**CIS2380 (second term) - Week 9 - File Systems**

(1) File Systems - general
The concept of a file is of a uniform logical view of information storage. This must be long term and
(i) store large amounts of data;
(ii) allow information stored to outlast the process using/creating it;
(iii) allow multiple processes to access the information concurrently.
A file is a named collection of related information that is recorded on secondary storage. Files are mapped by the operating system onto physical devices.

*File Management* is an operating system function which was developed to release programmer from the problem of creating software.

**File Attributes**
Possible file attributes are in Figure 1 and extensions to file names are in Figure 2.
File operations The main ones are as follows:
Other operations are Append, Seek (within a file), Get attributes, Set Attributes and Rename
A file system is composed of many different levels - each of which uses the features of lower levels:

I/O control and devices The device is the lowest level - I/O control consists of device drivers, interrupt handlers (etc) to transfer data between memory and disc.

basic file system control of physical blocks of data

file-organisation module knows about files and their logical blocks as well as physical blocks.

logical file system manages metadata - the file-system structure/directory structure.

application program Writes/read files

Performance depends on organisation - the logical file system. For example we can compare sequential and indexed sequential files of records. If a sequential file contains 1 million records, on average 500,000 accesses will be required to find a given record. However if the file is indexed and an index contains 1000 entries it will take on average 500 accesses to find the key, followed by 500 accesses in the main file.

In a similar manner a user will organise files via directories in order to identify and locate a selected file. There is also the need to manage files in a shared system so that their use is controlled. The need is to manage file storage (and access) block by block - not necessarily optimise storage.

Directories
(1) Provide information about files - attributes, location, ownership;
(2) Directory itself is a file created by operating system;
(3) Provide mapping between user file names and files themselves.

Figure 3 shows a single level directory system
A simple structure for a directory would be a list of entries, one for each file - an indexed sequential file, with the name of the file serving as the key.
Figure 4: A possible file system layout

Figure 5: (a) Contiguous allocation of disc space for seven files. (b) The state of the disc after files D and F have been removed.

However this is of no help in organising files. Also the user has to be careful not to use the same name for 2 different files.

Implementation - Disc Storage
File systems are stored on discs - which are generally partitioned with independent file systems on each partition. Sector 0 of the disk is the Master Boot Record (MBR) and is used to boot the computer via the boot block which potentially contains the OS. (The space on the disc is reserved even if it is not there.) The super block contains information about the file system. Figure 4 shows a possible arrangement. The simplest implementation of disk organisation is of contiguous storage as in Figure 5.

The following is a brief overview of two systems - Unix and Windows XP, how files are stored, directory structure, access control etc.

(2) Unix File System: From the point of view of the user there are three kinds of files: ordinary files, directories and special files. (See later for differences.) They are all accessed in storage by means of their file identifier or i number. The i number is used as an index into a system table or i list. The i list contains the i node (description of the file) which includes:

owner information - i.e. user and group-ID;

protection - bits indicating who can read to, write to, execute the file;
addresses of file in storage device;

size file size

time(s) of creation, modification, last use;

links - the number of times a file appears in a directory. See later for explanation;

file type - a code indicating whether the file is a directory, an ordinary file or a special file.

A file can be written across several blocks - Figure 6 indicates this. The purpose of an open (read or write) or create system call is to turn the (path) name given by the user into an i number using a search mechanism. The file can then be written to or read from via its i node. When a new file is created with a given name, an i node is allocated for it, with associated file description including a new i number. However if the file name already exists, it is emptied.

Link and Copy

An existing (ordinary) file can be linked to a new filename, so that it is possible to access a file via two different names. The new file name is associated with the file identifier of the existing file, and with its i node. If a file is copied, then a new i node is created with storage address(es) equal to the storage address(es) of the existing file.

Listing files

The command ‘more’ lists files to the terminal. The contents of the file is unaltered.

Types of File

An ordinary file contains whatever information a user places in it, e.g. programs, reports etc. A file of text is a string of characters; a binary file is a sequence of words etc. There is no particular structure.

A directory provides a mapping between the names of files and the files themselves. There is a structure to directories and they can only be written on by privileged programs. However anyone with appropriate permissions can read a directory. Figure 7 shows the steps in locating /usr/ast/mbox through the pathname.

Special files support I/O devices such as printers or the screen. These are protected from indiscriminate access.

Berkeley Fast File System

The University of Berkeley made some improvements to the Unix file system. These were as follows:

(i) File names of up to 255 characters were allowed.

(ii) Each directory consists of an integral number of disc blocks.

Within a directory, files and directories are in unsorted order. Entries may not span disc blocks, so there are unused bytes at the end of each disc block.
I-node

Attributes

Disk addresses

Single indirect block

Double indirect block

Triple indirect block

Addresses of data blocks

Figure 6: A UNIX i-node

Looking up
usr yields
i-node 6

I-node 6 says that
/usr is in
block 132

I-node 26
is /usr/ast
directory

I-node 26
says that
/usr/ast is in
block 406

/usr/ast/mbox

Figure 7: The steps in looking up /usr/ast/mbox
The disc is divided into cylinder groups - each with its own superblock, i-nodes and data blocks. The idea is to keep the i-node and data blocks of a file close together, to avoid long seeks. (See Figure 8.)

