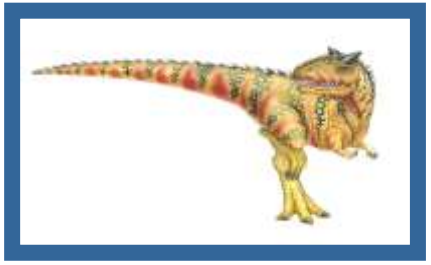
CS 3104 – OPERATING SYSTEMS

DCISM

Chapter 13 - File-System Interface



CHAPTER 13: FILE-SYSTEM INTERFACE

- File Concept
- Access Methods
- Directory Structure
- Protection
- Memory-Mapped Files

File-System Interface

- The file system provides the mechanism for on-line storage of and access to both data and programs of the operating system and all users of the computer system.

- It consists of two distinct parts:
  - a collection of files, each storing related data
  - a directory structure, which organizes and provides information about all the files in the system.

File Concept

- A file is a named collection of related information that is recorded on secondary storage.
- They are usually mapped by the operating system onto physical devices that are usually nonvolatile (persists between system reboots).
- Files commonly represent programs and data.

File Attributes

- A file's attributes vary from one operating system to another but typically consists of the following:
  - **Name** - Symbolic file name kept in human readable form.
  - **Identifier** - A unique tag that identifies the file within the system.
  - **Type** - Needed for systems that support different types of files.
  - **Location** - A pointer to a device and location of the file on that device.
  - **Size** - Size of the file in bytes, words, or blocks.
  - **Protection** - Determines who can do reading, writing, executing, etc.
  - **Timestamps and user identification** - Kept for creation, last modification, and last use.

File Operations

- The operating system can provide system calls to create, write, read, reposition, delete, and truncate files.
- Most operating systems require that files be opened before access, and closed after all access is complete.
- Information about currently open files is stored in an open file table containing the following:
  - File pointer - records the current position in the file
  - File-open count - How many times the current file has been opened.
  - Disk location of the file
  - Access rights

File Operations

- Some systems provide support for file locking.
- File locks allow one process to lock a file and prevent other processes from gaining access to it.
  - A shared lock is for reading only.
  - An exclusive lock is for writing and reading.
  - An advisory lock is informational only and not enforced.
  - A mandatory lock is enforced.
- UNIX uses advisory locks, while Windows uses mandatory locks.

File Types

- In designing a file system, we always consider whether the operating system should recognize and support file types.
- A file type is usually implemented by including it as a part of the file name. The file name is separated into the name itself, and an extension.
- The system uses the extension to indicate the file type and the different operations that can be done on that file.
- The UNIX system uses a magic number stored at the beginning of some binary files to indicate the type of data in the file. (Not all files have magic numbers)
- Extensions can be used or ignored by a given application, depending on the application's programmer.

Common File Types

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, perl, asm	source code in various languages
batch	bat, sh	commands to the command interpreter
markup	xml, html, tex	textual data, documents
word processor	xml, rtf, docx	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	gif, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	rar, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, mp3, mp4, avi	binary file containing audio or A/V information

File Structure

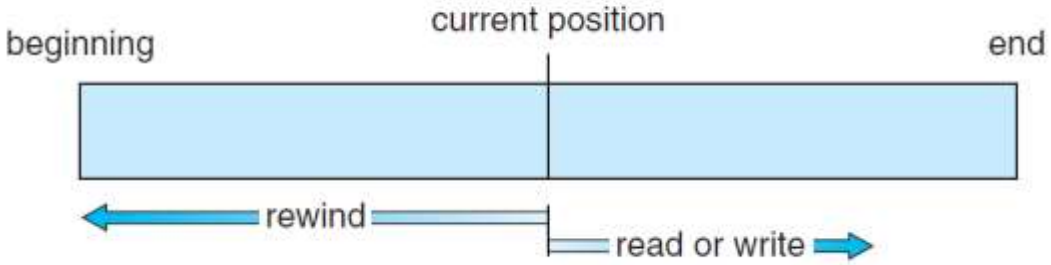
- File types can be used to indicate the internal structure of the file.
- Source and object files have structures that match the expectations of the programs that read them.
- The structure must be understood by the operating system.
- Operating systems that support multiple file structures tend to be large and cumbersome.
- UNIX treats all files as sequence of bytes, with no further consideration of the internal structure.
- Macintosh files have two forks - a resource fork and a data fork.
  - The resource fork contains information relating to the UI.
  - The data fork contains the code or data.

Internal File Structure

- Disk files are accessed in units of physical blocks, typically 512 bytes.
- Files are organized in units of logical units, which may be a single byte or up to a larger size depending on the data structure..
- The number of logical units which fit into one physical block determines its packing, and has an impact on the amount of internal fragmentation that occurs.
- Because disk space is always allocated in blocks, some portion of the last block of each file is generally wasted.
- All file systems suffer from internal fragmentation.

Access Method: Sequential Access

- The simplest access method, the information is processed in order, one record after the other.
- Reads and writes make up the bulk of the operations on a file.
- This mode of access is by far the most common.



Access Method: Direct Access

- The file is viewed as a numbered sequence of blocks or records.
- There are no restrictions on the order of reading or writing for a direct-access file.
- The file operations must be modified to include the block number.
  - Read(n), where n is the block number rather than read next()
  - Write(n)
  - Jump to record n
  - Query current record
- Sequential access can be easily emulated using direct access, but not the other way around.

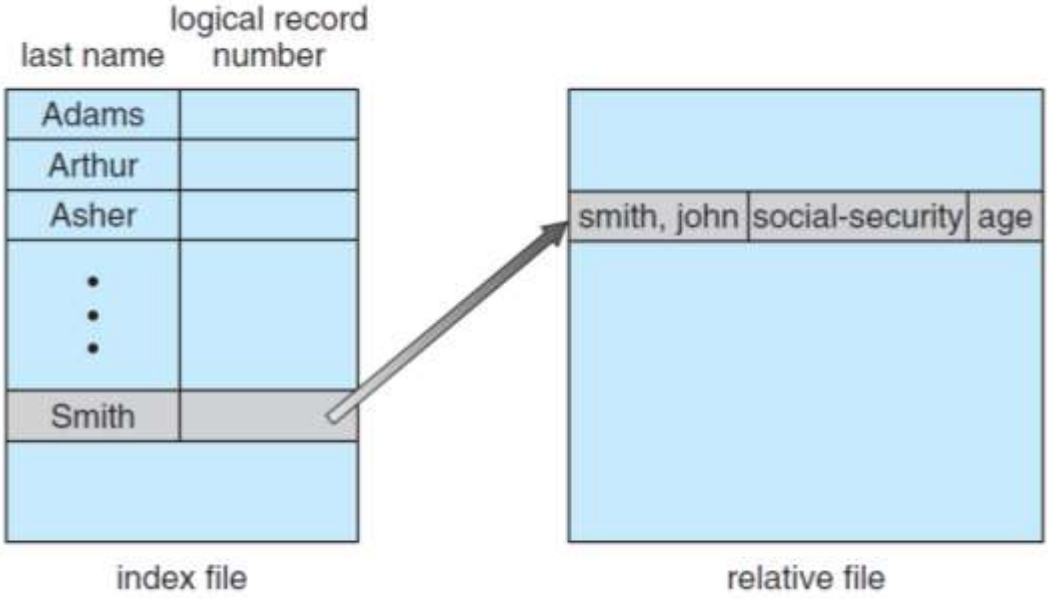
Simulation of Sequential Access on a Direct-Access File

sequential access	implementation for direct access
reset	cp = 0;
read_next	read cp; cp = cp + 1;
write_next	write cp; cp = cp + 1;

Other Access Methods

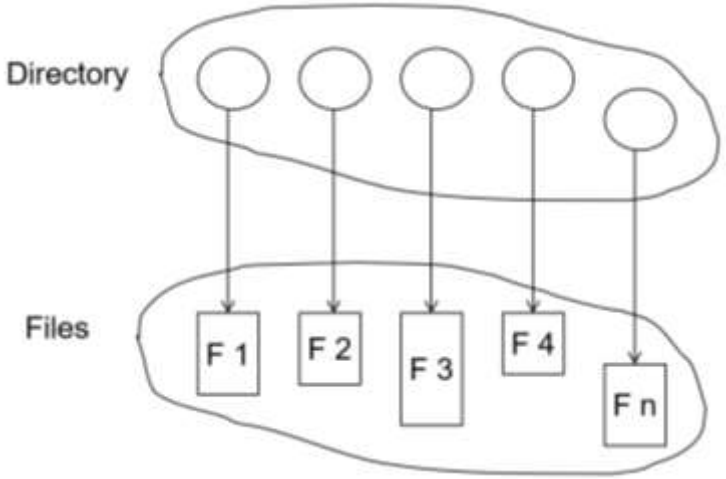
- Other access methods can be built on top of a direct-access method.
- These methods generally involve the construction of an index for the file.
- The index contains pointers to various blocks.
- To find a record in the file, we first search the index and then use the pointer to access the file directly and to find the desired record.
- With large files, the index file itself may become too large to be kept in memory.
- One solution is to create an index for the index file.

Other Access Methods



Directory Structure

- A collection of nodes containing information about all files

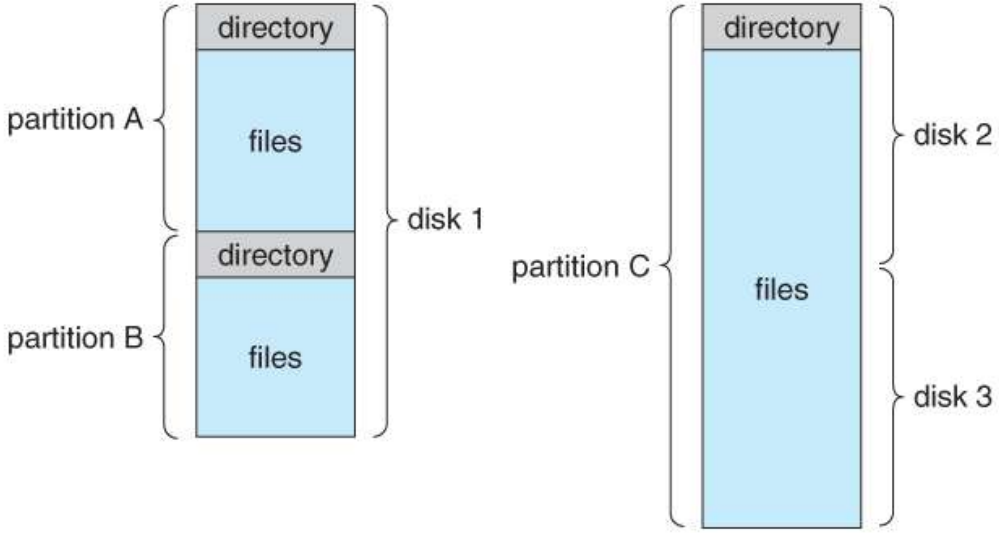


- Both the directory structure and the files reside on disk

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

A Typical File-System Organization



Types of File Systems

- We mostly talk of general-purpose file systems
- But systems frequently have many file systems:
  - **some general- purpose**
  - **some special- purpose**

- Consider **Solaris**, it 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

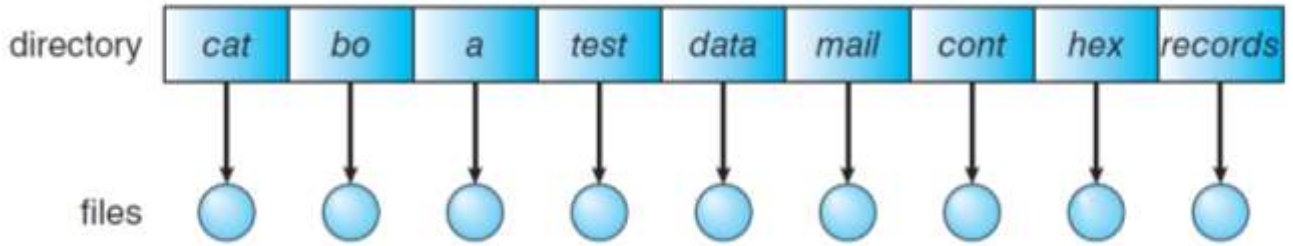
- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

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, etc.)

Single-Level Directory

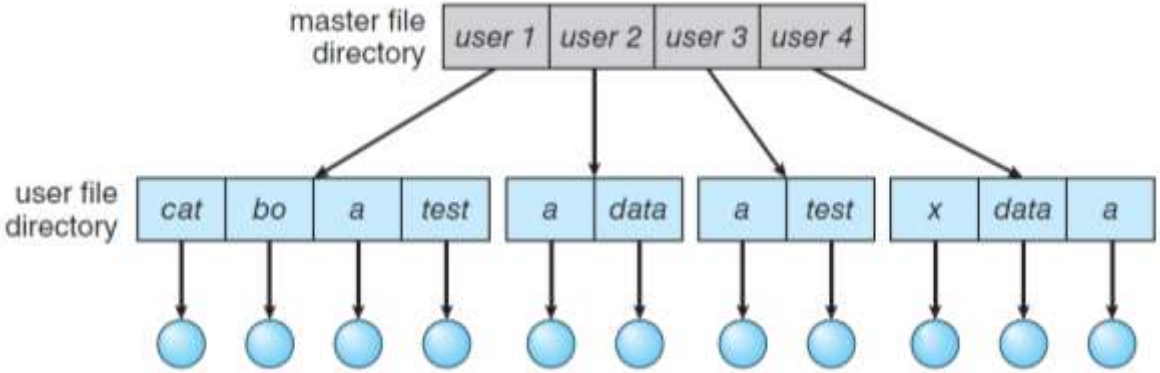
- A single directory for all users



- Naming problem
- Grouping problem

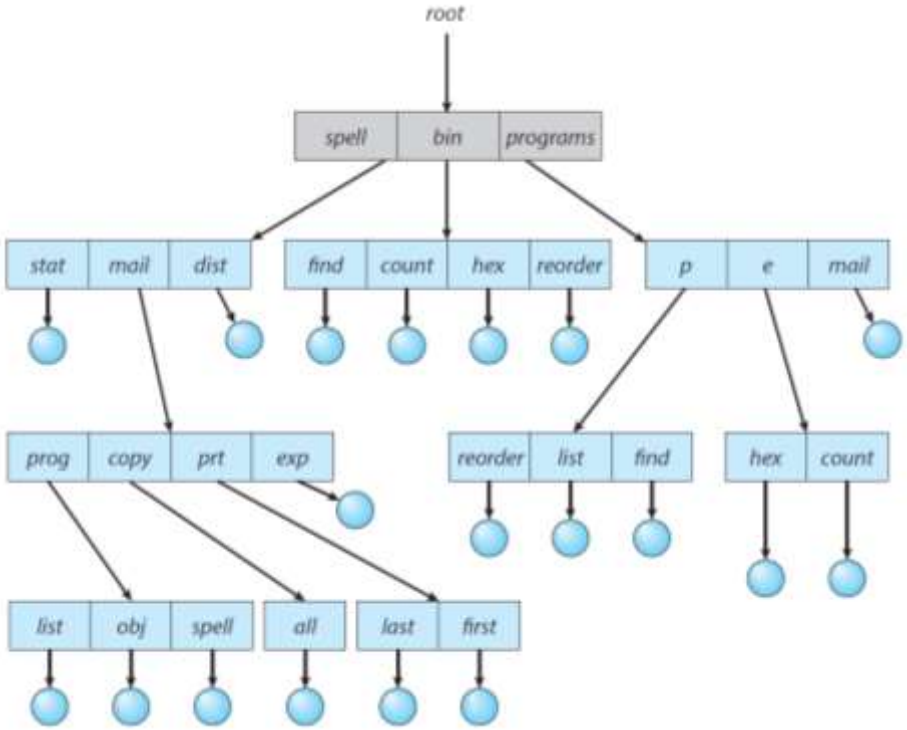
Two-Level Directory

- Separate directory for each user



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

Tree-Structured Directories



Tree-Structured Directories

- Efficient searching
- Grouping Capability
- Current directory (working directory)
  - `cd /spell/mail/prog`
  - `type list`

Tree-Structured Directories

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

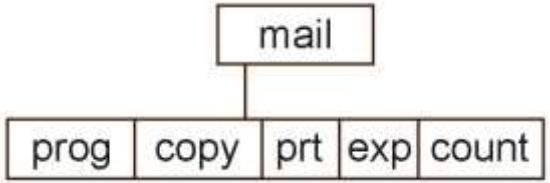
```
rm <file-name>
```

- Creating a new subdirectory is done in current directory

```
mkdir <dir-name>
```

Example: if in current directory `/mail`

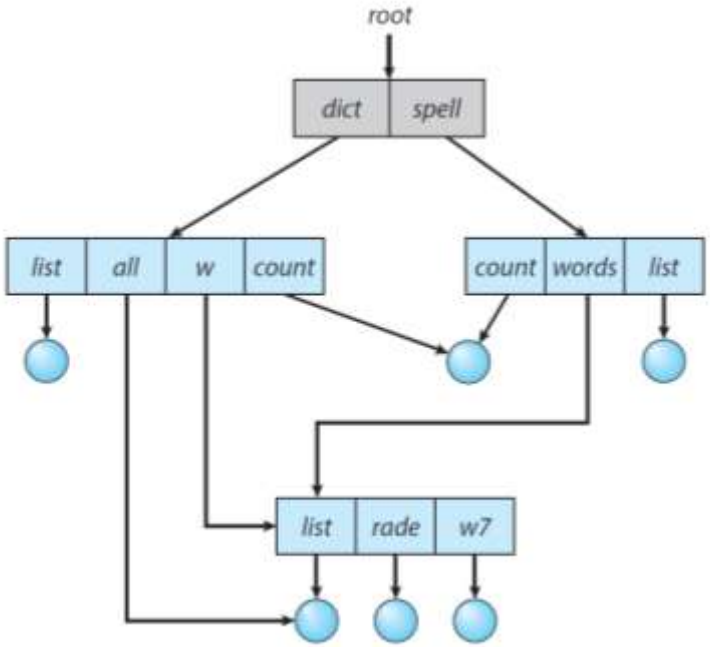
```
mkdir count
```



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

Acyclic-Graph Directories

- Have shared subdirectories and files



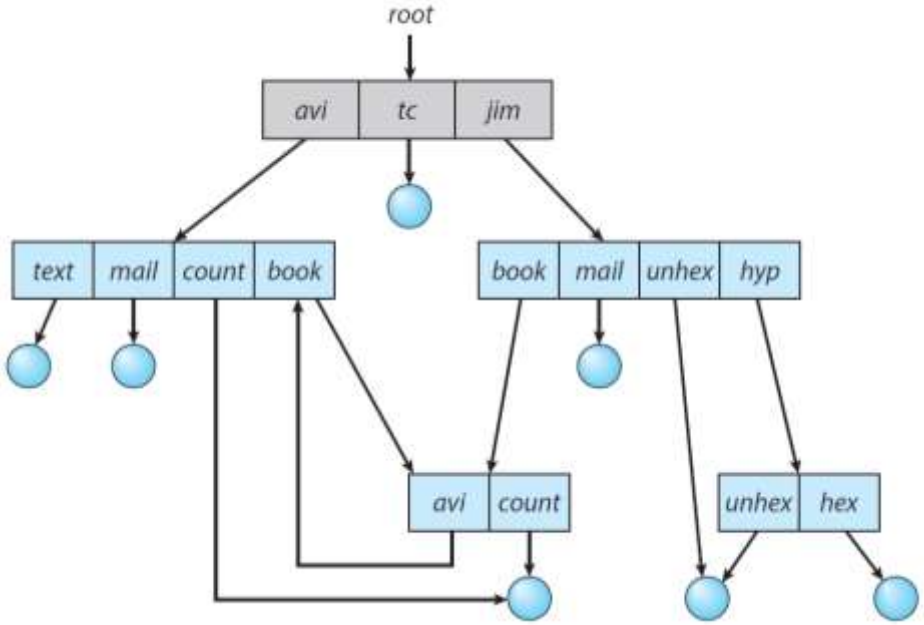
Acyclic-Graph Directories

- 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

General Graph Directory



General Graph Directory

- **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

Protection

- **File owner/creator should be able to control:**
  - what can be done
  - by whom

- **Types of Access:**
  - **Read**. Read from the file.
  - **Write**. Write or rewrite the file.
  - **Execute**. Load the file into memory and execute it.
  - **Append**. Write new information at the end of the file.
  - **Delete**. Delete the file and free its space for possible reuse.
  - **List**. List the name and attributes of the file.
  - **Attribute change**. Changing the attributes of the file.

Access Lists and Groups

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

  RWX

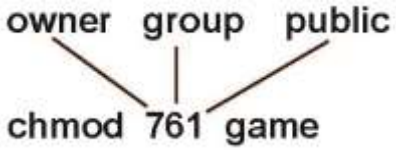
  a) **owner access** 7 ⇒ 1 1 1

  RWX

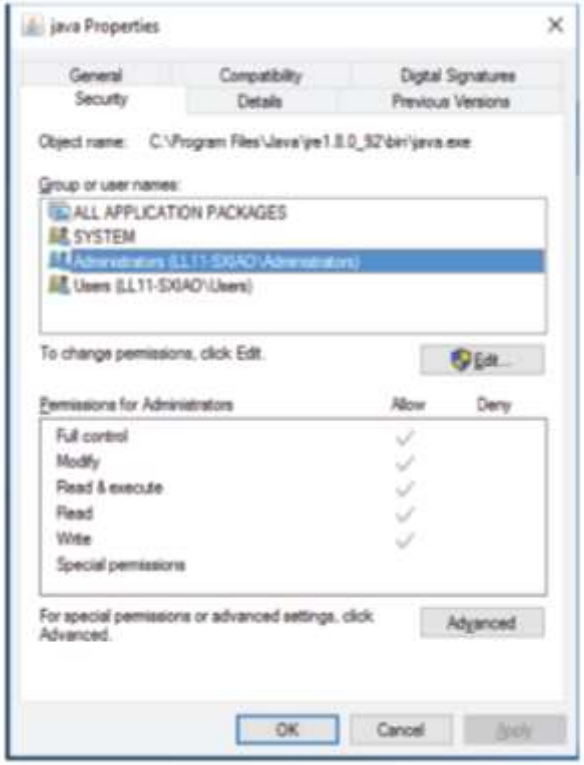
  b) **group access** 6 ⇒ 1 1 0

  RWX

  c) **public access** 1 ⇒ 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.



