

File Systems



Today

- Files and access methods
- Directory structures
- Sharing and protection
- File & directory implementation

Next

- File system management & examples

Files and file systems

Most computer applications need to:

- Store large amounts of data (larger than their address space)
 - that must survive process termination and
 - can be access concurrently by multiple processes
- Usual answer: Files – from user’s perspective, the smallest allotment of logical secondary storage

File system – part of the OS dealing with files

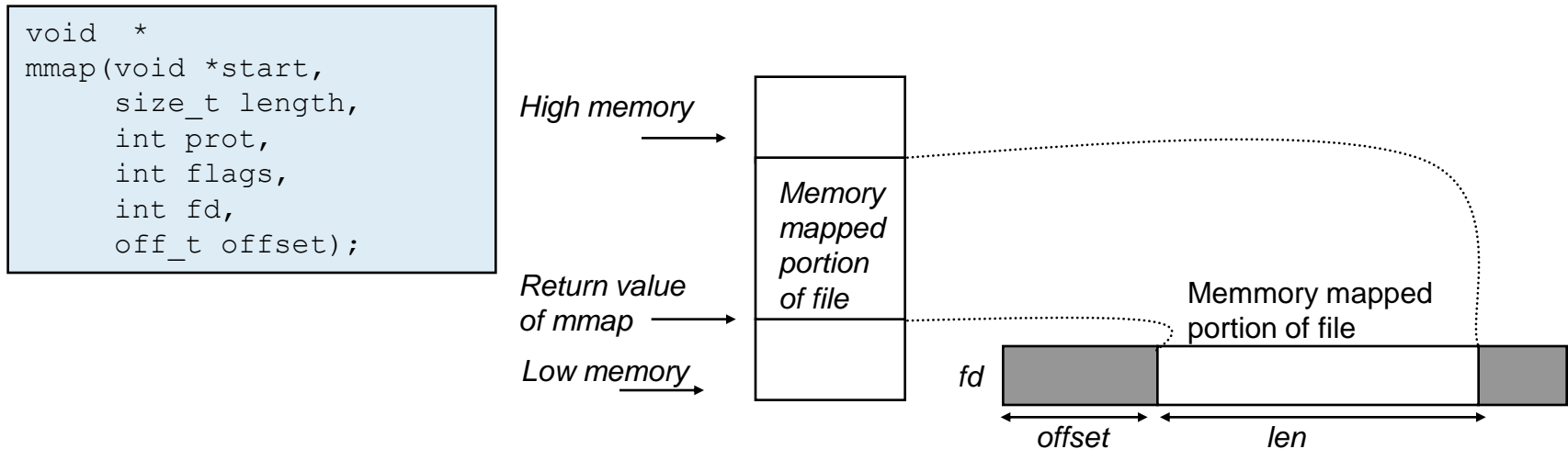
- Supports the file abstraction of storage
- Naming – how do users select files?
- Protection – users are not all equal
- Reliability – information must be safe for long periods of time
- Storage mgmt. – efficient use of storage and fast access to files

File – attributes, types and operations

- Files are collection of data with some attributes
 - Names & type, if supported, location, owner, last read/write times, ...
- Different OSs support different file types
 - Regular, binary, directories, ...
 - Character special (model terminals [/dev/tty], printers, etc) and block special files (model disks [/dev/hd1])
 - Extensions as hints & the use of magic numbers
 - Some typical file extensions
 - Pros and cons of strongly typed files
- Basic operations
 - Create, delete, write, read, file seek, truncate
 - Other operations can be built on this basic set (e.g. cp)

Memory-mapped files

- Accessing files & accessing memory
- Key idea – file is backing store of mapped memory
- Some problems
 - Not all files can be mapped
 - If file mapped is an output one, what's the size?
 - File mapped by one and open conventionally by another?!?
 - File size larger than virtual address space?!?



