only, read-write-execute, or some combination of these sorts of things. Then each memory reference can be checked to ensure it is accessing the memory in the appropriate mode.
- Valid / invalid bits can be added to "mask off" entries in the page table that are not in use by the current process.
- Note that the valid / invalid bits described above cannot block all illegal memory accesses, due to the internal fragmentation.

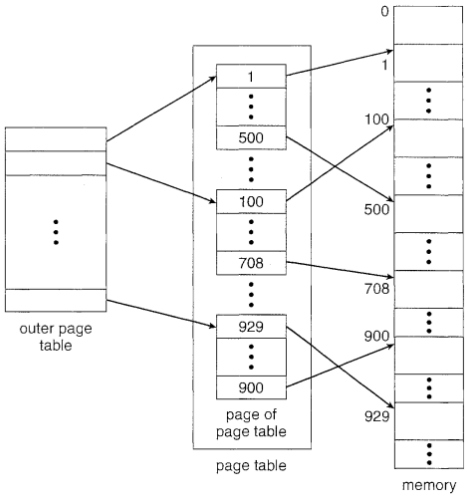


Valid (v) or invalid (i) bit in page table

4. Structure of the Page Table

Some of the common techniques for structuring the page table are,

1. Hierarchical Paging



a Two-Level Page-Table Scheme

2. Hashed Page Tables

A common approach for handling address spaces larger than 32 bits is to use a Hashed Page Table with the hash value being the virtual page number.

Each entry in the hash table contains a linked list of elements that hash to the same location (to handle collisions).

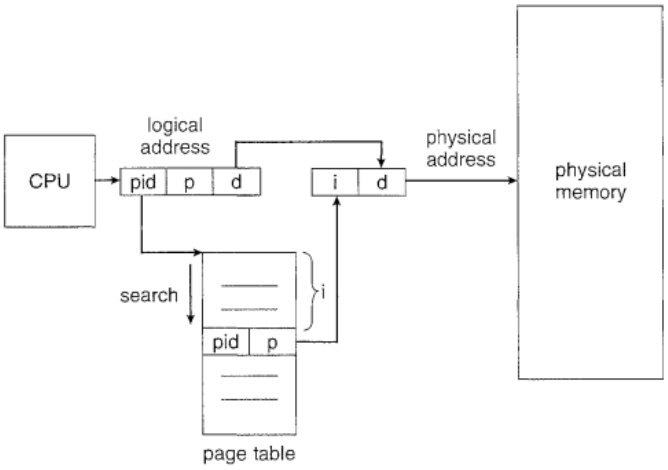
Each element consists of three fields:

(1) Virtual page number.

(2) The value of the mapped page frame and

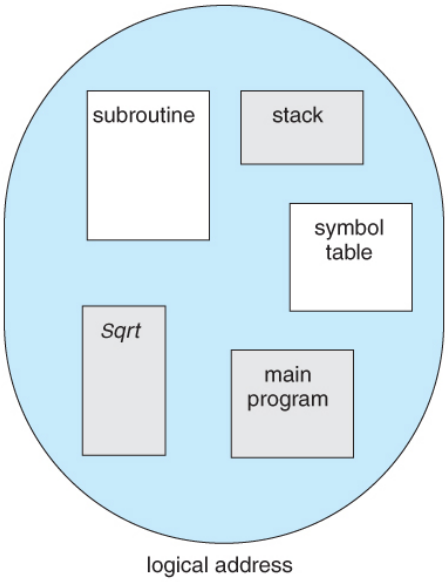
(3) A pointer to the next element in the linked list.

3. Inverted Page Tables



5. Segmentation

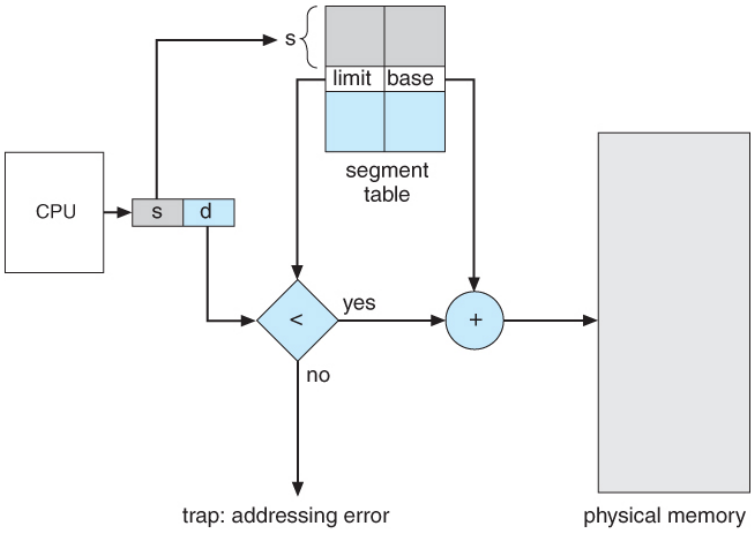
- Memory ***segmentation*** supports the view by providing addresses with a segment number (mapped to a segment base address) and an offset from the beginning of that segment.
- For example, a C compiler might generate 5 segments for the user code, library code, global (static) variables, the stack, and the heap, as shown in below Figure.



