3. File system

Thursday, April 16, 2020 9:15 PM

Concept of a file system

Question
What is a file? If you can't answer, then try to identify which ones among the followings are files?

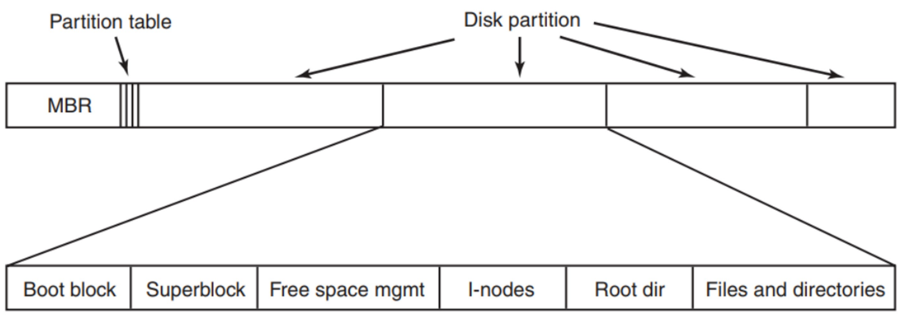
1) A document named a.txt on HDD
2) A folder named TaiLieu
3) Keyboard
4) Monitor
5) Printer
6) Storage devices such as HDD
7) Network socket

File definition in Introductory courses:

File is an object in computers which allow read/write operations via commands/calls provided by OS.

1) Document a.txt allows us to read from and write to => is a file
2) Folder Tailieu allows read/write content of the folder => is a file. This is a structured file with each record keeps the name, attributes (access permission, size, date etc.) of files within that folder.
3) Keyboard is a Read Only device, allows us to read data keyed in from the keyboard => OS consider keyboard is a file.
   Example:
   - Windows cmd: $copy\ CON\ a.txt$, copy from the keyboard file named Console to file a.txt
   - Keyboard in OS is named as the standard input device stdin = 0 as we see in C programming language:
     fscanf(stdin, "%d", &i); /* read from keyboard */
4) Monitor is a Write Only device allows us to write data to => is a file.
   Example:
   - File /dev/tty in Linux/Unix represents the output screen, we can $clear\ > /dev/tty$
   - Screen file is named as standard output stdout=1, and is also the output for standard error stderr=2, for instance
     $printf(stderr, "Lỗi\backslash n");$
5) Printer is a Write Only device, allow data to be written to => is a file.
   Example:
   - Windows cmd: $copy\ a.txt\ prn$
   - Linux/Unix: sudo cat a.txt > /dev/lp
6) All storage devices allow data to be read from / written to => are files.
   Example:
   - The 1st HDD is named /dev/sda (SCSI) hoặc /dev/hda (IDE) in Linux/Unix
   - 2nd HDD is named /dev/sdb (SCSI) hoặc /dev/hdb (IDE)
7) Network socket: this has already been addressed in Computer Network course. Socket allows data to be read from/written to => is a file.
   Example:
   ```
   s = socket(domain, type, protocol);
   write(s, buf, strlen(buf));
   ```

Disk structure and file system layout



Disk structure

Partition Table: keeps information about starting and end blocks of every partition. Among the partitions, one must be active and used for booting the system.

Master Boot Record: when the computer is turned on, BIOS reads and execute the code in MBR. The MBR program will locate the active partition and load the Boot Block into the memory. The program in Boot Block will then load the OS in that partition to complete the system initialization.

