FILE SYSTEM BASICS

Module Number 4. Section 1
COP4600 – Operating Systems
Richard Newman

MAJOR OS COMPONENTS

- Process management – who gets the CPU?
  - How do processes co-exist and interact?
- Input/Output – how do processes interact with the rest of the world?
  - User interactions – keyboard/video monitor/mouse/etc.
  - Peripheral interactions – mass storage, network, etc.
  - Non-sharable resources and deadlock
- Memory Management – where do we put processes?
  - How much memory does a process get? Where?
  - What it is not enough?
- File Systems – need for persistent storage
  - How do we name files?
  - How do we store/retrieve efficiently?
  - Interactions with other OS systems

MAJOR OS COMPONENTS

- Process management – who gets the CPU?
  - How do processes co-exist and interact?
- Input/Output – how do processes interact with the rest of the world?
  - User interactions – keyboard/video monitor/mouse/etc.
  - Peripheral interactions – mass storage, network, etc.
  - Non-sharable resources and deadlock
- Memory Management – where do we put processes?
  - How much memory does a process get? Where?
  - What it is not enough?
- File Systems – need for persistent storage
  - How do we name files?
  - How do we store/retrieve efficiently?
  - Interactions with other OS systems

FILE SYSTEMS TOPICS

- Introduction
- Directories
- File Allocation
- Block Management
- File System Reliability
- File System Optimization

FILE SYSTEMS INTRODUCTION

- Need for files
- File Structures
- File Types
- File Attributes
- File Operations
- System call for files

STORING/RETRIEVING INFORMATION

Essential requirements for long-term information storage:

1. It must be possible to store a **very large amount** of information.
2. The information must **survive the termination** of the process using it.
3. **Multiple processes** must be able to access the information concurrently.
4. Desirable to have **human-friendly name**.

WHAT IS A FILE?

A file is:

1. Logically named,
2. Persistent storage.

A file may have structure according to application.

The operating system and/or shell may know about the file structure from its type.

FILE SYSTEMS – UNDERLYING HARDWARE

Think of a disk as a linear sequence of fixed-size blocks and supporting two operations:

1. Read block *k*.
2. Write block *k*

The file system needs to get from human-friendly name space to disk blocks somehow!

FILE SYSTEMS – INFORMATION MANAGEMENT

Questions that quickly arise:

1. How do you find information?
2. How do you keep one user from reading another user's data?
3. How do you know which blocks are free?
4. How can you tell what type of file it is?
5. How can you associate a file type with applications?

