4/3

Biased System

Boot loader [GRAD]

=> If user wants to interact with hardware then shell helps to interact with them.

=> Kernal is first & last program.

=> Bash = Bourne against shell

Kernel :-
- → Monolithic kernel
- → Micrlithic kernel

=> Services of kernel are:-
- → Device drivers
- → File system service :- Write to create, read & open or modified get properties of file & then close it. We can do same thing with directories.

→ device IO :- linux come with device file, we can open it by having particular operations.

→ Process management :- By using this we can use multiple process at a time.

By dividing process in smaller process to perform it by not changing their design.

⇔ Advance process management :- (Multi threading services) :- lightweight process & similar as multi process.

⇔ Memory management :- (MMU) → From user to kernel. ~~based~~ no. It crosses barrier & calling to MMU in kernel then MMU does the memory allocation & provides spaces. As they are pointer in user space they have their own address spaces.

⇔ IPC mechanism :- (within a system, Process to process system communication).

⇔ Network application :-
⇔ Timer
⇔ Signals
⇔ Synchronization technique.

⇒ Monolithic :- User :- application
linux :- OS.

⇒ Microlithic :- ~~Mic~~ Manage process Not in a single space particular procedure to do work. (Size is less)

8/3/23

GNU Binary Utilities

objdump -D Ex filename

↳ Disassembly.

Bin →

Exe → assemble

.ro ⇒ Read only data segment

Init → Runtime linker

Runtime code

→ Init → Initialisation

→ Start_return → Execution

↳ fin → deallocate resource

plt ⇒ Procedure linkup table.

readelf :- used in debugging

↳ 'readelf -H main' → header

↳ 'readelf -S main'

↳ Symbol information

Non-negative integer no ⇒ FD → File descriptor.

0,1,2 is used by computer we can't use them.

User can open 1024 files with a single program & it varies from 0 to 1023

int open ("const. character path name"), flags, (mode)

system call

(read / write) ; permission of files (777)

(mode_t) <

kernel data type

Library calls are provided by C language tools & system calls by a particular kernel of linux.

O_trunk → erase all data & write from begining of file.

mode_t → Internally int data type & is system dependent file.

⇒ Before using read we had to open it.

Read ("file descriptor", void ptr to buffer, time particular bit size)

Current file offset :- Each & every open file are associated with it.

Date
8/3/22

int dup (int fd);
↓
That duplicates copy or create file descriptor copy.

Advantage :-
Interchangeable, Saved.
We can use both copies at a same time for different flags & the copies can communicate with other.
↙
[It means that exchan data in read from Write in other.]

int dup2 (int fd, int fd10);
↳ It creates d at fd10.

=> fcntl ("int fd, int cmd, ...);
↓
F_DUPFD → Duplic
↓
System care

manage properties file from prog. P.O.

fcntl(fd, F_DUPFD, 40); → dup2

because we are providing our own value

Read can be done in shared mode write is exclusive i.e., (When 1 is writing others had to wait to write).

Process :-

Kernel Services

(2) Process ~~directory~~ management.

./a.out → execute file from current directory

a.out → execute file in Hard disk.

A program which is in execution is called as process

Linux can run multiple program at same time & concurrently in order to perform multitasking operation at any given pt. of time tence of appl^n running on your linux server machine.

ps → pid, Fd, cpu usage, currently executable name

⇒ ps -ax

↓

Process state

Each & every process is identified by unique id.

⇒ ps -elf :- → Only for user process not for kernel process
Gives more value regarding file.

Process → CPU type, Memory, I/O resources → Process requirement

Processes are isolated from each other & they run in independent environment.

Each process have their own resources.

Process ID → (PID) → Process identification number

Each process is allocated every identifier. When with unique it is created it is ~~created~~ allocated OS.

In process delete message of pid is done that means we can use it after completion of particular process.

For new process :-
Pids of recently terminated is not assigned immediately but after some time we can use it.

Swapper Process :-

Pid == 0 [Core kernel process]

pid = 1 Init process] Parent process
→ User process,
It launches system in multiuser input.

/sbin/init → path to find program of init process

Q: Write a program called fork maintain separate child & parent & from child again fork use?

Process termination

1. main return 0; } Normal
2. exit(0); } termination
3. Ctrl + C → Abno

Zombie process:- When a process terminate it sends a exit status or termination status to parent process. PP have to fetch & read terminated child process. If PP fails to fetch exit status then child enter into state known as zombie status. The process in a zombie status is known as zombie process.

The resources for the process have not completely deallocated from the kernel.

Once parent read exit status all resources are deallocated.

10/3/22

Fork & vfork both are used to launch new process but vfork suspends the parent process until termination.

The moment process called exec program process loads a brand new program from the Harddisk into its PAS & start executes the new program. Throughout it no new PID generated.

Q:- Difference parent & child. from parent fork & from child exec.

```
#include <stdlib.h>
atexit ( fn );
```

⇒ deferred routine's (fn's that are after some time delayed execution out).

It takes fn as an argument & registers with the kernel to execute later.

It executes later on normal process termination executes the fn in the reverse order of their registration.

Maximum 32 functions can be registered using atexit function.

Executes that f^n which takes nothing & expected to return nothing.
~~(void f)~~ { void f^n (void) }.

Thus,

atexit (void (* fn-ptr)(void));

It calls the registered function which are previously registered for a process by atexit function & executes in order of reverse order of the registration.

_ exit (0) function is not calling the f^n which are register by atexit function.

NI => using this we can prioritize by increasing ~~the~~ or decreasing of process.

LOCHAN => address in process space where Kernel is sleeping.

* TTY -> where process started.

Includes sys / wait.h >

pid_t wait (int status of terminated process);

waitpid (pid_t pid, int * location, int options);
(Null) (For a

particular child pid
child pid for which
or had

From parent we should call wait.

→ Wait block parent process until child's are executing.

→ Wait immediately return the exit status of of terminated child.

→ Error → When no child & you call for wait.

Wait PID is waiting for particular child.

Disadvtg of Fork :-

So, fork to maintain a process allocates lots of resources & very expensive in terms of CPU usage, memory & I/O usage.

Threads.

Creating, maintaining & destroying threads far cheaper than compare to process. They are called as light weight process & needs only few resources.

pthread_exit → will terminate threads but still pending threads execute

pthread_t pthread_self ( );
↓
calling thread id

pthread_self returns the calling thread id

pthread id stores 2 pointer (p) & the same value is stored to pthread_t (thid) pointer in create function.

Pthread Termination :-

1. main ( ) return.
2. Pthread_exit ( );
   → term calling

3. int Pthread_cancel (thid); → (terminate).
   → (error) → is used to cancel other threads using thread id.
   _cancel

(on success) Using pthread_function we can also terminate the running thread by providing pthread_self.