Figure 8: (a) A BSD directory with three files. (b) The same directory after the file *voluminous* has been removed.

TWO block sizes are introduced rather than one. A small number of large blocks is more effective than many small ones. The small block size is useful for small files. The price paid is extra complexity in the code.

File access - Unix bit system (See also Week 8)

Each process gets the user id and group id of its owner and files created by that process inherit these ids. The processes determine the permissions for the file and include permissions for other users. Potential accesses are (rwx) and each is described by a bit (9 bits in total). A r or w or x means that the access right is granted a ‘-‘ (dash) a indicates ‘not granted’. For example for file1.txt

```
-rwrx----  tonyb  cabinet file1.txt
drwrx-----  tonyb  cabinet StateSecrets/
```

indicates that the owner of the file (tonyb) can read write and execute. The group (cabinet) can read, but not write or execute. Other users can not read or write or execute. (The equivalent binary is 111 100 000.) For the directory - only tonyb can access this.

Advantages and Disadvantages of Unix ‘bit’ (rwx) system

Disadvantages of this system are that access is limited by single user or group ownership. Advantages are that we use the same data structure for every file.
(3) **Windows XP Files**

The “New Technology” (NT) portable operating system was developed in 1988. There is a whole family of ‘NT’ operating systems and Windows XP is one member. Windows XP was released both as an update to the Windows 2000 desktop OS and a replacement for Windows 95/98. (See also Week 11 on OS Case Studies.)

Windows 2000 is characterised by the use of objects for they provide a uniform data structure for the OS. An object is created for a file when a disc file is accessed. Windows uses the NT file system (NTFS) and also continues to use FAT16 from MS-DOS to read floppies and to maintain interoperability with Windows 95/98. An NTFS file is not just a sequence of bytes (as Unix is). A file consists of multiple attributes, each of which is a byte stream. Most files have a few short streams (file name and object ID) and one long stream (data). However data can consist of several streams - for example a photo editing program has one stream for the main image and another for a thumbnail. Another example is word processing, where one stream is the temporary file and the other is the final edited version. The maximum stream length is $2^{67}$ - really huge!

The fundamental entity of NTFS is the volume - which is based on disc partitioning. However a volume can span several discs. Each volume is a linear sequence of blocks. The main data structure in each volume is the **Master File Table (MFT)** which contains records describing files or directories. Each record contains a file’s attributes and a list of disc addresses where its blocks are located (among other things). Figure 9 shows an MFT record with information about data blocks for a file of nine blocks. The blocks are in three ‘runs’ 20-23, 64-65, 80-82. In this case all the information fits one one MFT record - for larger files it may take more than one record.

**Directories** are hierarchical as for Unix. NTFS also supports hard links in the Unix sense. (The separator is a \ rather than a /.)a They are also
represented in MFT records, where each record contains the directory entries for the files/sub-directories. See Figure 10 which indicates steps in looking up a file C:\\maria\\web.htm after a call of

CreateFile(‘‘C:\\maria\\web.htm’’, ...)  

Explanation follows.  
(1) The directory \?? contains the names of all the MS-DOS device names such as A: for the floppy and C: (name chosen so it is alphabetically first!) 
(2) This object is the first partition of the first hard disc. 
(3) web.htm is eventually found in “maria” 
(4) a new object is created by the object manager and a “handle” to the object returned to the calling process.  

Access Control Lists  
“Objects” (such as files) need to be protected and the methodology is to introduce concept of domain - a set of (objects, rights) pairs where the right in the case of a file could be read, write, execute. A domain could be defined by its user or group ownership. A convenient way of storing this information is via Access Control Lists. This associates each object with an ordered list of domains that may access the object together with their rights.  
Example Suppose A is a process owned UserA; B is a process owned UserB File1 has list [A: RW; B: R]  
File2 has list [A: R; B: RWX] (etc)  
Owner access can also be via a group rather than a single user. For example The list for File1 could also include group id ZZ: File1 has list [A: RW; B: R; ZZ : RWX]  

Advantages and Disadvantages of ACL  
Disadvantages of ACL: Constructing ACLs is time consuming - we need to list all users with read access - we do not know potential users in advance.
Also Directory entry for file needs to be of variable length to retain the information.

Advantages of ACL: User can specify specific domains and is not limited by single user or group ownership. For example might wish to only allow 2 users read permission (example above).

(4) File Servers

RAID is short for Redundant Array of Independent (or Inexpensive) Disks. They are a category of disk drives that employ two or more drives in combination for fault tolerance and performance. RAID disk drives are used frequently on servers but aren't generally necessary for personal computers. Having a large number of disks in a system presents opportunities for improving the rate at which data can be read or written, if the disks are operated in parallel. Also - redundant information can be stored on multiple discs.

Data striping data can be split:

(1) by splitting the bits of each byte across multiple discs
(2) blocks are striped across multiple discs.

