Chap. 13) File System Interface

경희대학교 컴퓨터공학과

조 진 성

Basic Concept

- Requirements for long-term information storage
  - ✓ It must be possible to store a very large amount of information
  - ✓ The information must survive the termination of the process using it
  - ✓ Multiple processes must be able to access the information concurrently

- File system
  - ✓ Implement an abstraction for secondary storage (files)
  - ✓ Organize files logically (directories)
  - ✓ Permit sharing of data between processes, people, and machines (sharing)
  - ✓ Protect data from unwanted access (protection)

Files

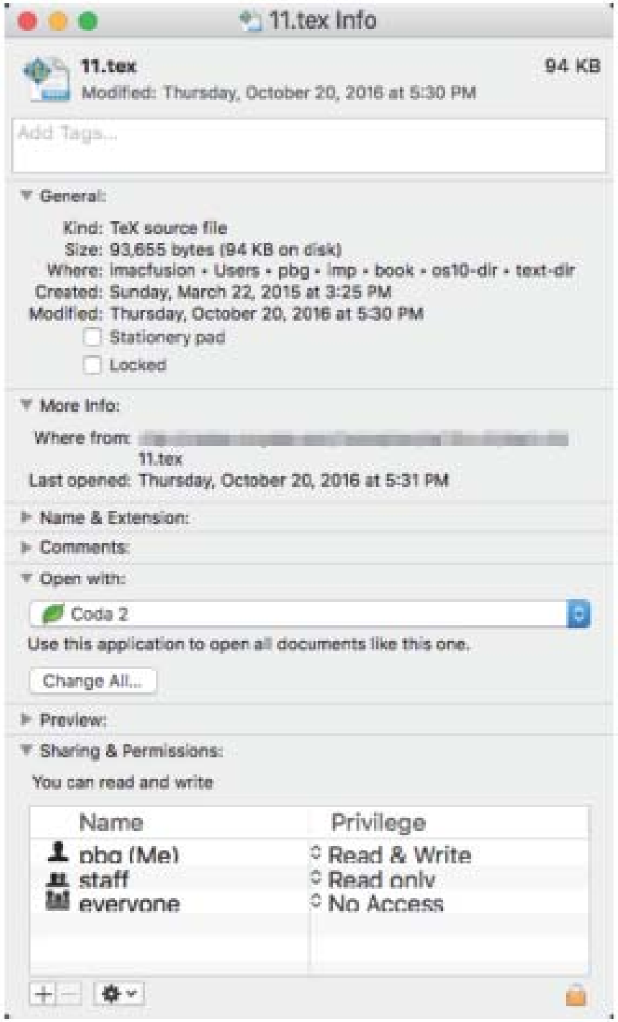
- File
  - ✓ A named collection of related information that is recorded on secondary storage
    - persistent through power failures and system reboots
  - ✓ OS provides a uniform logical view of information storage via files

- File structures
  - ✓ Flat: byte sequence
  - ✓ Structured:
    - Lines
    - Fixed length records
    - Variable length records

File Attributes

- Attributes or metadata

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 has last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to



11.tex Info

11.tex 94 KB

Modified: Thursday, October 20, 2016 at 5:30 PM

Add Tags...

General:

Kind: TeX source file
Size: 93,655 bytes (94 KB on disk)
Where: imacfusion ▸ Users ▸ pbg ▸ imp ▸ book ▸ os10-dir ▸ text-dir
Created: Sunday, March 22, 2015 at 3:25 PM
Modified: Thursday, October 20, 2016 at 5:30 PM

□ Stationery pad
□ Locked

More Info:

Where from: [redacted]
11.tex
Last opened: Thursday, October 20, 2016 at 5:31 PM

Name & Extension:

Comments:

Open with:

Coda 2

Use this application to open all documents like this one.

Change All...