File structures & access methods

- Several file structures, three common ways
 - Byte sequence - Unix & Windows; user imposes meaning
 - Record sequence – think about 80-column punch cards
 - Tree – records have keys, tree is sorted by it
- Access methods
 - Sequential – tape model
 - Read/write next; simplest and most common
 - Random/direct access – disk model
 - Two approaches: read/write x, or position to x and read/write
 - Retain sequential access – read/write + update last position
 - Other access methods
 - On top of direct access, normally using indexing
 - Multi-level indexing for big files (e.g. IBM Indexed Sequential Access Method)

Directory structure

- To manage volume of info.: partitions & directories
- Directory: set of nodes with information about all files
 - Name, type, address, current & max. length, ...
- Operations on directories
 - Open/close directories, create/delete/rename files from a directory, readdir, link/unlink, traverse the file system
- Directory organizations - goals
 - Efficiency – locating a file quickly.
 - Naming – convenient to users.
 - Grouping – logical grouping of files by properties (e.g. all Java progs., all games, ...)

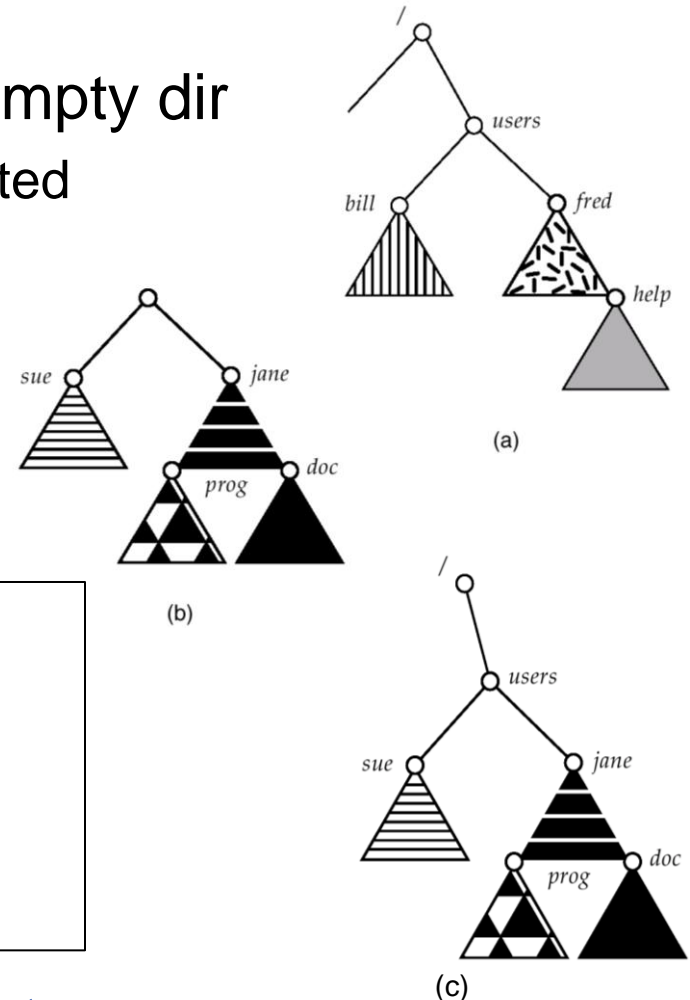
File system mounting

- A FS must be mounted to be available
 - What do you do if you have more than one disk? Put a self contained FS on each (C:...) or...
- Typically, a mount point is an empty dir
 - Existing file system (a) & unmounted partition (b)
 - After it was mounted (c)

```
# mount /dev/sda1 /users
```

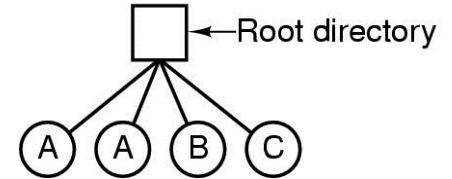
- `fstab` file in Unix

```
(10:41am) ~ % more /etc/fstab
# This file is edited by fstab-sync - see 'man fstab-sync' for details
LABEL=/          /                ext3 defaults 1 1
none             /dev/pts         devpts gid=5,mode=620 0 0
/dev/sdb1        /export          ext3 defaults 1 2
none             /proc            proc defaults 0 0
none             /dev/shm         tmpfs defaults 0 0
LABEL=/usr       /usr             ext3 defaults 1 2
...
```

