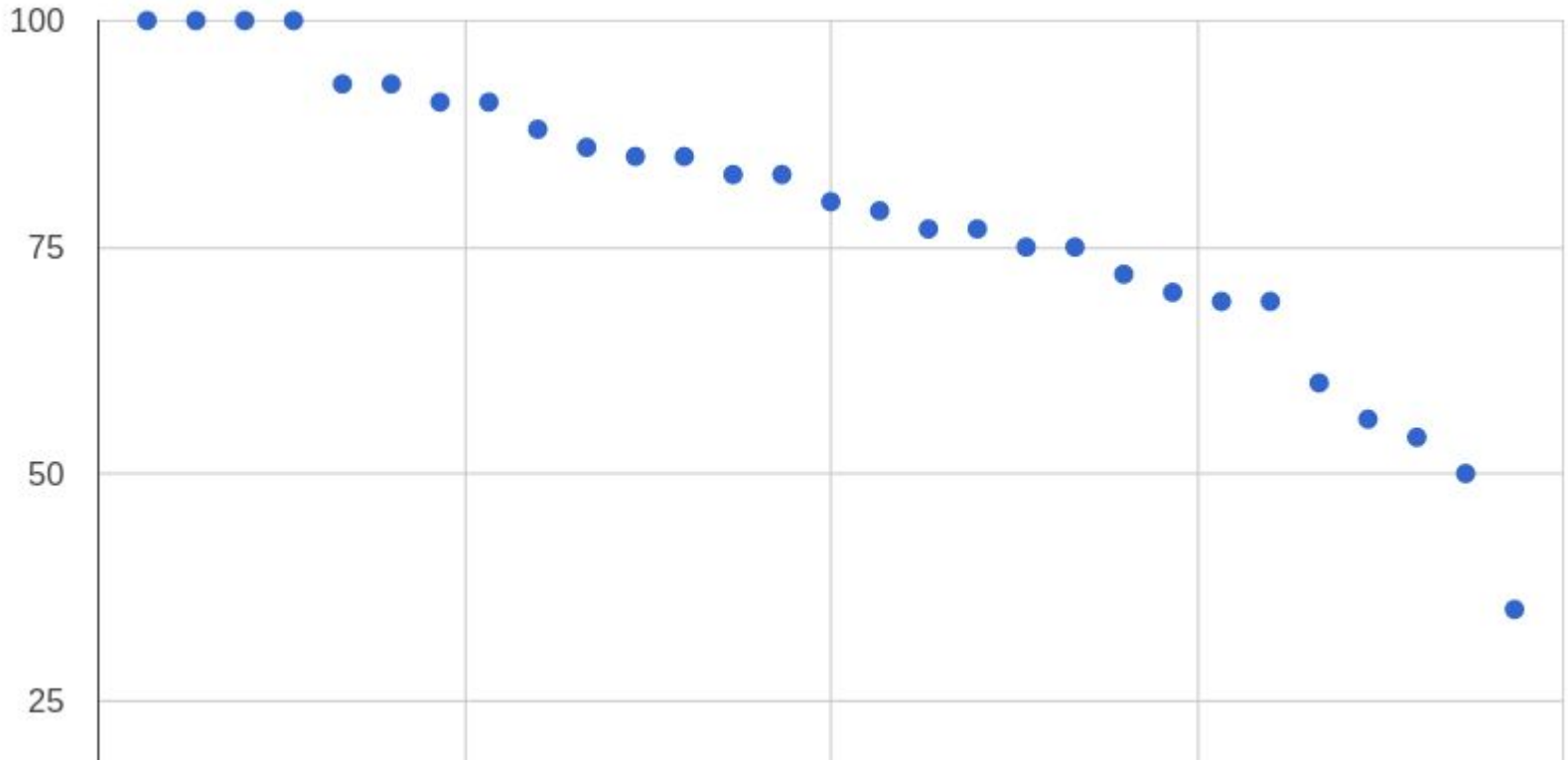
Files and Directories

Module 15

Exam 3 - Average = 78



Storage needs of processes

Processes need to store information for many reasons

Beyond memory, we require storage of information that meets the following requirements:

1. Possible to store very large amounts of data (larger than would fit in virtual address space)
2. Data must survive process termination
3. Multiple processes may access data simultaneously

Hardware

- Tape drives, CD/DVD Drives
  - Massive storage capabilities
  - Unbearably slow (use for backups)
- Magnetic Disks
  - Very large storage, adequate speed
  - Excellent for general purpose systems
- Solid State
  - Large storage, great speed
  - Better for laptops (moving parts), high performance

Hardware Models

- For now… we will model *all* storage devices the same way: a linear array of fixed-size blocks
- Operations:
  - read block *k*
  - write block *k*
- There are more -but these are the “primary” operations - from it we build a storage system

Huge Sequence of Bytes … to files

Modelling the entire disk as an array has problems:

- How do you find information?
- How do you limit access (per user)
- How do you know which blocks are free?

To solve this, we create another **abstraction**: a file

A file is a logical collection of blocks represent a unit of information.

File Abstraction

- Much like processes and pages - files are simply organizational structures
- Processes may create, read, update, and delete files
- File structure, along with operations process can perform, are called a ***file system***. It is the OS's responsibility.

Files: Naming

Filename rules and conventions vary from OS to OS

- **Lengths:** Typically 255 characters - but could be more or less
- **Characters:** Typically some special characters allowed, but not all
- **Case-sensitivity:** Unix is case-sensitive, older version of Windows are not.

Filesystems - examples

- Windows 95/98: MS-DOS filesystem (FAT-16)
- Windows 98 introduced FAT-32
- Windows XP-Windows 10 support FAT-32, but use NTFS by default
- UNIX-based systems have many filesystems:
- UFD, ext2/ext3/ext4, HFS (for OS X)

We'll discuss some of these along the way - but we'll largely talk about general design, rather than specifics

Files: Extensions

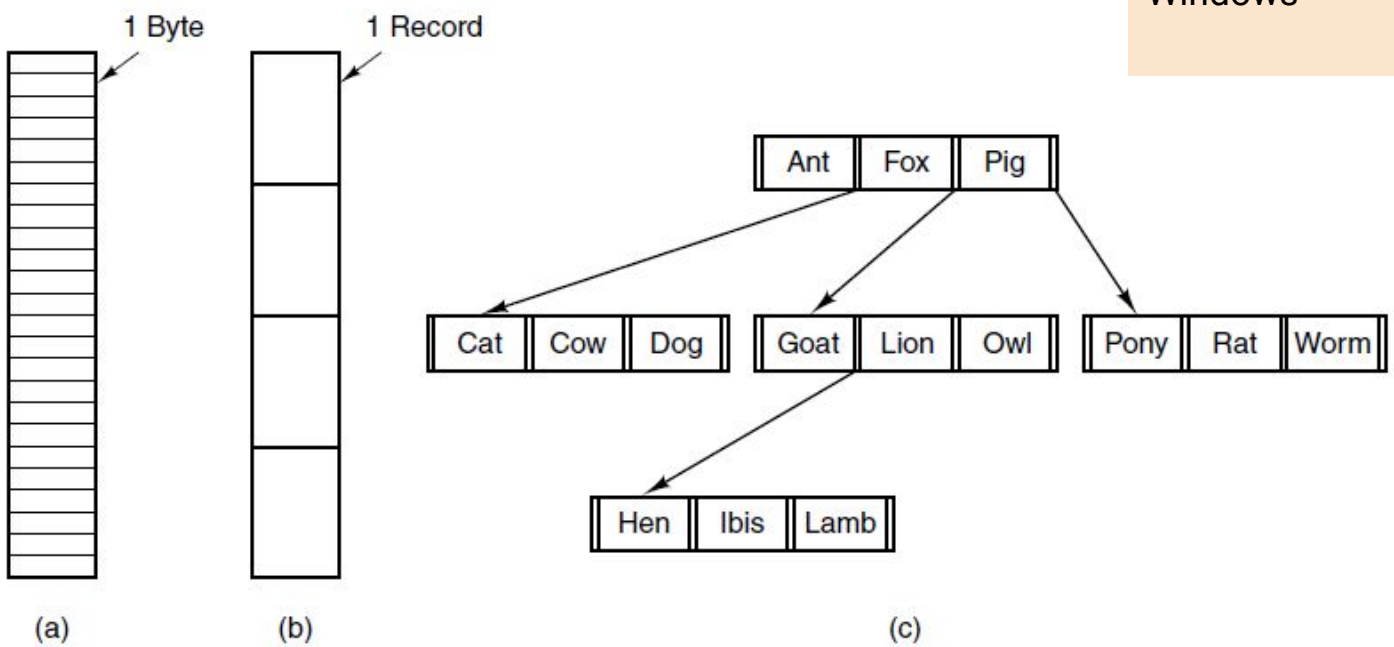
- Most filenames will allow user (process) to use two-part names
  - filename.extension
- On UNIX, these extension are not “known” to the operating system - they are reminders to the user
- On Windows, each extension is registered to an “owning” program - which is launched when the file is “run”