FILE STRUCTURE

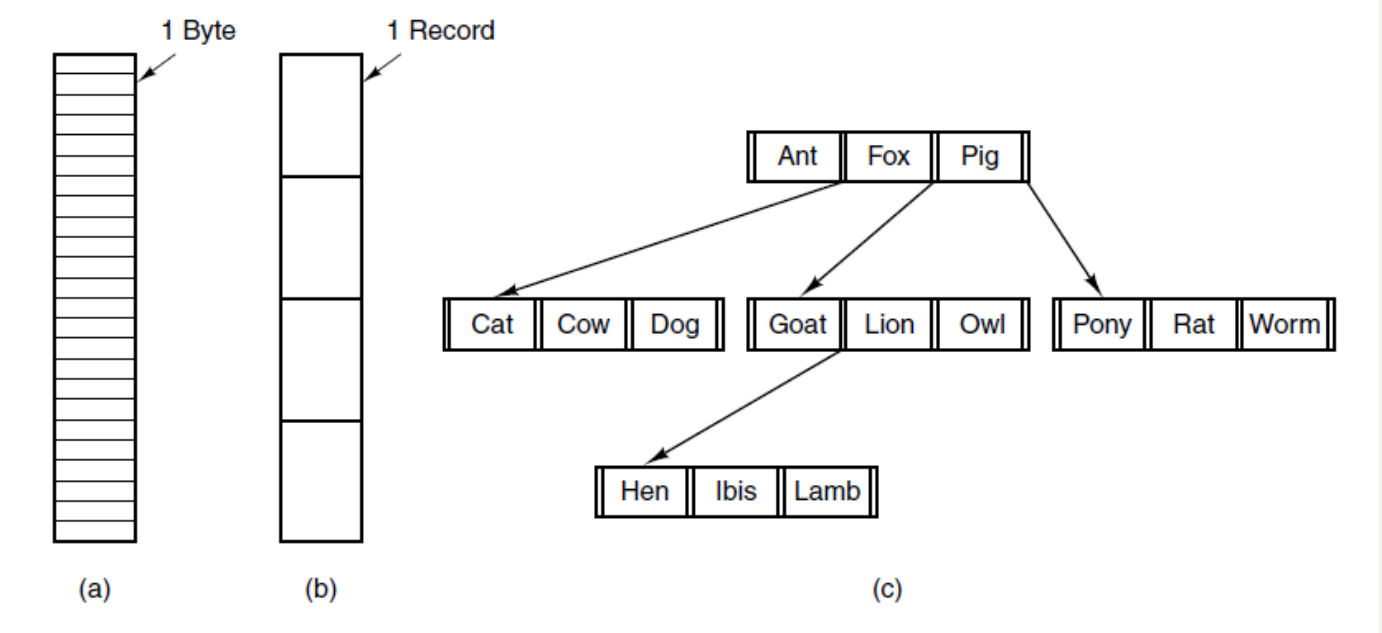


Figure 4-2. Three kinds of files. (a) Byte sequence. (b) Record sequence. (c) Tree.

Question: How do we know which structure?
And how do we find the disk blocks of a file?

FILE TYPES

Why care about type?

- Know how to interpret structure/contents
- Associate applications with types

How can you tell what type of file you have?

1. Via the name extension (e.g., 3 characters in MS/DOS)
2. Via the "magic number" (first few bytes of file)
3. Via a file attribute (in File Control Block)
4. Infer from contents (Bueno suerte!)

FILE TYPE VIA NAMING

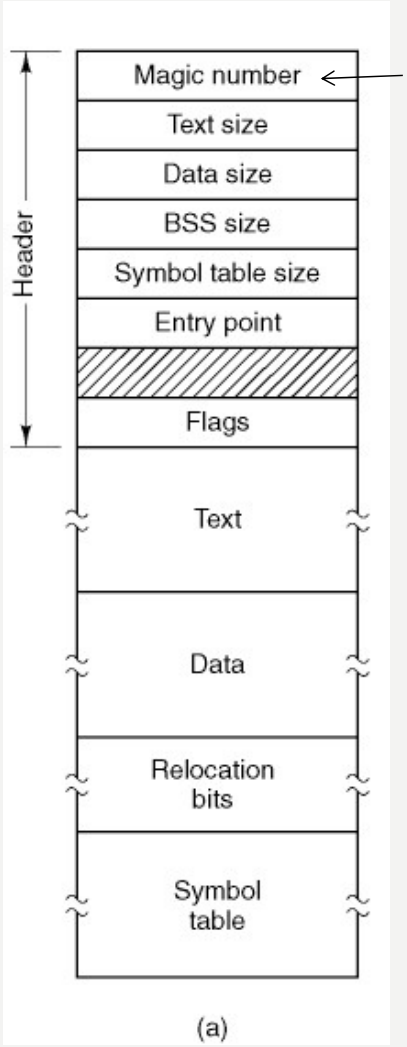
Extension	Meaning
file.bak	Backup file
file.c	C source program
file.gif	Graphical Interchange Format image
file.html	World Wide Web HyperText Markup Language document
file.iso	ISO image of a CD-ROM (for burning to CD)
file.jpg	Still picture encoded with the JPEG standard
file.mp3	Music encoded in MPEG layer 3 audio format
file.mpg	Movie encoded with the MPEG standard
file.o	Object file (compiler output, not yet linked)
file.pdf	Portable Document Format file
file.ps	PostScript file
file.tex	Input for the TEX formatting program
file.txt	General text file
file.zip	Compressed archive

Figure 5-1. Some typical file extensions.

