KTU NOTES

The learning companion.

KTU STUDY MATERIALS | SYLLABUS | LIVE NOTIFICATIONS | SOLVED QUESTION PAPERS

Website: www.ktunotes.in

Module 5

Contents

- File System: File concept - Attributes, Operations, types, structure – Access methods, Protection.
- File-system implementation, Directory implementation. Allocation methods.
- Storage Management: Magnetic disks, Solid-state disks, Disk Structure, Disk scheduling, Disk formatting.

File Concept

- Computers can store information on various storage media.
- The operating system abstracts from the physical properties of its storage devices to define a logical storage unit called a **file**
- Files are **mapped** by the operating system onto nonvolatile physical devices.
- A file is a **named collection** of **related information** that is recorded on secondary storage.
- For a user's perspective, a file is the **smallest allotment** of logical secondary storage.
- Files commonly represent programs (both source and object code) and data.
- Data file contents may be numeric, alphabetic, alphanumeric, or binary
- Files may be **free form** (e.g., text files) or **rigidly formatted**
- In general, a file is a **sequence** of bits, bytes, lines, or records, whose meaning is defined by the file's creator and user .

File Concept

- A file has a certain defined structure, which depends on its type.
- A **text file is a sequence of characters organized** into lines (and possibly pages).
- A **source file is a sequence of functions, each of** which is further organized as declarations followed by executable statements.
- An **executable file is a series of code sections that the loader can bring into** memory and execute.

File Attributes

- A file's attributes vary from one operating system to another but typically consist of these:
- **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 systems that support different types of files.
- **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, last modification, and last use. These data can be useful for protection, security, and usage monitoring.

File Operations

- The operating system can provide system calls to create, write, read, reposition, delete, and truncate files.
- **Creating a file.** File is created with no data.

- **Writing a file.** To write a file, make a system call specifying both the name of the file and the information to be written to the file. Given the name of the file, the system searches the directory to find the file's location. The system must keep a write pointer to the location in the file where the next write is to take place. The write pointer must be updated whenever a write occurs.

- **Reading a file.** To read from a file, use a system call that specifies the name of the file and where (in memory) the next block of the file should be put. The directory is searched for the associated entry, and the system needs to keep a read pointer to the location in the file where the next read is to take place. Once the read has taken place, the read pointer is updated.

File Operations

- **Repositioning within a file.** The current-file-position pointer is repositioned to a given value.This file operation is also known as a file seek . Used for random access.
- **Deleting a file.** To delete a file, we search the directory for the named file. Having found the associated directory entry, we release all file space, so that it can be reused by other files, and erase the directory entry.
- **Truncating a file.** The user may want to erase the contents of a file but keep its attributes. This function allows all attributes to remain unchanged—except for file length—but lets the file be reset to length zero and its file space released.
- These primitive operations can then be combined to perform other file operations.

File Operations

- Many systems require that an open() system call be made before a file is first used.
- The operating system keeps a table, called the **open-file table, containing** information about all open files.
- Typically, the operating system uses two levels of internal tables:

  1) a per-process table

  2) a system-wide table.

1) Per-process table

- The per process table tracks all files that a process has open.
- This table is storing the information regarding the process's use of the file.
- The current file pointer for each file is found here.
- Access rights to the file and accounting information can also be included.

File Operations

2)System-wide table.

- Each entry in the per-process table in turn points to a system-wide open-file table.
- The system-wide table contains process-independent information, such as the location of the file on disk, access dates, and file size.
- Once a file has been opened by one process, the system-wide table includes an entry for the file.
- Typically, the open-file table also has an **open count** associated with each file to indicate how many processes have the file open.

Informations associated with an open file.

- **File pointer.** The last read– write location in a file will be pointed to by a current-file-position pointer. This pointer is unique to each process operating on the file and therefore must be kept separate from the on-disk file attributes.

- **File-open count:** Multiple processes may have opened a file. The file-open count tracks the number of opens and closes and reaches zero on the last close. The system can then remove the entry.

- **Disk location of the file**:Most file operations require the system to modify data within the file. The information needed to locate the file on disk is kept in memory so that the system does not have to read it from disk for each operation.

- **Access rights.** Each process opens a file in an access mode. This information is stored on the per-process table so the operating system can allow or deny subsequent I/O requests.

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, perl, asm	source code in various languages
batch	bat, sh	commands to the command interpreter
markup	xml, html, tex	textual data, documents
word processor	xml, rtf, docx	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	gif, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	rar, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, mp3, mp4, avi	binary file containing audio or A/V information

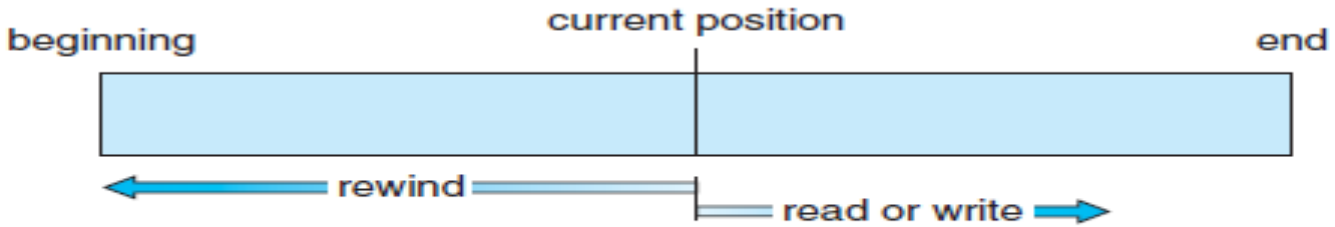
Common file types.

File Access Methods

- Files store information. When it is used, this information must be accessed and read into computer memory.
- The information in the file can be accessed in several ways.

1)Sequential Access

- The simplest access method is **sequential access.**
- **Information in the file is** processed in order, one record after the other.
- Editors and compilers usually access files in this fashion.
- A read operation—read next()—reads the next portion of the file and automatically advances a file pointer.
- The write operation—write next()—appends to the end of the file and advances to the end of the newly written material .



