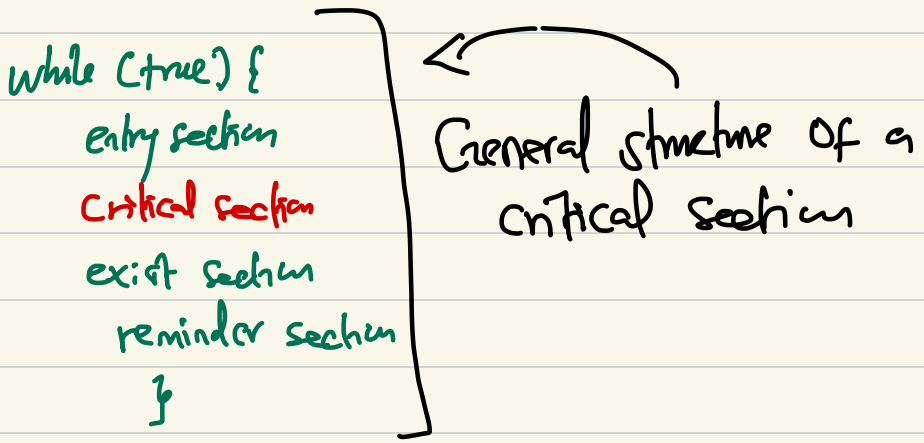
Chapter 6 : Synchronisation

* A critical section : Is a code segment that access shared variable and has to be executed as an atomic action.



Requirement for solution to critical-section problem

* Mutual exclusion :- if process P_i is executing in its critical section, then no other processes can be executing in their critical section.
* Progress :- if no process is executing in its critical-section, and some process wish to enter their critical section, then they shouldn't be postponed indefinitely.
* Bounded waiting :- a bound must exist on the number of times that other process are allowed to enter their critical section; after a process has made a request to enter its critical section before that request is granted.

Software Solution 1

* two process solution

Algorithm for Software solution 1

P_x

```
while (true) {
    while (turn == y);
    Critical section
    turn = y;
    reminder section
}
```

P_y

```
while (true) {
    while (turn == x);
    Critical section
    turn = x;
    reminder section
}
```

Note: Mutual exclusion is preserved. only one process's critical-section works at a time.

PETERSON'S SOLUTION

* peterson's algorithm is originally designed for two processes and it not very efficient for more than two processes.
  * it uses a busy-waiting loop to ensure mutual exclusion
  * each process that want to enter critical-section must contenously check the state of the other processes until it his turn to enter the critical-section.

two processes shared two variables:

int turn;

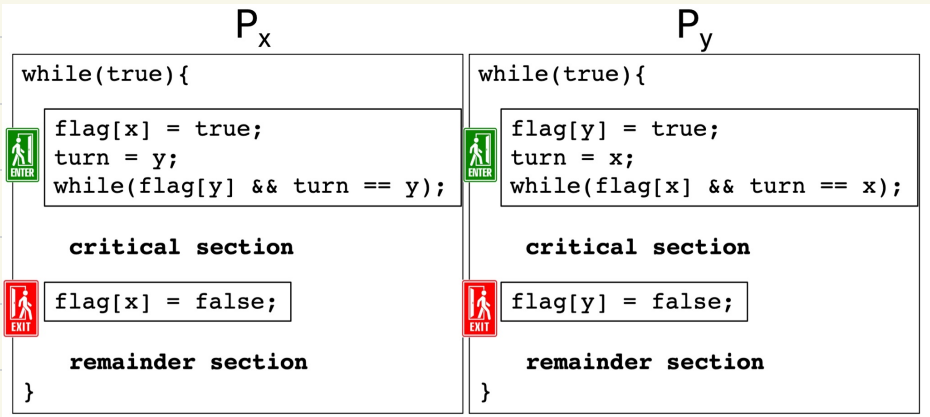
indicate whose turn its to enter (C-s)

is used to indicate if process is ready to enter (C.s)

boolean flag[2];

Flag [i] = true means p is ready

Algorithm for Peterson solution



Hard-ware support for perterson's solution

* Memory barrier : instruction that forces any changes in memory to be propagated to all other processors.

