OPERATING SYSTEMS

Storage Management

Suresh Jamadagni
Department of Computer Science

OPERATING SYSTEMS

File System

Suresh Jamadagni
Department of Computer Science

OPERATING SYSTEMS

Slides Credits for all the PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9th edition 2013 and some slides from 10th edition 2018
2. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9th edition 2018
3. Some presentation transcripts from A. Frank – P. Weisberg
4. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau

OPERATING SYSTEMS

File Concept

- Computers can store information on various storage media, such as magnetic disks, magnetic tapes, and optical disks.
- File is logical storage where as disks are physical storage unit
- What is a File?
  - collection of related information
  - Data cannot to be written to disk unless they are in files
  - Data can be Numeric, alphabetic, alphanumeric, binary
- A file is a sequence of bits, bytes, lines, or records, the meaning of which is defined by the file's creator and user
- Contents of file can be source or executable programs, numeric or text data, photos, music, video, and so on.

File Attributes

Files are named and are referred by their names by the human users.

Attributes of files

- **Name** – only information kept in human-readable form
- **Identifier** – unique tag (number) identifies file within file system
- **Type** – needed for systems that support different types
- **Location** – pointer to file location on device
- **Size** – current file size
- **Protection** – controls who can do reading, writing, executing
- **Time, date, and user identification** – data for protection, security, and usage monitoring
- Information about files are kept in the directory structure, which is maintained on the disk.

OPERATING SYSTEMS

File Attributes

- The information about all files is kept in the directory structure, which also resides on secondary storage.
- Directory entry consists of the file's name and its uniqueand its unique identifier.
- Identifier in turn locates the file attributes.

```
File: g.c
  Size: 218           Blocks: 0          IO Block: 4096   regular file
Device: 2h/2d   Inode: 337699720577240  Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2020-09-15 15:18:57.338585500 +0530
Modify: 2021-03-20 10:40:37.416576700 +0530
Change: 2021-03-20 10:40:37.416576700 +0530
```

File Operations

File is an abstract data type.

File Operations

- **Create**
- **Write** – Use the system call to write to a file. A write pointer points to the location in the file, where the next write is to take place.
- **Read** – – Use the system call to write to a file. A read pointer points to the location in the file from where the next read is to take place
- **Reposition within file -** Repositioning within a file need not involve any actual I/O. It is done with system call lseek. Repositioning is also called seek to location in a file
- **Delete:** Use the system call to delete/remove a file.
  - File to be deleted is searched in the directory.
  - Release the memory allocated after deletion

File Operations

- **Truncate:** The user may want to erase the contents of a file but keep its attributes.
  - Instead of deleting the file and then recreating it, use the function that sets the size of file to zero.
- **Appending** to the end of a existing file
- ***Open a file*** – use a system call open()
  - **Open-file table**: tracks open files
  - File pointer: pointer to last read/write location, per process that has the file open
  - **File-open count**: counter of number of times a file is open – to allow removal of data from open-file table when last processes closes it
- ***Close:*** *When a file is actively not used close it.*

OPERATING SYSTEMS

File Operations

- Several pieces of information are associated with an open file.
  - **File pointer**
  - **File-open count.**
  - **Disk location of the file.**
  - **Access rights**

Open Files

- Provided by some operating systems and file systems
  - Similar to reader-writer locks
  - **Shared lock** similar to reader lock – several processes can acquire concurrently
  - **Exclusive lock** similar to writer lock
- OS mediates access to a file
- Mandatory or advisory:
  - **Mandatory** – access is denied depending on locks held and requested
  - **Advisory** – processes can find status of locks and decide what to do

OPERATING SYSTEMS

File Types – Name, Extension

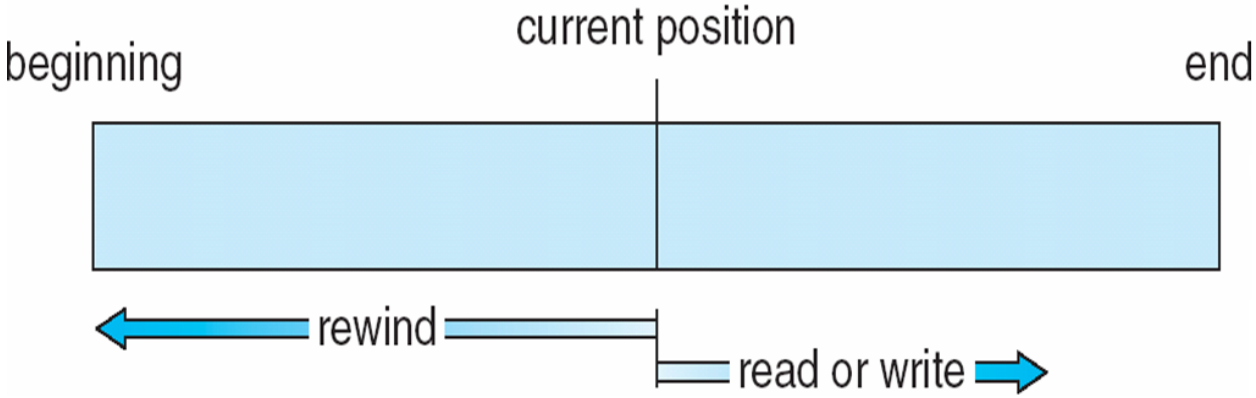
file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

OPERATING SYSTEMS

File Structure

- None - sequence of words, bytes
- Simple record structure
  - Lines
  - Fixed length
  - Variable length
- Complex Structures
  - Formatted document
  - Relocatable load file
- Can simulate last two with first method by inserting appropriate control characters
- Who decides:
  - Operating system
  - Program

Access Methods



Sequential-access File

- **Sequential Access**

  read next
  write next
  reset

- **Direct Access (or relative access)** – file is fixed length logical records

  read *n*
  write *n*
  position to *n*
  read next
  write next

  n = relative block number (i.e. index relative to the beginning of the file)

- Relative block numbers allow OS to decide where file should be placed

Simulation of Sequential Access on Direct-access File

sequential access	implementation for direct access
reset	$cp = 0;$
read next	*read cp*; $cp = cp + 1;$
write next	*write cp*; $cp = cp + 1;$

- cp = current position
- Some systems allow only sequential file access; others allow only direct access

Other Access Methods

- Can be built on top of base methods
- General involve creation of an index for the file
- Keep index in memory for fast determination of location of data to be operated on (consider UPC code plus record of data about that item with associated prices)
- If index file becomes too large, create index for the index file
  - Primary index file contains pointers to the secondary index files, which point to the actual data items
- IBM indexed sequential-access method (ISAM)
  - Small master index, points to disk blocks of secondary index (which points to the actual file blocks)
  - File kept sorted on a defined key
  - Binary search of the master index provides the block number of the secondary index and another binary search to find the block containing the desired record

THANK YOU

Suresh Jamadagni
Department of Computer Science Engineering

sureshjamadagni@pes.edu

OPERATING SYSTEMS

File Management

Suresh Jamadagni
Department of Computer Science

OPERATING SYSTEMS

File System

Suresh Jamadagni
Department of Computer Science

OPERATING SYSTEMS

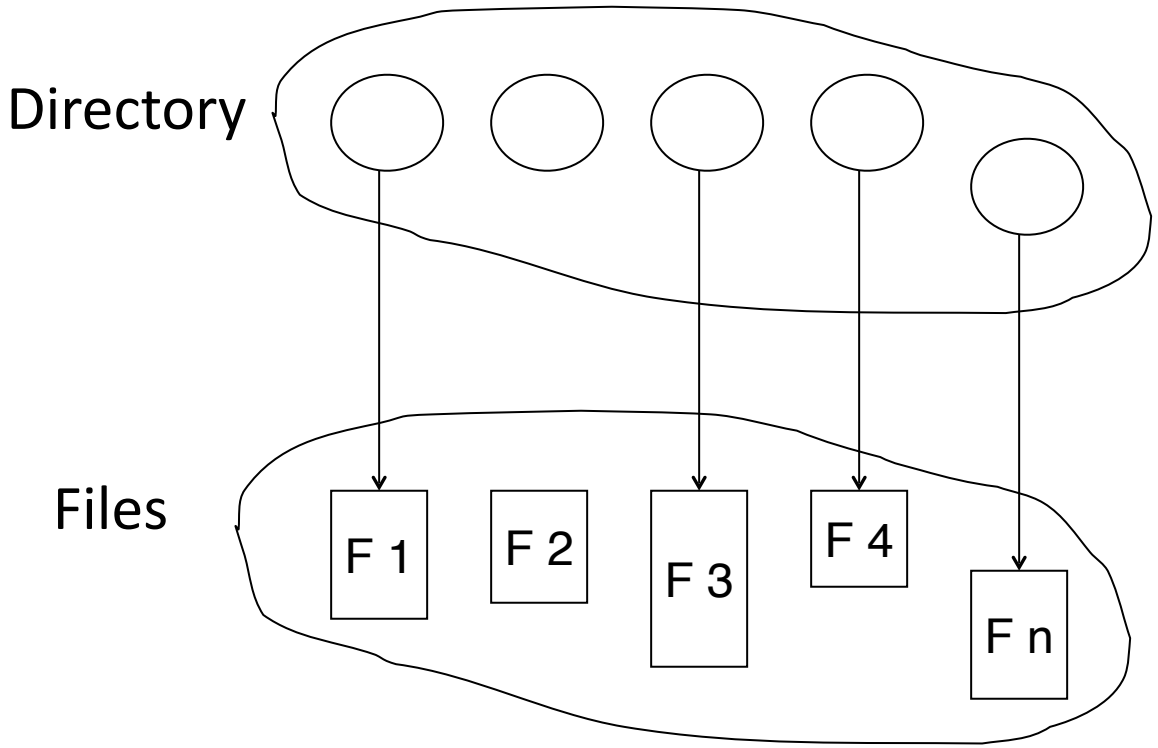
Slides Credits for all the PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9th edition 2013 and some slides from 10th edition 2018
2. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9th edition 2018
3. Some presentation transcripts from A. Frank – P. Weisberg
4. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau

Directory Structure

- A collection of nodes containing information about all files

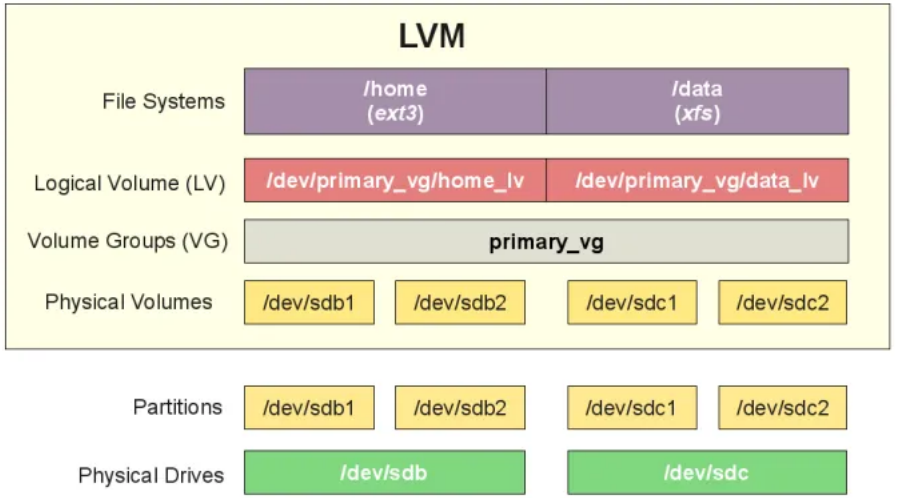


Both the directory structure and the files reside on disk

OPERATING SYSTEMS

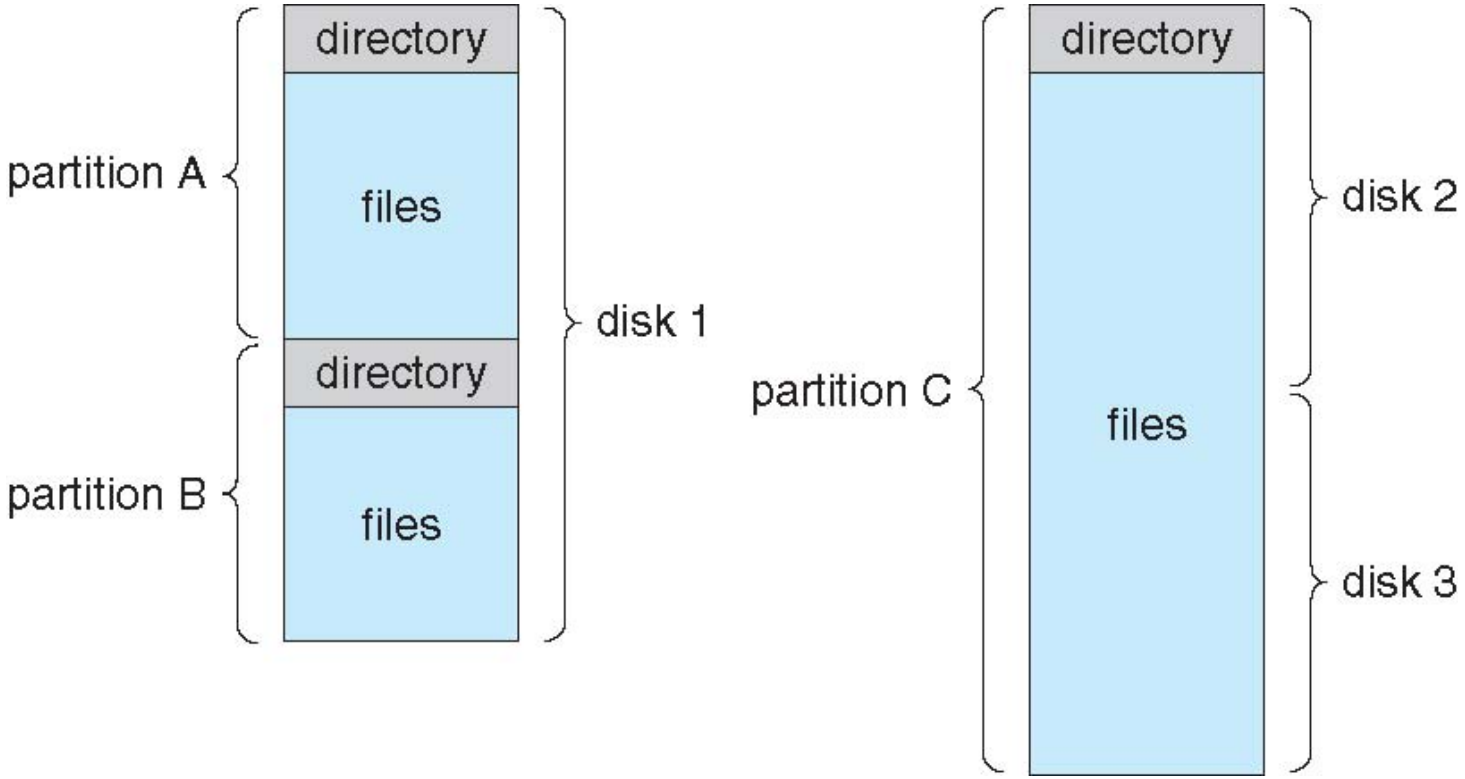
Disk Structure

- Disk can be subdivided into **partitions**
- Disks or partitions can be **RAID** protected against failure
- Disk or partition can be used **raw** – without a file system, or **formatted** with a file system
- Partitions also known as minidisks, slices
- Entity containing file system known as a **volume**
- Each volume containing file system also tracks that file system's info in **device directory** or **volume table of contents**
- As well as **general-purpose file systems** there are many **special-purpose file systems**, frequently all within the same operating system or computer



OPERATING SYSTEMS

A Typical File-system Organization



OPERATING SYSTEMS

Types of File Systems

- We mostly talk of general-purpose file systems
- But systems frequently have may file systems, some general- and some special- purpose
- Consider Solaris has
  - tmpfs – memory-based volatile FS for fast, temporary I/O
  - objfs – interface into kernel memory to get kernel symbols for debugging
  - ctfs – contract file system for managing daemons
  - lofs – loopback file system allows one FS to be accessed in place of another
  - procfs – kernel interface to process structures
  - ufs, zfs – general purpose file systems

Operations Performed on Directory

- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

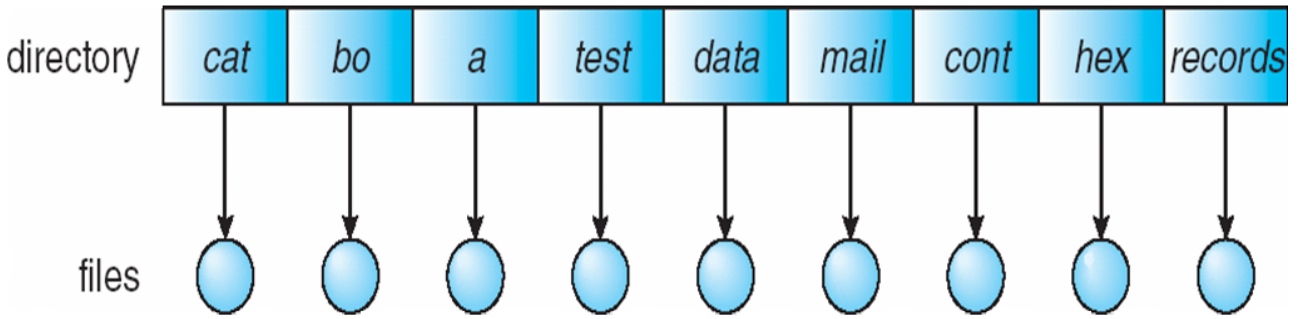
Directory Organization

The directory is organized logically to obtain

- Efficiency – locating a file quickly
- Naming – convenient to users
  - Two users can have same name for different files
  - The same file can have several different names
- Grouping – logical grouping of files by properties, (e.g., all Java programs, all games, …)

Single-Level Directory

- A single directory for all users

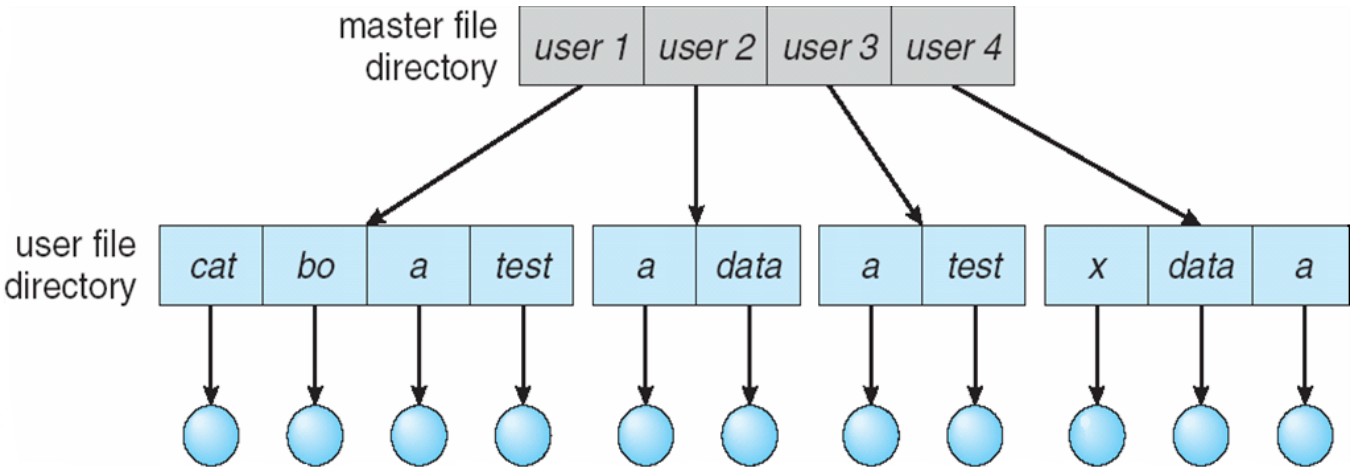


- Naming problem
- Grouping problem

OPERATING SYSTEMS

Two-Level Directory

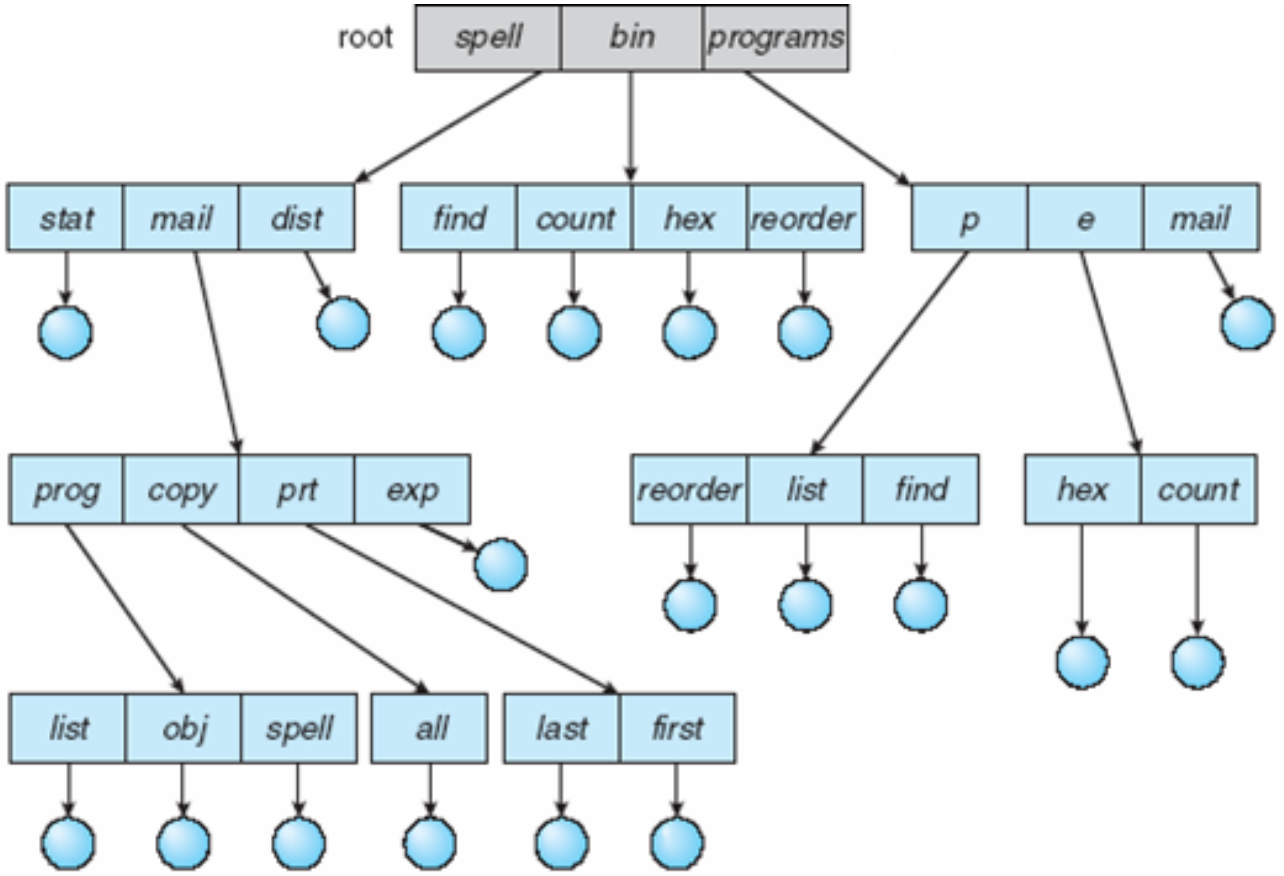
- Separate directory for each user



- Path name
- Can have the same file name for different user
- Efficient searching
- No grouping capability

OPERATING SYSTEMS

Tree-Structured Directories



- A tree structure is the most common directory structure.
- The tree has a root directory, and every file in the system have a unique path.

Tree-Structured Directories (Cont.)

- Efficient searching
- Grouping Capability
- Current directory (working directory)
  - **cd /spell/mail/prog**
  - **type list**
- We cannot share files

OPERATING SYSTEMS

Tree-Structured Directories (Cont.)

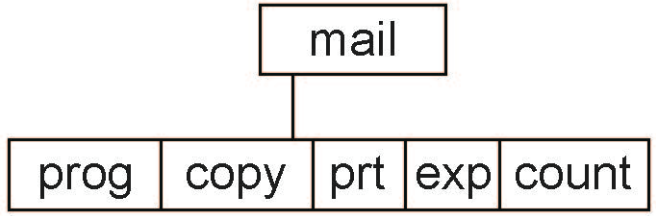
- **Absolute** or **relative** path name
- Creating a new file is done in current directory
- Delete a file

  rm <file-name>
- Creating a new subdirectory is done in current directory

  mkdir <dir-name>

- Example: if in current directory **/mail**

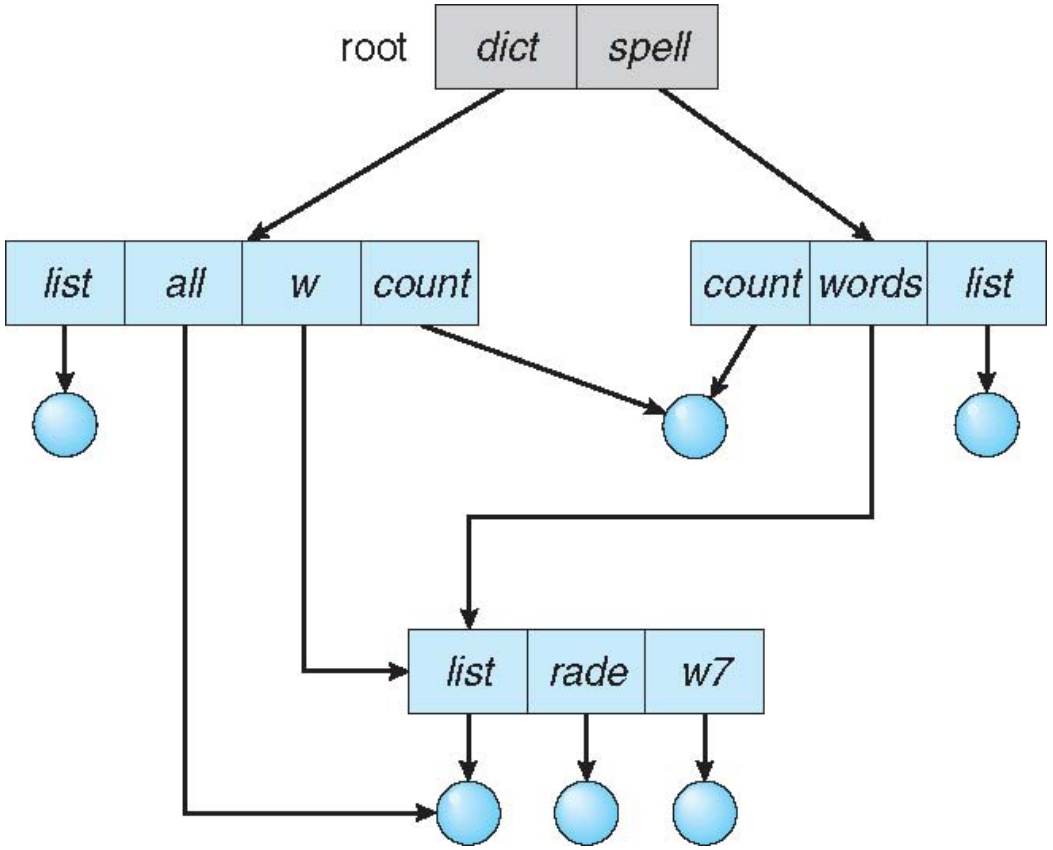
  mkdir count



Deleting “mail” ⇒ deleting the entire subtree rooted by “mail”

OPERATING SYSTEMS

Acyclic-Graph Directories



- An acyclic graph is a graph with no cycle and allows to share subdirectories and files.
- The same file or subdirectories may be in two different directories
- It is used in a situation like when two programmers are working on a joint project and they need to access files.

Acyclic-Graph Directories (Cont.)

- Two different names (aliasing)
- If ***dict*** deletes ***list*** ⇒ dangling pointer

  Solutions:

  - Backpointers, so we can delete all pointers
    Variable size records a problem
  - Backpointers using a daisy chain organization
  - Entry-hold-count solution
- New directory entry type
  - **Link** – another name (pointer) to an existing file
  - **Resolve the link** – follow pointer to locate the file

OPERATING SYSTEMS

General Graph Directory

Advantages:

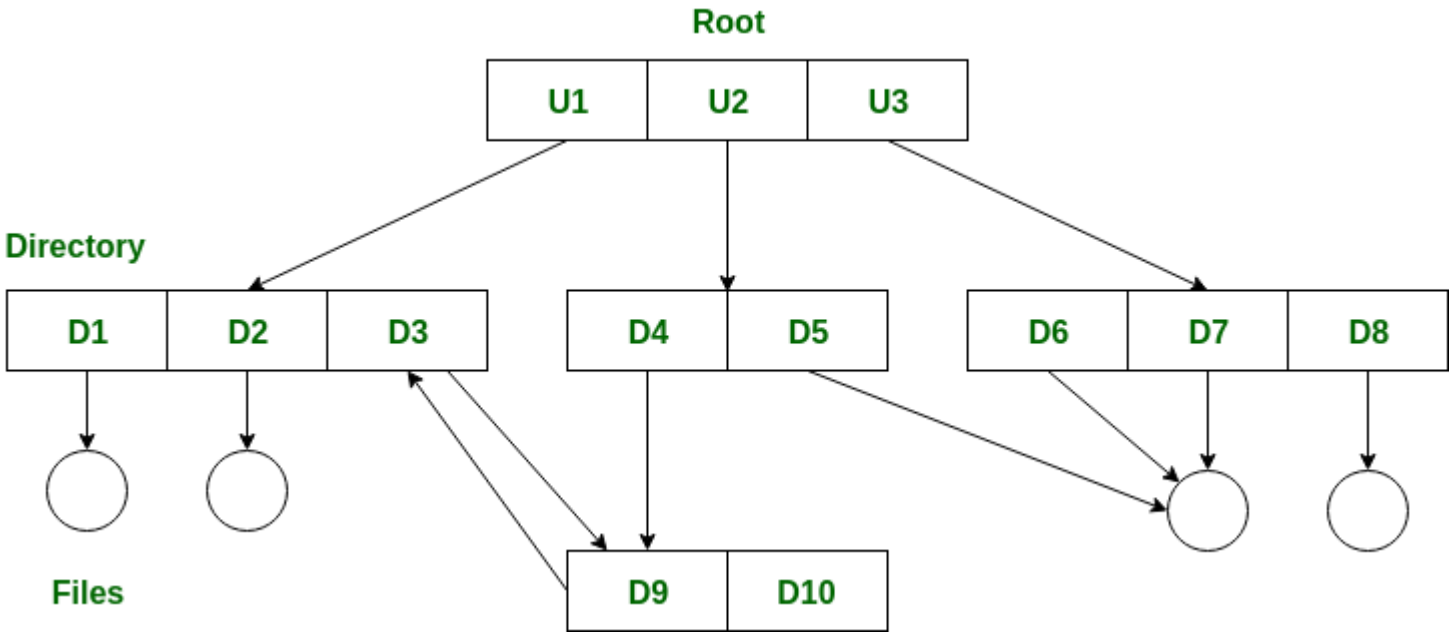
It allows cycles within a dir structure.

Multiple directories can be derived from more than one parent dir.

