File Systems

Operating Systems

G53OPS: Operating Systems

Graham Kendall

File Systems

Operating Systems

Why Use Files?

- It allows data to be stored between processes
- It allows us to store large volumes of data
- Allows more than one process to access the data at the same time

File Systems

Operating Systems

Two Views of File System

- User View
- Implementators View

File Systems

Operating Systems

File Naming

- Different operating systems have different file naming conventions
- MS-DOS only allows an eight character filename (and a three character extension)
- This limitation also applies to Windows 3.1
- Windows 95 and Windows NT allow filenames up to 255 characters (although the full path name is only allowed to be a maximum of 260 characters)

File Systems

Operating Systems

File Naming

- Restrictions as to the characters that can be used in filenames
- Some operating systems distinguish between upper and lower case characters
- To MS-DOS, the filename ABC, abc, and AbC all represent the same file. UNIX sees these as different files

File Systems

Operating Systems

File Extensions

- Filename are made up of two parts (typically PC based OS’s) separated by a full stop
- The part of the filename up to the full stop is the actual filename
- The part following the full stop is often called a file extension
- In MS-DOS the extension is limited to three characters
- UNIX and Windows 95/NT allow longer extensions

File Systems

Operating Systems

File Extensions

- Used to tell the operating system what type of data the file contains
- It associates the file with a certain application
- Using tools provided with the operating system the user is able to change the file associations
- UNIX allows a file to have more than one extension associated with it

File Systems

Operating Systems

Common File Extensions

Extension	File Contents
BIN	Binary File
C	C Program File
CPP	C++ Program File
DLL	Dynamic Link Library
DOC	Microsoft Word file
EXE	Executable File
HLP	Help File
TXT	Text File
XLS	Microsoft Excel File

File Systems

Operating Systems

File Attributes

- Each file has a set of attributes associated with it
- Typical attributes:

Attribute	Description
Archive Flag	Bit Field : has the file been archived?
Creation Date/Time	Date and Time file was created
Creator	User ID of the person creating the file
Hidden Flag	Bit Field : Is the file a hidden file?
Last Accessed Date/Time	Date and Time file was last accessed
Owner	The ID of the current owner
Password	Password required to access the file
Protection	Access rights to the file
Read-Only	Bit Field : Ids the file read only?
Size in Bytes	How large is the file
System Flag	Bit Field : Is the file a system file?
Temporary Flag	Bit Field : Should the file be deleted at end of the process?

File Systems

Operating Systems

File Structure

- Store the file as a sequence of bytes. It is up to the program that accesses the file to interpret the byte sequence
  - Fixed length records
  - Variable length records
  - Indexed Files

File Systems

Operating Systems

File Access

- Sequential Access
- Batch Updating Model
- Random Access

File Systems

Operating Systems

Directories

- Allow like files to be grouped together
- Allow operations to be performed on a group of files which have something in common. For example, copy the files or set one of their attributes
- Allow files to have the same filename (as long as they are in different directories). This allows more flexibility in naming files
- Typical directory entry contains a number of entries; one per file

User View

File Systems

Operating Systems

Directories

- All the data (filename, attributes and disc addresses) can be stored within the directory
- Alternatively, just the filename can be stored in the directory together with a pointer to a data structure which contains the other details
- Hierarchical Directory Structure
- Simulating a hierarchical directory structure?

User View

File Systems

Operating Systems

Path Names

Absolute path names

- C:\COURSES\OPS\FILE SYSTEMS
- OR
- \COURSES\OPS\FILE SYSTEMS

Relative path names

- Related to Current Working Directory (CWD)
- If CWD is C:\COURSES then the relative path name for the above file would be
- OPS\FILE SYSTEMS

User View

File Systems

Operating Systems

Working Directory

Finding out the CWD

- Under UNIX – PWD
- Under MS-DOS it is usual to change the command prompt so that the current working directory is displayed:
- PROMPT pg
- $p displays the current drive and working directory
- $g tells MS-DOS to display a '>'
- '.' and '..' – what do they represent?

User View

File Systems

Operating Systems

Operations

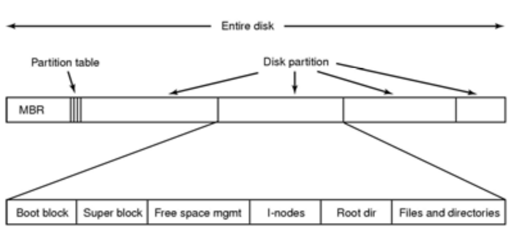
- Copy
- Move
- Rename
- etc..

