Before we start

What is RAM fragmentation?
Why does it happen?

What is the cure?

What is file-store defragmentation?
Why is it needed?

What (& why) is each of these:
- disk partition

- mounting e.g. a storage device

- hard & soft (symbolic) links in a file-store

MANCHESTER 1824
The University of Manchester

COMP25111: Operating Systems

Lecture 16: The File Manager

Oscar Palomar

School of Computer Science, University of Manchester

Autumn 2016

Overview & Learning Outcomes

1. Files & File Systems
2. Naming Service
3. Storage Service
   - Data Structures
   - Allocation
4. File Manager & Virtual Memory

What is a file?

Collection of related information on secondary storage:
e.g. data, programs (.java, .c, .h, .class, .o, binary, ...)

Structure: lines of text; or sequence of bytes; or ... ?

Types: should OS recognise/support?
– in the name: .com .exe etc. (MSDOS)
– “magic number” at the beginning of some files (Unix)

Attributes: (name?) size, last update, owner, ...
(try `ls -la`)

Operations: create, open, read, write, close, delete, ...

Access:
– Sequential: processed in order, from start to end
– Direct (Random): logical records, processed in any order

File system

File IDentifiers:
SFID – System – lifetime of file
UFID – User – lifetime of process
(Unix: “file descriptor”, Windows: “file handle”)

Requirements – system calls:

`open`: file-name → UFID
`read`: UFID & count → data
`write`: UFID & data →

Multiple OS Layers:

– naming service: e.g. `open`
– storage service: e.g. `read` & `write` (vector of bytes)
– disk driver: access disk sectors

File system organisation: Directories

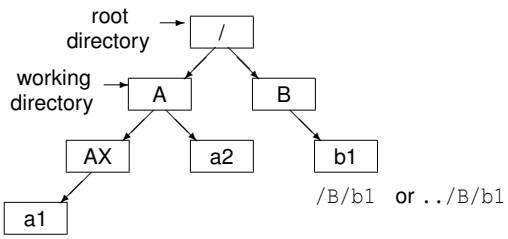
Directory: file-name → SFID
(SFID gives access to contents & attributes)

Originally:
1 directory per partition (1-level), or
1 per user (2-level)

Nowadays: Tree (or Forest) of directories

Stored on disk just like files but treated differently

Example Directory Tree-Structure



`/B/b1` or `../B/b1`

`/A/AX/a1` or `AX/a1`

Directories contain files & directories

A normal file is a leaf in the tree

Decoding a path name

Split path at separators (e.g. / or \)

Absolute: left-most component = root directory
Relative: implicitly starts with current working directory

Each component from left must:
– identify a directory
– contain the next component

Final component = file or directory

Data Structures

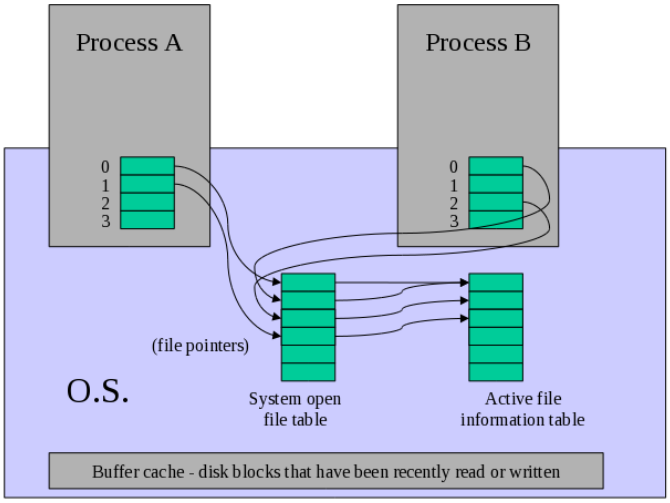
Each process has:
– working directory (inherited on creation)
– UFIDs