Disadvantages:

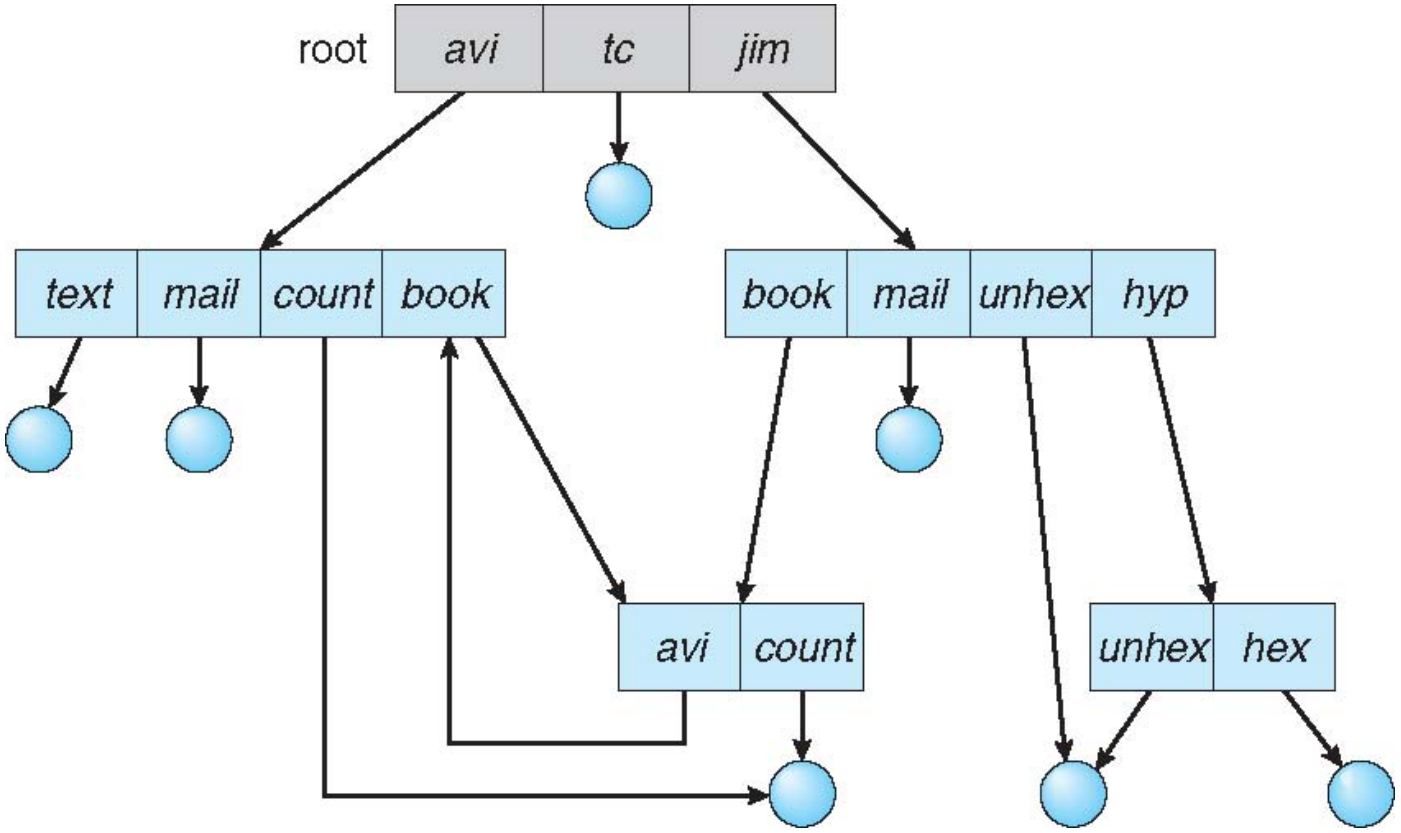
It is more costly than others.

It needs garbage collection (traversing the entire file system, marking everything that can be accessed)



General Graph Directory (Contd.)

- How do we guarantee no cycles?
  - Allow only links to file not subdirectories
  - **Garbage collection**
  - Every time a new link is added use a cycle detection algorithm to determine whether it is OK



THANK YOU

Suresh Jamadagni
Department of Computer Science Engineering

sureshjamadagni@pes.edu

OPERATING SYSTEMS

File Management

Suresh Jamadagni
Department of Computer Science

OPERATING SYSTEMS

File System – File-System Mounting, File Sharing and File Protection

Suresh Jamadagni
Department of Computer Science

OPERATING SYSTEMS

Slides Credits for all the PPTs of this course

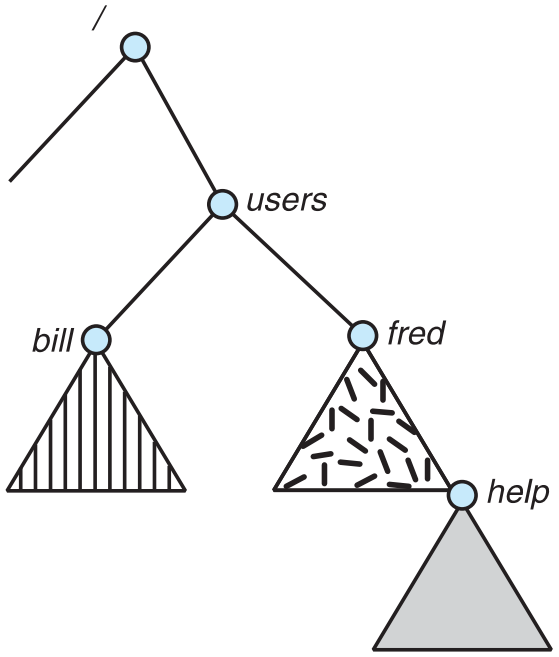
- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9th edition 2013 and some slides from 10th edition 2018
2. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9th edition 2018
3. Some presentation transcripts from A. Frank – P. Weisberg
4. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau

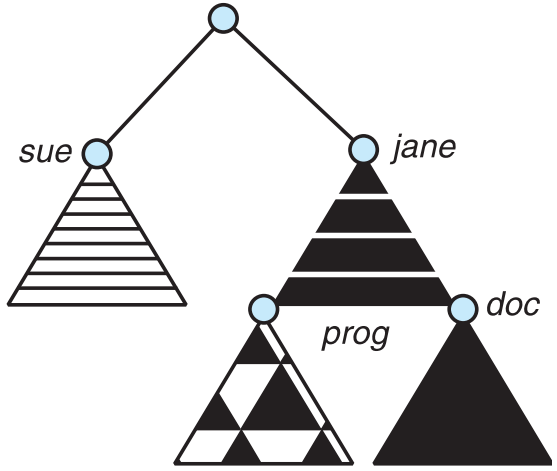
OPERATING SYSTEMS

File System Mounting

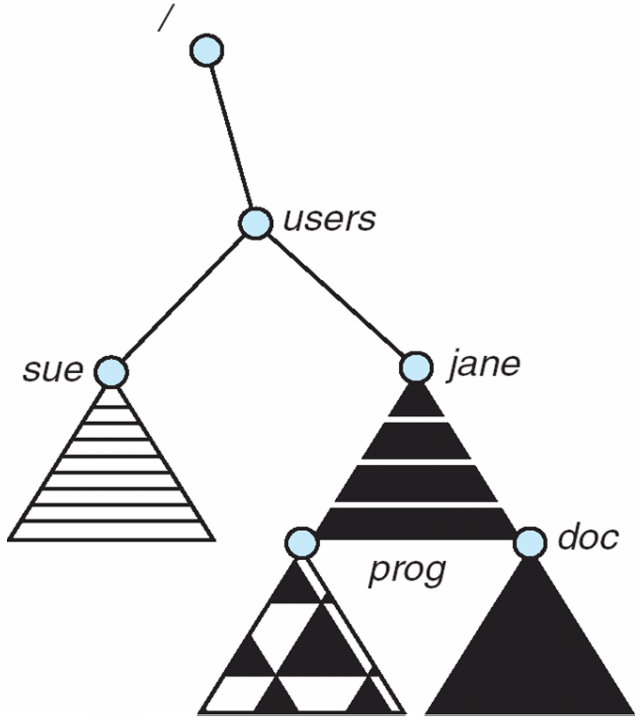
- A file system must be **mounted** before it can be accessed
- Fig (a) shows Existing File System
- A unmounted file system (i.e., Fig. (b)) is mounted at a **mount point**
- Fig (c) shows the effect of mounting



(a)



(b)



(c)

File Sharing

- Sharing of files on multi-user systems is desirable
- Sharing may be done through a **protection** scheme
- On distributed systems, files may be shared across a network
- Network File System (NFS) is a common distributed file-sharing method
- If multi-user system
  - **User IDs** identify users, allowing permissions and protections to be per-user
    Group IDs allow users to be in groups, permitting group access rights
  - Owner of a file / directory
  - Group of a file / directory

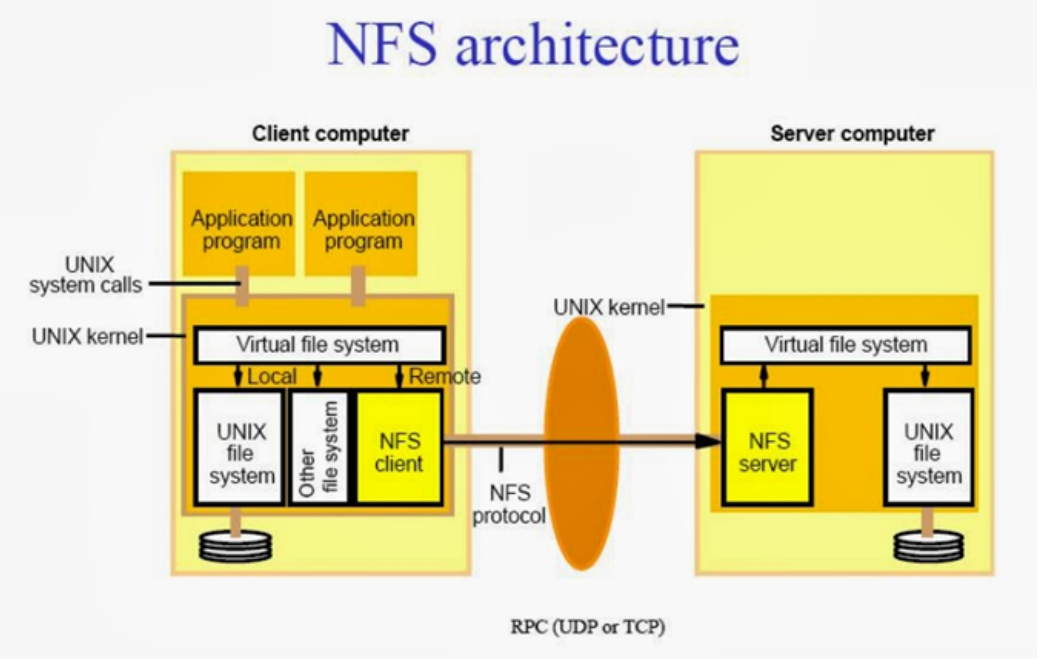
File Sharing – Remote File Systems

- Uses networking to allow file system access between systems
  - Manually via programs like FTP
  - Automatically, seamlessly using **distributed file systems**
  - Semi automatically via the **world wide web**

OPERATING SYSTEMS

File Sharing – Remote File Systems (Cont.)

- **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
  - Standard operating system file calls are translated into remote calls
- Distributed Information Systems **(distributed naming services)** such as LDAP, DNS, NIS, Active Directory implement unified access to information needed for remote computing

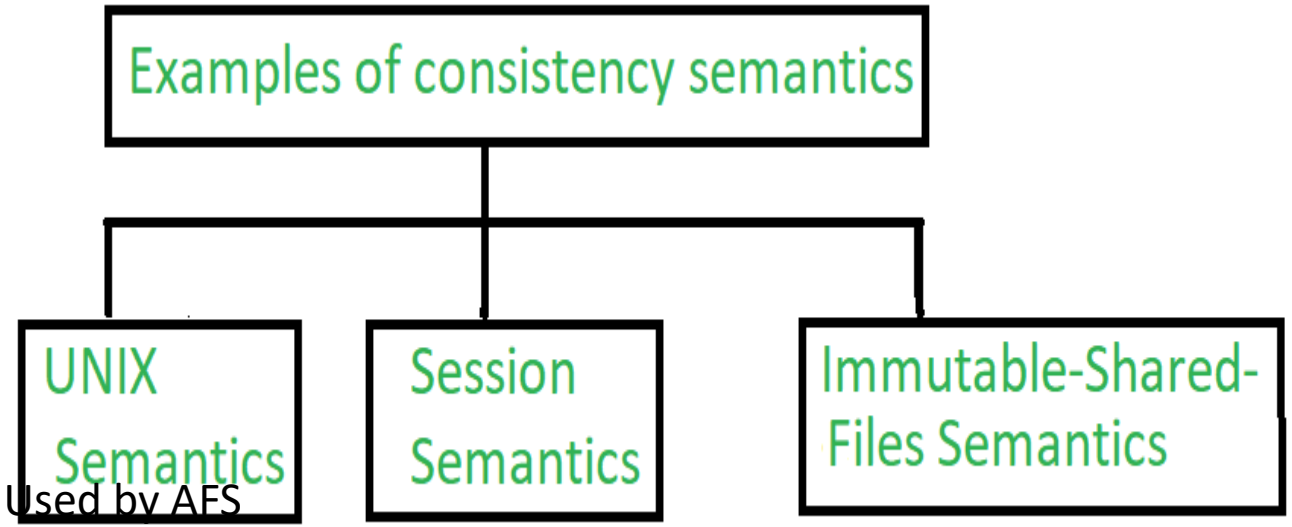


File Sharing – Failure Modes

- All file systems have failure modes
  - For example corruption of directory structures or other non-user data, called **metadata**
- Remote file systems add new failure modes, due to network failure, server failure
- Recovery from failure can involve **state information** about status of each remote request
- **Stateless** protocols such as NFS v3 include all information in each request, allowing easy recovery but less security

File Sharing – Consistency Semantics

- **Consistency Semantics** is a concept which is used by users to check file systems which are supporting file sharing in their systems.
- Basically, it is a specification to check how in a single system multiple users are getting access to same file and at same time.
  - like when will modification by some user in some file is noticeable to others.



File Sharing – Consistency Semantics (Contd.)

- Specify how multiple users are to access a shared file simultaneously
  - Similar to process synchronization algorithms
    - Tend to be less complex due to disk I/O and network latency (for remote file systems)
  - Andrew File System (AFS) implemented complex remote file sharing semantics
  - Unix file system (UFS) implements:
    - Writes to an open file visible immediately to other users of the same open file
    - Sharing file pointer to allow multiple users to read and write concurrently
  - AFS has session semantics
    - Writes only visible to sessions starting after the file is closed

Protection

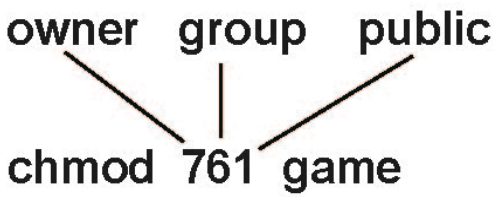
- File owner/creator should be able to control:
  - what can be done
  - by whom
- Types of access
  - **Read**
  - **Write**
  - **Execute**
  - **Append**
  - **Delete**
  - **List**

Access Lists and Groups

- Mode of access: read, write, execute
- Three classes of users on Unix / Linux

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

- Ask manager to create a group (unique name), say G, and add some users to the group.
- For a particular file (say *game*) or subdirectory, define an appropriate access.

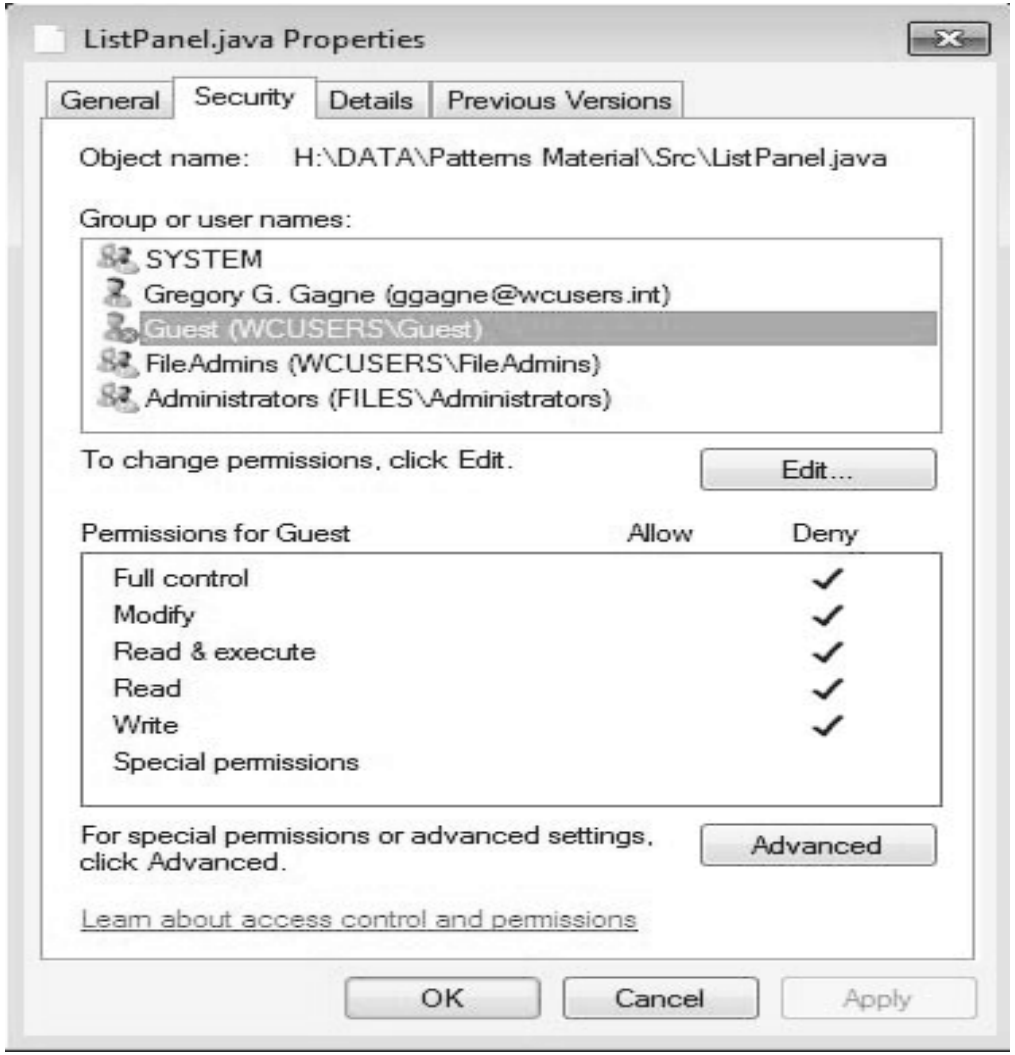


Attach a group to a file

```
chgrp  G  game
```

OPERATING SYSTEMS

Windows Access-Control List Management



THANK YOU

Suresh Jamadagni
Department of Computer Science and Engineering

sureshjamadagni@pes.edu

OPERATING SYSTEMS

File Management

Suresh Jamadagni
Department of Computer Science

OPERATING SYSTEMS

Files and Directories – System calls

Suresh Jamadagni
Department of Computer Science

OPERATING SYSTEMS

Slides Credits for all the PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9th edition 2013 and some slides from 10th edition 2018
2. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9th edition 2018
3. Some presentation transcripts from A. Frank – P. Weisberg
4. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau

Properties of a File

```
#include <sys/stat.h>

int stat(const char *restrict pathname, struct stat *restrict buf);

int fstat(int fd, struct stat *buf);

int lstat(const char *restrict pathname, struct stat *restrict buf);

int fstatat(int fd, const char *restrict pathname,
            struct stat *restrict buf, int flag);
```

All four return: 0 if OK, –1 on error

- **stat** function returns a structure of information about the named file
- The **fstat** function obtains information about the file that is already open on the descriptor fd.
- The **lstat** function returns information about the symbolic link, not the file referenced by the symbolic link
- The **fstatat** function returns the file statistics for a pathname relative to an open directory represented by the fd argument.

Structure that represents properties of a File

```
struct stat {
  mode_t          st_mode;    /* file type & mode (permissions) */
  ino_t           st_ino;     /* i-node number (serial number) */
  dev_t           st_dev;     /* device number (file system) */
  dev_t           st_rdev;    /* device number for special files */
  nlink_t         st_nlink;   /* number of links */
  uid_t           st_uid;     /* user ID of owner */
  gid_t           st_gid;     /* group ID of owner */
  off_t           st_size;    /* size in bytes, for regular files */
  struct timespec st_atim;    /* time of last access */
  struct timespec st_mtim;    /* time of last modification */
  struct timespec st_ctim;    /* time of last file status change */
  blksize_t       st_blksize; /* best I/O block size */
  blkcnt_t        st_blocks;  /* number of disk blocks allocated */
};
```

The timespec structure type defines time in terms of seconds and nanoseconds.

It includes at least the following fields:

time_t tv_sec;

long tv_nsec;

OPERATING SYSTEMS

File type macros

File type macros in <sys/stat.h>

Macro	Type of file
S_ISREG()	regular file
S_ISDIR()	directory file
S_ISCHR()	character special file
S_ISBLK()	block special file
S_ISFIFO()	pipe or FIFO
S_ISLNK()	symbolic link
S_ISSOCK()	socket

- Regular file - The most common type of file, which contains data of some form.
- Directory file - A file that contains the names of other files and pointers to information on these files
- Block special file - A type of file providing buffered I/O access in fixed-size units to devices such as disk drives
- Character special file - A type of file providing unbuffered I/O access in variable-sized units to devices.
- FIFO - A type of file used for communication between processes. It's sometimes called a named pipe
- Socket - A type of file used for network communication between processes
- Symbolic link - A type of file that points to another file

OPERATING SYSTEMS

File operations - creat, open and close

```
#include <fcntl.h>
int creat(const char *path, mode_t mode);
                    Returns: file descriptor opened for write-only if OK, –1 on error
```

- A new file can be created by calling the **creat** function.
- creat function is equivalent to **open** (path, O_WRONLY | O_CREAT | O_TRUNC, mode);

```
#include <fcntl.h>
int open(const char *path, int oflag, ... /* mode_t mode */ );
int openat(int fd, const char *path, int oflag, ... /* mode_t mode */ );
                    Both return: file descriptor if OK, –1 on error
```

openat function

```
#include <unistd.h>
int close(int fd);
                    Returns: 0 if OK, –1 on error
```

OPERATING SYSTEMS

File operations - lseek

```
#include <unistd.h>
off_t lseek(int fd, off_t offset, int whence);
                                    Returns: new file offset if OK, –1 on error
```

- Every open file has an associated ‘‘current file offset,’’ normally a non-negative integer that measures the number of bytes from the beginning of the file
- Read and write operations normally start at the current file offset and cause the offset to be incremented by the number of bytes read or written. By default, this offset is initialized to 0 when a file is opened, unless the O_APPEND option is specified.
- If whence is SEEK_SET, the file’s offset is set to offset bytes from the beginning of the file.
- If whence is SEEK_CUR, the file’s offset is set to its current value plus the offset. The offset can be positive or negative.
- If whence is SEEK_END, the file’s offset is set to the size of the file plus the offset. The offset can be positive or negative

OPERATING SYSTEMS

File operations - read

```
#include <unistd.h>

ssize_t read(int fd, void *buf, size_t nbytes);

                                Returns: number of bytes read, 0 if end of file, –1 on error
```

- Data is read from an open file with the read function
- If the read is successful, the number of bytes read is returned. If the end of file is encountered, 0 is returned.
- The read operation starts at the file's current offset. Before a successful return, the offset is incremented by the number of bytes actually read.

OPERATING SYSTEMS

File operations - write

```
#include <unistd.h>

ssize_t write(int fd, const void *buf, size_t nbytes);

                                    Returns: number of bytes written if OK, –1 on error
```

- Data is written to an open file with the write function.
- The return value is usually equal to the nbytes argument; otherwise, an error has occurred
- A common cause for a write error is either filling up a disk or exceeding the file size limit for a given process
- For a regular file, the write operation starts at the file's current offset.
- If the O_APPEND option was specified when the file was opened, the file's offset is set to the current end of file before each write operation.
- After a successful write, the file's offset is incremented by the number of bytes actually written

OPERATING SYSTEMS

Directories

```
#include <sys/stat.h>

int mkdir(const char *pathname, mode_t mode);

int mkdirat(int fd, const char *pathname, mode_t mode);
```

Both return: 0 if OK, –1 on error

```
#include <unistd.h>

int rmdir(const char *pathname);
```

Returns: 0 if OK, –1 on error

- Directories are created with the mkdir and mkdirat functions, and deleted with the rmdir function.
- The specified file access permissions, **mode**, are modified by the file mode creation mask of the process

OPERATING SYSTEMS

Reading Directories

```
#include <dirent.h>
DIR *opendir(const char *pathname);
DIR *fdopendir(int fd);
                                    Both return: pointer if OK, NULL on error
struct dirent *readdir(DIR *dp);
                          Returns: pointer if OK, NULL at end of directory or error
void rewinddir(DIR *dp);
int closedir(DIR *dp);
                                                Returns: 0 if OK, –1 on error
long telldir(DIR *dp);
                          Returns: current location in directory associated with dp
void seekdir(DIR *dp, long loc);
```

- Directories can be read by anyone who has access permission to read the directory. But only the kernel can write to a directory, to preserve file system sanity
- **fdopendir** provides a way to convert an open file descriptor into a DIR structure for use by the directory handling functions.
- The **dirent** structure defined in is implementation dependent.

```
ino_t  d_ino;          /* i-node number */
char   d_name[];       /* null-terminated filename */
```

Hard links

```
#include <unistd.h>

int link(const char *existingpath, const char *newpath);

int linkat(int efd, const char *existingpath, int nfd, const char *newpath,
           int flag);

                                                    Both return: 0 if OK, –1 on error
```

- **link** function or the **linkat** function can be used to create a link to an existing file.
- These functions create a new directory entry, newpath, that references the existing file existingpath.
- If the newpath already exists, an error is returned. Only the last component of the newpath is created. The rest of the path must already exist

```
#include <unistd.h>

int unlink(const char *pathname);

int unlinkat(int fd, const char *pathname, int flag);

                                                    Both return: 0 if OK, –1 on error
```

the file, the data in the file is still accessible through the other links

Symbolic links

```
#include <unistd.h>
int symlink(const char *actualpath, const char *sympath);
int symlinkat(const char *actualpath, int fd, const char *sympath);
                                    Both return: 0 if OK, –1 on error
```

- A symbolic link is created with either the symlink or symlinkat function
- A new directory entry, sympath, is created that points to actualpath. It is not required that actualpath exist when the symbolic link is created
- The symlinkat function is similar to symlink, but the sympath argument is evaluated relative to the directory referenced by the open file descriptor for that directory (specified by the fd argument)

THANK YOU

Suresh Jamadagni
Department of Computer Science and Engineering

sureshjamadagni@pes.edu

OPERATING SYSTEMS

File Management

Suresh Jamadagni
Department of Computer Science

OPERATING SYSTEMS

Implementing File-Systems

Suresh Jamadagni
Department of Computer Science

OPERATING SYSTEMS

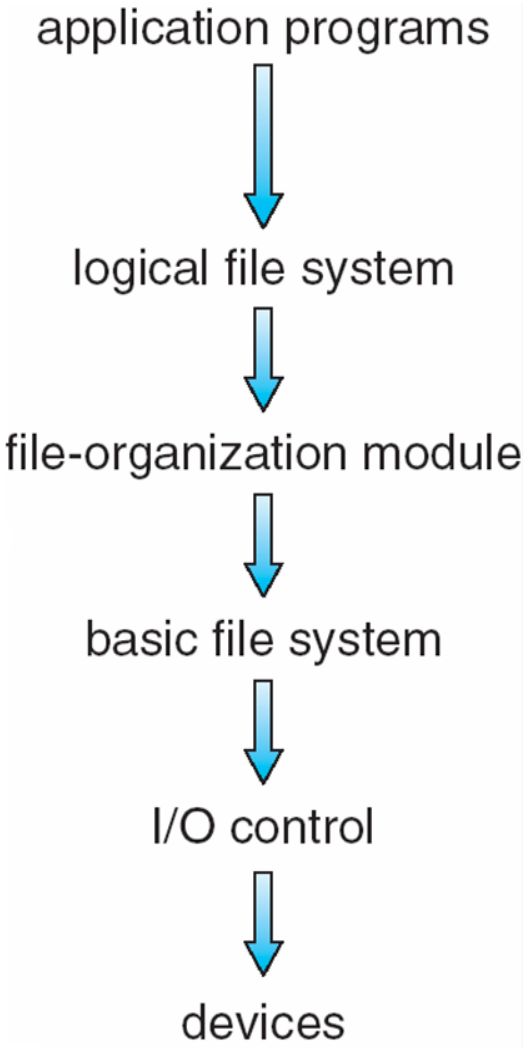
Slides Credits for all the PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9th edition 2013 and some slides from 10th edition 2018
2. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9th edition 2018
3. Some presentation transcripts from A. Frank – P. Weisberg
4. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau

OPERATING SYSTEMS

Layered File System



Manages metadata information, and the dir structure to provide the file-org module. Also responsible for protection

Translates logical to physical block addresses

Issues generic commands to the device driver, manages mem buffers

Device drivers and interrupt handlers

File-System Structure

- File structure
  - Logical storage unit
  - Collection of related information
- **File system** resides on secondary storage (disks)
  - Provided user interface to storage, mapping logical to physical
  - Provides efficient and convenient access to disk by allowing data to be stored, located retrieved easily
- Disk provides in-place rewrite and random access
  - I/O transfers performed in **blocks** of **sectors** (usually 512 bytes)
- **File control block (FCB)** – storage structure consisting of information about a file including ownership, permissions, and location of the file contents
- **Device driver** controls the physical device
- File system organized into layers

File System Layers

- **Device drivers** manage I/O devices at the I/O control layer
  - Given commands like “read drive1, cylinder 72, track 2, sector 10, into memory location 1060” outputs low-level hardware specific commands to hardware controller
- **Basic file system** given command like “retrieve block 123” translates to device driver
- Also manages memory buffers and caches (allocation, freeing, replacement)
  - Buffers hold data in transit
  - Caches hold frequently used data

File System Layers (Cont.)

- **File organization module** understands files, logical address, and physical blocks
- Translates logical block # to physical block #
- Manages free space, disk allocation
- **Logical file system** manages metadata information
  - Translates file name into file number, file handle, location by maintaining file control blocks (**inodes** in UNIX)
  - Directory management
  - Protection

OPERATING SYSTEMS

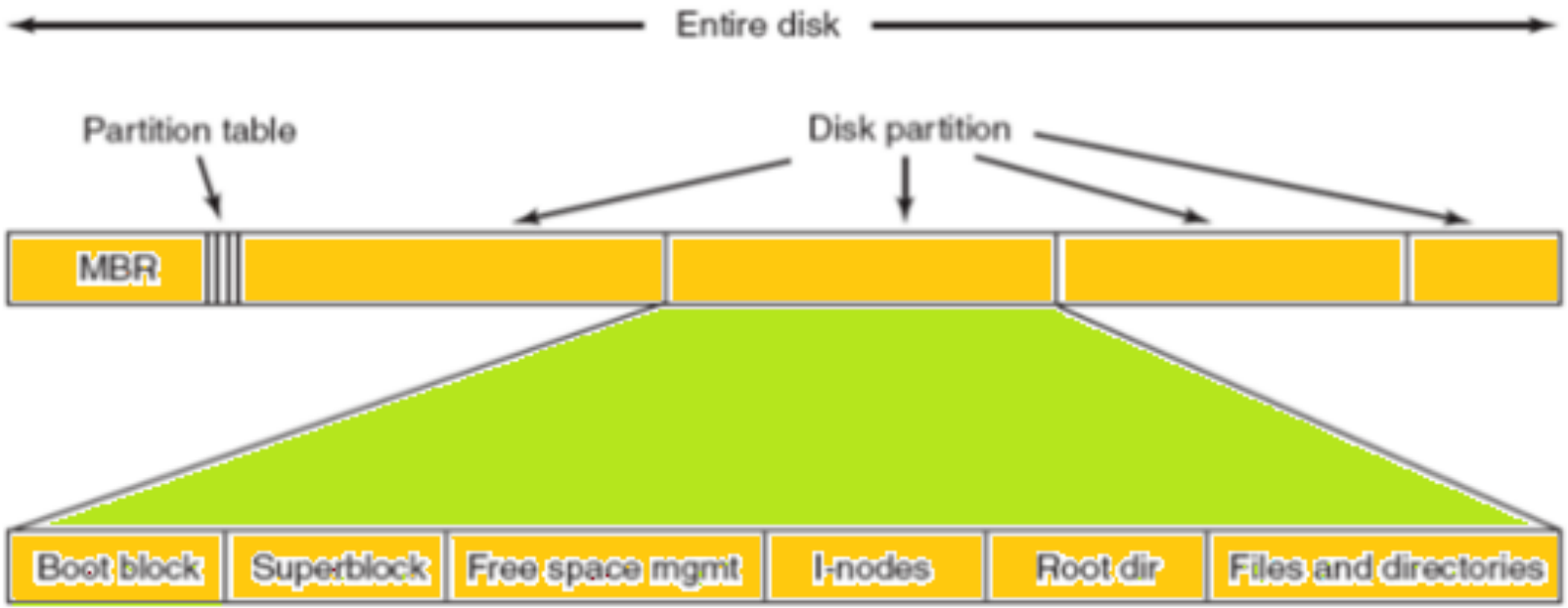
File System Layers (Cont.)