File system organization

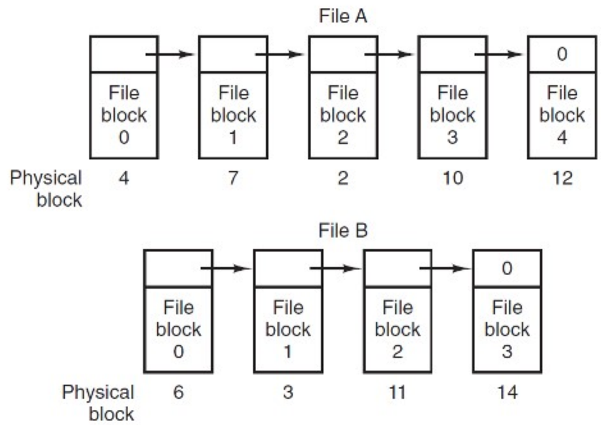
A partition is organized as follows

- Boot block: keeps the boot program to load OS into the memory
- Superblock: keeps information about other blocks, such as inode tables, root directory, data blocks etc.
- Free space management: manages the list of free blocks (used for file allocation)
- I-nodes: Index node table is used to manage blocks allocated to files
- Root dir: block where the root directory is located
- Files and Directories: data blocks allocated to sub directories and files

Maintain file content

Given the 1st disk block of a file, the question is how to link the remaining blocks of the file.

Linked list of disk blocks



A disk block is divided into 2 parts

- Pointer to the next block, and
- File content

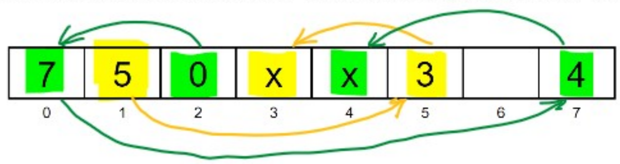
Remarks:

- size of the remaining part of a block is not a power of 2 any more => not convenient to locate a file record
- In order to jump to a record in the middle, one has to read all previous blocks sequentially

Linked list using file allocation table

Similar to the block linked list, but we separate the pointer linkage into an index table (Allocation table A). If block i keeps part of file content, then the next disk block is pointed by $A[i]$ in the allocation table. If i is the last block, then $A[i] = NULL$.

Example: The 1st block of file a.txt is 1, file b.txt has block 2 as the 1st block

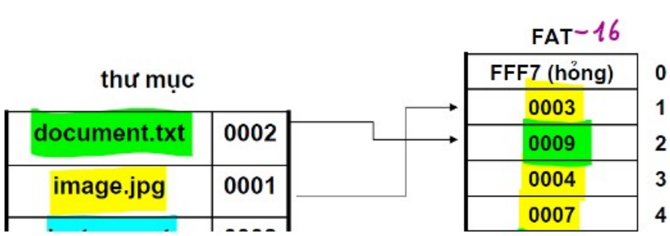


Case study: File Allocation Table in DOS/Windows:

Disk/partition layout

Boot block	FAT	FAT copy	Root dir	Data blocks ...

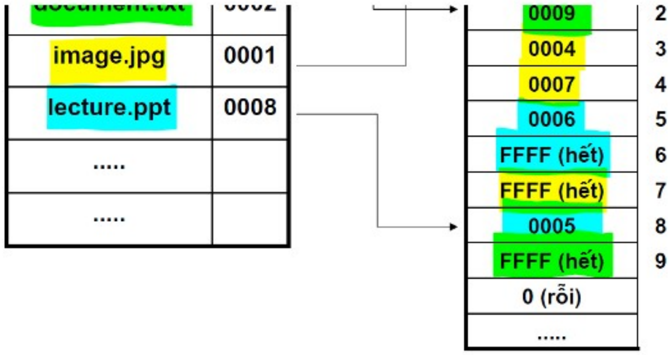
File allocation table



2^{16} khối $\times$ 16 KB

$= 2^{16} \times 2^{4}$ KB

$= 2^{20}$ KB $=$ 2GB



$= 2^{..} \times 2^{.}$ KB

$= 2^{20}$ KB = 2GB

$2^{32} \times 4B = 2^{39}$ B

$= 2^{29}$ KB

$= 2^{19}$ MB

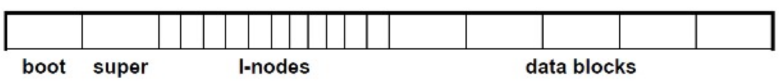
$= 2^{4}$ GB

Remarks on FAT:

- For better performance, FAT has to be kept in RAM => occupies a large amount of memory and not suitable for large file systems
- FAT does not manage user access permission

Index table with i-node (i-node table)

Each file is attached with an i-node. An i-node record has pointers to disk blocks containing file content.