Common extensions

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

File Structure

Type A is most flexible, and is used by UNIX and Windows



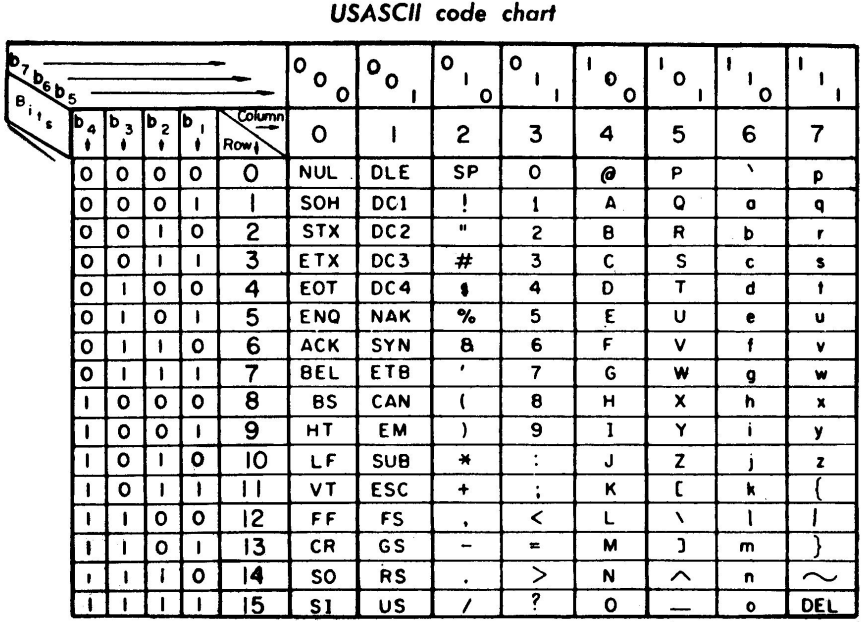
Three kinds of files. (a) Byte sequence. (b) Record sequence. (c) Tree.

Types of Files (not extensions!)

- **Directories**: Believe or not… directories **<u>are</u>** files. They have to be!
- **Regular Files**: Data files - containing user and/or system information
  - Ascii Text
  - Binary (not Ascii Text)
- **Character special files:** Files used to perform i/o for serial devices (terminals, printers, networks)
- **Block special files:** Used to model disk I/o

Ascii or Binary

- Ascii text files consist of basic lines of text
- Lines are delimited by Line feed (LF) and/or carriage return (CR)
- They can be handled by virtually any system or program