A thread task function can be shared by 2 different threads.

⇒ pthread attributes (properties) :- The behaviour of the thread, Once a thread created how it should behave.

↓

different Categories

- → Stack management.
- → Pthread scheduling Policy & privacy
- → Thread specific data
- → Thread synchronization.
- → Thread signal management.

~~provides~~

Pthread_attr_t :- Whenever U want change the default attributes are :-

declare an ~~of~~ an object of pthread_attr_t
initialize attribute object.

pthread_attr_init(pthread_attr_t * attr

Same type as of object.

appropriate thread appropriate
ead functions.

→ Spin lock :- Contension very expensive in terms of kernel resources.
Interrupts can be integrated ~~to~~ contd ~~seven~~ spinlocks.

→ Semaphore locks :-

- Declaration (Declaring a semaphore variable).

{ sem_t my_sem ; }

Data type to create an object

- Initialize the semaphore object :-

sem_init (my_sem , 0 , 1)

Non-zero (Shared b/w process).

Sharing the semaphore lock between threads of same process (so lock is visible to all threads of same process)

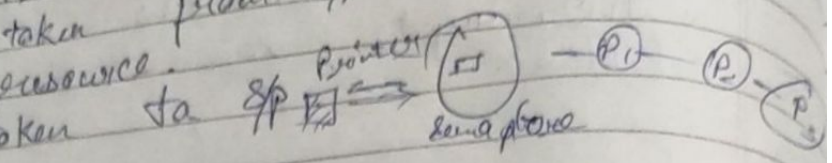
Call appropriate locking operation :- (lock)

sem_wait (sem_t *ptr);

sem_post (sem_t *ptr); (Unlock)

Process operation on semaphore :-

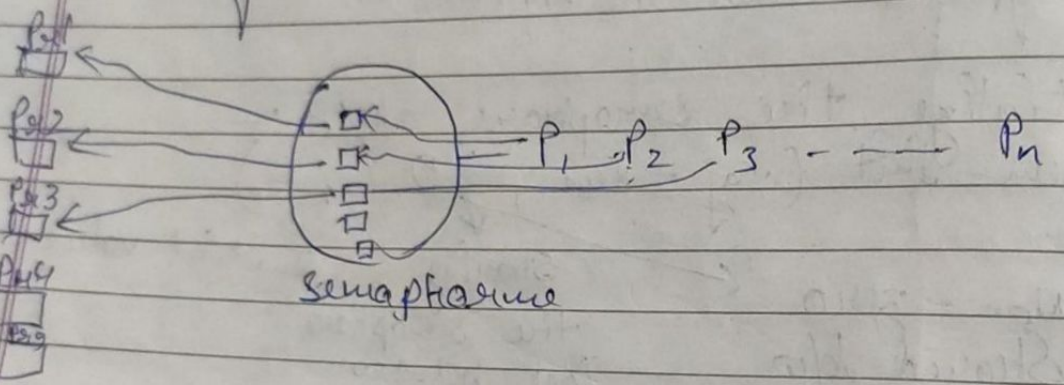
- Process take token from S/P.
- Use the resource.
- Give back token to S/P



Types of Semaphore :-

Binary Semaphore :- A semaphore whose token value changes between 0 & 1 or viceversa during all operation is known as binary semaphore.

Counting Semaphore :-



A semaphore whose token, value is greater than 1 is known as counting semaphore. In this every process take 1 semaphore variable & this reduce the token count move towards the assigned counter.

18/9/22

pthread_once ( )
↓
Initialization code, runs only 1 time

The purpose is to ~~create~~ implement initialization code.

pthread_once_t once; → (pthread_ONCE_INIT)

pthread_lib provides a macro = Pthread_ONCE_INIT;
↓
Setting this variable to initial value.

void (*funct, pt2)(v

=> int pthread_once (pthread_once_t *ptr,);
↓
a function which we want to initialize. (Initial code).

The moment pthread_once function executes checks for the 1st arg. whether the argument is initialized or not.

If it is [illegible]

If not initialized will never schedule the function

After successful execution of initialize code, the value is changed of variable.

Pthread_once schedule & initialize a function of type which takes no argument & return nothing.

pthread Scheduler Attribute

Inherit Scheduler member — Schedind — Pol & Pri.

$T_1 \rightarrow T_2 \rightarrow T_3$

Policy = X = X = ?
Priority = y = Y = ?

When a thread creates a new thread newly created thread will inherit the scheduling properties to newly thread according to posix library.

There is 1 member called as inherit scheduler member that specifies whether a thread is inheriting a policy & priority from its parent or managing on its own.

Inherit scheduler member

PTHREAD_INHERIT_SCHED
↓
From its parent getting policy & priority

PTHREAD_EXPLICIT_SCHED
↓
It is having policy & priority on its own.

pthread_attr_getinheritsched (&attr, int*);

pthread_attr_setinheritsched (&attr, &int);

Link thread scheduling priorities & policies

If the policy

policy
default RR FIFO
(Round robin)