- Layering useful for reducing complexity and redundancy, but adds overhead and can decrease performance.
- Logical layers can be implemented by any coding method according to OS designer

- Many file systems, sometimes many within an operating system
  - Each with its own format (CD-ROM is ISO 9660; Unix has **UFS**, FFS; Windows has FAT, FAT32, NTFS as well as floppy, CD, DVD Blu-ray, Linux has more than 40 types, with **extended file system** ext2 and ext3 leading; plus distributed file systems, etc.)
  - New ones – ZFS, GoogleFS, Oracle ASM, FUSE

OPERATING SYSTEMS

File-System Implementation



A possible file system layout.

File-System Implementation

- We have system calls at the API level, but how do we implement their functions?
  - On-disk and in-memory structures
- **Boot control block** contains info needed by system to boot OS from that volume
  - Needed if volume contains OS, usually first block of volume
- **Volume control block (superblock, master file table)** contains volume details
  - Total # of blocks, # of free blocks, block size, free block pointers or array
- Directory structure organizes the files
  - Names and inode numbers, master file table

File-System Implementation (Cont.)

- Per-file **File Control Block (FCB)** contains many details about the file
  - inode number, permissions, size, dates
  - NFTS stores into in master file table using relational DB structures

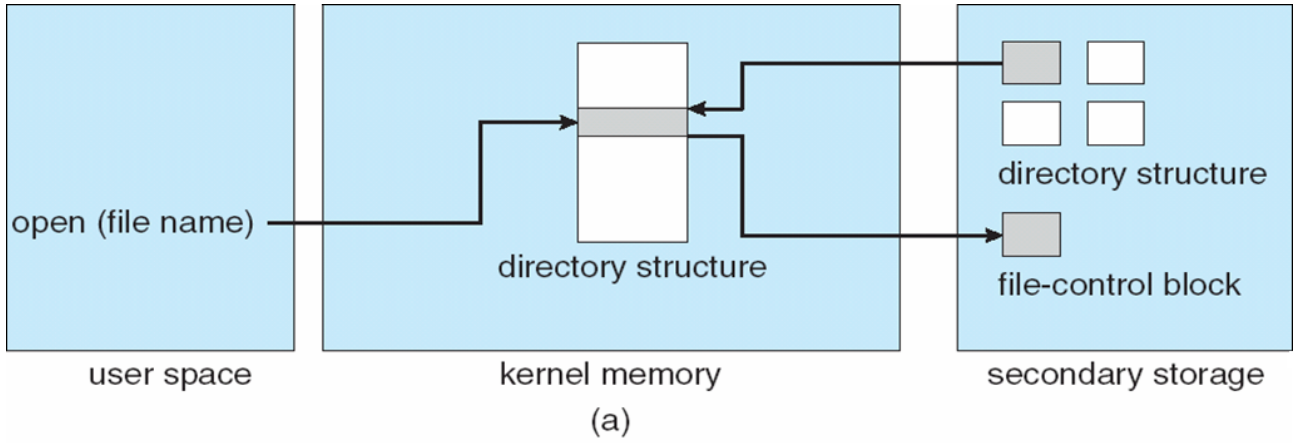
file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks or pointers to file data blocks

In-Memory File System Structures

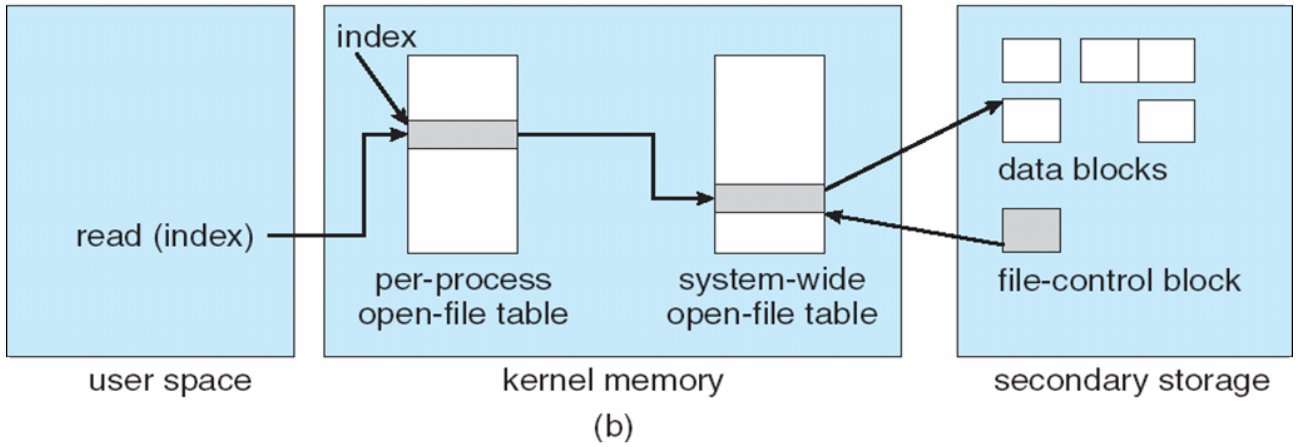
- Mount table storing file system mounts, mount points, file system types
- The following figure illustrates the necessary file system structures provided by the operating systems
- Figure (a) refers to opening a file
- Figure (b) refers to reading a file
- Plus buffers hold data blocks from secondary storage
- Open returns a file handle for subsequent use
- Data from read eventually copied to specified user process memory address

OPERATING SYSTEMS

In-Memory File System Structures



(a)



(b)

OPERATING SYSTEMS

Partitions and Mounting

- Partition can be a volume containing a file system ("cooked") or **raw** – just a sequence of blocks with no file system
- Boot block can point to boot volume or boot loader set of blocks that contain enough code to know how to load the kernel from the file system
  - Or a boot management program for multi-os booting
- **Root partition** contains the OS, other partitions can hold other OS's, other file systems, or be raw
  - Mounted at boot time
  - Other partitions can mount automatically or manually
- At mount time, file system consistency checked
  - Is all metadata correct?
    - If not, fix it, try again
    - If yes, add to mount table, allow access

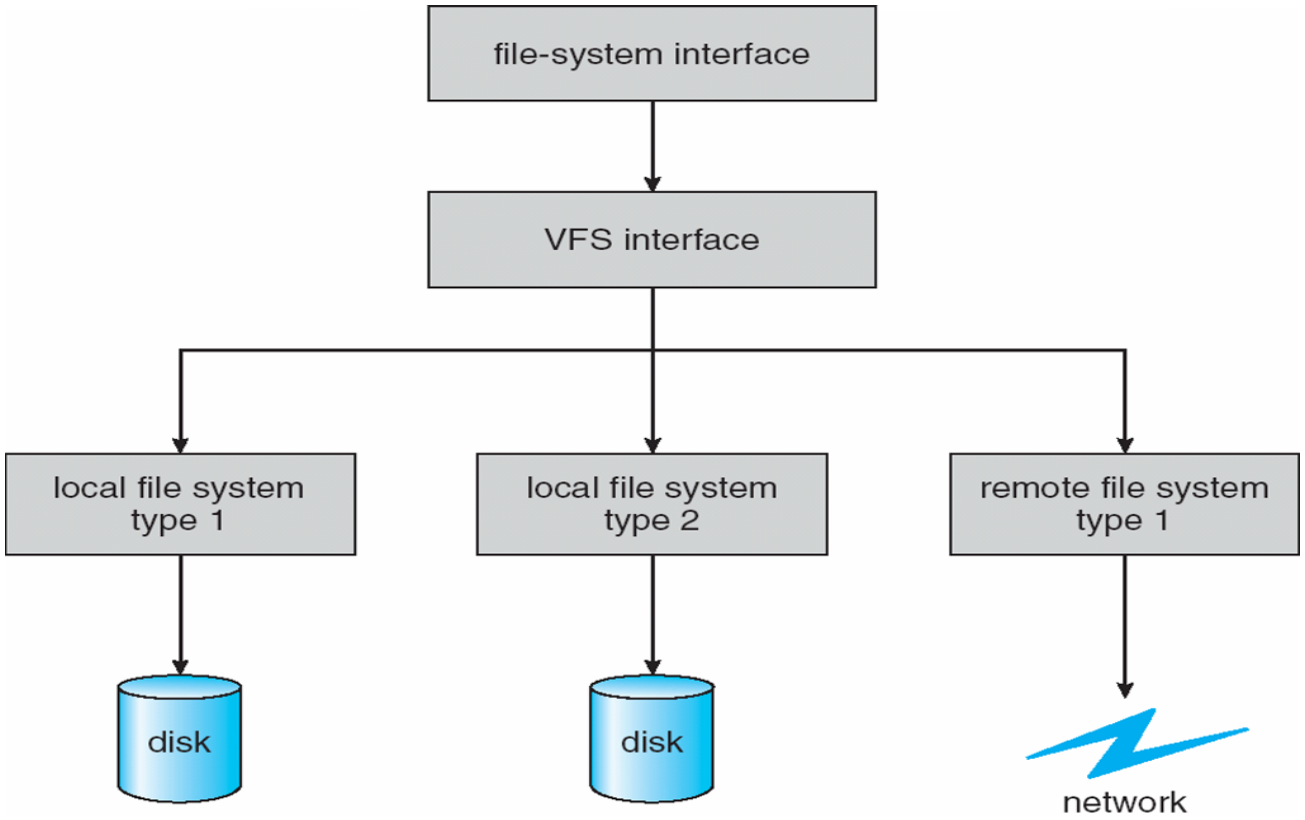
Virtual File Systems

- **Virtual File Systems** (**VFS**) on Unix provide an object-oriented way of implementing file systems
- VFS allows the same system call interface (the API) to be used for different types of file systems
  - Separates file-system generic operations from implementation details
  - Implementation can be one of many file systems types, or network file system
    - 4 Implements **vnodes** which hold inodes or network file details
  - Then dispatches operation to appropriate file system implementation routines

OPERATING SYSTEMS

Virtual File Systems

- The API is to the VFS interface, rather than any specific type of file system (FS)
- VFS provides a single set of commands for the kernel and developers to access all types of FSs
- VFS software calls the specific device driver required to interface to various types of FSs
- Device driver interprets the standard set of FS commands to a specific type of FS on the partition or logical volume



Virtual File Systems

- For example, Linux has four object types:
  - **Inode object, file object, superblock object, dentry object**
- VFS defines set of operations on the objects that must be implemented
  - Every object has a pointer to a function table
    Function table has addresses of routines to implement that function on that object
    For example:
    - **int open(. . .)**—Open a file
    - **int close(. . .)**—Close an already-open file
    - **ssize t read(. . .)**—Read from a file
    - **ssize t write(. . .)**—Write to a file
    - **int mmap(. . .)**—Memory-map a file

THANK YOU

Suresh Jamadagni
Department of Computer Science and Engineering

sureshjamadagni@pes.edu

OPERATING SYSTEMS

File Management

Suresh Jamadagni
Department of Computer Science

OPERATING SYSTEMS

Implementing File-Systems – Disk Space Allocation Methods

Suresh Jamadagni
Department of Computer Science

OPERATING SYSTEMS

Slides Credits for all the PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

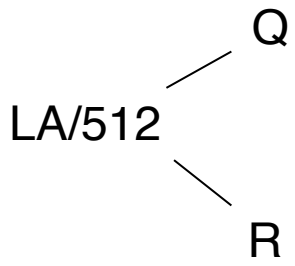
1. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9th edition 2013 and some slides from 10th edition 2018
2. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9th edition 2018
3. Some presentation transcripts from A. Frank – P. Weisberg
4. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau

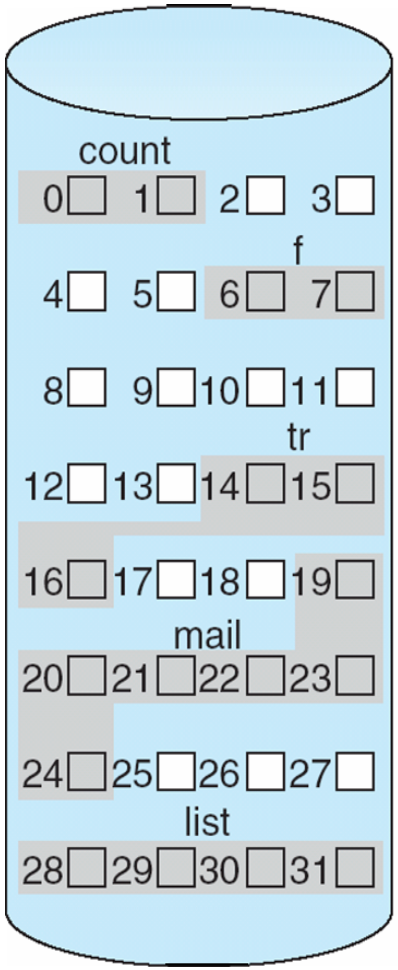
Allocation Methods - Contiguous

- An allocation method refers to how disk blocks are allocated for files:
- **Contiguous allocation** – each file occupies set of contiguous blocks
  - Best performance in most cases
  - Simple – only starting location (block #) and length (number of blocks) are required
  - Supports both sequential and direct access
  - Problems include finding space for file, knowing file size, external fragmentation, need for **compaction off-line** (**downtime**) or **on-line**

Contiguous Allocation

- Mapping from logical to physical





directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

- LA – Length of the area allocated for this file
- Q = displacement into index table
- Block to be accessed = Q + starting address
- Displacement into block = R

OPERATING SYSTEMS

Extent-Based Systems

- Many newer file systems (i.e., Veritas File System) use a modified contiguous allocation scheme
- Extent-based file systems allocate disk blocks in extents
- An **extent** is a contiguous block of disks
  - Extents are allocated for file allocation
  - A file consists of one or more extents

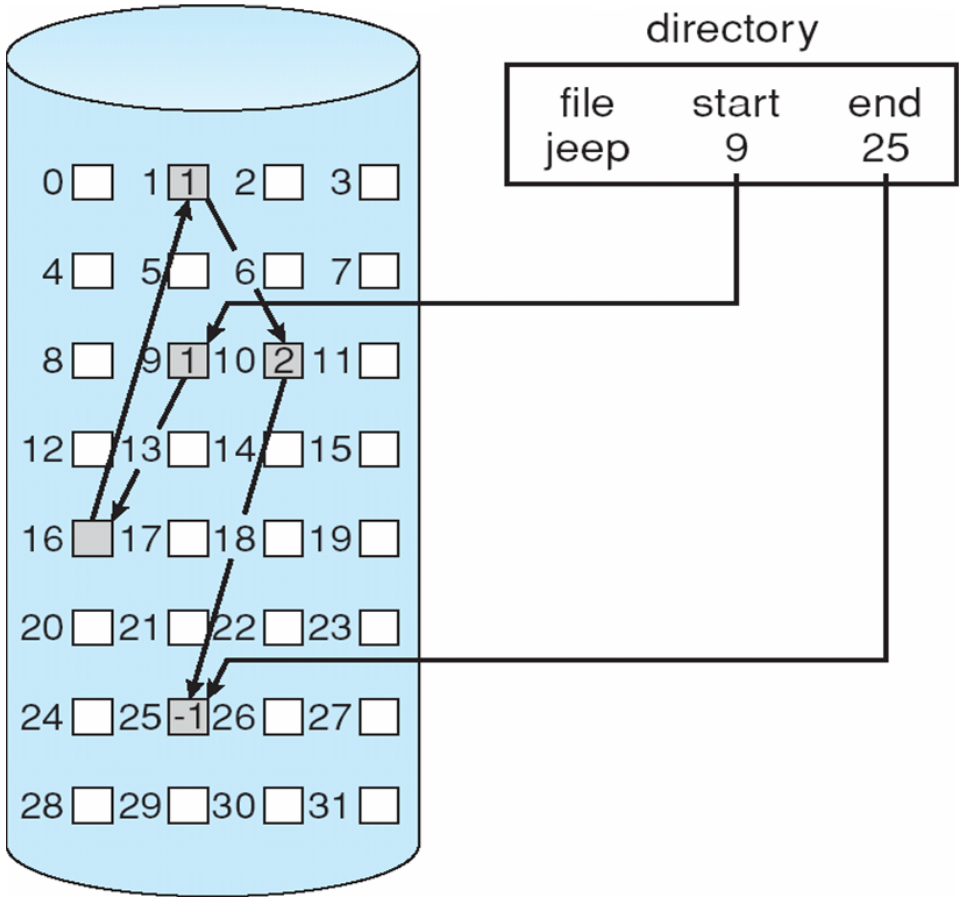
Allocation Methods - Linked

- **Linked allocation** – each file a linked list of blocks
  - File ends with null pointer
  - No external fragmentation
  - Each block contains pointer to next block
  - No compaction, external fragmentation
  - Free space management system called when new block needed
  - Improve efficiency by clustering blocks into groups but increases internal fragmentation
  - Reliability can be a problem
  - Locating a block can take many I/Os and disk seeks

Allocation Methods – Linked (Cont.)

- FAT (File Allocation Table) variation
  - Beginning of volume has table, indexed by block number
  - Much like a linked list, but faster on disk and cacheable
  - New block allocation simple

Linked Allocation

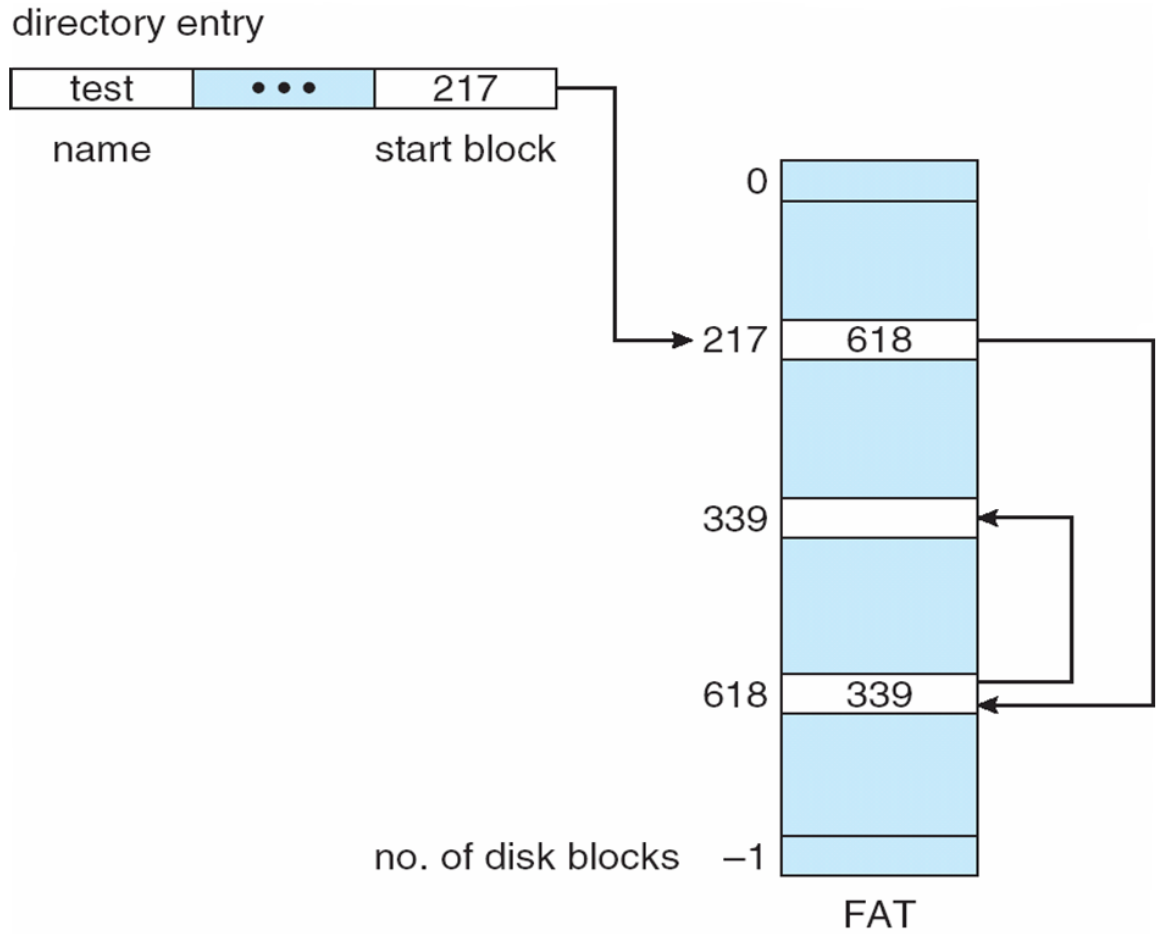


- Each block contains a pointer to the next block.
- If each block is 512 bytes and a disk address (pointer) requires 4 bytes then the user sees blocks of 508 bytes (i.e. some disk space is wasted)
- Collect blocks into multiples called **clusters** and allocate clusters rather than blocks

OPERATING SYSTEMS

File-Allocation Table (variation on linked allocation)

- Unused block is indicated by a table value of 0
- To allocate a new block to a file, find the first 0-valued table entry and replace the previous EOF value with the address of the new block.
- Then replace 0 with EOF value

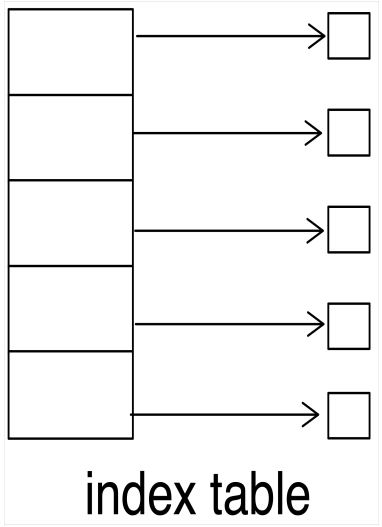


Allocation Methods - Indexed

Indexed allocation

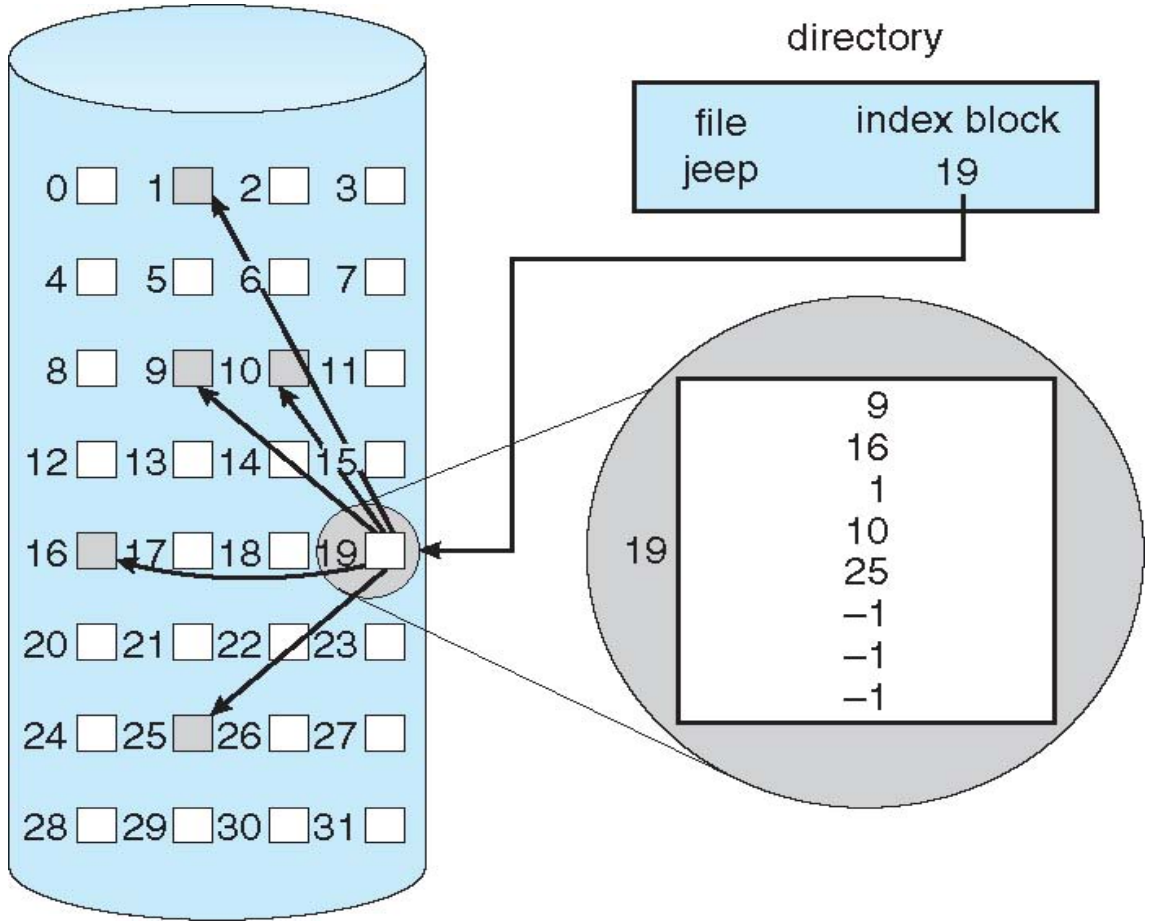
- Each file has its own **index block**(s) of pointers to its data blocks

- Logical view



Example of Indexed Allocation

- In this scheme, a special block known as the **Index block** contains the pointers to all the blocks occupied by a file.
- Each file has its own index block. The ith entry in the index block contains the disk address of the ith file block.
- The directory entry contains the address of the index block as shown in the figure.



Allocation Methods – Indexed (Cont.)

Advantages:

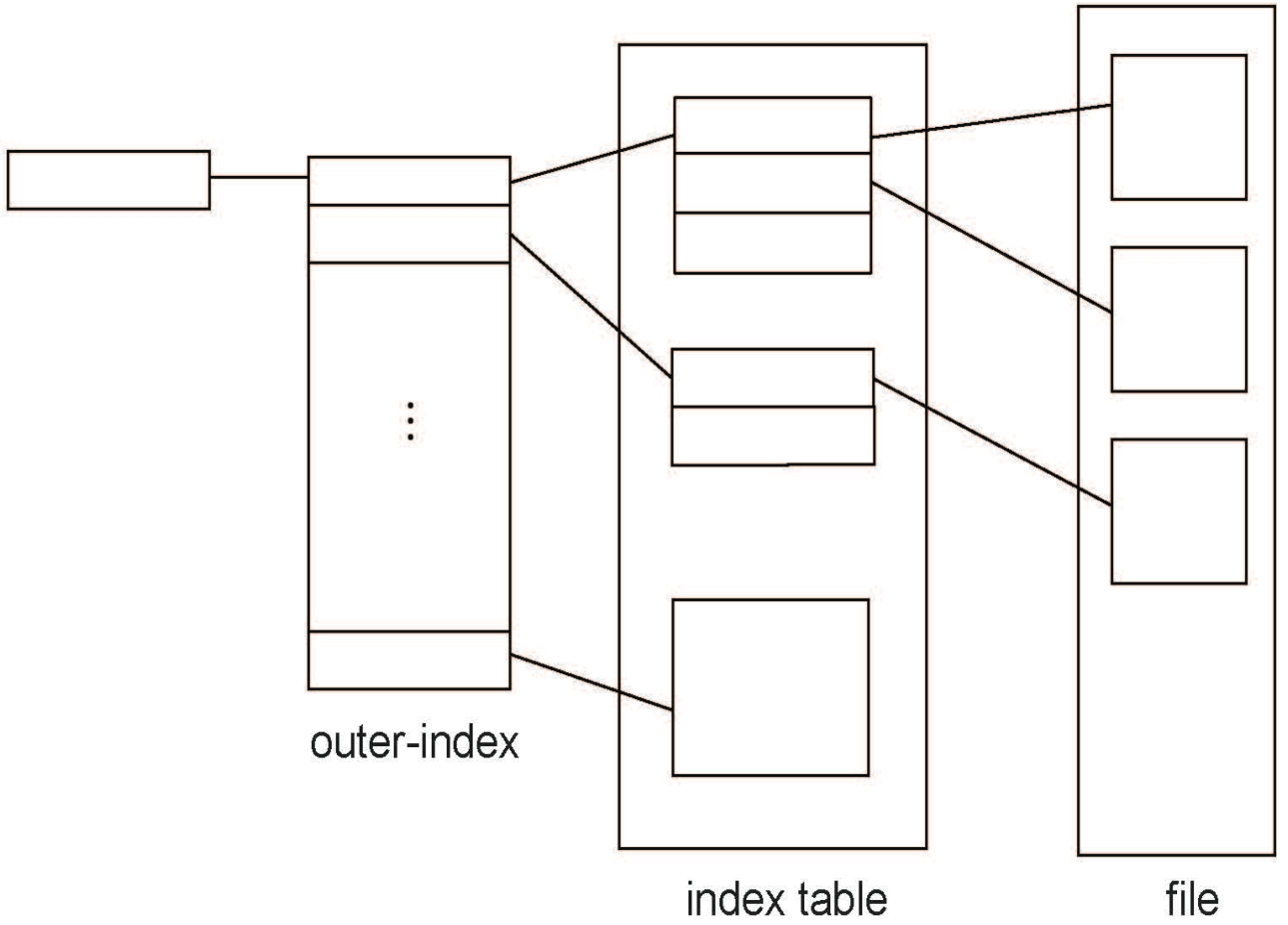
- This supports direct access to the blocks occupied by the file and therefore provides fast access to the file blocks.
- It overcomes the problem of external fragmentation.

Disadvantages:

- The pointer overhead for indexed allocation is greater than linked allocation.
- For very small files, say files that expand only 2-3 blocks, the indexed allocation would keep one entire block (index block) for the pointers which is inefficient in terms of memory utilization. (Note: In linked allocation we lose the space of only 1 pointer per block.)

Indexed Allocation – Mapping (Cont.)

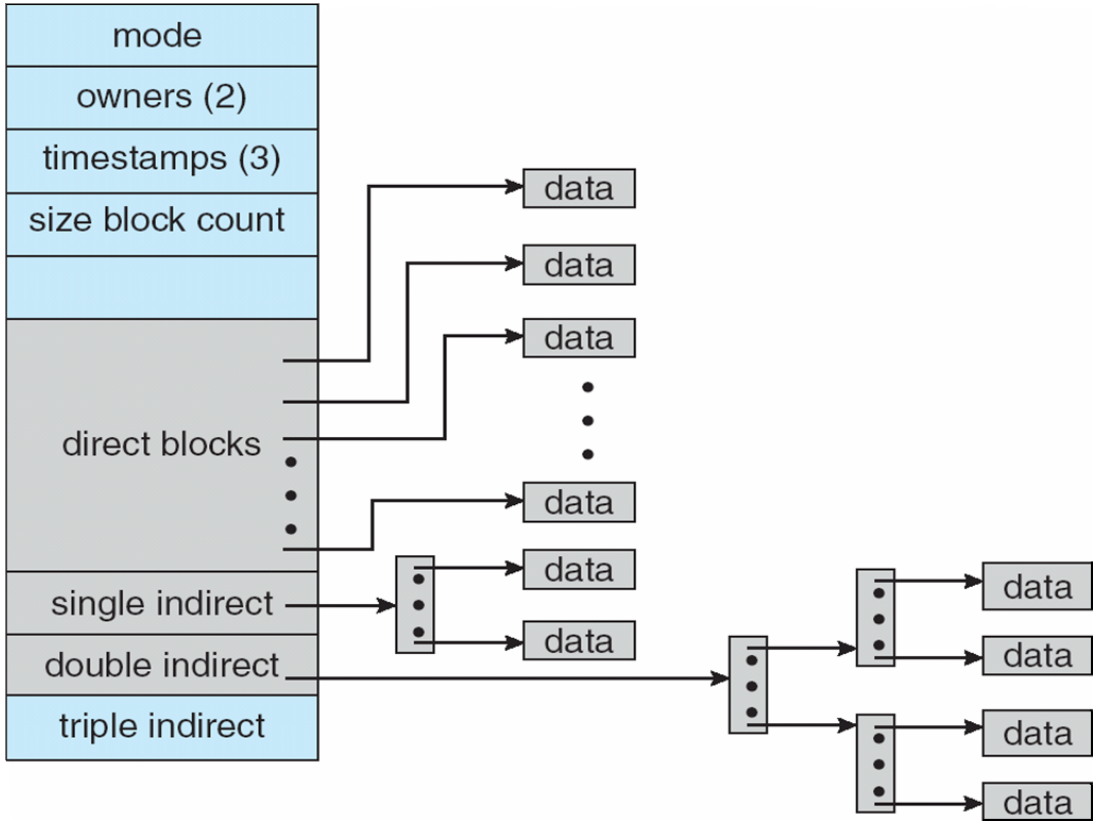
- For files that are very large, single index block may not be able to hold all the pointers.
- Other Schemes such as Linked scheme, Multilevel index and Combined Scheme are used.