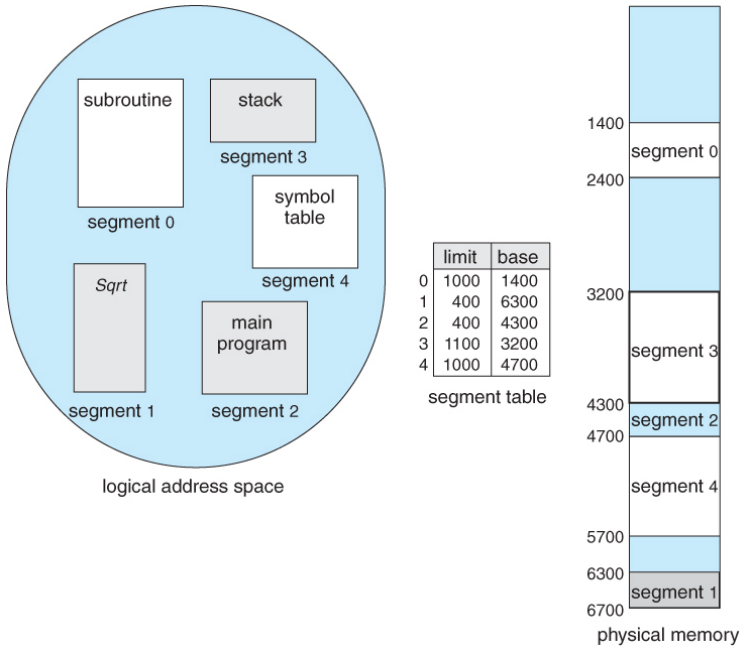
Programmer's view of a program.

Segmentation Hardware

A ***segment table*** maps segment-offset addresses to physical addresses, and simultaneously checks for invalid addresses, using a system similar to the page tables and relocation base registers.



Segmentation hardware



Example of segmentation

6. Demand Paging

Loading the entire program into memory results in loading the executable code for all options,

regardless of whether an option is ultimately selected by the user or not.

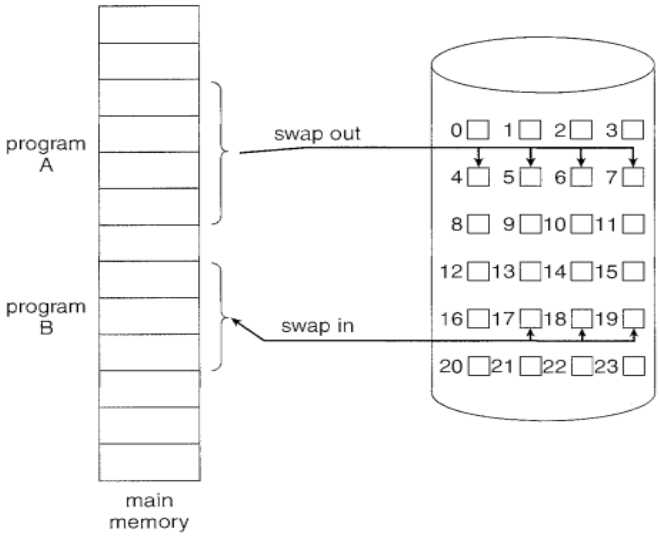
An alternative strategy is to load pages only as they are needed. This technique is known as Demand

paging and is commonly used in virtual memory systems.

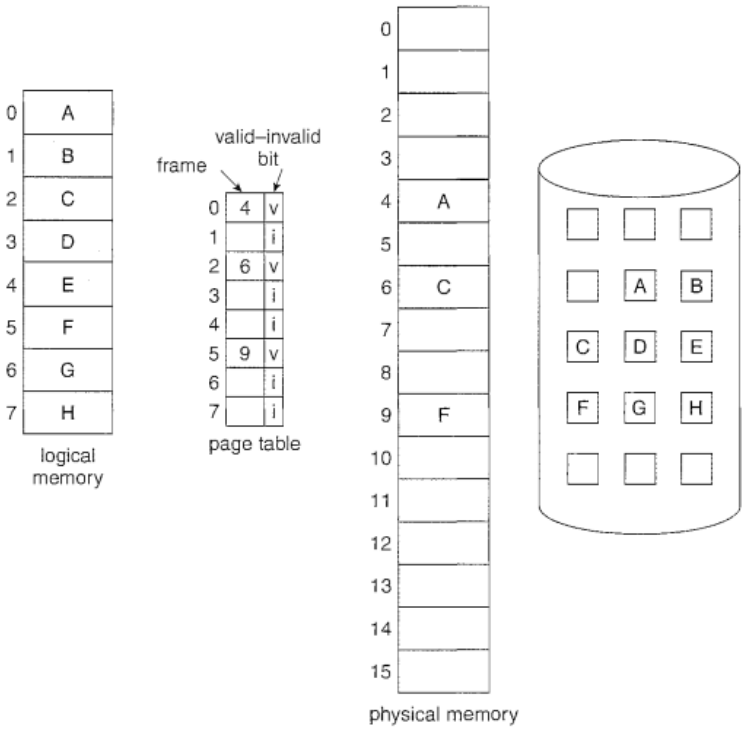
pages that are never accessed are thus never loaded into physical memory.

A demand-paging system is similar to a paging system with swapping

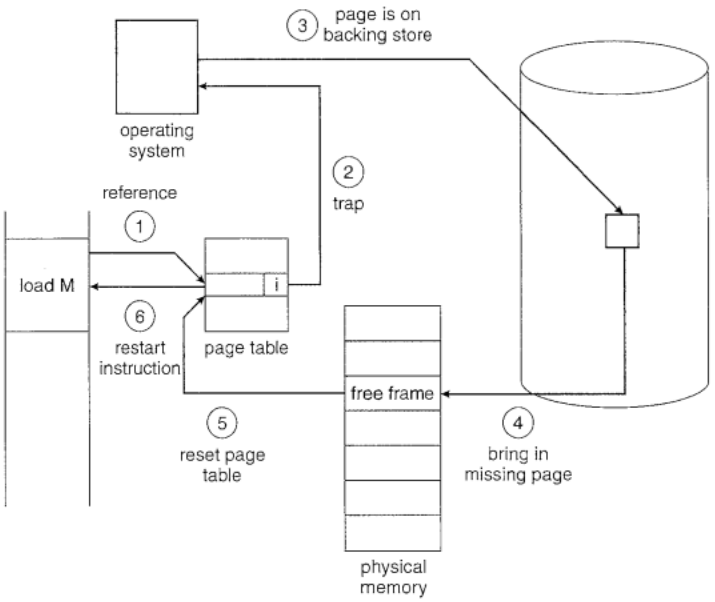
Rather than swapping the entire process into memory, however, we use a lazy swapper which never swaps a page into memory unless that page will be needed.