Windows 10 Access Control List Management



A Sample UNIX Directory Listing

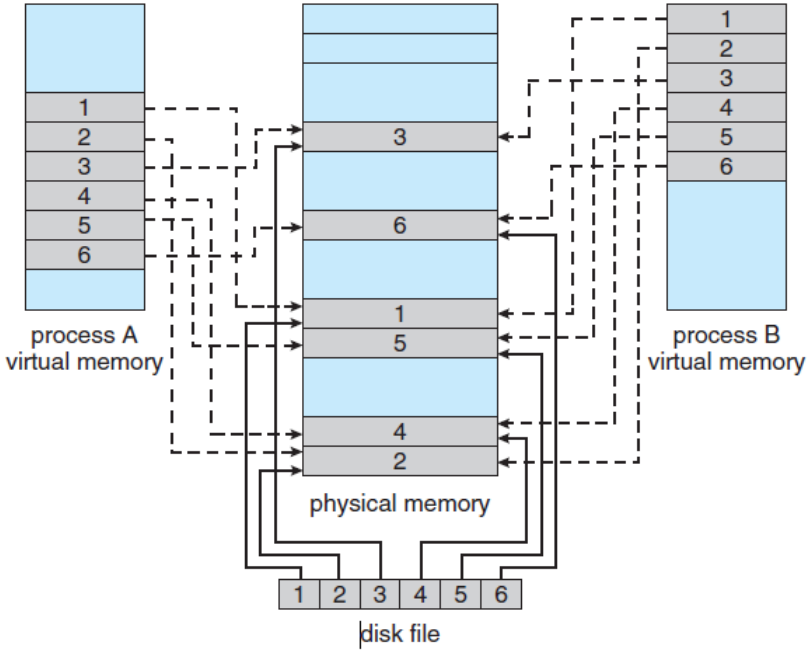
-rw-rw-r--	1 pbg	staff	31200	Sep 3 08:30	intro.ps
drwx------	5 pbg	staff	512	Jul 8 09.33	private/
drwxrwxr-x	2 pbg	staff	512	Jul 8 09:35	doc/
drwxrwx---	2 jwg	student	512	Aug 3 14:13	student-proj/
-rw-r--r--	1 pbg	staff	9423	Feb 24 2017	program.c
-rwxr-xr-x	1 pbg	staff	20471	Feb 24 2017	program
drwx--x--x	4 tag	faculty	512	Jul 31 10:31	lib/
drwx------	3 pbg	staff	1024	Aug 29 06:52	mail/
drwxrwxrwx	3 pbg	staff	512	Jul 8 09:35	test/

Memory-Mapped Files

- Mapping a disk block to a page (or pages) in memory
- Allows a part of the virtual address space to be logically associated with the file
- Uses the virtual memory techniques to treat file I/O as routine memory accesses

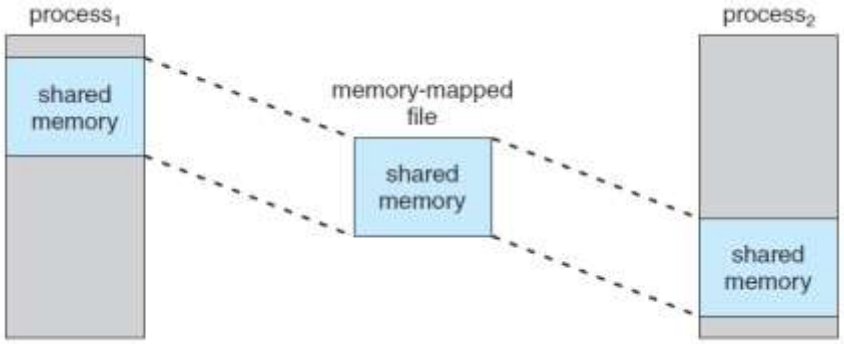
Memory-Mapped Files

- The virtual memory map of each sharing process points to the same page of physical memory (the page that holds a copy of the disk block)



Shared Memory using Memory-Mapped I/O

- Processes can communicate using shared memory by having the communicating processes memory-map the same file into their virtual address spaces



Shared Memory in the Windows API

Producer writing to shared memory using the Windows API

```
#include <windows.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
   HANDLE hFile, hMapFile;
   LPVOID lpMapAddress;

   hFile = CreateFile("temp.txt", /* file name */
      GENERIC_READ | GENERIC_WRITE, /* read/write access */
      0, /* no sharing of the file */
      NULL, /* default security */
      OPEN_ALWAYS, /* open new or existing file */
      FILE_ATTRIBUTE_NORMAL, /* routine file attributes */
      NULL); /* no file template */

   hMapFile = CreateFileMapping(hFile, /* file handle */
      NULL, /* default security */
      PAGE_READWRITE, /* read/write access to mapped pages */
      0, /* map entire file */
      0,
      TEXT("SharedObject")); /* named shared memory object */

   lpMapAddress = MapViewOfFile(hMapFile, /* mapped object handle */
      FILE_MAP_ALL_ACCESS, /* read/write access */
      0, /* mapped view of entire file */
      0,
      0);

   /* write to shared memory */
   sprintf(lpMapAddress,"Shared memory message");

   UnmapViewOfFile(lpMapAddress);
   CloseHandle(hFile);
   CloseHandle(hMapFile);
}
```

Consumer reading from shared memory using the Windows API

```
#include <windows.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
   HANDLE hMapFile;
   LPVOID lpMapAddress;

   hMapFile = OpenFileMapping(FILE_MAP_ALL_ACCESS, /* R/W access */
      FALSE, /* no inheritance */
      TEXT("SharedObject")); /* name of mapped file object */

   lpMapAddress = MapViewOfFile(hMapFile, /* mapped object handle */
      FILE_MAP_ALL_ACCESS, /* read/write access */
      0, /* mapped view of entire file */
      0,
      0);

   /* read from shared memory */
   printf("Read message %s", lpMapAddress);

   UnmapViewOfFile(lpMapAddress);
   CloseHandle(hMapFile);
}
```

CS 3104 – OPERATING SYSTEMS

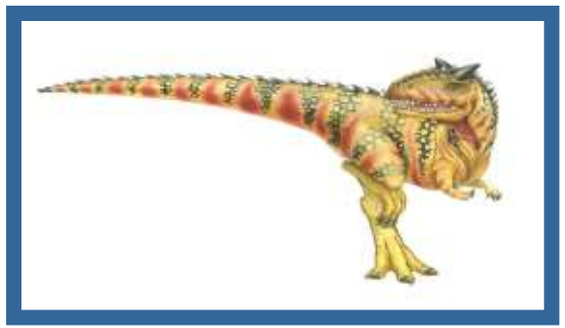
DCISM

End of Chapter 13
File-System Interface

CS 3104 – OPERATING SYSTEMS

DCISM
DEPARTMENT OF COMPUTER, INFORMATION SCIENCES AND MATHEMATICS

Chapter 14: File-System Implementation



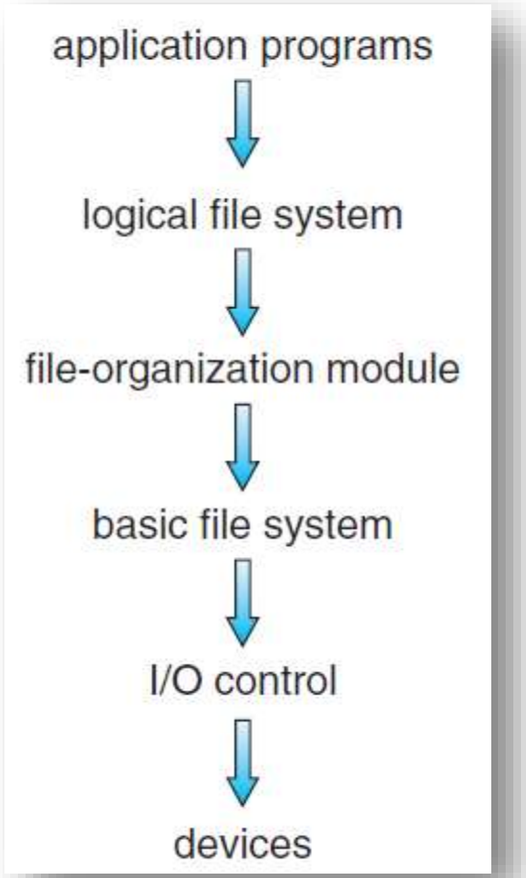
File-System Implementation

- File-System Structure
- File-System Operations
- Directory Implementation
- Allocation Methods
- Free-Space Management
- Efficiency and Performance
- Recovery
- Example: The WAFL File System

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
- **Device driver** controls the physical device
- **File system** organized into **layers**

Layered File System



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 organization module** understands files, logical address, and physical blocks
- Translates logical block # to physical block #
- Manages free space, disk allocation

File System Layers

- **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

- **Layering** is 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

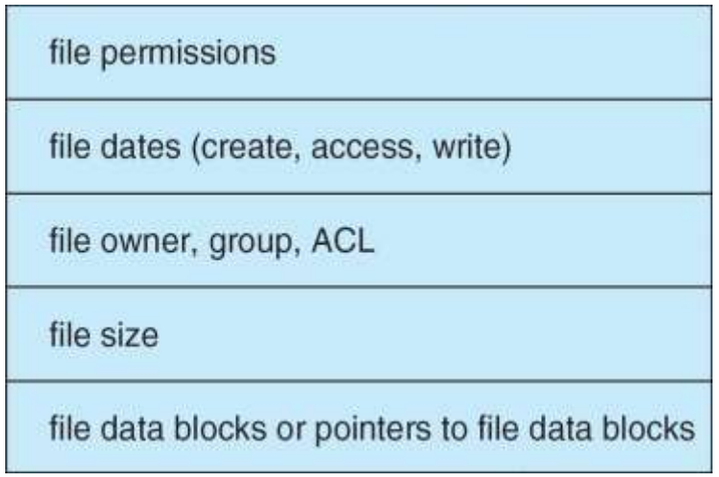
- **Many file systems**, sometimes many within an operating system
  - Each with its own format (CD-ROM is ISO 9660; Unix has **UFS**, FFS; Windows has FAT, FAT32, NTFS as well as floppy, CD, DVD Blu-ray, Linux has more than 130 types, with **extended file system** ext3 and ext4 leading; plus distributed file systems, etc.)
  - New ones still arriving – ZFS, GoogleFS, Oracle ASM, FUSE

File-System Operations

- 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 Operations

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



File-System Operations

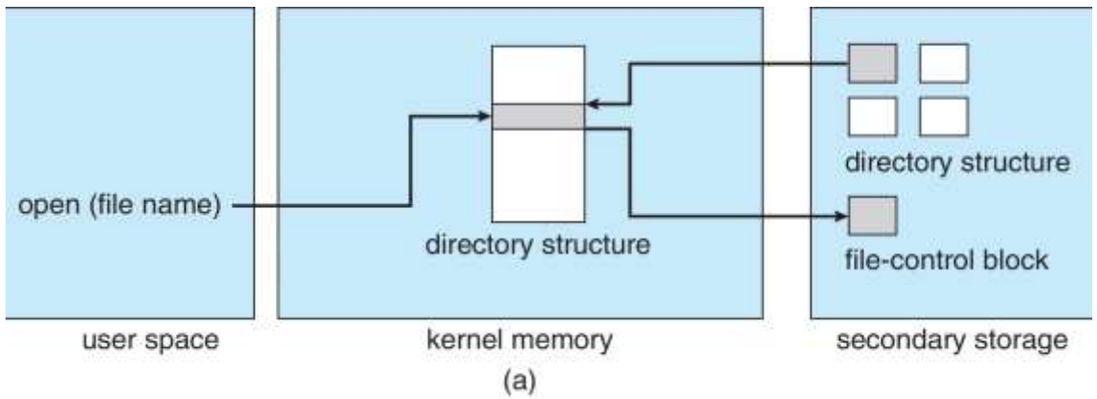
In-Memory File System Structures

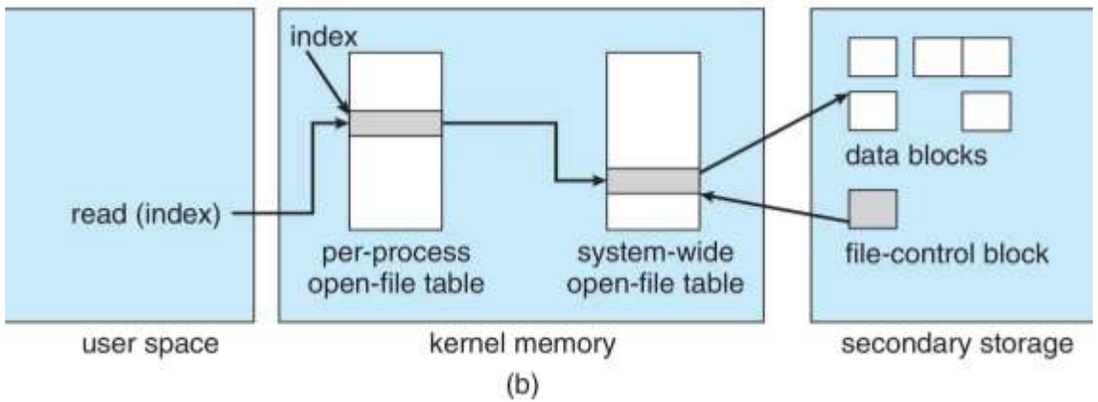
- **Mount table:** storing file system mounts, mount points, file system types
- **System-wide open-file table:** contains a copy of the FCB of each file and other info
- **Per-process open-file table:** contains pointers to appropriate entries in system-wide open-file table as well as other info
- The following figure illustrates the necessary file system structures provided by the operating systems
- Figure 12-3(a) refers to opening a file
- Figure 12-3(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

File-System Operations

In-Memory File System Structures

- The following figures illustrate 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





Directory Implementation

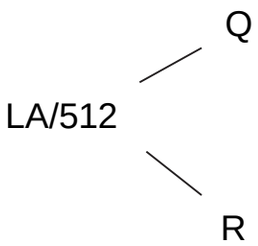
- **Linear list** of file names with pointer to the data blocks
  - Simple to program
  - Time-consuming to execute
    - ✓Linear search time
    - ✓Could keep ordered alphabetically via linked list or use B+ tree
- **Hash Table** – linear list with hash data structure
  - Decreases directory search time
  - **Collisions** – situations where two file names hash to the same location
  - Only good if entries are fixed size, or use chained-overflow method

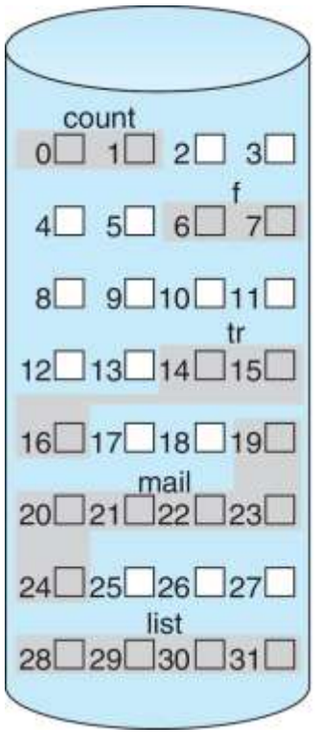
Allocation Methods: Contiguous

- 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
    - ✓ Problems include finding space for file, knowing file size, external fragmentation, need for **compaction off-line** (**downtime**) or **on-line**

Allocation Methods: Contiguous

- **Mapping from logical to physical**





directory

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

Block to be accessed = Q + starting address
Displacement into block = R

Allocation Methods: Contiguous

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

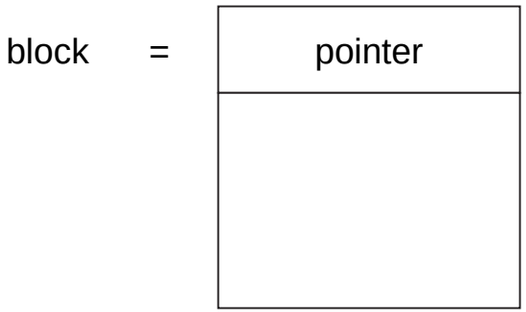
Allocation Methods: Linked

- **Linked allocation** – each file a linked list of blocks
  - File ends at nil 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

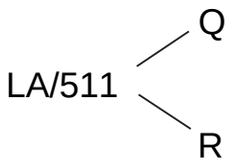
- FAT (File Allocation Table) variation
  - Beginning of volume has table, indexed by block number
  - Much like a linked list, but faster on disk and cacheable
  - New block allocation simple

Allocation Methods: Linked

- **Each file is a linked list of disk blocks:** blocks may be scattered anywhere on the disk

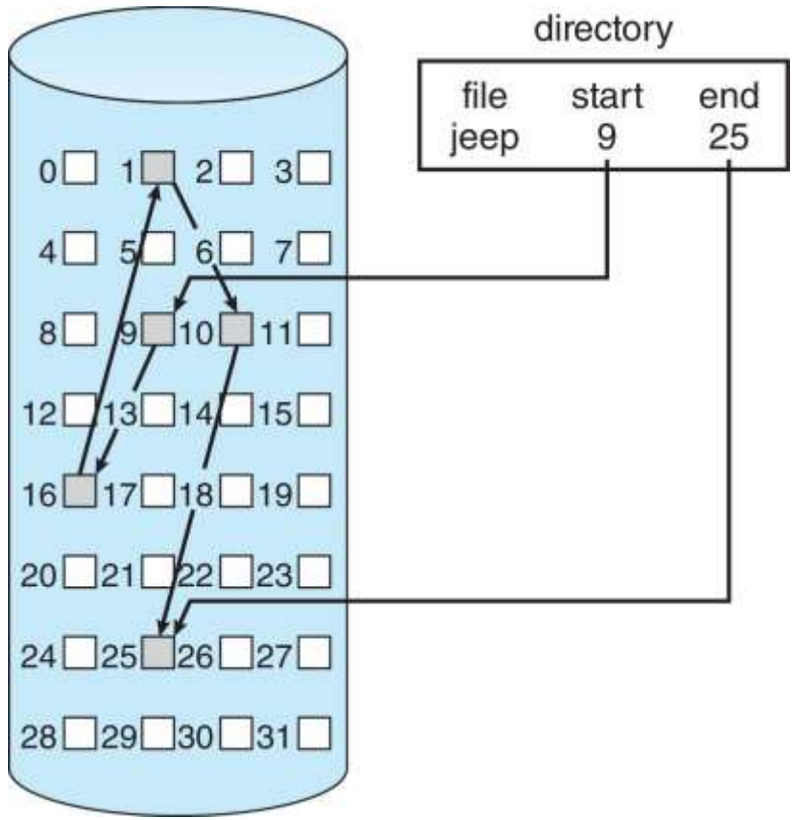


- **Mapping**



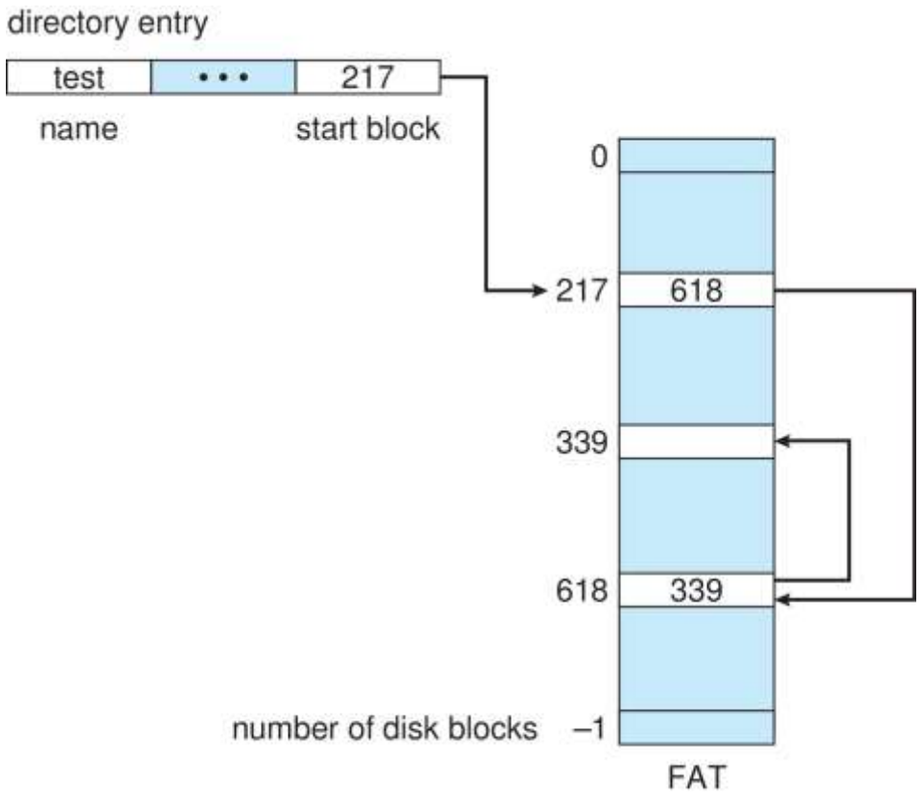
Block to be accessed is the **Qth block** in the linked chain of blocks representing the file.
Displacement into block = R + 1