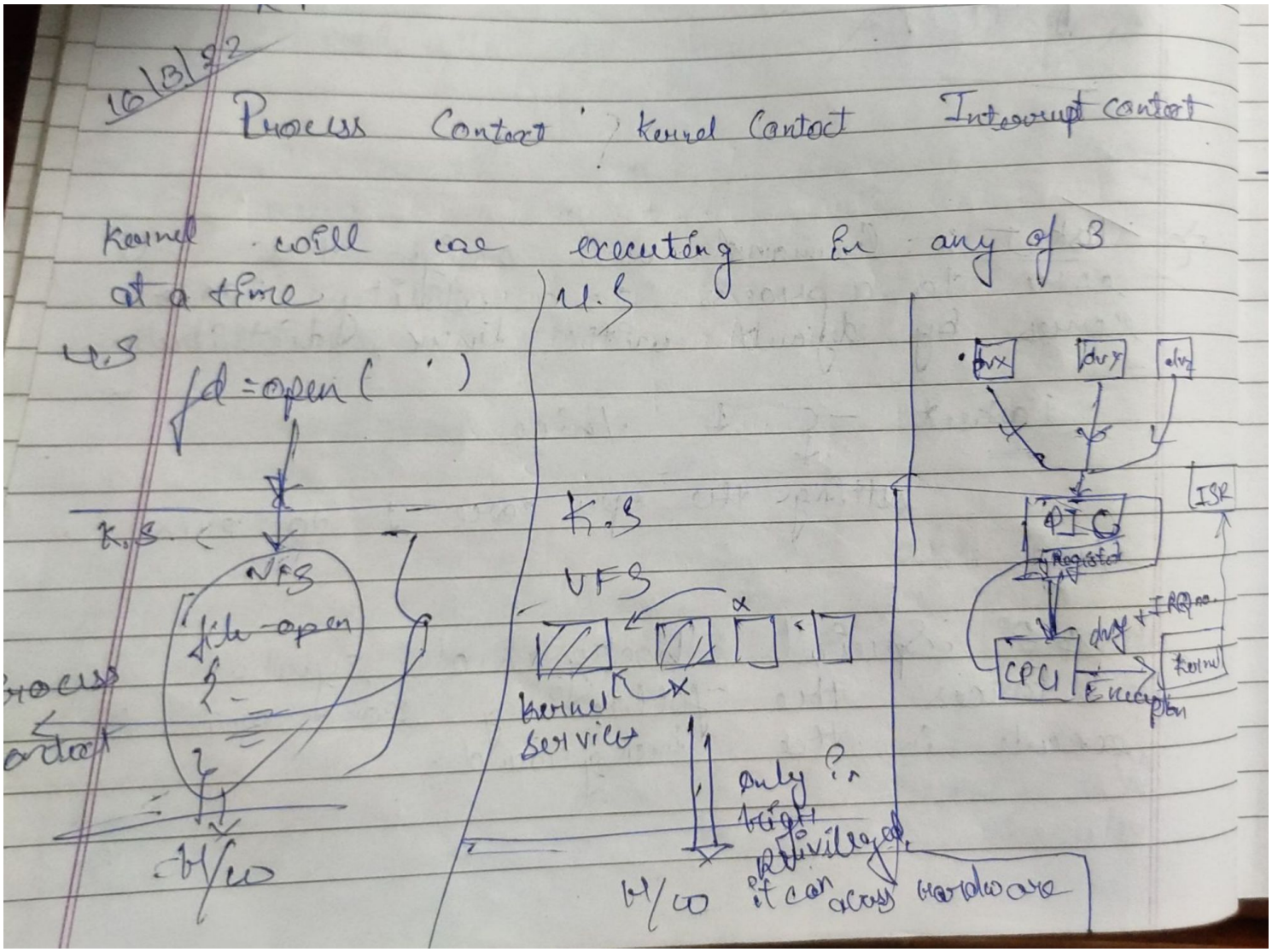
If the policy is default all the threads will have same priorities, CPU/time sharing & which run depends on FIFO.

Round-robin :- No issues of priority all the threads will get time in circular order

16/3/22

Process Context | Kernel Context | Interrupt context

Kernel will be executing in any of 3 at a time



Each & every hardware has a unique interrupt.

→ Kernel executing a piece of software on behalf of a process which has initiated the system call is called as process context.

→ Kernel executing a piece of software (kernel service) on behalf of another kernel service is known as kernel context.

→ Each & every on a system architecture is connected to Programmable IC (PIC), through a physical wire called as interrupt request (IRQ) lines.

→ Kernel executing a piece of software on behalf of interrupt is called as interrupt context.

Linux Memory Mangement (MMU)

Memory manipulation calls :-

memset (void * address, int c, size_t nbytes);

kernel datatype (int).

When process executes memset function jumps to the address location provided by pointer argument & start set the data with a given constant c for n no. of bytes.

void
memchr (void *addr, int c, size_t nbytes);

The moment process executes memchr function jumps to the address location provided by the pointer variable & & start scaning for the given constant c.

ret = interior ptr point into interior of the address
str = direct pointer pointing to start address

Memmove	Memcpy
Slow operation, reliable & generates guaranted data.	Fast operation, Not reliable,
- In case of memory overlapping (source & destination overlapped) it provides guaranted operation though the process is slow because having temporary ~~[illegible]~~ buffer.	- There is no temporary buffer.
- There is no risk of overlapping.	

Dynamic memory is allocated in heap.

& alloc a (size_t nbytes);

The ~~[illegible]~~ moment process executes alloc a memory call allocates memory from stack segment (heap segment is full) returns pointer to the allocated region on success.

When the new applications have done their job kernel will get back processes from swap partition to RAM is called as swap in process.

The entire operation is called as swapping

→ Memory ~~logs~~ locks :-

There could be a ~~logging~~ locking program kernel to swap-out we don't want locking process from RAM then my ~~logging~~ locking process memory ~~logs~~ locks to avoid swapping

ENOMEM

0 (Success)

→ int mlock(void* addr, size_t n bytes);

→ int mlockall(int flags);

(MCL_CURRENT)

The current process addr space is locked.

ENOMEM :- The error return by mlock ~~allocating more~~ when try to apply memory lock more than the permitted limit.

Date 1/10/22

problem with U.S & K.S design.

→ my memcpy (char *src, char* dstn, size_t nbytes);

⇒ Mmap condition :-

If an application is making repeated I/O request much of CPU time spent in submitting I/O operation, mmap is a POSIX memory function that maps a given kernel file region or a device region or some random kernel memory into the calling process address space.

