Phyton Project 25/06/25

week 1:-

① OSI Model → open system interconnection Model

windows ⟷ (RJ45 / Ethernet) ⟷ window easy to communicate

windows ⟷ mac/linux connection using OSI Model

then are 7 layer :- → ISO 1984 → developer or international ORGanization standardization.

7. Application
6. Presentation
5. Session
4. Transport → Heart of OSI
3. Network
2. Data link
1. Physical

Software layer

Hardware layer

Ex: Hi → Encrypt / decrypt

Application layer

→ Network application

→ chrome, Firefox → HTTP, HTTPS, FTP, NFT, NNTP, DHCP, SMTP

→ File transfer, web surfing, Email, virtual terminals...

→ FTP, HTTP/S, SMTP, Telnet

Presentation layer:

Presentation layer receives from application layer

Application → Presentation → DFGJUYRG 6457899 5412 → translation → convert binary → Data compression → and decryption Encryption SSL secure socket layer

Ex: 5MB file → 2MB file

Helpful as real time video and audio streaming

Session layer

→ This layer help to setting and managing connection
→ Enable send and recieve

Session helper

→ API → Application Programming interface

- session management
- authentication
- authorization.

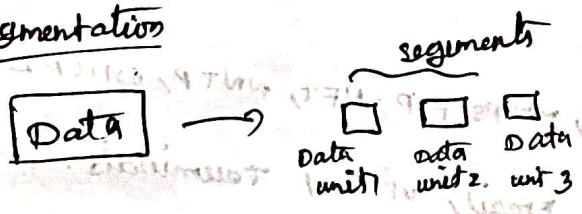
→ Session will be activate Data will transfer, session disconnect mean session timeout.

Transport layer :-

→ This layer control reliability of communication

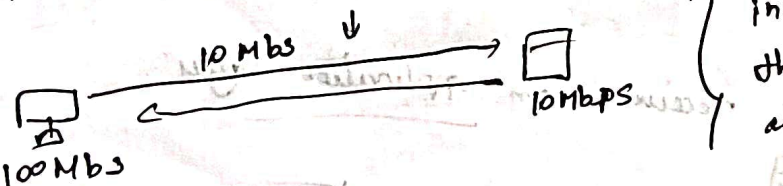
- segmentation
- connection and connection less Tx
- Flow control
- Error control

segmentation

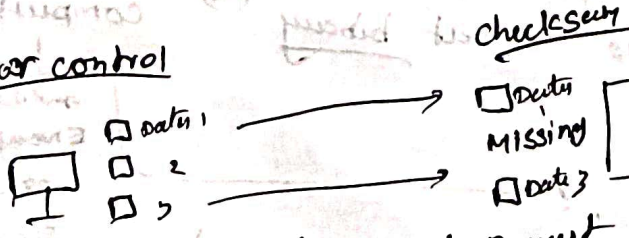


Flow control:

used to slow down the data transmission



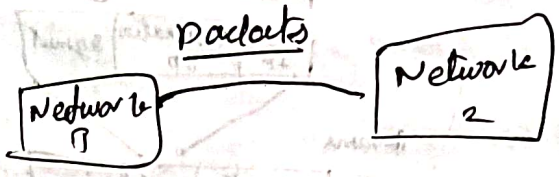
Error control



Protocols

→ TCP → Transmission control protocol → connection oriented transmission
→ UDP → user Datagram protocol → connectionless transmission

Network layer



Transport layer
↓ segment
Network layer

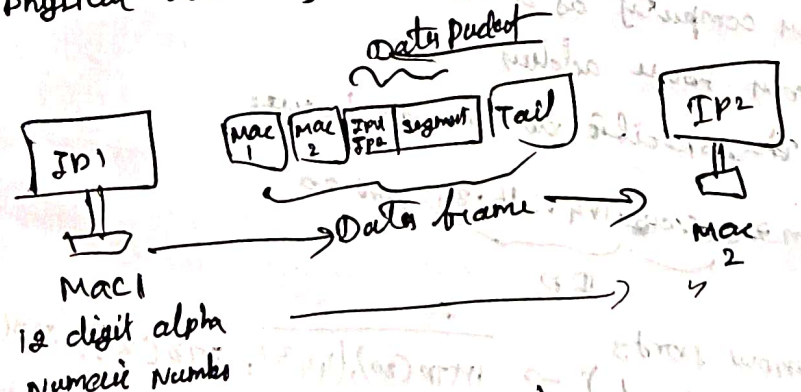
→ logical addressing → IPV4 and IPV6
→ path determination → ~~Hash~~ sender & receiver
→ Routing → IPV4 and IPV6 + Marking