Transfer of a paged memory to contiguous disk space.



Page table when some pages are not in main memory.

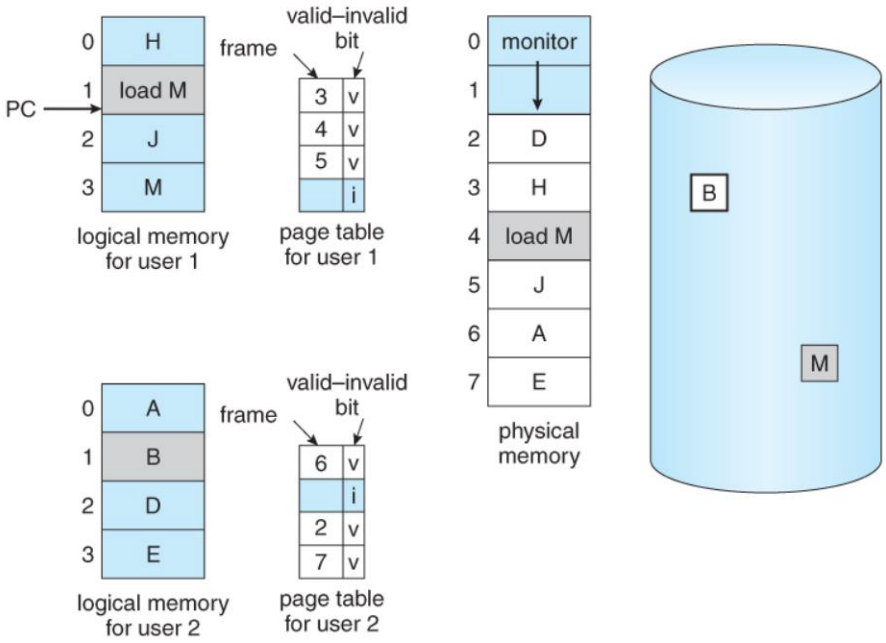


Steps in handling a page fault.

7. Page-Replacement Algorithms

In order to make the most use of virtual memory, we load several processes into memory at the same time. Since we only load the pages that are actually needed by each process at any given time, there is room to load many more processes than if we had to load in the entire process.

Page replacement is the process where the system uses paging for memory management, a page replacement algorithm is needed to decide which page needs to be replaced when new page comes in.

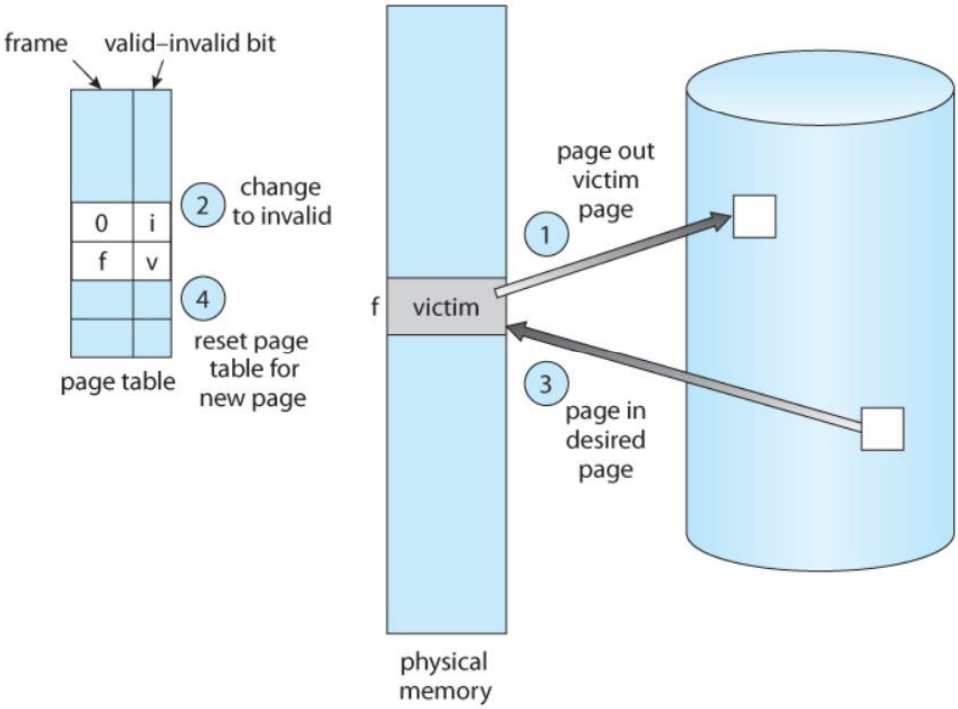


Need for page replacement.

Page Fault: A page fault happens when a running program accesses a memory page that is mapped into the virtual address space, but not loaded in physical memory.

Since actual physical memory is much smaller than virtual memory, page faults happen. In case of page fault, Operating System might have to replace one of the existing pages with the newly needed page.

Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce the number of page faults.



Page replacement.