Sequential-access file.

2)Direct Access

- Here, a file is made up of fixed-length logical records that allow programs to read and write records rapidly in no particular order.
- For direct access, the file is viewed as a numbered sequence of blocks or records.
- We may read block 14, then read block 53, and then write block 7.
- There are no restrictions on the order of reading or writing for a direct-access file.

- Databases are often direct access .
- When a query concerning a particular subject arrives, we compute which block contains the answer and then read that block directly to provide the desired information.
- Read operation for direct access is read(n), where *n is the block number*
- *Write* operation for direct access is *write(n)*

3) Other Access Methods

- Other access methods can be built on top of a direct-access method.
- These methods generally involve the construction of an index for the file.
- The **index,** contains pointers to the various blocks.
- To find a record in the file, we first search the index and then use the pointer to access the file directly and to find the desired record.

- With large files, the index file itself may become too large to be kept in memory.
- One solution is to create an index for the index file.
- The primary index file contains pointers to secondary index files, which point to the actual data items.

Allocation Methods

Allocation Methods

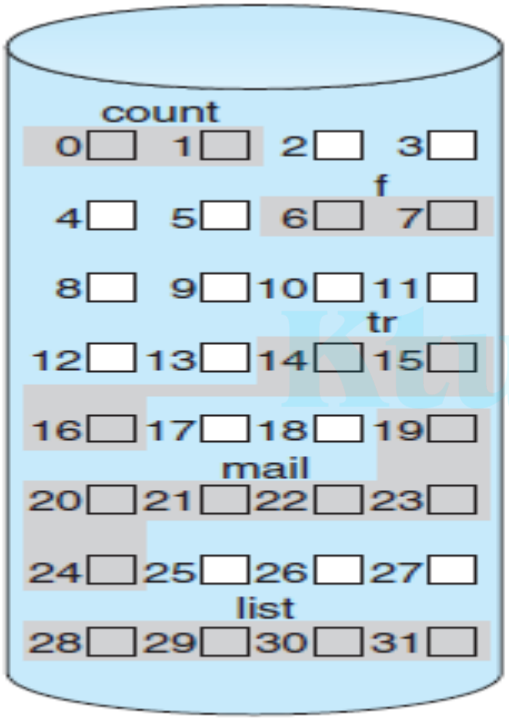
1)Contiguous Allocation

2)Linked Allocation

3)Indexed Allocation

- Contiguous allocation requires that each file occupy a set of contiguous blocks on the disk.
- Contiguous allocation of a file is defined by the disk address and length (in block units) of the first block.
- If the file is *n blocks long and starts at location b, then it occupies blocks b, b + 1, b + 2, ..., b + n − 1.*
- *The directory entry for* each file indicates the address of the starting block and the length of the area allocated for this file

Allocation Methods- Contiguous Allocation



directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Contiguous allocation of disk space.

Allocation Methods- Contiguous Allocation

- Both sequential and direct access can be supported by contiguous allocation.

Problem

External fragmentation.

Solution

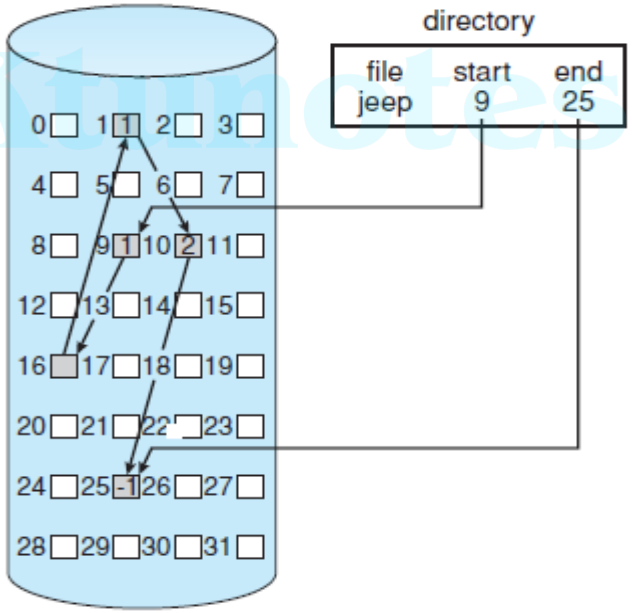
- Copy an entire file system onto another disk.
- The disk is then freed completely, creating one large contiguous free space.
- Then copy the files back onto the original disk by allocating contiguous space from this one large hole.
- This scheme effectively **compacts all free space** into one contiguous space, solving the fragmentation problem.

Allocation Methods- Contiguous Allocation

- Another problem with contiguous allocation is determining how much space is needed for a file.
- If we allocate too little space to a file, we may find that the file cannot be extended.
- A contiguous chunk of space is allocated initially. Then, if that amount proves not to be large enough, another chunk of contiguous space, known as an **extent, is added.**
- **The location of a file's blocks** is then recorded as a location and a block count, plus a link to the first block of the next extent.

Allocation Methods-Linked Allocation

- Linked allocation solves all problems of contiguous allocation.
- With linked allocation, each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on the disk.
- The directory contains a pointer to the first and last blocks of the file.



Linked allocation of disk space.

Allocation Methods-Linked Allocation

- Eg: A file of five blocks might start at block 9 and continue at block 16, then block 1, then block 10, and finally block 25.
- A write to the file causes the free-space management system to find a free block, and this new block is written to and is linked to the end of the file.
- To read a file, we simply read blocks by following the pointers from block to block.
- There is no external fragmentation with linked allocation, and any free block on the free-space list can be used to satisfy a request.
- The size of a file need not be declared when the file is created. A file can continue to grow as long as free blocks are available.