OSPF => (Open shortest path first)
BGP => (Border Gateway protocol)
IS-IS => (Intermediate system to intermediate system)

Data link layer

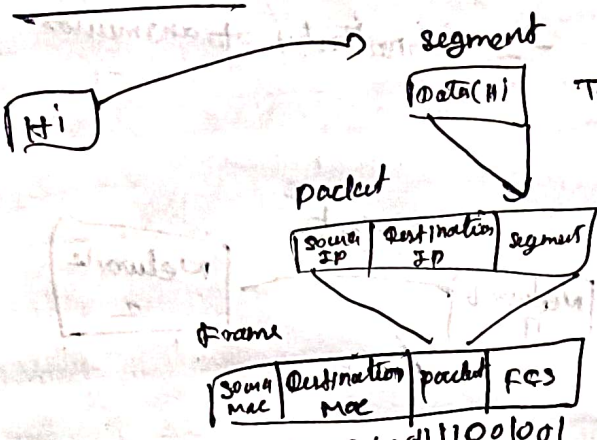
Network layer
↓ → Data packets from
Data link layer

→ logical addressing : Network layer (sender & receiver IP address)
→ physical addressing : Data link layer (system 1 and 2 IP address)



→ Framing (access the media)
→ (media access control) } control how data is placed and
& } received from the media
(Error detection)

physical layer



1001000 10111001001 → bit

↓

signals → sound & information

↓

media

security

application layer → web application Firewall, API security.

presentation layer → Data masking, Encrypt

session layer → session management, on timeout

transport layer → TCP security, HTTPS

network layer → IP security, Firewall

Data link layer → switch security, virtual Lan segmentation

physical layer → CCTV, access control

what is port Number :- → 16 bit unsigned int

a port Number is like as a door or channel inside a computer used to identify specific server or application when data arrives over the network

Ex → think of your computer as a house

→ ip address = your house address

→ port Number = the specific door in the house

https://example.com → IP:- 93.184.216.34, port 80

IP

0-1023 → well known ports (common protocol) → Ex HTTP(80), (443), FTP(21), SSH(22)

1024-49151 → Registry port (used by) → MySQL(3306), Docker, game used application)

49152-65535 → Dynamic/Ephemeral ports (temporary, auto assigned) → used by client communication

what is Mac address? → (48 bit) (6 byte) media access control

→ Mac address is a permanent address.

IP address → your home address (can change)

Mac address → your permanent Id card (unique and fixed)

Ex:
58 : 2D : 44 : 9B : 5C : D2

→ First 3 byte → Manufacturer ID (OUI - organizationally unique identifier)

→ last 3 byte → unique your device

what is socket :-

→ a socket like a virtual endpoint that enable communication between two device over the network

→ Think of it as a plug-point where your application connects to the Network

IP address → House address

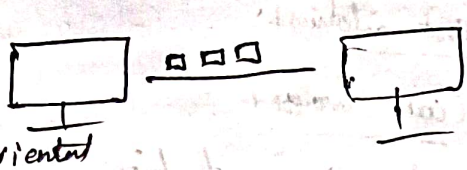
port Number → Door Number

Socket → Full connection (House + Door = complete address)