There are number of different RAID levels:

- Level 0 – Striped Disk Array without Fault Tolerance: Provides data striping (spreading out blocks of each file across multiple disk drives) but no redundancy. This improves performance but does not deliver fault tolerance. If one drive fails then all data in the array is lost.
- Level 1 – Mirroring and Duplexing: Provides disk mirroring. Level 1 provides twice the read transaction rate of single disks and the same write transaction rate as single disks.
- Level 2 – Error-Correcting Coding: Not a typical implementation and rarely used, Level 2 stripes data at the bit level rather than the block level.
- Level 3 – Bit-Interleaved Parity: Provides byte-level striping with a dedicated parity disk. Level 3, which cannot service simultaneous multiple requests, also is rarely used.
- Level 4 – Dedicated Parity Drive: A commonly used implementation of RAID, Level 4 provides block-level striping (like Level 0) with a parity disk. If a data disk fails, the parity data is used to create a replacement disk. A disadvantage to Level 4 is that the parity disk can create write bottlenecks.
- Level 5 – Block Interleaved Distributed Parity: Provides data striping at the byte level and also stripe error correction information. This results in excellent performance and good fault tolerance. Level 5 is one of the most popular implementations of RAID.
- Level 6 – Independent Data Disks with Double Parity: Provides block-level striping with parity data distributed across all disks. (etc)

There are many more RAID categories - the above is a sample.

Stateless/ Stateful Servers There are two approaches

(i) stateful: the server tracks each file being accessed by a client
(ii) stateless it simply provides blocks as they are requested by the client
without knowledge of the blocks usage.

**Stateful servers:** know about open files for they establish a connection identifier which is unique to the client and open file. Thus the server keeps main-memory information about its clients

**Stateless servers:** each request is self contained so that the file and position in the file is identified in full. Reads and writes take place as remote messages or cache lookups. Instead of open, the server sends a lookup message containing the file name, with a request to look it up and return the file handle. Unlike an open call, the lookup operation does not copy internal system tables. The read call contains the file handle, the offset and the number of bytes to be read. The server does not have to remember anything about open connections in between calls to it.

**Advantages and Disadvantages** Advantages of a Stateful service over a Stateless server is increased performance as file information is cached in main memory - saving disk accesses. In addition they can read ahead the next blocks.

Advantages of a Stateless server over a Stateful server is that the effects of failures and recovery are almost unnoticeable - since there is no state to be restored. No information about open files is lost for there is none.

**Caching** A technique used to improve performance is caching - where servers cache data to avoid disc accesses - but this is invisible to the clients. However caching also involves problems of coherence - if two clients are caching the same block for example. If one modifies the block the other does not necessarily read the updated version. A solution is to associate each block with a timer - when the timer expires the entry is discarded. In addition whenever a cached file is opened a message is sent to the server to find out when the file was last modified. If the modification occurred after the last local copy was cached, the cache copy is discarded and the new copy fetched from the server.

**Multimedia Files (and their presentation/delivery)**

A typical movie shown on a TV channel consists of several files: for example audio, video and text. The characteristics of these files are very different from the traditional text files that current file systems were designed for.

Two of the most important are:

(i) Multimedia uses extremely high data rates (because of nature of visual and acoustic information);

(ii) Multimedia requires real-time playback (e.g. frames must be delivered at approximately. 40ms for smoothness of look). Furthermore - a movie may consists of several files including video, audio in several languages and subtitles in several languages.

We will (briefly) look at delivery of movies via video servers. Two aspects are considered - of file placement (on a disk) for ‘near video on demand’ and ‘caching’.

**Video on demand** occurs when when customers can select a movie and
Figure 11: Optimal frame placement for near video on demand

have it displayed straight away on their TV or computer via (say) cable TV. ‘Near Video on demand’ is in contrast to this - the customer has to wait for a set time (say every 5 minutes). This means that the movie is going out in multiple staggered streams - this is displayed in Figure 11 for a movie running at 30 frames/sec with a new stream starting every 5 minutes.

Frame sets of 24 frames are concatenated and written to the disk as a single record. When stream 24 starts, stream 23 will have reached frame 9000 (= 5 x 30 x 60) so the disc arm can remain on track 1 until it is completed - then move to track 2.

**Caching** Since the usual access pattern is that videos are viewed from beginning to end sequentially, traditional caching methods do not work - for a block is unlikely to be needed a second time. However they can be used in the following manner if two users are watching the same movie - 2 secs apart. Thus whenever a block is used which may be needed again shortly (because a second or even third viewer will need it) it is cached - and only discarded if it will not be needed for some time. The idea can be taken further - for in some cases two streams can be merged. The frame rate for both movies is altered slightly - user 1 is slowed down until user 2 catches up (see Figure 12). After the two user streams have been merged caching can be discontinued.

Another way that caching can be used to make video delivery more efficient is to store each movie on DVD or tape (as it is not feasible to store 2GB movies entirely on disk). The first few minutes of each movie is on disk and while this is playing the whole DVD can be copied.
Figure 12: (a) Two users watching the same movie 10 sec out of sync. (b) Merging the two streams