Allocation Methods-Linked Allocation

Disadvantages

1)

- It can be used effectively only for sequential-access files.
- To find the *i*th *block of a file, we must start at the beginning of that file and follow the* pointers until we get to the *i*th *block.*

2)

- Space requirement for the pointers.
- If a pointer requires 4 bytes out of a 512-byte block, then 0.78 percent of the disk is being used for pointers, rather than for information.
- Solution to this problem is to collect blocks into multiples, called clusters, and to allocate clusters rather than blocks

Allocation Methods-Linked Allocation

3)

Not Reliable.

- A bug in the operating-system software or a disk hardware failure might result in picking up the wrong pointer.
- This error could in turn result in linking into the free-space list or into another file.

Solutions

- Use doubly linked lists.
- Store the file name and relative block number in each block.

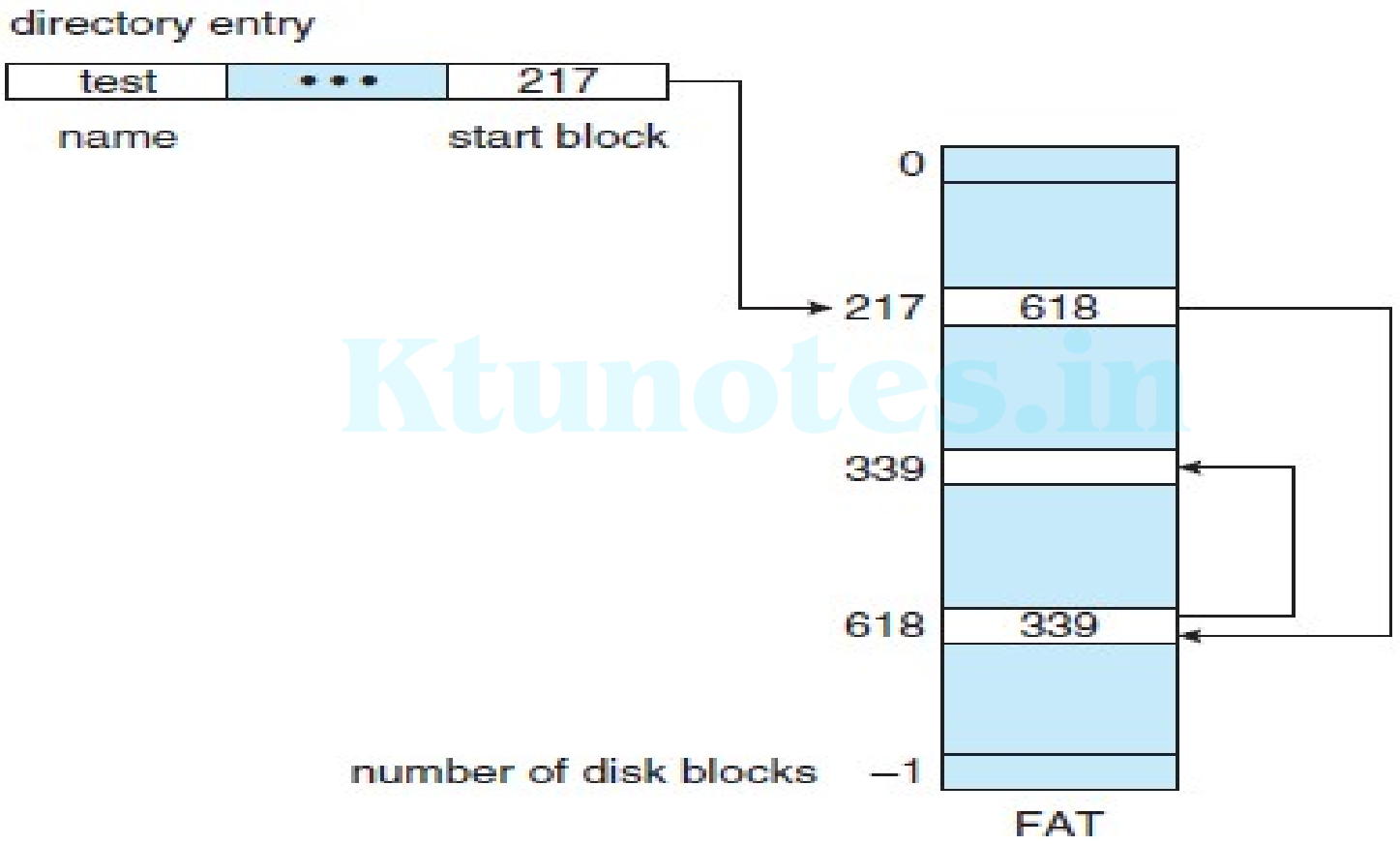
Allocation Methods-Linked Allocation

- Variation on linked allocation is the use of a **file-allocation table (FAT).**

File-allocation table (FAT)