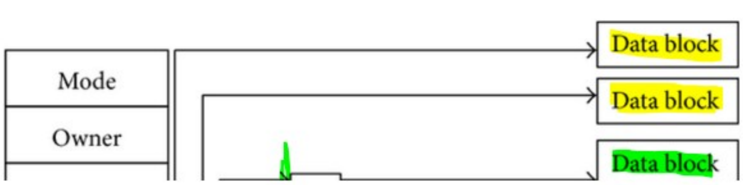


i-node structure

Mode
Link count
UID
GID
File size
Time
12 direct ptrs
single indirect
double indirect
triple indirect

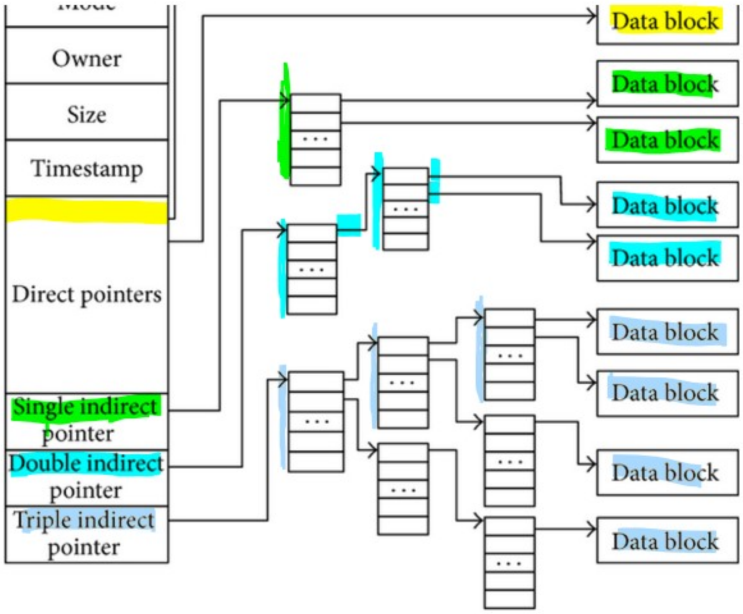
Mode: access right -rwx rwx rwx
Linkcount: count the number of files sharing the same i-node
UID/GID: owner/group ID
Size, time: size and time of file
Direct pointers: point directly to data blocks
Indirect pointers: point to an index table

i-node structure and multiple-level index table



4KB/khối

$12 \times 4 = \underline{48KB}$



4KB/khối

12x4 = 48KB

Direct pointers

- Directly point to data blocks of the file
- Suitable for small-size files
- Example: i-node with 12 direct pointers, each data block is 4KB, then the maximal size of a small file is $4\times 12 = 48\ KB$

Single Indirect Pointer, used for larger files

- Points to an index table of direct pointers to data blocks
- Example: 4KB/block, index table pointers are 32-bit, thus each table consists of 1024 pointers. The maximal size of a file of this kind is $4\times(12 + 1024) = 4144\ KB$

Double Indirect Pointers, used for even larger files

- Points to an index table of single indirect pointers
- Example: 4KB/block, index table pointers are 32-bit. The maximal size of a file of this kind is $4\times(12 + 1024 + 1024^2) = 4\ GB$

Triple Indirect Pointer, used for very large files

- Points to an index table of double indirect pointers
- Example: 4KB/block, index table pointers are 32-bit. The maximal size of a file of this kind is $4\times(12 + 1024 + 1024^2 + 1024^3) = 4\ TB$