Ex socket = IP + port = 192.168.1.10:8080

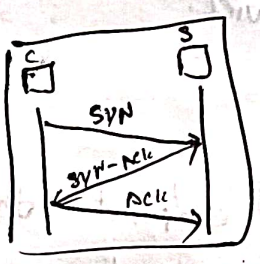
TCP / UDP

TCP

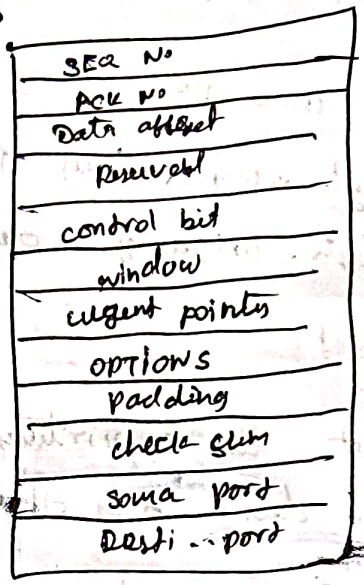


① connection oriented
② Seaquence
③ error checking, with packet retransmission

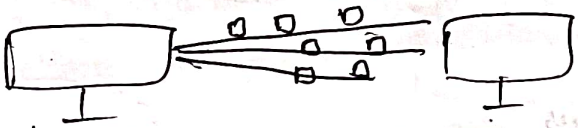
→ Highly Reliable
→ used in heavy apps
→ one to one
→ Delivery Guarantee
→ 3-way - Handshake
(Syn - Syn/Ack - Ack)
→ Fields
→ HTTP / HTTPS, SMTP, ~~Telnet~~ Telnet



Fields →



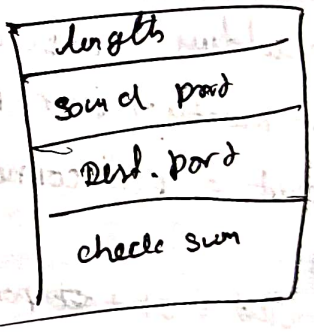
UDP



① connection less
② No seaquence
③ No error recovery

→ Fast
→ apps with fast Transmission → Games
→ Boardcast or Multicast
→ No Guarantee
→ No Handshake
→ fields
→ DNS, DHCP, VOIP

Fields →



26/06/25

State Machine:-

① Finite State Machine (FSM)

A finite state machine is a mathematical model of computation used to model behaviour.

→ a set of state (Idle, Processing, Error)

→ a start state (initial condition)

→ a set input event or condition

→ transition defines how the system moves from one state to another

Core concept:-

[IDLE] -(start_btn)-> [Running] -- (error) -> [ERROR] --(reset)-->[IDLE]

Key component in details:-

① States

State Represent the current situation or mode of operation

Ex:

→ For a vending Machine:- waiting_for_selection, waiting_for_Money, Dispensing, out_of_service

→ For uart → IDLE, Receiving, Transmitting.

② Event / inputs:

These are external / internal input that cause transition

Ex:

Button pressed, Timer timeout, sensore interrupt, received data from uart, internal flag (error_flag = 1)

③ Transition:-

The defines how you move from one state to another based on event

Eg.

IF (current_state == A) and (event == x)

Then move to state B do action y.

Action :-

→ During the transition (update a variable, send a signal)

→ inside the state repeatedly (blink led in Running)

Finite state Machine

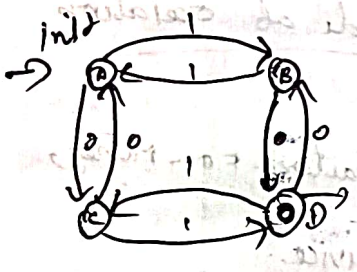
Finite Automata

- FA with output
  - Moore Machine
  - Mealy Machine
- FA without output
  - DFA
  - NFA
  - ε-NFA (Epsilon NFA)

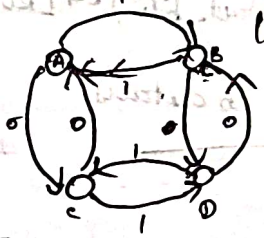
DFA - Deterministic Finite automata

↳ simplest Model of computation

↳ It has very limited memory



Structure



Non-deterministic Finite automata

A → 0 → B C

A → 1 → B

$(Q, \Sigma, q_0, F, \delta)$

Q = set of all state

Σ = inputs

q_0 = start state / initial state

F = set the Final state

δ = transition Function from $Q \times \Sigma \to Q$

Q = {A, B, C, D}

Σ = [0, 1]

q_0 = A

F = {D}

δ

	0	1
A	C	B
B	D	A
C	A	D
D	B	C

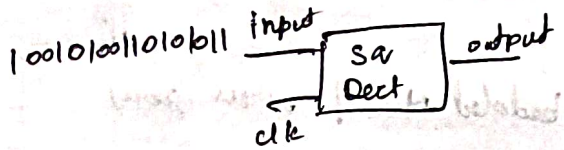
Mealy Machine

output depends on state + input. (Hardware)

Moore Machine

output depends only on current state. (sensors)

What is ~~Sequence~~ Savance Detector?