* Hardware instruction : Set of instruction that perform atomic action to change data. Example:

* test-and-set * swap * fetch-and-add * Compare-and-swap

* Atomic variables : provide atomic update on basic data types such as integer and booleans.

SYNCHRONIZATION TOOLS : MUTEX LOCKS

Mutex Locks:- is a boolean variable indicating if lock is available or not.

protect a critical section by:

- first acquire() a lock
- then release() the lock

→ must be atomic.

```
while(true){
    acquire lock
        critical section
    release lock
        remainder section
}
```

SYNCHRONIZATION TOOLS : SEMAPHORE

```
wait(lock){
    while (lock <= 0);
    lock--;
}
```

```
signal(lock) {
    lock++;
}
```

P_0 lock = false; P_1

```
while(true){
    wait(lock);
        critical section
    signal(lock);
        remainder section
}
```

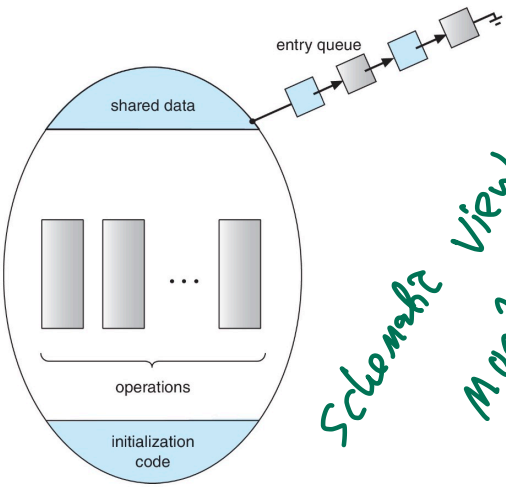
```
while(true){
    wait(lock);
        critical section
    signal(lock);
        remainder section
}
```

more sophisticated than mutex lock

SYNCHRONIZATION TOOLS : MONITORS

Monitors:- A high-level abstraction that provide effective mechanism for process Synchronisation.

only one process may be active within the monitor at a time



Schematic view of a Monitor

* **Liveness** : is a set of properties that a system must satisfy to ensure processes make process.

Example of liveness failure

Indefinite waiting

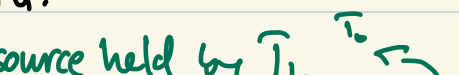
* **DEADLOCKS:** two or more processes are waiting indefinitely for an event that can be caused by only one of the waiting processes

Deadlock : Resources

* CPU cycles * Memory space * I/O devices

DEADLOCK condition

* Mutual exclusion: only one thread at a time can use a resource.
* Hold and wait: a thread holding atleast one resource is waiting to get additional resources held by other threads.
* No preemption: a resource can be released only voluntarily by the thread holding it, after that thread has completed its task.
* Circular wait: There exist a set of $\{T_0, T_1, \ldots T_n\}$ of waiting threads such that:
  * T_0 is waiting for a resource held by T_1,
  * T_1 is waiting for a resource held by T_2, ...



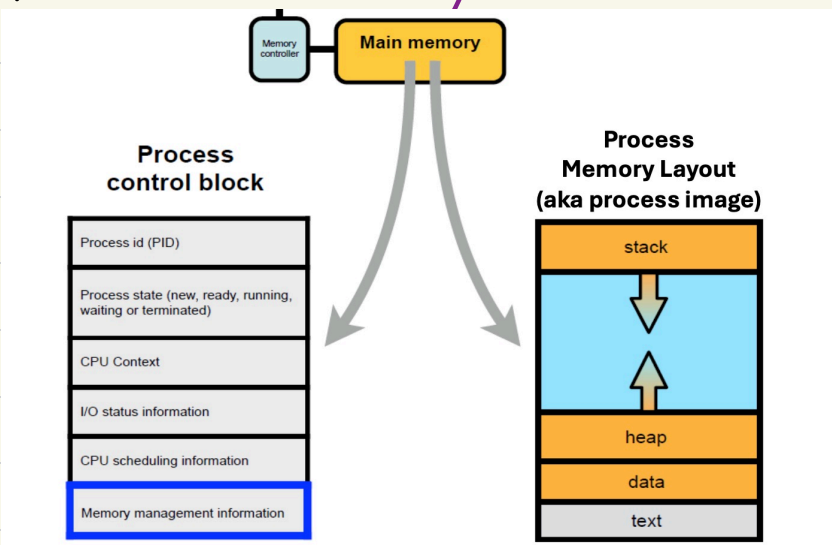
Methods for handling deadlocks

* Deadlock prevention:- prevent deadlock from happening in the first place: Invalidate one of the four necessary conditions for deadlock.
* Deadlock avoidance:- monitor the system for potential deadlock and take action to avoid them.
* Detect and recover:- Allow the system to enter a deadlock state, detect and then recover.
* Ignore the problem and pretend deadlocks never occur.