Allocation Methods: Linked



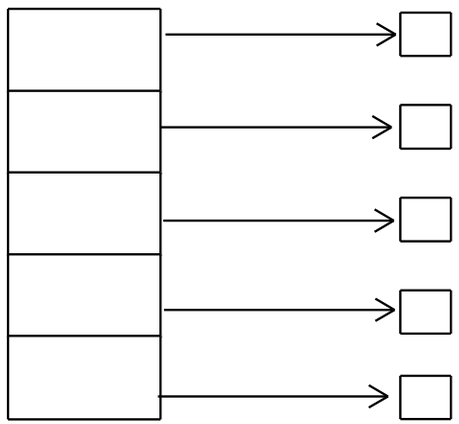
Allocation Methods: Linked

File-Allocation Table

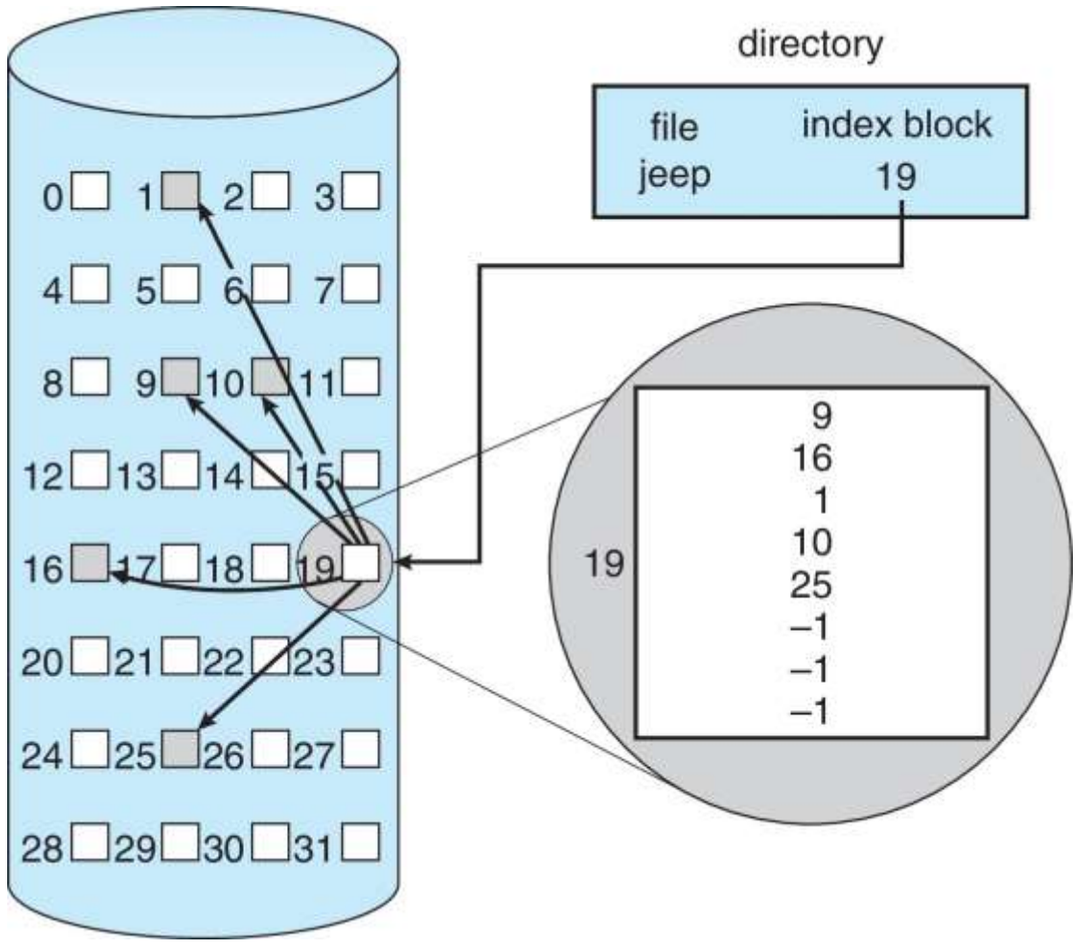


Allocation Methods: Indexed

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

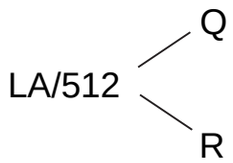


Allocation Methods: Indexed



Allocation Methods: Indexed

- Need index table
- Random access
- Dynamic access without external fragmentation, but have overhead of index block
- Mapping from logical to physical in a file of maximum size of 256K bytes and block size of 512 bytes (only 1 block for index table is needed).

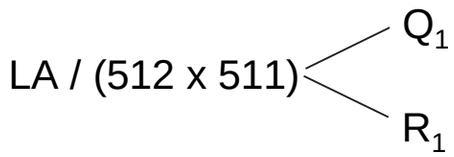


Q = displacement into index table
R = displacement into block

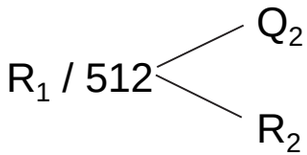
Allocation Methods: Indexed

Mapping

- **Mapping from logical to physical** in a file of unbounded length (block size of 512 words)
- **Linked scheme** – Link blocks of index table (no limit on size)



***Q_1* = block of index table**
***R_1* is used as follows:**

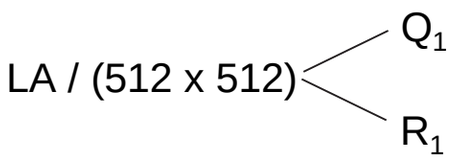


***Q_2* = displacement into block of index table**
***R_2* displacement into block of file**

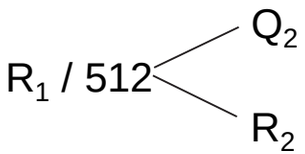
Allocation Methods: Indexed

Mapping

- **Two-level index** (4K blocks could store 1,024 four-byte pointers in outer index -> 1,048,567 data blocks and file size of up to 4GB)



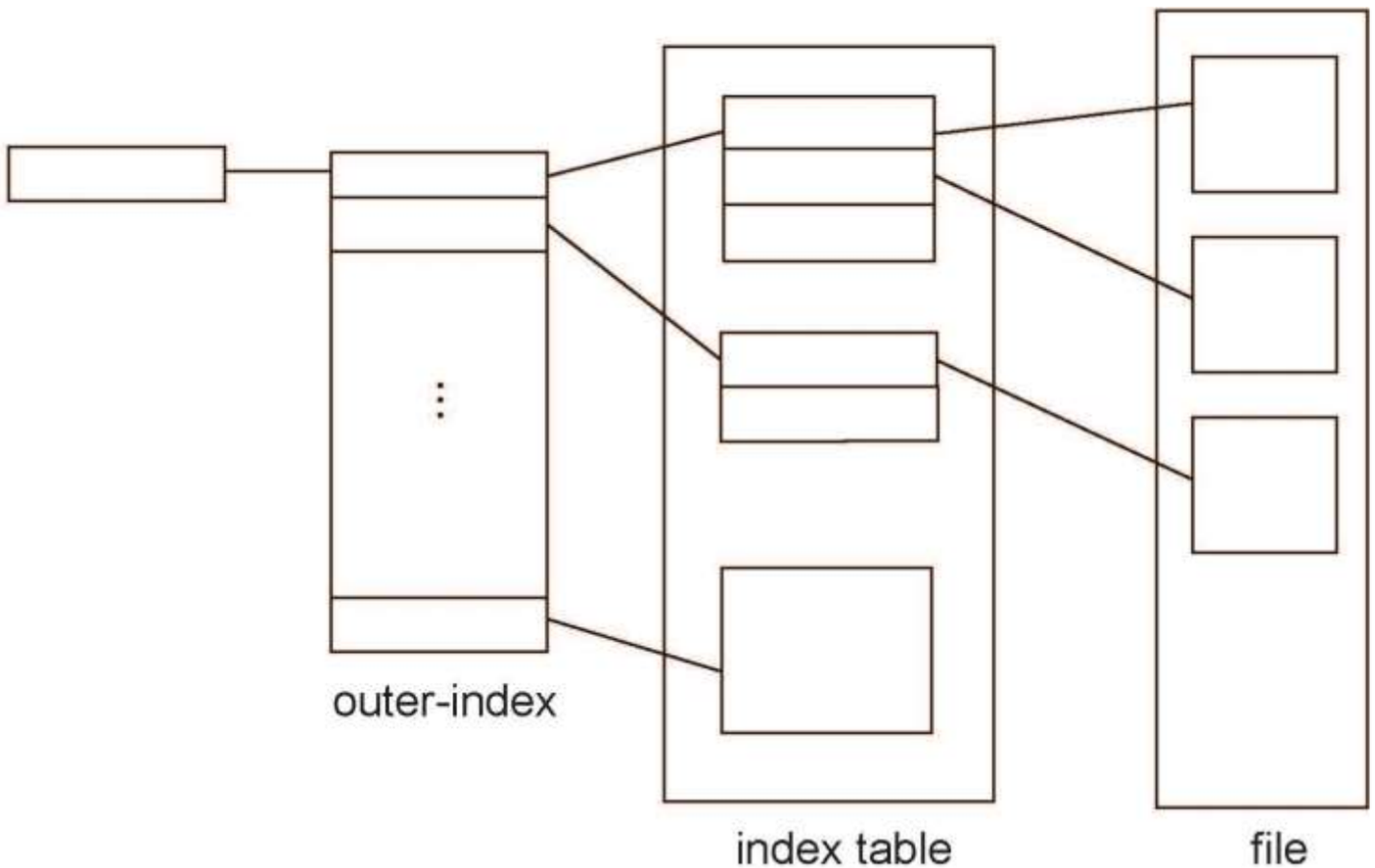
Q_1 = displacement into outer-index
R_1 is used as follows:



Q_2 = displacement into block of index table
R_2 displacement into block of file

Allocation Methods: Indexed

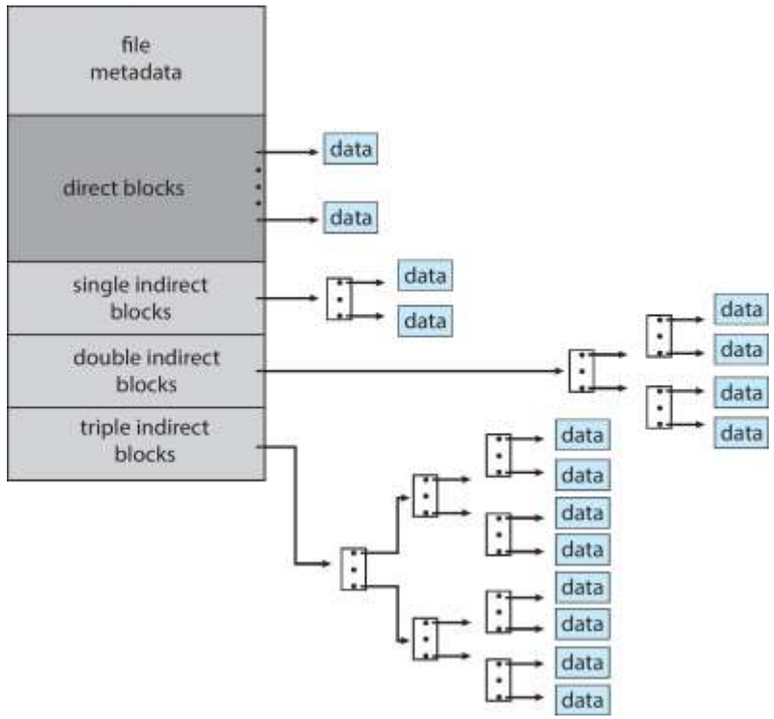
Mapping



Allocation Methods: Indexed

Combined Scheme: UNIX UFS

- 4K bytes per block, 32-bit addresses



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

Allocation Methods: Performance

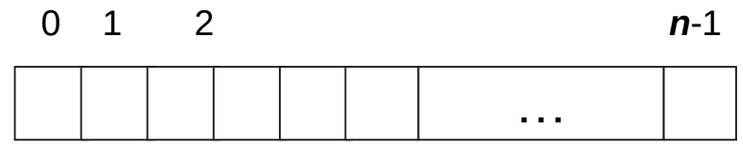
- Best method depends on file access type
  - Contiguous great for sequential and random
- Linked good for sequential, not random
- Declare access type at creation -> select either contiguous or linked
- Indexed more complex
  - Single block access could require 2 index block reads then data block read
  - Clustering can help improve throughput, reduce CPU overhead
- For NVM, no disk head so different algorithms and optimizations needed
  - Using old algorithm uses many CPU cycles trying to avoid non-existent head movement
  - With NVM goal is to reduce CPU cycles and overall path needed for I/O

Allocation Methods: Performance

- **Adding instructions to the execution path to save one disk I/O is reasonable**
  - Intel Core i7 Extreme Edition 990x (2011) at 3.46Ghz = 159,000 MIPS
    - ✓ https://en.wikipedia.org/wiki/Instructions_per_second
  - Typical disk drive at 250 I/Os per second
    - ✓ 159,000 MIPS / 250 = 630 million instructions during one disk I/O
  - Fast SSD drives provide 60,000 IOPS
    - ✓ 159,000 MIPS / 60,000 = 2.65 millions instructions during one disk I/O

Free-Space Management

- File system maintains **free-space list** to track available blocks/clusters
- (Using term “block” for simplicity)
- **Bit vector** or **bit map** (***n*** blocks)



bit[***i***] = { 1 ⇒ block[***i***] free
0 ⇒ block[***i***] occupied

- Block number calculation

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

- CPUs have instructions to return offset within word of first “1” bit

Free-Space Management

- **Bit map requires extra space**
- **Example:**

  block size = 4KB = 2^{12} bytes

  disk size = 2^{40} bytes (1 terabyte)

  n = $2^{40}/2^{12} = 2^{28}$ bits (or 32MB)

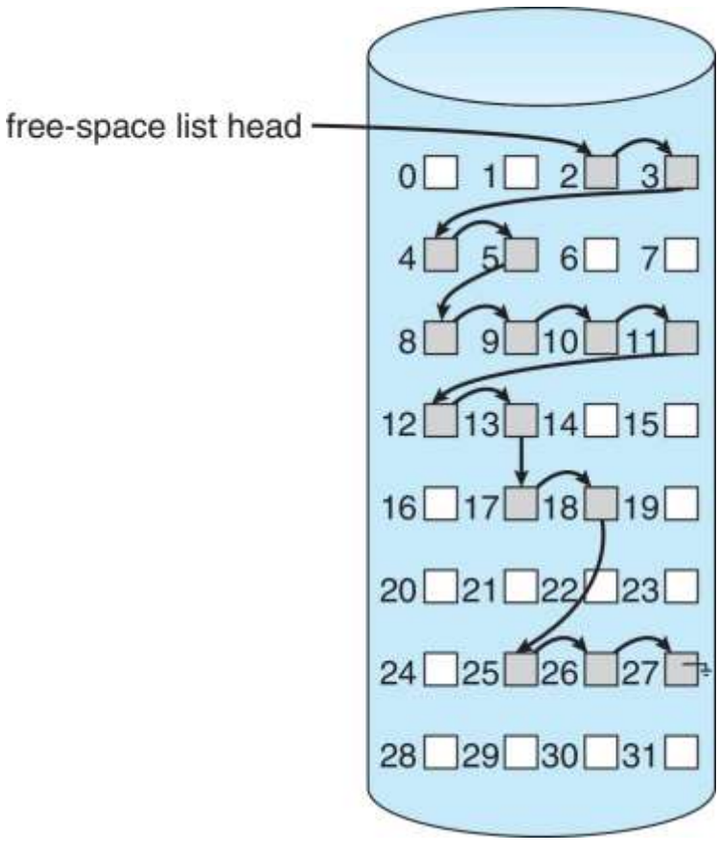
  if clusters of 4 blocks -> 8MB of memory

- Easy to get contiguous files

Free-Space Management

Linked Free-Space List on Disk

- **Linked list (free list)**
  - Cannot get contiguous space easily
  - No waste of space
  - No need to traverse the entire list (if # free blocks recorded)



Free-Space Management

- **Grouping**
  - Modify linked list to store address of next *n-1* free blocks in first free block, plus a pointer to next block that contains free-block-pointers (like this one)

- **Counting**
  - Because space is frequently contiguously used and freed, with contiguous-allocation allocation, extents, or clustering
    - ✓ Keep address of first free block and count of following free blocks
    - ✓ Free space list then has entries containing addresses and counts

Free-Space Management

- **Space Maps**
  - Used in **ZFS**
  - Consider meta-data I/O on very large file systems
    - ✓Full data structures like bit maps couldn't fit in memory -> thousands of I/Os
  - Divides device space into **metaslab** units and manages metaslabs
    - ✓Given volume can contain hundreds of metaslabs
  - Each **metaslab** has associated space map
    - ✓Uses counting algorithm
  - But records to log file rather than file system
    - ✓Log of all block activity, in time order, in counting format
  - **Metaslab** activity -> load space map into memory in balanced-tree structure, indexed by offset
    - ✓Replay log into that structure
    - ✓Combine contiguous free blocks into single entry

Free-Space Management

TRIMing Unused Blocks

- HDDS overwrite in place so need only free list
- Blocks not treated specially when freed
  - Keeps its data but without any file pointers to it, until overwritten
- Storage devices not allowing overwrite (like NVM) suffer badly with same algorithm
  - Must be erased before written, erases made in large chunks (blocks, composed of pages) and are slow
  - TRIM is a newer mechanism for the file system to inform the NVM storage device that a page is free
    - ✓Can be garbage collected or if block is free, now block can be erased

Efficiency and Performance

- **Efficiency** is 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

Efficiency and Performance

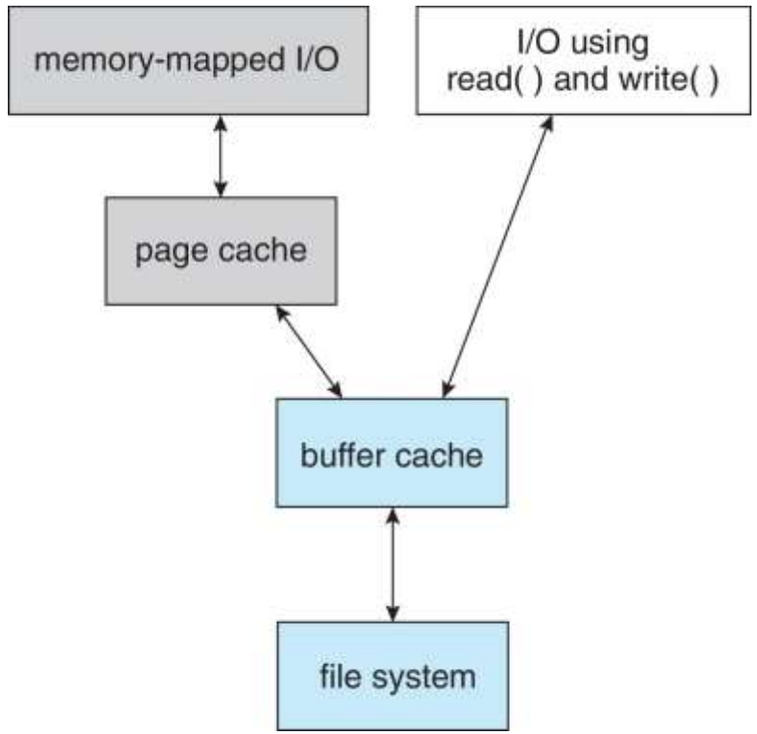
- **Performance**
  - Keeping data and metadata close together
  - **Buffer cache** – separate section of main memory for frequently used blocks
  - **Synchronous** writes sometimes requested by apps or needed by OS
    - ✓No buffering / caching – writes must hit disk before acknowledgement
    - ✓**Asynchronous**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

Efficiency and Performance

Page Cache

- A **page cache** caches pages rather than disk blocks using virtual memory techniques and addresses
- Memory-mapped I/O uses a page cache
- Routine I/O through the file system uses the buffer (disk) cache
- This leads to the figure at the right

I/O Without a Unified Buffer Cache

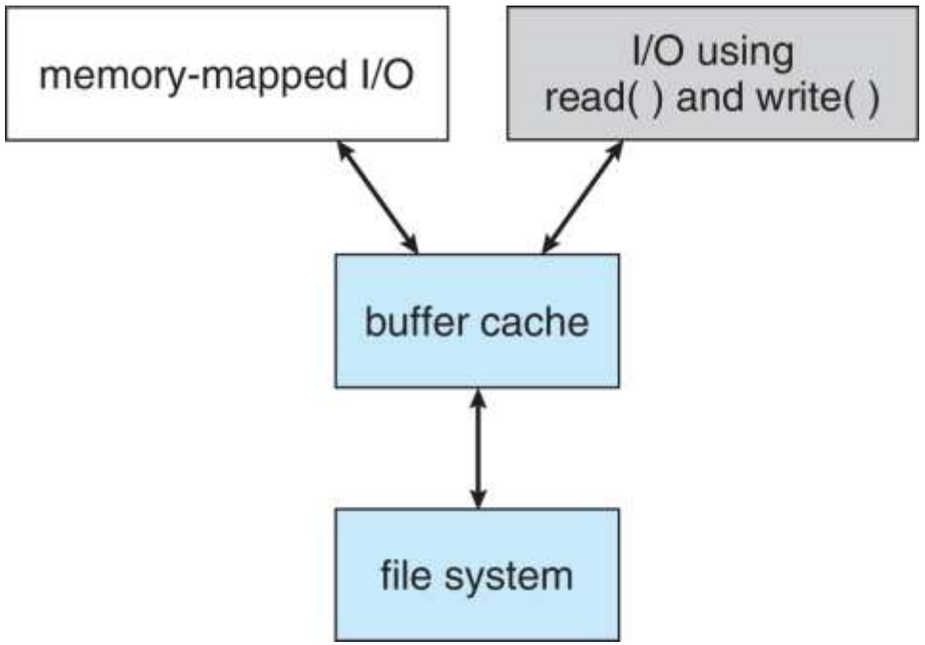


Efficiency and Performance

Unified Buffer Cache

- A **unified buffer cache** uses the same page cache to cache both memory-mapped pages and ordinary file system I/O to avoid **double caching**
- But which caches get priority, and what replacement algorithms to use?