The various Page Replacement Algorithms are,

a. FIFO Page Replacement

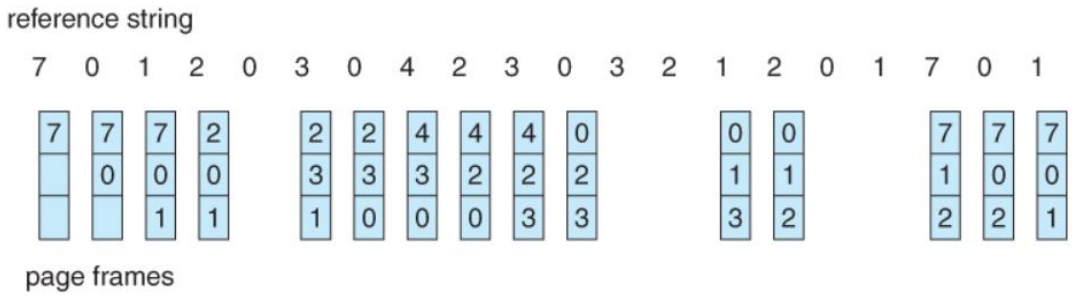
b. LRU Page Replacement

c. Optimal Page Replacement

a. FIFO Page Replacement :

The operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced, the page in the front of the queue is selected for removal.

Example :

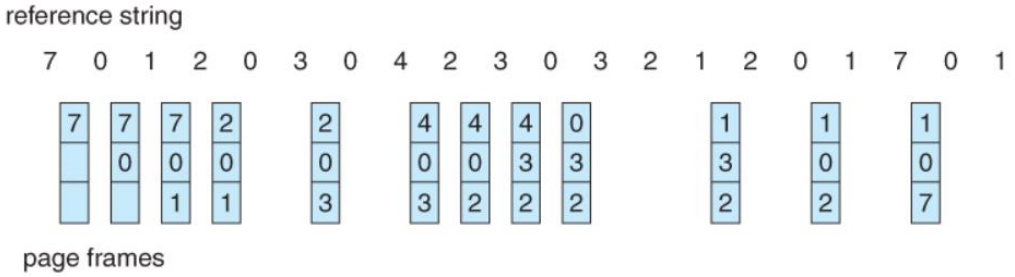


The number of page faults are 15

b. LRU Page Replacement :

In this algorithm page will be replaced which is least recently used.

Example :

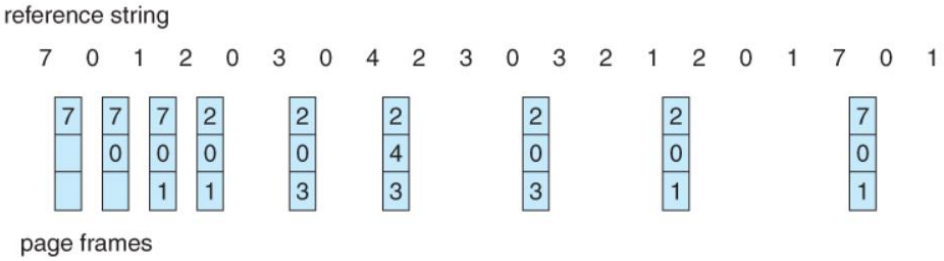


The number of page faults are 12

c. Optimal Page Replacement

In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.

Example:



The number of page faults are 9

8. Thrashing

If the number of frames allocated to a low-priority process falls below the minimum number required by the computer architecture, we must suspend that process's execution.
We should then page out its remaining pages, freeing all its allocated frames. This provision introduces a swap-in, swap-out level of intermediate CPU scheduling.
If the process does not have the number of frames it needs to support pages in active use, it will quickly page-fault. At this point, it must replace some page.
However, since all its pages are in active use, it must replace a page that will be needed again right away. Consequently, it quickly faults again, and again and again, replacing pages that it must bring back in immediately.
This high paging activity is called Thrashing.
A process is thrashing if it is spending more time paging than executing.

Part B (File-System Interface)

1. File Concept
2. Access Methods
3. Directory structure
4. File-System mounting
5. Files Sharing
6. Protection
7. File-System Structure
8. File-System implementation
9. Allocation Methods
10. Free-Space Management
11. Disk Structure
12. Disk Scheduling

****

1. File Concept

A file is a collection of related information that is recorded on secondary storage.
a file is the smallest allotment of logical secondary storage and data can not be written to secondary storage unless they are within a file.

Terms of Files are: Field, Record, File and Database

A **field** is the basic element of data. An individual field contains a single value, such as an employee's last name, a date, or the value of a sensor reading. It is characterized by its length and data type.

A **record** is a collection of related fields that can be treated as a unit by some application program. For example, an employee record would contain such fields as name, social security number, job classification, date of hire, and so on.

A **file** is a collection of similar records. The file is treated as a single entity by users and applications and may be referenced by name.

A **database** is a collection of related data. A database may contain all of the information related to an organization or project, such as a business or a scientific study. The database itself consists of one or more types of files.

Attributes of File are:

Name: The symbolic file name is the only information kept in human readable form.

Identifier: This unique tag, usually a number, identifies the file within the file system. it is the non-human-readable name for the file.

Type: This information is needed for those systems that support different types.

Location: This information is a pointer to a device and to the location of the file on that device.

Size: The current size of the file (in bytes, words, or blocks), and possibly the maximum allowed size are included in this attribute.

Protection: Access-control information determines who can do reading, writing, executing, and so on.

Time, date, and user identification: This information may be kept for creation, modification and last use. These data can be useful for protection, security, and usage monitoring.

Operations of File are :

1. Creating a file
2. Writing a file
3. Reading a file
4. Repositioning within a file
5. Deleting a file
6. Truncating a file

File Types:

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

2. Access Methods

When a file is used, this information must be accessed and read into computer memory. The information in the file can be accessed in several ways. There are two major access methods as follows:

Sequential Access: Information in the file is processed in order, one record after the other. A read operation reads the next portion of the file and automatically advances a file pointer,

which tracks the I/O location. Similarly, a write appends to the end of the file and advances to the end of the newly written material (the new end of file). Sequential access is based on a tape model of a file, and works as well on sequential-access devices as it does on random-access ones.