Chapter 7 : Main Memory

process creation

processes creation:- to run a program, the operating system must fetch the program executable from disk and place the program in a new process using fork() and exec() system calls in main memory.



Background:- program must be brought (from disk) into memory and placed within a process for it to be run.

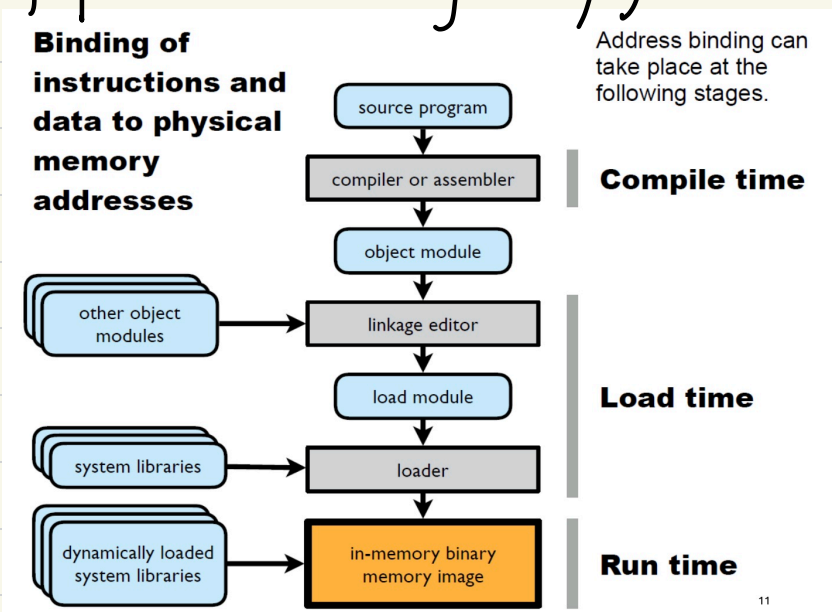
cache :- sits b/w main memory and CPU register

Memory protection:- is a mechanism that prevents one process from accessing the memory of another process.

Address binding scheme :- is the process of mapping logical address to physical addresses.

Types of Address binding scheme

* Compile-time address-binding : The compiler bind logical address to physical addresses during compilation.
* Load time address binding : The linker bind logical address to physical addresses when the program is loaded into memory.
* Execution-time address-binding: Logical address are bound to physical address during the program execution.



* Logical address (virtual address) : are generated by the CPU.
* physical address :- are addresses seen by the memory Unit.
* Logical address space : is the set of all logical addresses generated by a program.

aphysical address space : a set of all physical addresses generated by MMU

Main Memory Unit (MMU)

: It maintains number of functions such as memory protection, paging, and data segmentation. But its main function is to translate virtual addresses to physical addresses.

* Base register :- holds the starting physical addresses of a process's memory allocation.
* Limit register : specify the size or range of memory allocated to the process.
* Relocation register : holds the value to be added to a logical address to get the physical address. (as a dynamic base)

Contiguous Memory Allocation

* memory allocation :
  * fixed partition
  * Variable partition
  * Dynamic storage allocation
* fragmentation
* fixed-size partitioning :- The OS partition memory into partition of equal size.
* Variable partition : is the simplest method of allocating memory is to assign process to variably sized partition.

Dynamic storage-allocation problem

first fit: allocate the first hole that is big enough.
Best-fit: allocate the smallest hole that is big enough.
worst-fit: Allocate the largest hole; must search the entire list.

best fit and first-fit is better than worst-fit in terms of speed and storage utilisation.

fragmentation

External fragmentation: total memory space exist to satisfy a request, but its not contiguous.
Internal fragmentation: allocated memory may be slightly larger than requested memory; this size different is memory internal to a partition but not being used.
Compaction: shuffle memory contents to place all free memory together in one large block.

paging

paging :- a non contiguous; process is allocating physical memory whenever the latter is available.