EXECUTABLE FILE

Figure 5-3. (a) An executable file.

Know type from magic number



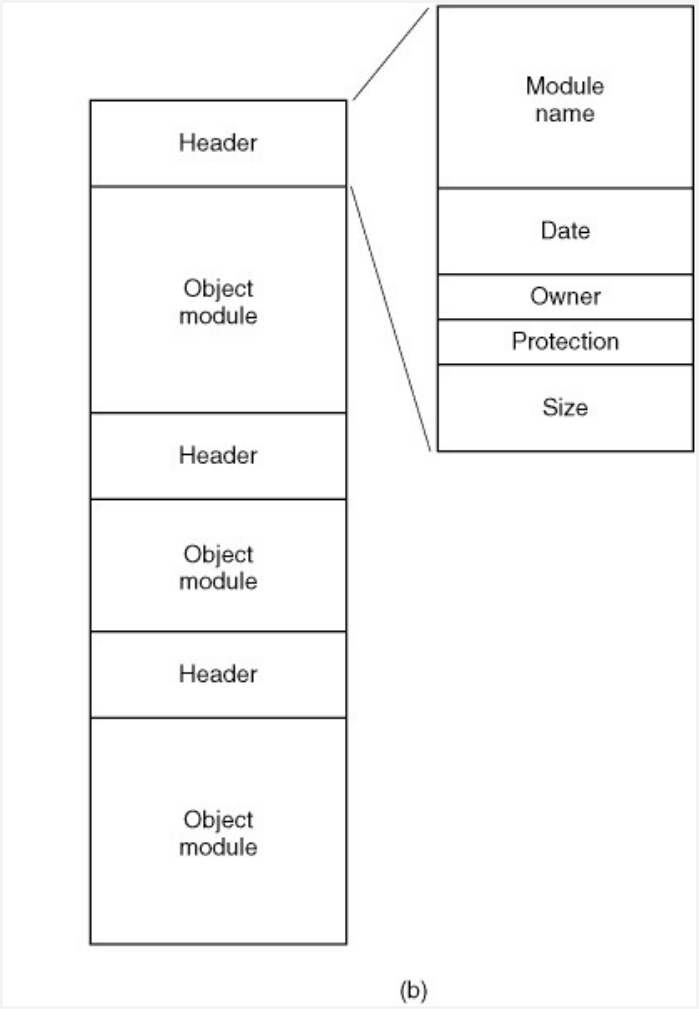
Some Magic Numbers

Executable and Linkable Format (ELF):	.ELF
Compiled Java class file:	CAFEBABE
GIF:	GIF89a or GIF87a
Shell script (optional):	#!<shell path>
Postscript:	%!PS
PDF:	%PDF
DOS executables:	MZ
Zip files:	PK
RAR archive:	Rar!..

ARCHIVE FILE

Figure 5-3. … (b) An archive.

Can tell by magic number and by extension



FILE CONTROL BLOCK

- FCB is concept (like Process Control Block)
  - May be implemented in distributed fashion
- FCB holds file attributes and storage information
- Directory holds FCB or means to get FCB

FILE ATTRIBUTES (1)

Attribute	Meaning
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	ID of the person who created the file
Owner	Current owner
Read-only flag	0 for read/write; 1 for read only
Hidden flag	0 for normal; 1 for do not display in listings
System flag	0 for normal files; 1 for system file
Archive flag	0 for has been backed up; 1 for needs to be backed up
ASCII/binary flag	0 for ASCII file; 1 for binary file
Random access flag	0 for sequential access only; 1 for random access
...	...

Figure 5-4. Some possible file attributes.