Preview:

Sharing & Permissions:

You can read and write

Name	Privilege
pbg (Me)	Read & Write
staff	Read only
everyone	No Access

File Operations

- Unix/Linux system calls

```
int creat (const char *pathname, mode_t mode);
int open (const char *pathname, int flags, mode_t mode);
int close (int fd);
ssize_t read (int fd, void *buf, size_t count);
ssize_t write (int fd, const void *buf, size_t count);
off_t lseek (int fd, off_t offset, int whence);
int stat (const char *pathname, struct stat *buf);
int chmod (const char *pathname, mode_t mode);
int chown (const char *pathname, uid_t owner, gid_t grp);
int flock (int fd, int operation);
int fcntl (int fd, int cmd, long arg);
```

File Types

- Files may have types
  - ✓ Understood by file systems
    - device, directory, symbolic link, etc.
  - ✓ Understood by other parts of OS or runtime libraries
    - executable, dll, source code, object code, text, etc.
  - ✓ Understood by application programs
    - jpg, mpg, avi, mp3, etc.

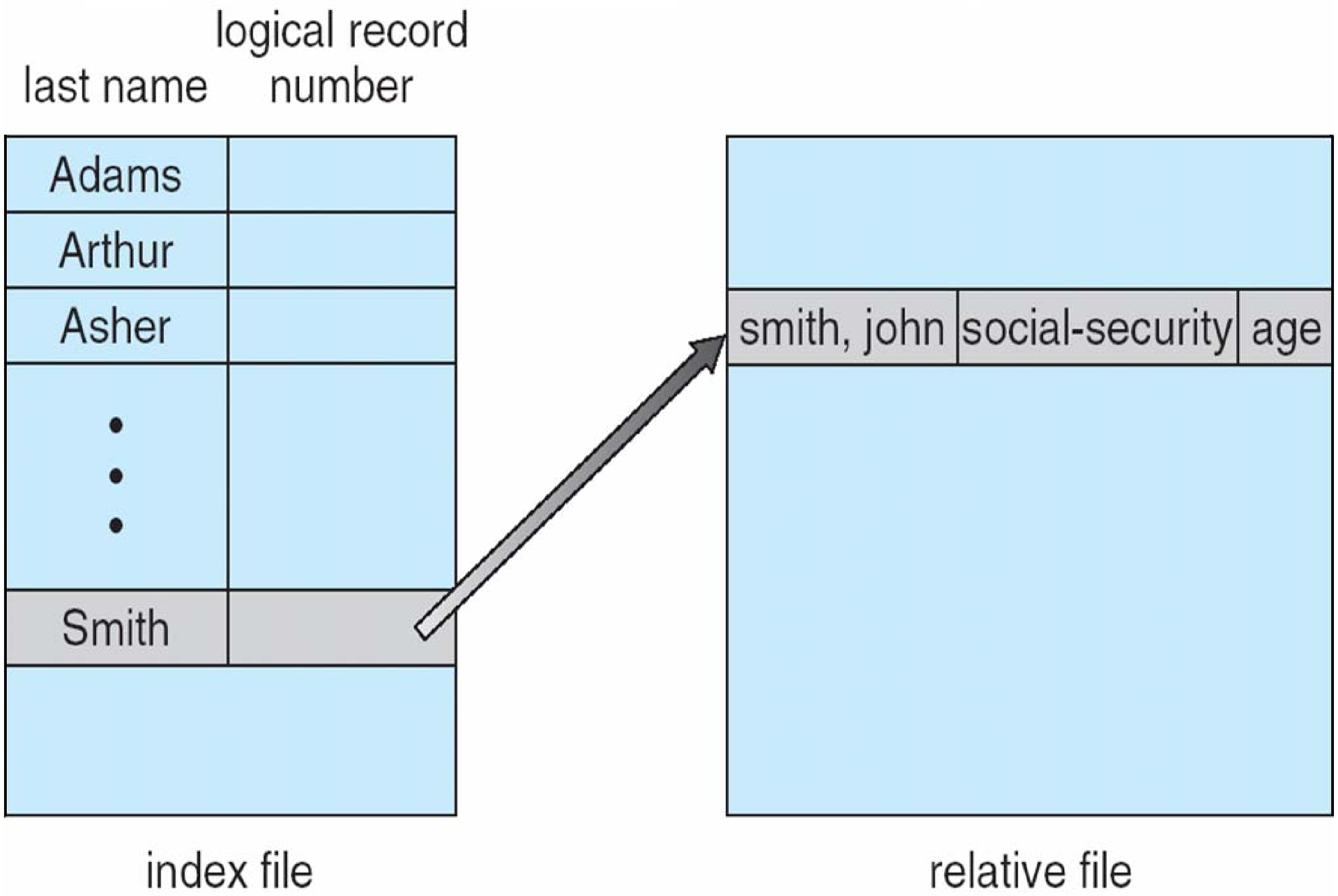
- Encoding file types
  - ✓ Windows encodes type in name
    - .com, .exe, .bat, .dll, .jpg, .avi, .mp3, etc.
  - ✓ Unix encodes type in contents
    - magic numbers (e.g., 0xcafebabe for Java class files)
    - initial characters (e.g., #! for shell scripts)

File Access

- Some file systems provide different access methods that specify different ways for accessing data in a file.