I/O Using a Unified Buffer Cache



Recovery

- **Consistency checking** – compares data in directory structure with data blocks on disk, and tries to fix inconsistencies
  - Can be slow and sometimes fails
- Use system programs to **back up** data from disk to another storage device (magnetic tape, other magnetic disk, optical)
- Recover lost file or disk by **restoring** data from backup

Recovery

Log-Structured File Systems

- **Log structured** (or **journaling**) file systems record each metadata update to the file system as a **transaction**

- **All transactions are written** to a **log**
  - A transaction is considered committed once it is written to the log (sequentially)
  - Sometimes to a separate device or section of disk
  - However, the file system may not yet be updated

- The transactions in the log are asynchronously written to the file system structures
  - When the file system structures are modified, the transaction is removed from the log

- If the file system crashes, all remaining transactions in the log must still be performed
- Faster recovery from crash, removes chance of inconsistency of metadata

Recovery

Other Solutions

- Employed by Network Appliance's WAFL file system and the Solaris ZFS file system
- File system never overwrites blocks with new data, rather, a transaction writes all data and metadata changes to new blocks
- File system can then remove the old pointers and the old blocks and make them available for reuse
- A **snapshot** is created if the old pointers and blocks are kept
  - **Snapshot** is a view of the file system at a specific point in time (before any updates after that time were applied).
- This solution should require no consistency checking if the pointer update is done atomically.

Recovery

Backup and Restore

- Storage devices sometimes fail, and care must be taken to ensure that the data lost in such a failure are not lost forever.
- To this end, system programs can be used to **back up** data from one storage device to another, such as a magnetic tape or other secondary storage device.
- Recovery from the loss of an individual file, or of an entire device, may then be a matter of **restoring** the data from backup.
- To minimize the copying needed, one can use information from each file's directory entry.

Recovery

Backup and Restore

- A **typical backup schedule** may then be as follows:
  - **Day 1**. Copy to a backup medium all files from the file system (this is called a **full backup**).
  - **Day 2**. Copy to another medium all files changed since day 1 (this is an **incremental backup**).
  - **Day 3**. Copy to another medium all files changed since day 2.

  . . .

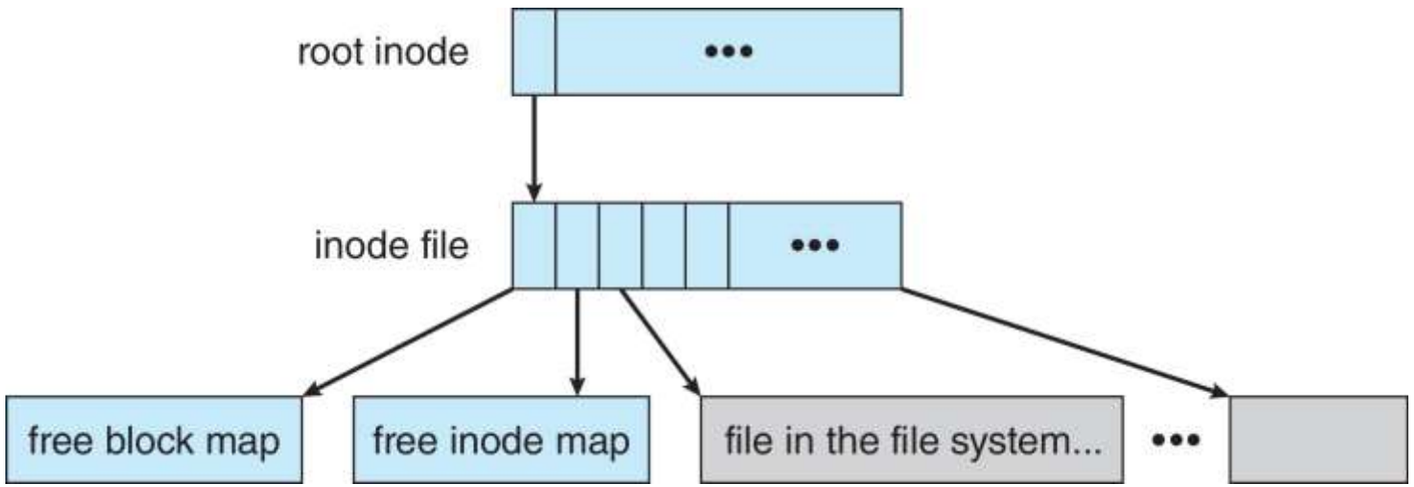
  - **Day *N***. Copy to another medium all files changed since day $N-1$, then go back to day 1.

Example: WAFL File System

- Used on Network Appliance “Filers” – distributed file system appliances
- “Write-anywhere file layout”
- Serves up NFS, CIFS, http, ftp
- Random I/O optimized, write optimized
  - NVRAM for write caching
- Similar to Berkeley Fast File System, with extensive modifications

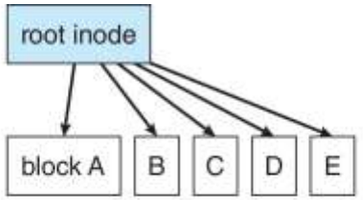
Example: WAFL File System

The WAFL File Layout

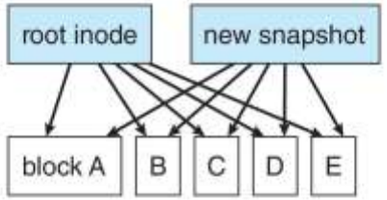


Example: WAFL File System

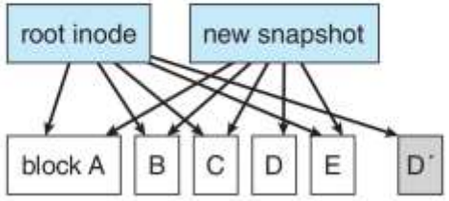
Snapshots in WAFL



(a) Before a snapshot.



(b) After a snapshot, before any blocks change.



(c) After block D has changed to D'.

Example: WAFL File System

THE APPLE FILE SYSTEM

In 2017, Apple, Inc., released a new file system to replace its 30-year-old HFS+ file system. HFS+ had been stretched to add many new features, but as usual, this process added complexity, along with lines of code, and made adding more features more difficult. Starting from scratch on a blank page allows a design to start with current technologies and methodologies and provide the exact set of features needed.

Apple File System (APFS) is a good example of such a design. Its goal is to run on all current Apple devices, from the Apple Watch through the iPhone to the Mac computers. Creating a file system that works in watchOS, I/Os, tvOS, and macOS is certainly a challenge. APFS is feature-rich, including 64-bit pointers, clones for files and directories, snapshots, space sharing, fast directory sizing, atomic safe-save primitives, copy-on-write design, encryption (single- and multi-key), and I/O coalescing. It understands NVM as well as HDD storage.

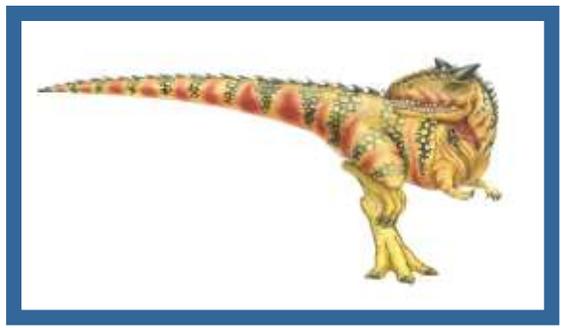
Most of these features we've discussed, but there are a few new concepts worth exploring. Space sharing is a ZFS-like feature in which storage is available as one or more large free spaces (containers) from which file systems can draw allocations (allowing APFS-formatted volumes to grow and shrink). Fast directory sizing provides quick used-space calculation and updating. Atomic safe-save is a primitive (available via API, not via file-system commands) that performs renames of files, bundles of files, and directories as single atomic operations. I/O coalescing is an optimization for NVM devices in which several small writes are gathered together into a large write to optimize write performance.

Apple chose not to implement RAID as part of the new APFS, instead depending on the existing Apple RAID volume mechanism for software RAID. APFS is also compatible with HFS+, allowing easy conversion for existing deployments.

CS 3104 – OPERATING SYSTEMS

DCISM
DEPARTMENT OF COMPUTER, INFORMATION SCIENCES AND MATHEMATICS

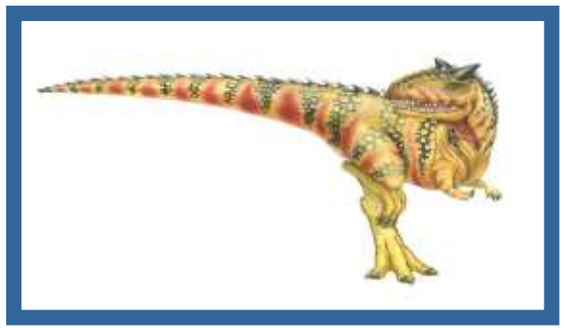
End of Chapter 14
File-System Implementation



CS 3104 – OPERATING SYSTEMS

DCISM
DEPARTMENT OF COMPUTER, INFORMATION SCIENCES AND MATHEMATICS

Chapter 15: File-System Internals



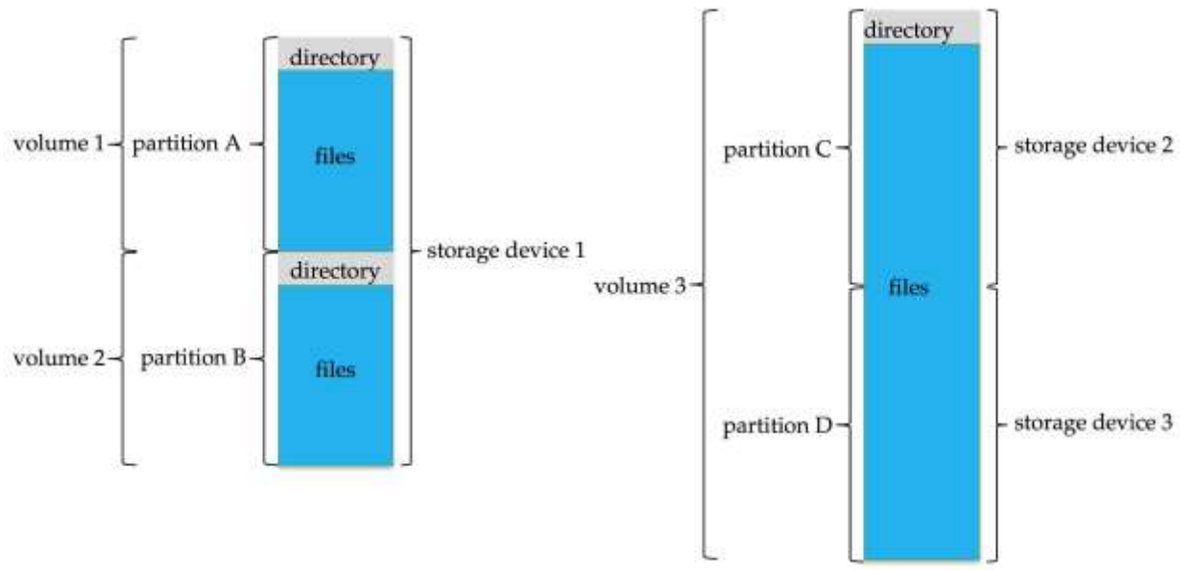
File-System Internals

- File Systems
- File-System Mounting
- Partitions and Mounting
- File Sharing
- Virtual File Systems
- Remote File Systems
- Consistency Semantics
- NFS

File Systems

- General-purpose computers can have multiple storage devices (to store files)
- Devices can be sliced into partitions, which hold volumes
- Volumes can span multiple partitions
- Each volume usually formatted into a file system
- # of file systems varies, typically dozens available to choose from

Typical storage device organization



File Systems

Example Mount Points and File Systems: Solaris

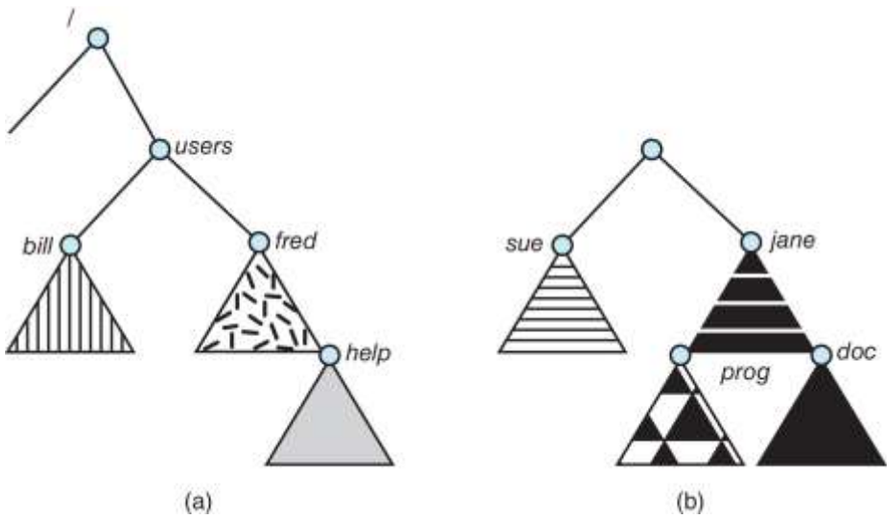
/	ufs
/devices	devfs
/dev	dev
/system/contract	ctfs
/proc	proc
/etc/mnttab	mntfs
/etc/svc/volatile	tmpfs
/system/object	objfs
/lib/libc.so.1	lofs
/dev/fd	fd
/var	ufs
/tmp	tmpfs
/var/run	tmpfs
/opt	ufs
/zpbge	zfs
/zpbge/backup	zfs
/export/home	zfs
/var/mail	zfs
/var/spool/mqueue	zfs
/zpbg	zfs
/zpbg/zones	zfs

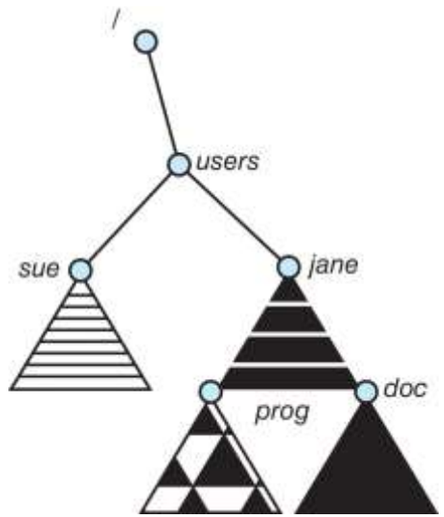
File-System Mounting

- A file system must be mounted before it can be available to processes on the system
- The directory structure may be built out of multiple file-system-containing volumes
  - must be mounted to make them available within the file-system name space.
- Mount procedure is straightforward
  - OS is given the name of the device and the **mount point**
  - **Mount point:** the location within the file structure where the file system is to be attached
- Typically, a mount point is an empty directory

File-System Mounting

File Mounting Illustrations





(a) Unix-like file system directory tree (an existing file system)

(b) Unmounted file system (an unmounted volume residing on /device/dsk)

- After mounting (b) into the existing directory tree
- Volume mounted at /users

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 OSes, other file systems, or be raw
  - Mounted at boot time
  - Other partitions can mount automatically or manually on **mount points** (location at which they can be accessed)
- 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

File Sharing

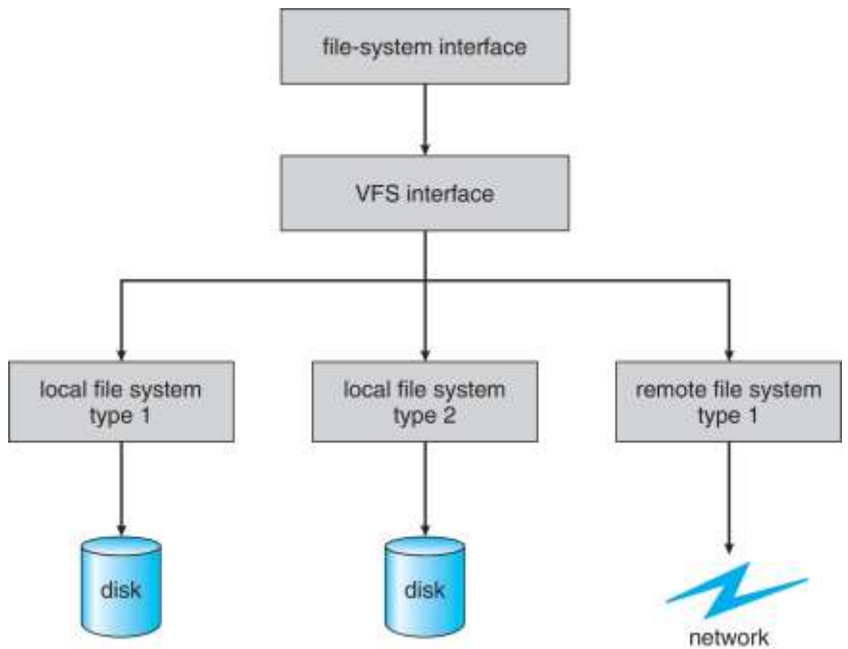
- Allows multiple users / systems access to the same files
- Permissions / protection must be implement and accurate
- Most systems provide concepts of owner, group member
- Must have a way to apply these between systems

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
    - ✓Implements **vnodes** which hold inodes or network file details
  - Then dispatches operation to appropriate file system implementation routines

Virtual File Systems

- The API is to the VFS interface, rather than any specific type of file system



Schematic View of a Virtual File System

Virtual File Systems

Virtual File System Implementation

- For example, Linux has four object types:
  - **inode, file, superblock, dentry**

- 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

Remote File Systems

- Sharing of files across a network
- First method involved manually sharing each file – programs like ftp
- Second method uses a **distributed file system** (**DFS**)
  - Remote directories visible from local machine
- Third method – **World Wide Web**
  - A bit of a revision to first method
  - Use browser to locate file/files and download /upload
  - **Anonymous** access doesn't require authentication

Remote File Systems

Client-Server Model

- **Sharing** between a server (providing access to a file system via a network protocol) and a client (using the protocol to access the remote file system)
- **Identifying** each other via network ID can be spoofed, encryption can be performance expensive
- **NFS** an example
  - User **authentication** info on clients and servers must match (**example:** UserIDs)
  - Remote file system mounted, file operations sent on behalf of user across network to server
  - Server checks permissions, file handle returned
  - Handle used for reads and writes until file closed

Remote File Systems

Distributed Information Systems

- Also known as **distributed naming services**, provide unified access to info needed for remote computing
- **Domain name system** (**DNS**) provides host-name-to-network-address translations for the Internet
- Others like **network information service** (**NIS**) provide user-name, password, userID, group information
- Microsoft's **common Internet file system** (**CIFS**) network info used with user auth to create network logins that server uses to allow to deny access
  - **Active directory** distributed naming service
  - **Kerberos** - derived network authentication protocol
- Industry is moving toward **lightweight directory-access protocol** (**LDAP**) as secure distributed naming mechanism

Remote File Systems

Failure Modes

- **Local file systems** can **fail** for a **variety of reasons**
  - failure of the drive containing the file system
  - corruption of the directory structure or other disk-management information (collectively called **metadata**)
  - disk-controller failure
  - cable failure
  - host-adapter failure
  - user or system-administrator failure (causes files to be lost or entire directories or volumes to be deleted)

- **Remote file systems** have even more failure modes
  - Most DFS protocols either enforce or allow delaying of file-system operations to remote hosts
  - Some kind of **state information** may be maintained on both the client and the server
  - NFS Version 3 takes a simple approach, implementing a **stateless** DFS
  - Industry standard NFS Version 4 makes NFS is stateful to improve its security, performance, and functionality

Consistency Semantics

- **Important criteria for evaluating file sharing-file systems**

- **Specify** how multiple users are to access shared file simultaneously
  - When modifications of data will be observed by other users
  - Directly related to process synchronization algorithms, but atomicity across a network has high overhead (see Andrew File System)

- The **series of accesses between file open and closed** called **file session**

- **UNIX Semantics**
  - Writes to open file immediately visible to others with file open
  - One mode of sharing allows users to share pointer to current I/O location in file
  - Single physical image, accessed exclusively, contention causes process delays
- **Session Semantics** (Andrew file system (OpenAFS))
  - Writes to open file not visible during session, only at close
  - Can be several copies, each changed independently
- **Immutable-Shared-Files Semantics**
  - Once a file is declared as shared by its creator, it cannot be modified
  - An **immutable file has two key properties**: its **name** may not be reused, and its **contents** may not be altered
  - **Name of an immutable file** signifies that the c**ontents of the file are fixed**
  - Implementation of these semantics in a distributed system is simple, because the sharing is disciplined (read-only)

NFS (Network File System)

- An implementation and a specification of a software system for accessing remote files across LANs (or WANs)
- The implementation originally part of SunOS operating system, now industry standard / very common
- Can use unreliable datagram protocol (UDP/IP) or TCP/IP, over Ethernet or other network
- **Notes:**
  - Network File System is a distributed file system protocol
  - Originally developed by Sun Microsystems in 1984
  - Allows a user on a client computer to access files over a computer network much like local storage is accessed.