Each file has attributes (metadata, File Control Block FCB):
– file size, permissions, owner, group, dates, ...
– where to find data on disk

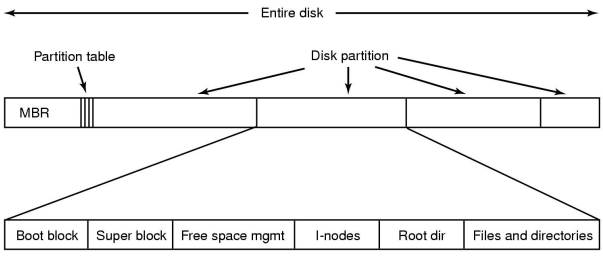
Open file table in memory:
– entry = attributes, number of readers & writers.
– indexed by UFID
`open`: create entry in the file table
(last) `close`: write attributes to disk

Implementations vary e.g. maybe also a table per process

Data Structures



Disks (MOS2 fig 6.11)



Physical structure: platters, tracks, sectors, etc.
Logical structure: blocks

Free Space

e.g. bit-map, or list of block-no? (no = number)

e.g. 100GB partition = 25M * 4kB blocks

Q: bitmap size (blocks)?

Q: block-no size (bytes)?

Q: list size (blocks)?

list can use free blocks, bitmap needs extra disk space

Q: search O(?)

File structure – contiguous blocks

(e.g. CD/DVD – ROM/WORM)

file = start-block-no & block-count

e.g. file starts at 3, uses 4 blocks:

1	2	3	4	5	6	7	8	9	
		F	I	L	E				: disk blocks

: block numbers

+ simple, fast
– interleaved user requests → seeks
– fragmentation

File structure – list of blocks (1)

"next" within block

e.g. file starts at 3 (next=0 indicates EOF)

1	2	3	4	5	6	7	8	9	
		F 4	I 7			L 9		E 0	: disk blocks

: block numbers

– random/direct access very slow

File structure – list of blocks (2)

"next" in separate monolithic table
(e.g. MSDOS FAT – File Allocation Table)

+ table can also hold free-block info.

e.g. file starts at 3; abc starts at 2 (–1 indicates free)

1	2	3	4	5	6	7	8	9	
	a	F	I	c		L	b	E	: disk blocks
–1	8	4	7	0	–1	9	5	0	: FAT

– need to cache table (e.g. 100MB) in memory

File structure – list of blocks (3)

"next" in separate partitioned data-structure

+ one table in RAM (proportional to file size) per open file

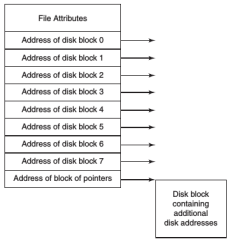
e.g. file table in block 1; abc table in block 6

1	2	3	4	5	6	7	8	9
3 4 7 9	a	F	I	c	2 8 5	L	b	E

e.g. UNIX i-node/inode

File structure – list of blocks (4)

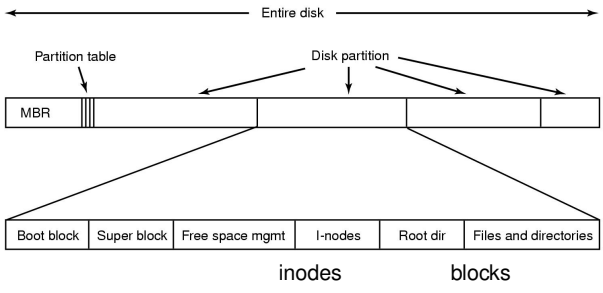
i-node = file-attributes + 11 to 15 block-nos
– first 8 to 12 = first blocks of file
– last 3 = block of block-nos, block of blocks of block-nos, ...