ADDRESS TRANSLATION SCHEME

Address generated by CPU is divided into:

- **Page number** (*p*) – used as an index into a **page table** which contains base address of each page in physical memory
- **Page offset** (*d*) – combined with base address to define the physical memory address that is sent to the memory unit

page number	page offset
p	d

PAGING EXAMPLE

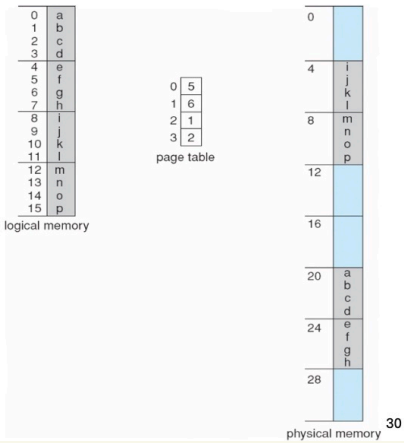
Using a page size of s = 4 bytes and a physical memory of 32 bytes (8 frames)

- Logical address 0 is page *p*=0, offset *d*=0; Page 0 is in frame *f*=5; **[(*f* × s) + d]**
  - Thus logical address **0** maps to physical address **20** [= (5 × 4) + 0];

Logical addr 3 -> physical addr 23

Logical addr 4 -> physical addr 24

Logical addr 13 -> physical addr 9



PAGING -- CALCULATING INTERNAL FRAGMENTATION

Page size = 2,048 bytes

Process size = 72,766 bytes

- 35 pages + 1,086 bytes (**36 frames** required)
- **Internal fragmentation** of 2,048 - 1,086 = 962 bytes

Worst case fragmentation = 1 frame – 1 byte

On average fragmentation = 1 / 2 frame size

So small frame sizes desirable?

- But each page table entry takes memory to track

Page sizes growing over time - typically either 4 KB or 8 KB

- Solaris supports two page sizes – 8 KB and 4 MB
- Windows 10 supports page sizes of 4 KB and 2 MB

Implementation of page table.

page table is kept in main memory.

* page table base register (PTBR): point to the page table
* page table length register (PTLR): indicate size of page table

EFFECTIVE ACCESS TIME

- **Hit ratio** – percentage of times that a page number is found in the TLB

An **80% hit ratio** means that we find the desired page number in the TLB 80% of the time.

Suppose that 10 nanoseconds to access memory.
- If we find the desired page in TLB then a mapped-memory access take 10 ns
- Otherwise we need two memory access so it is 20 ns

Effective Access Time (EAT)

EAT = 0.80 x 10 + 0.20 x 20 = 12 nanoseconds

implying 20% slowdown in access time

Consider a hit ratio of 99%,

EAT = 0.99 x 10 + 0.01 x 20 = 10.1ns

implying only 1% slowdown in access time.

39

* Memory protection: implemented by associating protection bit with each frame.

Swapping: A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution.

Standard Swapping:- involve moving the entire process b/w main memory and a backing store.

Swapping with paging:- a page of a process rather than an entire process can be swapped.

Chapter 8: Virtual memory.

Virtual memory

Virtual memory: is a technique that allow the execution of processes that are not completely in memory. it involve separation of user logical memory from physical memory.

Demand pagging

Demand paging: is a strategy that loads pages only as they are needed.

Valid - invalid Bit: when a process is executing, some pages will be in memory and some in secondary storage. the valid-invalid bit is used to distinguish between the two.

steps in handling page-fault

1- reference page
2- find free frame
3- Swap page into frame
4- Reset table
5- restart the instruction.

Performance (activities) of demand paging.

* Service the interrupt: is the process of handling the page fault that occurs when a program attempts to access a page that is not currently in physical memory.

* Read the page:- is the process of transfering page from secondary storge to physical memory.

* Restart the process :- is the process of restoring the CPU to its previous state and then resuming the execution of the program.