FILE ATTRIBUTES (2)

Question: where do we store file attributes?

. . .

Attribute	Meaning
Temporary flag	0 for normal; 1 for delete file on process exit
Lock flags	0 for unlocked; nonzero for locked
Record length	Number of bytes in a record
Key position	Offset of the key within each record
Key length	Number of bytes in the key field
Creation time	Date and time the file was created
Time of last access	Date and time the file was last accessed
Time of last change	Date and time the file has last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

Figure 5-4. Some possible file attributes.

FILE OPERATIONS

1. Create
2. Delete
3. Open
4. Close
5. Read
6. Write
7. Append
8. Seek
9. Get attributes
10. Set Attributes
11. Rename
12. Lock

FILE ACCESS

- Sequential Access
  - Bytes/records of file are accessed in linear order
  - May be able to "rewind" to first byte/record
  - Models tape storage and pipes
- Random Access
  - Bytes/records may be accessed in any order
  - Interface supports way to specify where to access
    - Read/write may include location (no cursor)
    - Seek may move cursor (current location)

POSIX SYSTEM CALLS - FILES

What *is* a file descriptor anyway?

File management

Call	Description
fd = open(file, how, ...)	Open a file for reading, writing, or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

Figure 1-18. Some of the major POSIX system calls for file access.

The return code *s* is −1 if an error has occurred. The return codes are: *fd* is a file descriptor, *n* is a byte count, *position* is an offset within the file, and *s* is the status.

EXAMPLE PROGRAM USING FILE SYSTEM CALLS

```
/* File copy program. Error checking and reporting is minimal. */

#include <sys/types.h>                  /* include necessary header files */
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[]);       /* ANSI prototype */

#define BUF_SIZE 4096                   /* use a buffer size of 4096 bytes */
#define OUTPUT_MODE 0700                /* protection bits for output file */

int main(int argc, char *argv[])
{
        int in_fd, out_fd, rd_count, wt_count;
        char buffer[BUF_SIZE];

        if (argc != 3) exit(1);         /* syntax error if argc is not 3 */

        /* Open the input file and create the output file */
```

Figure 4-5. A simple program to copy a file.

EXAMPLE PROGRAM USING FILE SYSTEM CALLS

```
if (argc != 3) exit(1);                              /* syntax error if argc is not 3 */

/* Open the input file and create the output file */
in_fd = open(argv[1], O_RDONLY);                     /* open the source file */
if (in_fd < 0) exit(2);                              /* if it cannot be opened, exit */
out_fd = creat(argv[2], OUTPUT_MODE);                /* create the destination file */
if (out_fd < 0) exit(3);                             /* if it cannot be created, exit */

/* Copy loop */
while (TRUE) {
        rd_count = read(in_fd, buffer, BUF_SIZE);    /* read a block of data */
if (rd_count <= 0) break;                            /* if end of file or error, exit loop */
        wt_count = write(out_fd, buffer, rd_count);  /* write data */
```

Figure 4-5. A simple program to copy a file.

EXAMPLE PROGRAM USING FILE SYSTEM CALLS

```
        /* Copy loop */
        while (TRUE) {
                rd_count = read(in_fd, buffer, BUF_SIZE); /* read a block of data */
        if (rd_count <= 0) break;                         /* if end of file or error, exit loop */
                wt_count = write(out_fd, buffer, rd_count); /* write data */
                if (wt_count <= 0) exit(4);               /* wt_count <= 0 is an error */
        }

        /* Close the files */
        close(in_fd);
        close(out_fd);
        if (rd_count == 0)                                /* no error on last read */
                exit(0);
        else
                exit(5);                                  /* error on last read */
}
```

Figure 4-5. A simple program to copy a file.

SUMMARY

- Need for files
  - Definition
- File Structures
- File Types
  - How can we tell file type?
- File Attributes
- File Operations
- System call for files
- Next: Directories