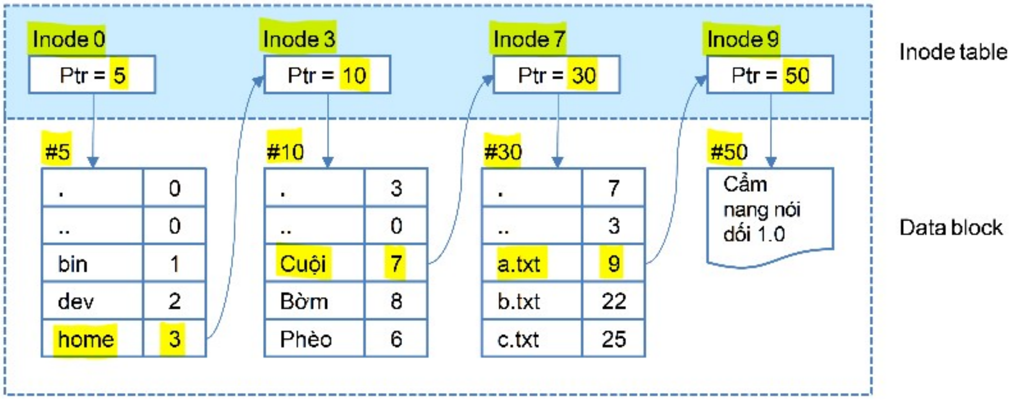
Getting access to the 1st block of a file

A directory is a record file of following structure

Tên tệp	Số i-node

Steps to get access to the 1st data block is illustrated in the following example:

Given a file name /home/Cuội/a.txt



1) The 1st block of root directory '/' is pointed by i-node no. 0, which is #5
2) We read data block #5, find a record with the name 'home' and get the file's i-node is 3, i-node 3 points to data block #10
3) Read data block #10, find a file named 'Cuội', and get i-node 7, i-node 7 points to data block #30
4) Read data block #30, find a file named 'a.txt', and get i-inode 9, i-node 9 points to data block #50
5) Data block #50 is the 1st block of file /home/Cuội/a.txt

Linked files

Hard link

Creating a hard link

$ whoami

    ntb

$ ln tepgoc teplienket

    -rw-rw-rw- *2* nva sinhvien 15 Mar 28 16:31 tepgoc
    -rw-rw-rw- *2* nva sinhvien 15 Mar 28 16:31 teplienket

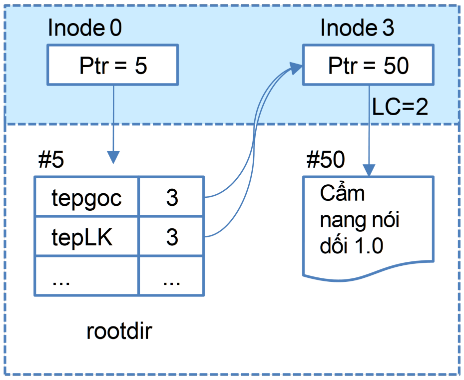
$ rm tepgoc

    -rw-rw-r-- *1* nva sinhvien 15 Mar 28 16:31 teplienket

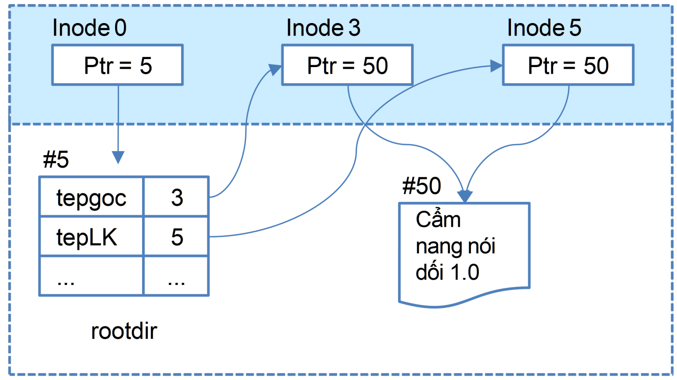
$ cat teplienket

    … content of the original file…

Two implementation approaches



PA1: 2 tệp dùng chung 1 inode



PA1: 2 tệp dùng 2 inode riêng, trỏ cùng tới 1 khối đĩa

Which approach is more feasible?
Approach 1, as we only need to check Link Count to determine whether to release the file's data blocks