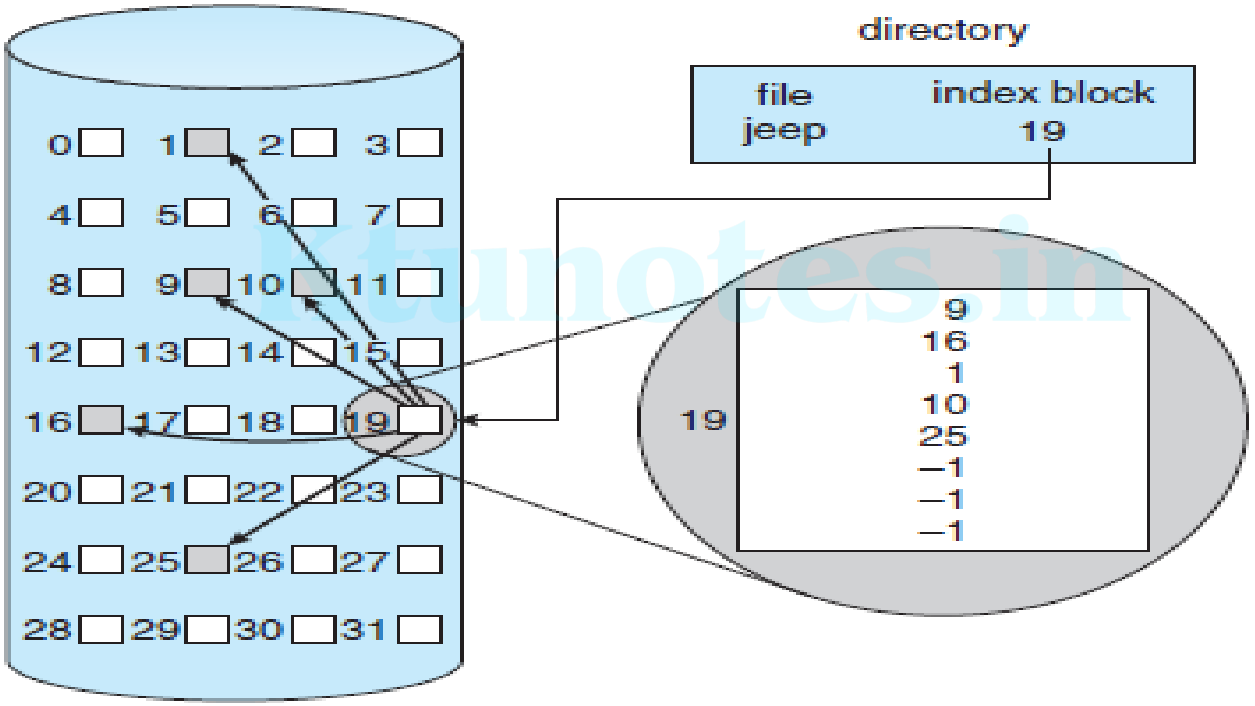
- A section of disk at the beginning of each volume is set aside to contain the table.
- The table has one entry for each disk block and is indexed by block number.
- The directory entry contains the block number of the first block of the file.
- The table entry indexed by that block number contains the block number of the next block in the file.
- This chain continues until it reaches the last block, which has a special end-of-file value as the table entry.
- An unused block is indicated by a table value of 0.

Allocation Methods-Linked Allocation



File-allocation table.

Allocation Methods-Indexed Allocation



Indexed allocation of disk space.

Allocation Methods-Indexed Allocation

- Each file has its own index block, which is an array of disk-block addresses.
- The *ith entry in the index block points to the ith block of the file.*
- *The directory* contains the address of the index.
- To find and read the *ith* block, we use the pointer in the *ith index-block entry.*
- When the file is created, all pointers in the index block are set to null.
- When the *ith block is first written, a block is obtained from the free-space* manager, and its address is put in the *ith index-block entry.*

Allocation Methods-Indexed Allocation

- Indexed allocation supports direct access, without suffering from external fragmentation.

Disadvantage

- The pointer overhead of the index block is generally greater than the pointer overhead of linked allocation.

Linked scheme.

- An index block is normally one disk block.
- Thus, it can be read and written directly by itself.
- To allow for large files, we can link together several index blocks.

Allocation Methods-Indexed Allocation

Multilevel index.

- A variant of linked representation uses a first-level index block to point to a set of second-level index blocks, which in turn point to the file blocks.
- To access a block, the operating system uses the first-level index to find a second-level index block and then uses that block to find the desired data block

Allocation Methods-Indexed Allocation

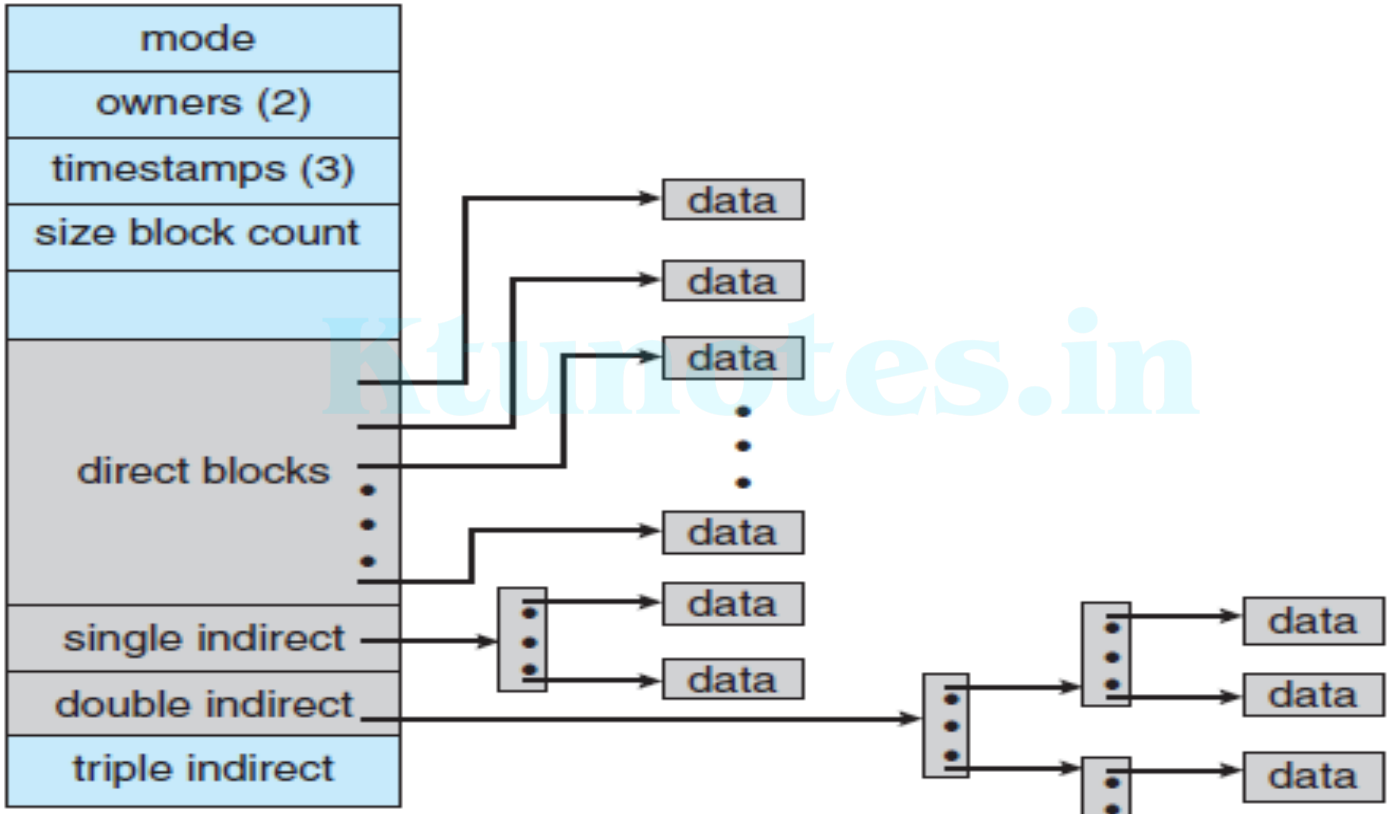
Combined scheme.