Implementators View

File Systems

Operating Systems

Possible File System Layout



Implementators View

File Systems

Operating Systems

Implementation (Contigous)

Contiguous Allocation

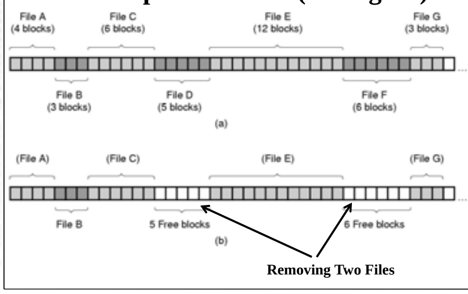
- Allocate *n* contiguous blocks to a file. If a file was 100K in size and the block was 1K then 100 contiguous blocks would be required

Operating Systems

Implementators View

File Systems

Implementation (Contigous)



Operating Systems

Implementators View

File Systems

Implementation (Contiguous)

Advantages
- It is simple to implement as keeping track of the blocks allocated to a file is reduced to storing the first block that the file occupies and its length
- The performance of such an implementation is good as the file can be read as a contiguous file. The read write heads have to move very little, if at all. You will never find a filing system that performs as well

Operating Systems

Implementators View

File Systems

Implementation (Contiguous)

Disadvantages
- Leads to fragmentation
- We need to keep a list of unused blocks (doable, but expensive)
- The operating system does not know, in advance, how much space a file can occupy
- Need to run defragmentation process periodically, but it is expensive

Operating Systems

Implementators View

File Systems

Implementation (Contiguous)

Question
- Can you think of a scenario where a contiguous file allocation scheme could be used?

Operating Systems

Implementators View

File Systems

Implementation (Contiguous)

Question
- Can you think of a scenario where a contiguous file allocation scheme could be used?

- Write once media (CDs, DVDs etc.)

Operating Systems

Implementators View

File Systems

Implementation (Linked List)

- Blocks of a file represented using linked lists
- All that needs to be held is the address of the first block that the file occupies
- Each block contains data and a pointer to the next block

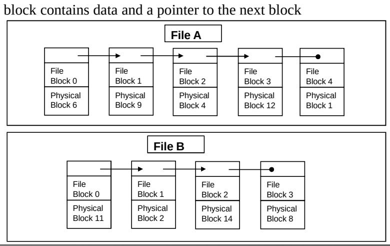
Implementators View

File Systems

Operating Systems

Implementation (Linked List)

- Blocks of a file represented using linked lists
- All that needs to be held is the address of the first block that the file occupies
- Each block contains data and a pointer to the next block



Implementators View

File Systems

Operating Systems

Implementation (Linked List)

Advantages

- Every block can be used, unlike a scheme that insists that every file is contiguous
- No space is lost due to external fragmentation (although there is internal fragmentation within the file)
- The directory entry only has to store the first block number. The rest of the file can be found from there
- The size of the file does not have to be known beforehand (unlike a contiguous file allocation scheme)
- When more space is required for a file any block can be allocated (e.g. the first block on the free block list)
- Reading a file sequentially is straightforward, although may require more disc accesses than a contiguous allocation

Implementators View

File Systems

Operating Systems

Implementation (Linked List)

Disadvantages

- Random access is very slow. It needs many disc reads to access a random point in the file (*n*-1 accesses are required to get to block *n*)
- Space is lost within each block due to the pointer. This does not allow the number of bytes to be a power of two. This is not fatal, but does have an impact on performance
- Reliability could be a problem. It only needs one corrupt block pointer and the whole system might become corrupted (e.g. writing over a block that belongs to another file)

Implementators View

File Systems

Operating Systems

Implementation (Linked List: Using a Table in Memory)

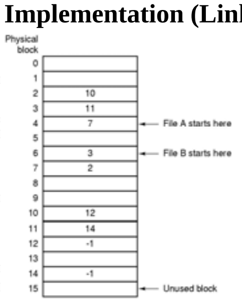
- Store the pointers in an index (often called a File Allocation Table (FAT))
- Does not waste space in the block
- Random access is possible as index is in memory
- Therefore, eliminates the two main disadvantages of using linked lists

Implementators View

File Systems

Operating Systems

Implementation (Linked List: Using an Index)



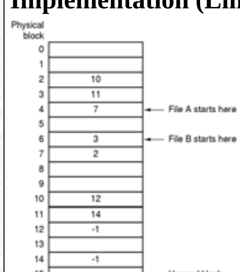
Disadvantages?