- Sequential access
  - ✓ read bytes one at a time, in order
- Direct access
  - ✓ random access given block/byte number
- Record access
  - ✓ File is an array of fixed- or variable-length records, read/written sequentially or randomly by record #
- Index access
  - ✓ File system contains an index to a particular field of each record in a file, reads specify a value for that field and the system finds the records via the index (DBs)

Example of Index and Relative Files



Directories

- Directories
  - ✓ For users, they provide a structured way to organize files
  - ✓ For the file system, they provide a convenient naming interface that allows the implementation to separate logical file organization from physical file placement on the disk

- A hierarchical directory system
  - ✓ Most file systems support multi-level directories
  - ✓ Most file systems support the notion of a current directory (or working directory)
    - Relative names specified with respect to current directory
    - Absolute names start from the root of directory tree

Directory Internals

- A directory is …
  - ✓ typically just a file that happens to contain special metadata
    - Only need to manage one kind of secondary storage unit
  - ✓ directory = list of (file name, file attributes)
  - ✓ attributes include such things as:
    - size, protection, creation time, access time,
    - location on disk, etc.
  - ✓ usually unordered (effectively random)
    - Entries usually sorted by program that reads directory

Directory Operations

- Unix operations
  - ✓ Directories implemented in files
    - Use file operations to manipulate directories
  - ✓ C runtime libraries provides a higher-level abstraction for reading directories
    - DIR *opendir (const char *name);
    - struct dirent *readdir (DIR *dir);
    - void seekdir (DIR *dir, off_t offset);
    - int closedir (DIR *dir);
  - ✓ Other directory-related system calls
    - int rename (const char *oldpath, const char *newpath);
    - int link (const char *oldpath, const char *newpath);
    - int unlink (const char *pathname);