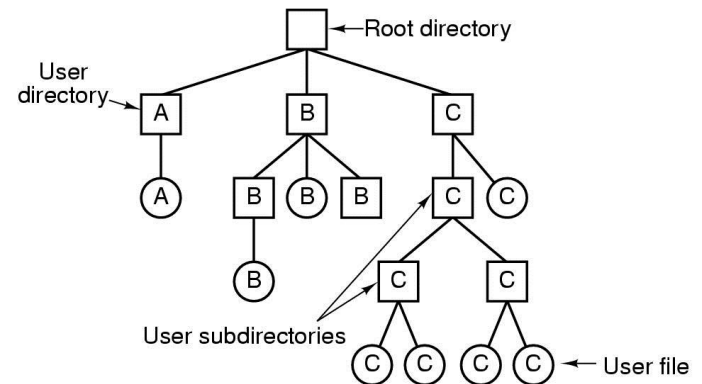


Single and hierarchical directory systems

- A single level directory system
 - Early PCs & supercomp. (CDC 6600), embedded systems?
 - Fast file searches, but name clashing



- Hierarchical
 - Avoid name clashing for users (MULTICS)
 - Powerful structuring tool for organization (decentralization)



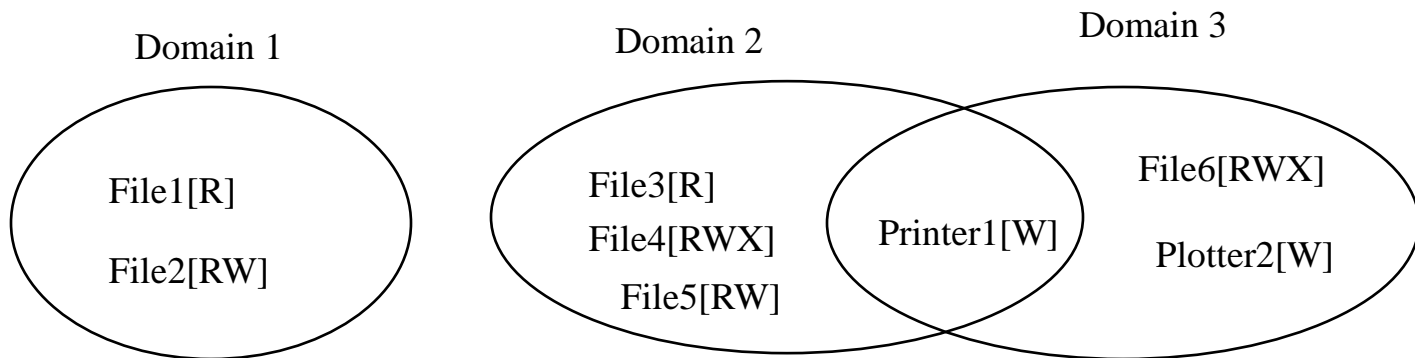
- Path names
 - Now you need a way to specify file names; two approaches: absolute and relative
 - “.” & “..”

Protection ...

- FS must implement some kind of protection system
 - to control who can access a file (user)
 - to control how they can access it (e.g., read, write, or exec)
- More generally:
 - generalize files to objects (the “what”)
 - generalize users to principals (the “who”, user or program)
 - generalize read/write to actions (the “how”, or operations)
- A protection system dictates whether a given action performed by a given principal on a given object should be allowed
 - e.g., you can read or write your files, but others cannot
 - e.g., your can read /etc/motd but you cannot write to it

Protection ...

- Useful to discuss protection mechanisms: domains
 - A domain – a set of (*object, rights*) pairs
 - At every instant in time, process runs in some domain
 - In Unix, this is defined by (UID, GID); exec a process with SETUID or SETGID bit on is effectively switching domains



Protection domains

- Keeping track of domains
- Conceptually, a large protection matrix

		Object							
		File1	File2	File3	File4	File5	File6	Printer1	Plotter2
Domain	1	Read	Read Write						
	2			Read	Read Write Execute	Read Write		Write	
	3						Read Write Execute	Write	Write