Symbolic link

Creating a symbolic link

$ whoami

    ntb

$ ln -s tepgoc.txt teplienket.txt

    -rw-rw-rw-   1 nva   sinhvien   15 Mar 28 16:31 tepgoc
    lrwxrwxrwx 1 ntb   sinhvien   15 Mar 28 16:31 teplienket=>tepgoc

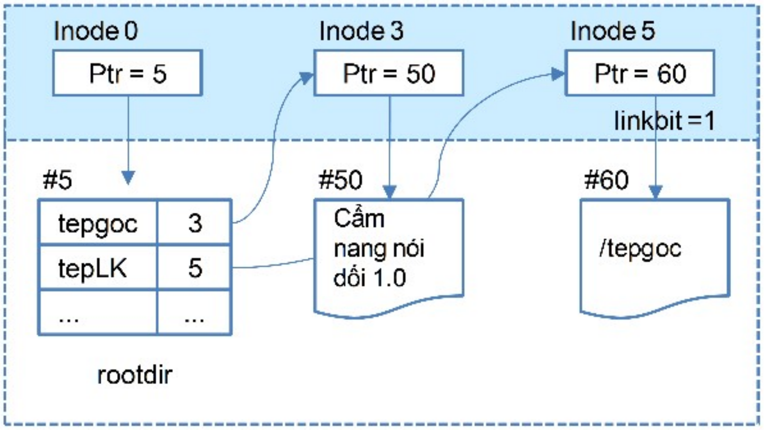
$ rm tepgoc

    lrwxrwxrwx 1 ntb   sinhvien   15 Mar 28 16:31 teplienket=>tepgoc

$ cat teplienket

    Lỗi: find not found

Symbolic link implementation



Tệp liên kết có nội dung là đường dẫn tới tệp gốc

The original and linked files have 2 different i-nodes, therefore they have different file attributes.

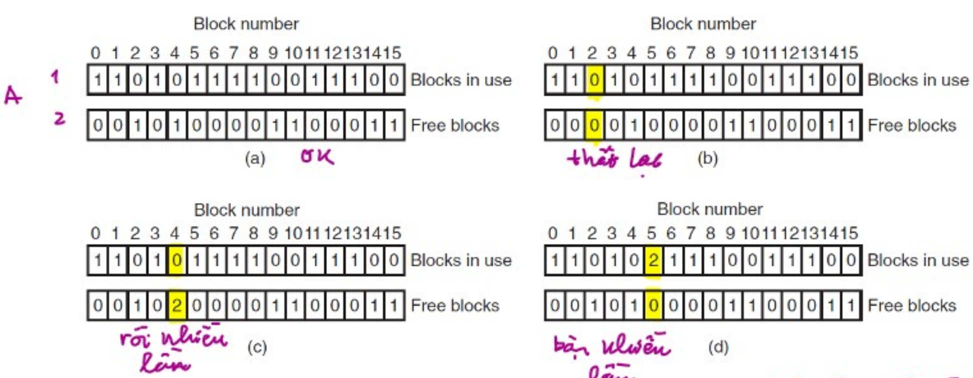
Symbolic link has a special attribute bit named bit l turned on, and the file content is the path to the original file.