Direct Access: A file is made up of fixed length **logical records** that allow programs to read and write records rapidly in no particular order. The direct-access method is based on a disk model of a file, since disks allow random access to any file block. For direct access, the file is viewed as a numbered sequence of blocks or records. A direct-access file allows arbitrary blocks to be read or written. There are no restrictions on the order of reading or writing for a direct-access file. For the direct-access method, the file operations must be modified to include the block number as a parameter. Thus, we have *read n,* where *n* is the block number, rather than *read next,* and *write n* rather than *write next.*

3. Directory structure

A directory is an object that contains the names of file system objects. File system allows the users to organize files and other file system objects through the use of directories. The structure created by placement of names in directories can take a number of forms: Single-level tree, Two-level tree, multi-level tree or cyclic graph.

1. Single-Level Directory: The simplest directory structure is the single-level directory. All files are contained in the same directory, which is easy to support and understand. A single-level directory has significant limitations, when the number of files increases or when the system has more than one user. Since all files are in the same directory, they must have unique names.

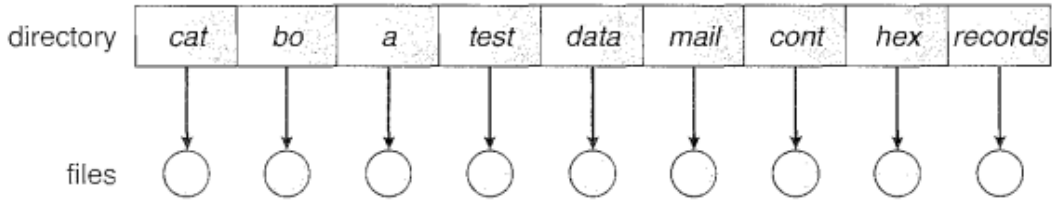


Figure 10.8 Single-level directory.

2. Two-Level Directory: In the two-level directory structure, each user has its own **user file directory (UFD).** Each UFD has a similar structure, but lists only the files of a single user. When a user job starts or a user logs in, the system's **master file directory (MFD)** is searched. The MFD is indexed by user name or account number, and each entry points to the UFD for that user.

When a user refers to a particular file, only his own UFD is searched. Different users may have files with the same name, as long as all the file names within each UFD are unique.

To create a file for a user, the operating system searches only that user's UFD to ascertain whether another file of that name exists. To delete a file, the operating system confines its search to the local UFD; thus, it cannot accidentally delete another user's file that has the same name.

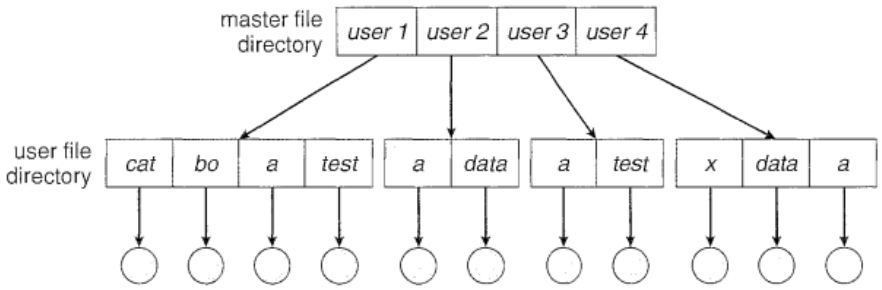


Figure 10.9 Two-level directory structure.

3.Tree-structured directories: A tree structure is A more powerful and flexible approach to organize files and directories in hierarchical. There is a master directory, which has under it a number of user directories. Each of these user directories may have sub- directories and files as entries. This is true at any level: That is, at any level, a directory may consist of entries for subdirectories and/or entries for files.

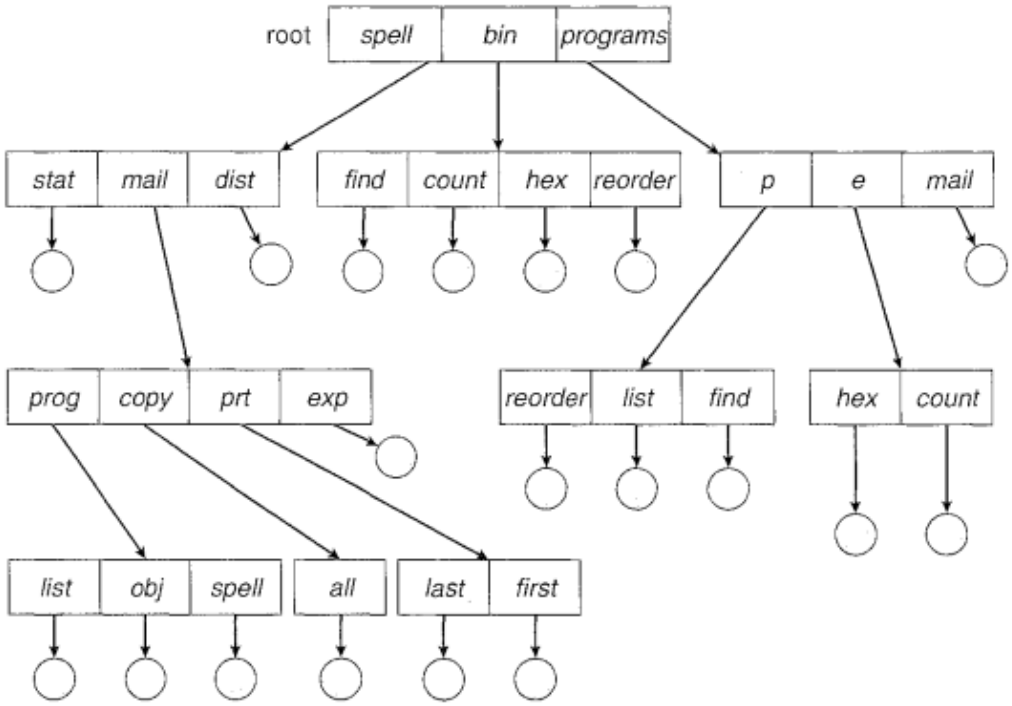


Figure 10.10 Tree-structured directory structure.