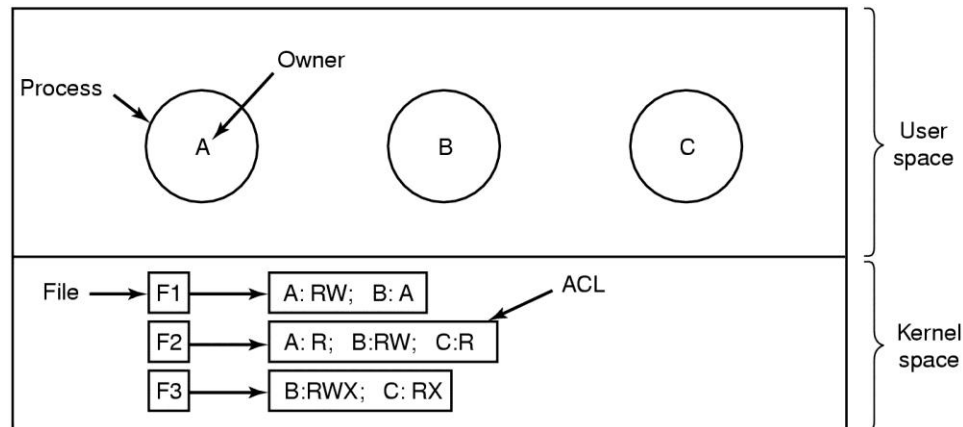
- A protection matrix with domains as objects
 - Now you can control domain switching

		Object										
		File1	File2	File3	File4	File5	File6	Printer1	Plotter2	Domain1	Domain2	Domain3
main	1	Read	Read Write								Enter	
	2			Read	Read Write Execute	Read Write		Write				
	3						Read Write Execute	Write	Write			

- A global table – too large & sparse ...

Implementing access matrices

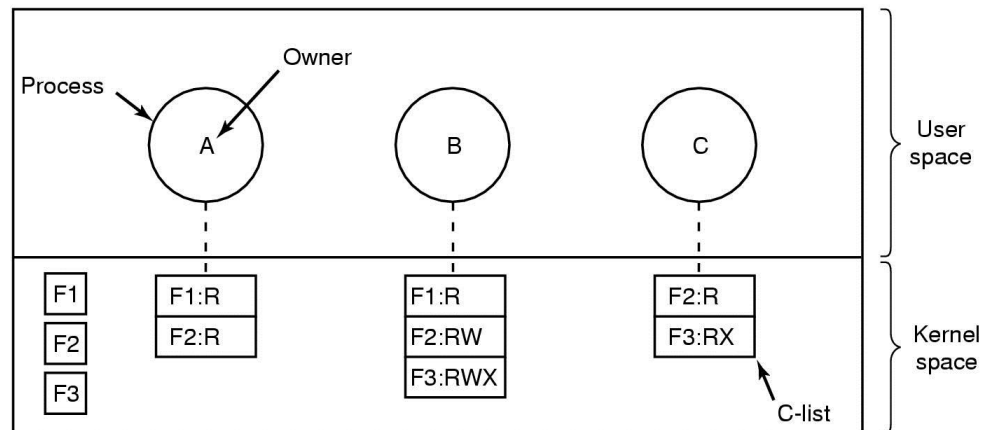
- Access control list
 - Associating w/ each object a list of domain that may access it (and how)
 - Users, groups and roles



Implementing access matrices

- Capabilities

- Slice the matrix by rows
- Associate w/ process a list of objects & rights
- Need to protect the C-list
 - Tagged architectures (IBM AS/400)
 - Keep it in the kernel (Hydra)
 - Manage them cryptographically (Amoeba)



- *Capabilities are faster to use but do not support selective revocation*

Protection

- Unix: short version access lists & groups
 - Objects – individual files
 - Principals – owners/group/world (the domain of a process is given by its UID and GID)
 - Actions: read, write, execute (3 bits per access mode)
 - Mask provides a default (creation with 777, mask 022 → 755)
- More general access lists - setfacl & getfacl