- Keep the first, say, 15 pointers of the index block in the file's inode.
- The first 12 of these pointers point to direct blocks; that is, they contain addresses of blocks that contain data of the file.
- Thus, the data for small files (of no more than 12 blocks) do not need a separate index block.
- If the block size is 4 KB, then up to 48 KB of data can be accessed directly.

Allocation Methods-Indexed Allocation

- The next three pointers point to indirect blocks.
- The first points to a single indirect block, which is an index block containing not data but the addresses of blocks that do contain data.
- The second points to a **double indirect block,** which contains the address of a block that contains the addresses of blocks that contain pointers to the actual data blocks.
- The last pointer contains the address of a **triple indirect block.**

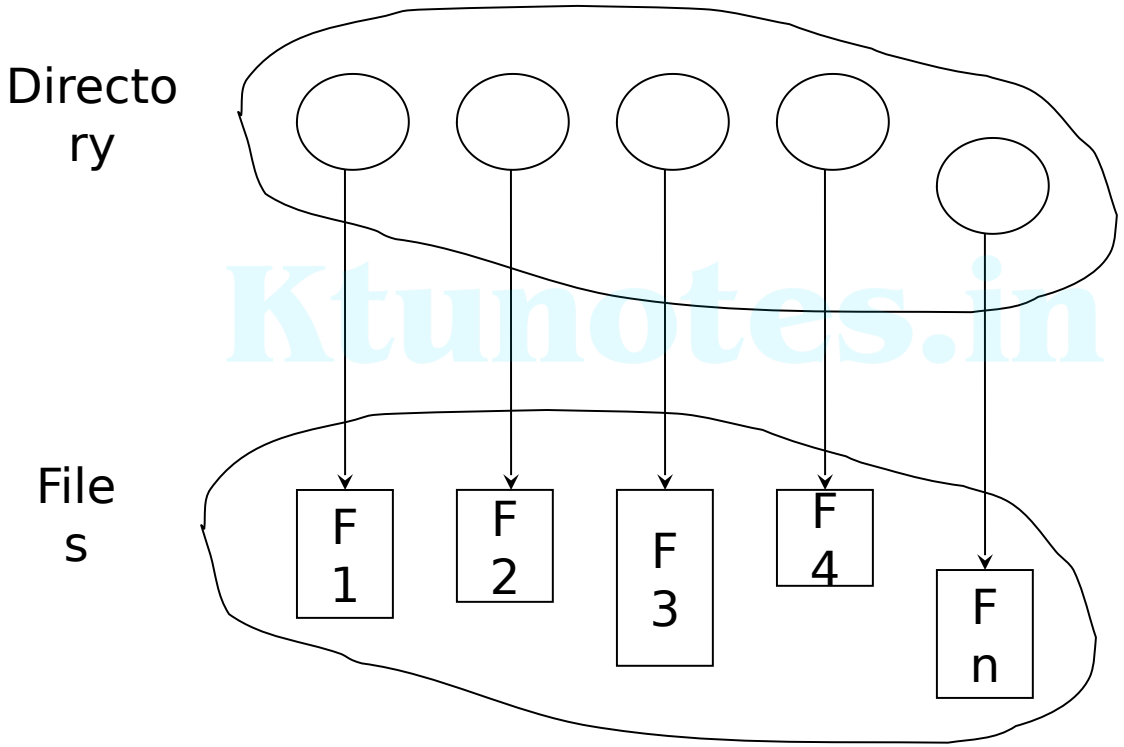
Allocation Methods-Indexed Allocation



The UNIX inode.