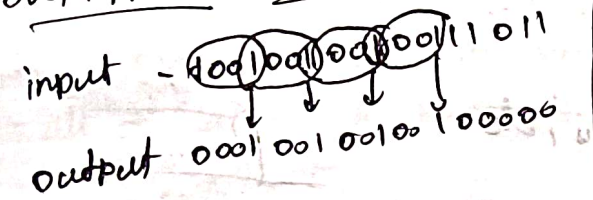


two type

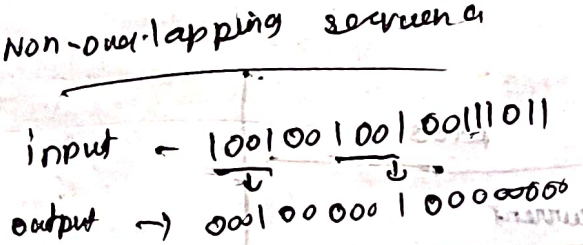
① over lapping sequence dector ⇒ last few bit part as another sequence.

② Non-over lapping " " → it finish, reset, search a new sequence.

① over lapping sequence Dectector:-

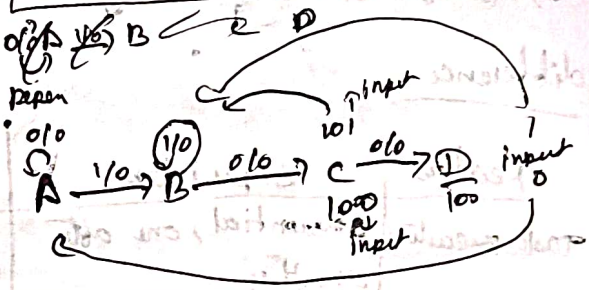


② Non-over lapping sequence

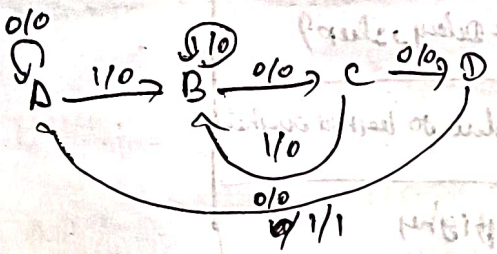


1001 → over lapping

Stat	Received bit
A	0
B	1
C	10
D	100



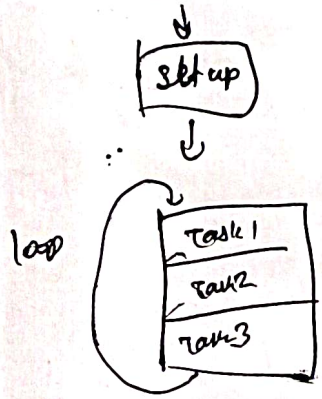
NON-overlapping



RTOS (VS) superloop

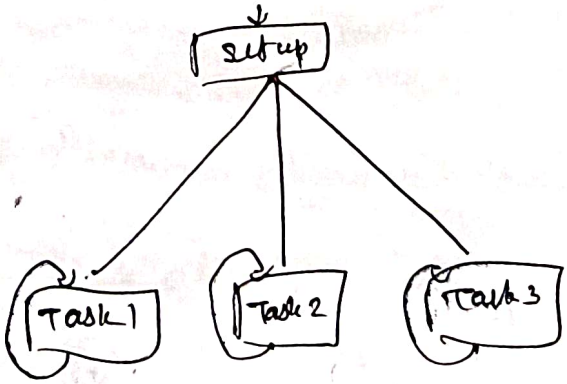
Super loop

Entry point (main)



RTOS

Entry point (main)



super loop:

) Run loop infinite time,.

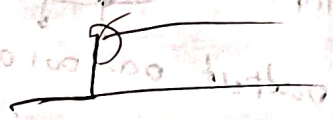
) a super loop is a bare-Metal embedded Model where your ode run in a continuous loop

→ common in small microcontroller (STM32, AVR)

RTOS :-

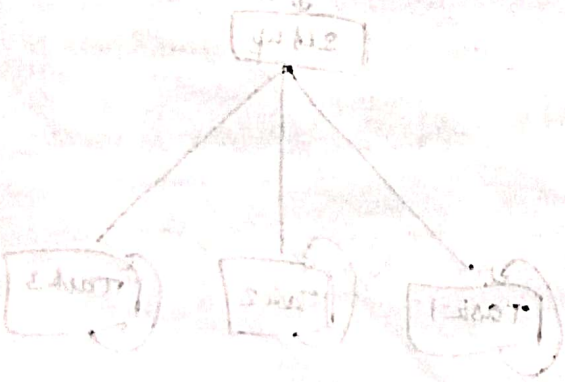
→ it allow multitasking (and multiple task (thread) run independently.