(MOS4 Fig. 4-13)

inodes in separate disk area (earlier slide)

Disks (again)



inodes in separate area from file/directory blocks

Directory structure

File Name → directory-entry sizes
– max length → fixed-size
(e.g. MSDOS = 8+3, early UNIX = 14)
– unlimited → variable-size (including strlen)
– unlimited → fixed-size (+ “heap” for strings)

File Attributes:
– in directory entry (e.g. FAT)
– pointed at by directory entry (e.g. in inode)

Disk Address:
e.g. FAT: block-number of start of file
e.g. inode: via inode-number

Q: efficiency? – what are the commonest directory operations?

Directory structure (MOS4)

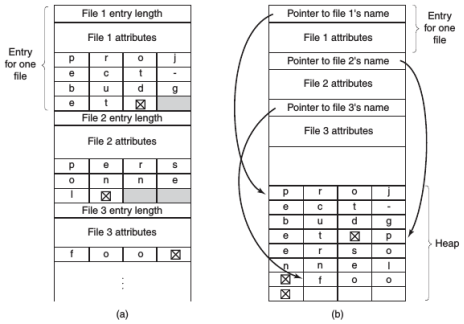


Figure 4-15. Two ways of handling long file names in a directory. (a) In-line. (b) In a heap.

Other Issues

Concurrency: how should multiple accesses be coordinated?
– allow either 1 writer; or many readers (inflexible?)
– applications (e.g. database) define specific protocols

Performance:
– cache
– efficiency dependent on algorithms/types of data
– RAID (Redundant Array of Independent Disks): striping

Access Protection:
– R/W/X permissions (man chmod)
– access control lists (e.g. man acl)

Recovery: backups!
– consistency checking (MSDOS:chkdsk, Linux:fsck)
– partitions
– journalling
– RAID: mirroring

Free Space Management

How to keep track of free space?

Keep a bit vector (0 or 1; 1 for a free disk block)
– simple to find first free block (bits_per_word) x (0_words) + (offset_first_1_bit)
– 4GB disk with 512 byte blocks needs 1MB space

Linked list of free blocks
– Need I/O to locate n free blocks
– Alternatively: keep info in the FAT

Grouping: for every n free blocks record info in 1 about the n-1

Virtual Memory and Storage

Virtual Memory & File managers both copy info RAM ⇔ disk

Unified VM/File Manager: memory-mapped files

File Operation	Equivalent VM Operation
open	map file into Virtual Address Space
read	access virtual address: page fault causes disk read
write	access virtual address: (eventual) page rejection causes disk write [n.b. zero fill on write to unallocated page?]
close	unmap pages

Pros: programs & libraries; sharing
Cons: different access patterns?

Summary of key points

1. Files & File Systems
2. Naming Service
3. Storage Service
   - Data Structures
   - Allocation
4. File Manager & Virtual Memory

Your Questions

For next time

A file system uses inodes which contain 8 block-numbers. These are for the first 7 blocks of the file and an indirect block, which just contains block-numbers for the remaining blocks in the file.
A block-number occupies 2 bytes. Each block is 4k bytes.

What is the maximum size of a file in bytes?

What is the maximum total size of directories and files in a single disk partition?

Exam Questions

Explain the algorithm used to locate the file referred to by a full path name in a hierarchical file system. (5 marks)

Using a FAT16 file system (i.e. each FAT entry occupies 16 bits) how much space would be available on a 160MB disk for directories and files, for block sizes of 2KB and of 4KB? Explain your reasoning. (5 marks)

A disk storing a hierarchical file system will hold three forms of data: directories, file contents, and metadata. Illustrate this for a system using a File Allocation Table (FAT). Explain how this information is used and modified by a process making a new copy of an existing file on the disk. (9 marks)

Glossary

file
file attributes
sequential access
direct/random access
magic number
naming service
storage service
directory
hierarchical (tree-structured) directories
SFID, UFID, file descriptor/handle
pathname
relative v. absolute pathnames
metadata, FCB
File Allocation Table, FAT
free space
memory-mapped file

Reading

MOS2: 6.1-6.3

MOS3: 4.1-4.3

OSC/J: 10, 11.1-11.5