Error detection for the file system

LC = ~~1~~ 0 free LC = ~~2~~ 1
tệp = ~~2~~ 1 ~~tệp~~ 1

As the file system is built with many linkages starting from the file record and i-node to data blocks, there may exist inconsistencies. Possible errors are:

1) The number of hardlinks in an i-node does not match with the number of files using that i-node
2) Data block state: (a) Orphan; (b) duplicated free blocks; (c) cross links to the same data block; (d) inconsitent of block state



Lỗi: $A[1,i] + A[2,i] \neq 1$

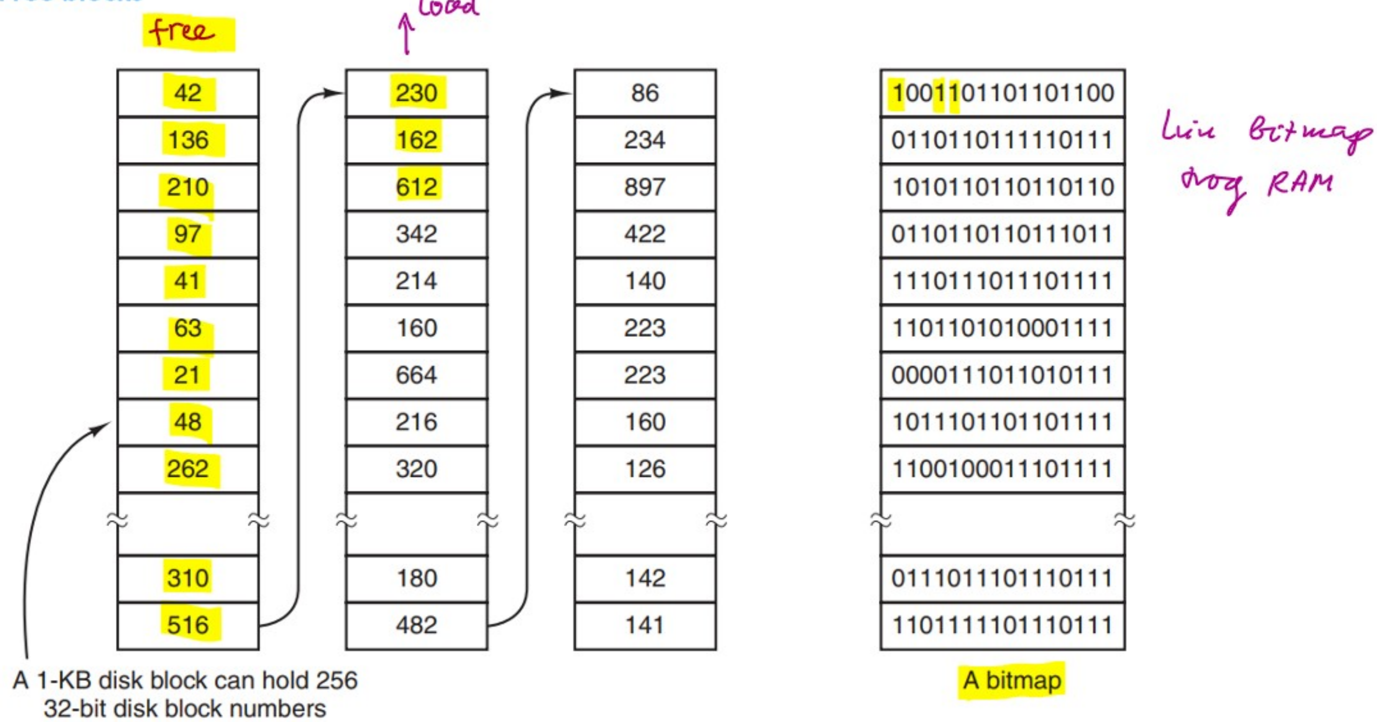
1) Inconsistency on hardlink count:
   Solution: Browse through all files, count the number of files using the same i-node, then update back to the linkcount field in the i-node