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

* mmap(void *address, size_t nbytes, int protection, int flags, Fd/-1, off_t offset);
```

void * addr :- return my process address space we had to map.

size_t nbytes :- no. of bytes we want to map.

int protection

- PROT_READ → Pages may be read
- PROT_WRITE → Pages may be written
- PROT_EXEC → pages may be executed.

int flags :-
- MAP_LOCKS → applying mlock to shared region
- MAP_SHARRED.
- MAP_PRIVATE
- MAP_ANONYMOUS.

(doesn't belong to any file & 6th argument is - & 6 is 0).

(The shared memory is shared b/w n no. of process.)

MAP_PRIVATE

Create a private copy & changes will not reflect to other process which are mapping to shared region.

It is recommended to use 0 as address argument. 0 indicates asking Kernel to map it free process address space.

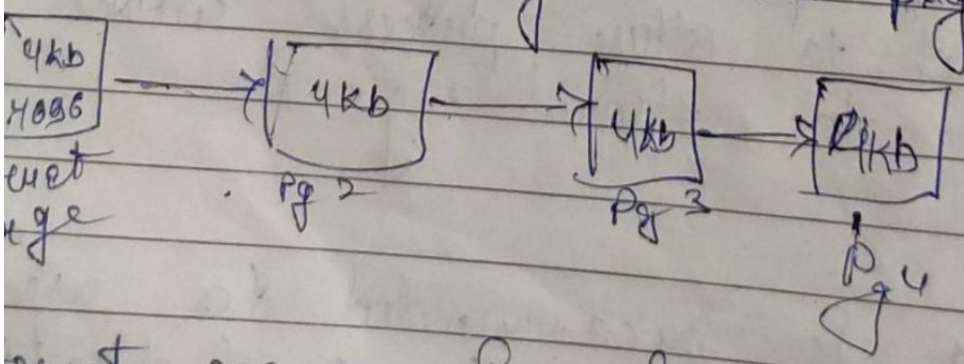
- Fds — In case of file ofd or if there is no file then -1
- offset :- It is kept 0.

Linux MMU

- High level MMU → independent of process architecture
- Low level MMU → dependent on process architecture.
  (~~based on ARM~~)

When kernel booting starts low level MMU also starts & creates ~~lots of~~ & initialize lots of kernel memory data structure.

Low level MMU converting entire memory into struct pages. Entire memory is linked list of struct pages.



struct page size is 32B.

Entire memory translation

~~For every~~ I/O take the page no. & Processor will look for a matching frame no. in page frame relation table. & page no. fetches the respective frame number.

Now take the frame no. & add to offset leading to physical address of variable x.

For each & every I/O request operation processor has to perform logical to physical address translation.

These entire operation is known as virtual memory address translation

Virtual memory: A section of memory created temporarily on swap partition of the storage devices

Linux uses RAM & virtual memory & virtual address are assign to their process.

→ few command provide amount of available virtual & virtual memory used by system

At time of swapping it creates a virtual memory in swapping memory.

→ vmstat :- complete virtual memory statistics reporter.

→ cat /proc/meminfo :- To get details information about memory

table vmstat command

'b' - uninterrupted process

'free' - available virtual memory

'buff' - virtual memory in terms of buffer

'si' - Swap in

'so' - Swap out

'bi' - blocks received from blocked device.

'bo' - block sent to blocked device

'in' - no. of interrupts per second.

'cs' - no. of context switch per sec.

→ Library :- It is a group of precompiled object files.

(okay, because linker only need object files to generate code).

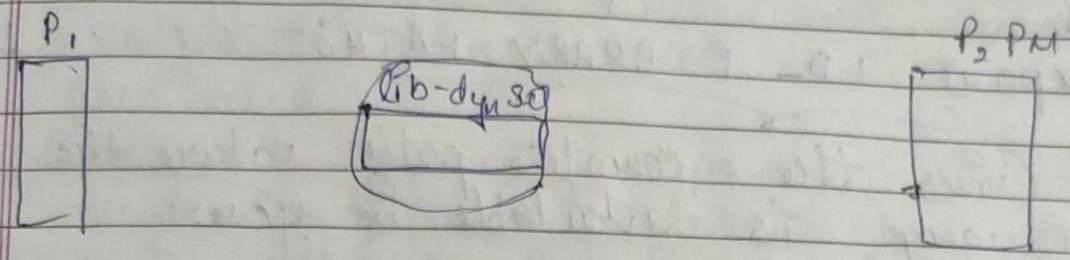
→ Static Library :- Libraries are 1 which are statically linked to program executable time at compile time.

→ Dynamic libraries :- Libraries are 1 which are dynamically linked to programs executable time at runtime.

Ldd main

21/3/22

Dynamic files have extension .so & also called as shared libraries.



∴ PIC (Position independent code)

A dynamic library creation uses a flag called as PIC. Dynamic libraries should be relocatable that means the instruction of dynamic library should get ready to get load into any memory address space.

Eg:- A ~~process~~ executable file needs a dynamic library so need the address of dynamic library & ~~to~~ DL merge with the file in a such a way that dynamic library appears to part of file. Later a another executable file requires a dynamic ~~to~~ the instruction of library should merge with memory of executable file.

'-shared' is used to generate dynamic library

In static library more size is required because it include all libraries.

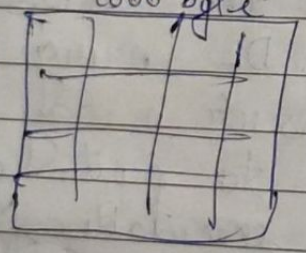
'ldd' – A linux command line tool will provide the dependencies required for executable file.

=> export LD_LIBRARY_PATH= ./ ./ ./ ./ library

Gives the complete path where the library is available in your directory.

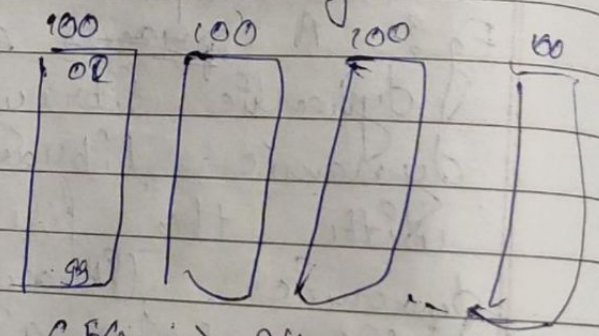
PROCESSOR

Real mode
→ Series of arrays of bytes
1000 byte



CPU → 650

Protected mode
1000 bytes



650 → Blk no. 6 + off 50
Virtual address

It all depends on processor how it is looking at memory. Based on OS will form policy to manage memory. processor will look memory into 2 different ways 'real mode' & 'protected mode'.
Once a system is power on during execution of bios code & boot loader code processor is in real mode looking memory as series (array of bytes.

Once the Kernel boot-up starts processor shifted to protected mode & looking memory as a set of blocks (linkedlist of pages) & this processor is called as processor initialization

Kernel OS creating some illusion & making the CPU to look memory as set of blocks.

~~GNU (G~~

GDB (GNU debugger TOOL):-

GDB is a open source software is a command line tool works with executable file that are produced by compilation process.

Difference B/w next & step?

'run' — To run to particular break point if no breakpoint then whole program is executed.

'next' — moving to next step.
'q' — To quit the program.

'print' — will print the variables value.

gdb --quit./gdb — To remove licence detail
(gdb --q ./gdb)

Codes to debugg program using command line.

→ gdb --args ./executable file argements

→ gdb./exe

gdb & argument
→ gdb ./exe
(gdb) set args argument

→ If we observe real time debugging of most of the times bugs happen to be from user point of view.
Eg:- accidentally dereferencing a NULL pointer, dereferencing uninitialized pointer, dereferencing a pointer which is

22/3/22

'ptype -variable name'' :- Gives the data type & information regarding variable.

'attribute_alligned' :- It specifies phase provide minimum allignment for the structure variable otherwise it takes 4 bytes as default for allignment.

gdb fails to track down heap memory violation.

'p sizeof' - To get size of particular variable by writing their name in front of them.

=> 'electric fence' :- It is an library (not a debugging tool) which has got its own implementation of malloc & calloc allocation calls.

Using it when requesting dynamic memory allocation using malloc & calloc will allocate for specified amount of memory.

- defence

Standard C library malloc & calloc allocating more no. of bytes then actually requested for. (In pages allocation is done by multiple of 16 bytes as a page ~~of~~ 1-kb of space).

After particular occution of electric fence library if it is of upperboundation then it takes time to change its preference from upper to lower to then we had to use environment variable by 'extern int EF_PROTECT_BELOW' to change its priority on a instant.

It is because it is configured for one time failed

Electric fens is configured to report heap memory violation error for upper ~~or~~ lower boundary region but not at a time.

Export 'EF_PROTECT_BELOW' - configuring electric fence to report lower boundary memory violation.

GDB fails to fetch memory violation at the, thats why we are using electric fence library.

(Makefile ≡ MF)

23/3/22

Make file

{ Set of code which are well organised.

It is called as program building tools in Linux & Unix OS. It is a set of commands similar to terminal commands but the difference is makefile will have an organised text instruction. Make will have variable, single target or multiple target.

↳ Kernel build makefile :- (/lib/modul/< * >/ build/arch/x86) (Linux OS version) (makefile)

↳ Processor Arch Makefile :- (lib/modules/kernel version)

↳ Subsystem level

v.s makefile → Execute appl^n ⇒ find appl^n build

↳ Linux kernel OS.

⇒ Kernel build MF :- Entire linux kernel OS is build into MF called as 'kernel build MF' located at "/lib/modules/ irrespective kernel version/ build/makefile".

DATE:

Rules to make MF :-

-> Makefile should start with 'M' without any extension.
-> All the sourcefile & MF should be in current directory.
-> MF start with target & name & ':'
-> Next instruction should start with tap.
-> 'make' is a linux utility tool used to generate application executables.
-> 'Make' tool by default comes with linux distribution.
-> 'make' will reduce the workload of recompilation
-> 'make' will execute makefile of current directory & jumps to makefile & start executing the instructions of makefile.
- gcc is indirectly called MF, intern develops execution application.

The primary objective of make tool is to breakdown large project source code into smaller piece & assess the smaller piece of code whether they required recompilation or not.

MF will contain make variables (cc) & may contain a single target & multiple target.

types of target.

PAGE NO.:
DATE:

Target is a filename that is ge by makefile program.
A target can be application exec as a target can be an object file. Target can be action such as clean

=> appln : main.o sub.o mean.o
(Sub targets) :- In this MF variable multiple targets. Here, create object files as targets. If I want recompile any individual file this will reduce workload of recompil all files because make is going to recompile only modified files.

=> System call (fork Execl("ls -l") wait).
parent child

System system call is implementatio of fork, exec & wait.