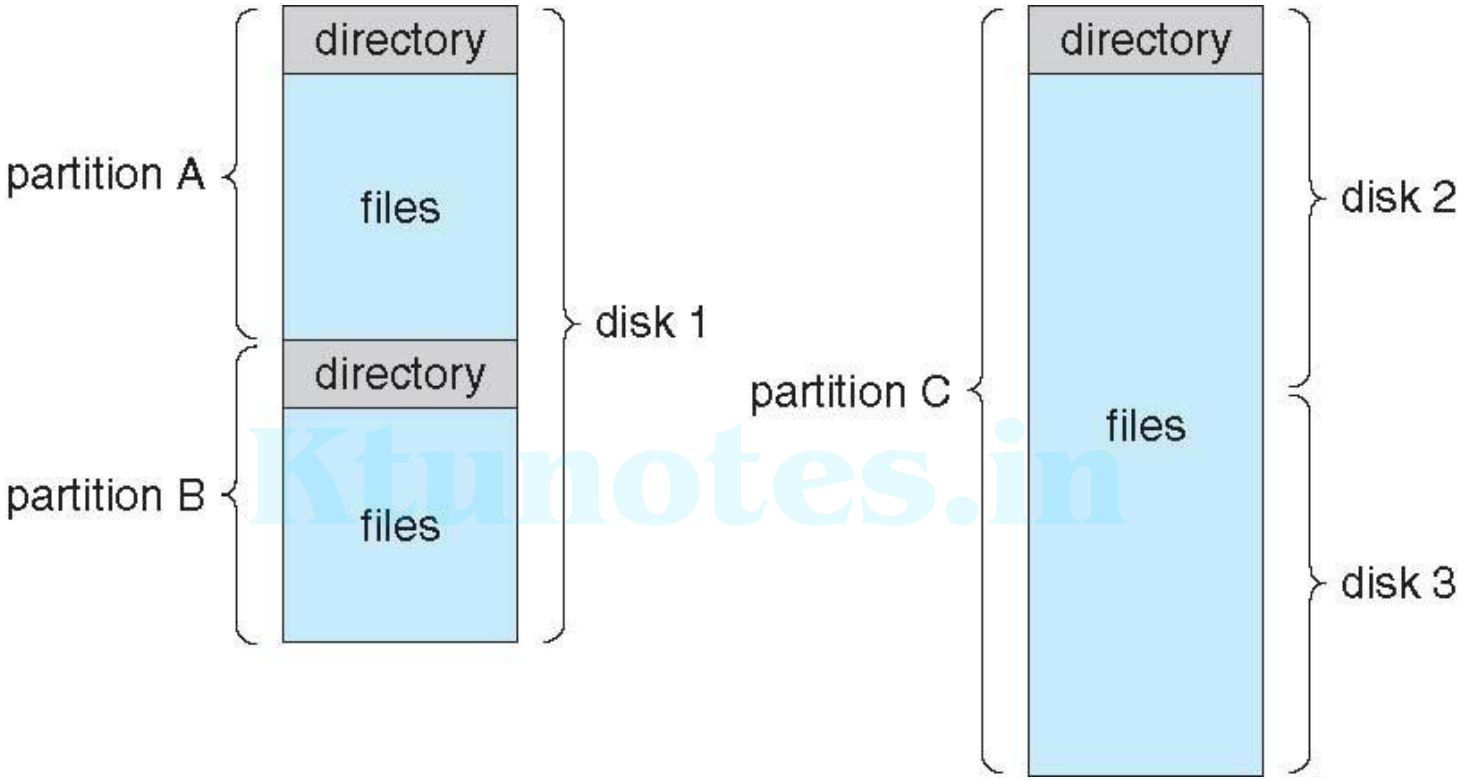
Directory Structure

- A collection of nodes containing information about all files



Both the directory structure and the files reside on disk

A Typical File-system Organization



Types of File Systems

- We mostly talk of general-purpose file systems
- But systems frequently have may file systems, some general- and some special- purpose
- Consider Solaris has
  - tmpfs – memory-based volatile FS for fast, temporary I/O
  - objfs – interface into kernel memory to get kernel symbols for debugging
  - ctfs – contract file system for managing daemons
  - lofs – loopback file system allows one FS to be accessed in place of another
  - procfs – kernel interface to process structures
  - ufs, zfs – general purpose file systems

Operations Performed on Directory

- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

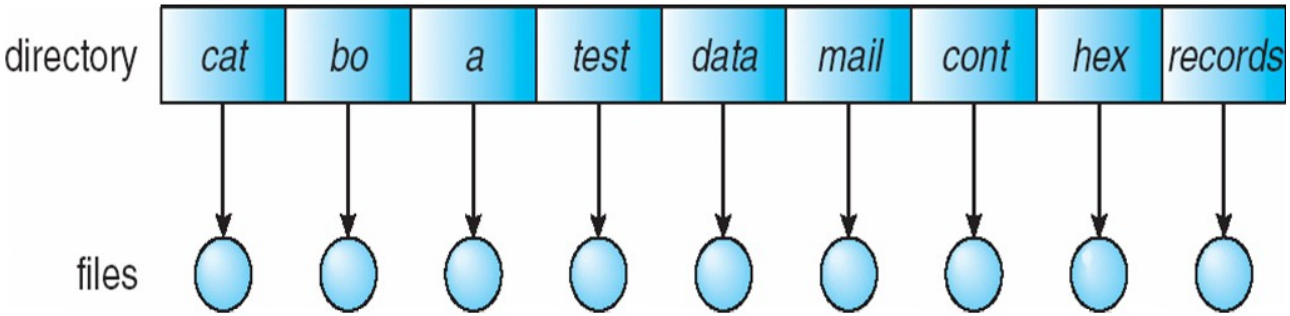
Directory Organization

The directory is organized logically to obtain

- Efficiency – locating a file quickly
- Naming – convenient to users
  - Two users can have same name for different files
  - The same file can have several different names
- Grouping – logical grouping of files by properties, (e.g., all Java programs, all games, ...)