→ Manage CPU time, delay, priorities, etc



difference

Feature	Superloop	RTOS
Task Execution	sequential, one after another	concurrent
interrupt use	Heavily relies on interrupt	Task & interrupt
Delay	uses blocking delay bad-delay()	use non-blocking delay (vTaskDelay, sleep)
code size	small	large due to kernal overhead
Ram/Rom usage	low	Higher
Debugging	Easier	Complex due to multitasking



27/06/25

Thread :-

what is thread

→ A thread is the smallest unit of execution in a process

→ A process an independent program (with its own memory space)

→ A thread lightweight sub-task within a process
(shared memory with other thread in the process)

→ One process have multiple thread

→ all thread in a process share
  → "code"
  → "data"
  → File description But each thread has its own stack, registers, and instruction pointer

use case :-

concurrency → perform multiple tasks at once

performance → Thread reduce context switch, compared to process

parallelism → on multicore cpu, thread can truly run in parallel

resource sharing, responsiveness

types.

① user-level Thread (ULT)
  - Managed by : user space
  - Fast context switching
  - limitation, Fast portable
  - os doesn't know about them
  - Blocking one thread may block all

posix thread (pthread)

② Kernal-level thread (KLT)

- Managed by : the os kernal
- Each thread is know do the os and scheduled independently
- slowal context switching, More powerful
- one thread blocks, other can still run, haaviel

③ Hybrid thread

- combination ab ULT and KLT
- more complex do implement
- can manage the blocking

Linux

① using top or htop

→ top shows processes by default
→ press H in top to toggle thread views
→ CPU, memory usage per thread

what is thread size ?

① Stack size
② memory size

Each thread has
↳ own stack (for local variable, return address)

Linux default size
→ usually 8 MB per thread.
→ check using (ulimit -s)

Thread is actually working
→ shared the address space.

show the cpu threads
sudo perf top

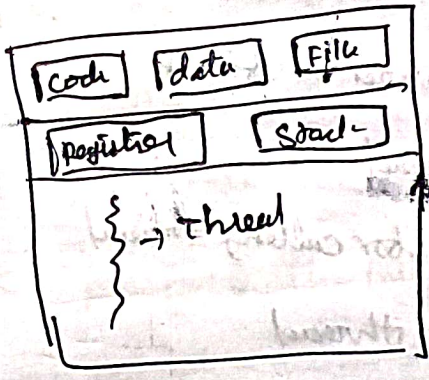
see program thread
ps -elf | grep <program name>

ps -elf # show all threads
pid stat -t -p <pid>
show per-thread cpu usage

using strace (system call)
strace -p <thread_pid>

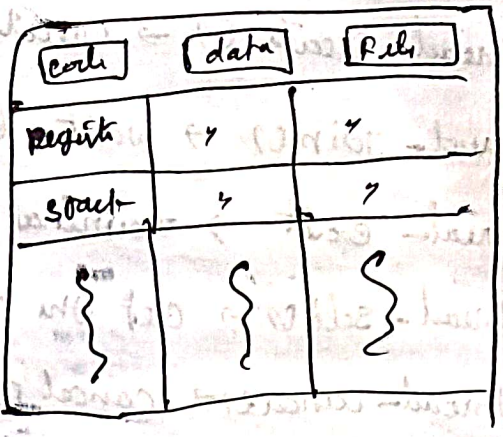
ulimit -s (stack size)
default size (8192)
8 mb
ulimit -v total thread

single thread process.

