Directory implementation:

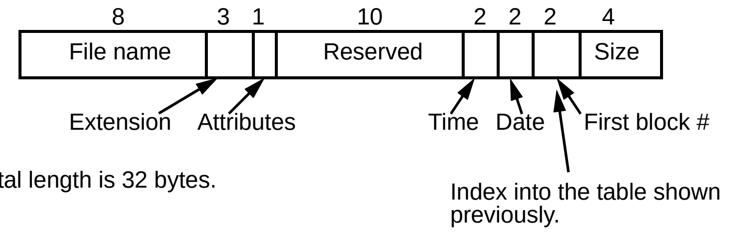
Main function of directory system is to map the ASCII file name onto the location of data on the disk.

Directory attributes can be stored:

In the directory entry (MS-DOS).

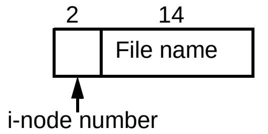
In the i-node (UNIX).

MS-DOS Directory Entry:



Total length is 32 bytes.

UNIX Directory Entry:



All other info, e.g. type, size, times, etc. are kept in the i-node.

Disk Space Management:

Two possibilities of storing an n-byte file:

- Contiguously as n consecutive bytes (analogous to pure segmentation).
- File split up into (possibly non-contiguous) fixed sized blocks (analogous to paging).

Option #1 has same problem that pure segmentation has - if the file grows, it will probably have to be moved.

Almost all file systems use option #2.

Block Size:

Large size: wastes space (medium file size in UNIX found to be 1K).

Small size: makes access slow - many seeks and rotational delays, e.g.,

Given disk has 32,768 bytes/track.

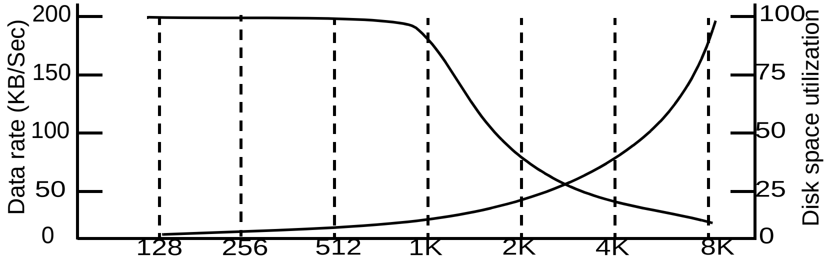
Rotational time of 16.67 msec.

Average seek time is 30 msec.

To read a block of k bytes is (seek+rotational delay+transfer time):

30 + 8.3 + (k/32768)*16.67 msec.

Assume all files are 1K : Time and space efficiency in conflict



Keeping track of free blocks:

One possibility is to keep a table of free disk blocks numbers in a set of free disk blocks.

With 1K blocks and 32 bit disk block numbers: each block can hold 255 free disk block numbers + a pointer to the next block.

A second possibility is to use a bit map.

One bit per block; 1 - block is free, 0 - block is allocated.

Requires less space than first scheme (1/32 as much).

File System Performance:

Memory access: 10s of nanoseconds.
Disk access: 10s of milliseconds.
Difference: Up to 100,000 times slower !!!

Solution: **Buffer cache** - disk blocks are kept in memory.

If buffer is full and a miss occurs, a cache block must be selected for replacement.

Page replacement algorithms are applicable here.
Since cache references are infrequent, LRU is feasible.

LRU unmodified is undesirable because:

Critical blocks, such as an i-nodes, need to be updated on disk immediately in case of a system crash.
Double indirect blocks are rarely referenced two times within a short interval.

Blocks can be catagorized into:

- i-node blocks
- indirect blocks
- directory blocks
- full data blocks
- partly full data blocks

LRU modified:

- Indirect blocks on the front of list, to be replaced quickly.
- Critical blocks (all but data blocks) immediately written to disk.

Keeping even data blocks too long in the cache can make users angry.

User works all day on a document using an editor.
User saves the document periodically to guard against crashes.
However, update to disk is not performed and system crash causes user to loose data !

UNIX runs update to SYNC the cache with the disk every 30 seconds.
MS-DOS use **write-through** caches.

Other ways of increasing the performance of a file system:

- Choose consecutive blocks or blocks in the same cylinder when creating a file.
- Allocate disk storage in units of 2 or more blocks - reduces seeks by a factor of two.
- Distribute i-nodes across the disk instead of keeping them all on track 0.