```
#include <stdlib.h>
int system (const char * command);
```
Shell command

The return value system system call
0 & 1 ~~values~~ will talk about status of
shell.
0 :- 0 shell is not available because
it is in executing command
1 :- when NULL ~~prod~~ given as
command the return value is
1 (non-zero value) indicates shell
is available.

=> Atomic Locking method

atomic_t → tiny locks
data type → Used for integer variable

data structure → Not used to safe guard
{ members } resource like printer, memory
}

Remember declare a variable of type
atomic_t kernel provides a set of
function to safeguard the declared
integer variable.
In a process
lock
x++; equivalent to atomic_incr (x);
unlock

24/12/22

pthread_exit() will terminate the current task & allows the pending task to get execute.

pthread condition variable :-

→ condition
→ wait
→ Signal.

condition is used only in mutex

pthread_cond_t my cond;

int pthread_cond_wait (pthread_cond_t *pts, pthread_mutex_t *int)

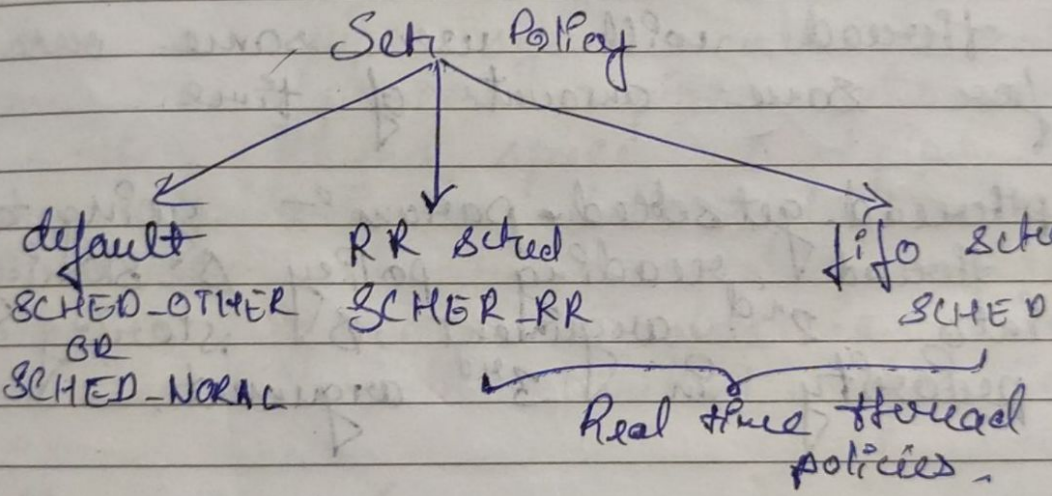
~~event to happen~~ pthread is ~~has~~ acquiring a mutex lock & waiting event to happen. The moment ~~process~~ thread execute 'pthread_cond_wait' call thread is unlocking mutex lock automatically & places itself in the wait queue of condition variable

pthread_cond_signal (condt)
↳ It is used to wake up particular thread whose name is passed inside the parenthesis.

pthread_cond_broadcast(pthread_cond_t *tp)

pthread_cond_signal will read the condition variable argument if any thread waiting on the condition then it will send a wakeup signal to thread.

pthread_cond_broad :- if a thread execute this function reads pore the condition argument & wakeup all the threads which are sleeping on condition variable.



Linux threads will have default scheduling policy known as SCHED_OTHER. In this case priority is dynamic that by system be changed based on thread behaviour.

25/3/22

VIJAYANT
PAGE NO.:
DATE:

Inter process Communication (IPC):—

Http client process
Http server process

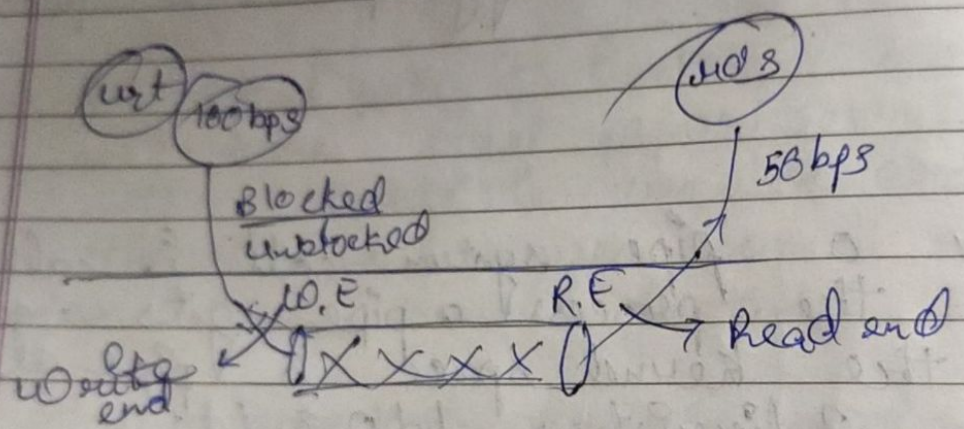
Each machine (node) in the network is identified ~~Every~~ by unique 32 bit IP address & is also used for connect & communicate & 16 bit port address is used to identify process in particular node.

When a pipe system call is called from the process a pipe gets created in the kernel space & pipe also return 2 discriptors fd0 & fd1.
fd0 ~~also~~ always associated with readent of file & fd1 is always associated with writent of file

Creating a pipe.
→ int a[2];
→ int read_fd
→ int write_fd;
→ pipe ();
→ data to write into fd.

→ When pipe is created
When parent is calling fork child is created.
→ Parent copy ~~data~~ fd to the child process (child inherit a pipe file from parent). Thus a pipe is limited to parent & child.

Pipe capacity is limited



→ Writer process writing if a writer at higher speed then the receiver process which is consuming data at slower rate. After sometime pipe gets full & if a writer wants to write the data it cannot write eventhough, writer has data