DEMAND PAGING EXAMPLE

Memory access time = 200 nanoseconds

Average page-fault service time = 8 milliseconds

EAT = (1 – p) x 200 + p (8 milliseconds)

= (1 – p) x 200 + p x 8,000,000

= 200 + p x 7,999,800

If one access out of 1,000 causes a page fault, then

EAT = 8.2 microseconds.

This is a slowdown by a factor of 40!!

If want performance degradation < 10 percent

- 220 > 200 + 7,999,800 x p
  20 > 7,999,800 x p
- p < .0000025
- < one page fault in every 400,000 memory accesses

Copy-on-write

COW :- allows both parent and child process to initially share the same page in memory. if either process modifies a shared page, only then is the page copied.

page replacement

refers to the selection of a frame of physical memory to be replaced when a new page is allocated.

BASIC PAGE REPLACEMENT

1. Find the location of the desired page on disk
2. Find a free frame:
   - If there is a free frame, use it
   - If there is no free frame, use a page replacement algorithm to select a **victim frame**
   - Write victim frame to disk if dirty
3. Bring the desired page into the (newly) free frame; update the page and frame tables
4. Continue the process by restarting the instruction that caused the trap

Note now potentially 2 page transfers for page fault – increasing EAT

PAGE AND FRAME REPLACEMENT ALGORITHMS

Frame-allocation algorithm determines
- How many frames to give each process
- Which frames to replace

Page-replacement algorithm
- Want lowest page-fault rate on both first access and re-access

Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string
- String is just page numbers, not full addresses
- Repeated access to the same page does not cause a page fault
- Results depend on number of frames available

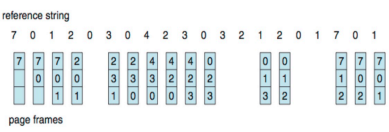
In all our examples, the **reference string** of referenced page numbers is

7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1

FIRST-IN-FIRST-OUT (FIFO) ALGORITHM

Reference string: **7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1**

3 frames (3 pages can be in memory at a time per process)



15 page faults

Can vary by reference string: consider 1,2,3,4,1,2,5,1,2,3,4,5
- Adding more frames can cause more page faults!
  - **Belady's Anomaly**

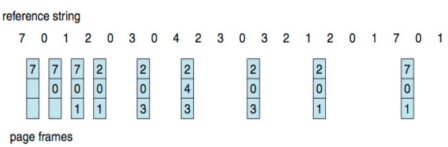
How to track ages of pages?
- Just use a FIFO queue

OPTIMAL ALGORITHM

Replace page that will not be used for longest period of time
- 9 is optimal for the example

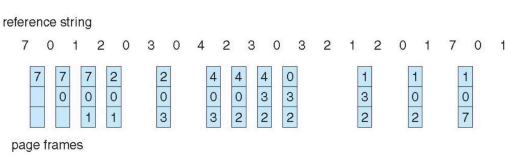
How do you know this?
- Can't read the future

Used for measuring how well your algorithm performs



LEAST RECENTLY USED (LRU) ALGORITHM

- Use past knowledge rather than future
- Replace page that has not been used in the most amount of time
- Associate time of last use with each page

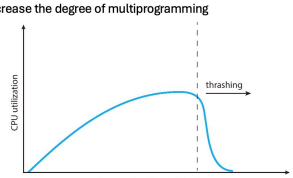


- 12 faults – better than FIFO but worse than OPT
- Generally good algorithm and frequently used
- But how to implement?

THRASHING

Thrashing. A process is busy swapping pages in and out

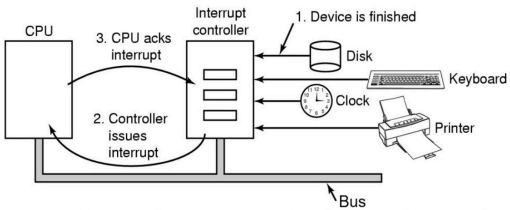
If a process does not have "enough" pages, the page-fault rate is very high
- Page fault to get page
- Replace existing frame
- But quickly need replaced frame back
- This leads to:
  - Low CPU utilization
  - Operating system thinking that it needs to increase the degree of multiprogramming
  - Another process added to the system



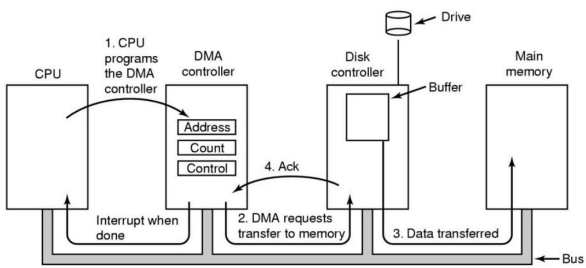
Chapter 9: its a very lengthy slide, so i just did what i can do. from A–Z.