Single-Level Directory

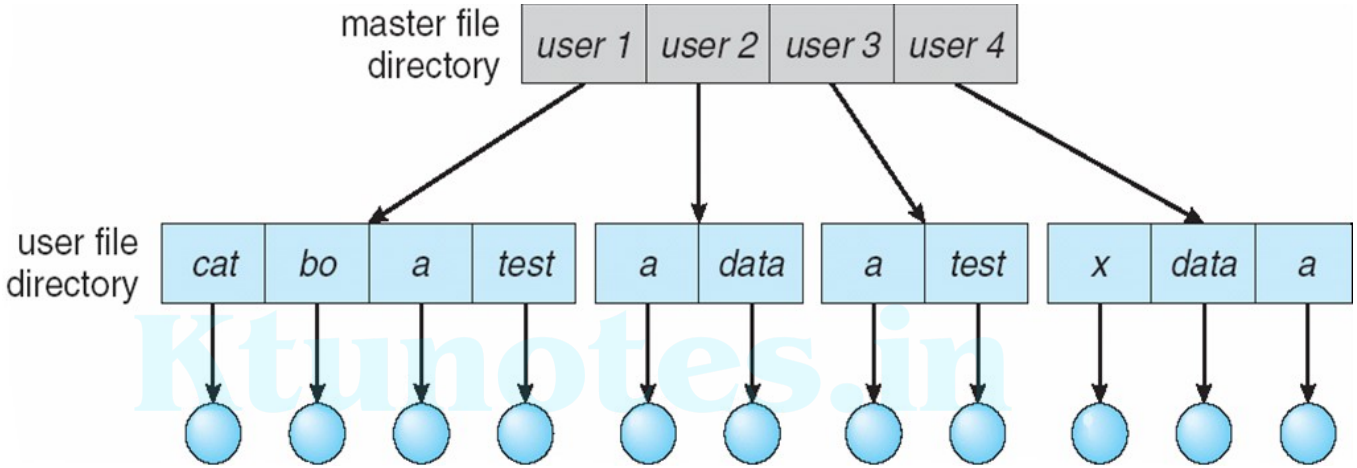
- A single directory for all users



- Naming problem
- Grouping problem

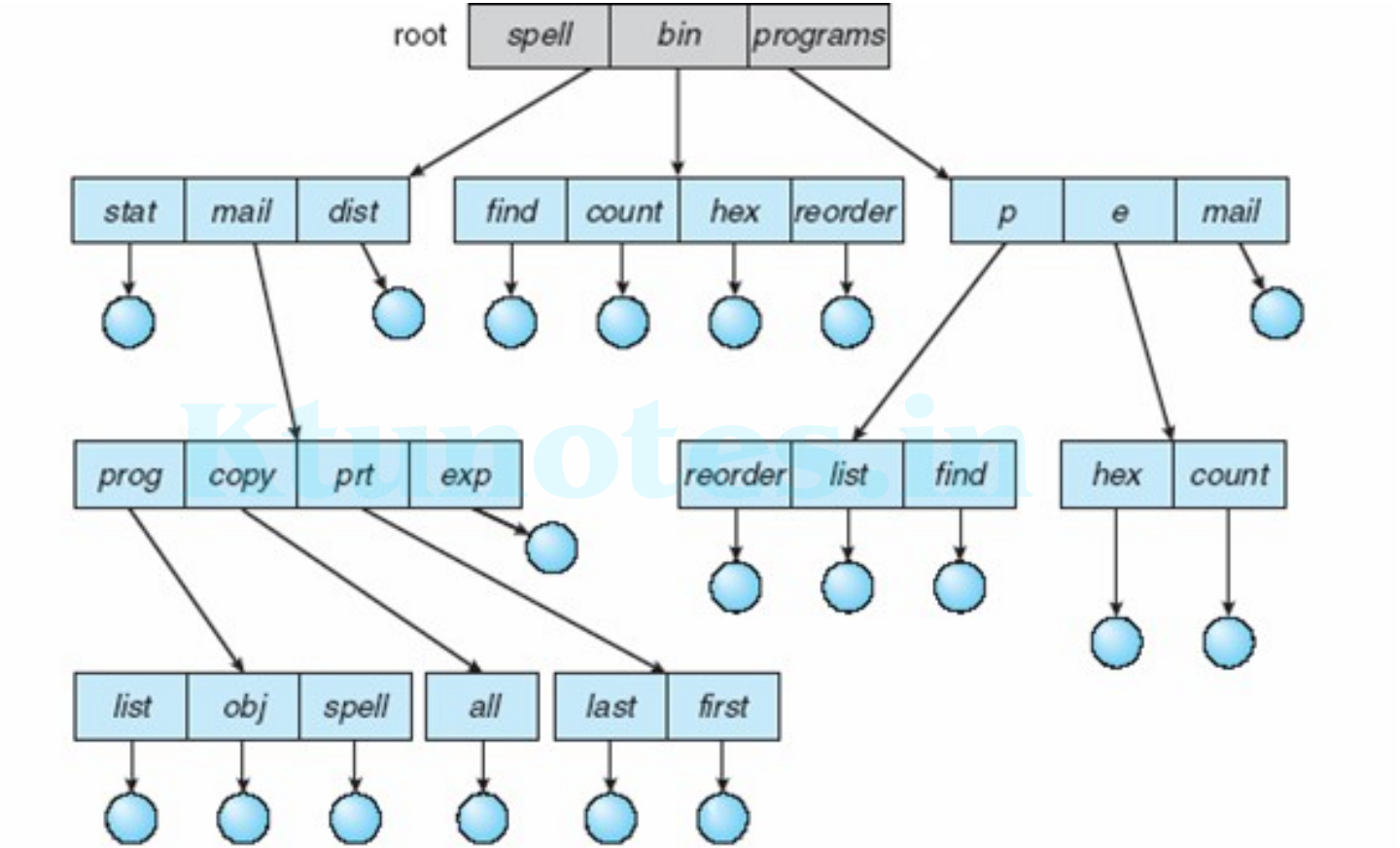
Two-Level Directory

- Separate directory for each user



- Path name
- Can have the same file name for different user
- Efficient searching
- No grouping capability

Tree-Structured Directories



Protection

Goals of Protection

- Prevent the mischievous, intentional violation of an access restriction by a user.
- Need to ensure that each program component active in a system uses system resources only in ways consistent with stated policies.
- An unprotected resource cannot defend against use (or misuse) by an unauthorized or incompetent user.

Goals of Protection

- The role of protection in a computer system is to provide a **mechanism for the enforcement of the policies** governing resource use.
- These policies can be established in a variety of ways.
  - Some are fixed in the design of the system, while others are formulated by the management of a system.
  - Some are formulated by the individual users to protect their own files and programs.
- A protection system must have the flexibility to enforce a variety of policies.

Goals of Protection

- Policies for resource use may vary by application, and they may change over time.
- *Mechanisms are distinct from policies.*
- *Mechanisms determine how something will be done; policies decide what will be done.*
- Policies are likely to change from place to place or time to time.
- In the worst case, every change in policy would require a change in the underlying mechanism.