as a result the writer process get blocked until some space or room created inside (child reading).

28/3/2022

2 code

VIJAYANT
PAGE NO.:
DATE:

Examining C source code

C scope :- It is a linux utility tool used for software development process to examine the C source code. We can examine symbols (variable, macros, function) of source code.

To check what values they are initialised & where they are used. Using C scope tool u can also verify specific funct^n calling what all other funct^n & also check specific funct^n called by what other function. Can check & change the strings. Find specific file, check for # including header files, also use an assignment variable option to change the variable value.

ctags --list-languages :- Show all the programming languages supported by this tool ctags. Index for name found in source file & header

name → name, Pathname, line (where the obj (name) is in source file)

ctags → locators → Name / tag → Name, location, line, [After this generate tags]

ctags => linux utility tool. It generates a tage file i.e., a index file for the names found in source code & header file. Index of object refers to name, pathname, the line where is used in source code. ctags provides a quick reference to source code. Eg:- need to find defination of function 'x' then use ctag generate its history where it is used.

ctags uses locators & locators will locate object & pathname of object & line in the source code where they are used. & out of these it is going to generate tag file for quick reference.

Crit project

- Working
  - ~~8 diff changed files~~
  - 8 diff files
  - modified/changed files
- Staging
  - Staged files
  - marked as ready to move next level
- Repository
  - Local repo permanent stored
  - Safely stored in stored

Crit CONFIG Variables :-

$ git config --global user.name " "

configuration variable used to set or get git configuration

'--global' :- get information & store in local machine.

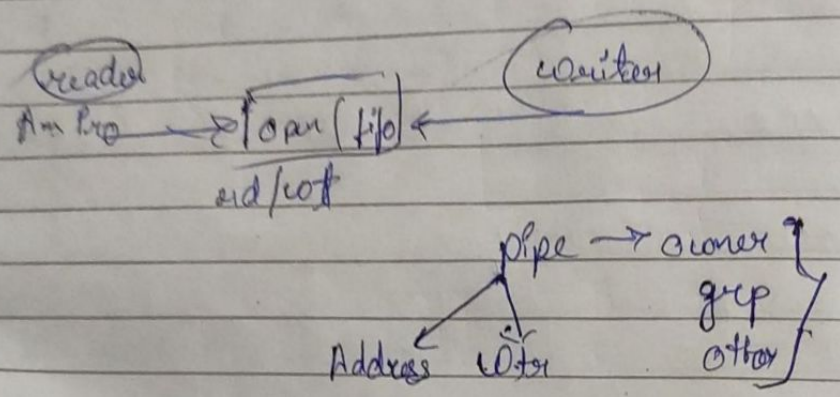
git config --global user.email " "

<sys/type.h>, <sys/stat.h>

int mkfifo (const char *pathname, mode_t mode)

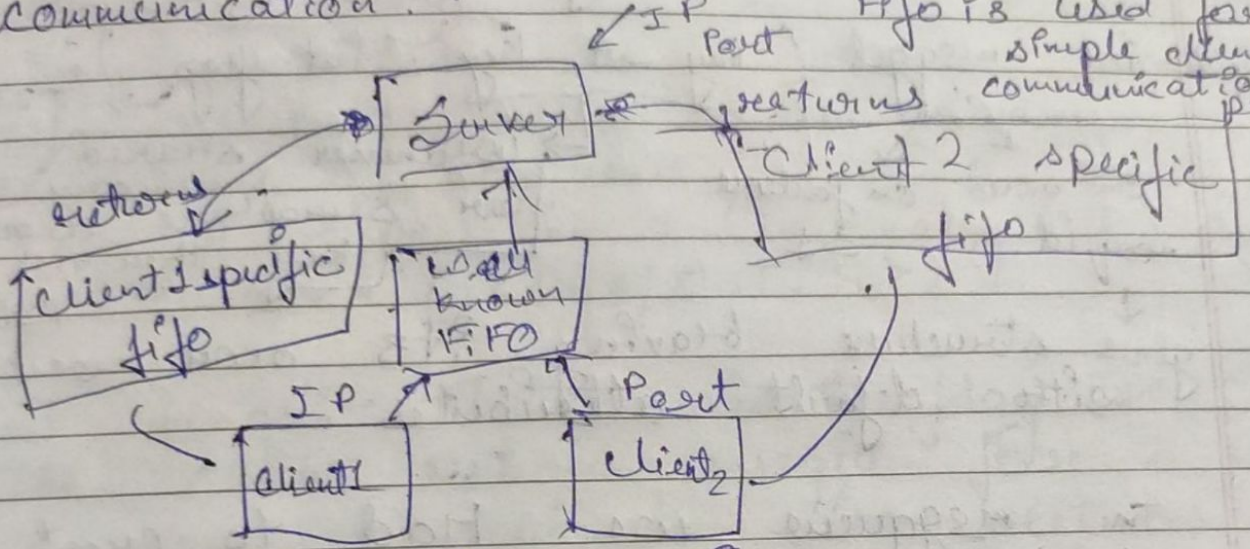
0 1

→ fifo
- → Enter O.S file system
- → special file attribute.



The mount process execute mkfifo function a fifo file gets created & makes an entry into linux kernel O.S file system. A fifo file is a special file which has all the attributes, Any process can open it for reading & writing like any other normal regular file. Ensure that fifo is opened from both the end simultaneously.

If your process is opening a fifo for reading operation has to wait for other side process to open for write operation & vice-versa.
A fifo can have multiple readers & writers & support connection oriented communication.



Message Queue

'ipcs -q' = no. of msg que in system

By default 0 msg queue in system initially it is showing only the structure.