NFS (Network File System)

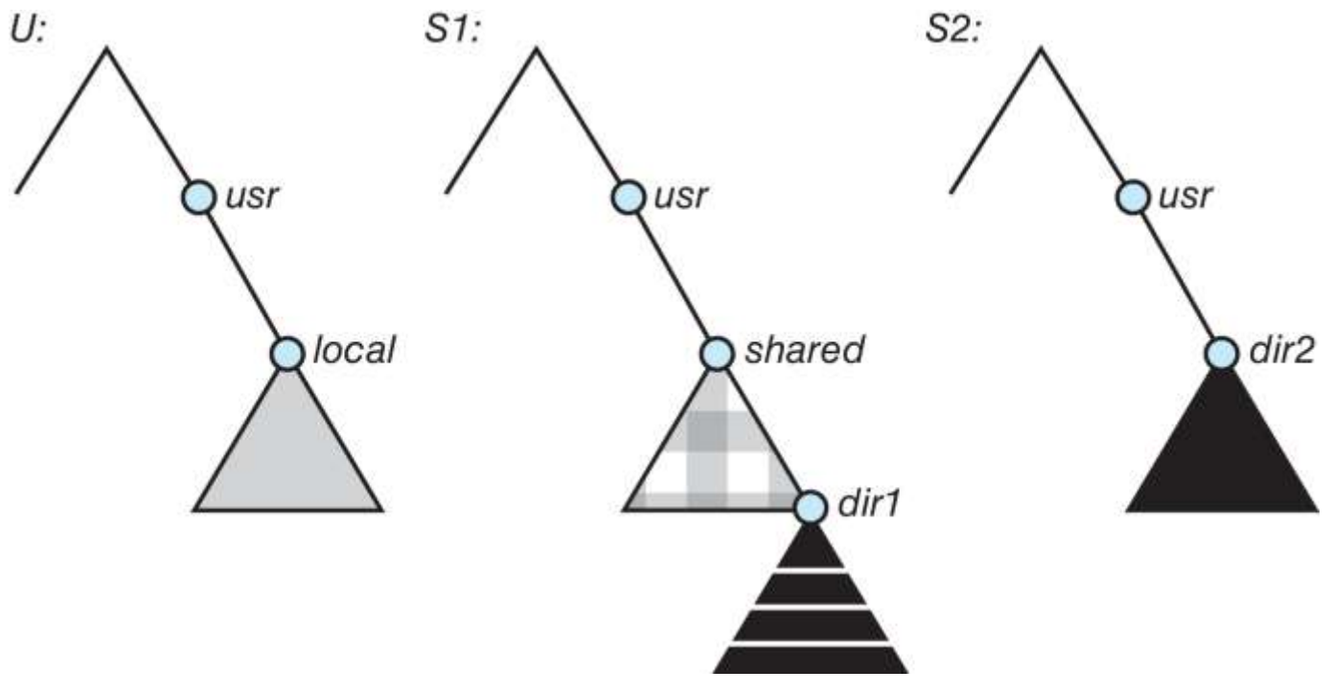
- Interconnected workstations viewed as a set of independent machines with independent file systems, which allows sharing among these file systems in a transparent manner
  - A **remote directory is mounted** over a local file system directory
    - ✓The mounted directory looks like an integral subtree of the local file system, replacing the subtree descending from the local directory
  - **Specification of the remote directory for the mount operation** is nontransparent; the host name of the remote directory has to be provided
    - ✓Files in the remote directory can then be accessed in a transparent manner
  - **Subject to access-rights accreditation**, potentially any file system (or directory within a file system), can be mounted remotely on top of any local directory

NFS (Network File System)

- NFS is designed to operate in a heterogeneous environment of different machines, operating systems, and network architectures; the NFS specifications independent of these media
- This independence is achieved through the use of RPC primitives built on top of an External Data Representation (XDR) protocol used between two implementation-independent interfaces
- The NFS specification distinguishes between the services provided by a mount mechanism and the actual remote-file-access services

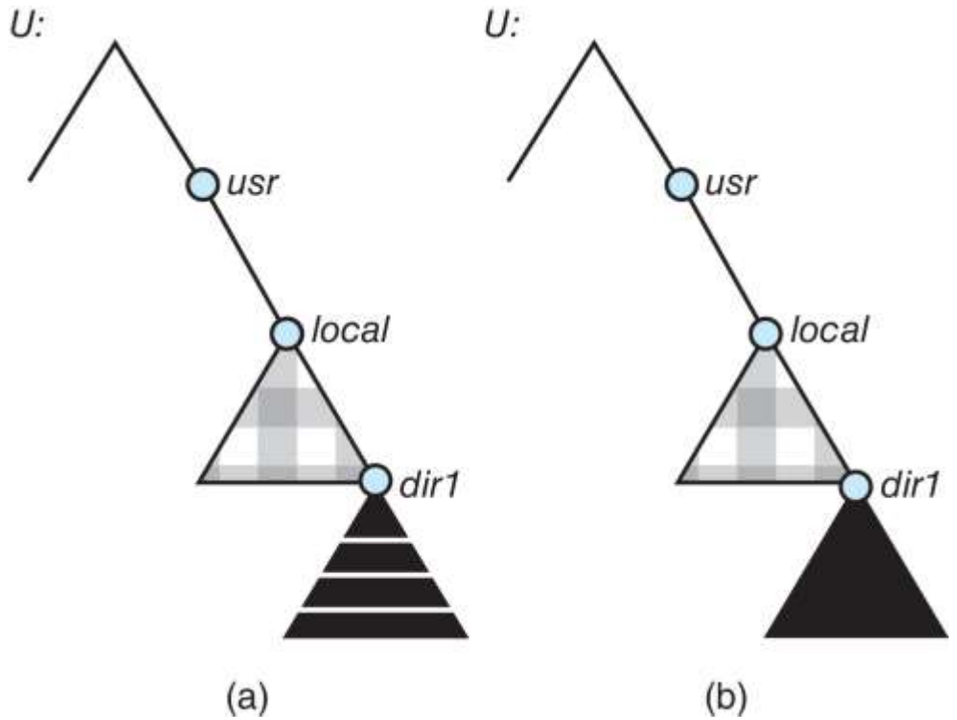
NFS (Network File System)

Three Independent File Systems



NFS (Network File System)

Mounting in NFS



(a) Mounts

(b) Cascading mounts

NFS (Network File System)

Mount Protocol

- Establishes initial logical connection between server and client
- Mount operation includes name of remote directory to be mounted and name of server machine storing it
  - Mount request is mapped to corresponding RPC and forwarded to mount server running on server machine
  - Export list – specifies local file systems that server exports for mounting, along with names of machines that are permitted to mount them
- Following a mount request that conforms to its export list, the server returns a file handle — a key for further accesses
- File handle – a file-system identifier, and an inode number to identify the mounted directory within the exported file system
- The mount operation changes only the user's view and does not affect the server side

NFS (Network File System)

NFS Protocol

- Provides a set of remote procedure calls for remote file operations.
  - The procedures support the following operations:
    - ✓searching for a file within a directory
    - ✓reading a set of directory entries
    - ✓manipulating links and directories
    - ✓accessing file attributes
    - ✓reading and writing files

- NFS servers are **stateless**; each request has to provide a full set of arguments (NFS V4 is newer, less used – very different, stateful)

- Modified data must be committed to the server's disk before results are returned to the client (lose advantages of caching)

- The NFS protocol does not provide concurrency-control mechanisms

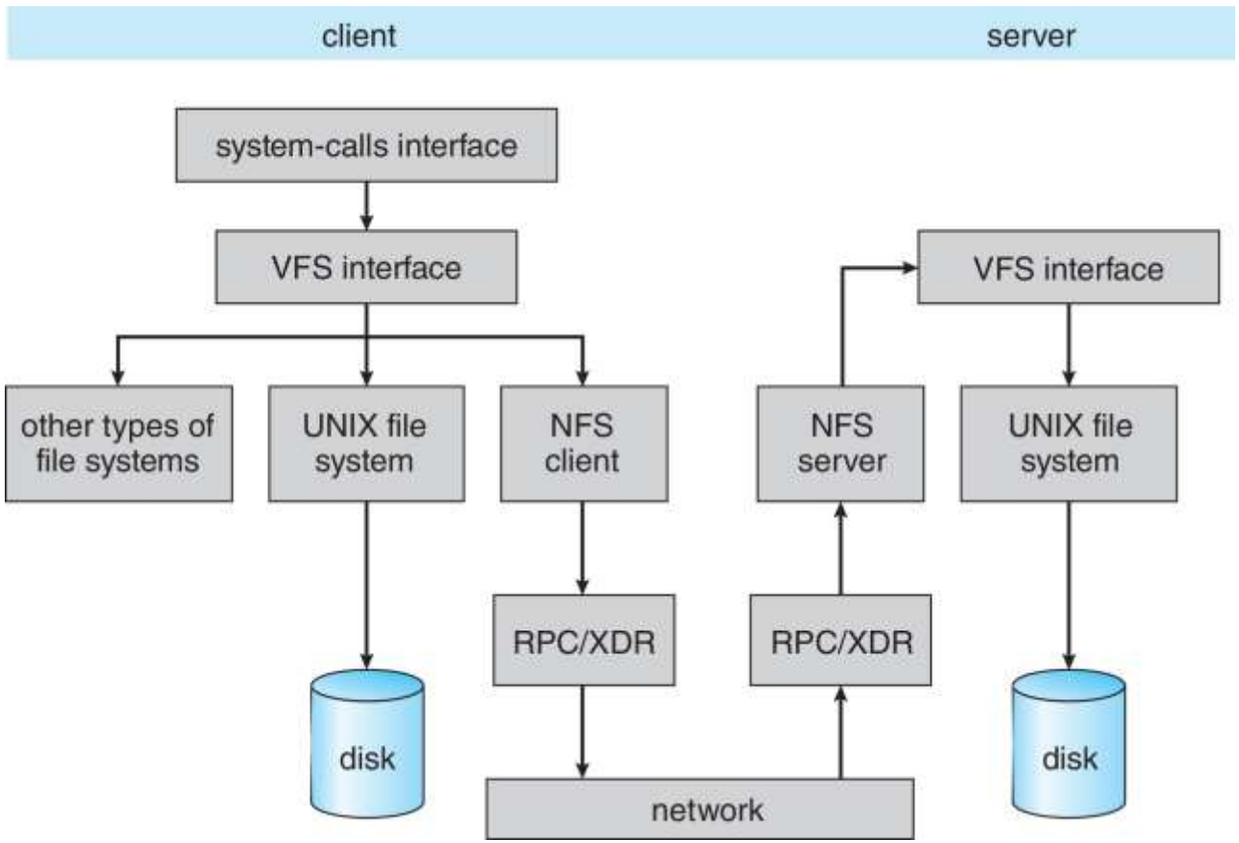
NFS (Network File System)

Three Major Layers of NFS Architecture

- **UNIX file-system interface** (based on the **open, read, write**, and **close** calls, and **file descriptors**)
- **Virtual File System (VFS) layer** – distinguishes local files from remote ones, and local files are further distinguished according to their file-system types
  - The VFS activates file-system-specific operations to handle local requests according to their file-system types
  - Calls the NFS protocol procedures for remote requests
- **NFS service layer** – bottom layer of the architecture
  - Implements the NFS protocol

NFS (Network File System)

Schematic View of the NFS Architecture



NFS (Network File System)

Path-Name Translation

- Performed by breaking the path into component names and performing a separate NFS lookup call for every pair of component name and directory **vnode**
- To make lookup faster, a directory name lookup cache on the client's side holds the vnodes for remote directory names

NFS (Network File System)

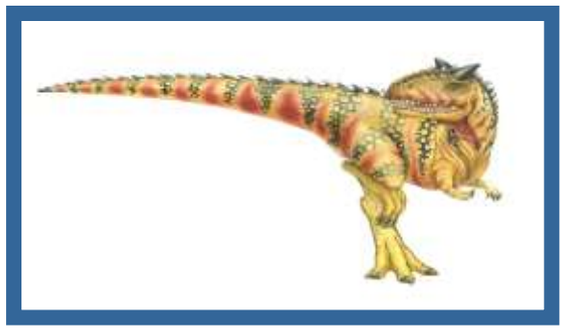
Remote Operations

- Nearly one-to-one correspondence between regular UNIX system calls and the NFS protocol RPCs (except opening and closing files)
- NFS adheres to the remote-service paradigm, but employs buffering and caching techniques for the sake of performance
- File-blocks cache – when a file is opened, the kernel checks with the remote server whether to fetch or revalidate the cached attributes
  - Cached file blocks are used only if the corresponding cached attributes are up to date
- File-attribute cache – the attribute cache is updated whenever new attributes arrive from the server
- Clients do not free delayed-write blocks until the server confirms that the data have been written to disk

CS 3104 – OPERATING SYSTEMS

DCISM
DEPARTMENT OF COMPUTER, INFORMATION SCIENCES AND MATHEMATICS

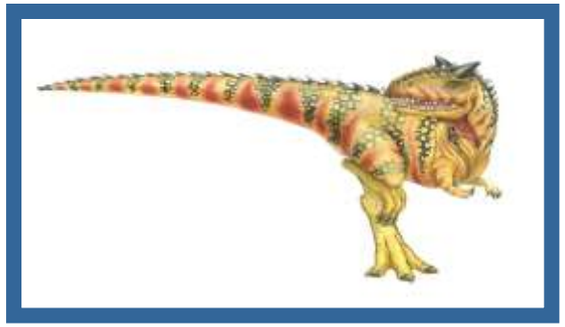
End of Chapter 15
File-System Internals



CS 3104 – OPERATING SYSTEMS

DCISM
DEPARTMENT OF COMPUTER, INFORMATION SCIENCES AND MATHEMATICS

Chapter 16: Security



Security

- The Security Problem
- Program Threats
- System and Network Threats
- Cryptography as a Security Tool
- User Authentication
- Implementing Security Defenses
- An Example: Windows 10

The Security Problem

- System is **secure** if its resources are used and accessed as intended under all circumstances
  - Unachievable
- **Intruders** (**crackers**) attempt to breach security
- **Threat** is potential security violation
- **Attack** is attempt to breach security
- **Attack** can be **accidental or malicious**
- Easier to protect against accidental than malicious misuse

The Security Problem

Security Violation Categories

- **Breach of confidentiality**
  - Unauthorized reading of data
- **Breach of integrity**
  - Unauthorized modification of data
- **Breach of availability**
  - Unauthorized destruction of data
- **Theft of service**
  - Unauthorized use of resources
- **Denial of service (DOS)**
  - Prevention of legitimate use of the system

The Security Problem

Security Violation Methods

- **Masquerading** (breach **authentication**)
  - Pretending to be an authorized user to escalate privileges
- **Replay attack**
  - As is **or** with **message modification**
- **Man-in-the-middle attack**
  - Intruder sits in data flow, masquerading as sender to receiver and vice versa
- **Session hijacking**
  - Intercept an already-established session to bypass authentication
- **Privilege escalation**
  - Common attack type with access beyond what a user or resource is supposed to have

The Security Problem

Security Measure Levels

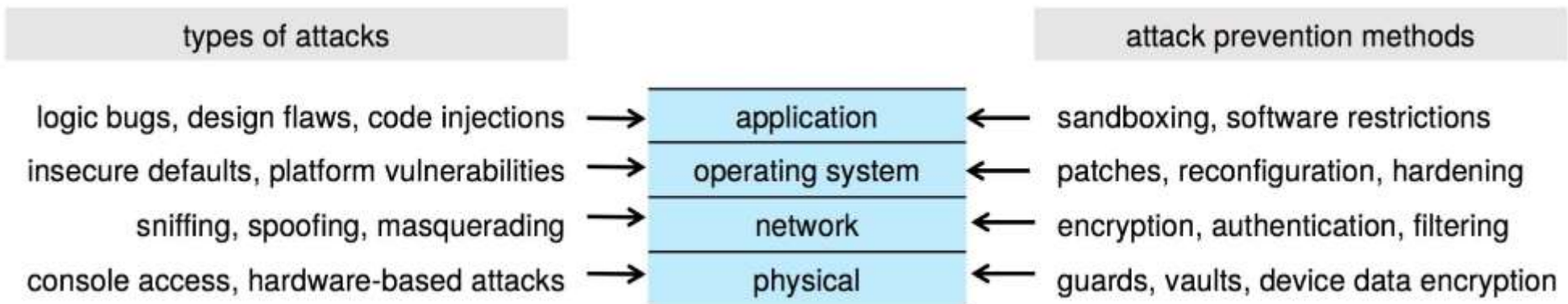
- Impossible to have absolute security, but make cost to perpetrator sufficiently high to deter most intruders

- **Security** must occur **at four levels** to be effective:
  - **Physical**
    - Data centers, servers, connected terminals
  - **Application**
    - Benign or malicious apps can cause security problems
  - **Operating System**
    - Protection mechanisms, debugging
  - **Network**
    - Intercepted communications, interruption, DOS

- Security is as weak as the weakest link in the chain
- Humans are risks, too (via **phishing** and **social-engineering** attacks)
- But can too much security be a problem?

The Security Problem

Four-layered Model of Security



Program Threats

- Many variations, many names
- **Trojan Horse**
  - Code segment that misuses its environment
  - Exploits mechanisms for allowing programs written by users to be executed by other users
- **Spyware, pop-up browser windows, covert channels**
  - Up to 80% of spam delivered by spyware-infected systems
- **Trap Door**
  - Specific user identifier or password that circumvents normal security procedures
  - Could be included in a compiler
  - How to detect them?

Program Threats

- **Malware -** Software designed to exploit, disable, or damage computer
- **Trojan Horse** – Program that acts in a clandestine manner
- **Spyware** – Program frequently installed with legitimate software to display adds, capture user data
- **Ransomware** – locks up data via encryption, demanding payment to unlock it
- Others include trap doors, logic bombs
- All try to violate the Principle of Least Privilege:

> *THE PRINCIPLE OF LEAST PRIVILEGE*
>
> "The principle of least privilege. Every program and every privileged user of the system should operate using the least amount of privilege necessary to complete the job. The purpose of this principle is to reduce the number of potential interactions among privileged programs to the minimum necessary to operate correctly, so that one may develop confidence that unintentional, unwanted, or improper uses of privilege do not occur."—Jerome H. Saltzer, describing a design principle of the Multics operating system in 1974: https://pdfs.semanticscholar.org/1c8d/06510ad449ad24fbdd164f8008cc730cab47.pdf.

- Goal frequently is to leave behind Remote Access Tool (RAT) for repeated access

Program Threats

Code Injection

- **Code-injection attack** occurs when system code is not malicious but has bugs allowing executable code to be added or modified
  - Results from poor or insecure programming paradigms, commonly in low level languages like C or C++ which allow for direct memory access through pointers
  - Goal is a buffer overflow in which code is placed in a buffer and execution caused by the attack
  - Can be run by script kiddies – use tools written but exploit identifiers

Program Threats

C Program with Buffer-overflow Condition

```
#include <stdio.h>
#define BUFFER_SIZE 256
int main(int argc, char *argv[])
{
          char buffer[BUFFER_SIZE];
          if (argc < 2)
                    return -1;
          else {
                    strcpy(buffer,argv[1]);
                    return 0;
          }
}
```

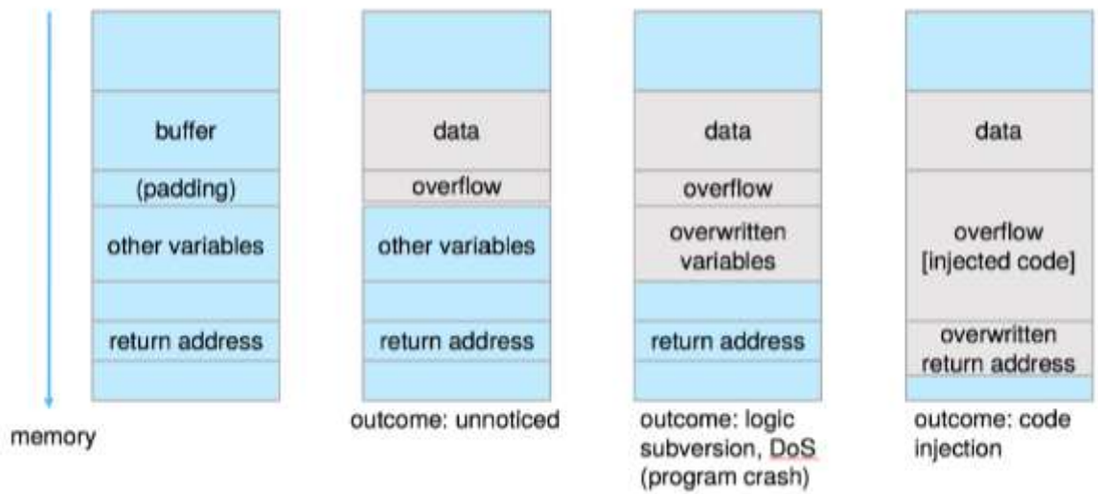
```
#include<stdio.h>
#include<string.h>
int main() {
    bufferOverflow();
 }
  bufferOverflow() {
    char textLine[10];
    printf("Enter your line of text: ");
    gets(textLine);
    printf("You entered: %s", textLine);
    return 0;
 }
```

Code review can help (programmers review each other's code, looking for logic flows, programming flaws)

Program Threats

Code Injection

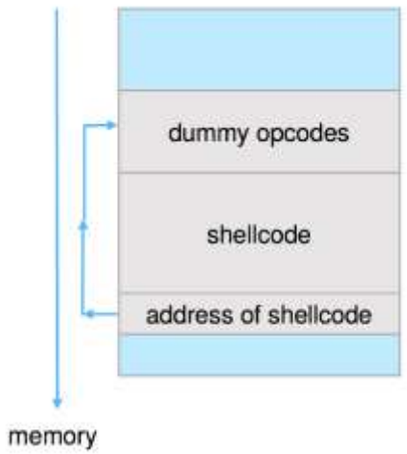
- Outcomes from code injection include:



Program Threats

Code Injection

- Frequently use trampoline to code execution to exploit buffer overflow:



Program Threats

Great Programming Required?