4. Acyclic-Graph Directories :

An **acyclic graph** allows directories to have shared subdirectories and files. The

same file or subdirectory may be in two different directories. An acyclic graph is a natural generalization of the tree structured directory scheme.

A shared file (or directory) is not the same as two copies of the file. With two copies, each programmer can view the copy rather than the original, but if one programmer changes the file, the changes will not appear in the other's copy.

Shared files and subdirectories can be implemented in several ways. A common way is to create a new directory entry called a link. A **link** is a pointer to another file or subdirectory.

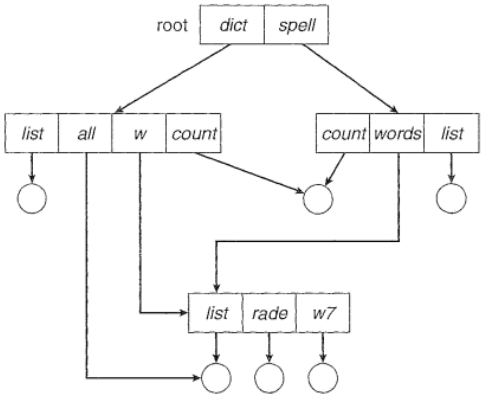


Figure 10.11 Acyclic-graph directory structure.

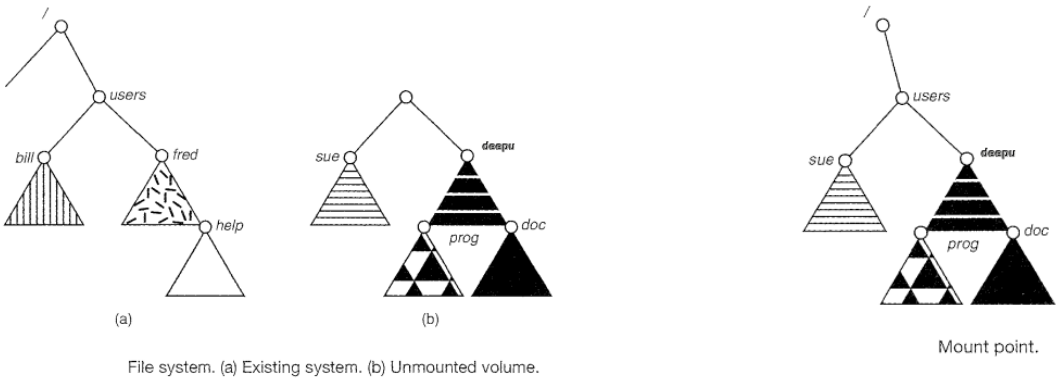
4. File-System mounting

Just as a file must be opened before it is used, a file system must be mounted before it can be available to processes on the system.

More specifically, the directory structure may be built out of multiple volumes, which must be mounted to make them available within the file-system name space.

The operating system is given the name of the device and the mount point-location within the file structure where the file system is to be attached.

Typically, a mount point is an empty directory. For instance, on a UNIX system, a file system containing a user's home directories might be mounted as /home. then, to access the directory structure within that file system, we could precede the directory names with /home, as in /home/deepu.



File system. (a) Existing system. (b) Unmounted volume.

5. Files Sharing

File sharing is very desirable for users who want to collaborate and to reduce the effort required to achieve a computing goal.

File sharing is used for

a. Multiple users.

b. Remote File systems.

Client-Server Model

Distributed Information Systems

Failure Modes

c. Consistency Semantics

UNIX Semantics

Session Semantics

Immutable-Shared-Files Semantics

6. Protection

Protection can be provided in many ways. For a small single-user system, we might provide protection by physically removing the floppy disks and locking them in a desk drawer or file cabinet. In a multiuser system, however, other mechanisms are needed, they are

a. Types of Access :

Systems that do not permit access to the files of other users do not need protection. Thus,

we could provide complete protection by prohibiting access depending on

several factors for different types of operations, they are,

Read : Read from the file.

Write : Write or rewrite the file.

Execute : Load the file into memory and execute it.

Append : Write new information at the end of the file.

Delete : Delete the file and free its space for possible reuse.

List : List the name and attributes of the file.

b. Access Control :

The most common approach to the protection problem is to make access dependent on the identity of the user. Different users may need different types of access to a file or directory.

To condense the length of the access-control list, many systems recognize three classifications of users in connection with each file, they are

Owner : The user who created the file is the owner.

Group : A set of users who are sharing the file and need similar access is a group.

Universe : All other users in the system constitute the universe.

7. File-System Structure

Disks provide the bulk of secondary storage on which a file system is maintained. They have two characteristics that make them a convenient medium for storing multiple files.

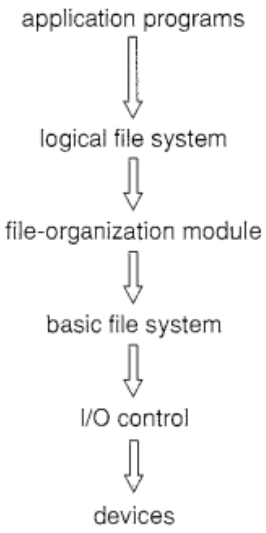
1. A disk can be rewritten in place. it is possible to read a block from the disk, modify the block, and write it back into the same place.

2. A disk can access directly any block of information it contains. Thus, it is simple to access any file either sequentially or randomly, and switching from one file to another requires only moving the read-write heads and waiting for the disk to rotate.

To improve I/0 efficiency, I/0 transfers between memory and disk are performed in units of blocks. Each block has one or more sectors. Depending on the disk drive, sector size varies from 32 bytes to 4,096 bytes; the usual size is 512 bytes.

File systems provide efficient and convenient access to the disk by allowing data to be stored, located, and retrieved easily.

The file system itself is generally composed of many different levels. Each level in the design uses the features of lower levels to create new features for use by higher levels.



