1B11 Operating Systems

Files

Prof. Steve R Wilbur
s.wilbur@cs.ucl.ac.uk

Lecture Objectives

- What properties do we need from files?
- Why does the OS implement files?
- How is protection managed?

Why have Files?

- Up to 1970s disks were rare and small (1.2MB disk satisfied annual teaching needs of UCL CS in 1975)
- Users "owned" disks and directly allocated space (sectors and cylinders) for data storage - primitive ***files***
- They kept paper records ***(catalogues)*** of where data was stored
- As use became more common, catalogues were kept on disk - primitive ***directories***

1B11-4 1999 Slide 3

Why have Files - 2?

- Nowadays, disks are large, e.g.:
  - laptop now has 4-10 Gigabytes
  - UCL-CS has > 400 Gigabytes of server file-store
- User files are mostly small - statistics vary with business, but:
  - many files < 1 kbyte,
  - some at 10-100s kbytes
  - few of > 10 Mbyte
- But, ***file size*** needs increase over time:
  - Reason 1 -
  - Reason 2 -

1B11-4 1999 Slide 4

Why have Files - 2?

- User needs:
  - Easy way to store and retrieve data over long periods
  - Lots of files (Unix philosophy: "files are cheap")
  - Various sizes and can change during life-time
  - Virtual files - e.g. "larger" than disk/file system
  - Ability to control access to files - restrict/share

Why have Files - 3?

- OS needs:
  - To present a clean, maintained abstraction
  - To organise disk storage and allocate it efficiently
  - To track ***ownership*** and ***access rights/permissions***
  - In context of multi-user environment
  - These are main reasons why OS has to manage the ***file system***
  - OS is also a user:
    - long-term tables (e.g. passwords) stored as files
    - OS needs to load programs

Disk Organisation

- OS provides access to I/O devices such as disk via ***device drivers*** (software) which deal with competing I/O demands from the various processes
- Typically they offer ***"raw"*** access to an abstract disk that looks like an array of blocks of a fixed size, often 1 or 4 or 16kbytes - "chosen" when disk is ***formatted***
- Access is near random (i.e. similar time to get any given block)

1B11-4 1999 Slide 7

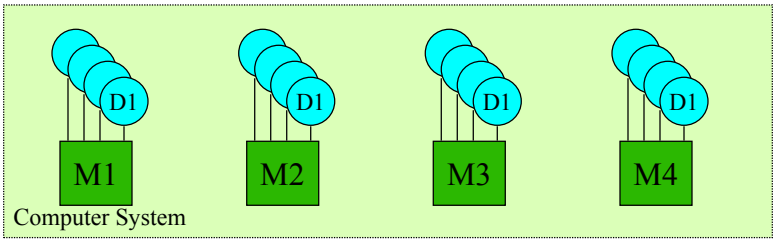
Disk Organisation - 2

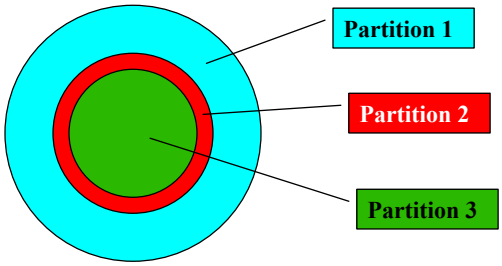
- Users may want to "buy" certain area for own use
- We may want to divide a disk into logical regions too - called ***"partitions"***

- Single machine may have many disks
- How do we manage this scale?
- … Hierarchy of directories/folders
- So we have:
  - Machine:Disk:Partition:Directory:File:Block