2) Data block state
 - Orphan blocks: add to the list of free blocks
 - Duplication of free blocks: remove the redundant free blocks out of the list
 - Crosslink of data blocks: allocate additional blocks, copy the content to the new blocks, then let each file points to a different block
 - Inconsistent of block state (occupied and free at the same time): remove out

of the free list

Managing free blocks, bad blocks, disk quota

Free blocks



Using bit map
The state of each block is represented as a bit in the corresponding position in the bit map.
Pros: size of the bit map is relatively small
Cons: the bit map has to be fully loaded into RAM for the performance of allocation

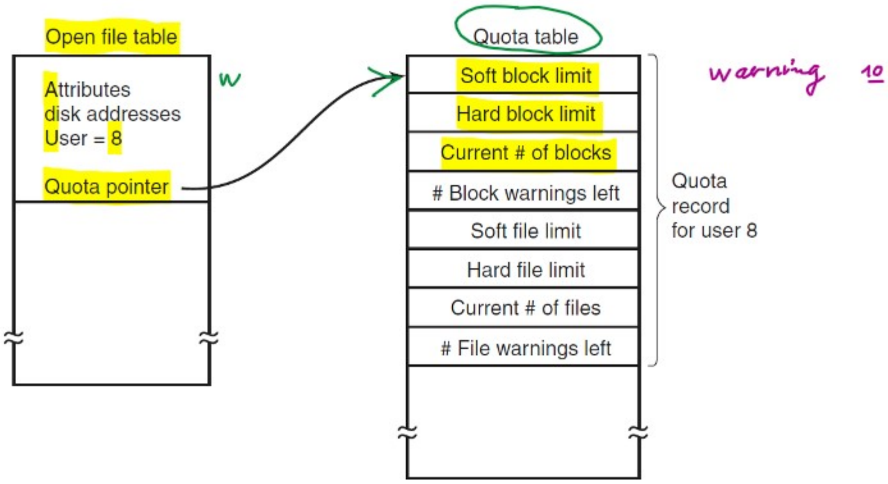
Using linked list of disk blocks containing list of freeblocks
Chain some free blocks together. Each block contains a list of free blocks.
Cons: size on disk is larger than the bit map
Pros: only 1 block of freelist needs to be loaded to RAM. Blocks in the chain themselves are also freeblocks, thus does not actually require dedicated disk space.

Bad blocks

Bad blocks if detected at low-level formatting will be substituted by additional blocks available at the end of each disk track. No need for the intervention of OS.

Bad blocks detected after the disk format will be gathered by OS and kept in a hidden file, not accessible to users, thus will not be allocated to user's files.

Disk quota



A user profile is attached with a disk quota record. This record is loaded to memory whenever a file related to that user is open.
When a file is open, a quota pointer is added to Open File Table, pointing to the user's quota record. Any increase/decrease of file size results to the increase/decrease of data blocks recorded in the quota record.
Soft limit is the limit that user can exceed before certain number of warnings runs out. Hard limit is the limit that user can not exceed.

Management of file access permission

Representing access permission using a sparse matrixbiểu diễn quyền

Object → tệp

Domain ↓ ng dùng	File1	File2	File3	File4	File5	File6	Printer1	Plotter2	
1	Read	Read Write							nva
2			Read	Read Write Execute	Read Write		Write		ntb
3						Read Write Execute	Write	Write	nvc

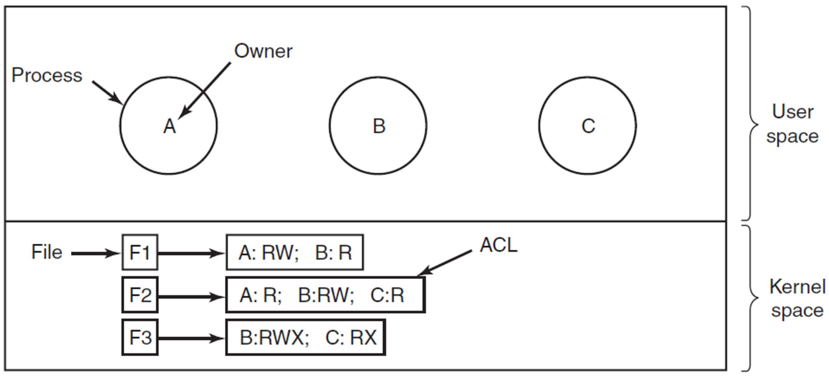
Object: represents computer resources
Domain: represents users (the scope of users' access)