* **polling:-** is a technique used to check the status of a device at a regular interval.
* **Direct Memory access (DMA):** is used to avoid programmed I/O for large data movement.

I/O SYSTEMS:
INTERRUPT-DRIVEN I/O (CONT.)



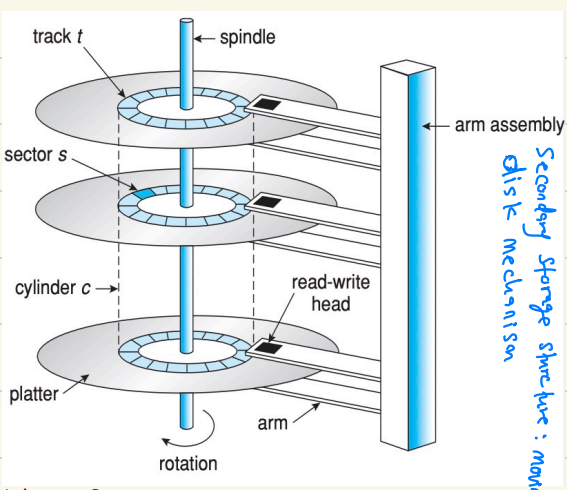
How interrupts happens. Connections between devices and interrupt controller actually use interrupt lines on the bus rather than dedicated wires



Operation of a DMA transfer

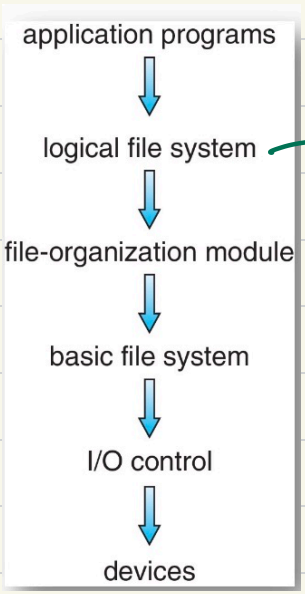
6 step to perform DMA Transfer

1- Device driver initialization
2- Driver Controller setup
3- DMA Transfer initiation
4- DMA Data Transfer
5. Compilation interrupts
6- CPU Task Resumption



* mounting a file :- is making a file system available for its user

file system layer



FCB: Contain many details about the file.

Boot Control Block: Contain information needed by the system to boot OS from that volume

metadata

Volume Control Block: Contain volume details.

Methods for allocating disk block to file

* Contigous: best performance, sequential & Random access
* linked: linked list of blocks
* Indexed: use index block to store pointers

file allocation table (FAT): is a variation on liked allocation

file attributes: Name, Identifier, Type, location, size, protection etc

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

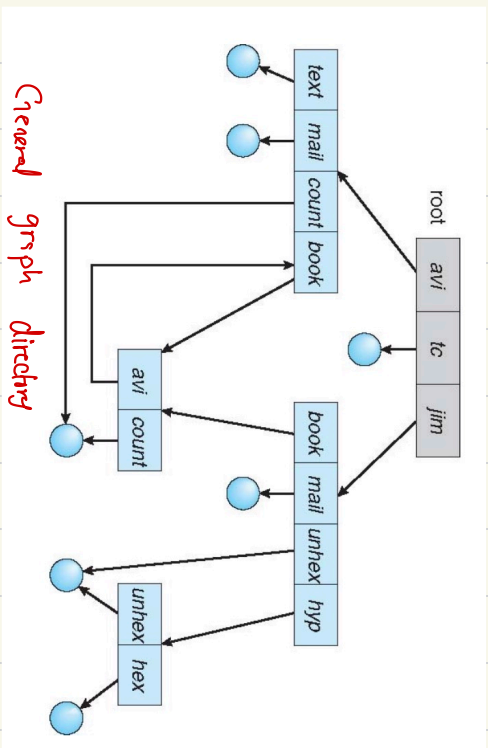
FILE SYSTEM:

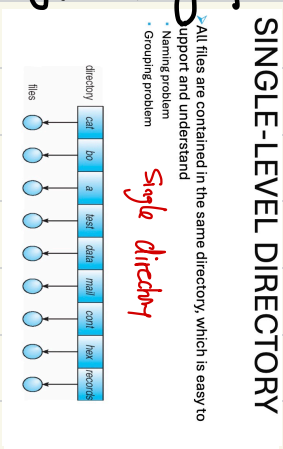
FILE OPERATIONS

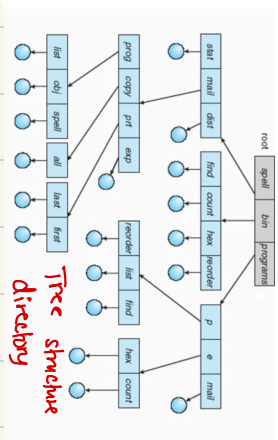
- **Creating** a file. Requires two steps:
  1. space in the file system must be found for the file.
  2. an entry for the new file must be made in a directory.
- **Opening** a file. All operations except create and delete require a file open() first. If successful, the open call returns a file handle that is used as an argument in the other calls.
- **Writing** a file. Need a system call specifying both the open file handle and the information to be written. The system must keep a write pointer to the location in the file for the next write.
- **Reading** a file. We use a system call that specifies the file handle and where (in memory) the next block of the file should be put. The system needs to keep a read pointer to the location in the file where the next read is to take place, if sequential.
- **Reposition within file(seek)** The current-file-position pointer of the open file is repositioned to a given value.
- **Deleting** a file. We search the directory for the named file. Once found the associated directory entry, we release all file space, and erase or mark as free the directory entry.
- **Truncating** a file. Allow user to erase the contents of a file but keep its attributes.

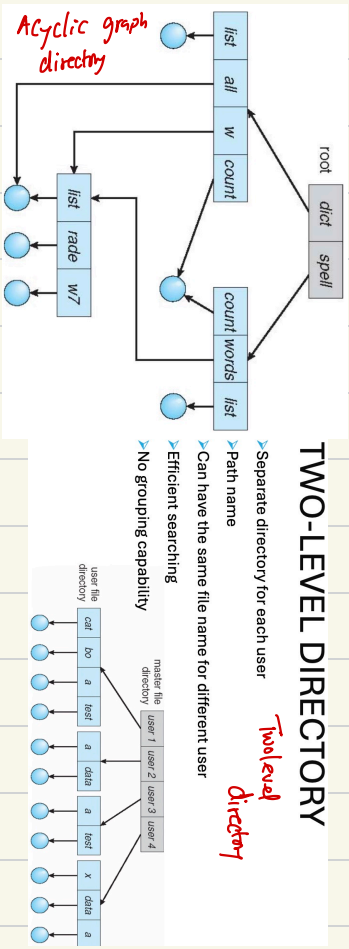
Directory structure

* Single-level * Two level * Tree-structure
* Acyclic-Graph * General graph.









FILE SYSTEM:

PROTECTION

- Protection mechanisms provide controlled access by limiting the types of file access that can be made.
- Access is permitted or denied depending on several factors, one of which is the type of access requested.
- 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.

FILE SYSTEM:

ACCESS LISTS AND GROUPS IN UNIX

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

Access		R W X
owner	7	1 1 1
group	6	1 1 0
public	1	0 0 1

Chapter 10

Chapter 10: protection & Security

Security & protection

Security: aim to protect against malesious intent and unauthorised access.