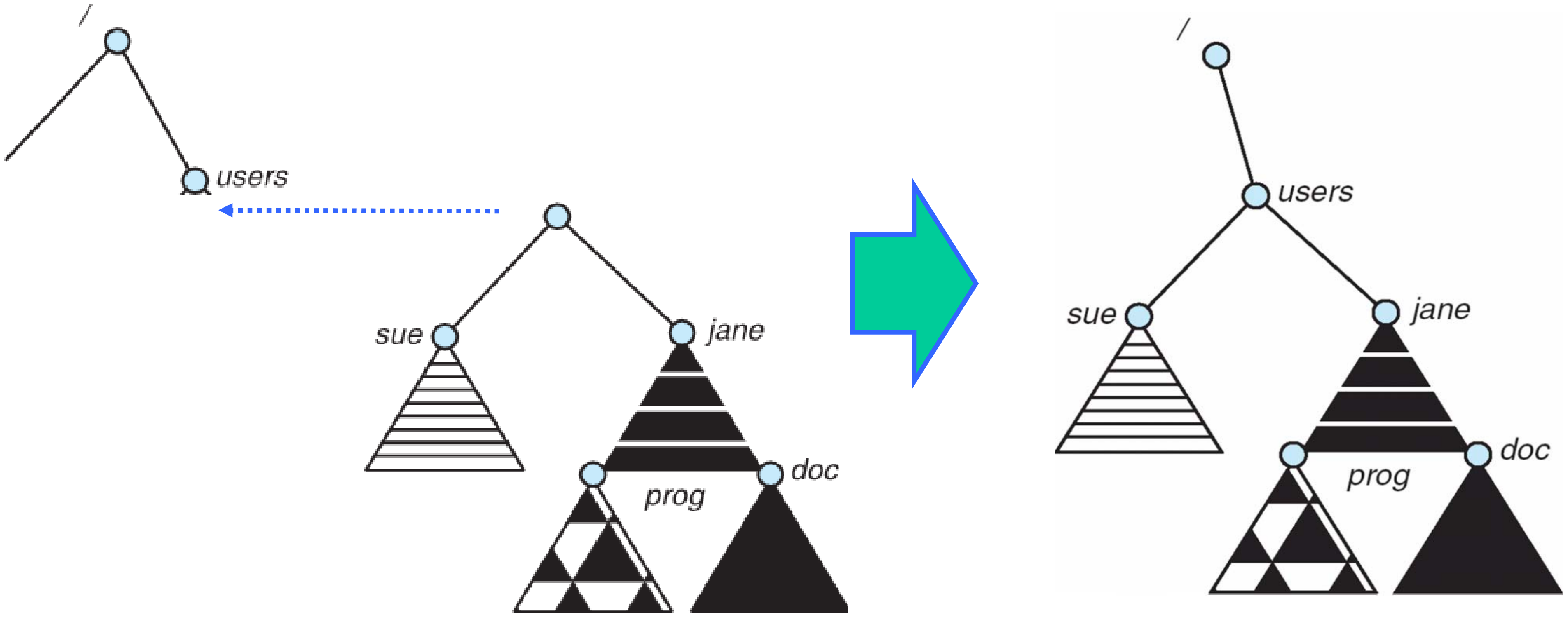
Pathname Translation

- open(“/a/b/c”, …)
  - ✓ Open directory “/” (well known, can always find)
  - ✓ Search the directory for “a”, get location of “a”
  - ✓ Open directory “a”, search for “b”, get location of “b”
  - ✓ Open directory “b”, search for “c”, get location of “c”
  - ✓ Open file “c”
  - ✓ (Of course, permissions are checked at each step)

- System spends a lot of time walking down directory paths
  - ✓ This is why open is separate from read/write
  - ✓ OS will cache prefix lookups to enhance performance
    - /a/b, /a/bb, /a/bbb, etc. all share the “/a” prefix

File System Mounting

- *Example*
  - ✓ Windows: to drive letters (e.g., C:\, D:\, …)
  - ✓ Unix/Linux: to an existing empty directory (= mount point)



File Sharing – Remote File Systems

- Client-server model allows clients to mount remote file systems from servers
  - ✓ Server can serve multiple clients
  - ✓ Client and user-on-client identification is insecure or complicated
  - ✓ NFS is standard UNIX client-server file sharing protocol
  - ✓ CIFS is standard Windows protocol
- Distributed Information Systems (distributed naming services) such as LDAP, DNS, NIS, Active Directory implement unified access to information needed for remote computing

Protection

- Representing protection
  - ✓ Access control lists (ACLs)
    - For each object, keep list of subjects and their allowed actions
  - ✓ Capabilities
    - For each subject, keep list of objects and their allowed actions

			objects	
		/etc/passwd	/home/hong	/home/guest
subjects	root	rw	rw	rw
	hong	r	rw	r
Capability	guest	-	-	r

ACL

Protection

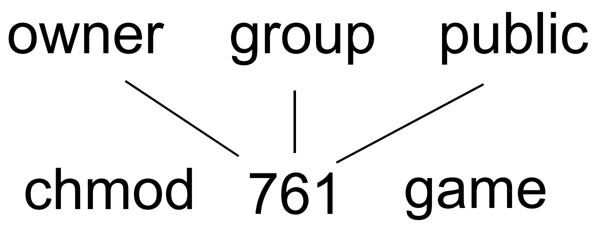
- ACLs vs. Capabilities
  - ✓ Two approaches differ only in how the table is represented
  - ✓ Capabilities are easy to transfer
    - They are like keys; can hand them off
    - They make sharing easy
  - ✓ In practice, ACLs are easier to manage
    - Object-centric, easy to grant and revoke
    - To revoke capabilities, need to keep track of all subjects that have the capability – hard to do, given that subjects can hand off capabilities
  - ✓ ACLs grow large when object is heavily shared
    - Can simplify by using “groups”
    - Additional benefit: change group membership affects all objects that have this group in its ACL

Access Lists in Unix/Linux

- Mode of access: read, write, execute
- Three classes of users (**u**ser / **g**roup / **o**thers)

	RWX		
a) **owner access**	7	⇒	1 1 1
	RWX		
b) **group access**	6	⇒	1 1 0
	RWX		
c) **public access**	1	⇒	0 0 1