OPERATING SYSTEMS

Combined Scheme: UNIX UFS

4K bytes per block, 32-bit addresses



More index blocks than can be addressed with 32-bit file pointer

Performance

- Best method depends on file access type
  - Contiguous great for sequential and random
- Linked good for sequential, not random
- Declare access type at creation -> select either contiguous or linked
- Indexed more complex
  - Single block access could require 2 index block reads then data block read
  - Clustering can help improve throughput, reduce CPU overhead

THANK YOU

Suresh Jamadagni
Department of Computer Science Engineering

sureshjamadagni@pes.du

OPERATING SYSTEMS

File Management

Suresh Jamadagni
Department of Computer Science

OPERATING SYSTEMS

File Management – Free space management

Suresh Jamadagni
Department of Computer Science

OPERATING SYSTEMS

Slides Credits for all the PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9th edition 2013 and some slides from 10th edition 2018
2. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9th edition 2018
3. Some presentation transcripts from A. Frank – P. Weisberg
4. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau

Free-Space Management

- Need to reuse the disk space from deleted files for new files
- To keep track of free disk space, the system maintains a **free-space list**
- The free-space list records all **free disk blocks**
- When a file is created, the free-space list is searched for the required amount of space and allocate that space to the new file. This space is then removed from the free-space list.
- When a file is deleted, its disk space is added to the free-space list

Bit Vector

- The free-space list is implemented as a **bit map** or **bit vector**.
- Each block is represented by 1 bit.
- If the block is free, the bit is 1; if the block is allocated, the bit is 0
- For example, consider a disk where blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, and 27 are free and the rest of the blocks are allocated.
- The free-space bit map would be 00111100111111000110000011100000 ...
- The main advantage of this approach is its relative simplicity and its efficiency in finding the first free block or n consecutive free blocks on the disk

Bit Vector

- One technique for finding the first free block on a system that uses a bit-vector to allocate disk space is to sequentially check each word in the bit map to see whether that value is not 0, since a 0-valued word contains only 0 bits and represents a set of allocated blocks
- The first non-0 word is scanned for the first 1 bit, which is the location of the first free block.
- The calculation of the block number is

  (number of bits per word) × (number of 0-value words) + offset of first 1 bit

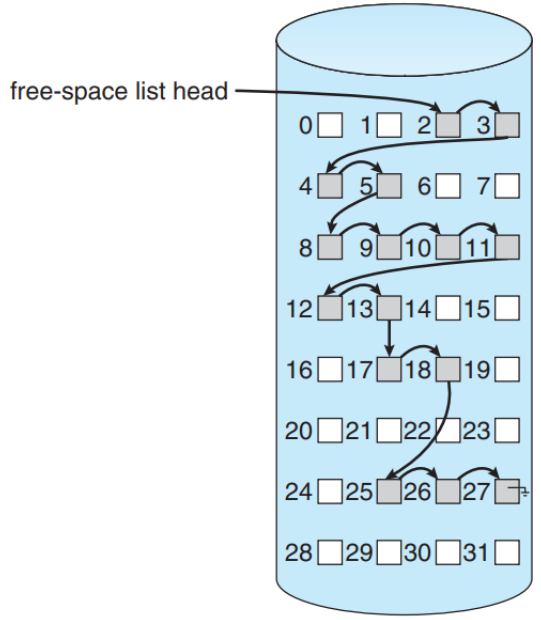
Bit Vector

- Bit vectors are inefficient unless the entire vector is kept in main memory
- Keeping the bit vector in main memory is possible for smaller disks but not necessarily for larger ones
- A 1.3-GB disk with 512-byte blocks would need a bit map of over 332 KB to track its free blocks
- A 1-TB disk with 4-KB blocks requires 32 MB to store its bit map.

OPERATING SYSTEMS

Linked List

- Link together all the free disk blocks, keeping a pointer to the first free block in a special location on the disk and caching it in memory.
- The first free block contains a pointer to the next free disk block, and so on
- **Example:** Consider blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, and 27 are free and the rest of the blocks are allocated.
- The pointer would point to block 2 as the first free block. Block 2 would contain a pointer to block 3, which would point to block 4 and so on
- This scheme is not efficient; to traverse the list, one must read each block, which requires substantial I/O time.



Grouping

- A modification of the free-list approach stores the addresses of n free blocks in the first free block.
- The first n−1 of these blocks are actually free.
- The last block contains the addresses of another n free blocks, and so on.
- The addresses of a large number of free blocks can now be found quickly, unlike the situation when the standard linked-list approach is used.

Counting

- Generally, several contiguous blocks may be allocated or freed simultaneously, particularly when space is allocated with the contiguous-allocation algorithm or through clustering.
- Rather than keeping a list of n free disk addresses, we can keep the address of the first free block and the number (n) of free contiguous blocks that follow the first block
- Each entry in the free-space list then consists of a disk address and a count.
- The entries can be stored in a balanced tree, rather than a linked list, for efficient lookup, insertion, and deletion.

THANK YOU

Suresh Jamadagni
Department of Computer Science Engineering

sureshjamadagni@pes.edu

OPERATING SYSTEMS

File Management

Suresh Jamadagni
Department of Computer Science

OPERATING SYSTEMS

File-Systems – Case Study: Linux

Suresh Jamadagni
Department of Computer Science

OPERATING SYSTEMS

Slides Credits for all the PPTs of this course

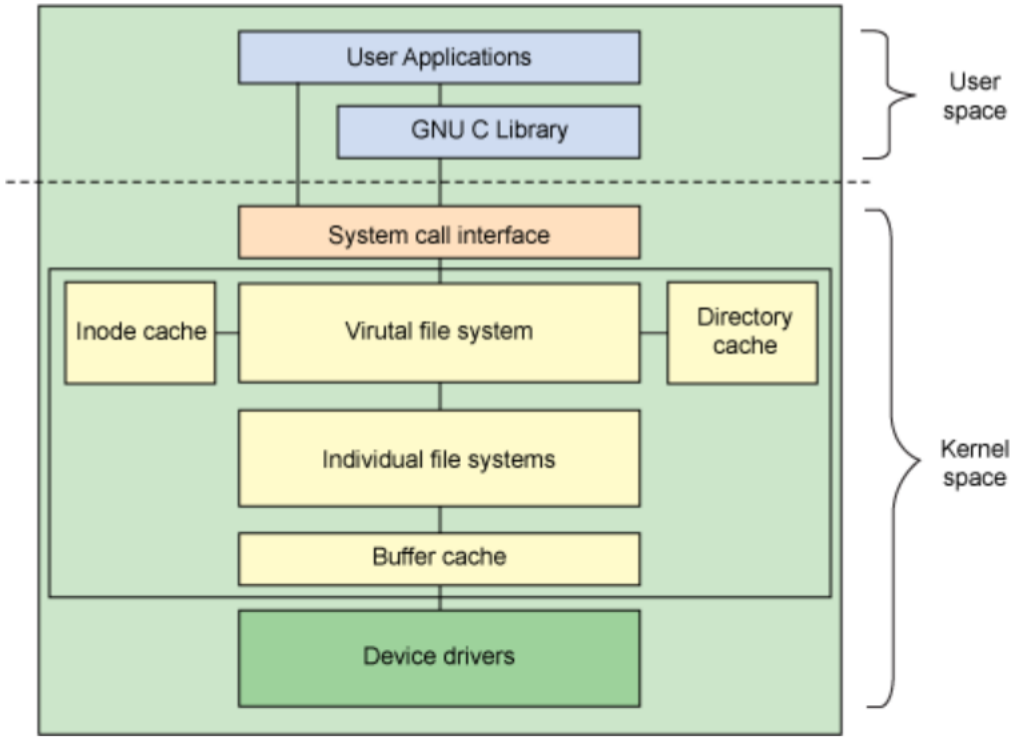
- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9th edition 2013 and some slides from 10th edition 2018
2. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9th edition 2018
3. Some presentation transcripts from A. Frank – P. Weisberg
4. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau

OPERATING SYSTEMS

Linux File System - Architecture

- User space contains the applications which provides the user interface for the file system calls (open, read, write, close).
- The system call interface acts as a switch, funneling system calls from user space to the appropriate endpoints in kernel space.
- The VFS is the primary interface to the underlying file systems. This component exports a set of interfaces and then abstracts them to the individual file systems, which may behave very differently from one another.
- Two caches exist for file system objects (inodes and dentries). Each provides a pool of recently-used file system objects.



Reference: https://developer.ibm.com/tutorials/l-linux-filesystem/

Linux File System - Architecture

- Each individual file system implementation, such as ext2, JFS, and so on, exports a common set of interfaces that is used by the VFS.
- The buffer cache buffers requests between the file systems and the block devices that they manipulate. For example, read and write requests to the underlying device drivers migrate through the buffer cache. This allows the requests to be cached there for faster access (rather than going back out to the physical device).
- The buffer cache is managed as a set of least recently used (LRU) lists. You can use the sync command to flush the buffer cache out to the storage media (force all unwritten data out to the device drivers and, subsequently, to the storage device).

OPERATING SYSTEMS

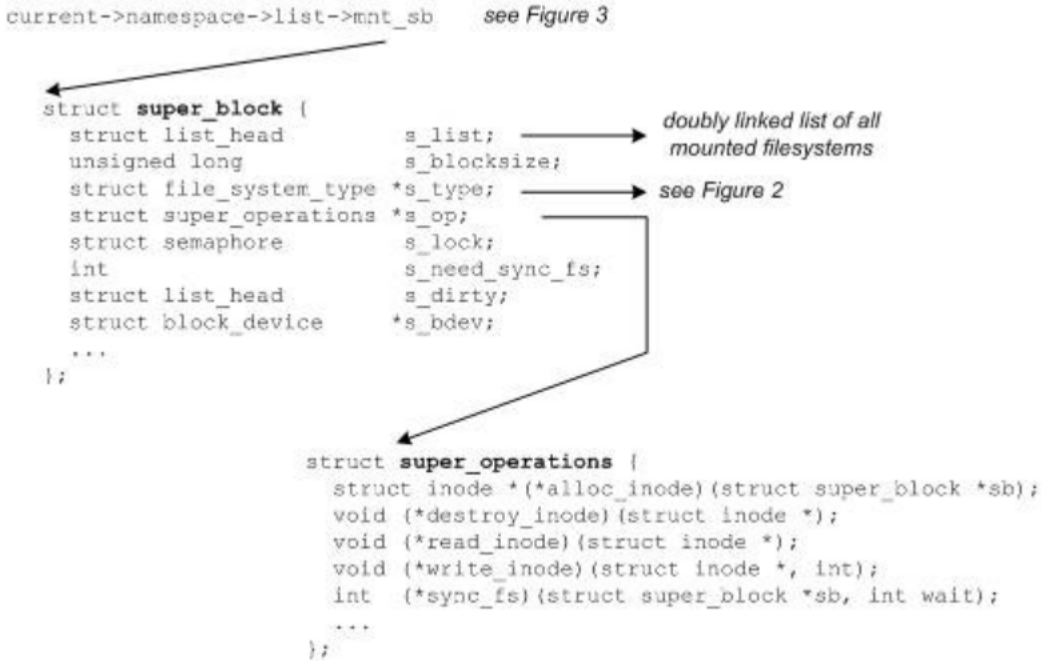
Linux File System – Common set of objects

- Linux views all file systems from the perspective of a common set of objects.
- These objects are the **superblock, inode, dentry,** and **file**.
- At the root of each file system is the superblock, which describes and maintains state for the file system.
- Every object that is managed within a file system (file or directory) is represented in Linux as an inode.
- The inode contains all the metadata to manage objects in the file system (including the operations that are possible on it).
- Another set of structures, called dentries, is used to translate between names and inodes, for which a directory cache exists to keep the most-recently used around.
- The dentry also maintains relationships between directories and files for traversing file systems.
- A VFS file represents an open file (keeps state for the open file such as the write offset, and so on).

OPERATING SYSTEMS

Linux File System – Superblock

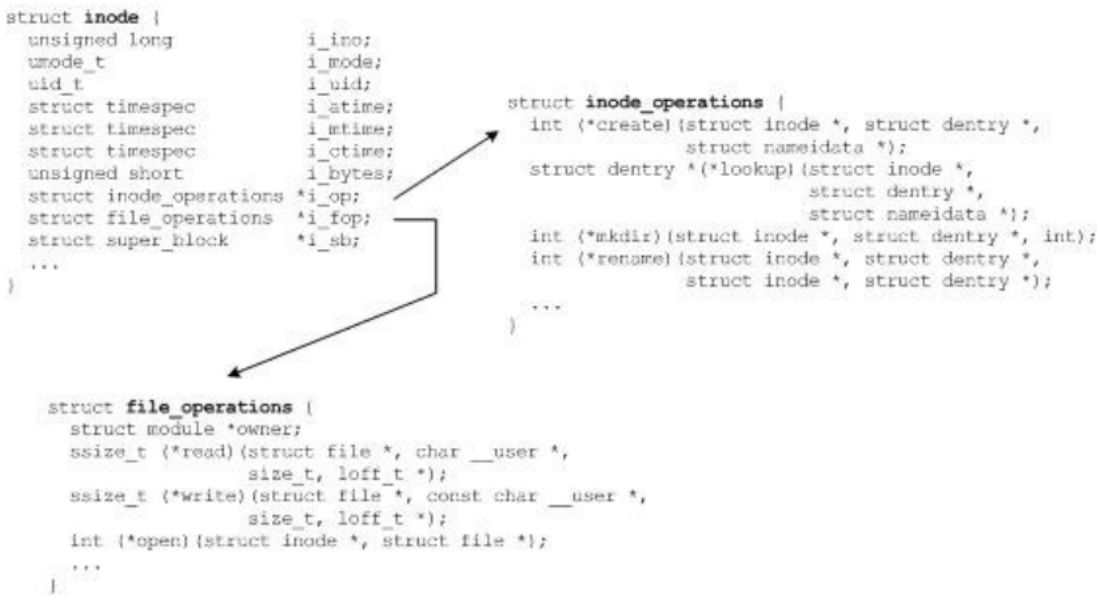
- The superblock is a structure that represents a file system.
- It includes the necessary information to manage the file system during operation.
- It includes the file system name (such as ext2), the size of the file system and its state, a reference to the block device, and metadata information (such as free lists and so on).
- The superblock is typically stored on the storage medium
- Superblock structure is available in ./linux/include/linux/fs.h.



OPERATING SYSTEMS

Linux File System – inode and dentry

- The inode represents an object in the file system with a unique identifier.
- The individual file systems provide methods for translating a filename into a unique inode identifier and then to an inode reference.
- inode_operations and file_operations structures refers to the individual operations that may be performed on the inode, file and directories



Reference: https://opensource.com/article/18/7/how-check-free-disk-space-linux

OPERATING SYSTEMS

Linux File System – Buffer Cache

- Buffer cache keeps track of read and write requests from the individual file system implementations and the physical devices (through the device drivers).
- For efficiency, Linux maintains a cache of the requests to avoid having to go back out to the physical device for all requests. Instead, the most-recently used buffers (pages) are cached here and can be quickly provided back to the individual file systems.
- Linux supports a wide range of file systems such as MINIX, MS-DOS, and ext2. Linux also supports the new journaling file systems such as ext3, JFS, and ReiserFS. Additionally, Linux supports cryptographic file systems such as CFS and virtual file system such as /proc.
- Linux also supports **Filesystem in USErspace** (**FUSE**) filesystem that lets non-privileged users create their own file systems without editing kernel code. This is achieved by running file system code in user space while the FUSE module provides only a bridge to the actual kernel interfaces

THANK YOU

Suresh Jamadagni
Department of Computer Science and Engineering

sureshjamadagni@pes.edu

OPERATING SYSTEMS

File Management

Suresh Jamadagni
Department of Computer Science

OPERATING SYSTEMS

File Management – Efficiency and Performance

Suresh Jamadagni
Department of Computer Science

OPERATING SYSTEMS

Slides Credits for all the PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9th edition 2013 and some slides from 10th edition 2018
2. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9th edition 2018
3. Some presentation transcripts from A. Frank – P. Weisberg
4. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau

Efficiency

Efficiency dependent on:

- Disk allocation and directory algorithms
- Types of data kept in file's directory entry
- Pre-allocation or as-needed allocation of metadata structures
- Fixed-size or varying-size data structures

Performance

Performance dependent on:

- Keeping data and metadata close together
- Buffer cache – separate section of main memory for frequently used blocks
- Synchronous writes sometimes requested by apps or needed by OS
  - No buffering / caching – writes must hit disk before acknowledgement
  - Asynchronous writes more common, buffer-able, faster
- Free-behind and read-ahead – techniques to optimize sequential access
- Reads frequently slower than writes

OPERATING SYSTEMS

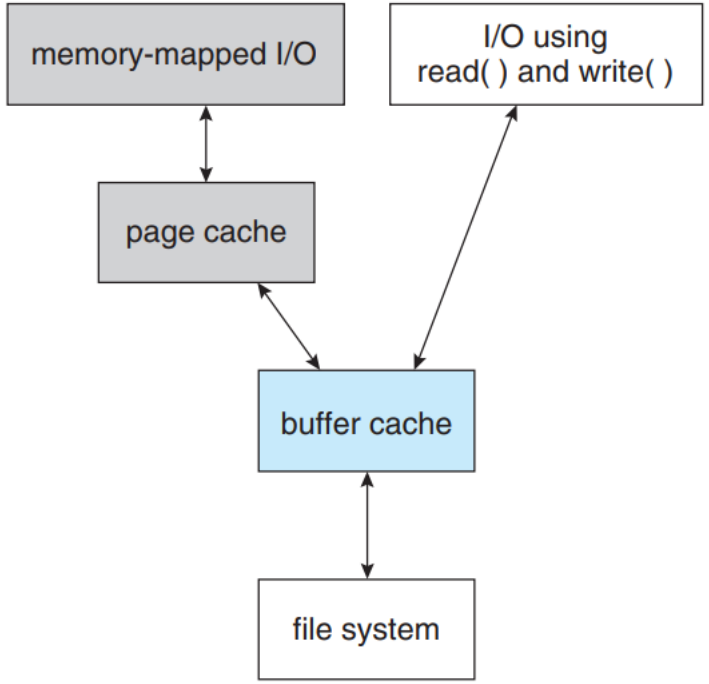
Page cache

- Some systems maintain a separate section of main memory for a **buffer cache**, where blocks are kept under the assumption that they will be used again shortly.
- Other systems **cache file data using a page cache**. The page cache uses virtual memory techniques to cache file data as pages rather than as file-system-oriented blocks.
- Caching file data using virtual addresses is far more efficient than caching through physical disk blocks, as accesses interface with virtual memory rather than the file system.
- Several systems—including Solaris, Linux, and Windows —use page caching to cache both process pages and file data. This is known as **unified virtual memory**.

OPERATING SYSTEMS

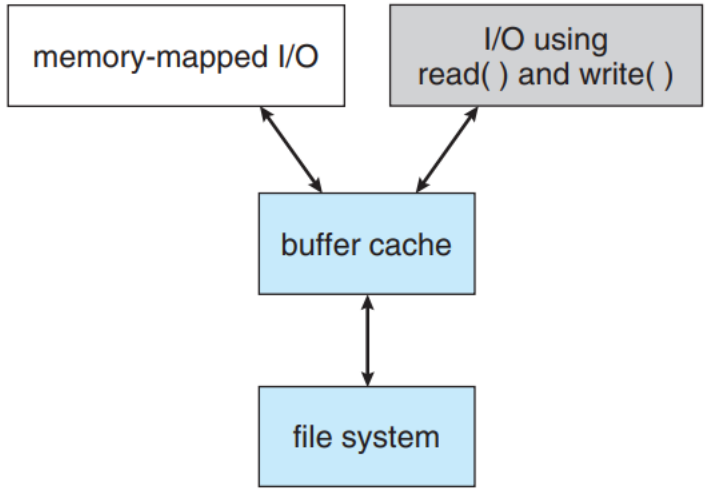
I/O without a unified buffer cache

- Consider standard system calls read() and write()
- Without a unified buffer cache, the read() and write() system calls go through the buffer cache.
- The memory-mapping call, requires two caches—the page cache and the buffer cache.
- A memory mapping proceeds by reading in disk blocks from the file system and storing them in the buffer cache. Because the virtual memory system does not interface with the buffer cache, the contents of the file in the buffer cache must be copied into the page cache.
- This situation, known as **double caching**, requires caching file-system data twice. Not only does it waste memory but it also wastes significant CPU and I/O cycles due to the extra data movement within system memory.



I/O using a unified buffer cache

- When a unified buffer cache is provided, both memory mapping and the read() and write() system calls use the same page cache.
- This has the benefit of avoiding double caching, and it allows the virtual memory system to manage file-system data



Synchronous and Asynchronous writes

- Another issue that can affect the performance of I/O is whether writes to the file system occur synchronously or asynchronously.
- Synchronous writes occur in the order in which the disk subsystem receives them, and the writes are not buffered. Thus, the calling routine must wait for the data to reach the disk drive before it can proceed.
- In an asynchronous write, the data are stored in the cache, and control returns to the caller.
- Most writes are asynchronous, metadata writes can be synchronous

OPERATING SYSTEMS

Page replacement

- Sequential access can be optimized by techniques known as free-behind and read-ahead.
- **Free-behind** removes a page from the buffer as soon as the next page is requested. The previous pages are not likely to be used again and waste buffer space.
- With **read-ahead**, a requested page and several subsequent pages are read and cached. These pages are likely to be requested after the current page is processed.
- Retrieving these data from the disk in one transfer and caching them saves a considerable amount of time

OPERATING SYSTEMS

Disk cache

- When data is written to a disk file, the pages are buffered in the disk cache, and the disk driver sorts its output queue according to disk address to minimize disk-head seeks
- Unless synchronous writes are required, a process writing to disk simply writes into the cache, and the system asynchronously writes the data to disk when convenient
- The user process sees very fast writes.
- When data are read from a disk file, the block I/O system does some read-ahead
- Writes are much more nearly asynchronous than are reads. Thus, output to the disk through the file system is often faster than is input for large transfers, counter to intuition

THANK YOU

Suresh Jamadagni
Department of Computer Science Engineering

sureshjamadagni@pes.edu

OPERATING SYSTEMS

Storage Management

Chandravva Hebbi
Department of Computer Science

OPERATING SYSTEMS

Mass-Storage Structure

Chandravva Hebbi
Department of Computer Science

OPERATING SYSTEMS

Slides Credits for all PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

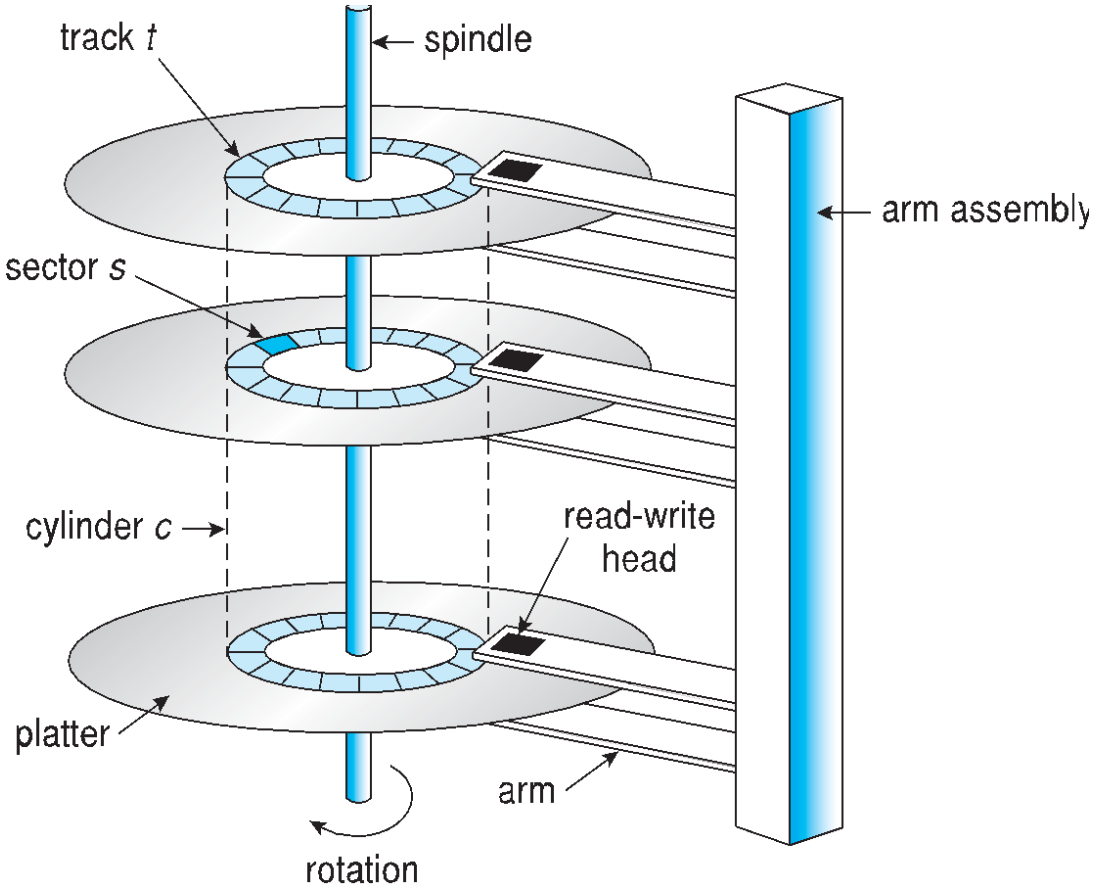
1. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9th edition 2013 and some slides from 10th edition 2018
2. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9th edition 2018
3. Some presentation transcripts from A. Frank – P. Weisberg
4. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau

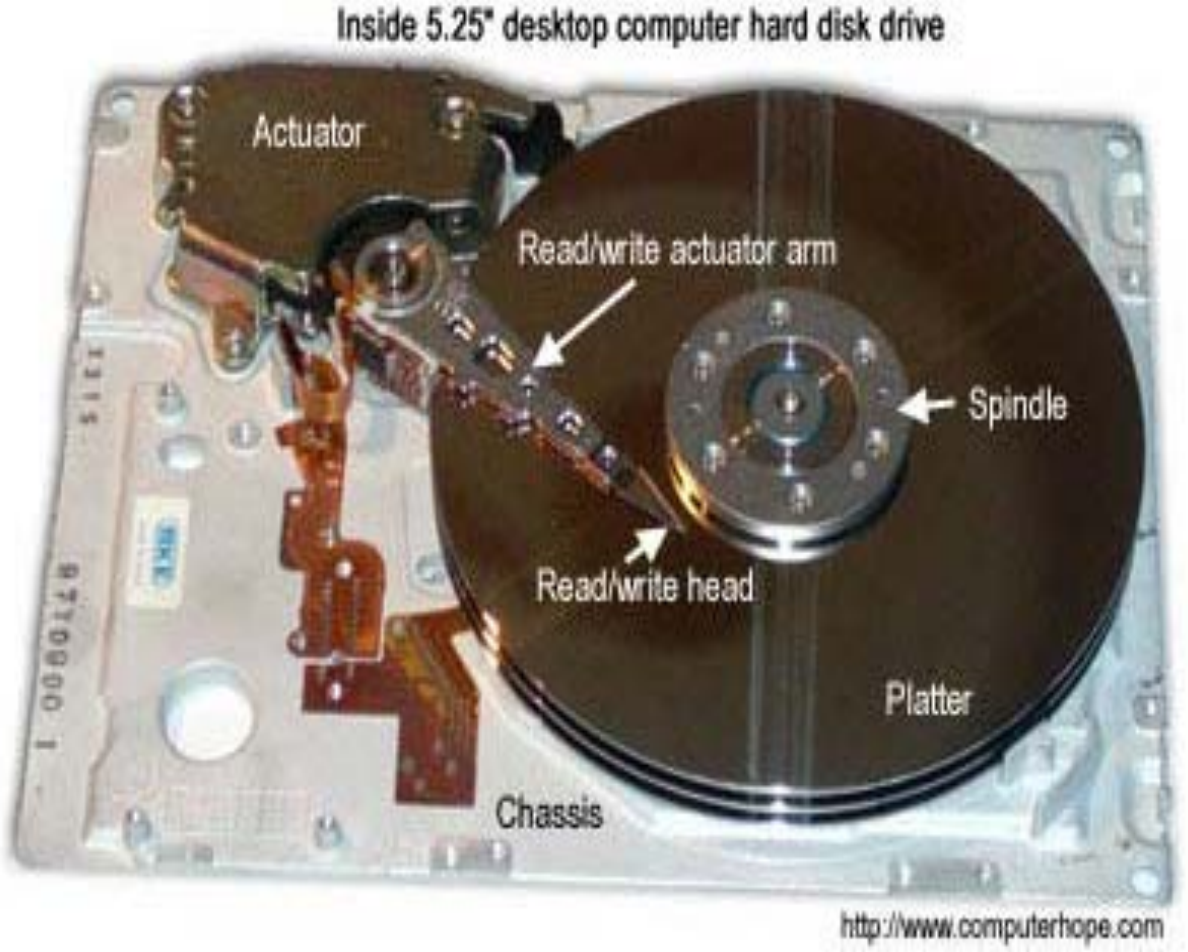
Overview of Mass Storage Structure

- **Magnetic disks** provide bulk of secondary storage of modern computers.
- Each disk platter has a flat circular shape, like a CD. Common platter diameters range from 1.8 to 3.5 inches.
- The two surfaces of a platter are covered with a magnetic material.
- The heads are attached to a disk arm that moves all the heads as a unit.
- The surface of a platter is logically divided into circular tracks, which are subdivided into sectors.
- The set of tracks that are at one arm position makes up a cylinder.

OPERATING SYSTEMS

Moving-head Disk Mechanism





Overview of Mass Storage Structure

- When the disk is in use, a drive motor spins it at high speed. Most drives rotate 60 to 250 times per second.

Disk speed has two parts.

- The transfer rate is the rate at which data flow between the drive and the computer.
- The positioning time, or random-access time, consists of two parts
  - The time necessary to move the disk arm to the desired cylinder, called the **seek time**.
  - the time necessary for the desired sector to rotate to the disk head, called the **rotational latency**.