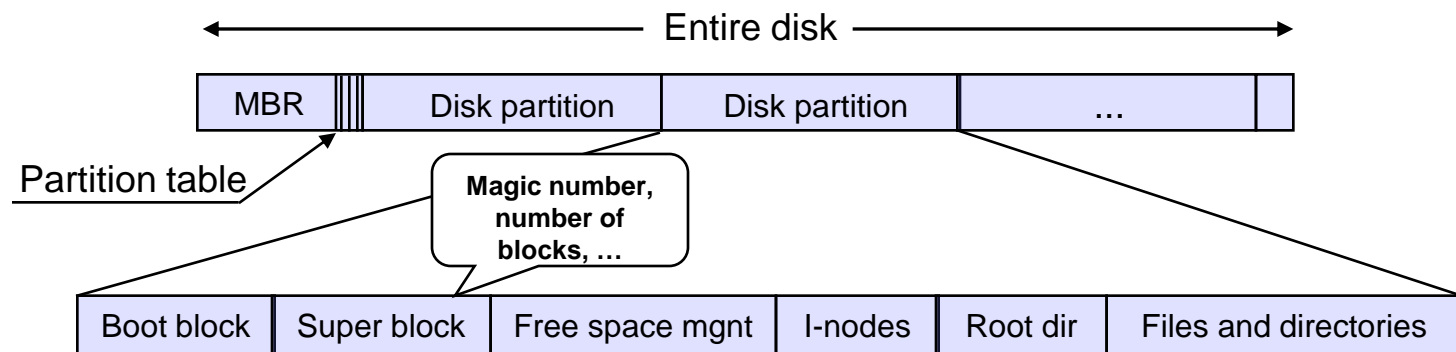
```
% getfacl -a exam.tex
# file: exam.tex
# owner: fabianb
# group: other
user::rw-
group::r--      #effective:r--
mask:r--
other:r--
```

Intersection of
specified permissions
and mask field.

```
% setfacl -r -m u:sbirrer:rw- exam.tex
% getfacl -a exam.tex
# file: exam.tex
# owner: fabianb
# group: other
user::rw-
user:sbirrer:rw-      #effective:rw-
group::r--            #effective:r--
mask:rw-
other:r--
```

File system layout

- Disk divided into 1+ partitions – one FS per partition
- Sector 0 of disk – MBR (Master Boot Record)
 - Used to boot the machine
- Followed by Partition Table (one marked as active)
 - (start, end) per partition; one of them active
- Booting: BIOS → MBR → Active partition's boot block → OS
- What else in a partition?

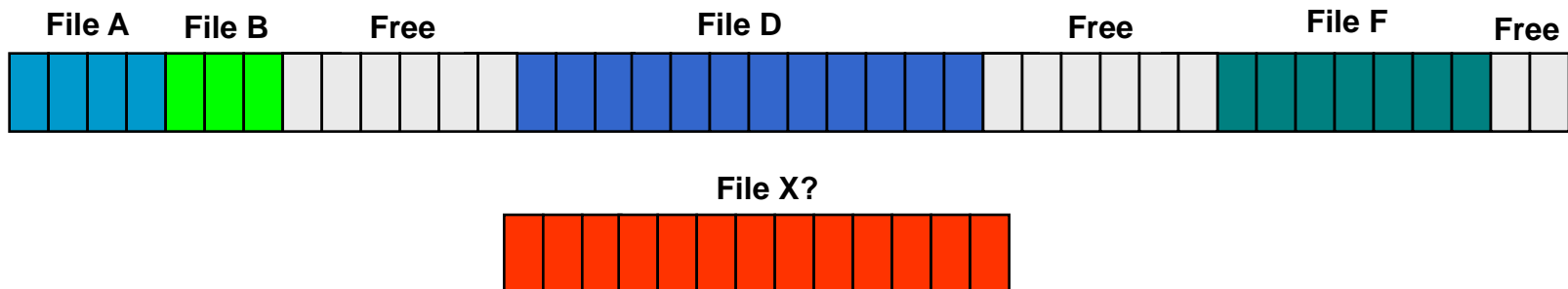


Implementing files

Keeping track of what blocks go with which file

- Contiguous allocation
 - Each file is a contiguous run of disk blocks
 - e.g. IBM VM/CMS
 - Pros:
 - Simple to implement
 - Excellent read performance
 - Cons:
 - Fragmentation

Where would it make sense?