USASCII code chart

b_7 b_6 b_5 →	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b_4 b_3 b_2 b_1 ↓ (Bits; Column →; Row ↓)	0	1	2	3	4	5	6	7
0 0 0 0	0	NUL	DLE	SP	0	@	P	`	p
0 0 0 1	1	SOH	DC1	!	1	A	Q	a	q
0 0 1 0	2	STX	DC2	"	2	B	R	b	r
0 0 1 1	3	ETX	DC3	#	3	C	S	c	s
0 1 0 0	4	EOT	DC4	$	4	D	T	d	t
0 1 0 1	5	ENQ	NAK	%	5	E	U	e	u
0 1 1 0	6	ACK	SYN	&	6	F	V	f	v
0 1 1 1	7	BEL	ETB	'	7	G	W	g	w
1 0 0 0	8	BS	CAN	(	8	H	X	h	x
1 0 0 1	9	HT	EM	)	9	I	Y	i	y
1 0 1 0	10	LF	SUB	*	:	J	Z	j	z
1 0 1 1	11	VT	ESC	+	;	K	[	k	{
1 1 0 0	12	FF	FS	,	<	L	\	l	|
1 1 0 1	13	CR	GS	-	=	M	]	m	}
1 1 1 0	14	SO	RS	.	>	N	^	n	~
1 1 1 1	15	SI	US	/	?	O	_	o	DEL

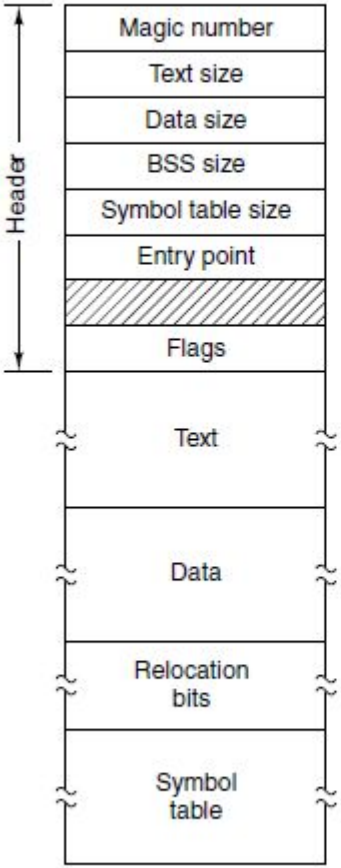
ASCII or Binary

- Binary files consist of bytes of data (just like ASCII) but the format is “unknown”
- If you try to open a binary file in a text editor (which expects ASCII) - you get incomprehensible results!
- The structure is known by the “owning” process however!

Executable files

Some **regular files** are also executable:

Executable files are *binary*, but they have headers (front-matter) that the OS can decipher



File Access

- Due to device constraints, there was a time where all read/write was ***sequential***.
- With magnetic disks (and solid state), all file-systems support ***random access***
  - New operation: seek

Metadata - File Attributes

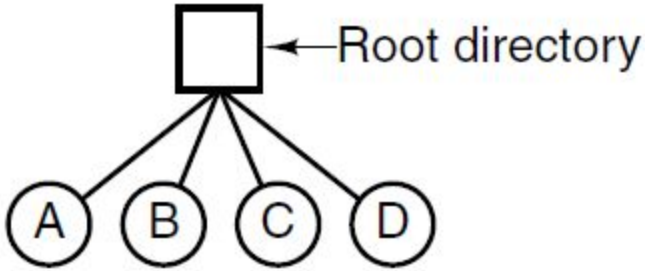
Attribute	Meaning
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	ID of the person who created the file
Owner	Current owner
Read-only flag	0 for read/write; 1 for read only
Hidden flag	0 for normal; 1 for do not display in listings
System flag	0 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
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 was last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

File attributes are actually often stored in the **directory in which they are contained**.