- For the first step of determining the bug, and second step of writing exploit code, yes
- **Script kiddies** can run pre-written exploit code to attack a given system
- Attack code can get a shell with the processes' owner's permissions
  - Or open a network port, delete files, download a program, etc
- Depending on bug, attack can be executed across a network using allowed connections, bypassing firewalls
- Buffer overflow can be disabled by disabling stack execution or adding bit to page table to indicate "non-executable" state
  - Available in SPARC and x86
  - But still have security exploits

Program Threats

- **Viruses**
  - Code fragment embedded in legitimate program
  - Self-replicating, designed to infect other computers
  - Very specific to CPU architecture, operating system, applications
  - Usually borne via email or as a macro
  - **Visual Basic Macro to reformat hard drive:**

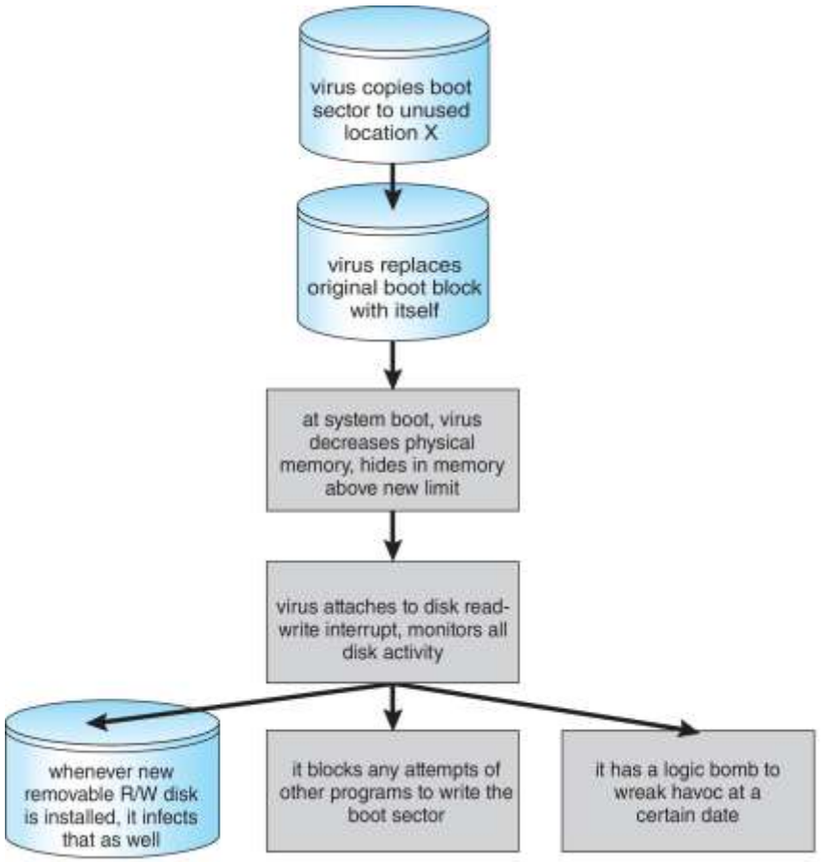
```
Sub AutoOpen()
Dim oFS
    Set oFS = CreateObject(”Scripting.FileSystemObject”)
    vs = Shell(”c:command.com /k format c:”,vbHide)
End Sub
```


Program Threats

- **Virus dropper** inserts virus onto the system
- Many **categories of viruses**, literally many thousands of viruses
  - File / parasitic
  - Boot / memory
  - Macro
  - Source code
  - Polymorphic to avoid having a **virus signature**
  - Encrypted
  - Stealth
  - Tunneling
  - Multipartite
  - Armored

Program Threats

A Boot-sector Computer Virus



Program Threats

The Threat Continues

- Attacks still common, still occurring
- Attacks moved over time from science experiments to tools of organized crime
  - Targeting specific companies
  - Creating botnets to use as tool for spam and DDOS delivery
  - **Keystroke logger** to grab passwords, credit card numbers

- Why is Windows the target for most attacks?
  - Most common
  - Everyone is an administrator
    - Licensing required?
  - **Monoculture** considered harmful

System and Network Threats

- Some systems **"open"** rather than **secure by default**
  - Reduce **attack surface**
  - But harder to use, more knowledge needed to administer
- **Network threats harder to detect, prevent**
  - Protection systems weaker
  - More difficult to have a shared secret on which to base access
  - No physical limits once system attached to internet
    - Or on network with system attached to internet
  - Even determining location of connecting system difficult
    - IP address is only knowledge

System and Network Threats

- **Worms** – use **spawn** mechanism; **standalone program**
- **Internet worm**
  - Exploited UNIX networking features (remote access) and bugs in *finger* and ***sendmail*** programs
  - Exploited trust-relationship mechanism used by ***rsh*** to access friendly systems without use of password
  - **Grappling hook** program uploaded main worm program
    - 99 lines of C code
  - Hooked system then uploaded main code, tried to attack connected systems
  - Also tried to break into other users accounts on local system via password guessing
  - If target system already infected, abort, except for every 7^{th} time

System and Network Threats

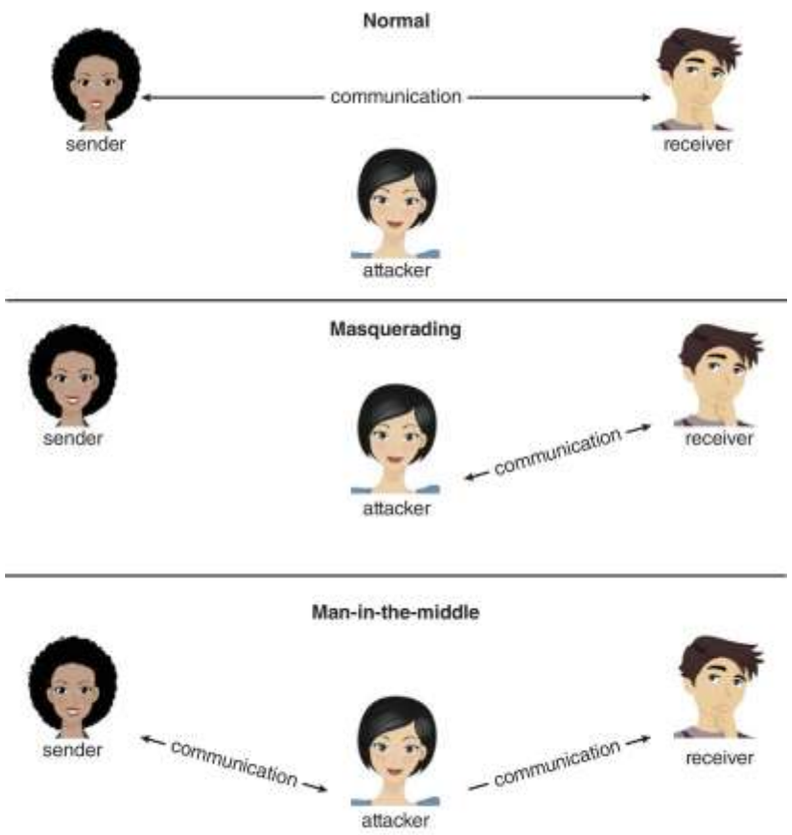
- **Port scanning**
  - Automated attempt to connect to a range of ports on one or a range of IP addresses
  - Automated tool to look for network ports accepting connections
  - Detection of answering service protocol
  - Detection of OS and version running on system
  - **nmap** scans all ports in a given IP range for a response
  - **nessus** has a database of protocols and bugs (and exploits) to apply against a system
  - Frequently launched from **zombie systems**
  - To decrease trace-ability
  - Used for good and evil

System and Network Threats

- **Denial of Service**
  - Overload the targeted computer preventing it from doing any useful work
  - **Distributed Denial-of-Service** (**DDoS**) come from multiple sites at once
    - Consider the start of the IP-connection handshake (SYN)
  - How many started-connections can the OS handle?
    - Consider traffic to a web site
  - How can you tell the difference between being a target and being really popular?
  - Accidental – CS students writing bad **fork()** code
  - Purposeful – extortion, punishment

System and Network Threats

Standard Security Attacks



Cryptography as a Security Tool

- Broadest security tool available
  - Internal to a given computer, source and destination of messages can be known and protected
    - OS creates, manages, protects process IDs, communication ports
  - Source and destination of messages on network cannot be trusted without cryptography
    - Local network – IP address?
      - ✓Consider unauthorized host added
    - WAN / Internet – how to establish authenticity
      - ✓Not via IP address

Cryptography as a Security Tool

Cryptography

- **Means to constrain** potential senders (*sources*) and / or receivers (*destinations*) of *messages*
  - Based on secrets (**keys**)
  - Enables
    - Confirmation of source
    - Receipt only by certain destination
    - Trust relationship between sender and receiver

Cryptography as a Security Tool

Encryption

- Constrains the set of possible receivers of a message
- **Encryption** algorithm consists of
  - Set K of keys
  - Set M of messages
  - Set C of ciphertexts (encrypted messages)
  - A function $E : K \rightarrow (M \rightarrow C)$
    - That is, for each $k \in K$, E_k is a function for generating ciphertexts from messages
    - Both E and E_k for any k should be efficiently computable functions
  - A function $D : K \rightarrow (C \rightarrow M)$.
    - That is, for each $k \in K$, D_k is a function for generating messages from ciphertexts
    - Both D and D_k for any k should be efficiently computable functions

Cryptography as a Security Tool

Encryption

- **An encryption algorithm must provide this essential property:**
  Given a ciphertext $\mathbf{c} \in \mathbf{C}$, a computer can compute **m** such that $\mathbf{E_k(m) = c}$ only if it possesses ***k***
  - Thus, a computer holding *k* can decrypt ciphertexts to the plaintexts used to produce them, but a computer not holding *k* cannot decrypt ciphertexts
  - Since ciphertexts are generally exposed (for example, sent on the network), it is important that it be infeasible to derive *k* from the ciphertexts

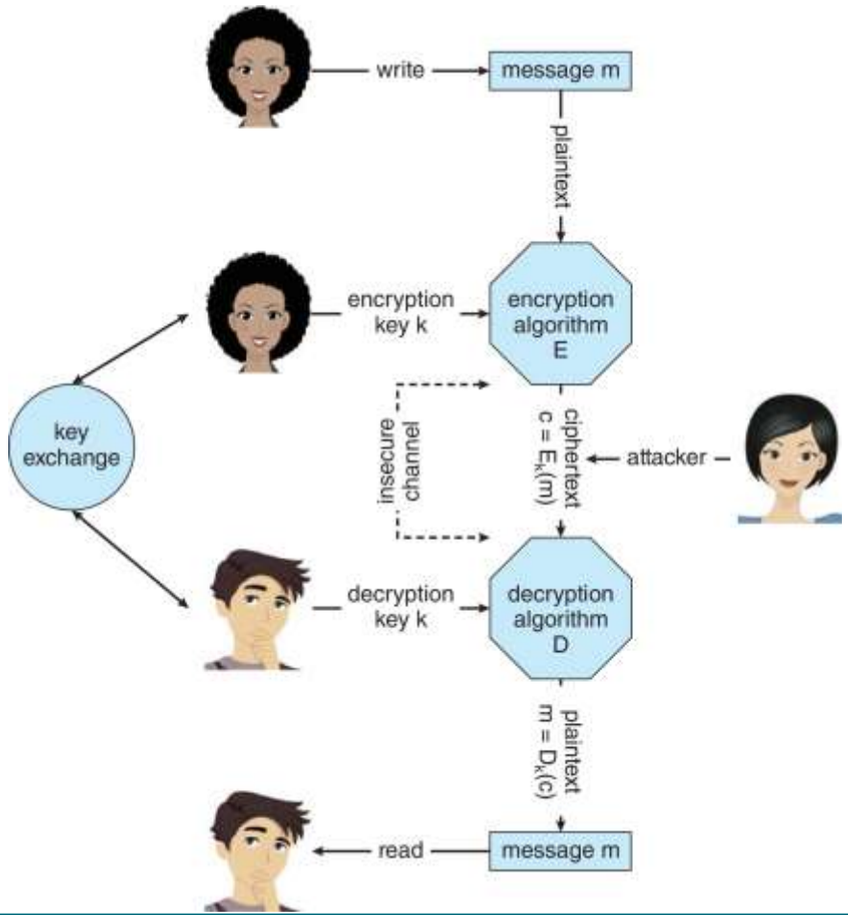
Cryptography as a Security Tool

Symmetric Encryption

- Same key used to encrypt and decrypt
  - Therefore *k* must be kept secret
- DES was most commonly used symmetric block-encryption algorithm (created by US Gov't.)
  - Encrypts a block of data at a time
  - Keys too short so now considered insecure
- Triple-DES considered more secure
  - Algorithm used 3 times using 2 or 3 keys
  - For example
- 2001 NIST adopted new block cipher - Advanced Encryption Standard (**AES**)
  - Keys of 128, 192, or 256 bits, works on 128 bit blocks
- RC4 is most common symmetric stream cipher, but known to have vulnerabilities
  - Encrypts/decrypts a stream of bytes (i.e., wireless transmission)
  - Key is a input to pseudo-random-bit generator
    - Generates an infinite **keystream**

Cryptography as a Security Tool

Secure Communication over Insecure Medium



Cryptography as a Security Tool

Asymmetric Encryption

- **Public-key encryption** based on each user having two keys:
  - **public key** – published key used to encrypt data
  - **private key** – key known only to individual user used to decrypt data

- Must be an **encryption scheme that can be made public** without making it easy to figure out the decryption scheme
  - Most common is **RSA** (Rivest, Shamir, and Adleman) block cipher
  - Efficient algorithm for testing whether or not a number is prime
  - No efficient algorithm is know for finding the prime factors of a number

Cryptography as a Security Tool

Asymmetric Encryption

- Formally, it is computationally infeasible to derive $k_{d,N}$ from $k_{e,N}$, and so k_e need not be kept secret and can be widely disseminated
  - k_e is the **public key**
  - k_d is the **private key**
  - N is the product of two large, randomly chosen prime numbers p and q (for example, p and q are 512 bits each)
  - Encryption algorithm is $E_{ke,N}(m) = m^{ke} \bmod N$, where k_e satisfies $k_e k_d \bmod (p-1)(q-1) = 1$
  - The decryption algorithm is then $D_{kd,N}(c) = c^{kd} \bmod N$

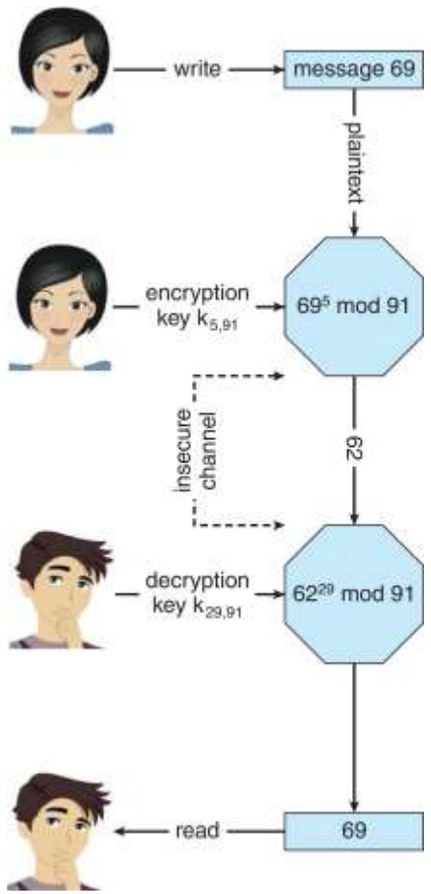
Cryptography as a Security Tool

Asymmetric Encryption Example

- For example, make $p = 7$ and $q = 13$
- We then calculate $N = 7*13 = 91$ and $(p-1)(q-1) = 72$
- We next select k_e relatively prime to 72 and < 72, yielding 5
- Finally, we calculate k_d such that $k_e k_d \bmod 72 = 1$, yielding 29
- We how have our keys
  - Public key, $k_{e,N} = 5, 91$
  - Private key, $k_{d,N} = 29, 91$
- Encrypting the message 69 with the public key results in the cyphertext 62
- Cyphertext can be decoded with the private key
  - Public key can be distributed in cleartext to anyone who wants to communicate with holder of public key

Cryptography as a Security Tool

Encryption using RSA Asymmetric Cryptography



Cryptography as a Security Tool

Cryptography

- **Important Notes:**
  - Symmetric cryptography is based on transformations
  - Asymmetric cryptography is based on mathematical functions
  - Asymmetric cryptography is much more compute intensive
    - Typically not used for bulk data encryption

Cryptography as a Security Tool

Authentication

- **Constraining set of potential senders of a message**
  - Complementary to encryption
  - Also can prove message unmodified

- **Algorithm components**
  - A set K of keys
  - A set M of messages
  - A set A of authenticators
  - A function $S : K \rightarrow (M \rightarrow A)$
    - That is, for each $k \in K$, S_k is a function for generating authenticators from messages
    - Both S and S_k for any k should be efficiently computable functions
  - A function $V : K \rightarrow (M \times A \rightarrow \{\text{true, false}\})$.
    - That is, for each $k \in K$, V_k is a function for verifying authenticators on messages
    - Both V and V_k for any k should be efficiently computable functions

Cryptography as a Security Tool

Authentication

- For a message ***m***, a computer can generate an authenticator ***a*** $\in$ ***A*** such that $V_k(m, a)$ = true only if it possesses ***k***
- Thus, computer holding ***k*** can generate authenticators on messages so that any other computer possessing ***k*** can verify them
- Computer not holding ***k*** cannot generate authenticators on messages that can be verified using V_k
- Since authenticators are generally exposed (for example, they are sent on the network with the messages themselves), it must not be feasible to derive ***k*** from the authenticators
- Practically, if $V_k(m,a)$ = **true** then we know ***m*** has not been modified and that send of message has ***k***
  - If we share ***k*** with only one entity, know where the message originated

Cryptography as a Security Tool

Authentication – Hash Functions

- Basis of authentication
- Creates small, fixed-size block of data **message digest (hash value)** from ***m***
- Hash Function ***H*** must be collision resistant on ***m***
  - Must be infeasible to find an ***m'*** ≠ ***m*** such that ***H*(*m*) = *H*(*m'*)**
- If ***H*(*m*) = *H*(*m'*)**, then ***m* = *m'***
  - The message has not been modified
- Common message-digest functions include **MD5**, which produces a 128-bit hash, and **SHA-1**, which outputs a 160-bit hash
- Not useful as authenticators
  - For example, ***H(m)*** can be sent with a message
    - But if ***H*** is known, someone could modify ***m*** to ***m'*** and recompute ***H(m')***, and modification not detected
    - So must authenticate ***H(m)***

Cryptography as a Security Tool

Authentication - MAC

- Symmetric encryption used in **message-authentication code** (**MAC**) authentication algorithm
- Cryptographic checksum generated from message using secret key
  - Can securely authenticate short values
- If used to authenticate *H(m)* for an *H* that is collision resistant, then obtain a way to securely authenticate long message by hashing them first
- Note that *k* is needed to compute both S_k and V_k, so anyone able to compute one can compute the other

Cryptography as a Security Tool

Authentication – Digital Signature

- Based on asymmetric keys and digital signature algorithm
- Authenticators produced are **digital signatures**
- Very useful – ***anyone*** can verify authenticity of a message
- In a digital-signature algorithm, computationally infeasible to derive k_s from k_v
  - V is a one-way function
  - Thus, k_v is the public key and k_s is the private key
- Consider the RSA digital-signature algorithm
  - Similar to the RSA encryption algorithm, but the key use is reversed
  - Digital signature of message $S_{ks}(m) = H(m)^{ks} \bmod N$
  - The key k_s again is a pair (d, N), where N is the product of two large, randomly chosen prime numbers p and q
  - Verification algorithm is $V_{kv}(m, a)$ $(a^{kv} \bmod N = H(m))$
    - Where k_v satisfies $k_v k_s \bmod (p-1)(q-1) = 1$

Cryptography as a Security Tool

Authentication

- Why authentication if it is a subset of encryption?
  - Fewer computations (except for RSA digital signatures)
  - Authenticator usually shorter than message
  - Sometimes want authentication but not confidentiality
    - Signed patches et al
  - Can be basis for **non-repudiation**

Cryptography as a Security Tool

Key Distribution

- Delivery of symmetric key is huge challenge
- Sometimes done **out-of-band**
- Asymmetric keys can proliferate – stored on **key ring**
- Even asymmetric key distribution needs care – man-in-the-middle attack