Security

Major concern of file system: protecting information against unauthorized usage.

Passive intruders: Want to read files they are not authorized to read.
Active intruders: Want to make unauthorized changes to data.

The effort that goes into making a system secure should consider the kind of intruder one is protecting against:

- Casual prying by nontechnical users, e.g., reading your girlfriend/boyfriend's email.
- Hackers, e.g., challenged to break the security of a computer system.
- Commercial or military espionage, e.g., stealing trade secrets.

Famous security flaws:

UNIX:

- lpr utility has option of removing a file after printing it. Early version allowed user to print and then delete the password file.
- Link core to password file. Intruder caused core dump of a SETUID program which overwrote password file.
- *mkdir foo* - mkdir is a SETUID program owned by root.

    It first created the i-node for the directory foo (MKNOD) and THEN changed the owner of *foo* (CHOWN) from its effective uid (root) to its read uid (the user's uid).

    When system was slow, user could sometimes remove the directory i-node and make a link to the password file under the name *foo* after MKNOD but before CHOWN.

    When *mkdir* did the CHOWN, it made the user the owner of the password file.

Other discussed in book for TENEX and OS/360.

Trojan horse programs:

Seemingly innocent programs that are widely distributed.

Actually do more than what they say they do, such as email data to remote user.

Logic bomb:

Piece of code inserted into OS by an employee.

Employee feeds it a password daily to get it subdued.

If employee fired, it goes off - boom !

'Going off' may mean that it encrypts essential files.

Company may choose to hire ex-employee as a consultant for an astronomical sum of money to fix the problem.

Worm:

Complete program in itself (Most famous: Robert Tapper Morris, Nov 2, 1988).

Virus:

Piggybacks on existing programs - a program fragment.

- Attaches to the end of a binary.
- First instruction is replaced with a jump to virus code.
- Virus checks other binaries on hard drive and infects.
- Executes replaced instruction.
- Jumps to the old start-up address + 1.

Tiger Teams: Experts whose job it is to break into existing systems.

Consideration when designing/testing/using a system:

- Make sure that allocation requests for memory, disks or tapes don't allow new owner to read old owner's information.
- Protect and test system calls carefully, especially the parameters.
- Beware terminal login programs.
- Try variations of X for manual page warning "do not do X".

General principles for designing secure systems for larger systems:

- System design should be public domain.
- Default access should be no access.
- Check for current authority, e.g., permission to read/write open files should be checked periodically.
- Give each process the least privilege possible.
- Protection should be simple, uniform and built into the lowest layers of the system.
- Scheme should be psychologically acceptable.

User Authentication:

Problem of identifying users when they log in.

Passwords most popular method, e.g. UNIX,

- Login program asks user for password.
- It is immediately encrypted.
- Password file checked for match to login + encrypted password.

With 95 printable characters and a 7 character password, $95^7 = 7X10^{13}$ possible combinations.

At 1000 encryptions per second, it would take 2000 years to build the list to check the password file.

However, most break-ins occur by guessing user name and password combinations because most people use very simple passwords.

Study by Morris and Thompson (1979): Compiled list of likely passwords:

- First and last names
- Street names
- City names
- Words from a dictionary
- Words spelled backwards
- License plate numbers
- Short strings of random characters

They encrypted these using a known password encryption algorithm.
Checked password file(s) for matches and found over 86% of the passwords showed up in their list !

Salting the password file:

- Store an n-bit random number with each password entry unencrypted.
- Change the number when user changes the password.
- Encryption is done using concatenation of random number + the user password.

Compiling a list of common passwords does not work anymore.

One-time password: Users given a book of passwords. Each login uses the next password in the book.

Encrypted questions and answers: one chosen at random at login.

Challenge-response:

- User picks an algorithm, e.g., x^2 at account creation time.
- Login prints an arg, e.g., 7 and user types in the answer, e.g. 49.

Physical ID:

Check if the user has some item, e.g., plastic card + magnetic strip.

- ATM uses this method along with a password (pin #).

Finger print or voice print.

Signature analysis: Save the pen motions for better security.

Finger length analysis.
Instant urinalysis.
Thumbtack (for blood sample) + spectrograph.

Which ones are socially acceptable ?

Countermeasures:

Users only allowed to log in from specific terminals/days/times.

Dial-ups: User calls and logs in - computer breaks connection and calls user back.

System should take at least 10 seconds to authenticate - 3 fails should disable account and notify security.