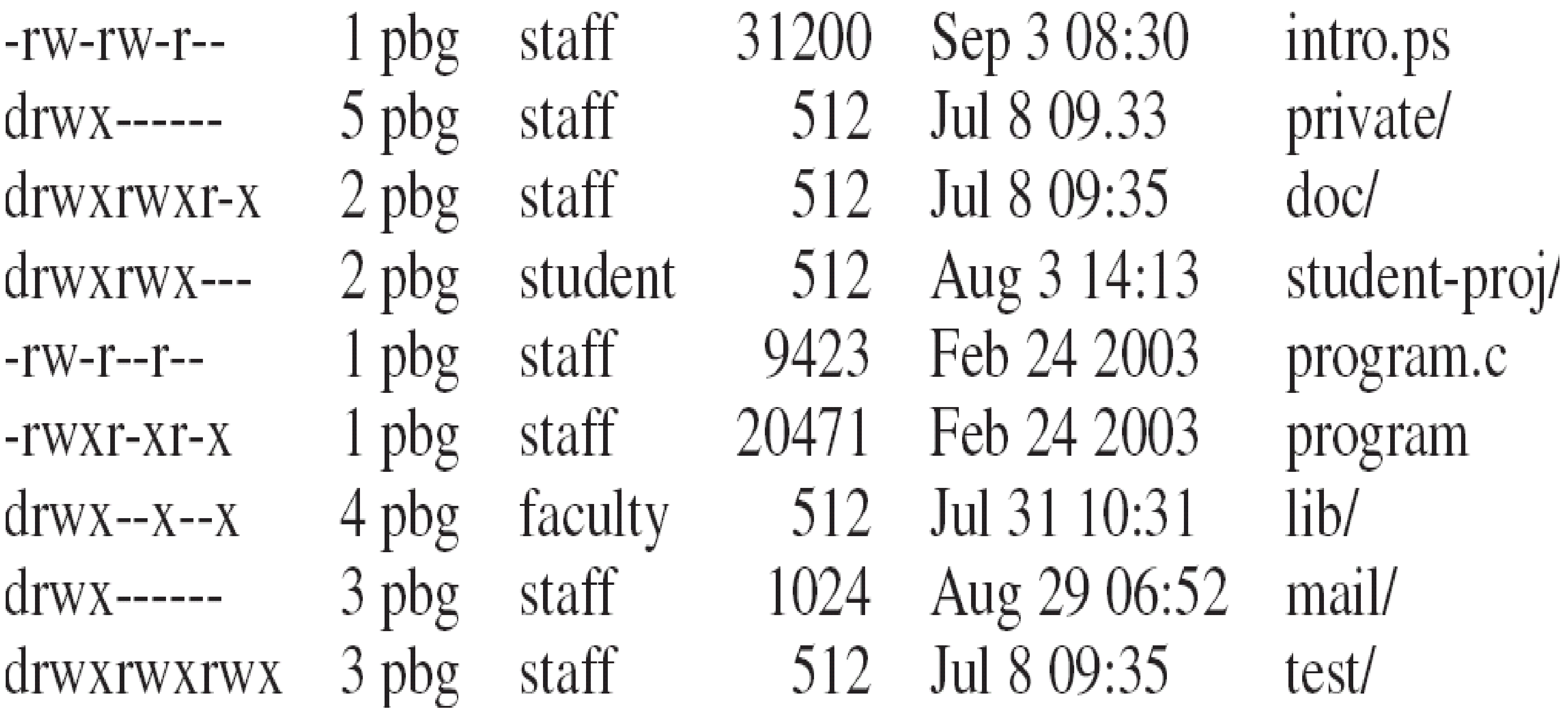
- For a particular file (say *game*) or subdirectory, define an appropriate access