Implementators View

File Systems

Operating Systems

Implementation (Linked List: Using an Index)



- Main disadvantage is that the entire table must be in memory all the time
- For a (small) disc of 20GB, with a 1K block size, that requires 20 million entries. At 3 bytes per entry that is 60MB in main memory

Implementators View

File Systems

Operating Systems

Implementation (I-Nodes)

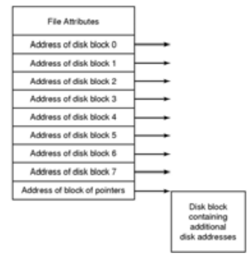
- All the attributes for the file is stored in an i-node entry, which is loaded into memory when the file is opened
- The i-node also contains a number of direct pointers to disc blocks. Typically there are twelve direct pointers
- Only keep the i-node in memory if the file is open.
- If each i-node has *n* bytes and a maximum of *k* files can be open then the i-nodes take a maximum of *nk* bytes, regardless of disc size

Implementators View

File Systems

Operating Systems

Implementation (i-Nodes)



Implementators View

File Systems

Operating Systems

Implementation (UNIX i-Nodes)

- UNIX V7 File System (PDP-11)
- A UNIX directory contains one entry for each file in that directory
- Each entry is very simple (name (14 bytes)/i-node number (2 bytes))

Implementators View

File Systems

Operating Systems

Implementation (UNIX i-Nodes)

- UNIX V7 File System (PDP-11)
- In addition there are three additional indirect pointers. These pointers point to further data structures which eventually lead to a disc block address
- The first of these pointers is a single level of indirection, the next pointer is a double indirect pointer and the third pointer is a triple indirect pointer

Implementators View

File Systems

Operating Systems

Implementation (UNIX i-Nodes)

Attributes

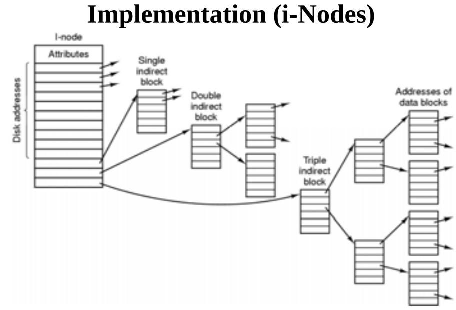
- File size
- Three times (creation, last accessed, last modified)
- Owner
- Group
- Protection information
- Number of directory entries pointing to that i-node (to cater for links)

Implementators View

File Systems

Operating Systems

Implementation (i-Nodes)

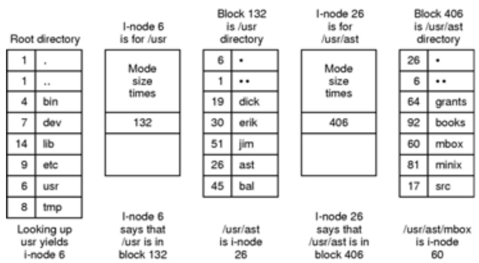


Implementators View

File Systems

Operating Systems

Implementation (i-Nodes)



Implementators View

File Systems

Operating Systems

Implementation (MS-DOS vs i-node)

- Under MS-DOS a directory entry is 32 bytes long. It is split as follows

8 (bytes)	3 (bytes)	1 (byte)	10 (bytes)	2 (bytes)	2 (bytes)	2 (bytes)	4 (bytes)
Filename	Extension	Attributes	Reserved	Time	Date	First Block	Size

- Under UNIX we only need to store the file name and i-node number (as all the attributes are stored in the i-node)