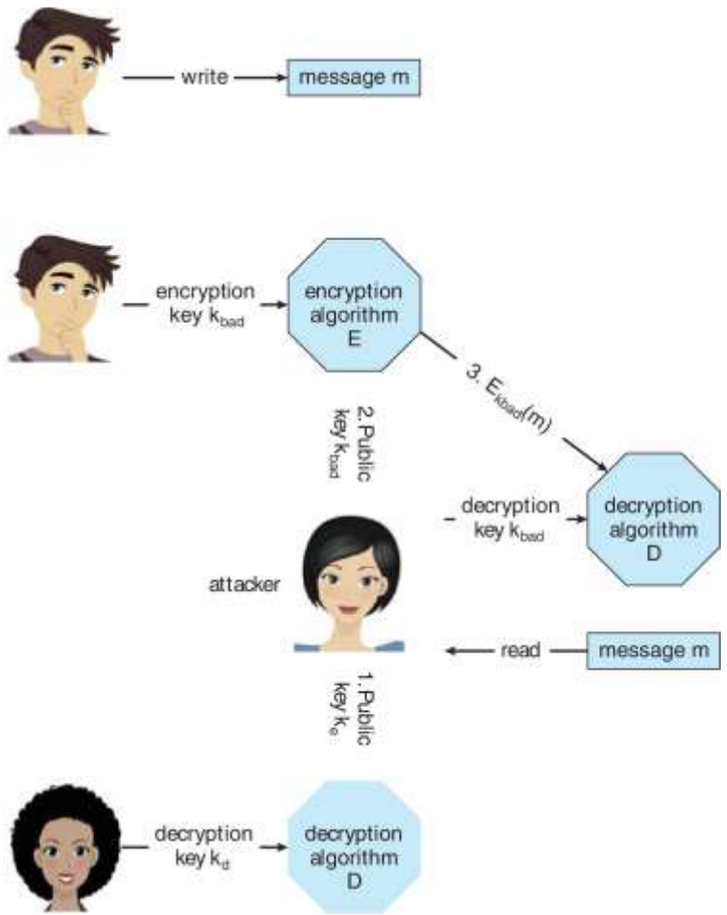
Cryptography as a Security Tool

Digital Certificates

- Proof of who or what owns a public key
- Public key digitally signed a trusted party
- Trusted party receives proof of identification from entity and certifies that public key belongs to entity
- **Certificate authority** are trusted party – their public keys included with web browser distributions
- They vouch for other authorities via digitally signing their keys, and so on

Cryptography as a Security Tool

Man-in-the-middle Attack on Asymmetric Cryptography

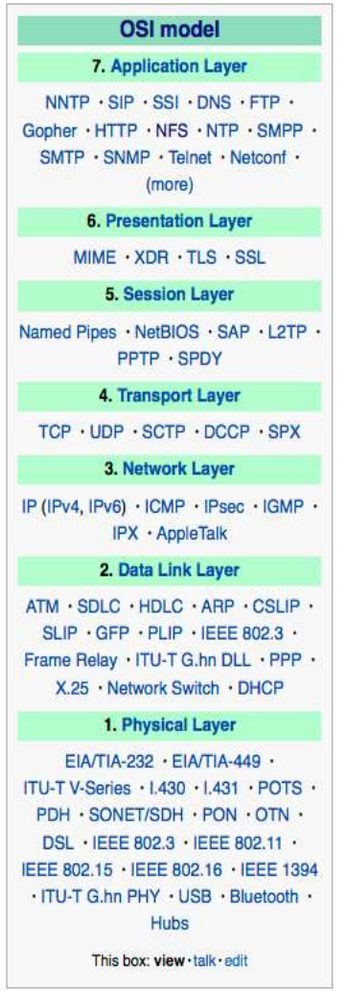


Cryptography as a Security Tool

Implementation of Cryptography

- **Can be done at various layers of ISO Reference Model**
  - SSL at the Transport layer
  - Network layer is typically **IPSec**
    - **IKE** for key exchange
    - Basis of **Virtual Private Networks (VPNs)**

- **Why not just at lowest level?**
  - Sometimes need more knowledge than available at low levels
    - i.e., User authentication
    - i.e., e-mail delivery



OSI model
7. Application Layer
NNTP · SIP · SSI · DNS · FTP · Gopher · HTTP · NFS · NTP · SMPP · SMTP · SNMP · Telnet · Netconf · (more)
6. Presentation Layer
MIME · XDR · TLS · SSL
5. Session Layer
Named Pipes · NetBIOS · SAP · L2TP · PPTP · SPDY
4. Transport Layer
TCP · UDP · SCTP · DCCP · SPX
3. Network Layer
IP (IPv4, IPv6) · ICMP · IPsec · IGMP · IPX · AppleTalk
2. Data Link Layer
ATM · SDLC · HDLC · ARP · CSLIP · SLIP · GFP · PLIP · IEEE 802.3 · Frame Relay · ITU-T G.hn DLL · PPP · X.25 · Network Switch · DHCP
1. Physical Layer
EIA/TIA-232 · EIA/TIA-449 · ITU-T V-Series · I.430 · I.431 · POTS · PDH · SONET/SDH · PON · OTN · DSL · IEEE 802.3 · IEEE 802.11 · IEEE 802.15 · IEEE 802.16 · IEEE 1394 · ITU-T G.hn PHY · USB · Bluetooth · Hubs
This box: view · talk · edit

OSI Model			
	Data unit	Layer	Function
Host layers	Data	7. Application	Network process to application
		6. Presentation	Data representation, encryption and decryption, convert machine dependent data to machine independent data
		5. Session	Interhost communication
	Segments	4. Transport	End-to-end connections and reliability, flow control
Media layers	Packet/Datagram	3. Network	Path determination and logical addressing
	Frame	2. Data Link	Physical addressing
	Bit	1. Physical	Media, signal and binary transmission

Cryptography as a Security Tool

Encryption Example - TLS

- Insertion of cryptography at one layer of the ISO network model (the transport layer)
- SSL – Secure Socket Layer (also called TLS)
- Cryptographic protocol that limits two computers to only exchange messages with each other
  - Very complicated, with many variations
- Used between web servers and browsers for secure communication (credit card numbers)
- The server is verified with a **certificate** assuring client is talking to correct server
- Asymmetric cryptography used to establish a secure **session key** (symmetric encryption) for bulk of communication during session
- Communication between each computer then uses symmetric key cryptography
- More details in textbook

User Authentication

- Crucial to identify user correctly, as protection systems depend on user ID
- User identity most often established through **passwords**, can be considered a special case of either keys or capabilities
- Passwords must be kept secret
  - Frequent change of passwords
  - History to avoid repeats
  - Use of “non-guessable” passwords
  - Log all invalid access attempts (but not the passwords themselves)
  - Unauthorized transfer
- Passwords may also either be encrypted or allowed to be used only once
  - Does encrypting passwords solve the exposure problem?
    - Might solve **sniffing**
    - Consider **shoulder surfing**
    - Consider Trojan horse keystroke logger
    - How are passwords stored at authenticating site?

User Authentication

Passwords

- **Encrypt to avoid having to keep secret**
  - But keep secret anyway (i.e. Unix uses superuser-only readably file /etc/shadow)
  - Use algorithm easy to compute but difficult to invert
  - Only encrypted password stored, never decrypted
  - Add “salt” to avoid the same password being encrypted to the same value
- **One-time passwords**
  - Use a function based on a seed to compute a password, both user and computer
  - Hardware device / calculator / key fob to generate the password
    - Changes very frequently
- **Biometrics**
  - Some physical attribute (fingerprint, hand scan)
- **Multi-factor authentication**
  - Need two or more factors for authentication
    - i.e., USB “dongle”, biometric measure, and password

User Authentication

Passwords

STRONG AND EASY TO REMEMBER PASSWORDS

It is extremely important to use strong (hard to guess and hard to shoulder surf) passwords on critical systems like bank accounts. It is also important to not use the same password on lots of systems, as one less important, easily hacked system could reveal the password you use on more important systems. A good technique is to generate your password by using the first letter of each word of an easily remembered phrase using both upper and lower characters with a number or punctuation mark thrown in for good measure. For example, the phrase "My girlfriend's name is Katherine" might yield the password "Mgn.isK!". The password is hard to crack but easy for the user to remember. A more secure system would allow more characters in its passwords. Indeed, a system might also allow passwords to include the space character, so that a user could create a passphrase which is easy to remember but difficult to break.

Implementing Security Defenses

- **Defense in depth** is most common security theory – multiple layers of security
- **Security policy** describes what is being secured
- Vulnerability assessment compares real state of system / network compared to security policy
- Intrusion detection endeavors to detect attempted or successful intrusions
  - **Signature-based** detection spots known bad patterns
  - **Anomaly detection** spots differences from normal behavior
    - Can detect **zero-day** attacks
  - **False-positives** and **false-negatives** a problem
- Virus protection
  - Searching all programs or programs at execution for known virus patterns
  - Or run in **sandbox** so can't damage system
- Auditing, accounting, and logging of all or specific system or network activities
- Practice **safe computing** – avoid sources of infection, download from only "good" sites, etc.

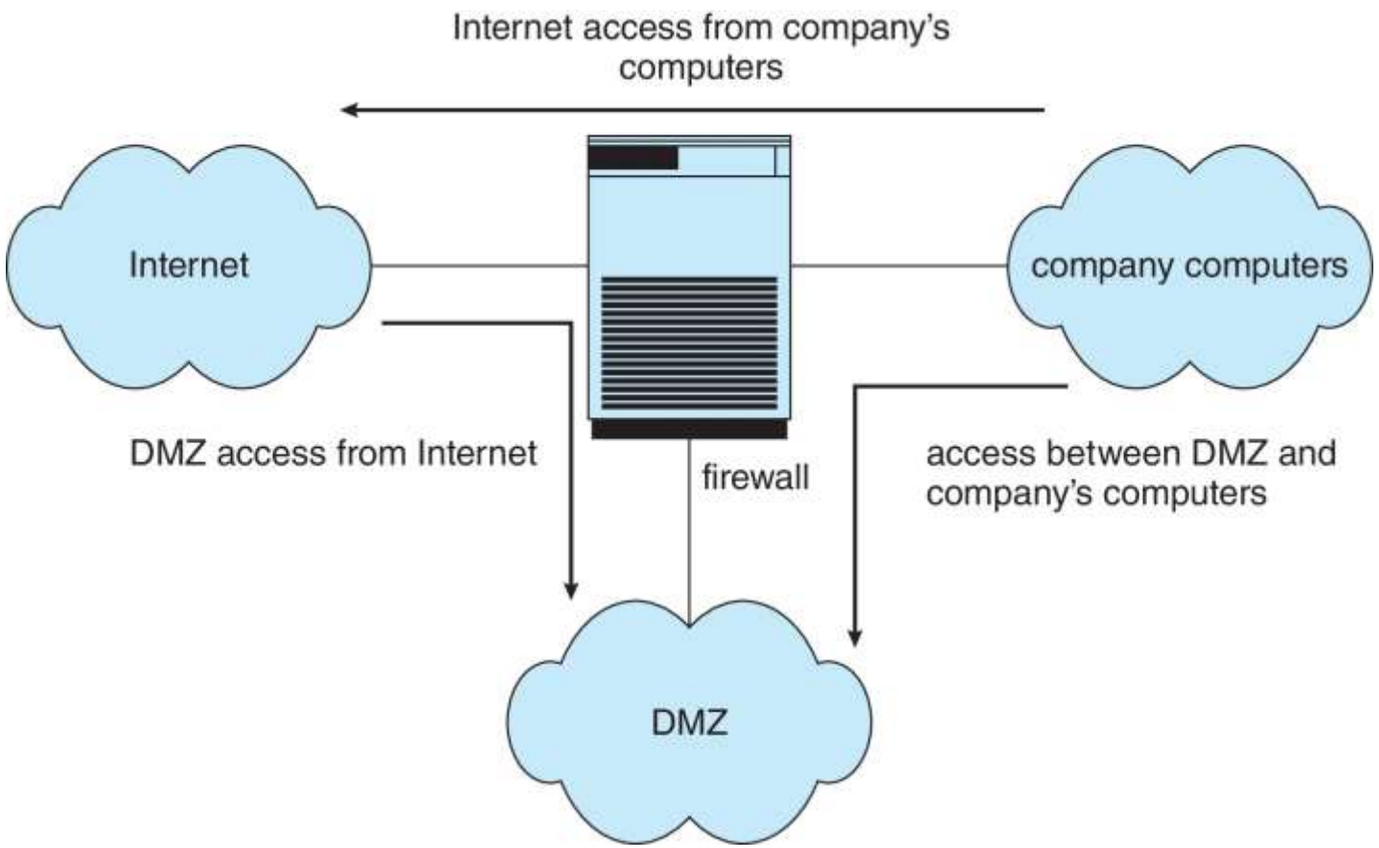
Implementing Security Defenses

Firewalling to Protect Systems and Networks

- A network **firewall** is placed between trusted and untrusted hosts
  - The firewall limits network access between these two **security domains**
- Can be tunneled or spoofed
  - Tunneling allows disallowed protocol to travel within allowed protocol (i.e., telnet inside of HTTP)
  - Firewall rules typically based on host name or IP address which can be spoofed
- **Personal firewall** is software layer on given host
  - Can monitor / limit traffic to and from the host
- **Application proxy firewall** understands application protocol and can control them (i.e., SMTP)
- **System-call firewall** monitors all important system calls and apply rules to them (i.e., this program can execute that system call)

Implementing Security Defenses

Network Security Through Domain Separation Via Firewall



Implementing Security Defenses

Computer Security Classifications

- U.S. Department of Defense outlines four divisions of computer security: **A**, **B**, **C**, and **D**
- **D** – Minimal security
- **C** – Provides discretionary protection through auditing
  - Divided into **C1** and **C2**
    - **C1** identifies cooperating users with the same level of protection
    - **C2** allows user-level access control
- **B** – All the properties of **C**, however each object may have unique sensitivity labels
  - Divided into **B1**, **B2**, and **B3**
- **A** – Uses formal design and verification techniques to ensure security

Implementing Security Defenses

Security Defenses Summarized

- By applying appropriate layers of defense, we can keep systems safe from all but the most persistent attackers.
- In summary, these layers may include the following:
  - Educate users about safe computing — don't attach devices of unknown origin to the computer, don't share passwords, use strong passwords, avoid falling for social engineering appeals, realize that an e-mail is not necessarily a private communication, and so on
  - Educate users about how to prevent phishing attacks — don't click on email attachments or links from unknown (or even known) senders; authenticate (for example, via a phone call) that a request is legitimate
  - Use secure communication when possible
  - Physically protect computer hardware
  - Configure the operating system to minimize the attack surface; disable all unused services
  - Configure system daemons, privileges applications, and services to be as secure as possible

Implementing Security Defenses

Security Defenses Summarized

- Use modern hardware and software, as they are likely to have up-to-date security features
- Keep systems and applications up to date and patched
- Only run applications from trusted sources (such as those that are code signed)
- Enable logging and auditing; review the logs periodically, or automate alerts
- Install and use antivirus software on systems susceptible to viruses, and keep the software up to date
- Use strong passwords and passphrases, and don’t record them where they could be found
- Use intrusion detection, firewalling, and other network-based protection systems as appropriate
- For important facilities, use periodic vulnerability assessments and other testing methods to test security and response to incidents
- Encrypt mass-storage devices, and consider encrypting important individual files as well
- Have a security policy for important systems and facilities, and keep it up to date

An Example: Windows 10

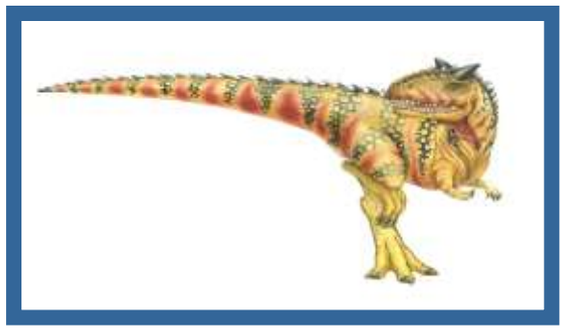
- Security is based on **user accounts**
  - Each user has unique security ID
  - Login to ID creates **security access token**
    - Includes security ID for user, for user's groups, and special privileges
    - Every process gets copy of token
    - System checks token to determine if access allowed or denied
- Uses a **subject** model to ensure access security
  - A subject tracks and manages permissions for each program that a user runs
- Each object in Windows has a security attribute defined by a security descriptor
  - For example, a file has a **security descriptor** that indicates the access permissions for all users

An Example: Windows 10

- **Windows** added mandatory integrity controls – assigns **integrity label** to each securable object and subject
  - Subject must have access requested in discretionary access-control list to gain access to object
- Security attributes described by **security descriptor**
  - Owner ID, group security ID, discretionary access-control list, system access-control list
- **Objects** are either **container objects** (containing other objects, for example, a **file system directory**) or **noncontainer objects**
  - By default an object created in a container inherits permissions from the parent object
- Some **Win 10 security challenges** result from security settings being weak by default, the number of services included in a Win 10 system, and the number of applications typically installed on a Win 10 system

CS 3104 – OPERATING SYSTEMS

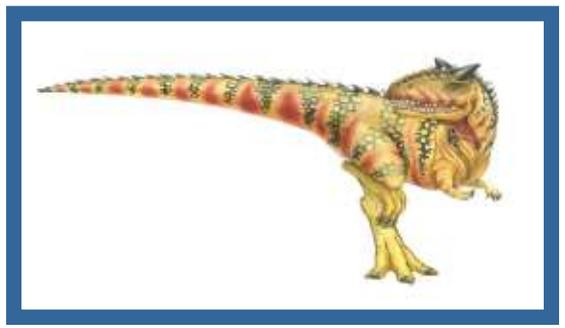
End of Chapter 16
Security



CS 3104 – OPERATING SYSTEMS

DCISM
DEPARTMENT OF COMPUTER, INFORMATION SCIENCES AND MATHEMATICS

Chapter 17: Protection



Protection

- Goals of Protection
- Principles of Protection
- Protection Rings
- Domain of Protection
- Access Matrix
- Implementation of Access Matrix
- Revocation of Access Rights
- Role-based Access Control
- Mandatory Access Control (MAC)
- Capability-Based Systems
- Other Protection Improvements Methods
- Language-Based Protection

Goals of Protection

- In one protection model, computer consists of a collection of objects, hardware or software
- Each object has a unique name and can be accessed through a well-defined set of operations
- **Protection problem:**
  - Ensure that each object is accessed correctly and only by those processes that are allowed to do so

Principles of Protection

- Guiding principle – **principle of least privilege**
  - Programs, users and systems should be given just enough **privileges** to perform their tasks
  - Properly set **permissions** can limit damage if entity has a bug, gets abused
  - Can be static (during life of system, during life of process)
  - Or dynamic (changed by process as needed) – **domain switching**, **privilege escalation**
  - **Compartmentalization** a derivative concept regarding access to data
    - Process of protecting each individual system component through the use of specific permissions and access restrictions

Principles of Protection

- Must consider “grain” aspect
  - Rough-grained privilege management easier, simpler, but least privilege now done in large chunks
    - For example, traditional Unix processes either have abilities of the associated user, or of root
  - Fine-grained management more complex, more overhead, but more protective
    - File ACL lists, RBAC
- Domain can be user, process, procedure
- **Audit trail** – recording all protection-orientated activities, important to understanding what happened, why, and catching things that shouldn’t
- No single principle is a panacea for security vulnerabilities – need **defense in depth**

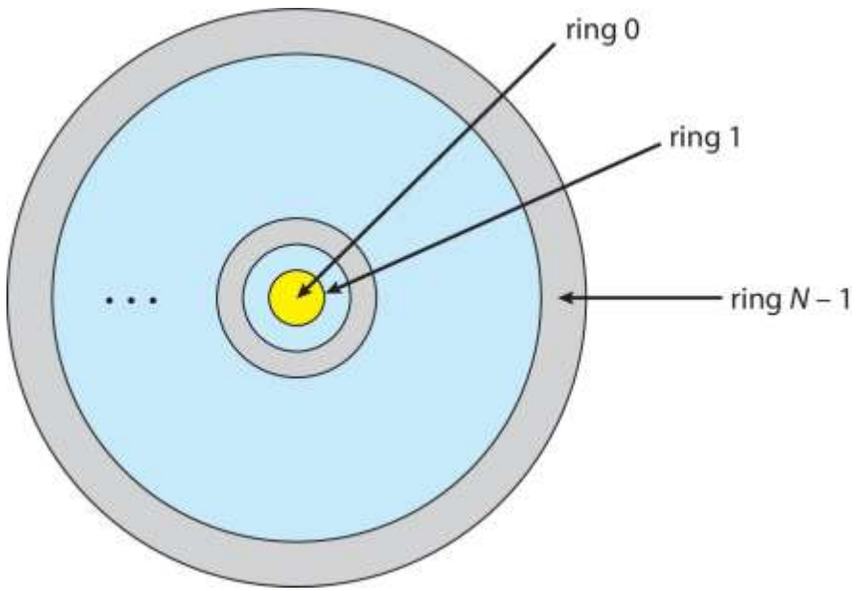
Protection Rings

- Components ordered by amount of privilege and protected from each other
  - For example, the kernel is in one ring and user applications in another
  - This privilege separation requires hardware support
  - Gates used to transfer between levels, for example, the **syscall** Intel instruction
  - Also traps and interrupts
  - **Hypervisors** introduced the need for yet another ring
  - ARMv7 processors added **TrustZone** (**TZ**) ring to protect crypto functions with access via new **Secure Monitor Call** (**SMC**) instruction
    - Protecting NFC secure element and crypto keys from even the kernel