Each & every msg queue is allocated with structure name ~~'struct msqid~~ 'struct msqid_ds'.
↳ { IPC_perm ;
no. of msgs in queue ;
max. no. of bytes in queue ;
pid of last msg sent/recived ;

Abhishek KUMAR

time last msg send/received;
msg - changetime;
};

⇒ # Include < sys/msg.h >

msgid = int msgget (key_t key, int flags);

(This is agreed by both sender/receiver)

IPC_CREAT

(Unnamed shared memory or semaphore request key then this is required of type int)

On success msg_id

On failure → -1

msg_id → gives structure having its own members with default attributes.

In msgqueue we had to create first because we don't have any default msgqueue in our system.

IPC_CREAT :- Returning a message qid for an existing msg queue or newly created ~~when~~ msgid when it is freshly created.

The ~~per~~ moment process calls a msgget call a new msg queue is created in K.S with a new & unique msqid. For newly created msqueue a structure msqid_ds is allocated with default initial values. '~~key_t key~~':-

key_t key:- Whenever a process requesting for resources such as shared memory, semaphore & msqueue. O.S needs a key of type key_t (internally int) defined in header (<sys/stat.h>)

Int flags:- Talks about flag IPC_CREAT. IPC_CREAT returns msqid for a new msqueue created. It also returns msqid for an existing msqueue of same key.

msgsnd(int msqid, const void *ptr, size_t nbytes, int flags);

The moment process execute msgsnd pointer pointing to msg is send to a msg queue whose id provided by ~~msqid~~ msqid. On success msg is sent.

ipcrm -q 'Id' => For removing the particular msqueue

ipcrm -m 'Id'
ipcrm -M 'key'

We can do particular operation by passing cmd in cmd line without putting cmd in cmd by simply putting cmd in cmd & it works

=> msgctl (msqid, int cmd, struct msqid_ds *buf);

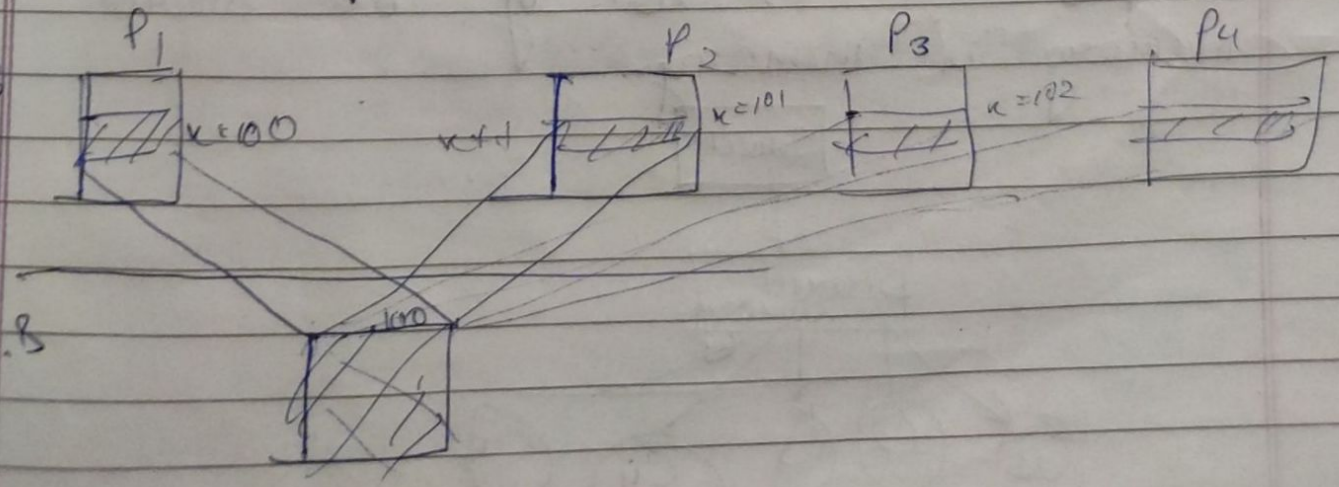
buf

Struct msqid_ds	IPC_STAT IPC_SET IPC_RMID	User id change grp id no msg is
Permission Size message		

Simply removed

Shared Memory

Fastest of all IPC communication.

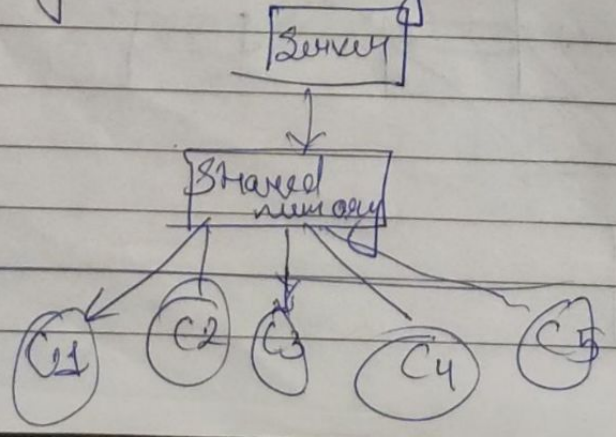


One of the IPC technique. Shared memory allows 2 or more process to access a given region of shared memory & it is fastest IPC technique because need not to jump from 1 memory location to other memory location for data read & write operation between reader & writer process or client & server.

In between client & server the only thing we had to manage synchronized access by applying the semaphore locking technique

Shared memory can be used between server & client only trick is synchronizing access of given shared memory region with semaphore technique.

'ipcs - m' => list of all shared memory in memory.



30/3/22

VIJAYANT
PAGE NO.:
DATE:

MCU

Introduction to Microcontroller

CPU doesn't have RAM, ROM, I/O ~~internal~~ in internal parts that's why we had to connect externally on it.

→ Set Crystal frequency so that performance is high but the power consumption reduces.