2 (bytes)	14 (bytes)
i-node #	Filename

Implementators View

File Systems

Operating Systems

Case Study

- Chapter 10 of the course textbook (ed. 2) is a case study of Unix and Linux

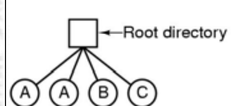
Implementators View

File Systems

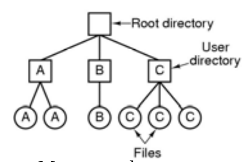
Operating Systems

Directories

- **Single Level Directory Structure vs Two Level Directory Structure**



- Simple
- Problems with multiple filenames
- Still has its uses



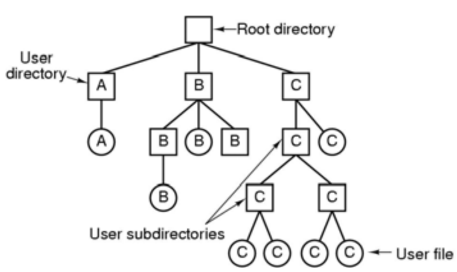
- More complex
- More flexible
- Assumes a user structure

Implementators View

File Systems

Operating Systems

Directories

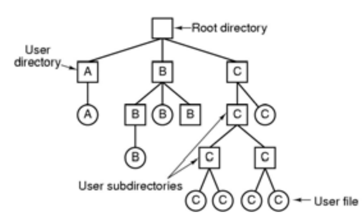


Implementators View

File Systems

Operating Systems

Directories



- Should we allow users to access other user’s files?
- Probably yes, but now we have to include security
- Also enables to have common resources (e.g. executables)

Implementators View

File Systems

Operating Systems

Disk Space Management

- Whatever block size we choose then every file must occupy this amount of space as a minimum
- If we choose a large allocation unit, such as a cylinder then even a 1K file will occupy a cylinder
- Choosing a small allocation size (of say 1K) means that files will occupy many blocks which results in more time accessing the file as more blocks have to be located and accessed
- There is a compromise between a block size, fast access and wasted space. The usual compromise is to use a block size of 512 bytes, 1K bytes or 2K bytes

Implementators View

File Systems

Operating Systems

Disk Space Management

- Some of the free blocks (which are no longer be free!) hold disc block numbers that are free
- The blocks that contain the free block numbers are linked together so we end up with a linked list of free blocks

Implementators View

File Systems

Operating Systems

Disk Space Management

- We can calculate the maximum number of blocks we need to hold a complete free list (i.e. an empty disc) using the following reasoning:
  - Assume that we need a 16-bit number to store a block number (that is block numbers can be in the range 0 to 65535)
  - Assume that we are using a 1K block size
  - A block can hold 512 block addresses. That is, 1024*8 [number of bits in a block] / 16 [bits needed for a block address]
  - Assume that one of the addresses is used as a pointer to the next block that contains list of free blocks
  - For a 20Mb disc we need, at most, 41 blocks to hold all the free block numbers. That is, 20*1024 [maximum number of blocks] / 511 [number of disc addresses in a block]

Implementators View

File Systems

Operating Systems

Disk Space Management

- A bit map is used to keep track of the free blocks
- That is, there is a bit for each block on the disc
- If the bit is 1 then the block is free. If the bit is zero, the block is in use
- To put it another way, a disc with *n* blocks requires a bit map with *n* entries
- The directory entry may also contain the attributes of the file (i-node) or may contain a pointer to a data structure

Implementators View

File Systems

Operating Systems

Disk Space Management

- Consider a 20Mb disc with 1K blocks, then we can calculate the number of blocks needed to hold the disc map.
- A 20Mb disc has 20,480 (20 * 1024) blocks
- We need 20,480 bits for the map, or 2,560 (20,480 / 8) bytes
- A block can store 1024 bytes so we need 2.5 blocks (2560 / 1024) blocks to hold a complete bit map of the disc. This would obviously be rounded up to 3

Implementators View

File Systems

Operating Systems

Disk Space Management

- Generally, bit maps requires a lesser number of blocks than a linked list
- Only when the disc is nearly full does the linked list implementation need fewer blocks

File Systems

Disk Space Management

Advantage of Linked List Over Bit Map

- When only a small amount of memory can be given over to keeping track of free blocks
- Assume, the operating system can only allow one block to be held in memory and that the disc is nearly full
- Using a bit map scheme, there is a good chance that the free block list will indicate that every block is being used
- This means a disc access must be done in order to get the next part of the bit map
- With a linked list scheme, once a block containing pointers of free blocks has been brought into memory then we will be able to allocate 511 blocks before doing another disc access