1B11-4 1999 Slide 8

Disk/File Organisation

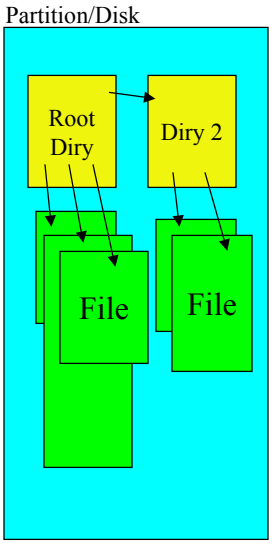




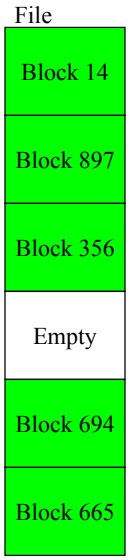
- Partitions are usually arranged as cylinders
- Minimises head movement when partition being used

1B11-4 1999 Slide 9

Disk/File Organisation



- Partition will have ***root*** or ***master*** directory
- File system will spring from this as hierarchy
- Directories contain ***names*** of files (and sub-directories) and other ***attributes***
- File is a ***sequence of bytes***
- Implemented by ***mapping*** blocks of partition to these byte addresses
- Disk space ***not allocated*** if no data at that file address



1B11-4 1999 Slide 10

File Store Implementation

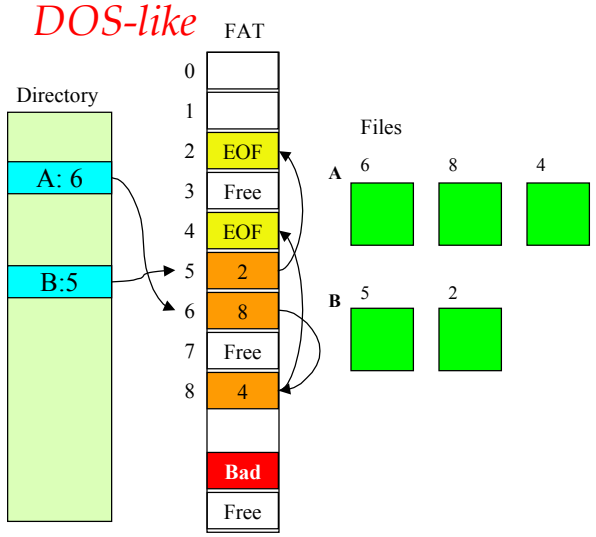
- We'll treat partitions and disks interchangeably
- Important data structures:
  - ***Free space list*** - free blocks - may be bit map or free blocks are chained
  - ***Bad block list*** - must be separate list! - identified when disk formatted, possibly later too
  - ***Directories*** - relate naming, protection and other info to files
  - ***File maps*** - indicate where blocks of file are on disk
- Directories and file maps often kept together
- Free space list and root directory in fixed place

File Store Implementation - 2

- So, how are files created, extended, etc?
- Unix and DOS separate directory from file map
  - DOS: File Allocation Table - FAT - also contains free space list
  - Unix: Free list and I-nodes

File Store Implementation - 3

DOS-like

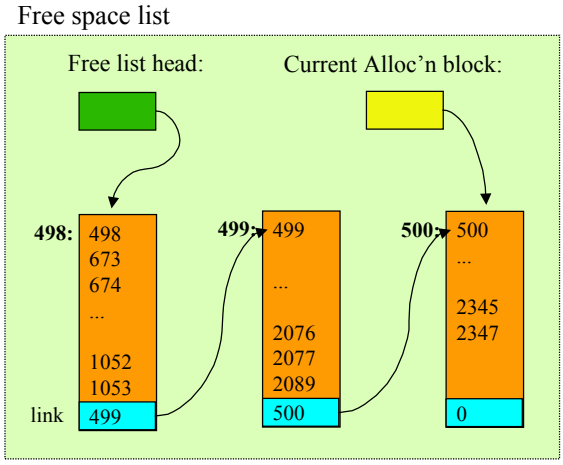


- Directory contains address (block number) of start of file
- FAT contains entry for each block of disk
- Bad blocks flagged
- Linked list of blocks making up a file
- End of file marker at end (e.g. FFFF)