Cons: sparse matrices costs lots of storage space

Compress the sparse matrix by column - Access Control List (ACL)

For each object, we attach a list of domain and corresponding access permissions.
Domain can be a user or a group of users. role

Example:



ACL list of file F1: user A has RW permission, user B has R permission.

When a process of user A tries to access file F1, OS will search among ACL list for its appropriate permission.

Remarks:

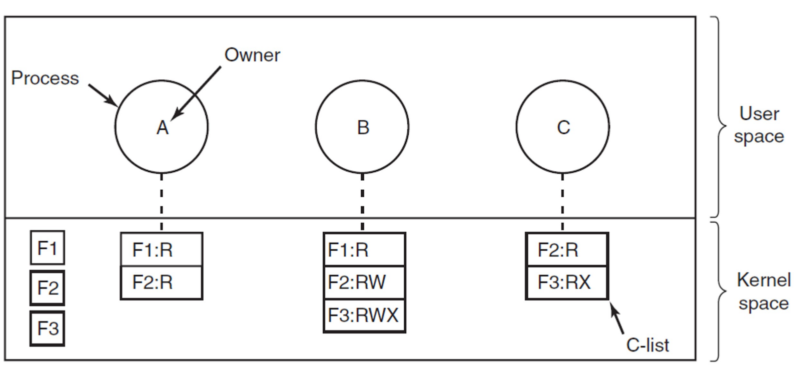
- The search operation takes certain time as ACL list needs to be loaded to memory, and if ACL gives a group certain permission it still has to check whether the user belongs to that group
- ACL however can define groups and roles, which allow the administrator easily revokes access permission to a selected group, for example we can revoke permission of everybody to file F1, except user nva of group $student$

  File F1 - $nva, student: RW$; $*, *: (none)$
- This is the method used in Windows, Unix ACL

Compress the sparse matrix by row - Capability-List (C-list)

For each domain (i.e user), we attach a list of files and their access permissions (i.e capability).

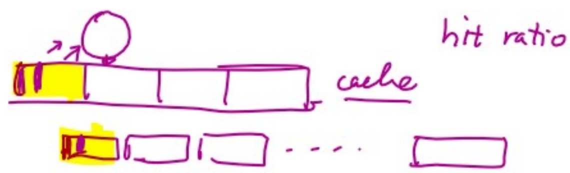
Example:



Process of user A is attached with a list of files it has access permission to.

Remarks:
- C-List allows direct access to the resources by following capability pointers in C-list without the need to search as in ACL
- Each capability is a pointer to the resource and therefore may point to another C-List. This allow sub-domain definition.
- C-List does not allows one easily revoke permission of a domain by certain selected permission as ACL, as it has to search all lists attached to all domains in the system
- Removing a user but not having C-List updated timely or vice versa will also cause problems in the system
- This is the method used in the old network OS Novell Netware long ago

Caching



Data blocks on disks with frequent access should be kept in RAM for better performance, for instance the i-node table.

Cache is a limited-size buffer, thus should only keep what are used the most frequently.

Question: can we apply swapping algorithms, such as LRU, for cache as we did with virtual memory?
- Yes, but we have to pay attention to the consistency problem of data on important disk blocks which are frequently used by the file system, such as the i-node table
    - Need to regularly update cache content of these blocks to disk
    - Apply write-through cache: i.e caching for read operation, but not for write operation