owner group public

chmod 761 game

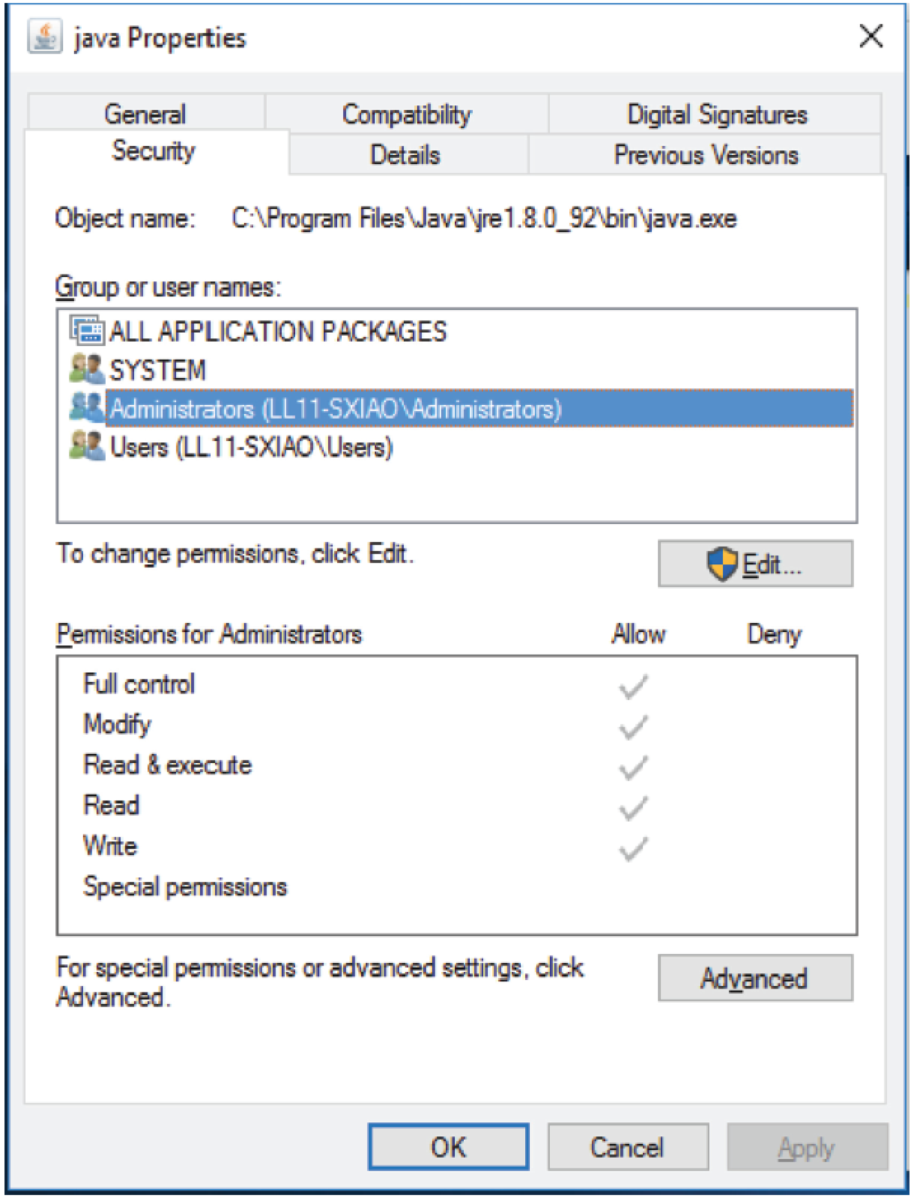
Access Lists in Unix/Linux

- $ ls –l



```
-rw-rw-r--    1 pbg   staff     31200   Sep 3 08:30    intro.ps
drwx------    5 pbg   staff       512   Jul 8 09.33    private/
drwxrwxr-x    2 pbg   staff       512   Jul 8 09:35    doc/
drwxrwx---    2 pbg   student     512   Aug 3 14:13    student-proj/
-rw-r--r--    1 pbg   staff      9423   Feb 24 2003    program.c
-rwxr-xr-x    1 pbg   staff     20471   Feb 24 2003    program
drwx--x--x    4 pbg   faculty     512   Jul 31 10:31   lib/
drwx------    3 pbg   staff      1024   Aug 29 06:52   mail/
drwxrwxrwx    3 pbg   staff       512   Jul 8 09:35    test/
```

Access Lists in Windows



Object name: C:\Program Files\Java\jre1.8.0_92\bin\java.exe

Group or user names:

- ALL APPLICATION PACKAGES
- SYSTEM
- Administrators (LL11-SXIAO\Administrators)
- Users (LL11-SXIAO\Users)