Disk Latency = Seek time + Rotational latency + Transfer time

OPERATING SYSTEMS

Magnetic Disk Specifications – Reference: www.seagate.com

Specifications	Barracuda®	Constellation®	Constellation ES	SV35 Series™
Primary Applications	Optimised for PC and personal external storage	Optimised for 2.5-inch business server and external storage arrays	Optimised for 3.5-inch business server and external storage arrays	Optimised for video surveillance applications
Capacity (GB)	250, 320, 500, 750, 1,000 1,500, 2,000, 3,000	250, 500, 1,000	500, 1,000, 2,000, 3,000	1,000, 2,000, 3,000
Spin Speed (RPM)	7,200	7,200	7,200	7,200
SATA interface (Gb/s)	1.5/3.0/6.0	1.5/3.0/6.0	1.5/3.0/6.0	1.5/3.0/6.0
SAS interface (Gb/s)	—	3.0/6.0	3.0/6.0	—
Rotational vibration (RV) (radians/s/s)	5.5 narrow spectrum up to 300Hz	16 broad spectrum up to 1,800Hz	12.5 broad spectrum up to 1,500Hz	5.5 narrow spectrum up to 300Hz
Seek time, average read/write (ms)	<8.5/<9.5	8.5/9.5	8.5/9.5	<8.5/<9.5
Cache (MB)[1]	16, 64	Up to 64	Up to 64	64
Non-recoverable read errors per bits read	1 sector per 10^{14}	1 sector per 10^{15}	1 sector per 10^{15}	1 sector per 10^{14}
Power-on hours (POH)	2,400 – 8x5	8,760 - 24x7	8,760 - 24x7	8,760- 24x7
Streaming capabilities	—	Multiple sequential streams	Multiple sequential streams	Up to 20 simultaneous HD streams[3]
POH usage profile	8x5 – On as needed	24x7 – Always on	24x7 – Always on	Up to 64 cameras 24x7 – Always on
MTBF (hours)	700,000	1.4 million	1.2 million	1 million
Power, average – idle (W)[2]	4.6	2.25 to 3.85	>3.74	—
Power, average – Idle2 (W)[2]	3.4 to 5.4	—	—	3.4 to 5.4
Acoustics, typical – idling (bels)	2.2 to 2.4	2.2	1.9 to 2.7	2.2 to 2.4
Shock, operating/non-operating (Gs)	70 to 80/300 to 350	70/400	40 to 70/300	80/300 to 350
Ambient temperature, operating/ non-operarting (°C)	0 to 60/–40 to 70	5 to 60/–40 to 70	5 to 60/–40 to 70	0 to 70/–40 to 70
RAID support	0, 1	0, 1, 3, 4, 5, 6, 10	0, 1, 3, 4, 5, 6, 10	0, 1, 3, 4, 5, 6, 10
RAID Rebuild™ support		•	•	
Enterprise expert support		•	•	•

Overview of Mass Storage Structure

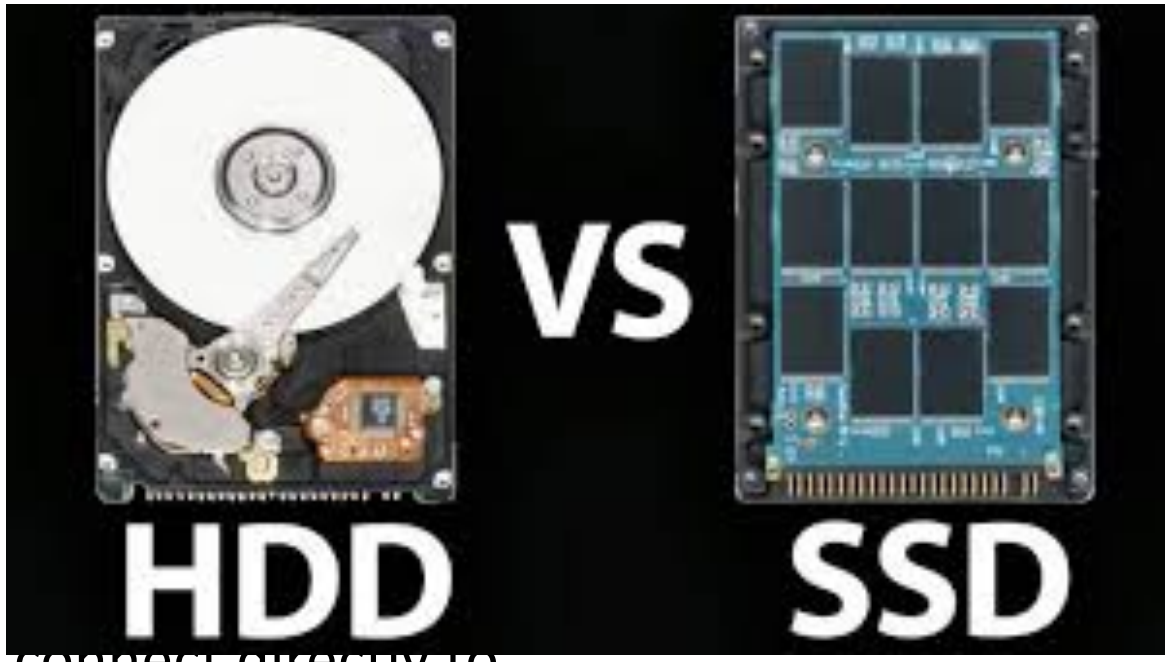
- Disk head flies on an extremely thin cushion of air.
- head will sometimes damage the magnetic surface when it makes contact with it.
  - Head crash
  - Not recoverable
- A disk can be **removable.**
- Removable magnetic disks generally consist of one platte
- Examples CDs, DVDs, and Blu-ray discs as well as removable flash-memory

Mass Storage Structure (Cont.)

- Drive attached to computer via **I/O bus**
  - Busses vary, including **EIDE**, **ATA**, **SATA**, **USB**, **Fiber Channel**, **SCSI, SAS, Firewire**
    - SCSI is a set of parallel interface standards developed by ANSI
    - allows more hard disks per computer compared to IDE
    - SCSI is harder to configure compared to SATA and IDE
    - EIDE has a 133 megabytes per second speed rate, while SATA has up to a 150 megabytes per second speed rate.
  - **Host controller** in computer uses bus to talk to **disk controller** built into drive or storage array

Solid-State Disks

- Nonvolatile memory used like a hard drive
  - Many technology variations
- Can be more reliable than HDDs
- More expensive per MB
- Maybe have shorter life span
- Less capacity
- But much faster
- Standard Bus interfaces can be too slow -> connect directly to the system bus (PCI, for example)
- No moving parts, so no seek time or rotational latency



Magnetic Tape

- Was early secondary-storage medium
  - Evolved from open spools to cartridges
- Relatively permanent and holds large quantities of data
- Access time slow
- Random access ~1000 times slower than disk
- Mainly used for backup, storage of infrequently-used data, transfer medium between systems

Magnetic Tape (Cont.)

- Kept in spool and wound or rewound past read-write head
- Moving to the correct spot on a tape can take minutes, but once positioned, tape drives can write data at speeds comparable to disk drives.
- Tape capacities vary greatly, depending on the particular kind of tape drive, with current capacities exceeding several terabytes.
- Some tapes have built-in compression that can more than double the effective storage.
- Tapes and their drivers are usually categorized by width, including 4, 8, and 19 millimeters and 1/4 and 1/2 inch.
- Some are named according to technology, such as LTO-5 and SDLT.

THANK YOU

Chandravva Hebbi
Department of Computer Science Engineering

chandravvahebbi@pes.edu

OPERATING SYSTEMS

Storage Management

Chandravva Hebbi
Department of Computer Science

OPERATING SYSTEMS

Mass-Storage Structure – Disk Scheduling

FCFS, SSTF, SCAN, C-SCAN, LOOK

Chandravva Hebbi
Department of Computer Science

OPERATING SYSTEMS

Slides Credits for all PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9th edition 2013 and some slides from 10th edition 2018
2. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9th edition 2018
3. Some presentation transcripts from A. Frank – P. Weisberg
4. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau

Disk Scheduling

- The operating system is responsible for using hardware efficiently — for the disk drives, this means having a fast access time and disk bandwidth
- Minimize seek time
- Seek time ≈ seek distance
- Disk **bandwidth** is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer

OPERATING SYSTEMS

Disk Scheduling (Cont.)

- There are many sources of disk I/O request
  - OS
  - System processes
  - Users processes
- I/O request includes input or output mode, disk address, memory address, number of sectors to transfer
- OS maintains queue of requests, per disk or device
- Idle disk can immediately work on I/O request, busy disk means work must queue
  - Optimization algorithms only make sense when a queue exists

Disk Scheduling (Cont.)

- Note that drive controllers have small buffers and can manage a queue of I/O requests (of varying “depth”)
- Several algorithms exist to schedule the servicing of disk I/O requests
- The analysis is true for one or many platters
- We illustrate scheduling algorithms with a request queue (0-199)

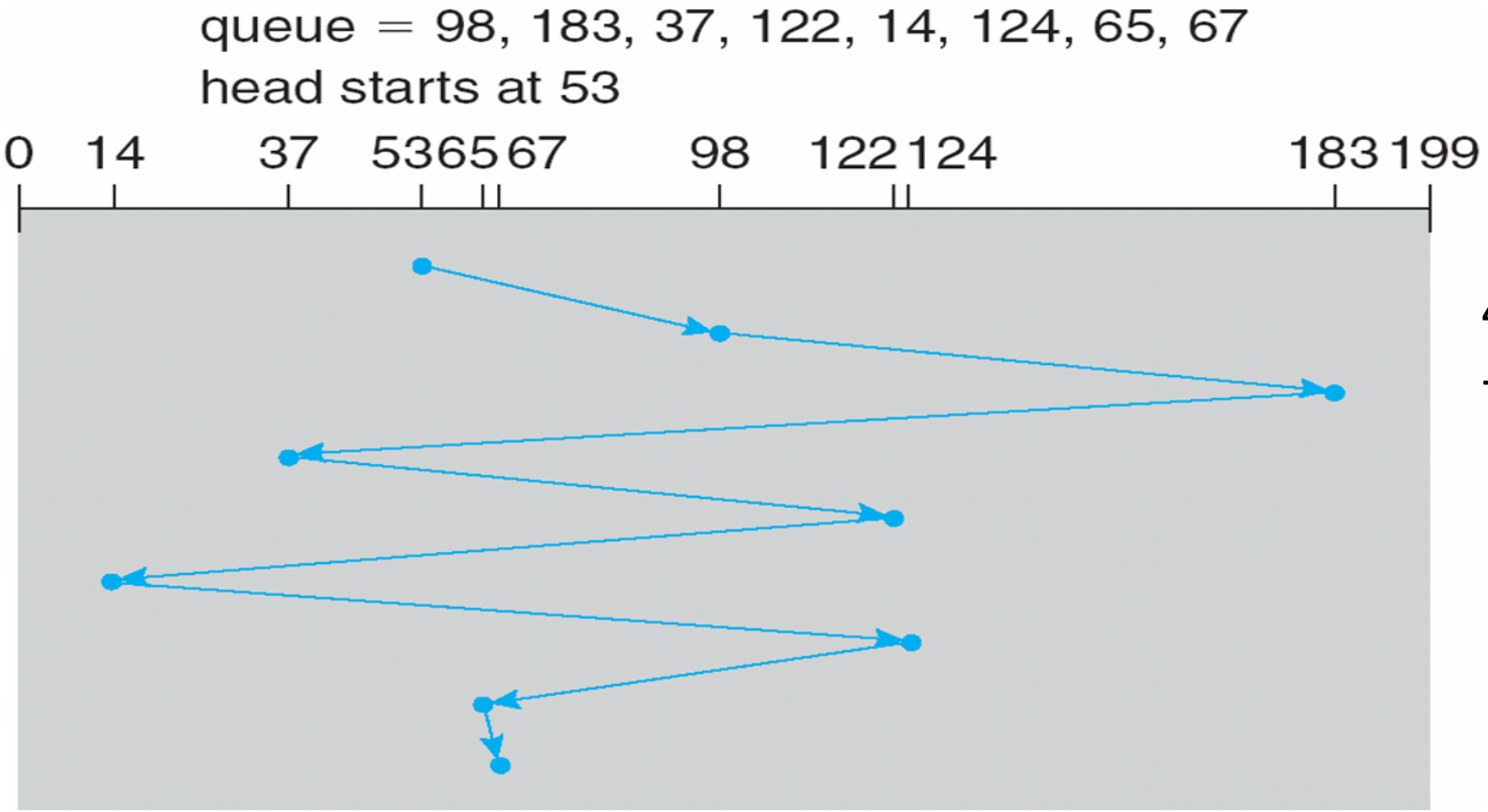
98, 183, 37, 122, 14, 124, 65, 67

Head pointer 53

OPERATING SYSTEMS

FCFS

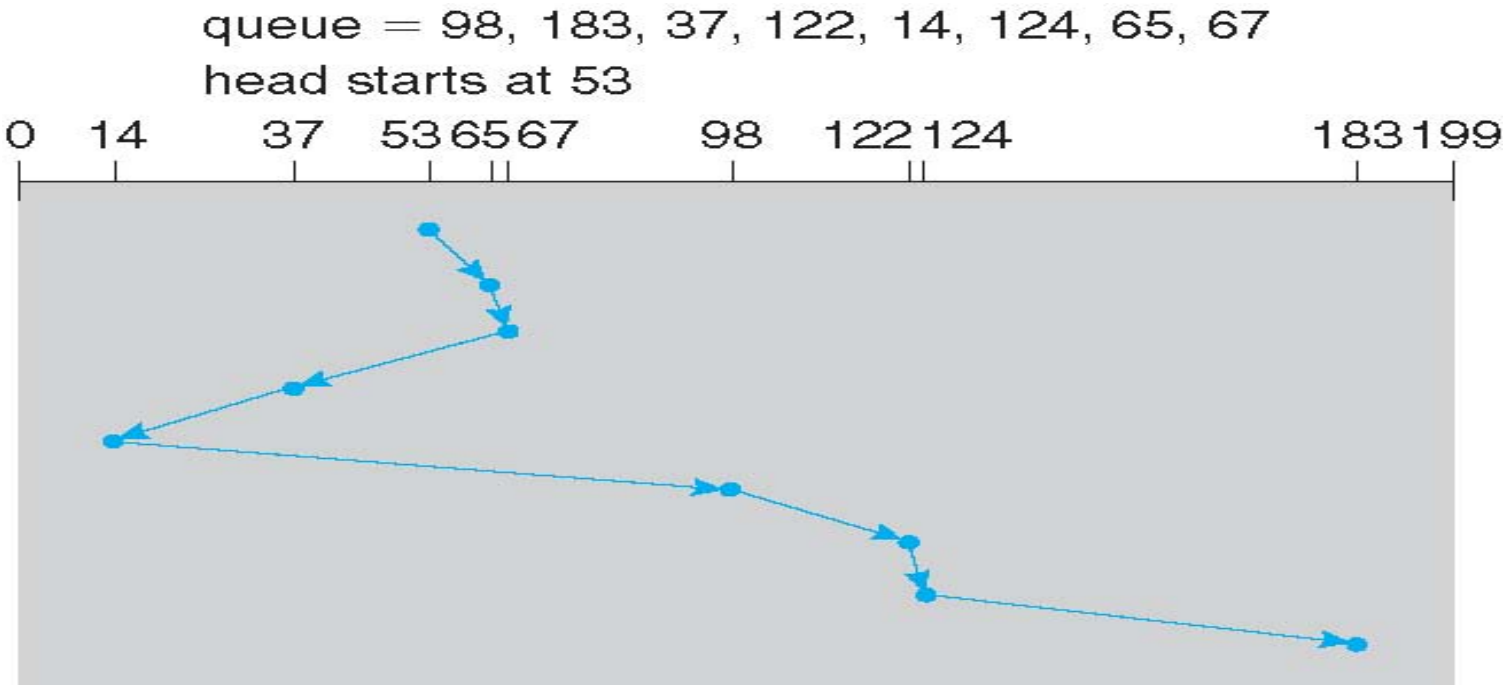
Illustration shows total head movement of 640 cylinders



45+85+146+85+108+110+59
+2 = 640 cylinders

SSTF

- Shortest Seek Time First selects the request with the minimum seek time from the current head position
- SSTF scheduling is a form of SJF scheduling; may cause starvation of some requests
- Illustration shows total head movement of 236 cylinders

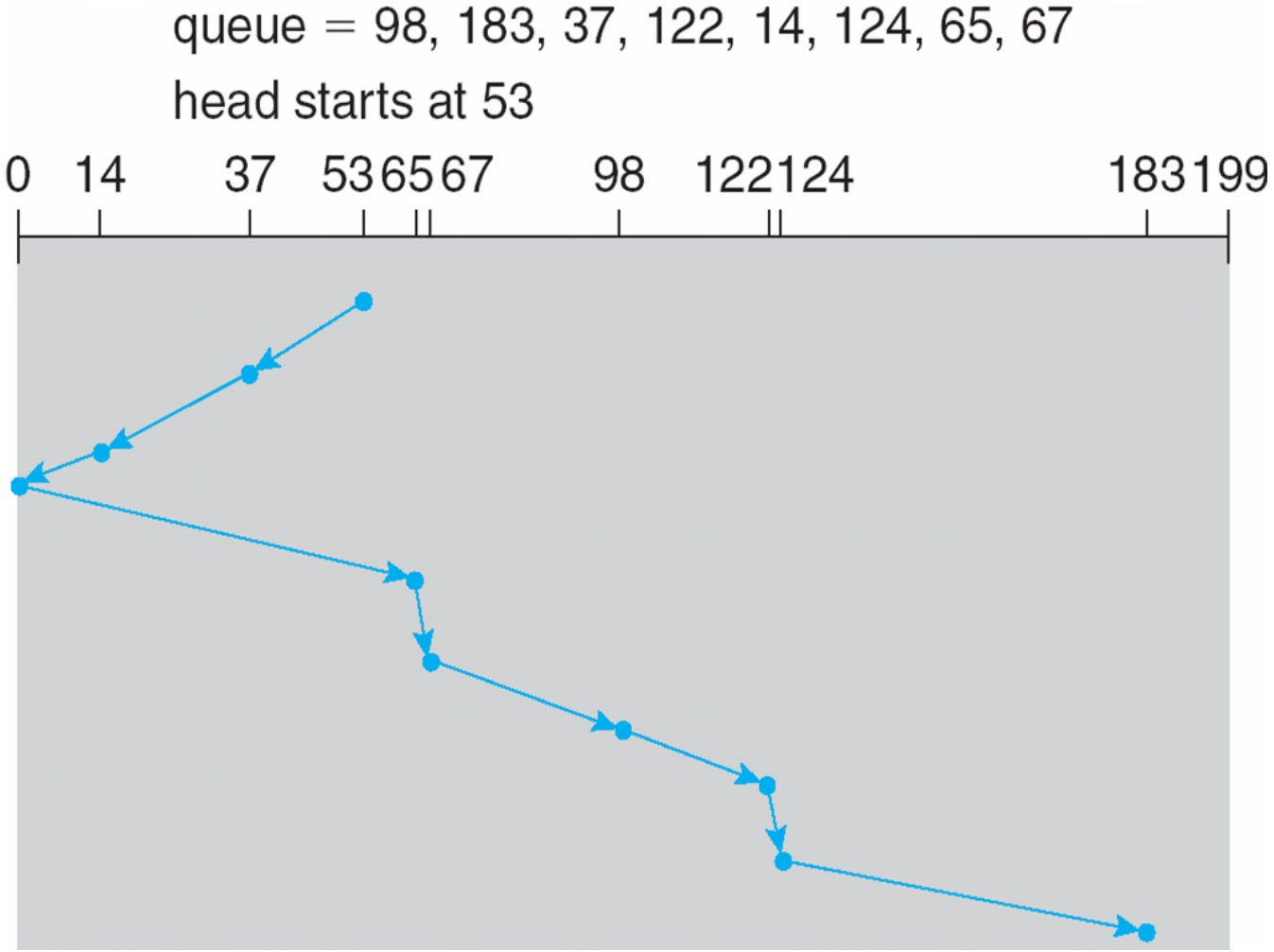


12+2+30+23+84+24+2+59 = 236 cylinders

SCAN

- The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues.
- **SCAN algorithm** Sometimes called the **elevator algorithm**
- But note that if requests are uniformly dense, largest density at other end of disk and those wait the longest

SCAN (Cont.)



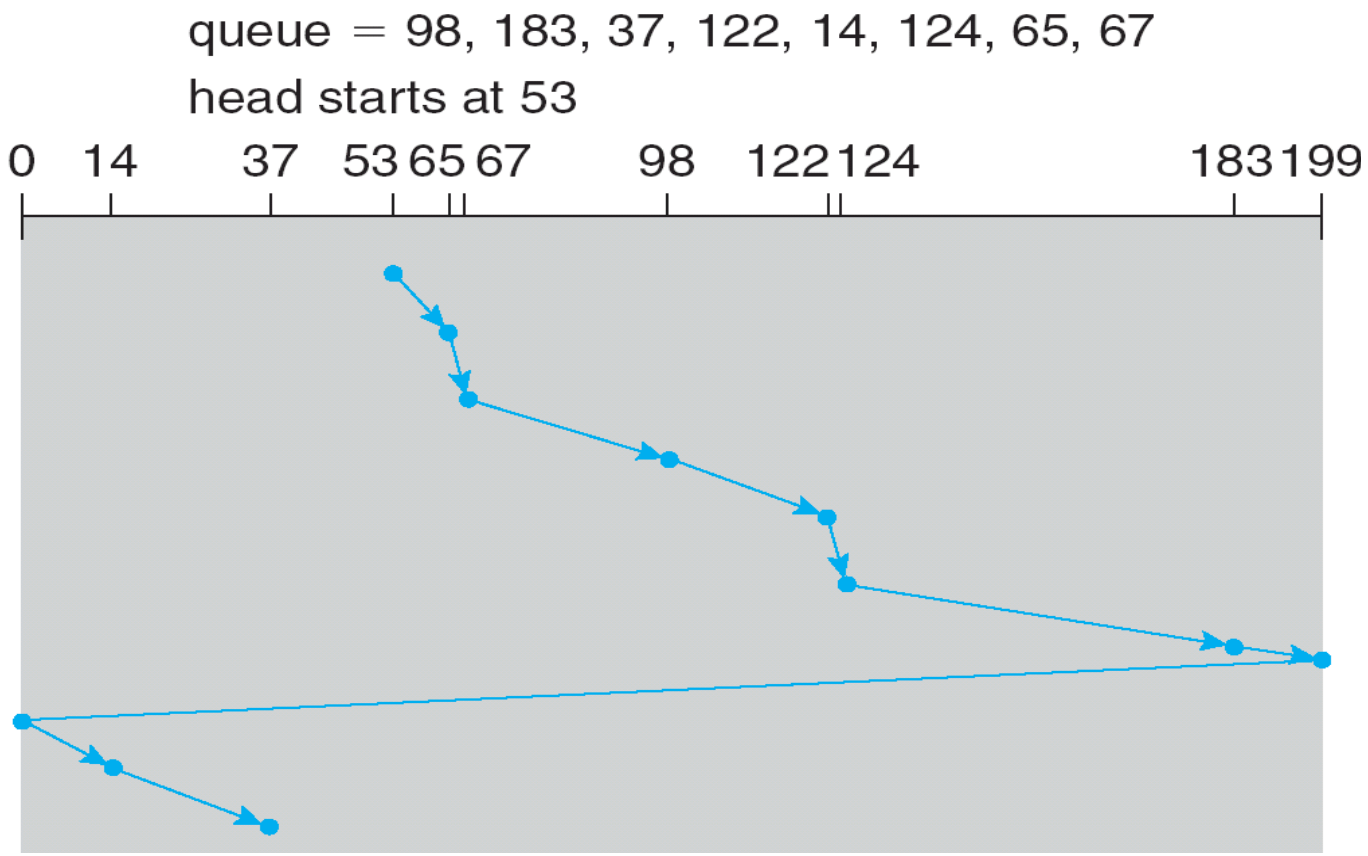
Number of cylinder moves=
16+23+79+2+31+24+2+59=236

C-SCAN

- Provides a more uniform wait time than SCAN
- The head moves from one end of the disk to the other, servicing requests as it goes
  - When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip
- Treats the cylinders as a circular list that wraps around from the last cylinder to the first one
- Total number of cylinders?

OPERATING SYSTEMS

C-SCAN (Cont.)



= (65-53)+(67-65)+(98-67)+(122-98)+(124-122)+(183-124)+(199-183)+(199-0)+(14-0)+(37-14)

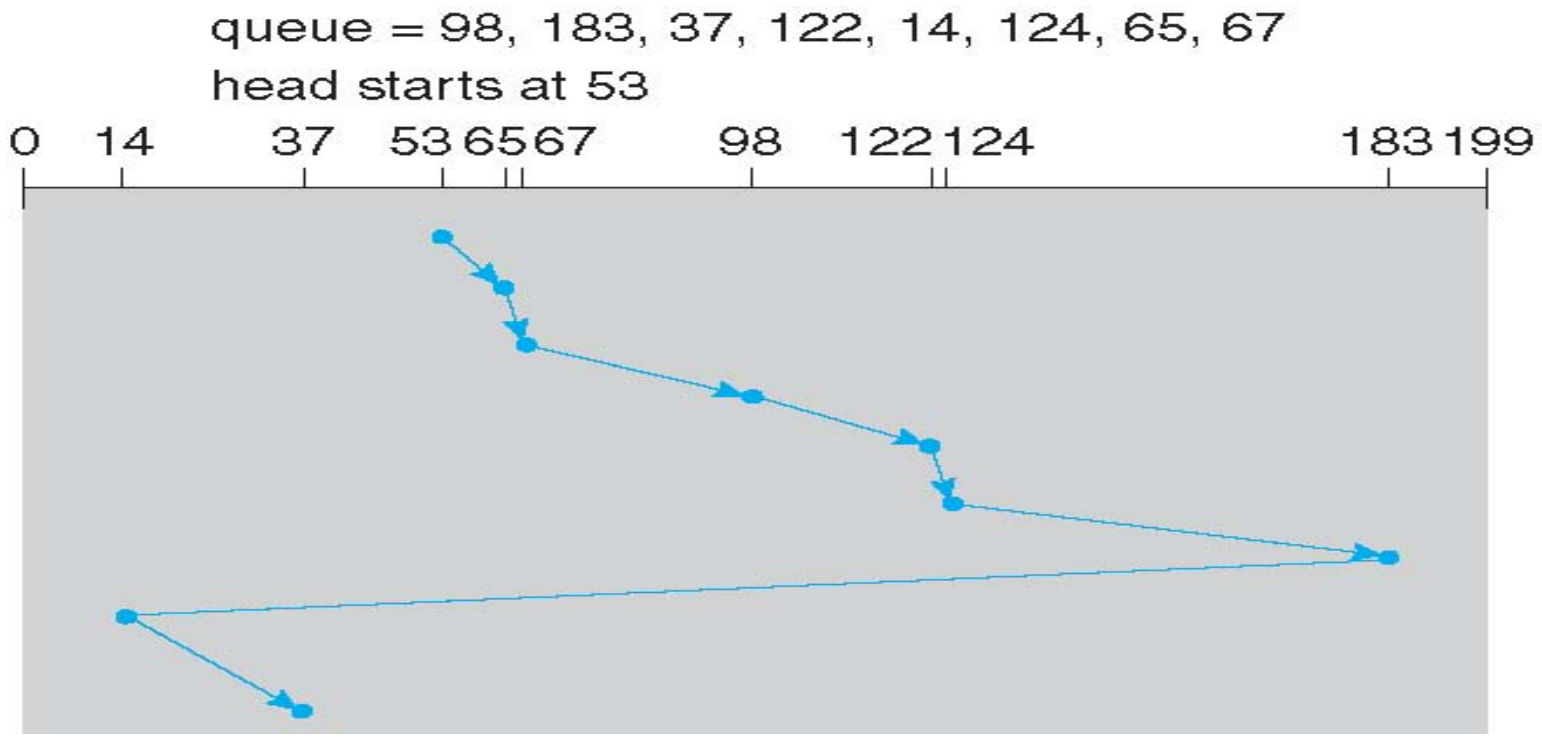
=12+2+31+24+2+59+16+199+14+23

= 382

LOOK & C-LOOK

- LOOK a version of SCAN, C-LOOK a version of C-SCAN
- Arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk
- Total number of cylinders?

C-LOOK (Cont.)



Total head movements incurred while servicing these requests

= (65 – 53) + (67 – 65) + (98 – 67) + (122 – 98) + (124 – 122) + (183 – 124) + (183 – 14) + (37 – 14)

= 12 + 2 + 31 + 24 + 2 + 59 + 169 + 23

= 322

Selecting a Disk-Scheduling Algorithm

- SSTF is common and has a natural appeal
- SCAN and C-SCAN perform better for systems that place a heavy load on the disk
  - Less starvation
- Performance depends on the number and types of requests
- Requests for disk service can be influenced by the file-allocation method
  - And metadata layout
- The disk-scheduling algorithm should be written as a separate module of the operating system, allowing it to be replaced with a different algorithm if necessary
- Either SSTF or LOOK is a reasonable choice for the default algorithm

Selecting a Disk-Scheduling Algorithm (Cont.)

- What about rotational latency?
  - Difficult for OS to calculate
  - The rotational latency can be nearly as large as the average seek time.
  - It is difficult for the operating system to schedule for improved rotational latency, though, because modern disks do not disclose the physical location of logical blocks.
  - Disk manufacturers have been alleviating this problem by implementing disk-scheduling algorithms in the controller hardware built into the disk drive.
- How does disk-based queueing effect OS queue ordering efforts?
  - If the OS sends a batch of requests to the controller, the controller can queue them and then schedule them to improve both the seek time and the rotational latency.

THANK YOU

Chandravva Hebbi
Department of Computer Science Engineering

chandravvahebbi@pes.edu

OPERATING SYSTEMS

Storage Management

Chandravva Hebbi
Department of Computer Science

OPERATING SYSTEMS

Mass-Storage Structure – Swap Space and RAID

Chandravva Hebbi
Department of Computer Science

OPERATING SYSTEMS

Slides Credits for all PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9th edition 2013 and some slides from 10th edition 2018
2. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9th edition 2018
3. Some presentation transcripts from A. Frank – P. Weisberg
4. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau

Swap-Space Management

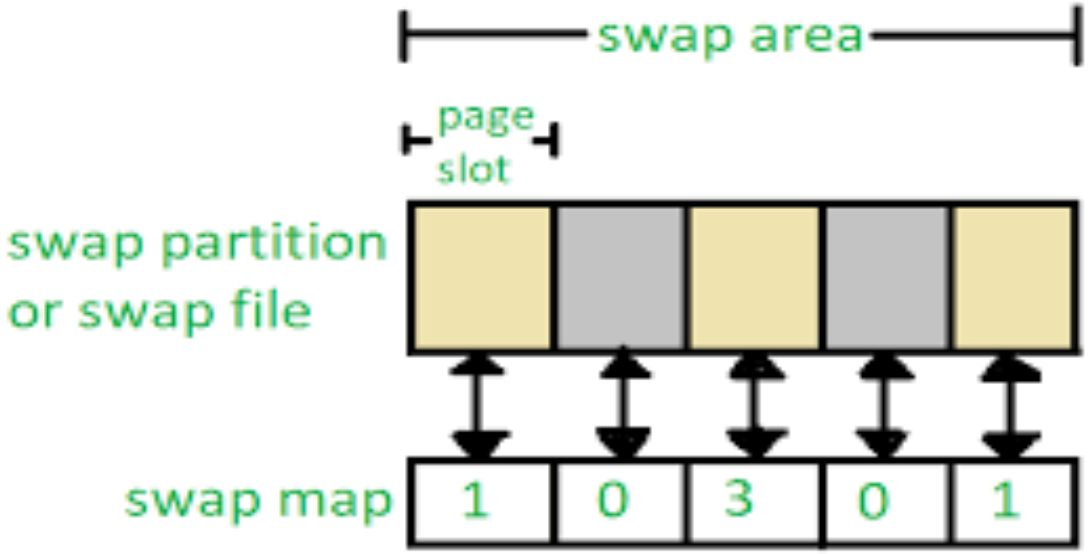
- Swap-space — Virtual memory uses disk space as an extension of main memory
  - Less common now due to memory capacity increases
- Swap-space can be carved out of the normal file system, or, more commonly, it can be in a separate disk partition (raw)
- Swap-space management
  - 4.3BSD allocates swap space when process starts; holds text segment (the program) and data segment
  - Kernel uses **swap maps** to track swap-space use
  - Some systems allow the use of multiple swap spaces – both files and dedicated swap partitions

Swap-Space Management (Cont.)