Implementing files

- **Linked list**

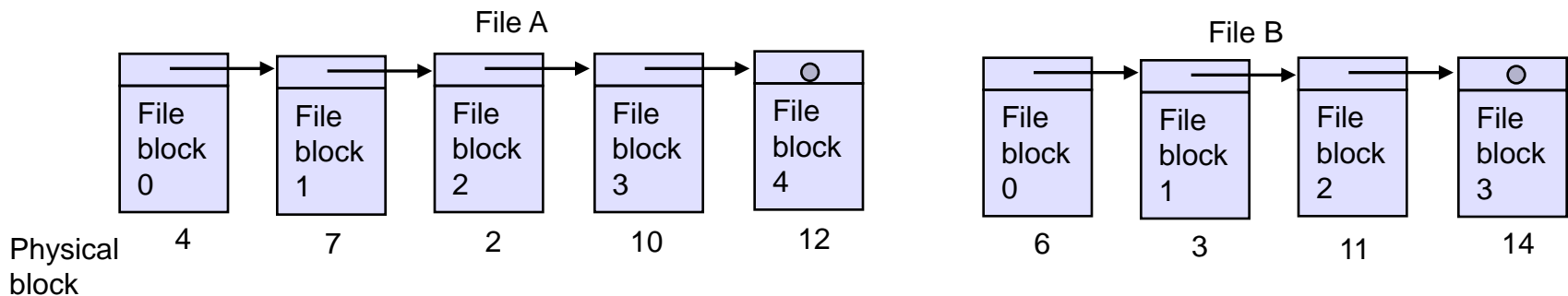
- Files as a linked list of blocks

- Pros:

- Every block gets used
- Simple directory entry per file (address of first block)

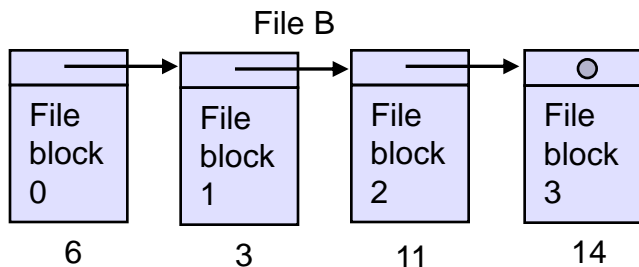
- Cons:

- Random access is a pain
- List info in block → block data size not a power of 2
- Reliability (file kept together by pointers scattered throughout the disk)



Implementing files

- Linked list with a table in memory
 - Files as a linked list of blocks
 - Pointers kept in FAT (File Allocation Table)
 - Pros:
 - Whole block free for data
 - Random access is easy
 - Cons:
 - Overhead on seeks or
 - Keep the entire table in memory
- 20GB disk & 1KB block size →
20 million entries in table →
4 bytes per entry ~ 80MB of memory