Protection Rings

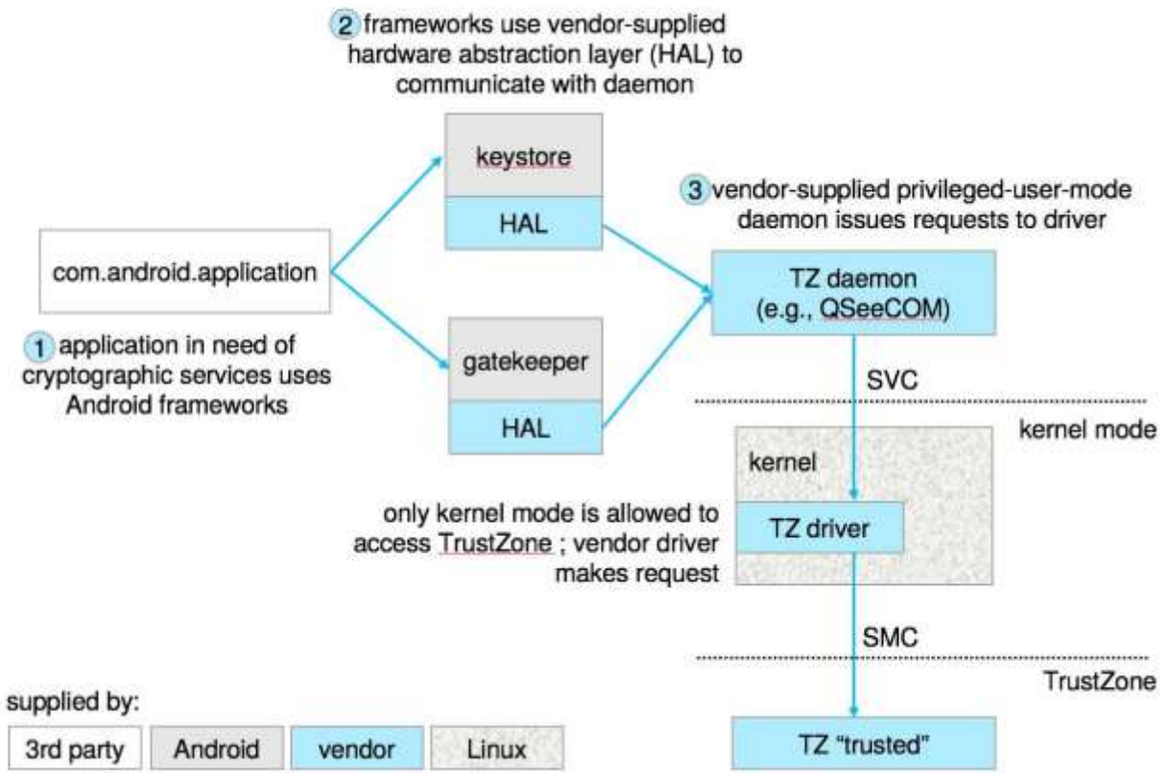
Protection Rings (MULTICS)

- Let D_i and D_j be any two domain rings
- If $j < I \Rightarrow D_i \subseteq D_j$



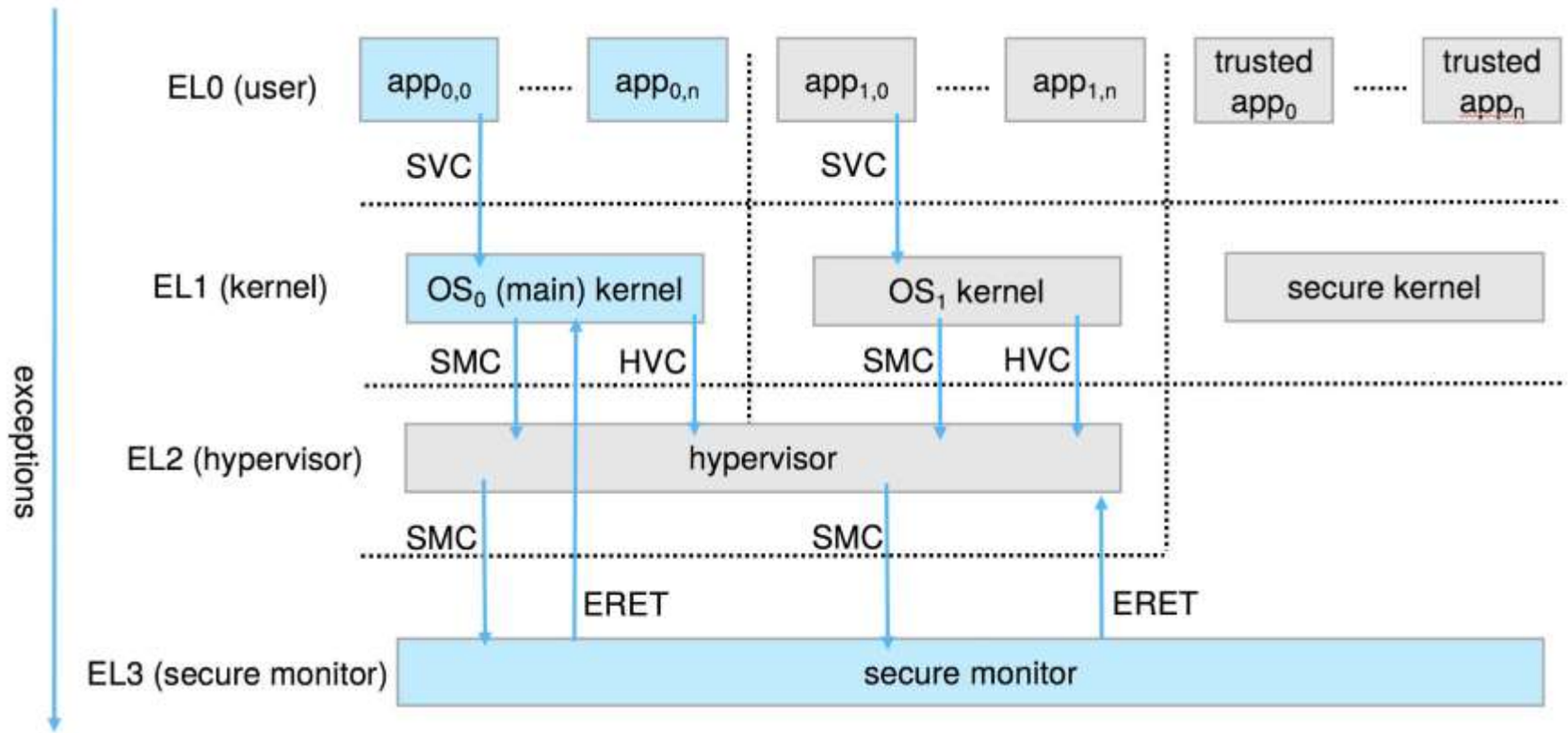
Protection Rings

Android use of TrustZone



Protection Rings

ARM CPU Architecture



Domain of Protection

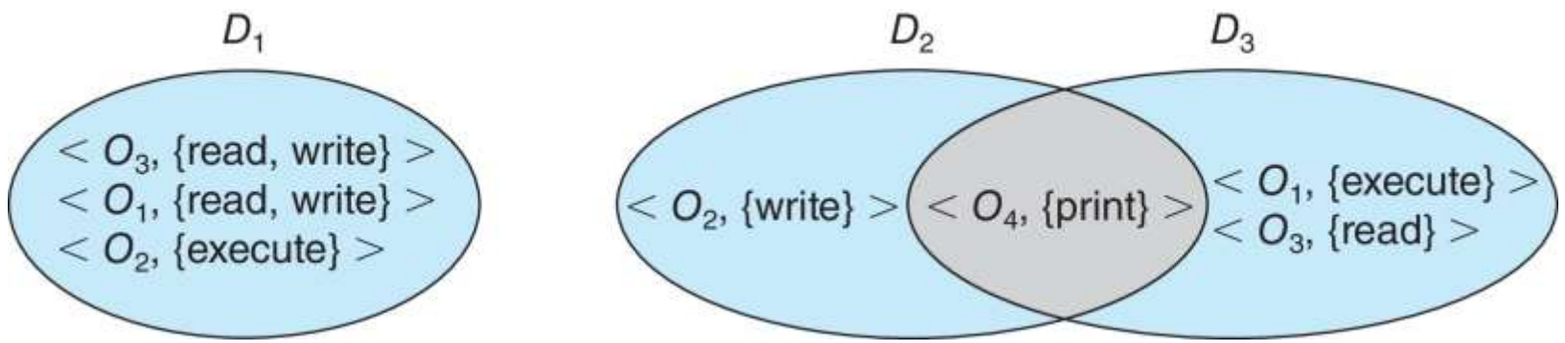
- Rings of protection separate functions into domains and order them hierarchically
- Computer can be treated as processes and objects
  - **Hardware objects** (such as devices) and **software objects** (such as files, programs, semaphores
- Process for example should only have access to objects it currently requires to complete its task – the **need-to-know** principle
- Implementation can be via process operating in a **protection domain**
  - Specifies resources process may access
  - Each domain specifies set of objects and types of operations on them
  - Ability to execute an operation on an object is an **access right**
    - <object-name, rights-set>
  - Domains may share access rights
  - Associations can be **static** or **dynamic**
  - If dynamic, processes can **domain switch**

Domain of Protection

Domain Structure

- **Access-right** = <*object-name*, *rights-set*>
  where *rights-set* is a subset of all valid operations that can be performed on the object

- **Domain** = set of access-rights



Domain of Protection

Domain Implementation (UNIX)

- Domain = user-id
- Domain switch accomplished via file system
  - Each file has associated with it a domain bit (setuid bit)
  - When file is executed and setuid = on, then user-id is set to owner of the file being executed
  - When execution completes user-id is reset
- Domain switch accomplished via passwords
  - **su** command temporarily switches to another user's domain when other domain's password provided
- Domain switching via commands
  - **sudo** command prefix executes specified command in another domain (if original domain has privilege or password given)

Domain of Protection

Domain Implementation (Android App IDs)

- In Android, distinct user IDs are provided on a per-application basis
- When an application is installed, the **installd** daemon assigns it a distinct user ID (UID) and group ID (GID), along with a private data directory (/data/data/<appname>) whose ownership is granted to this UID/GID combination alone
- Applications on the device enjoy the same level of protection provided by UNIX systems to separate users
- A quick and simple way to provide isolation, security, and privacy
- The mechanism is extended by modifying the kernel to allow certain operations (such as networking sockets) only to members of a particular GID (for example, AID INET, 3003)
- A further enhancement by Android is to define certain UIDs as “isolated,” prevents them from initiating RPC requests to any but a bare minimum of services

Access Matrix

- View protection as a matrix (**access matrix**)
- Rows represent domains
- Columns represent objects
- **Access(i, j)** is the set of operations that a process executing in **Domain$_i$** can invoke on **Object$_j$**

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

Figure A

Access Matrix

Use of Access Matrix

- If a process in Domain D_i tries to do "op" on object O_j, then "op" must be in the access matrix
- User who creates object can define access column for that object
- Can be expanded to dynamic protection
  - Operations to add, delete access rights
  - Special access rights:
    - *owner of* O_i
    - *copy op from* O_i *to* O_j *(denoted by "*")*
    - *control –* D_i *can modify* D_j *access rights*
    - *transfer – switch from domain* D_i *to* D_j
  - *Copy* and *Owner* applicable to an object
  - *Control* applicable to domain object

Access Matrix

Use of Access Matrix

- **Access matrix** design **separates mechanism from policy**
  - **Mechanism**
    - Operating system provides access-matrix + rules
    - If ensures that the matrix is only manipulated by authorized agents and that rules are strictly enforced
  - **Policy**
    - User dictates policy
    - Who can access what object and in what mode
- But does not solve the general confinement problem

Access Matrix

Access Matrix of Figure A (Slide 14) with Domains as Objects

domain \ object	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			

Figure B

Access Matrix

Access Matrix with *Copy* Rights

object / domain	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute		

(a)

object / domain	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute	read	

(b)

Access Matrix

Access Matrix With *Owner* Rights

object / domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		read* owner	read* owner write
D_3	execute		

(a)

object / domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		owner read* write*	read* owner write
D_3		write	write

(b)

Access Matrix

Modified Access Matrix of Figure B (Slide 17)

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 control
D_3		read	execute					
D_4	write		write		switch			

Implementation of Access Matrix

- Generally, a sparse matrix
- **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 $< D_i, O_j, R_k >$
    - with $M \in R_k$
  - But table could be large -> won't fit in main memory
  - Difficult to group objects (consider an object that all domains can read)

Implementation of Access Matrix

- **Option 2 – Access lists for objects**
  - Each column implemented as an access list for one object
  - 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

Implementation of Access Matrix

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

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

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

  Object F1 – Read
  Object F4 – Read, Write, Execute
  Object F5 – Read, Write, Delete, Copy

Implementation of Access Matrix

- **Option 3 – Capability list for domains**
  - Instead of object-based, list is domain based
  - **Capability list** for domain is list of objects together with operations allows 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 by domain
    - Rather, protected object, maintained by OS and accessed indirectly
    - Like a “secure pointer”
    - Idea can be extended up to applications

Implementation of Access Matrix

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

Implementation of Access Matrix

Comparison of Implementations

- Many trade-offs to consider
  - Global table is simple, but can be large
  - Access lists correspond to needs of users
    - Determining set of access rights for domain non-localized so difficult
    - Every access to an object must be checked
      - ✓Many objects and access rights -> slow
  - 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

Implementation of Access Matrix

Comparison of Implementations

- 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

Revocation of Access Rights

- **Various options** to remove the access right of a domain to an object
  - **Immediate vs. delayed**
  - **Selective vs. general**
  - **Partial vs. total**
  - **Temporary vs. permanent**

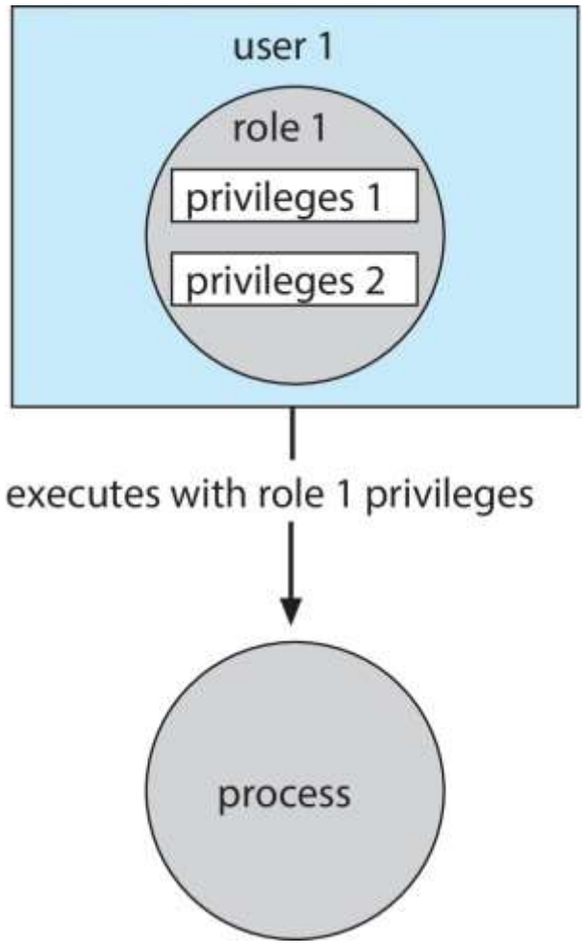
- **Access List** – Delete access rights from access list
  - **Simple** – search access list and remove entry
  - **Immediate**, **general** or **selective**, **total** or **partial**, **permanent** or **temporary**

Revocation of Access Rights

- **Capability List** – Scheme required to locate capability in the system before capability can be revoked
  - **Reacquisition** – periodic delete, with require and denial if revoked
  - **Back-pointers** – set of pointers from each object to all capabilities of that object (Multics)
  - **Indirection** – capability points to global table entry which points to object – delete entry from global table, not selective (CAL)
  - **Keys** – unique bits associated with capability, generated when capability created
    - Master key associated with object, key matches master key for access
    - Revocation – create new master key
    - Policy decision of who can create and modify keys – object owner or others?

Role-based 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 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



Mandatory Access Control (MAC)

- Operating systems traditionally had discretionary access control (DAC) to limit access to files and other objects (for example, UNIX file permissions and Windows access control lists (ACLs))
  - Discretionary is a weakness – users / admins need to do something to increase protection
- Stronger form is mandatory access control, which even root user can't circumvent
  - Makes resources inaccessible except to their intended owners
  - Modern systems implement both MAC and DAC, with MAC usually a more secure, optional configuration (Trusted Solaris, TrustedBSD (used in macOS), SELinux, Windows Vista MAC)
- At its heart, labels assigned to objects and subjects (including processes)
  - When a subject requests access to an object, policy checked to determine whether or not a given label-holding subject is allowed to perform the action on the object

Capability-Based Systems

- Hydra and CAP were first capability-based systems
- Now included in Linux, Android and others, based on POSIX.1e (that never became a standard)
  - Essentially slices up root powers into distinct areas, each represented by a bitmap bit
  - Fine grain control over privileged operations can be achieved by setting or masking the bitmap
  - Three sets of bitmaps – permitted, effective, and inheritable
    - Can apply per process or per thread
    - Once revoked, cannot be reacquired
    - Process or thread starts with all privileges, voluntarily decreases set during execution
    - Essentially a direct implementation of the principle of least privilege
- An improvement over root having all privileges but inflexible (adding new privilege difficult, etc.)

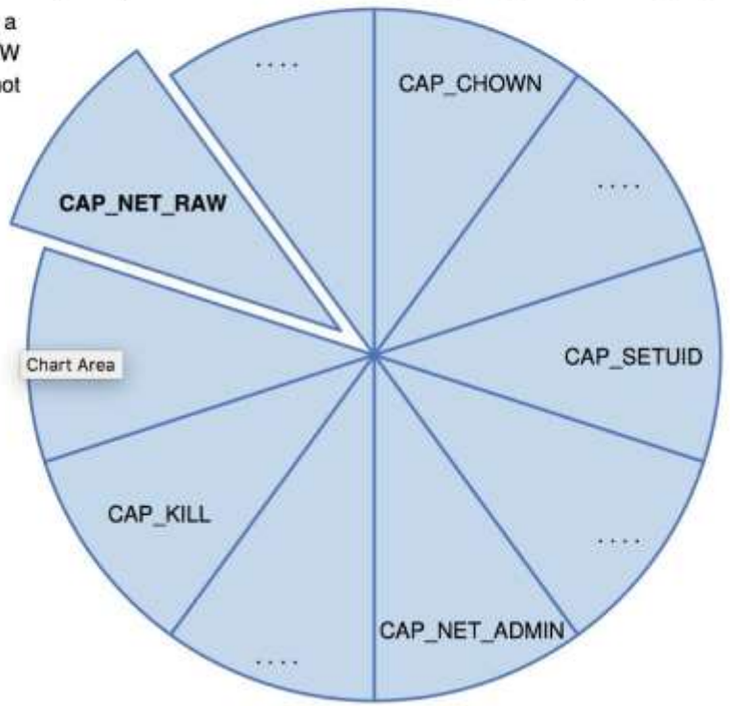
Capability-Based Systems

Capabilities in POSIX.1e

In the old model, even a simple `ping` utility would have required root privileges, because it opens a raw (ICMP) network socket

Capabilities can be thought of as "slicing up the powers of root" so that individual applications can "cut and choose" only those privileges they actually require

With capabilities, ping can run as a normal user, with CAP_NET_RAW set, allowing it to use ICMP but not other extra privileges



Other Protection Improvements Methods

- **System integrity protection (SIP)**
  - Introduced by Apple in macOS 10.11
  - Restricts access to system files and resources, even by root
  - Uses extended file attributes to mark a binary to restrict changes, disable debugging and scrutinizing
  - Also, only code-signed kernel extensions and only code-signed apps are allowed
- **System-call filtering**
  - Like a firewall, for system calls
  - Can also be deeper –inspecting all system call arguments
  - Linux implements via SECCOMP-BPF (Berkeley packet filtering)

Other Protection Improvements Methods

- **Sandboxing**
  - Running process in limited environment
  - Impose set of irremovable restrictions early in startup of process (before main())
  - Process then unable to access any resources beyond its allowed set
  - Java and .net implement at a virtual machine level
  - Other systems use MAC to implement
  - Apple was an early adopter, from macOS 10.5's "seatbelt" feature
    - Dynamic profiles written in the Scheme language, managing system calls even at the argument level
    - Apple now does SIP, a system-wide platform profile

Other Protection Improvements Methods

- **Code signing** allows a system to trust a program or script by using crypto hash to have the developer sign the executable
  - So code as it was compiled by the author
  - If the code is changed, signature invalid and (some) systems disable execution
  - Can also be used to disable old programs by the operating system vendor (such as Apple) cosigning apps, and then invaliding those signatures so the code will no longer run

Language-Based Protection

- Specification of protection in a programming language allows the high-level description of policies for the allocation and use of resources
- Language implementation can provide software for protection enforcement when automatic hardware-supported checking is unavailable
- Interpret protection specifications to generate calls on whatever protection system is provided by the hardware and the operating system

Language-Based Protection

Protection in Java 2

- Protection is handled by the Java Virtual Machine (JVM)
- A **class** is assigned a protection domain when it is loaded by the JVM
- The protection domain indicates what operations the class can (and cannot) perform
- If a library **method** is invoked that performs a privileged operation, the stack is **inspected** to ensure the operation can be performed by the library
- Generally, Java's load-time and run-time checks enforce **type safety**
- Classes effectively **encapsulate** and protect data and methods from other classes

Language-Based Protection

Stack Inspection

protection domain:	untrusted applet	URL loader	networking
socket permission:	none	*.lucent.com:80, connect	any

class:	gui:
	...
	get(url);
	open(addr);
	...
		get(URL u):
		...
		doPrivileged {
		open('proxy.lucent.com:80');
		}
		<request u from proxy>
		...
			open(Addr a):
			...
			checkPermission (a, connect);
			connect (a);
			...

CS 3104 – OPERATING SYSTEMS

DCISM
DEPARTMENT OF COMPUTER, INFORMATION SCIENCES AND MATHEMATICS

End of Chapter 17
Protection