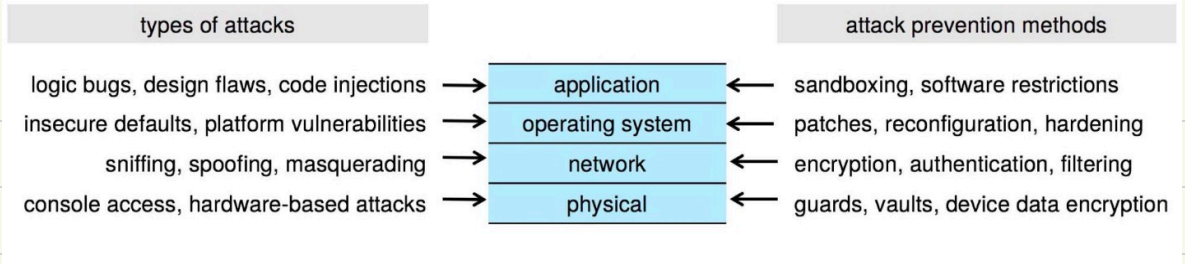
protection: - aim to protect against accidental or enviromental threats to the physical system and its data.

Thinking about security

* Hypothetical case:
  * loss of equipment
  * loss of data
* physical vandalism
* Electronic vandalism

four layered model of security

* physical layer:
  * Data Centre, Servers and connected terminals
* application layers:
  * Benign, or meliciaus app can cause problem.
* Operating system:
  * Debugging * protection mechanism
* Network layer:
  * DOS, Interuption, Intercepted Communication.



CONCEPTS

- **Privacy**: Protecting sensitive information from unauthorized access, use, modification, or destruction.
- **Authentication**: Verifying the identity of a user or device to ensure they are who they claim to be.
- **Integrity**: Ensuring the accuracy and completeness of data and systems, and preventing unauthorized modifications.
- **Protection**: Safeguarding digital assets, such as networks, systems, and data, from cyber threats.
- **Policy and Mechanism**: A set of rules (policy) and the technical safeguards (mechanism) to implement those rules, such as firewalls, intrusion detection systems, and encryption.
- **Encryption**: Transforming plain text into ciphertext to protect it from unauthorized access.

- **Principle of Least Privilege** (POLP): This principle states that users and programs should be granted only the minimum level of access necessary to perform their intended functions
- Programs, users and systems should be given just enough privileges to perform their tasks

Resource protection

* Set of subjects :- processes executing in a specific protection domain
* Set of objects :- all the passive elements of the system plus all the subject.
* Set of rule specifying the protection policy

Domain Structure :

* protection domain :- Set of right a process have at any given time
* Access right :- ability to execute operation in an object

Access matrix :-

* View protection as a matrix
* rows represent domains
* Column represent object

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	

- A process executes in domain D_1, can read files F_1 and F_3
- A process executing in domain D_4, have additional rights (write), on F_1 and F_3

Authentication :- Verifying the identity of a user or device to ensure they are who they claim to be.

Encryption :- Transforming plaintext into cipher-text to protect it from Unauthorised access.

System THREATS

* **Technical Vulnerabilities :-** like Software flaws, Configuration issue or outdated software. eg malware, Spyware, Trojan-horse, Ransomware, logic bomb, Trap door (back door), code injection, buffer overflow, virus, DDOS.
* **Human factors :** where user could be attack through social engineering, phishing attacks, or insider threads.
* **Organizational weakness :-** poor security policy and procedure

THREATS MONITORING

- **Intrusion Detection Systems (IDS)**:
  - Real-time monitoring: Continuously analyzes network traffic and system activity for anomalies.
  - Signature-based detection: Identifies known attack patterns.
  - Anomaly-based detection: Detects deviations from normal behavior.
- **Log Analysis**:
  - Audit log review: Examines system logs for suspicious activity, such as failed login attempts, unauthorized access, or unusual data transfers.
  - Security Information and Event Management (SIEM): Correlates and analyzes logs from multiple sources to identify potential threats.
- **Vulnerability Scanning**:
  - Regular scans: Periodically identifies and assesses vulnerabilities in systems and applications.
  - Penetration testing: Simulates attacks to uncover potential weaknesses.
- By combining these techniques, organizations can effectively detect and respond to threats, minimizing potential damage.

MONOCULTURE

- The use of identical software and hardware configurations, poses significant security risks.
- A single vulnerability can compromise multiple systems, making it easier for attackers to exploit weaknesses.
- By diversifying IT infrastructure with a variety of operating systems and hardware configurations, it becomes significantly more challenging for attackers to develop and deploy widespread exploits, thus enhancing overall system security.