- Solaris 2 allocates swap space only when a dirty page is forced out of physical memory, not when the virtual memory page is first created
  - File data written to swap space until write to file system requested
  - Other dirty pages go to swap space due to no other home
  - Text segment pages thrown out and reread from the file system as needed
- What if a system runs out of swap space?
- Some systems allow multiple swap spaces

Data Structures for Swapping on Linux Systems

- Linux allows one or more swap areas to be established.
- A swap area may be in either a swap file on a regular file system o a dedicated swap partition.
- Each swap area consists of a series of 4-KB **page slots**, which are used to hold swapped pages.
- Associated with each swap area is a **swap map**—an array of integer counters, each corresponding to a page slot in the swap area.
  - 0 => page slot is available
  - 3 => swapped page is mapped to 3 different processes



RAID Basics

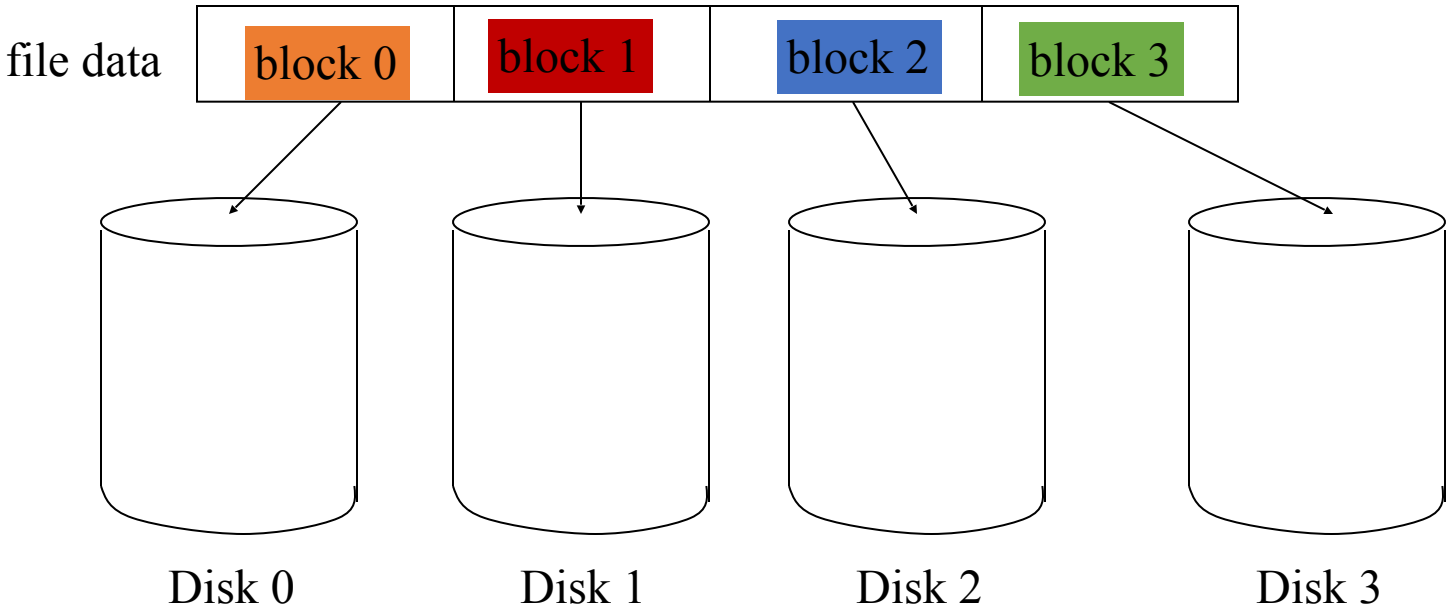
- RAID is a Disk Organization technique
  - Addresses performance and reliability issues
  - Multiple disk drives provide (or improve) reliability via **redundancy**
- Many systems today need to store many terabytes of data
- Don't want to use single, large disk
  - too expensive
  - failures could be catastrophic
- Would prefer to use many smaller disks
- **Redundant Array of Independent (inexpensive) Disks**

RAID Basics (Cont.)

- Basic idea is to connect multiple disks together to provide
  - large storage capacity
  - faster access to reading data
  - redundant data
- Many different levels of RAID systems
  - differing levels of redundancy, error checking, capacity, and cost

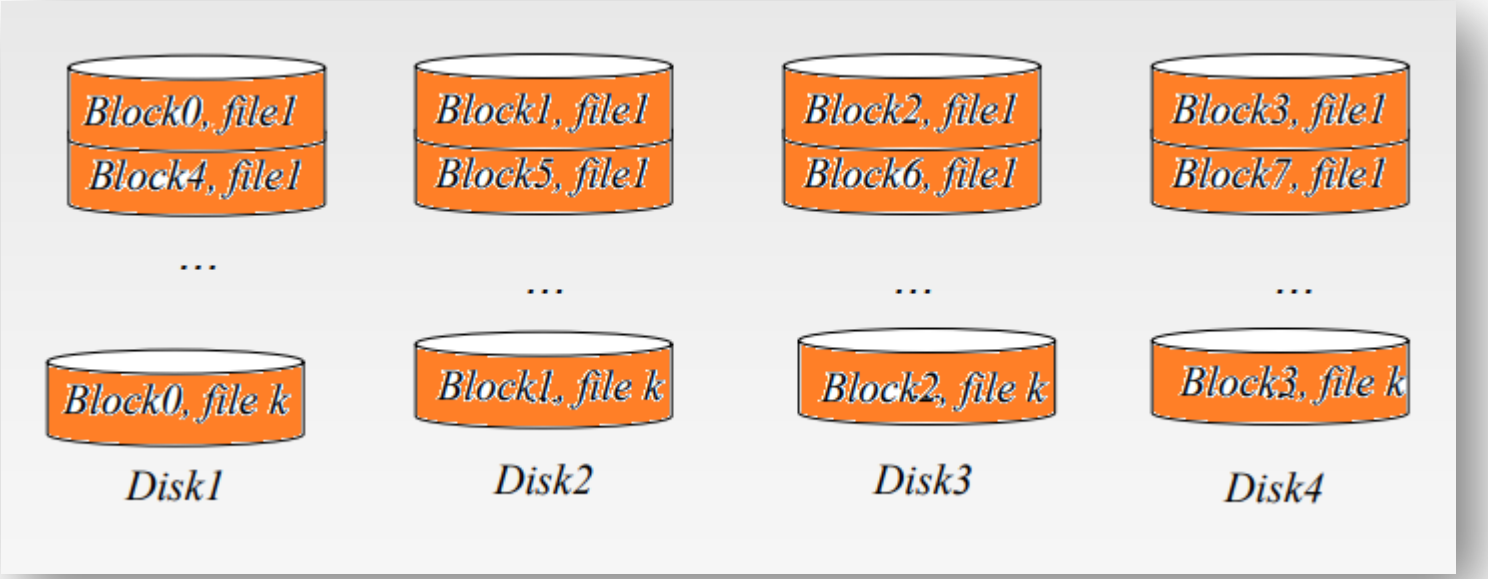
Striping

- Take file data and map it to different disks
- Allows for reading data in parallel
- Transfer rate is improved by striping data across the disks
  - ➢ **Bit-level striping and block-level striping (most common)**



Block-level Striping

- With n disks, block i of a file goes to disk (i mod n) + 1
  - Requests for different blocks can run in parallel if the blocks reside on different disks
  - A request for a long sequence of blocks can utilize all disks in parallel



Mirroring

- Keep two copies of data on two separate disks
- Gives good error recovery
  - if some data is lost, get it from the other source
- Expensive
  - requires twice as many disks
- Write performance can be slow
  - have to write data to two different spots
- Read performance is enhanced
  - can read data from file in parallel

RAID Structure

- **Mean time to failure**
  - If MTTF of a single disk is 100,000 hours, MTTF of some disk in an array of 100 disks = 100000/100 = 1000 hours or 41.66 days
- **Mean time to repair –** time taken to replace a failed disk and to restore the data on it
- **Mean time to data loss** based on above factors
  - If mirrored disks fail independently (i.e., not related to power failures and natural disasters), consider disk with MTTF of 100,000 hours and MTTR is 10 hours
  - Mean time to data loss of a mirrored disk system is $100,000^2 / (2 * 10) = 500 * 10^6$ hours, or 57,000 years!

RAID (Cont.)

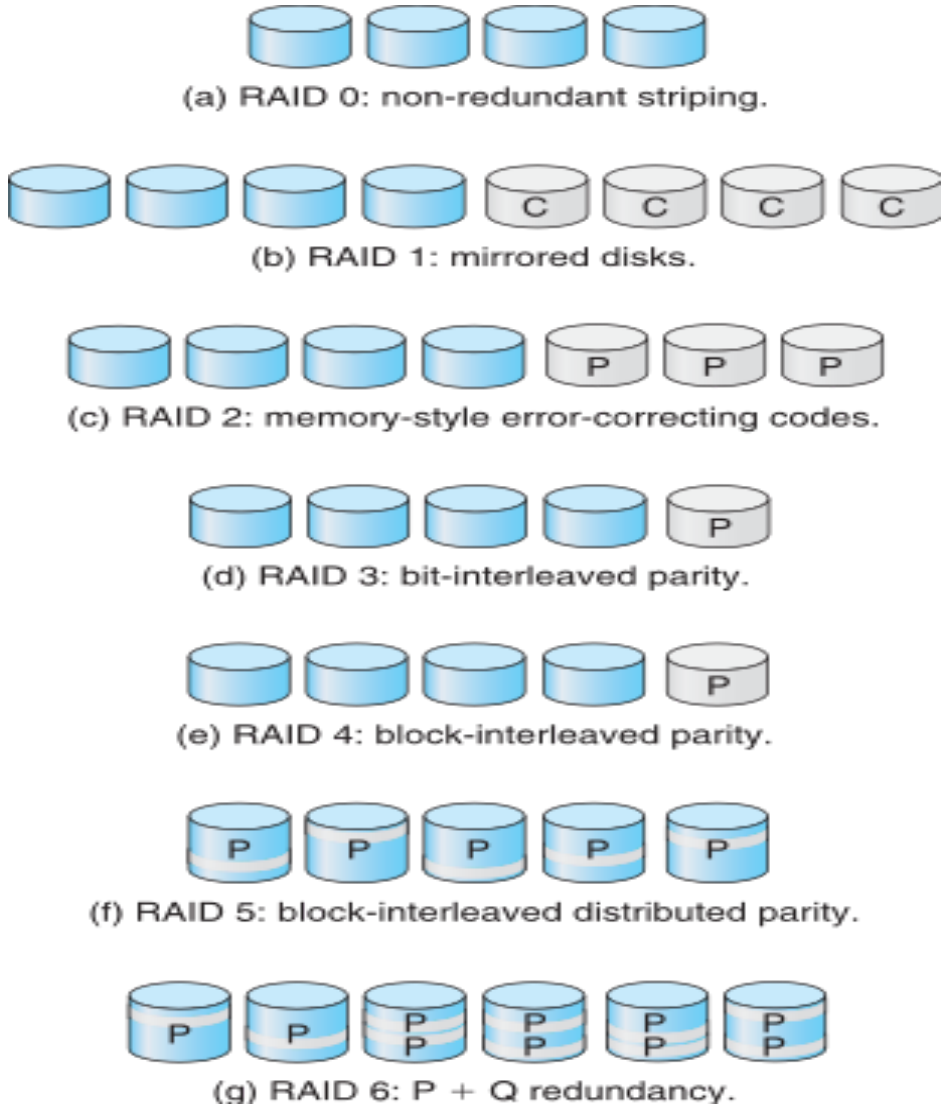
- Frequently combined with **NVRAM** to improve write performance
- Several improvements in disk-use techniques involve the use of multiple disks working cooperatively
- Disk **striping** uses a group of disks as one storage unit
- RAID is arranged into six different levels
- RAID schemes improve performance and improve the reliability of the storage system by storing redundant data
  - **Mirroring** or **shadowing** (**RAID 1**) keeps duplicate of each disk
  - Striped mirrors (**RAID 1+0**) or mirrored stripes (**RAID 0+1**) provides high performance and high reliability
  - **Block interleaved parity** (**RAID 4, 5, 6**) uses much less redundancy

RAID (Cont.)

- RAID within a storage array can still fail if the array fails, so automatic **replication** of the data between arrays is common
- Frequently, a small number of **hot-spare** disks are left unallocated, automatically replacing a failed disk and having data rebuilt onto them

OPERATING SYSTEMS

RAID Levels



(a) RAID 0: non-redundant striping.

(b) RAID 1: mirrored disks.

(c) RAID 2: memory-style error-correcting codes.

(d) RAID 3: bit-interleaved parity.

(e) RAID 4: block-interleaved parity.

(f) RAID 5: block-interleaved distributed parity.

(g) RAID 6: P + Q redundancy.

P indicates error-correcting bits and C indicates a second copy of the data

P + Q redundancy scheme uses 2 parity values P and Q

OPERATING SYSTEMS

RAID Levels 0 and 1

- **RAID level 0** refers to disk arrays with striping at the level of blocks
  - No redundancy (such as mirroring or parity bits)
  - lots of disks means low Mean Time To Failure (MTTF)

(a) RAID 0: non-redundant striping.

- **RAID level 1** refers to disk mirroring.
  - A complete file is stored on a single disk
  - A second disk contains an exact copy of the file
  - Provides complete redundancy of data
  - Read performance can be improved
  - file data can be read in parallel
  - Write performance suffers
  - must write the data out twice
  - Most expensive RAID implementation
  - requires twice as much storage space

(b) RAID 1: mirrored disks.

RAID Level-2

- **RAID level 2** is also known as memory-style error correcting code (ECC) organization.

(c) RAID 2: memory-style error-correcting codes.

  - Stripes data across disks similar to Level-0
    - ➢ difference is data is **bit** interleaved instead of **block** interleaved
    - ➢ For ex, the first bit of each byte can be stored in disk 1, the second bit in disk 2, and so on until the eighth bit is stored in disk 8; the error-correction bits are stored in further disks.
  - Uses ECC to monitor correctness of information on disk
    - ➢ Multiple disks record the ECC information to determine which disk is in fault
    - ➢ A parity disk is then used to reconstruct corrupted or lost data
  - RAID level 2 requires only 3 disks' overhead for 4 disks of data, unlike RAID level 1, which requires 4 disks' overhead.

RAID Level-3

- **RAID level 3**, or bit-interleaved parity organization;
  - One big problem with Level-2 is the number of extra disks needed to detect which disk had an error
  - Modern disks can already determine if there is an error
    - ➢ using ECC codes with each sector
  - So just need to include a parity disk
    - ➢ if a sector is bad, the disk itself tells us, and use the parity disk to correct it
  - Transfer rate for reading or writing a single block is faster than RAID level 1.
  - But supports fewer I/Os per second, since every disk has to participate in every I/O request.
  - Has performance problem [illegible] nputing and writing the parity.

(d) RAID 3: bit-interleaved parity.

RAID Level-4

(e) RAID 4: block-interleaved parity.

- **RAID level 4** interleaves file blocks
  - allows multiple small I/O's to be done at once
- Consists of block-level striping with dedicated parity.
- Still use a single disk for parity
- Now the parity is calculated over data from multiple blocks
  - Level-2,3 calculate it over a single block
- If an error detected, need to read other blocks on other disks to reconstruct data
- Doing multiple small reads is now faster than before
- However, writes are still very slow
  - this is because of calculating and writing the parity blocks
- Also, only one write is allowed at a time
  - all writes must access the check disk so other writes have to wait

RAID Level-5

- **RAID level 5** stripes file data and checks data over all the disks
  - ➢ no longer a single check disk
  - ➢ no more write bottleneck

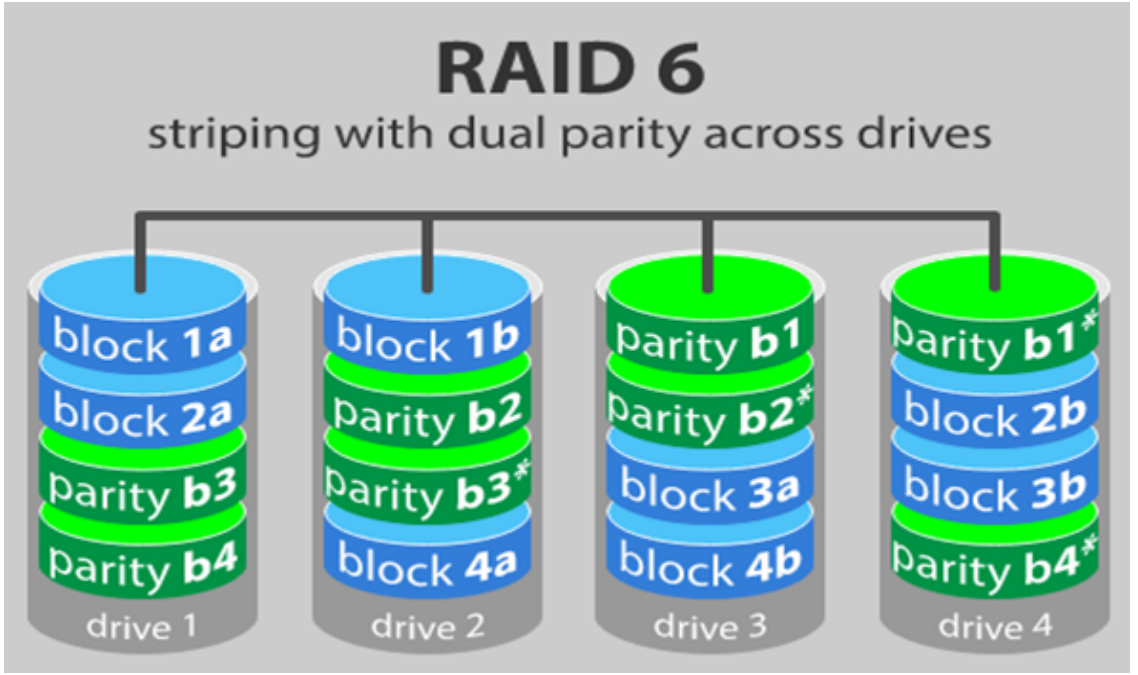
(f) RAID 5: block-interleaved distributed parity.

- Consists of block-level striping with distributed parity.
  - ➢ Unlike RAID 4, parity information is distributed among the drives
- Drastically improves the performance of multiple writes
  - ➢ they can now be done in parallel
- Slightly improves reads
  - ➢ one more disk to use for reading
- read and write performance close to that of RAID Level-1
- requires as much disk space as Levels-3,4

RAID Level-6

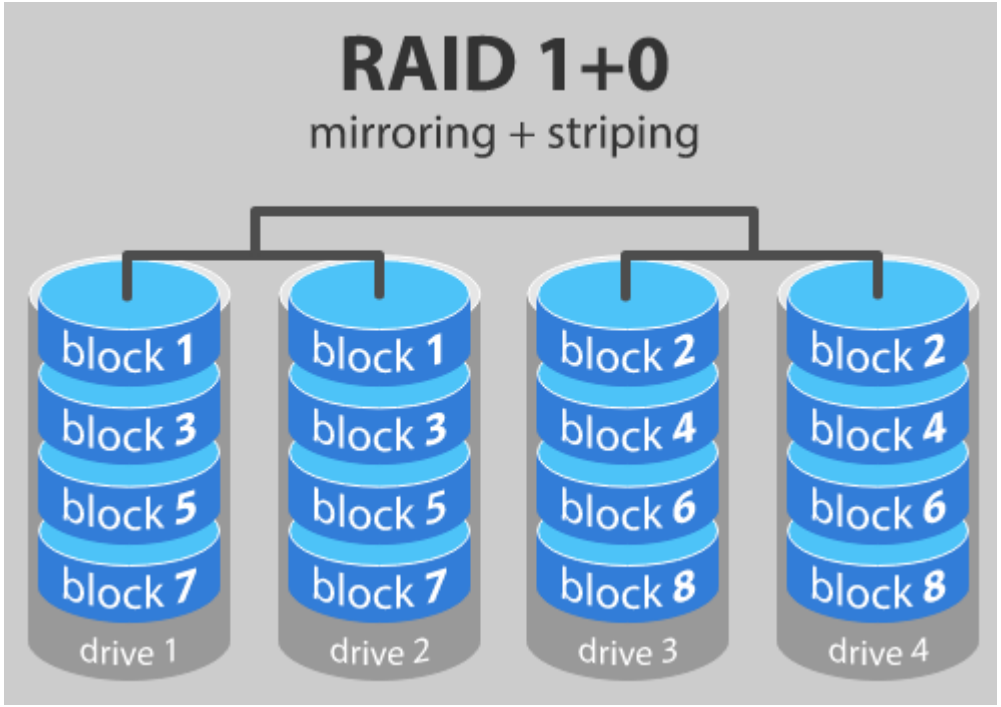
- RAID 6 is like RAID 5, but the parity data are written to two drives.
  - That means it requires at least 4 drives and can withstand 2 drives dying simultaneously.
- Write data transactions are slower than RAID 5 due to the additional parity data that have to be calculated.

(g) RAID 6: P + Q redundancy.



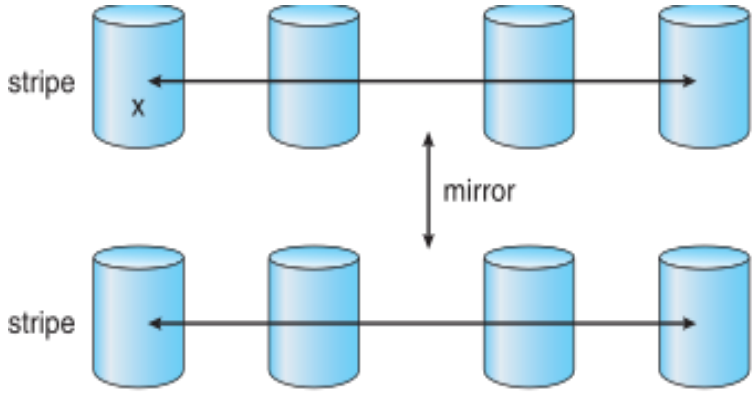
RAID level 10 - combining RAID 1 and RAID 0

- Provides security by mirroring all data on secondary drives while using striping across each set of drives to speed up data transfers.
- Rebuild time is very fast
- Half of the storage capacity goes to mirroring
  - so compared to large RAID 5 or RAID 6 arrays, this is an expensive way to have redundancy.



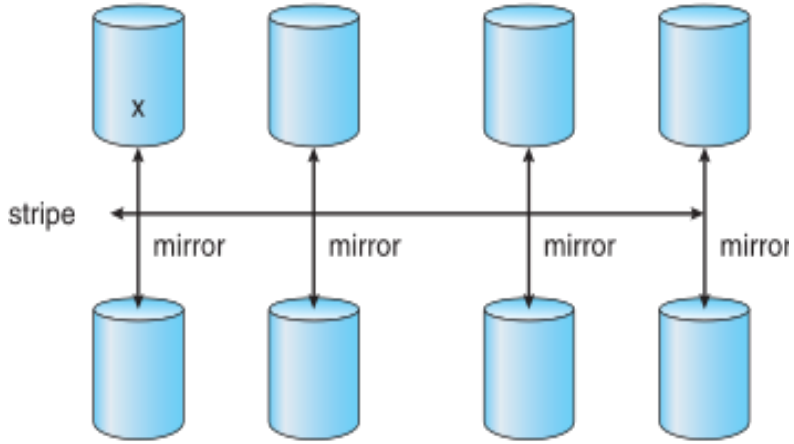
OPERATING SYSTEMS

RAID (0 + 1) and (1 + 0)



a) RAID 0 + 1 with a single disk failure.

Disks are striped and then the stripe is mirrored to another, equivalent stripe. If a single disk fails, an entire stripe is inaccessible.



b) RAID 1 + 0 with a single disk failure.

Disks are mirrored in pairs and then the resulting mirrored pairs are striped. This is better than 0+1 when a single disk fails

Selecting a RAID level

- One consideration is rebuild performance.
  - If a disk fails, the time needed to rebuild its data can be significant.
  - This may be an important factor if a continuous supply of data is required, as it is in high-performance or interactive database systems.
  - Furthermore, rebuild performance influences the mean time to failure.
  - Rebuild performance varies with the RAID level used.
    - Rebuilding is easiest for RAID level 1, since data can be copied from another disk.

Selecting a RAID level (Cont.)

- For the other levels, we need to access all the other disks in the array to rebuild data in a failed disk.
- Rebuild times can be hours for RAID 5 rebuilds of large disk sets.

- RAID level 0 is used in high-performance applications where data loss is not critical.
- RAID level 1 is popular for applications that require high reliability with fast recovery.
- RAID 0 + 1 and 1 + 0 are used where both performance and reliability are important—for example, for small databases.
- Due to RAID 1's high space overhead, RAID 5 is often preferred for storing large volumes of data.

Selecting a RAID level (Cont.)

- RAID system designers and administrators of storage have to make several other decisions as well.
  - How many disks should be in a given RAID set?
  - How many bits should be protected by each parity bit?
  - If more disks are in an array, data-transfer rates are higher, but the system is more expensive.
  - If more bits are protected by a parity bit, the space overhead due to parity bits is lower, but the chance that a second disk will fail before the first failed disk is repaired is greater, and that will result in data loss.

THANK YOU

Chandravva Hebbi
Department of Computer Science Engineering

chandravvahebbi@pes.edu

OPERATING SYSTEMS

Protection

Suresh Jamadagni
Department of Computer Science

OPERATING SYSTEMS

Slides Credits for all PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9th edition 2013 and some slides from 10th edition 2018
2. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9th edition 2018
3. Some presentation transcripts from A. Frank – P. Weisberg
4. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau

OPERATING SYSTEMS

Protection - Introduction

- Protection refers to a mechanism for controlling the access of programs, processes, or users to the resources defined by a computer system.
- This mechanism must provide a means for specifying the controls to be imposed, together with a means of enforcement
- Protection was originally conceived as an adjunct to multiprogramming operating systems, so that untrustworthy users might safely share a common logical name space, such as a directory of files, or share a common physical name space, such as memory.
- Modern protection concepts have evolved to increase the reliability of any complex system that makes use of shared resources.

OPERATING SYSTEMS

Goals of protection

- Need to prevent mischievous, intentional violation of an access restriction by a user
- Need to ensure that each program component active in a system uses system resources only in ways consistent with stated policies.
- A protection-oriented system should provides means to distinguish between authorized and unauthorized usage
- Improve reliability by detecting latent errors at the interfaces between component subsystems. Early detection of interface errors can often prevent contamination of a healthy subsystem by a malfunctioning subsystem.
- Provide a mechanism for the enforcement of the policies governing resource use. A protection system must have the flexibility to enforce a variety of policies.

OPERATING SYSTEMS

Principles of protection

- A key, time-tested guiding principle for protection is the **principle of least privilege**. It dictates that programs, users, and even systems be given just enough privileges to perform their tasks.
- An operating system following the principle of least privilege implements its features, programs, system calls, and data structures so that failure or compromise of a component does the minimum damage and allows the minimum damage to be done.
- An operating system should provide system calls and services that allow applications to be written with fine-grained access controls. It should provide mechanisms to enable privileges when they are needed and to disable them when they are not needed.

OPERATING SYSTEMS

Principles of protection

- Also beneficial is the creation of audit trails for all privileged function access.
- The audit trail allows the programmer, system administrator, or law-enforcement officer to trace all protection and security activities on the system
- Managing users with the principle of least privilege entails creating a separate account for each user, with just the privileges that the user needs.
- Some systems implement role-based access control (RBAC) to provide this functionality
- Computers implemented in a computing facility under the principle of least privilege can be limited to running specific services, accessing specific remote hosts via specific services, and doing so during specific times

OPERATING SYSTEMS

Domain of protection

- A computer system is a collection of processes and objects.
- ***Objects*** include both **hardware objects** (such as the CPU, memory segments, printers, disks, and tape drives) and **software objects** (such as files, programs, and semaphores).
- Each object has a unique name that differentiates it from all other objects in the system, and each can be accessed only through well-defined and meaningful operations.
- Objects are essentially abstract data types.
- The operations that are possible may depend on the object.
- Processes should be allowed to access only those resources for which it has authorization.
- At any time, a process should be able to access only those resources that it currently requires to complete its task. This requirement, commonly referred to as the **need-to-know principle**, is useful in limiting the amount of damage a faulty process can cause in the system

OPERATING SYSTEMS

Domain structure

- A process operates within a **protection domain**, which specifies the resources that the process may access.
- Each domain defines a set of objects and the types of operations that may be invoked on each object.
- The ability to execute an operation on an object is an **access right**.
- A domain is a collection of access rights, each of which is an ordered pair **<object-name, rights-set>**.
- For example, if domain *D* has the access right <file *F*, *{*read,write*}*>, then a process executing in domain *D* can both read and write file *F.* It cannot, however, perform any other operation on that object.
- Domains may share access rights

OPERATING SYSTEMS

Domain structure

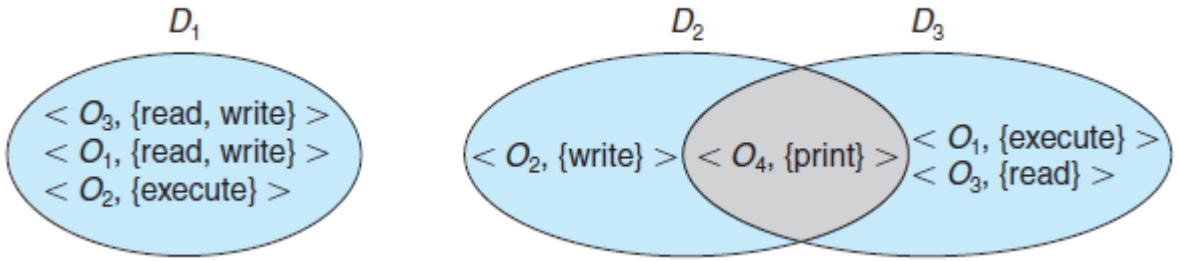
- The association between a process and a domain may be either **static**, if the set of resources available to the process is fixed throughout the process's lifetime, or **dynamic**.
- Establishing dynamic protection domains is more complicated than establishing static protection domains.
- If the association between processes and domains is fixed, and we want to adhere to the need-to-know principle, then a mechanism must be available to change the content of a domain.
- The reason stems from the fact that a process may execute in two different phases and may, for example, need read access in one phase and write access in another.
- If a domain is static, we must define the domain to include both read and write access. However, this arrangement provides more rights than are needed in each of the two phases, since we have read access in the phase where we need only write access, and vice versa

Domain structure

- Thus, the need-to-know principle is violated. We must allow the contents of a domain to be modified so that the domain always reflects the minimum necessary access rights.
- If the association is dynamic, a mechanism is available to allow **domain switching**, enabling the process to switch from one domain to another.
- We may also want to allow the content of a domain to be changed. If we cannot change the content of a domain, we can provide the same effect by creating a new domain with the changed content and switching to that new domain when we want to change the domain content

OPERATING SYSTEMS

Domain structure example



- We have three domains: *D*1, *D*2, and *D*3.
- The access right <*O*4, *{*print*}*> is shared by *D*2 and *D*3, implying that a process executing in either of these two domains can print object *O*4.
- A process must be executing in domain *D*1 to read and write object *O*1, while only processes in domain *D*3 may execute object *O*1

OPERATING SYSTEMS

Domain structure implementation

A domain can be realized in a variety of ways:

- Each ***user*** may be a domain. In this case, the set of objects that can be accessed depends on the identity of the user. Domain switching occurs when the user is changed—generally when one user logs out and another user logs in.
- Each ***process*** may be a domain. In this case, the set of objects that can be accessed depends on the identity of the process. Domain switching occurs when one process sends a message to another process and then waits for a response.
- Each ***procedure*** may be a domain. In this case, the set of objects that can be accessed corresponds to the local variables defined within the procedure. Domain switching occurs when a procedure call is made.

OPERATING SYSTEMS

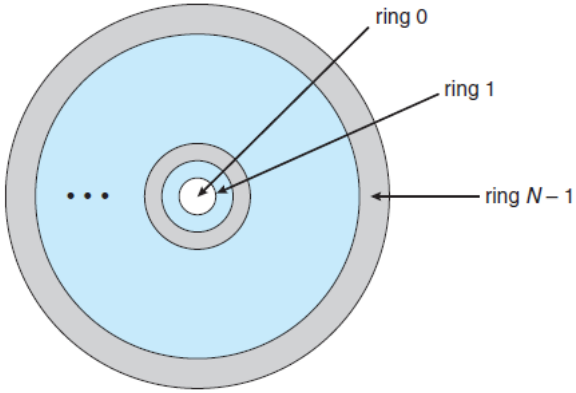
Domain structure implementation - UNIX

- In the UNIX operating system, a domain is associated with the user.
- Switching the domain corresponds to changing the user identification temporarily.
- This change is accomplished through the file system as follows.
  - An owner identification and a domain bit (known as the **setuid bit**) are associated with each file.
  - When the setuid bit is on, and a user executes that file, the userID is set to that of the owner of the file. When the bit is off, however, the userID does not change.
  - For example, when a user *A* (that is, a user with userID = *A*) starts executing a file owned by *B,* whose associated domain bit is off, the userID of the process is set to *A.* When the setuid bit is on, the userID is set to that of the owner of the file: *B.*
  - When the process exits, this temporary userID change ends.