Principles of Protection

Principle of least privilege

- It dictates that programs, users, and even systems be given just enough privileges to perform their tasks.
- An operating system following the principle of least privilege implements its features, programs, system calls, and data structures so that failure or compromise of a component does the minimum damage and allows the minimum damage to be done.
- Such an operating system also provides system calls and services that allow applications to be written with fine-grained access controls.
- It provides mechanisms to enable privileges when they are needed and to disable them when they are not needed

Principles of Protection

- Managing users with the principle of least privilege entails creating a separate account for each user, with just the privileges that the user needs.
- Computers implemented in a computing facility under the principle of least privilege can be limited to running specific services, accessing specific remote hosts via specific services, and doing so during specific times

Domain of Protection

- A computer system is a collection of processes and objects.
- *Objects*, *means* both hardware objects (such as the CPU,memory segments,printers,disks,and tape drives) and software objects (such as files, programs, and semaphores).

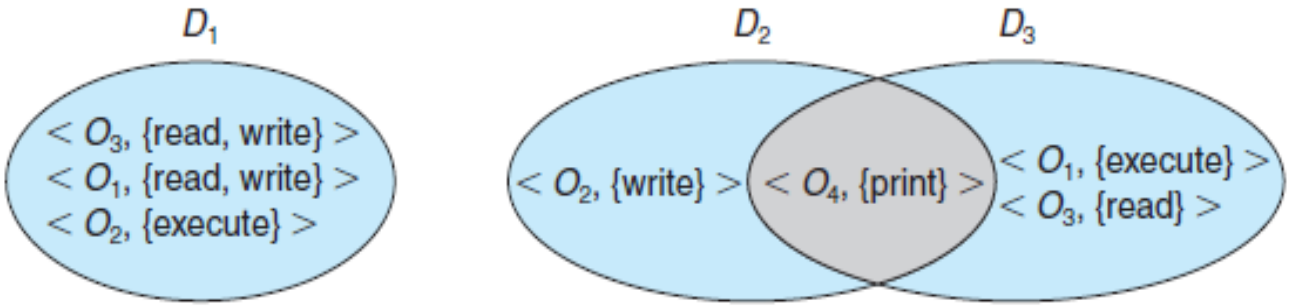
- Each object has a unique name that differentiates it from all other objects in the system, and each can be accessed only through well-defined and meaningful operations.

- A process should be allowed to access only those resources for which it has authorization.

Domain of Protection

Domain Structure

- A domain is a collection of access rights, each of which is an ordered pair <object-name, rights-set>.
- For example, if domain *D has the access* right < file *F*, *{read,write}*>, *then a process executing in domain D can both* read and write file *F. It cannot, however, perform any other operation on that* object.



System with three protection domains.

Domain of Protection

- A domain can be realized in a variety of ways:
  - Each ***user may be a domain. In this case, the set of objects that can be*** accessed depends on the identity of the user. Domain switching occurs when the user is changed—generally when one user logs out and another user logs in.
  - Each ***process may be a domain. In this case, the set of objects that can be*** accessed depends on the identity of the process. Domain switching occurs when one process sends a message to another process and then waits for a response.
  - Each ***procedure may be a domain. In this case, the set of objects that can be*** accessed corresponds to the local variables defined within the procedure. Domain switching occurs when a procedure call is made.

Access Matrix

- General model of protection can be viewed abstractly as a matrix, called an **access matrix.**
- The rows of the access matrix represent domains, and the columns represent objects.
- Each entry in the matrix consists of a set of access rights.
- The entry access(*i,j) defines the set of operations* that a process executing in domain *Di can invoke on object Oj* .

Access Matrix

object / domain	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

Access matrix.

- There are four domains and four objects—three files (*F1*, *F2*, *F3) and one* laser printer.
- A process executing in domain *D1 can read files F1 and F3.*
- *A* process executing in domain *D4 has the same privileges as one executing in* domain *D1; but in addition, it can also write onto files F1 and F3.*
- *The laser* printer can be accessed only by a process executing in domain *D2.*

Access Matrix

- **Processes should be able to switch from one domain to another**.
- Switching from domain *Di to domain Dj is allowed if and only if the access right switch* ∈ access(*i*, *j*).

object / domain	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

- Process executing in domain *D2 can switch* to domain *D3 or to domain D4. A process in domain D4 can switch to D1, and* one in domain *D1 can switch to D2.*

Access Matrix

- The ability to **copy an access right from one domain (or row) of the access matrix to another** is denoted by an asterisk (*) appended to the access right.
- The copy right allows the access right to be copied only within the column (that is, for the object) for which the right is defined.

object / domain	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute		

(a)

object / domain	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute	read	

(b)

Access matrix with ***copy*** rights.

Access Matrix

- In the above Figure (a) a process executing in domain *D2 can copy the read operation into any* entry associated with file *F2*

Access Matrix

- **Addition of new rights and removal of some rights. The owner right controls these operations**.
- If access(*i*, *j*) *includes* the owner right, then a process executing in domain *Di can add and remove* any right in any entry in column *j*.

domain \ object	F_1	F_2	F_3
D_1	owner execute		write
D_2		read* owner	read* owner write
D_3	execute		

(a)

domain \ object	F_1	F_2	F_3
D_1	owner execute		write
D_2		owner read* write*	read* owner write
D_3		write	write

(b)

Access matrix with owner rights.

Access Matrix

- Figure (a), domain *D1* is the owner of *F1 and thus can add and delete any valid right in column F1.*
- Similarly, domain *D2 is the owner of F2 and F3 and thus can add and remove* any valid right within these two columns.
- Thus, the access matrix of Figure (a) can be modified to the access matrix shown in Figure (b).

Access Matrix

- The copy and owner rights allow a process to change the entries in a column.
- **The control right is applicable only to domain objects**. If access(*i*, *j*) *includes the* control right, then a process executing in domain *Di can remove any access* right from row *j*.

domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

Fig(a)

domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch control
D_3		read	execute					
D_4	write		write		switch			

modified access matrix of fig(a)