9. Allocation Methods

There are various methods which can be used to allocate disk space to the files.

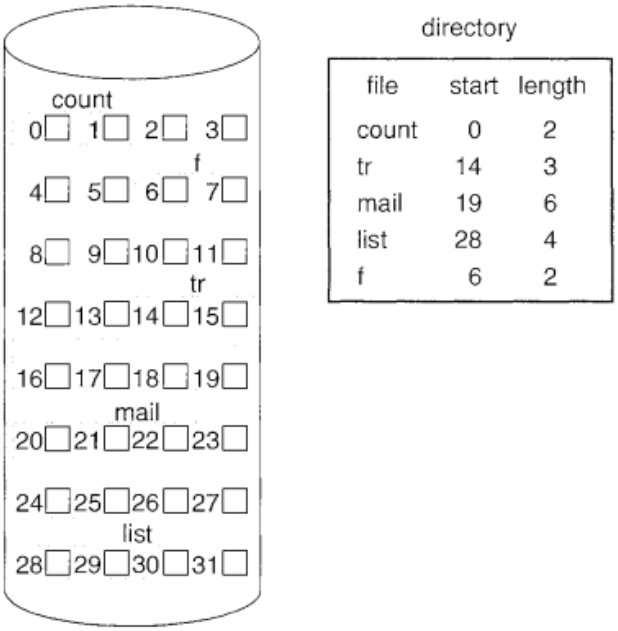
Selection of an appropriate allocation method will significantly affect the performance and efficiency of the system.

Allocation method provides a way in which the disk will be utilized and the files will be accessed, they are

1. Contiguous Allocation
2. Linked Allocation
3. Indexed Allocation

1. Contiguous Allocation

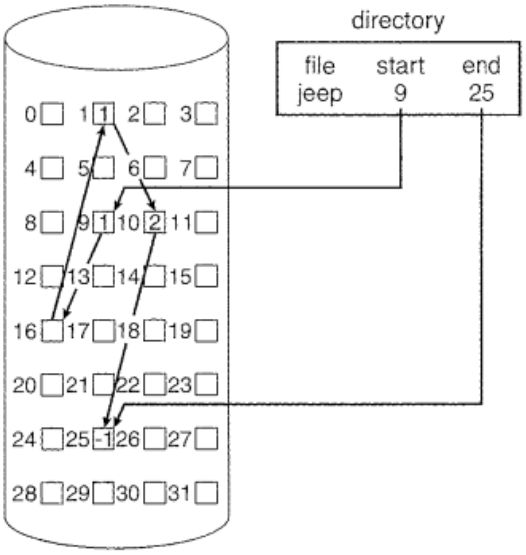
If the blocks are allocated to the file in such a way that all the logical blocks of the file get the contiguous physical block in the hard disk then such allocation scheme is known as contiguous allocation.



Contiguous allocation of disk space.

2. Linked Allocation

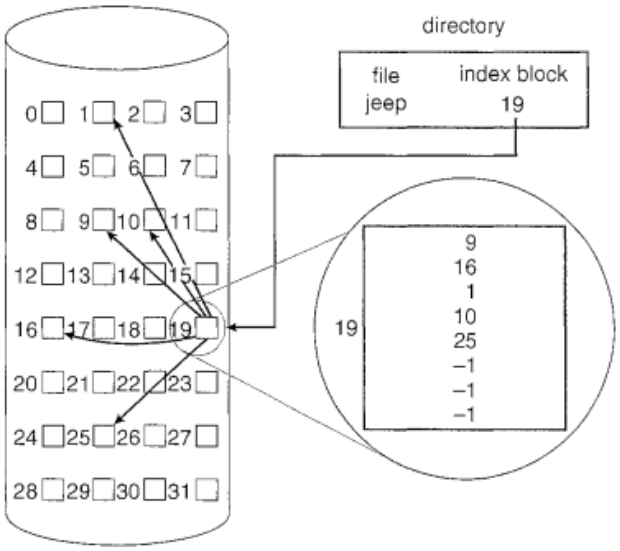
Linked List allocation solves all problems of contiguous allocation. In linked list allocation, each file is considered as the linked list of disk blocks. However, the disks blocks allocated to a particular file need not to be contiguous on the disk. Each disk block allocated to a file contains a pointer which points to the next disk block allocated to the same file.



Linked allocation of disk space.

3.Indexed allocation :

This scheme stores all the disk pointers in one of the blocks called as indexed block. Indexed block doesn't hold the file data, but it holds the pointers to all the disk blocks allocated to that particular file. Directory entry will only contain the index block address.



Indexed allocation of disk space.

12. Disk Scheduling

The **seek time** is the time for the disk arm to move the heads to the cylinder containing the desired sector. The **rotational latency** is the time waiting for the disk to rotate the desired sector to the disk head. The disk **bandwidth** is the total number of bytes transferred divided by the total time between the first request for service and the completion of the last transfer.

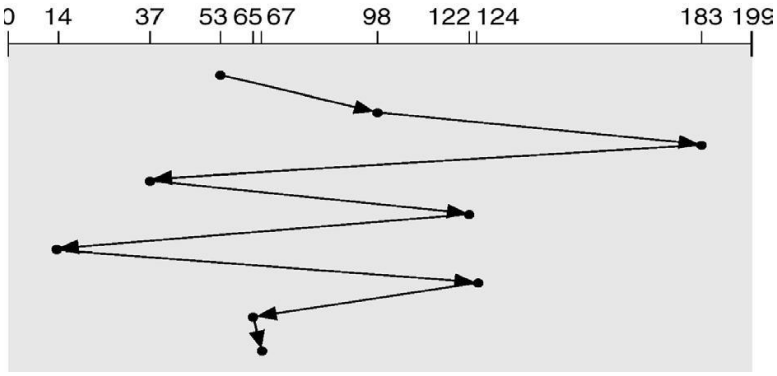
We can improve both the access time and the bandwidth by scheduling the servicing of disk I/O requests in a good order. Several algorithms exist to schedule the servicing of disk I/O requests as follows:

- **FCFS Scheduling**

The simplest form of scheduling is first-in-first-out (FIFO) scheduling, which processes items from the queue in sequential order.
We illustrate this with a request queue (0-199):

98, 183, 37, 122, 14, 124, 65, 67

Consider now the Head pointer is in cylinder 53.