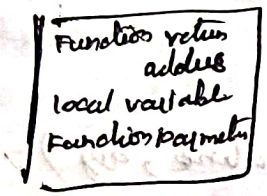


and TCB thread control block

multi-thread process



Stack layout per thread



top of stack (high address)

bottom of stack (low address)

State of thread / process :

R, S, D, Z, T

R → running

S → sleeping (waiting for event)

D → uninterrupt sleep

Z → zombie

T → stop or traced

Type of Thread

1. P-Thread → pthread_create()
2. K-Thread → kworker, kswapd is ps or top
3. O-Thread → operating system thread } Java, python, C#
   L) user + Kernel
4. Green thread → user space thread
5. Hybrid Thread → both (green + kernel)
6. RTOS Thread → light-weigh thread → Bare-metal.
   Ex: FreeRTOS task create().

Thread life cycle

New → Ready → Running → Blocking → Terminate

Functions :-

① pthread_create() → create a new thread

② pthread_join() → wait for a thread to finish

③ pthread_exit → terminal the thread

④ pthread_self() → Get the thread Id for calling thread

⑤ pthread_cancel() → cancel the running thread

⑥ pthread_detach() → Detach the thread
↓ clean ↓ thread complete clean the thread

① pthread_create() :-

int pthread_create (*thread, *att, start routine, arg);
↓ thread Id ↓ thread attribute default (NULL)
function to run in the thread
argument pass to function

Ex:
pthread_t tid;
pthread_create (&tid, NULL, My_funda, NULL);

② pthread_exit :-

void pthread_exit (void *retval);

Ex
pthread_exit (NULL);

③ pthread_join() :-

wait for a thread to finish (like wait() for process).

pthread_join (tid, NULL); → return value → main thread wait for others, avoid memory leak (zombie thread)

④ pthread_detach() :-

Ex: pthread_detach (tid);

⑤ pthread_self() :-

by returning the calling thread's Id

Printf ("Thread Id: %u", self());

⑥ pthread_equal():

→ compare two threads

int pthread_equal (pthread_t t1, pthread_t t2);

⑦ pthread init / destroy:

pthread_attr_init(&attr);

pthread_attr_destroy(&attr);

⑧ pthread_attr_setstacksize():-

pthread_attr_setstacksize(&attr, 1024*1024); 1MB

⑨ pthread_cancel(tid);

synchornization function (important in multithreading)

Pthread_mutex_init, lock, unlock,

pthread_cond_wait, signal, broadcast

wait for a condition variable — signal condition — wake all waiting thread

---

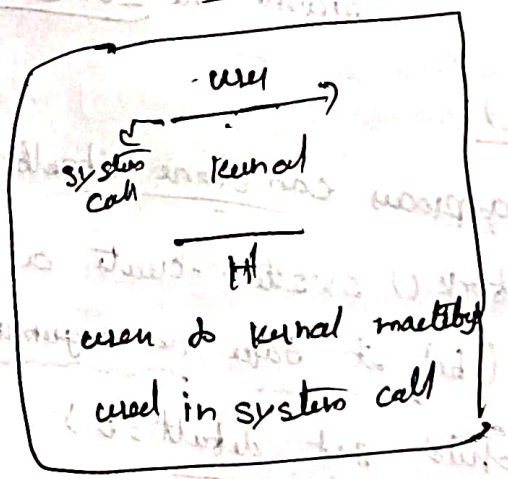
28/06/25

Linux Process

Process creation:-

- system() → library function
- fork()
- wait()
- wait pid()
- exec
- done()

→ system call



---

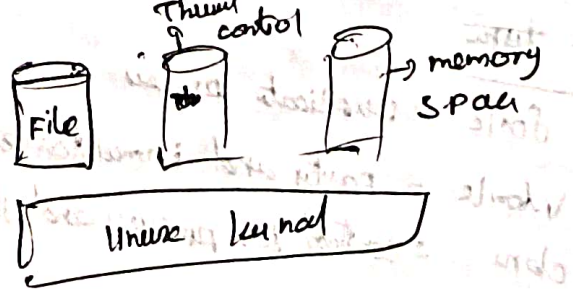
linux internals :- IPC → Inter process communication

Basic concept

① Process Model

② file system model

③ IPC



Process Model :-

- linux is everything has a process
- process identification
  (PID, PPID, SID, UID, GUID)
- top, ps -l, ps -u, man ps
  terminal command

- CPU registers
- PC
- stack pointer
- memory map
- signal handler
- scheduling parameter
- (UID) (GID)

} process

How do process is created :-

- init = Root process (OS) booting
- fork = make an almost exact copy of parent
  (COW) copy-on-write
- exec → overwriting memory image from file (or) code
  ↓
  keep the same PID, but load a new code
- wait → parent uses wait() to wait for the child to terminate and collect exit status
  ↓
  prevent the zombie process