16 bits address bus means it can access maximum of 2^{16} addresses

DIP => 2 sides pin (ARM processor)
QIP => 4 sides pin.

→ (11.059MHz, 5Volt).

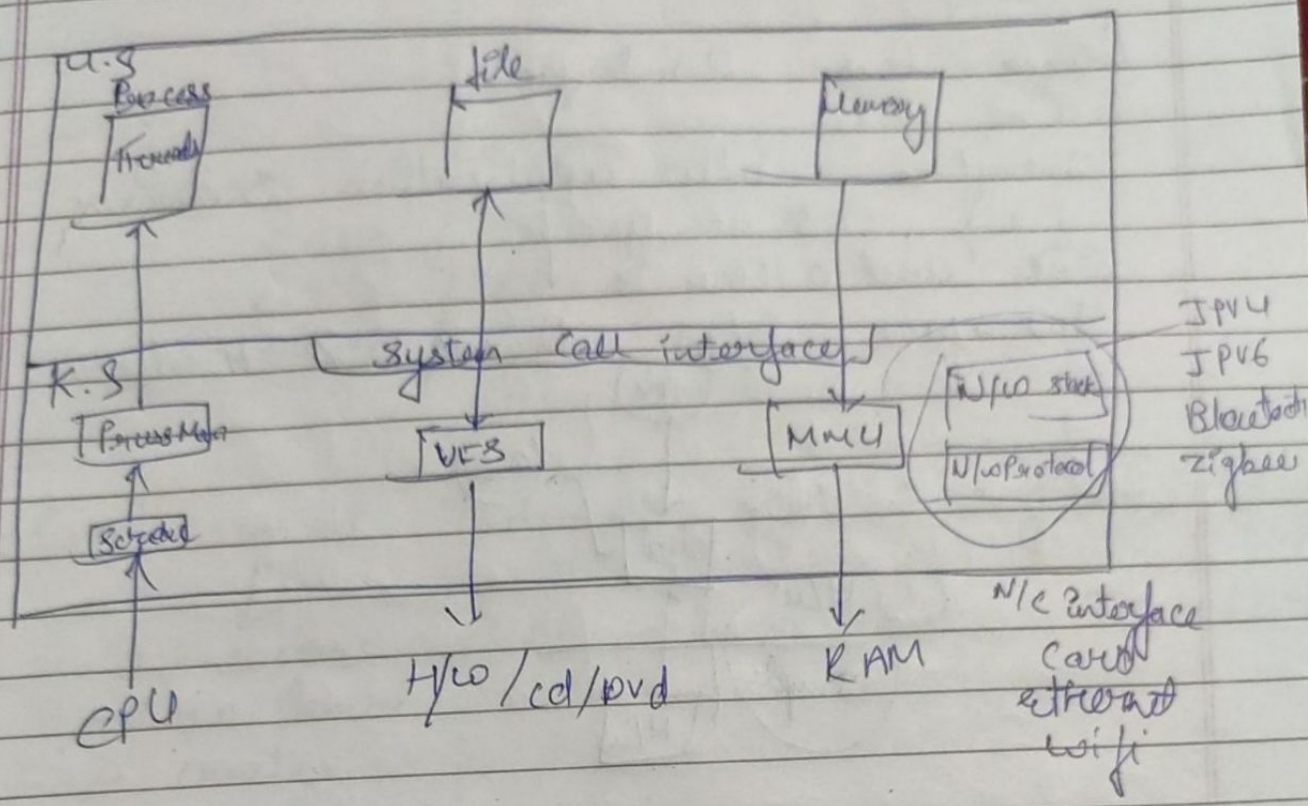
→ Crystal oscillator work as a clock for controller & is also named 'Heart of CPU'.

$\boxed{0000H}$ → When reset the controller it reinitialize the variable to its initial state.

ALE = 0 => address
ALE = 1 => data.

31/3/2

Linux kernel Architecture

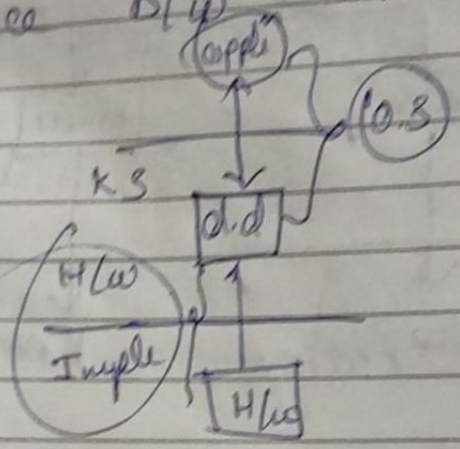


System calls are kernel space services not user space services. They are meant to invoke K.S. Each & every system call has to pass thru system call interface & then invoke a appropriate K.S.

ce driver from kernel P.O.V: Device driver is a piece of software that communicates & manage the hardware & bring the functionality to the end device to the user.
In linux O.S device driver resides in kernel space because only in the

kernel space driver has privileges talk to the Hardware

Driver has 2 interfaces :-
One is
Interface b/w application & driver
i.e, O.S specific
and other is
Interface b/w Driver & Hardware



Semaphores

One of the IPC technique. It deals with array of semaphore. Array of semaphore is a bit complex issue but in large application software process needs to work on lots of resources need more protection having array of semaphore is big advantage semget(), semop(), semctl() are designed to work on array of semaphore

semaphore( ) ⟶ array of semaphore
Complex

⇒ <sys / types.H>
<sys / sem.H>
<sys / Ipc.H>

semid = int semget (key_t key, int no. of sem, int sem flag);
(1) IPC_CREAT

On success return semid & creates a semaphore

On failure '-1'

⇒ int semop (int semid, struct sembuf *ptr, size_t no. of structures);

struct sembuf {
sem_no. — 0 (because only 1 semaphore we are using)
sem_op — -1 → Waiting for semaphore
+1 → Semaphore available
(Value of semaphore)
sem_flg → SEM_UNDO

It is used to change the value of semaphore.

01/09 Linux

Signals :-

They are software interrupts they notify process about an event occurred & they asynchronous in nature they can happen at any time CPU kernel or any software running on CPU can trigger signals to the process

A process with enough permission can send a signals to other process & a process can send a signal to itself

Terminal generated signals :-

~~The~~ Under terminal signal generated by certain keys

Ctrl + C interupt delevered by kernel → Signal SIGINT

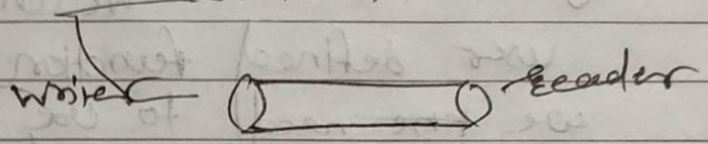
Ctrl + Z Stop the execution + SIGTSTP → to process then process will move to wait Queue

kill function can send signal to process only if we are owner of the process

kill command

Software Releated Signals:-

SIGURG1
SIGURG2



alarm (x) i.e SIGALRM

Signal to notify process some event occurred

Inter

A signal between signal generation & delivery is known as pending signal

~~then~~ In response to signal process has to do some action process can run some default action process can ignore signal process can use defined function