1B11-4 1999 Slide 13

File Store Implementation - 4

Unix-like

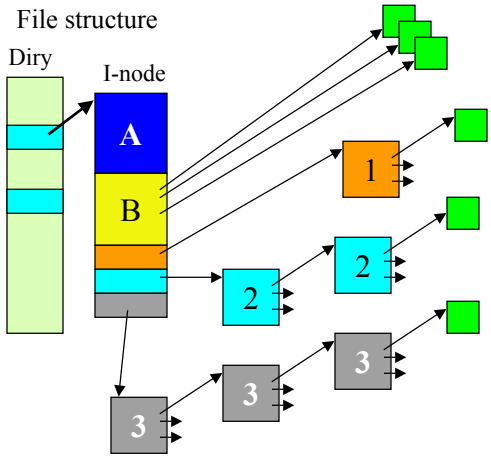


- One example shown
- Free blocks used to hold free space list
- Block's own address is last to be freed in its set - i.e. it itself gets used
- When disk is formatted bad blocks are excluded from the free list

1B11-4 1999 Slide 14

File Store Implementation - 5

Unix-like



- A contains file attributes, e.g. owner
- B - access to 10 file blocks
- 1 - further 256 blocks accessed by indirect block
- 2 - further 256*256 blocks
- 3 - further 256*256*256 blocks

- ***Indirect block*** scaling related to block size and word size

1B11-4 1999 Slide 15

File Store Implementation - 6

- DOS
  - Simpler structure
  - File size < disk size
  - Good sequential access
  - Poor random access
  - File "holes" take up disk space

- Unix
  - Separates free space and file structures
  - Sequential or random access good performance
  - Max. 3 disk accesses for address of any part of file
  - Holes may take no space

1B11-4 1999 Slide 16

User Interface to Files

- The user has an ***Applications Programming Interface (API)*** to the file system
  - Unix and C - standard I/O library
  - Java VM by a class library
- API makes ***system calls*** (traps) to operating system, e.g.:
  - *open(), close(), read(), write(), seek(), unlink()*

File System Performance

- Disk access is 1000s times slower than memory
- OS keeps copies of file blocks in use in its own memory ***(buffers*** or ***cache)***
- ***Principle of Locality*** so OS needs to manage read/write of blocks to disk and buffer replacement policy
- Same principle suggests high likelihood of sequential file access
- Unix uses this to ***read-ahead*** next block so ready when (if) user needs it

File System Performance - 2

- To avoid loss of data due to power glitches OS periodically ***flushes*** buffers to disk
- Also need cache write-through policy to ensure file updates not lost

Directory Entries

- A directory entry on disk might hold:
  - file name + disk address (I-node in Unix)
  - attributes of the file:
    - owner
    - size, type
    - protection information, e.g. rwx for owner, group, others
    - creation time, last access or modified time
- Few systems (except MACs) put the type in the directory - elsewhere:
  - implicit in the name/extension, by convention
  - derivable by looking at part of the file
  - e.g. .txt, .c, .psd, .doc, ...

Kinds of Files

- Most OSs nowadays support two kinds of file:
  - Array of bytes
  - Indexed sequential
- Windows and Unix only support the first
- Programmers need to enhance the "array of byte" model - but this is easy…
- Just write another file as the index (there are Unix application programmer's libraries to help you do just this)

1B11-4 1999 Slide 21

File System Differences

- The basic ideas in MAC-OS, Unix and Windows file systems are very similar nowadays, but there are a lot of detailed differences - some include:
- maximum file or disk or partition size supported
  - model of user id, permissions
  - file type information
  - file name restrictions.

1B11-4 1999 Slide 22

Backup/Archive of Files...

- Disks are not 100% reliable, nor are people - files get lost
- Backup systems periodically copy all changed files (incremental) to tape (or CD or Zip drive)
- Less frequently, do full "dump" of file system, and then store tapes somewhere different (possibly 2 places)
- UCL CS has tapes going back to early '80s and can in theory retrieve any file from any day since then……in theory:-)

Summary

- Files implemented by OS as convenient way of centrally managing disk space
- File system needs to deal flexibly with range of file sizes and give good performance
- Also, in memory caching used to improve performance
- Basic byte-array file can be extended by libraries in user space