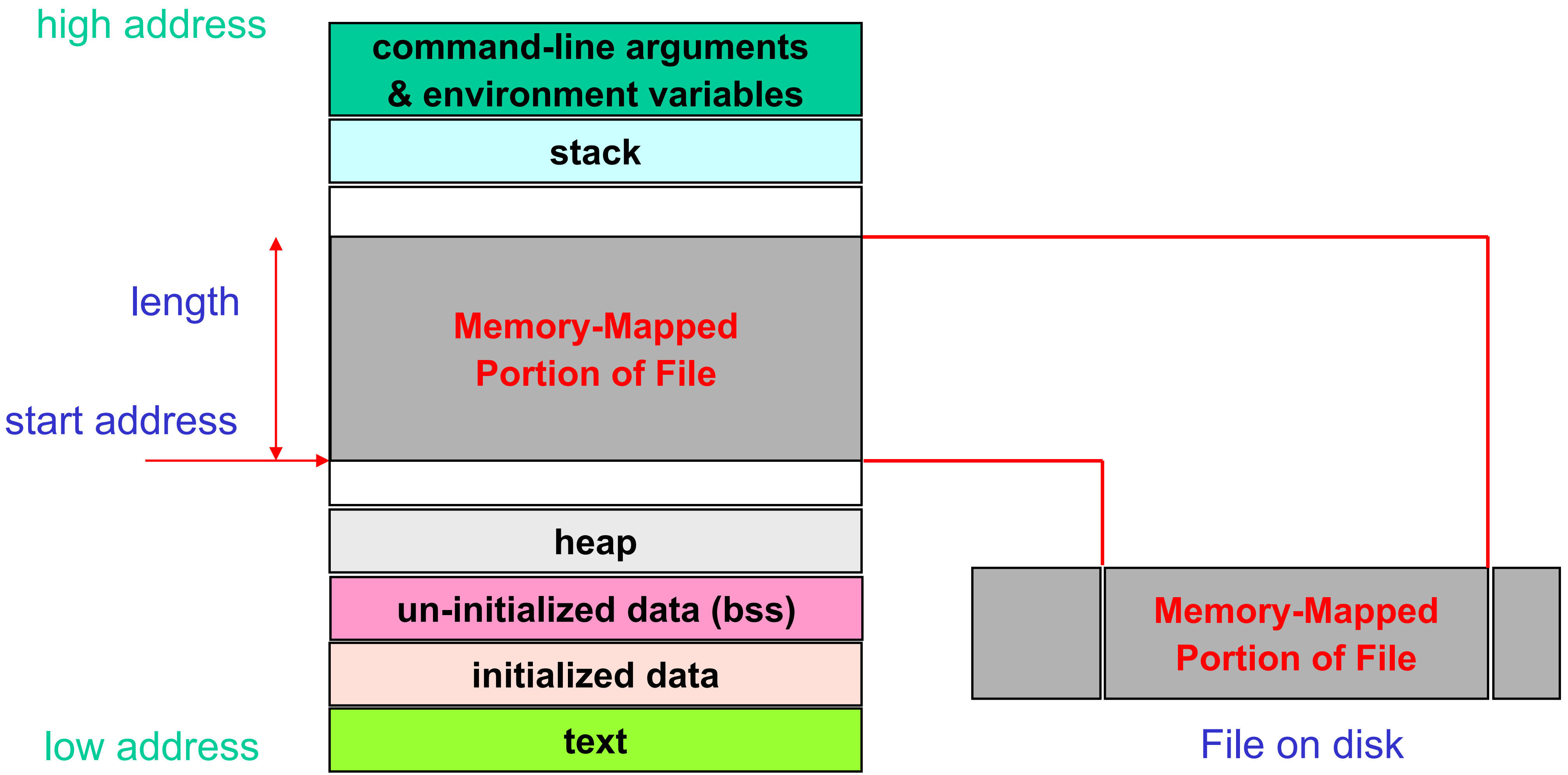
To change permissions, click Edit.

Permissions for Administrators	Allow	Deny
Full control	✓	
Modify	✓	
Read & execute	✓	
Read	✓	
Write	✓	
Special permissions		

For special permissions or advanced settings, click Advanced.

Memory-Mapped File

- Map a file on disk into a buffer in memory (**`mmap()`** system call)
  - ✓ perform I/O without using **`read`** or **`write`**


high address
command-line arguments & environment variables
stack
length
Memory-Mapped Portion of File
start address
heap
un-initialized data (bss)
initialized data
text
low address
Memory-Mapped Portion of File
File on disk