Fork():

- a process can clone itself by calling fork
- fork() system-create a new process called child process. (but it take no argument and return PID)
  child get default = 0)

Ex:-

```
pid_t pid = fork();
if (pid == 0)
    child
{
}
else
{
  // parent
}
```

child :0

- use PTE (Page table entry) flag

Fork type

- fork = Duplicate process
- vfork = Fasty and immediate (exec) ↓
- clone = control for process and threads

Fork():

① do_fork (SIGCHLD, ...) // signal on child exit

Vfork():

do_fork (Clone_VFork | Clone_VM | SIGCHLD...)

clone():

do_fork (Clone_VM | Clone_Thread | Clone_Ideand, ...)

shared memory, signal, parent TID

Diffent :-

aspect	fork()	Vfork()
copies page table	yes (cow)	no
Parent block	no	yes
memory shared	no	yes
intended for	General process creation	Fast (exec)

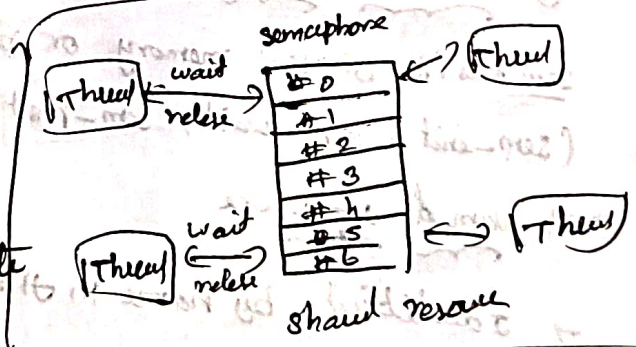
--init is used to only during initialization of boot time (or) Module insertion time.

--exit is used only module removed time for build in sections.

Semaphores

.a semaphore is a synchronization primitive.

it helps process or thread coordinate access the shared resource without conflicts.



County with automic operation.

→ Decrement (P or wait) value >0 decrement and counting

→ Increment (V or signal) increment the county, it process are waiting, wake up one

two types of methods,

pv() / wait()
vc() / signal()

① counting semaphore:- (allow count value)

→ The county can be >1 - controls access to a pool of resources (like N identical units)

② binary semphore (mutex - like) (allow only on 1 thread)

→ The county is 0 or 1 ; locks and unlocks the single resource

Types of semaphores

① system V semaphores

→ unix ipc semaphore

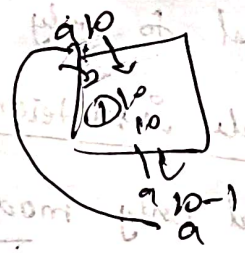
→ identified by an integer Id

→ operation : semget(), semop(), semctl().

→ manage the array of semaphore

sem_init(&sem, 0, 3)
up to 3 thread can be critical section
4 will be block

② posix semaphore :-



→ more modern

→ two type

① unnamed semaphore

sem_t is in shared memory or process memory

(sem_init, sem_wait, sem_post, sem_destory)

② named semaphore

→ Identified by name in the file system

(sem_open, sem_close, sem_unlink)

③ kernal semaphore :-

→ used for inside the kernal for synchronizing the kernal thread or interrupt handler

Ex: sorcet semaphore in linux kernal code

mutex :- owner only unlock

→ prevent the multiple thread from concurrent execution.

→ it work allowing lock and unlock the critical section.

→ only one thread can hold the mutex at a time

↳ if a second thread tries to lock it, it blocks until the mutex is unlocked

lock ↓0
critical
unlock ↓0

where mutex are used

→ protecting critical section as code

→ updating shared data structures (queue, counter, buffer)

Ex:

pthread_mutex_t lock = mutex_init;

pthread_mutex_lock(&lock);

// critical section.

unlock(&lock);

0 - mean lock
1 - unlock

spinlock :- (short critical section)

→ a spinlock is a low level locking primitive used to protect the shared data in multi core systems.

→ it also blocking and sleep like a mutex → no sleeping (busy wait)

cpu only unlock

Key idea

→ Light weight

→ No context switch

→ Good for short critical section

Ex:

while (lock is locked)
{
// spin - spin until it become free
}