OPERATING SYSTEMS

Domain structure implementation - MULTICS

- In the MULTICS system, the protection domains are organized hierarchically into a ring structure. The rings are numbered from 0 to 7.
- MULTICS has a segmented address space; each segment is a file, and each segment is associated with one of the rings. A segment description includes an entry that identifies the ring number.
- Each ring corresponds to a single domain.
- Let *Di* and *Dj* be any two domain rings. If $j < i$, then *Di* is a subset of *Dj* . That is, a process executing in domain *Dj* has more privileges than does a process executing in domain *Di*. A process executing in domain D_0 has the most privileges



OPERATING SYSTEMS

Domain structure implementation - MULTICS

- A current-ring-number counter is associated with each process, identifying the ring in which the process is executing currently.
- When a process is executing in ring i, it cannot access a segment associated with ring j ($j < i$). It can access a segment associated with ring k ($k \geq i$).
- Domain switching in MULTICS occurs when a process crosses from one ring to another by calling a procedure in a different ring
- The main disadvantage of the ring (or hierarchical) structure is that it does not allow us to enforce the need-to-know principle.
- In particular, if an object must be accessible in domain *Dj* but not accessible in domain *Di* , then we must have $j < i$. But this requirement means that every segment accessible in *Di* is also accessible in *Dj* .

THANK YOU

Suresh Jamadagni
Department of Computer Science Engineering

sureshjamadagni@pes.edu

OPERATING SYSTEMS

Access Matrix

Suresh Jamadagni
Department of Computer Science

OPERATING SYSTEMS

Slides Credits for all PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9th edition 2013 and some slides from 10th edition 2018
2. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9th edition 2018
3. Some presentation transcripts from A. Frank – P. Weisberg
4. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau

OPERATING SYSTEMS

Access Matrix

- A general model of protection can be viewed abstractly as a matrix, called an **access matrix**.
- The rows of the access matrix represent domains, and thecolumns represent objects. Each entry in the matrix consists of a set of access rights.
- Because the column defines objects explicitly, we can omit the object name from the access right.
- The entry access(*i,j*) defines the set of operations that a process executing in domain *Di* can invoke on object *Oj* .

OPERATING SYSTEMS

Access Matrix example

object / domain	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

- There are four domains and four objects - three files (*F*1, *F*2, *F*3) and one laser printer.
- A process executing in domain *D*1 can read files *F*1 and *F*3.
- A process executing in domain *D*4 has the same privileges as one executing in domain *D*1; but in addition, it can also write onto files *F*1 and *F*3.
- The laser printer can be accessed only by a process executing in domain *D*2.

OPERATING SYSTEMS

Access Matrix

- The access-matrix scheme provides us with the mechanism for specifying a variety of policies.
- The mechanism consists of implementing the access matrix and ensuring that the semantic properties we have outlined hold.
- More specifically, we must ensure that a process executing in domain *Di* can access only those objects specified in row *i*, and then only as allowed by the access-matrix entries.
- The access matrix can implement policy decisions concerning protection.
- The policy decisions involve which rights should be included in the (*i, j*)*th* entry.
- We must also decide the domain in which each process executes. This last policy is usually decided by the operating system.

OPERATING SYSTEMS

Access Matrix – Domain switching

- The access matrix provides an appropriate mechanism for defining and implementing strict control for both static and dynamic association between processes and domains.
- When we switch a process from one domain to another, we are executing an operation (switch) on an object (the domain).
- We can control domain switching by including domains among the objects of the access matrix.
- Similarly, when we change the content of the access matrix, we are performing an operation on an object: the access matrix.
- Again, we can control these changes by including the access matrix itself as an object
- Processes can switch from domain *Di* to domain *Dj* if and only if the access right switch $\in$ access(*i*, *j*)

OPERATING SYSTEMS

Access Matrix – Domain switching

object / domain	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

- A process executing in domain *D*3 or to domain *D*4.
- A process in domain *D*4 can switch to *D*1, and one in domain *D*1 can switch to *D*2.
- Allowing controlled change in the contents of the access-matrix entries requires three additional operations: copy, owner, and control
- The ability to copy an access right from one domain (or row) of the access matrix to another is denoted by an asterisk (*) appended to the access right.
- The copy right allows the access right to be copied only within the column (that is, for the object) for which the right is defined

Implementation of the Access Matrix – Global table

- The simplest implementation of the access matrix is a global table consisting of a set of ordered triples <domain, object, rights-set>.
- Whenever an operation *M* is executed on an object *Oj* within domain *Di* , the global table is searched for a triple <*Di* , *Oj* , *Rk*>, with $M \in Rk$.
- If this triple is found, the operation is allowed to continue; otherwise, an exception (or error) condition is raised.
- This implementation suffers from several drawbacks. The table is usually large and thus cannot be kept in main memory, so additional I/O is needed.
- Virtual memory techniques are often used for managing this table.
- In addition, it is difficult to take advantage of special groupings of objects or domains.
- For example, if everyone can read a particular object, this object must have a separate entry in every domain.

Implementation of the Access Matrix – Access list for objects

- Each column in the access matrix can be implemented as an access list for one object.
- Obviously, the empty entries can be discarded.
- The resulting list for each object consists of ordered pairs <domain, rights-set>, which define all domains with a nonempty set of access rights for that object.
- This approach can be extended easily to define a list plus a ***default*** set of access rights.
- When an operation *M* on an object *Oj* is attempted in domain *Di* , we search the access list for object *Oj* , looking for an entry *<Di , Rk>* with $M \in Rk$.
- If the entry is found, we allow the operation; if it is not, we check the default set.
- If *M* is in the default set, we allow the access. Otherwise, access is denied, and an exception condition occurs

Implementation of the Access Matrix – capability list for domains

- Rather than associating the columns of the access matrix with the objects as access lists, we can associate each row with its domain.
- A **capability list** for a domain is a list of objects together with the operations allowed on those objects.
- An object is often represented by its physical name or address, called a **capability**.
- To execute operation *M* on object *Oj* , the process executes the operation *M*, specifying the capability (or pointer) for object *Oj* as a parameter.
- Simple **possession** of the capability means that access is allowed.
- The capability list is associated with a domain, but it is never directly accessible to a process executing in that domain. Rather, the capability list is itself a protected object, maintained by the operating system and accessed by the user only indirectly.

OPERATING SYSTEMS

Implementation of the Access Matrix – capability list for domains

- Capability-based protection relies on the fact that the capabilities are never allowed to migrate into any address space directly accessible by a user process (where they could be modified).
- If all capabilities are secure, the object they protect is also secure against unauthorized access.

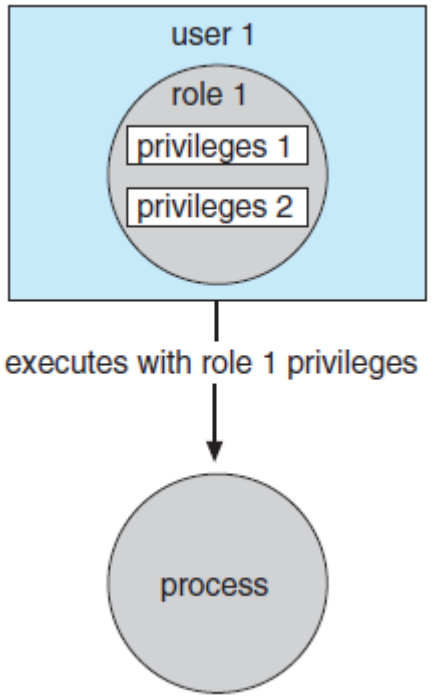
Implementation of the Access Matrix – lock-key mechanism

- The **lock–key scheme** is a compromise between access lists and capability lists.
- Each object has a list of unique bit patterns, called **locks**.
- Similarly, each domain has a list of unique bit patterns, called **keys**.
- A process executing in a domain can access an object only if that domain has a key that matches one of the locks of the object.
- As with capability lists, the list of keys for a domain must be managed by the operating system on behalf of the domain.
- Users are not allowed to examine or modify the list of keys (or locks) directly.

OPERATING SYSTEMS

Role based access control

- A privilege is the right to execute a system call or to use an option within that system call (such as opening a file with write access).
- Privileges can be assigned to processes, limiting them to exactly the access they need to perform their work.
- Privileges and programs can also be assigned to **roles**.
- Users are assigned roles or can take roles based on passwords to the roles.
- A user can take a role that enables a privilege, allowing the user to run a program to accomplish a specific task.
- This implementation of privileges decreases the security risk associated with superusers and setuid programs.



Revocation of access rights

In a dynamic protection system, we may need to revoke access rights to objects shared by different users.

- **Immediate versus delayed**. Does revocation occur immediately or later?
- **Selective versus general**. When an access right to an object is revoked, does it affect all the users who have an access right to that object, or can we specify a select group of users whose access rights should be revoked?
- **Partial versus total**. Can a subset of the rights associated with an object be revoked, or must we revoke all access rights for this object?
- **Temporary versus permanent**. Can access be revoked permanently (that is, the revoked access right will never again be available), or can access be revoked and later be obtained again?

Revocation of access rights

- With an access-list scheme, revocation is easy.
- The access list is searched for any access rights to be revoked, and they are deleted from the list. Revocation is immediate and can be general or selective, total or partial, and permanent or temporary.
- Capabilities, however, present a much more difficult revocation problem.
- Since the capabilities are distributed throughout the system, we must find them before we can revoke them.
- Schemes that implement revocation for capabilities include the following:
  - **Reacquisition**. Periodically, capabilities are deleted from each domain. If a process wants to use a capability, it may find that that capability has been deleted. The process may then try to reacquire the capability. If access has been revoked, the process will not be able to reacquire the capability

Revocation of access rights

- Schemes that implement revocation for capabilities include the following:
  - **Reacquisition**. Periodically, capabilities are deleted from each domain. If a process wants to use a capability, it may find that that capability has been deleted. The process may then try to reacquire the capability. If access has been revoked, the process will not be able to reacquire the capability
  - **Back-pointers**. A list of pointers is maintained with each object, pointing to all capabilities associated with that object. When revocation is required, we can follow these pointers, changing the capabilities as necessary. Implementation of this scheme is costly.
  - **Indirection**. The capabilities point indirectly, not directly, to the objects. Each capability points to a unique entry in a global table, which in turn points to the object. We implement revocation by searching the global table for the desired entry and deleting it

Revocation of access rights

Schemes that implement revocation for capabilities include the following:

- **Keys**. A key is a unique bit pattern that can be associated with a capability. This key is defined when the capability is created, and it can be neither modified nor inspected by the process that owns the capability. A **master key** is associated with each object. Revocation replaces the master key with a new value via the set-key operation, invalidating all previous capabilities for this object

THANK YOU

Suresh Jamadagni
Department of Computer Science Engineering

sureshjamadagni@pes.edu

OPERATING SYSTEMS

I/O Management, System Protection and Security

Arya S S
Department of Computer Science

OPERATING SYSTEMS

System Protection – Implementation of Access Matrix, Access control, Access rights

Arya S S
Department of Computer Science

OPERATING SYSTEMS

Slides Credits for all PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9th edition 2013 and some slides from 10th edition 2018
2. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9th edition 2018
3. Some presentation transcripts from A. Frank – P. Weisberg
4. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau

Implementation of Access Matrix

- Generally, a sparse matrix (i.e. most of the entries will be empty)
- **Option 1 – Global table**
  - Store ordered triples **<domain, object, rights-set>** in table
  - A requested operation M on object O_j within domain D_i -> search table for a triple $< D_i, O_j, R_k >$
    - with $M \in R_k$
    - If triple found, operation is allowed to continue; otherwise an exception or condition is raised
  - But table could be large -> won't fit in main memory
    - Virtual memory techniques can be used for managing this table
  - Difficult to group objects
    - consider an object that all domains can read, this object must have a separate entry in every domain)

Implementation of Access Matrix (Cont.)

- **Option 2 – Access lists for objects**
  - Each **column** implemented as an access list for one object i.e. specifying user names and the types of access allowed for each user (empty entries can be discarded)
  - Resulting per-object list consists of ordered pairs **<domain, rights-set>** defining all domains with non-empty set of access rights for the object
  - Easily extended to contain default set -> If M ∈ default set, also allow access
    - For efficiency, check the default set first and then search the access list

Implementation of Access Matrix (Cont.)

- Each column = Access-control list for one object
  Defines who can perform what operation

  Domain 1 = Read, Write
  Domain 2 = Read
  Domain 3 = Read

- Each Row = Capability List (like a key)
  For each domain, what operations allowed on what objects

  Object F1 – Read

  Object F4 – Read, Write, Execute

  Object F5 – Read, Write, Delete, Copy

Implementation of Access Matrix (Cont.)

- **Option 3 – Capability list for domains**
  - Instead of object-based (i.e column wise), list is domain based (i.e row wise)
  - **Capability list** for domain is list of objects together with operations allowed on them
  - Object represented by its name or address, called a **capability**
  - Execute operation M on object O_j, process requests operation and specifies capability as parameter
    - **Possession of capability** means access is allowed
  - Capability list associated with domain but never directly accessible to a process executing in that domain
    - Rather, protected object, maintained by OS and accessed by the user indirectly
    - Like a “secure pointer”
    - Idea can be extended up to the application level

Implementation of Access Matrix (Cont.)

- **Option 4 – Lock-key**
  - Compromise between access lists and capability lists
  - Each object has a list of unique bit patterns, called **locks**
  - Each domain has a list of unique bit patterns called **keys** (managed by the OS)
  - Process in a domain can only access object if domain has a key that matches one of the locks

Comparison of Implementations

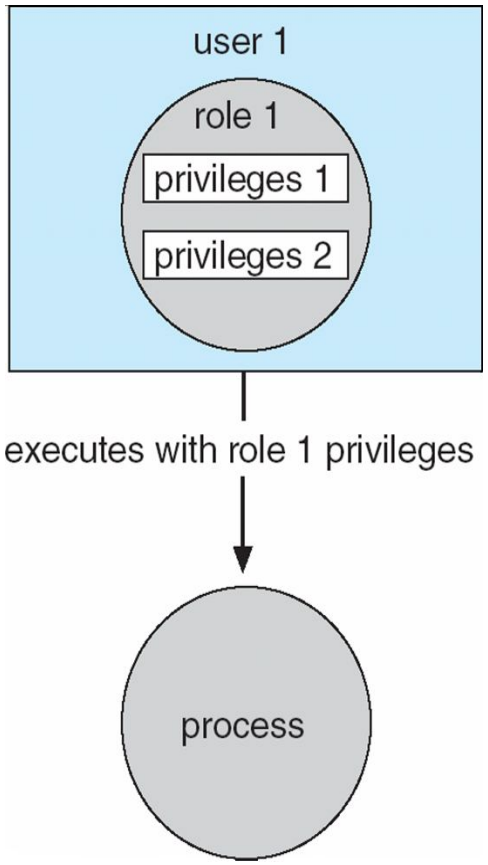
- Many trade-offs to consider
  - Global table is simple, but can be large
  - Access lists correspond to needs of users
    - Access rights for a particular domain is non-localized, so difficult to determine the set of access rights for each domain
    - Every access to an object must be checked
      - Many objects and access rights -> slow (i.e not suitable for large system with long access lists)
  - Capability lists useful for localizing information for a given process
    - But revocation capabilities can be inefficient
  - Lock-key effective and flexible, keys can be passed freely from domain to domain, easy revocation

Comparison of Implementations (Cont.)

- Most systems use combination of access lists and capabilities
  - First access to an object -> access list searched
    - If allowed, capability created and attached to process
      - Additional accesses need not be checked
    - After last access, capability destroyed
    - Consider file system with ACLs per file recorded in a new entry in a file table (file table maintained by the OS such as UNIX and protection is ensured)

Access Control

- Protection can be applied to non-file resources
- Oracle Solaris 10 provides **role-based access control** (**RBAC**) to implement least privilege
  - ***Privilege*** is right to execute system call or use an option (ex: write access for a file) within a system call
  - Can be assigned to processes
  - Users assigned ***roles*** granting access to privileges and programs
    - Enable role via password to gain its privileges
  - Similar to access matrix



Revocation of Access Rights

- Various options to remove the access right of a domain to an object
  - **Immediate vs. delayed** (i.e. when revocation will occur)
  - **Selective vs. general** (i.e. select group of users or all the users)
  - **Partial vs. total** (i.e. subset of the rights or all the rights)
  - **Temporary vs. permanent** (can access right be revoked and obtained later?)
- Access List – Delete access rights from access list
  - **Simple** – search access list and remove entry, revocation is easy
  - **Immediate**, **general** or **selective**, **total** or **partial**, **permanent** or **temporary**

Revocation of Access Rights (Cont.)

- **Capability List** – Scheme required to locate capability in the system before capability can be revoked
  - **Reacquisition** – periodic delete from each domain, with reacquire and denial if revoked by a process
  - **Back-pointers** – set of pointers from each object to all capabilities of that object , follow these pointers for revocation (adopted in Multics)
  - **Indirection** – capability points to global table entry which in turn points to object – delete entry from global table, selective revocation not allowed
  - **Keys** – unique bit pattern associated with a capability, generated when capability is created
    - Master key associated with object, key matches master key for access
    - Revocation – create new master key (with a new value)
    - Policy decision of who can create and modify keys – object owner or others?

THANK YOU

Arya S S
Department of Computer Science Engineering

aryadeep@pes.edu

OPERATING SYSTEMS

Storage Management

Suresh Jamadagni
Department of Computer Science

OPERATING SYSTEMS

Storage Management – System calls

Suresh Jamadagni
Department of Computer Science

OPERATING SYSTEMS

Slides Credits for all PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9th edition 2013 and some slides from 10th edition 2018
2. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9th edition 2018
3. Some presentation transcripts from A. Frank – P. Weisberg
4. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau

OPERATING SYSTEMS

File access permission

- There are nine permission bits for each file, divided into three categories
- The term user in the first three rows refers to the owner of the file
- The first rule is that to open any type of file by name, user must have execute permission in each directory mentioned in the name, including the current directory.
- The execute permission bit for a directory is often called the search bit
- For example, to open the file /usr/include/stdio.h, user would need execute permission in the directory /, execute permission in the directory /usr, and execute permission in the directory /usr/include and appropriate permission for the file stdio.h

st_mode mask	Meaning
S_IRUSR	user-read
S_IWUSR	user-write
S_IXUSR	user-execute
S_IRGRP	group-read
S_IWGRP	group-write
S_IXGRP	group-execute
S_IROTH	other-read
S_IWOTH	other-write
S_IXOTH	other-execute

OPERATING SYSTEMS

File access permission

- The read permission for a file determines whether user can open an existing file for reading: the O_RDONLY and O_RDWR flags for the open function.
- The write permission for a file determines whether user can open an existing file for writing: the O_WRONLY and O_RDWR flags for the open function.
- User must have write permission for a file to specify the O_TRUNC flag in the open function.
- User cannot create a new file in a directory unless they have write permission and execute permission in the directory.
- To delete an existing file, user needs write permission and execute permission in the directory containing the file.
- Users do not need read permission or write permission for the file itself

Set-User-ID and Set-Group-ID

real user ID real group ID	who we really are
effective user ID effective group ID supplementary group IDs	used for file access permission checks
saved set-user-ID saved set-group-ID	saved by `exec` functions

- Every process has six or more IDs associated with it
- The real user ID and real group ID identify who the user really is. These two fields are taken from an entry in the password file when the user logs in
- The effective user ID, effective group ID, and supplementary group IDs determine file access permissions for the user
- The saved set-user-ID and saved set-group-ID contain copies of the effective user ID and the effective group ID, respectively, when a program is executed.

OPERATING SYSTEMS

Set-User-ID and Set-Group-ID

```
#include <unistd.h>
int setuid(uid_t uid);
int setgid(gid_t gid);
```

Both return: 0 if OK, −1 on error

- We can set the real user ID and effective user ID with the setuid function.
- We can set the real group ID and the effective group ID with the setgid function.
- If the process has superuser privileges, the setuid function sets the real user ID, effective user ID, and saved set-user-ID to uid.
- If the process does not have superuser privileges, but uid equals either the real user ID or the saved set-user-ID, setuid sets only the effective user ID to uid. The real user ID and the saved set-user-ID are not changed.
- If neither of these two conditions is true, errno is set to EPERM and −1 is returned

OPERATING SYSTEMS

access and faccessat

```
#include <unistd.h>
int access(const char *pathname, int mode);
int faccessat(int fd, const char *pathname, int mode, int flag);
                                        Both return: 0 if OK, –1 on error
```

mode	Description
R_OK	test for read permission
W_OK	test for write permission
X_OK	test for execute permission

- When a user tries to open a file, the kernel performs access tests based on the effective user and group IDs
- Sometimes, however, a process wants to test accessibility based on the real user and group IDs. This is useful when a process is running as someone else, using either the set-user-ID or the set-group-ID feature.
- The **access** and **faccessat** functions base their tests on the real user and group IDs.

umask

```
#include <sys/stat.h>
mode_t umask(mode_t cmask);
                                    Returns: previous file mode creation mask
```

- The umask function sets the file mode creation mask for the process and returns the previous value
- The cmask argument is formed as the bitwise OR of any of the nine constants
- The file mode creation mask is used whenever the process creates a new file or a new directory.

st_mode mask	Meaning
S_IRUSR	user-read
S_IWUSR	user-write
S_IXUSR	user-execute
S_IRGRP	group-read
S_IWGRP	group-write
S_IXGRP	group-execute
S_IROTH	other-read
S_IWOTH	other-write
S_IXOTH	other-execute

OPERATING SYSTEMS

chmod, fchmod, and fchmodat

```
#include <sys/stat.h>
int chmod(const char *pathname, mode_t mode);
int fchmod(int fd, mode_t mode);
int fchmodat(int fd, const char *pathname, mode_t mode, int flag);
                                    All three return: 0 if OK, –1 on error
```

- The chmod, fchmod, and fchmodat functions allow us to change the file access permissions for an existing file
- The chmod function operates on the specified file, whereas the fchmod function operates on a file that has already been opened.
- The fchmodat function behaves like chmod when the pathname argument is absolute or when the fd argument has the value AT_FDCWD and the pathname argument is relative
- To change the permission bits of a file, the effective user ID of the process must be equal to the owner ID of the file, or the process must have superuser permissions

OPERATING SYSTEMS

chown, fchown, fchownat, and lchown

```
#include <unistd.h>

int chown(const char *pathname, uid_t owner, gid_t group);

int fchown(int fd, uid_t owner, gid_t group);

int fchownat(int fd, const char *pathname, uid_t owner, gid_t group,
             int flag);

int lchown(const char *pathname, uid_t owner, gid_t group);
```

All four return: 0 if OK, –1 on error

- The chown functions allow users to change a file's user ID and group ID, but if either of the arguments owner or group is −1, the corresponding ID is left unchanged
- The fchown function changes the ownership of the open file referenced by the fd argument.
- lchown and fchownat (with the AT_SYMLINK_NOFOLLOW flag set) change the owners of the symbolic link itself, not the file pointed to by the symbolic link

OPERATING SYSTEMS

File truncation

```
#include <unistd.h>

int truncate(const char *pathname, off_t length);

int ftruncate(int fd, off_t length);
```

Both return: 0 if OK, –1 on error

- **truncate** and **ftruncate** functions truncate an existing file to *length* bytes.
- If the previous size of the file was greater than *length*, the data beyond *length* is no longer accessible.
- If the previous size was less than *length*, the file size will increase and the data between the old end of file and the new end of file will read as 0 (i.e., a hole is probably created in the file)

THANK YOU

Suresh Jamadagni
Department of Computer Science Engineering

sureshjamadagni@pes.edu

OPERATING SYSTEMS

I/O Management, System Protection and Security and Case Study

Kakoli Bora
Department of Computer Science

OPERATING SYSTEMS

Case Study – Windows File System

Kakoli Bora
Department of Computer Science

OPERATING SYSTEMS

Slides Credits for all PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9th edition 2013 and some slides from 10th edition 2018
2. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9th edition 2018
3. Some presentation transcripts from A. Frank – P. Weisberg
4. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau
5. Internet source

File System - Basics

- FAT file system is still used for portability on other systems such as cameras, flash memory and external disks
- FAT file system does not restrict file access to authorized users.
- NTFS uses ACLs to control access to individual files and supports encryption.
- NTFS supports data recovery, fault tolerance, very large files and file systems, journaling, file compression, etc
- A file in NTFS is not a simple byte stream, as in MS-DOS or UNIX, rather, it is a structured object consisting of attributes
- Attributes like file name, creation time, descriptor, ACLs, etc

File System – Basics (Cont.)

- The fundamental structure of Windows file system (NTFS) is a *volume*
  - Created by the disk administrator utility
  - Based on a logical disk partition
  - May occupy a portions of a disk, an entire disk, or span across several disks
- All *metadata*, such as information about the volume, is stored in a regular file
- NTFS uses ***clusters*** as the underlying unit of disk allocation
  - A cluster is a number of disk sectors that is a power of two
  - The default cluster size is based on the volume size - 4 KB for volumes > 2 GB
  - Because the cluster size is smaller than for the 16-bit FAT file system, the amount of internal fragmentation is reduced

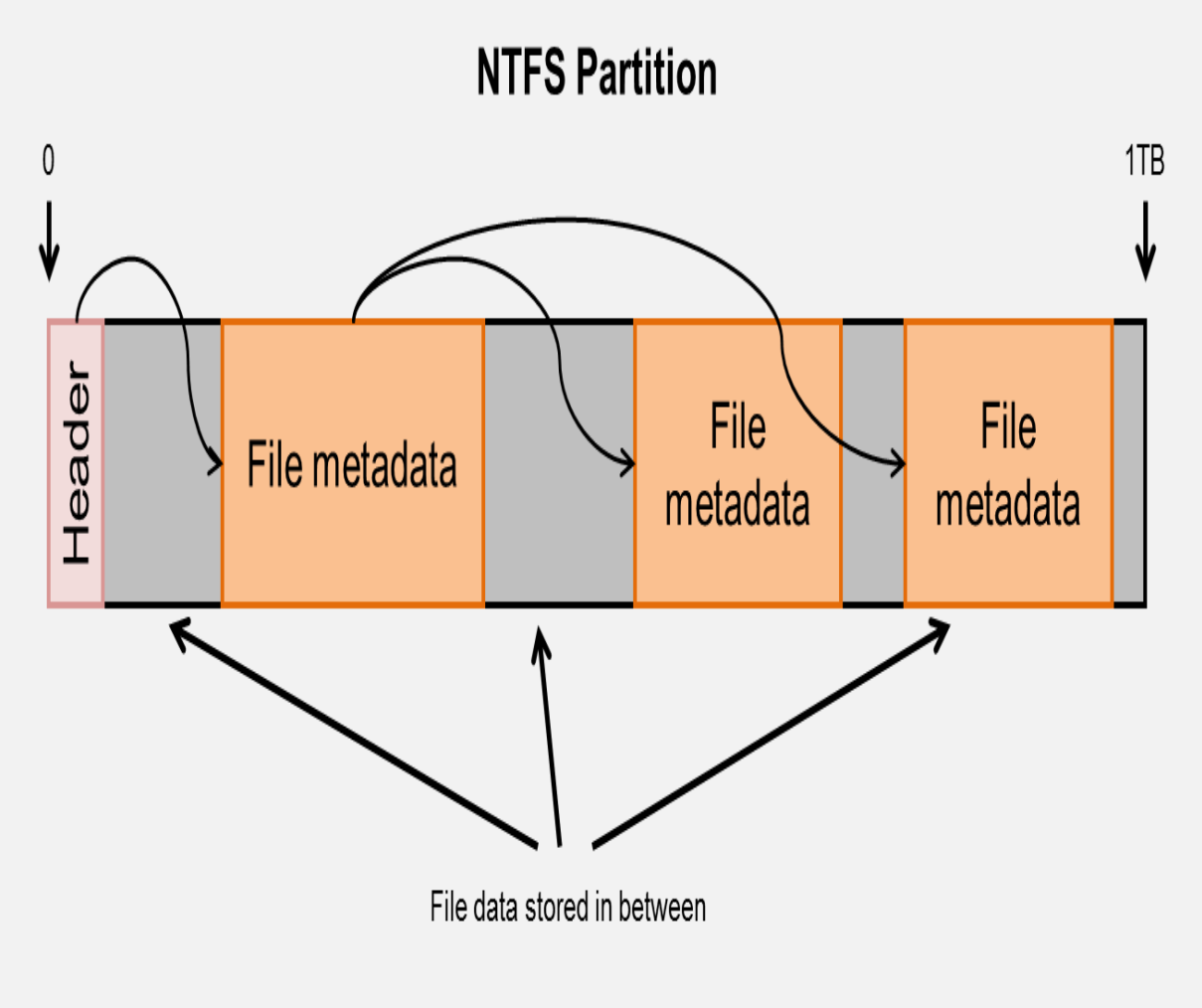
OPERATING SYSTEMS

File System — NTFS vs FAT

Features	NTFS	FAT32	FAT16	FAT12
Max Partition Size	2TB	32GB	4GB	16MB
Max File Size	16TB	4GB	2GB	Less than 16MB
Cluster Size	4KB	4KB to 32KB	2KB to 64KB	0.5KB to 4KB
Fault Tolerance	Auto Repair	No	No	No
Compression	Yes	No	No	No
Security	Local and Network	Only Network	Only Network	Only Network
Compatibility	Windows 10/8/7/XP/Vista/2000	Windows ME/2000/XP/7/8.1	Windows ME/2000/XP/7/8.1	Windows ME/2000/XP/7/8.1

OPERATING SYSTEMS

File System — NTFS (Cont.)



❖NTFS, the filesystem used by all modern versions of Windows

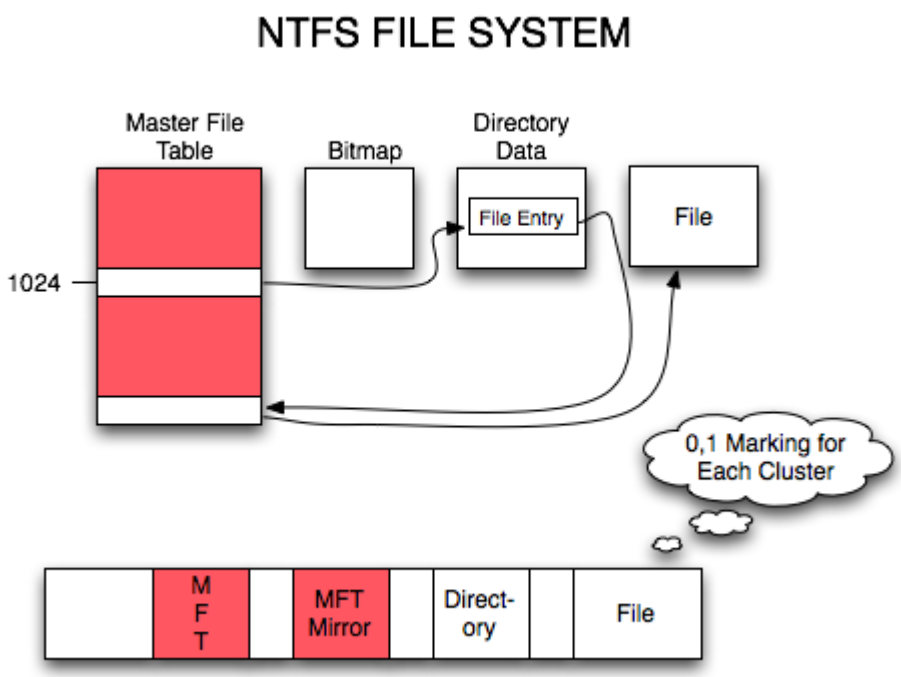
❖Instead of storing file metadata in tables scattered across the partition (usually to optimise hard disk seek times going from file metadata to actual file data), NTFS stores all the file metadata in a few large contiguous blocks.

❖These blocks are collectively known as the MFT (master file table).

❖Helps to quickly scan all the files in the partition to get each file's associated metadata. Instead of recursing through each directory manually and enumerating contents (like when computing summary statistics for directories)

OPERATING SYSTEMS

File System — NTFS (Cont.)



❖Every file in NTFS is described by one or more records in an array stored in a special file called the Master File Table (MFT)

❖Windows puts the Master File Table at the front of the drive, with a mirror file in the middle of the drive with only the first directory containing critical information.