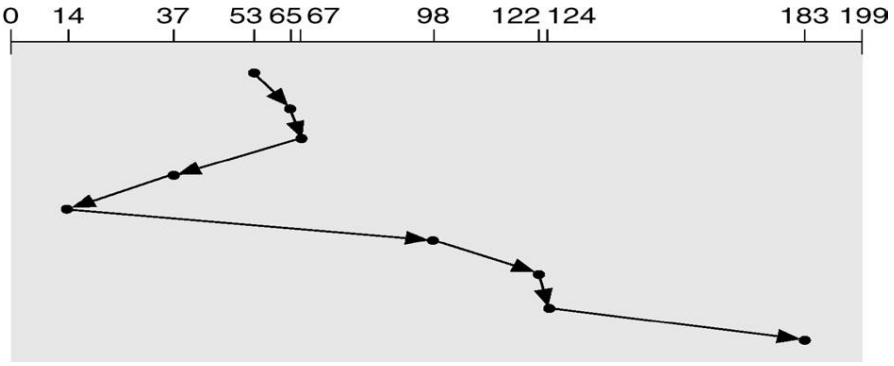
- **SSTF Scheduling**

It stands for shortest-seek-time-first (SSTF) algorithm. The SSTF algorithm selects the request with the minimum seek time from the current head position. Since seek time increases with the number of cylinders traversed by the head, SSTF chooses the pending request closest to the current head position.

We illustrate this with a request queue (0-199):

98, 183, 37, 122, 14, 124, 65, 67

Consider now the Head pointer is in cylinder 53.



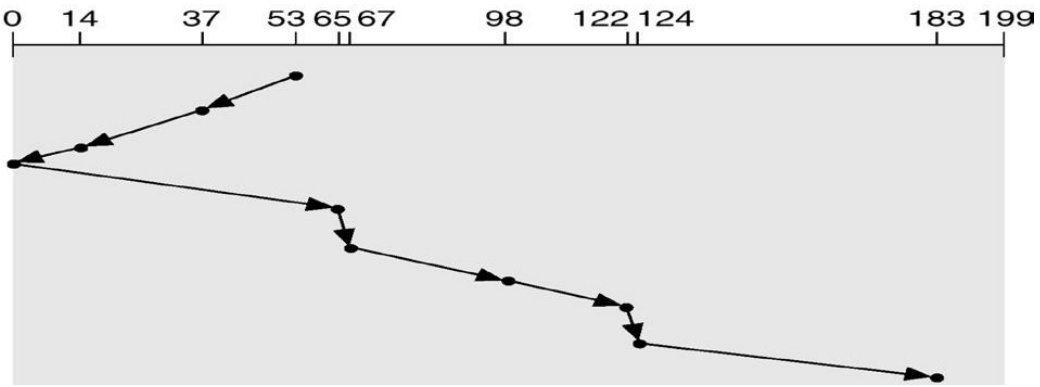
- **SCAN Scheduling**

In the SCAN algorithm, the disk arm starts at one end of the disk, and moves toward the other end, servicing requests as it reaches each cylinder, until it gets to the other

end of the disk. At the other end, the direction of head movement is reversed, and servicing continuNes. The head continuously scans back and forth across the disk.

We illustrate this with a request queue (0-199):
98, 183, 37, 122, 14, 124, 65, 67

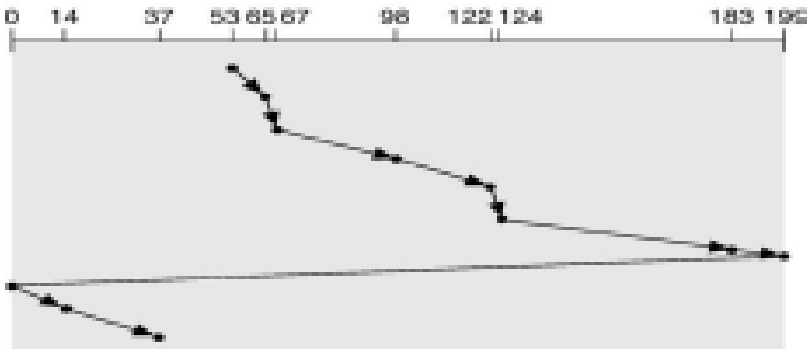
Consider now the Head pointer is in cylinder 53.



- **C-SCAN Scheduling**

Circular **SCAN (C-SCAN)** scheduling is a variant of SCAN designed to provide a more uniform wait time. Like SCAN, C-SCAN moves the head from one end of the disk to the other, servicing requests along the way. When the head reaches the other end, it immediately returns to the beginning of the disk, without servicing any requests on the return trip. The C- SCAN scheduling algorithm essentially treats the cylinders as a circular list that wraps around from the final cylinder to the first one.

We illustrate this with a request queue (0-199):
98, 183, 37, 122, 14, 124, 65, 67. C
Consider now the Head pointer is in cylinder 53.



- **LOOK Scheduling**

Practically, both SCAN and C-SCAN algorithm is not implemented this way. More

commonly, the arm goes only as far as the final request in each direction. Then, it reverses direction immediately, without going all the way to the end of the disk. These versions of SCAN and C-SCAN are called **LOOK** and **C-LOOK** scheduling, because they look for a request before continuing to move in a given direction.

We illustrate this with a request queue (0-199): 98, 183, 37, 122, 14, 124, 65, 67
Consider now the Head pointer is in cylinder 53.