All login should be recorded - each login should report previous login.

Baited traps: provide an easy login and password to break, e.g. guest and guest.

Protection Mechanisms:

Detailed technical ways used in OSs to protect files and other things.

Policy and mechanism are often separated:

Policy: Whose data are to be protected from whom.

Mechanism: How the system enforces the policy.

Focus will be on mechanism.

Reference monitor: A program used to provide protection.

All accesses to protected resources are checked first by the reference monitor - which looks at its policy tables and makes a decision.

There are many types of “objects” that need to be protected on a computer system:

CPU, memory segments, disk drives, printers, etc.

Processes, files, data bases, semaphores, etc.

Each object has a unique name and a finite set of operations, e.g. Wait and Signal for semaphores.

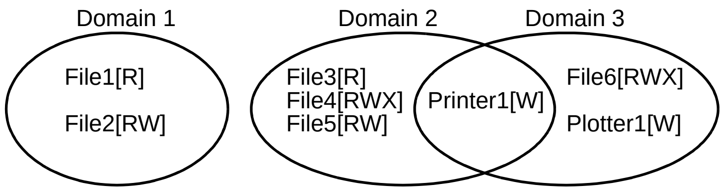
Needed:

- A mechanism to prohibit processes from accessing objects that they are not authorized to access.
- A mechanism to restrict processes to a subset of legal operations.

Domain:

A domain is a set of (object, rights) pairs.

Rights indicate permission to perform one of the operations.



Rights consist of Read, Write and Execute.

Printer1 in two domains - can have different rights in each domain.

At any instant, each process runs in some protection domain.

Domain switching is possible but controlled.

In UNIX, domain of a process is defined by its **uid** and **gid**.

Given any combination of (uid, gid):

It is possible to make a complete list of all objects, e.g. files, I/O device access.

And the rights (RWX) possible.

Each process in UNIX has two halves: a user part and a kernel part.

When a process makes a system call, it switches domains.

When a process does an EXEC on a file with SETUID or SETGID bit on, it acquires a new effective uid or gid - domain switch.

How does the kernel keep track of which object belongs to which domain ?

Conceptually, this can be envisioned as a large matrix:

Domains	File1	File2	File3	File4	Printer	Plotter	Domains 1	Domains 2	Domains 3
1	Read	Read Write						Enter	
2			Read	Read Write Execute		Write			
3					Write	Write			

Domain switches are included in matrix model by considering them an object.

Processes in domain 1 can switch to domain 2, but they can not go back.
Simulates a SETUID program in UNIX.

Access Control Lists:

Matrix implementation is rarely done because it is large and sparse.

Two practical methods:

- Store non-empty rows of matrix.
- Store non-empty columns of matrix.

Access Control List (**ACL**) - Column storage:

Each object possesses an ordered list of all domains and rights.

In UNIX, this can be implemented by storing this list in a block and including the block number in the i-node list.

Full-blown ACL example:

Users:
Jim: Professor
Kamika: Student
Terry: System
Sarah: Student

File0:(Jim, *, RWX)
File1:(Jim, Professor, RWX)
File2:(Jim, *, RW-), (Terry, System, RW-)
File3:(*, Student, R--)
File4:(Kamika, *, ---), (*, Student, R--)

File0: RWX by uid = Jim and gid = anything

File4: Kamika in any group has no access, all other students have R.

UNIX provides three bits, rwx, per file for owner, group and other.

This is an ACL scheme but compressed to 9 bits. Simpler and cheaper to implement.

Capabilities - Row storage:

Associated with each process is a list of objects + operations.

List is called a capabilities list (C-list) and individual items are capabilities.

Each Capability has a type, rights and an object field:

#	Type	Rights	Object
0	File	R--	Ptr to File3
1	File	RWX	Ptr to File4
2	File	RW-	Ptr to File5
3	Plotter	--X	Ptr to Plotter

Capability list are themselves object and may be pointed to by others.

Capability lists MUST be protected from tampering.

Three possible methods:

- Tagged architecture:
  - Each memory word has an extra bit.
  - Indicates if word is a capability.
  - Arithmetic and comparison operations modified to consider these bits.
  - Only kernel mode allowed to access/change.

- Maintain C-list in OS and have processes reference capabilities by their slot number.

- Keep C-list in user space but encrypt each capability with a secret key unknown to the user.

- In capability systems, revoking access is difficult.
  - OS must find all capabilities for an object.
  - C-lists are stored all over the disk.