❖The Bitmap is used to determine allocation/non-allocation of clusters.

File System — Internal Layout

- NTFS uses logical cluster numbers (LCNs) as disk addresses
- Physical disk offset in bytes = LCN x cluster size
- A file in NTFS is not a simple byte stream, as in MS-DOS or UNIX, rather, it is a structured object consisting of attributes
- Each file on an NTFS volume has a unique ID called a file reference.
  - 64-bit quantity that consists of a 48-bit file number and a 16-bit sequence number
  - Sequence number is incremented every time an MFT entry is reused, can be used to perform internal consistency checks (to catch a stale reference to a deleted file)
- The NTFS name space is organized by a hierarchy of directories; the index root contains the top level of the B+ tree
  - B+ tree is an extension of the B tree

File System—B+ Tree vs. B Tree

B+ Tree	B Tree
Data is only saved on the leaf nodes.	Both leaf nodes and internal nodes can store data
Data stored on the leaf node makes the search more accurate and faster.	Searching is slow due to data stored on Leaf and internal nodes.
Deletion is not difficult as an element is only removed from a leaf node.	Deletion of elements is a complicated and time-consuming process.
Linked leaf nodes make the search efficient and quick.	You cannot link leaf nodes.

The drawback of B-tree used for indexing, however is that it stores the data pointer (a pointer to the disk file block containing the key value), corresponding to a particular key value, along with that key value in the node of a B-tree. This technique, greatly reduces the number of entries that can be packed into a node of a B-tree, thereby contributing to the increase in the number of levels in the B-tree, hence increasing the search time of a record.

B+ tree eliminates the above drawback by storing data pointers only at the leaf nodes of the tree.

File System - Recovery

- All file system data structure updates are performed inside transactions that are logged.
  - Before a data structure is altered, the transaction writes a log record that contains redo and undo information.
  - After the data structure has been changed, a commit record is written to the log to signify that the transaction succeeded.
  - After a crash, the file system data structures can be restored to a consistent state by processing the log records.

File System – Recovery (Cont.)

- This scheme does not guarantee that all the user file data can be recovered after a crash, just that the file system data structures (the metadata files) are undamaged and reflect some consistent state prior to the crash.
- The log is stored in the third metadata file at the beginning of the volume.
- The logging functionality is provided by the *log-file service which keeps track of free space in the log file*
  - *halts new I/O operations in case free space gets too low*

File System - Security

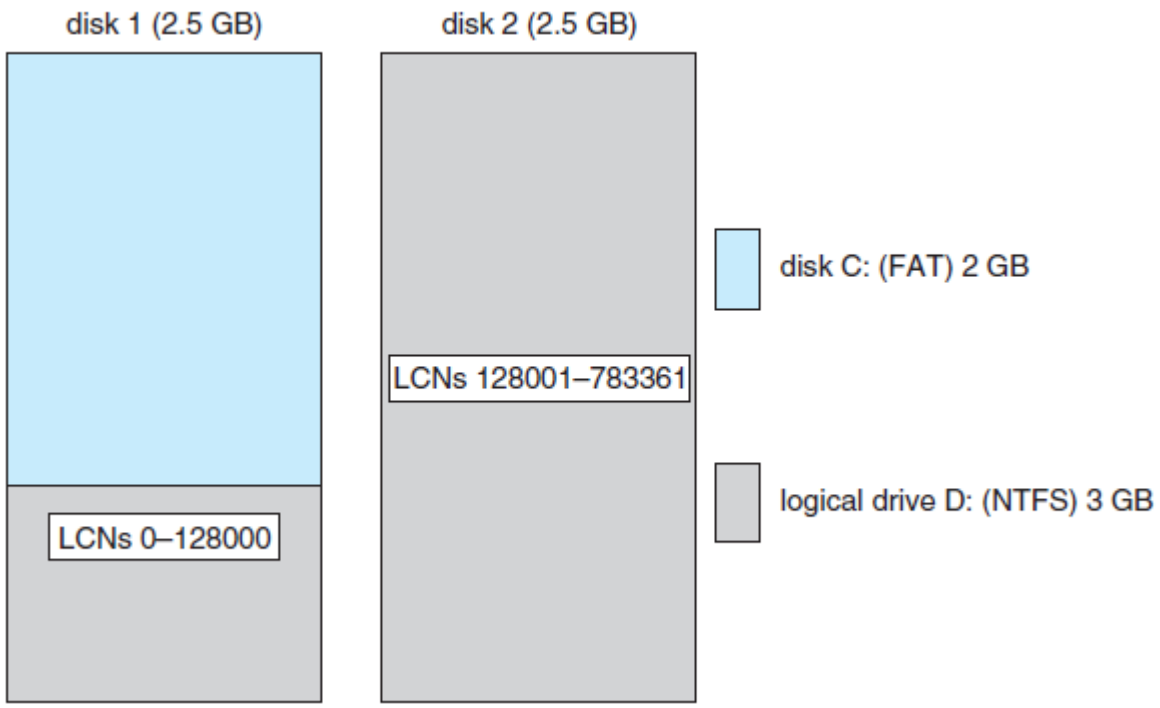
- Security of an NTFS volume is derived from the Windows object model.
- Each file object has a security descriptor attribute stored in this MFT record.
- This attribute contains the access token of the owner of the file, and an access control list that states the access privileges that are granted to each user that has access to the file.

OPERATING SYSTEMS

Volume Management and Fault Tolerance

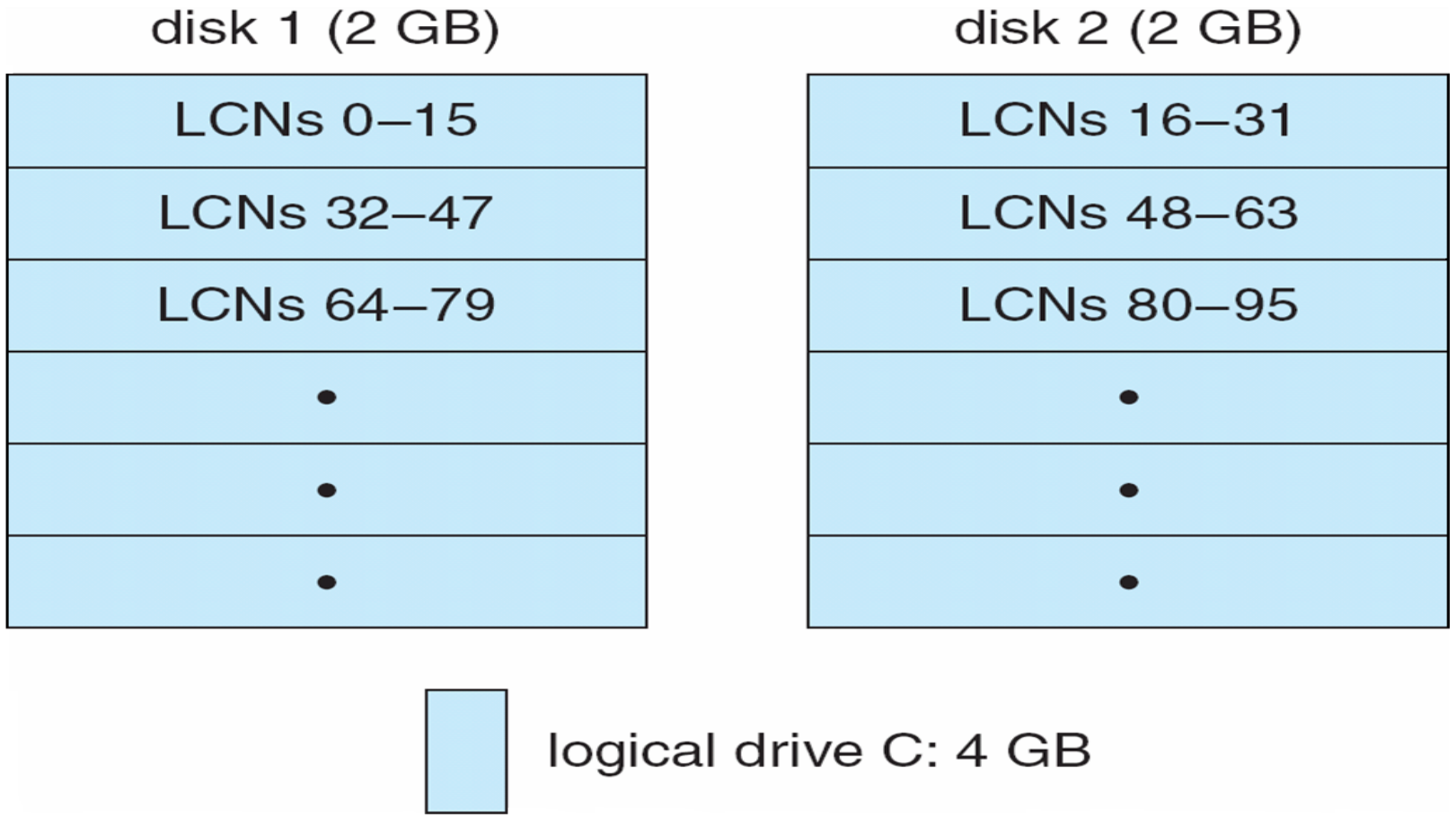
- **FtDisk**, the fault tolerant disk driver provides several ways to combine multiple SCSI disk drives into one logical volume
- Logically concatenate multiple disks to form a large logical volume, a *volume set*
- Interleave multiple physical partitions in round-robin fashion to form a ***stripe set*** (also called RAID level 0, or "disk striping")
  - FtDisk uses a stripe size of 64 KB – 1st 64 KB of the logical volume are stored in the 1st physical partition, 2nd 64 KB in the 2nd physical partition, and so on
  - Windows also supports RAID level 1 (mirroring) and RAID level 5 (*stripe set with parity)*
- Disk mirroring, or RAID level 1, is a robust scheme that uses a ***mirror set*** — two equally sized partitions on tow disks with identical data contents
- To deal with disk sectors that go bad, FtDisk, uses a hardware technique called ***sector sparing*** *(i.e. formatting a disk drive leaves extra sectors unmapped as spares)* and NTFS uses a software technique called ***cluster remapping*** *(i.e changes the pointers in the MFT from bad block to unallocated block)*

Volume Set On Two Drives

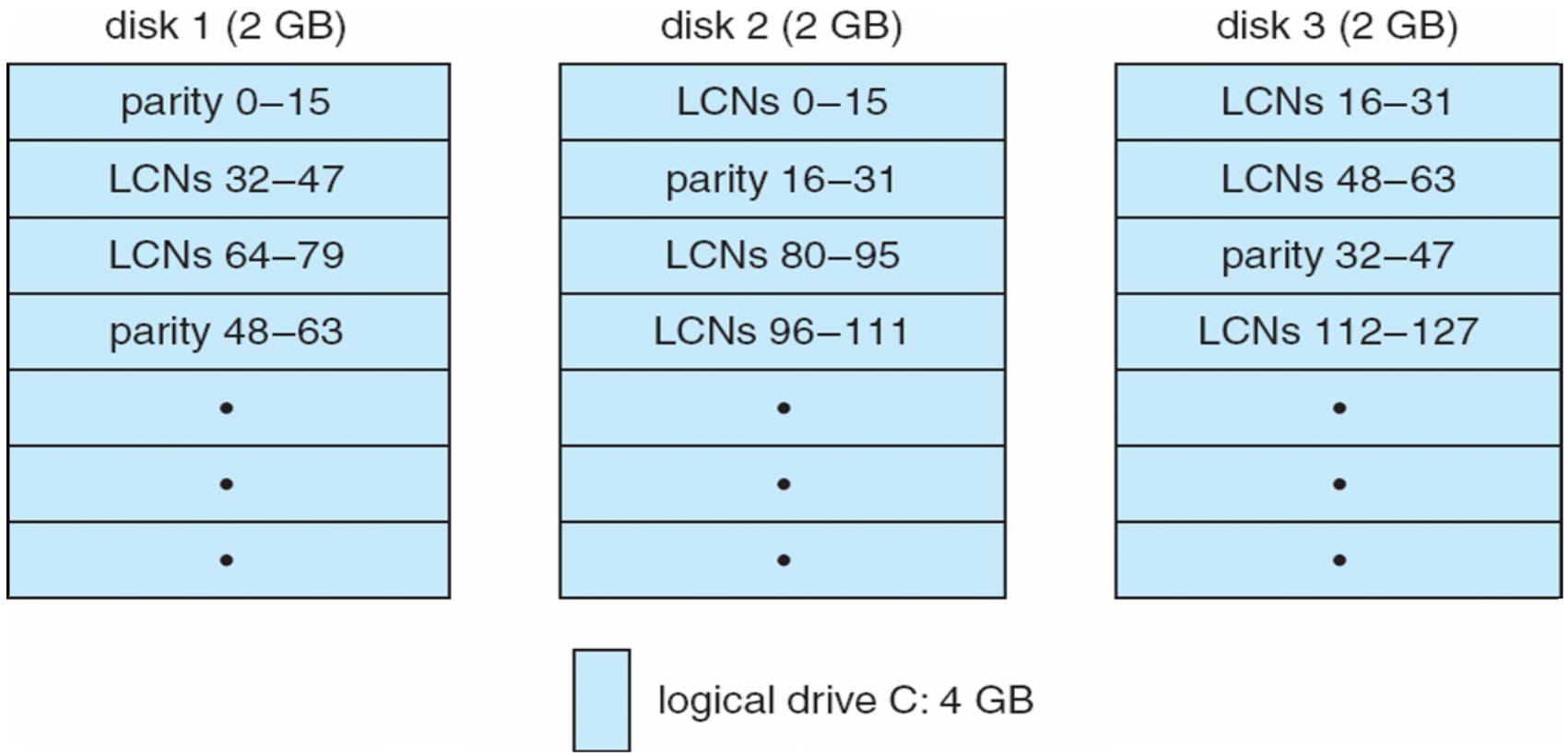


Logical volume or a *volume set can consist of up to 32 physical partitions*
A volume set can be extended by extending the bitmap metadata on the NTFS volume to cover the newly added space (bitmap contains information about free or used data blocks/clusters on a volume)
Fig 1==> 128000 x 4 KB = 0.5 GB
Fig 2 => 655360 x 4 KB = 2.5 GB

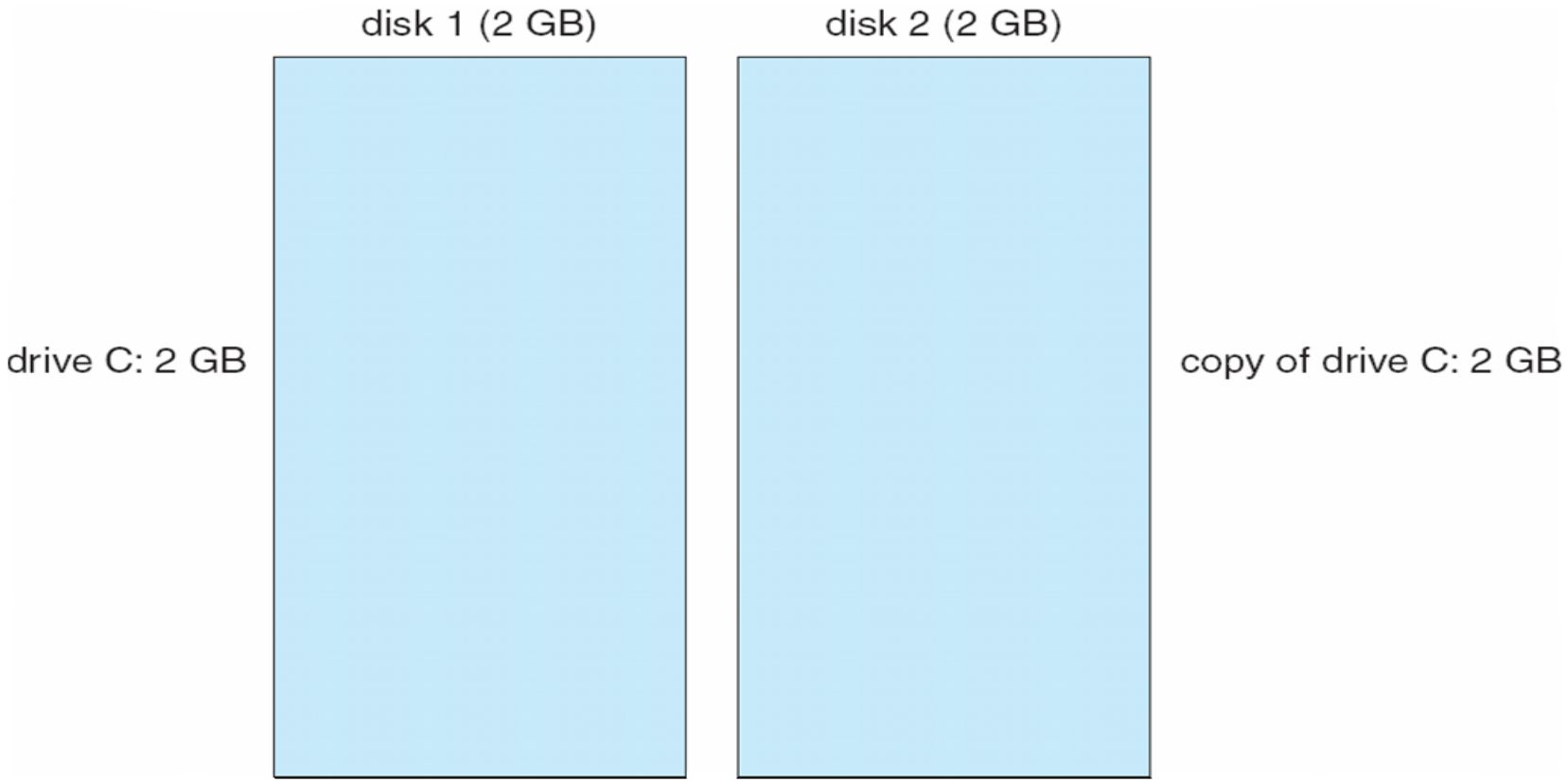
Stripe Set on Two Drives



Stripe Set With Parity on Three Drives



Mirror Set on Two Drives



File System — Compression

- To compress a file, NTFS divides the file's data into *compression units*, which are blocks of 16 contiguous clusters.
  - To improve performance when reading contiguous compression units, NTFS pre-fetches and decompresses ahead of the application requests.
- For sparse files, NTFS uses another technique to save space.
  - Clusters that contain all zeros (ie. never been written) are not actually allocated or stored on disk.
  - Instead, gaps are left in the sequence of virtual cluster numbers stored in the MFT entry for the file.
  - When reading a file, if a gap in the virtual cluster numbers is found, NTFS just zero-fills that portion of the caller's buffer.

File System (Contd.)

- Mount Points, Symbolic links, Hard Links
  - Hard links: A single file has an entry in more than one directory of the same volume
- NTFS Journal – describes all changes that have been made to the file system.
  - Used by services such as User-mode (to identify what files have changed), search indexer (to re-index files) and file-replication (to replicate files across the network)
- Snapshots to create a shadow copy of volume for backup (and recovery of files if accidentally deleted)

THANK YOU

Kakoli Bora
Department of Computer Science Engineering

k_bora@pes.edu

OPERATING SYSTEMS

Security

Suresh Jamadagni
Department of Computer Science

OPERATING SYSTEMS

Slides Credits for all PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9th edition 2013 and some slides from 10th edition 2018
2. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9th edition 2018
3. Some presentation transcripts from A. Frank – P. Weisberg
4. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau

OPERATING SYSTEMS

Security

- **Security**, requires not only an adequate protection system but also consideration of the ***external*** environment within which the system operates.
- Computer resources must be guarded against unauthorized access, malicious destruction or alteration, and accidental introduction of inconsistency.
- These resources include information stored in the system (both data and code), as well as the CPU, memory, disks, tapes, and networking that are the computer.
- In many applications, ensuring the security of the computer system is worth considerable effort
- We say that a system is **secure** if its resources are used and accessed as intended under all circumstances

Security

- Security violations (or misuse) of the system can be categorized as intentional (malicious) or accidental.
- It is easier to protect against accidental misuse than against malicious misuse
- A **threat** is the potential for a security violation, such as the discovery of a vulnerability, whereas an **attack** is the attempt to break security.

Security Threats

- **Breach of confidentiality**. This type of violation involves unauthorized reading of data (or theft of information). Typically, a breach of confidentiality is the goal of an intruder
- **Breach of integrity**. This violation involves unauthorized modification of data.
- **Breach of availability**. This violation involves unauthorized destruction of data.
- **Theft of service**. This violation involves unauthorized use of resources.
- **Denial of service**. This violation involves preventing legitimate use of the system.

OPERATING SYSTEMS

Security Threats

- Attackers use several standard methods in their attempts to breach security. The most common is **masquerading**, in which one participant in a communication pretends to be someone else (another host or another person).
- By masquerading, attackers breach **authentication**, the correctness of identification; they can then gain access that they would not normally be allowed or escalate their privileges—obtain privileges to which they would not normally be entitled.
- A **replay attack** consists of the malicious or fraudulent repeat of a valid data transmission. Sometimes the replay comprises the entire attack. But frequently it is done along with **message modification**, again to escalate privileges

Security Threats

- **man-in-the-middle attack**, in which an attacker sits in the data flow of a communication, masquerading as the sender to the receiver, and vice versa.
- In a network communication, a man-in-the-middle attack may be preceded by a **session hijacking**, in which an active communication session is intercepted.
- Absolute protection of the system from malicious abuse is not possible, but the cost to the perpetrator can be made sufficiently high to deter most intruders

OPERATING SYSTEMS

Security measures

To protect a system, we must take security measures at four levels:

1. **Physical** - The site or sites containing the computer systems must be physically secured against armed or surreptitious entry by intruders
2. **Human** - Authorization must be done carefully to assure that only appropriate users have access to the system
3. **Operating system** - The system must protect itself from accidental or purposeful security breaches.
4. **Network** - Much computer data in modern systems travels over private leased lines, shared lines like the Internet, wireless connections, or dial-up lines. Intercepting these data could be just as harmful as breaking into a computer, and interruption of communications could result in diminishing users' trust in the system

Program Threats

- Processes, along with the kernel, are the only means of accomplishing work on a computer.
- Therefore, writing a program that creates a breach of security, or causing a normal process to change its behavior and create a breach, is a common goal of attackers

Trojan Horse

- A code segment that misuses its environment is called a **Trojan horse**
- Many systems have mechanisms for allowing programs written by users to be executed by other users. If these programs are executed in a domain that provides the access rights of the executing user, the other users may misuse these rights

Program Threats

Trojan Horse

- A variation of the Trojan horse is a program that emulates a login program. An unsuspecting user starts to log in at a terminal and notices that he has apparently mistyped his password. He tries again and is successful. What has happened is that his authentication key and password have been stolen by the login emulator, which was left running on the terminal by the thief.
- Another variation on the Trojan horse is **spyware**. Spyware sometimes accompanies a program that the user has chosen to install. Most frequently, it comes along with freeware or shareware programs, but sometimes it is included with commercial software.
- The goal of spyware is to or capture information from the user's system and return it to a central Site
- Spyware is a micro example of a macro problem: violation of the principle of least privilege

Program Threats

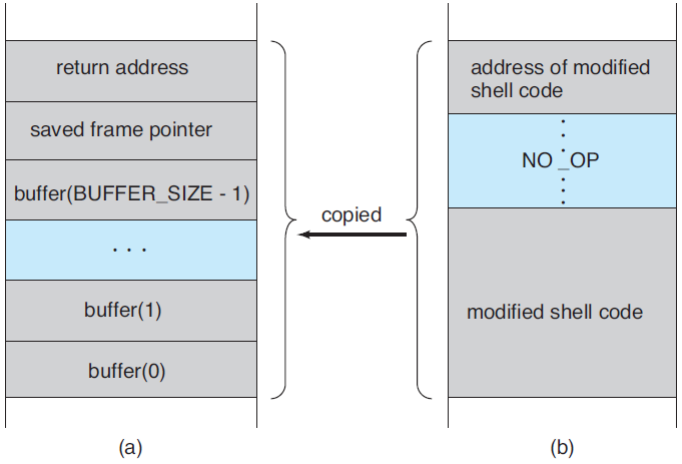
Trap Door

- The designer of a program or system might leave a hole in the software that only he/she is capable of using
- For instance, the code might check for a specific user ID or password, and it might circumvent normal security procedures.
- A clever trap door could be included in a compiler. The compiler could generate standard object code as well as a trap door, regardless of the source code being compiled
- Trap doors pose a difficult problem because, to detect them, we have to analyze all the source code for all components of a system

Program Threats

Stack and Buffer overflow

- The stack- or buffer-overflow attack is the most common way for an attacker outside the system, on a network or dial-up connection, to gain unauthorized access to the target system.
- The attack exploits a bug in a program. The bug can be a simple case of poor programming, in which the programmer neglected to code bounds checking on an input field. In this case, the attacker sends more data than the program was expecting.



Hypothetical stack frame for (a) before and (b) after.

OPERATING SYSTEMS

Program Threats

- **Viruses**
- A virus is a fragment of code embedded in a legitimate program.
- Viruses are self-replicating and are designed to “infect” other programs.
- They can wreak havoc in a system by modifying or destroying files and causing system crashes and program malfunctions.
- As with most penetration attacks, viruses are very specific to architectures, operating systems, and applications.
- Once a virus reaches a target machine, a program known as a **virus dropper** inserts the virus into the system.
- The virus dropper is usually a Trojan horse, executed for other reasons but installing the virus as its core activity. Once installed, the virus may do any one of a number of things.

OPERATING SYSTEMS

System Threats

- System and network threats involve the abuse of services and network connections.
- System and network threats create a situation in which operating-system resources and user files are misused.
- Sometimes, a system and network attack is used to launch a program attack, and vice versa.
- The more ***open*** an operating system is—the more services it has enabled and the more functions it allows—the more likely it is that a bug is available to exploit

Worms

- A **worm** is a process that uses the **spawn** mechanism to duplicate itself.
- The worm spawns copies of itself, using up system resources and perhaps locking out all other processes

OPERATING SYSTEMS

Defenses against system attacks

- The broadest tool available to system designers and users to thwart attacks is **cryptography**.
- In an isolated computer, the operating system can reliably determine the sender and recipient of all inter-process communication, since it controls all communication channels in the computer.
- In a network of computers, the situation is quite different. A networked computer receives bits “from the wire” with no immediate and reliable way of determining what machine or application sent those bits.
- **Cryptography** is used to constrain the potential senders and/or receivers of a message.
- Modern cryptography is based on secrets called **keys** that are selectively distributed to computers in a network and used to process messages.
- Cryptography enables a recipient of a message to verify that the message was created by some computer possessing a certain key. Similarly, a sender can encode its message so that only a computer with a certain key can decode the message

OPERATING SYSTEMS

Defenses against system attacks

- The broadest tool available to system designers and users to thwart attacks is **cryptography**.
- In an isolated computer, the operating system can reliably determine the sender and recipient of all inter-process communication, since it controls all communication channels in the computer.
- In a network of computers, the situation is quite different. A networked computer receives bits “from the wire” with no immediate and reliable way of determining what machine or application sent those bits.
- **Cryptography** is used to constrain the potential senders and/or receivers of a message.
- Modern cryptography is based on secrets called **keys** that are selectively distributed to computers in a network and used to process messages.
- Cryptography enables a recipient of a message to verify that the message was created by some computer possessing a certain key. Similarly, a sender can encode its message so that only a computer with a certain key can decode the message

Defenses against system attacks

- Constraining the set of potential senders of a message is called **authentication**.
- Authentication is thus complementary to encryption
- Authentication is also useful for proving that a message has not been modified

SSL 3.0

- SSL 3.0 is a cryptographic protocol that enables two computers to communicate securely—that is, so that each can limit the sender and receiver of messages to the other.
- It is perhaps the most commonly used cryptographic protocol on the Internet today, since it is the standard protocol by which web browsers communicate securely with web servers.
- The SSL protocol is initiated by a client *c* to communicate securely with a server.
- Asymmetric cryptography is used so that a client and a server can establish a secure **session key** that can be used for symmetric encryption of the session between the two

Defenses against system attacks

User authentication

- A major security problem for operating systems is **user authentication**.
- The protection system depends on the ability to identify the programs and processes currently executing, which in turn depends on the ability to identify each user of the system
- The most common approach to authenticating a user identity is the use of **passwords**.

Securing passwords

- The UNIX system uses secure hashing to avoid the necessity of keeping its password list secret.
- Because the list is hashed rather than encrypted, it is impossible for the system to decrypt the stored value and determine the original password
- Some systems do not allow the use of dictionary words as passwords
- Multi-factor authentication requires users to provide more than one piece of information to authenticate successfully to an account or Linux host. The additional information may be a one-time password (OTP)

OPERATING SYSTEMS

Defenses against system attacks

ssh

- SSH, or Secure Shell, constitutes a cryptographic network protocol designed to enable secure communication between two systems over networks that may not be secure.
- This protocol is widely employed for remote access to servers and the secure transmission of files between computers. In essence, SSH acts as a secure conduit, establishing a confidential channel for communication in scenarios where the network may pose security risks
- This technology is instrumental for professionals seeking a reliable and secure method of managing servers and transferring sensitive data across computers in a controlled and protected manner.
- ssh runs at TCP/IP port 22.
- https://www.geeksforgeeks.org/ssh-command-in-linux-with-examples/

Implementing Security Defenses

Security policy

- The first step toward improving the security of any aspect of computing is to have a **security policy**.
- Policies vary widely but generally include a statement of what is being secured
- Without a policy in place, it is impossible for users and administrators to know what is permissible, what is required, and what is not allowed

Vulnerability Assessment

- The core activity of most vulnerability assessments is a **penetration test**, in which the entity is scanned for known vulnerabilities.
- Vulnerability scans typically focus on short or easy-to-guess passwords, unauthorized privileged programs such as setuid programs, unauthorized programs in system directories, Improper directory protections on user and system directories and changes to system programs detected with checksum values

Implementing Security Defenses

Intrusion detection

- **Intrusion detection**, as its name suggests, strives to detect attempted or successful intrusions into computer systems and to initiate appropriate responses to the intrusions
- Intrusion Detection (IDS) systems raise an alarm when an intrusion is detected, while Intrusion Prevention (IDP) systems act as routers, passing traffic unless an intrusion is detected at which point that traffic is blocked
- **Anomaly detection**, attempts through various techniques to detect anomalous behavior within computer systems. Of course, not all anomalous system activity indicates an intrusion, but the presumption is that intrusions often induce anomalous behavior
- An example of an anomaly-detection tool is the **Tripwire file system** integrity checking tool for UNIX, developed at Purdue University. Tripwire operates on the premise that many intrusions result in modification of system directories and files.

Implementing Security Defenses

Virus protection

- The best protection against computer viruses is prevention, or the practice of **safe computing**.
- Antivirus programs are often used to provide this protection.
- Some of these programs are effective against only particular known viruses. They work by searching all the programs on a system for the specific pattern of instructions known to make up the virus.
- When they find a known pattern, they remove the instructions, **disinfecting** the program.
- Antivirus programs may have catalogs of thousands of viruses for which they search

Implementing Security Defenses

Auditing, Accounting, and Logging

- Auditing, accounting, and logging can decrease system performance, but they are useful in several areas, including security.
- Logging can be general or specific. All system-call executions can be logged for analysis of program behavior (or misbehavior). More typically, suspicious events are logged.
- Authentication failures and authorization failures can tell us quite a lot about break-in attempts.
- Accounting can be used to find performance changes, which in turn can reveal security problems.

Implementing Security Defenses

Firewalling to Protect Systems and Networks

- A **firewall** is a computer, appliance, or router that sits between the trusted and the untrusted.
- A network firewall limits network access between the two **security domains** and monitors and logs all connections.
- It can also limit connections based on source or destination address, source or destination port, or direction of the connection.
- For instance, web servers use HTTP to communicate with web browsers. A firewall therefore may allow only HTTP to pass from all hosts outside the firewall to the web server within the firewall.
- Of course, a firewall itself must be secure and attack-proof. Otherwise, its ability to secure connections can be compromised
- **System-call firewalls** sit between applications and the kernel, monitoring system-call execution.

THANK YOU

Suresh Jamadagni
Department of Computer Science Engineering

sureshjamadagni@pes.edu