Full Operation Set

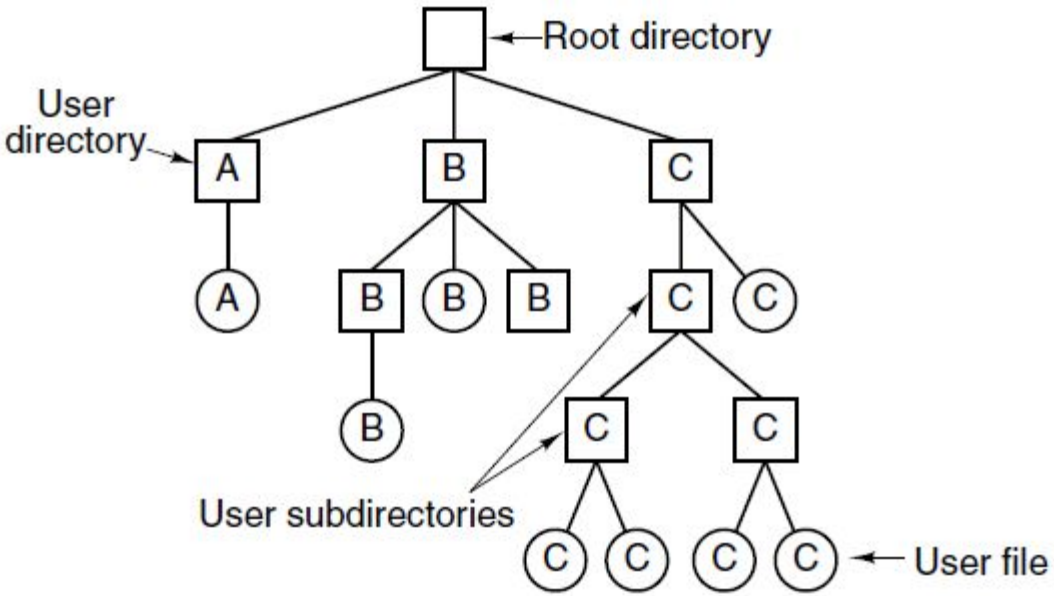
Create
Delete
Open
Close
Read
Write
Append
Seek
Get Attributes
Set Attributes
Rename

Directory Structures: Single-Level



What practical problems does this present?

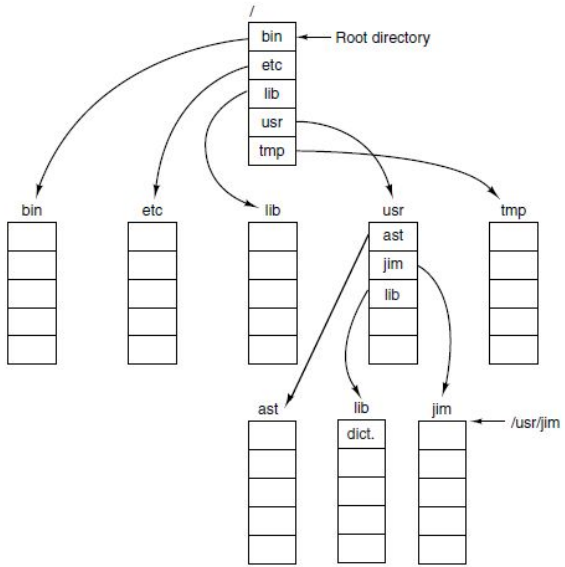
Directory Structure: Hierarchical



Mod ern systems all are based on this approach

Paths

A path represents a filename and it's **location** within the hierarchy



Unix, the root is /

In Windows, it's the drive letter (C:, D:, etc.)

Paths

Absolute Path: Starts with the root symbol and lists each directory

Relative Path: Starts with folder / file name

`.` represents “current directory”

`..` represents parent directory

Directory Operations

Create
Delete

New directories only have . and .. in them

Only empty directories may be deleted

Opendir
Closedir
Readdir

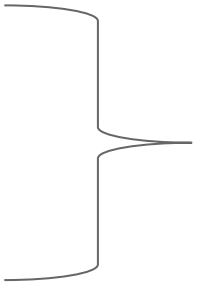
Directories can be read byte-for-byte, but then we'd need to know the internal structure. Instead, the OS will likely provide an API for reading file entries from the directory file

Directory Operations

Rename:

Link

Unlink



We may specify a new path name for a file, so the file entry actually links to another location

Up next...

We've basically seen the *user's* view of the filesystem

We will now focus on the implementation - please read 4.3!