FAT	
0	
1	
2	10
3	11
4	7
5	
6	3
7	2
8	
9	
10	12
11	14
12	-1
13	
14	-1
15	

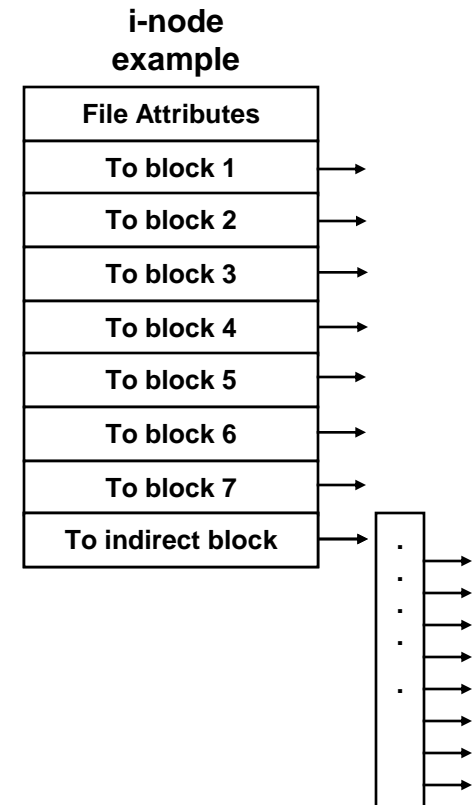
File A starts here

File B starts here

Implementing files

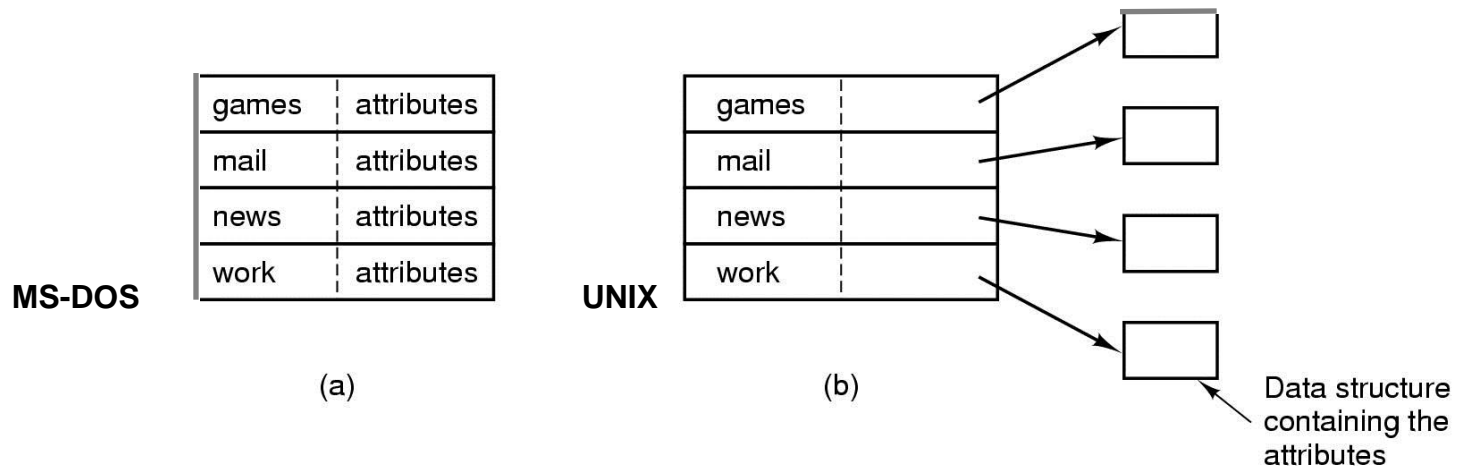
- I-nodes - index-nodes
 - Files as linked lists of blocks, all pointers in one location: i-node
 - Each file has its own i-node
 - Pros:
 - Support direct access
 - No external fragmentation
 - Only a file i-node needed in memory (proportional to # of open files instead of to disk size)
 - Cons:
 - Wasted space (how many entries?)
 - More entries – what if you need more than 7 blocks?

Save entry to point to address of block of addresses



Implementing directories

- Directory system function: map ASCII name onto what's needed to locate the data
- Related: where do we store files' attributes?
 - A simple directory: fixed size entries, attributes in entry (a)
 - With i-nodes, use the i-node for attributes as well (b)
- As a side note, you find a file based on the path name; this mixes what your data is with where it is – *what's wrong with this picture?*



Path name translation

- To open a file such as `“/a/b/c”` for editing
`fd = open (“/a/b/c”, O_RDWR);`
- What does it take?
 - Open directory `“/”` (well known)
 - Search directory, for `“a”`, get location of `“a”`
 - Open directory `“a”`, search directory, for `“b”`, get location of `“b”`
 - Open directory `“b”`, search for `“c”`, get location for `“c”`
 - Open `“c”`
- FS spend a lot of time at this
 - This is why we use `open` first
 - OS can then cache prefix lookups to enhance performance
 - `“/”`, `“/a”`, `“/a/b”`, ...

Shared files

- Links and directories implementation
 - Leave file's list of disk blocks out of directory entry (i-node)
 - Each entry in the directory points to the i-node
 - Use symbolic links
 - Link is a file w/ the path to shared file
 - Good for linking files from another machine
- Problem with first solution
 - Accounting
 - C creates file, B links to file, C removes it
 - B is the only user of a file owned by C!
- Problem with symbolic links
 - Performance (extra disk accesses)

Next Time

- Details on file system implementations and some examples ...