* Disadvantages of fork():

1) fork(), to maintain a process allocates lots of resources and very expensive in terms of CPU, memory, IO resource.

* Threads:-

→ Creating, maintaining, destroying threads are cheaper than compare to process.

→ Threads called as LWP (light weight Proces) need only less resource.

⇒ Thread is parallel context of Execution
└ set of instructions that are executing to perform specific task.

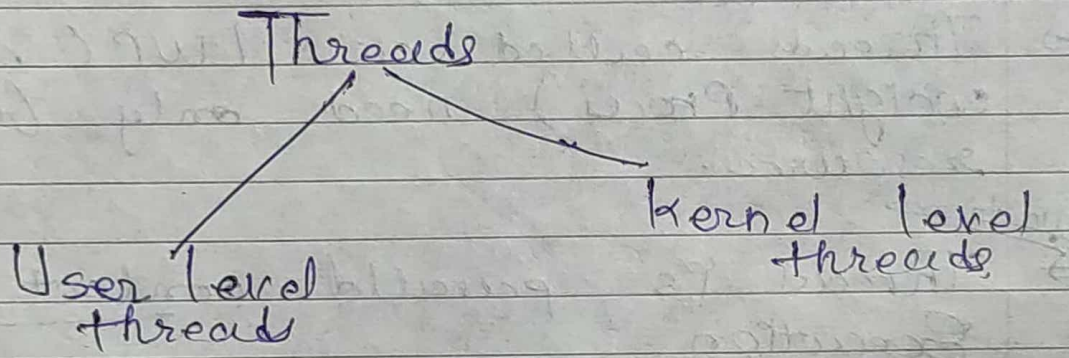
* Multithreading:-

→ Multiple, Parallel context of Execution.
↳ Sets of instructions that are executing simultaneously / concurrently

* Thread Resources

→ ① Own stack region
→ ② Own register
③ Thread specific data
④ Own scheduling policy & priority
⑤ Own signals sig block & sig mask

* Linux uses a thread model called as ~~1:1~~ thread model.
→ One-to-One



* Advantages

→ They do not carry their own address space
→ All the threads share the address space, Comm. b/w process and space by default.

* Disadvantage

→ At least one thread having a bug then entire thread & and application will scratch.

* Linux (thread)

Native

clone

(thd's)

using

Windows (thd)

Can't understand Linux clone

* IEEE POSIX :-

Portable operating System Interface

IEEE

POSIX

↓

set of functionalities

* POSIX :-

POSIX

↓

Posix libraries

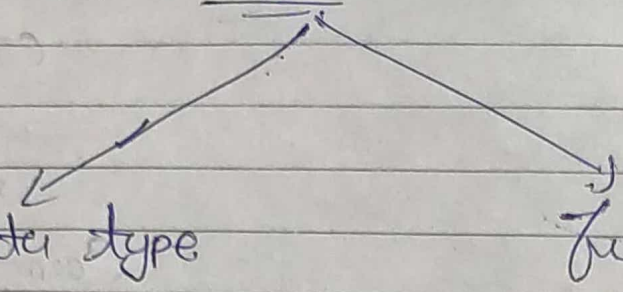
↓

defining - design of thread

↓

POSIX thread.

Posix thread Creation



data type

pthread_xxx_t

pthread_t

function

pthread_xxx();

thread obj

```
# include <pthread.h>
```

Int pthread_create (pthread_t, *ptr, *attr, (void) (* fun ptr) (void) void *arg);

→ Thread on a ~~proces~~ ~~return~~ execute doesn't return value

* Pthread_join (tid, NULL);

1) Join given thread id to the main process execution

2) forces process to wait for completion of thread id

two threads

```
#include <pthread.h>

void* start_one (void *arg)
{
    int i;
    for (i=0; i<1000; i++)
        printf("End of con1 \n");

Void start_one (void *arg)
{
    ..
    ..
}

main () {
    pthread_t p1, p2;
    getchar();
    pthread_create (&p1, NULL, startone,
                    NULL);
    ..
    getchar ();
    printf ("end of main thread
                \n");

}
```

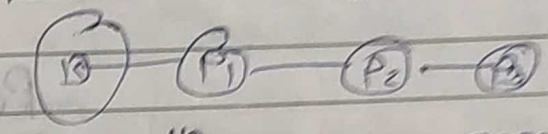
* Semaphore:

1) Declaration: (semaphore variable)
   sem_t my_sem;

2) Initializing the semaphore obj.
   sem_init (&my_sem, 0, 1,)

3) sem_wait (sem_t, ptr)

4) sem_post (sem_t, * ptr);

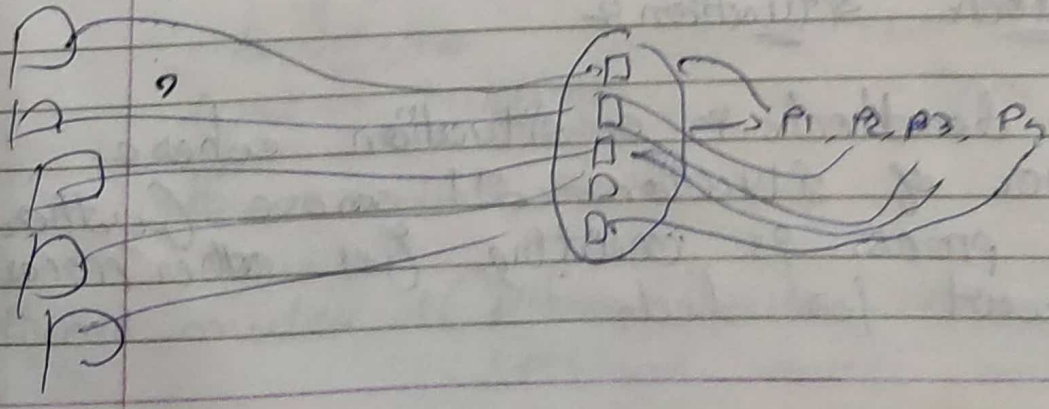


⇒

* Process operation on semaphores-
  (1) Process take the token from s/e
  (2) Use the resources.
  (3) Give back the token to s/e.

* Types of Semaphores-
  1) Binary Semaphore

A semaphore whose values changes b/w '0' and '1' in process called binary semaphore.



* Counting Semaphore:-
→ A semaphore taken value > 1 is it is a counting semaphore.

* P. threads and spin locks :-

① Declaring.
pthread_spinlock_t my_spin;

② Initializing
pthread_spin_lock_init
(pthread_spin_lock_t *lock, int shad

Public Priva

③ pthread_spin_lock
(pthread_spin_lock_t *lo

* Interrupt :-

* Context - Switching:
→ Process of switching from one executing process to another task.

* Deadlock situation :-

Deadlock is a situation where neither of situation will move further
→ each process is waiting for other process to wait for lock

PAGE NO.
DATE

✱ Mutex :-

→ Similar to semaphore, with the usage count of 1.
(binary s/p)

→ Process / thread acquired mutex lock, busy to unlock

→ process can not exit, while holding a lock

✱ Pthread Mutexes :-

① Declaration

pthread_mutex_t my_mutex;
has its own attr.

(pthread_mutex_attr_t)

② Initializing

Dynamic
advantage
during runtime you can change the properties of mutex

static
→ mutex attribute can't be changed they are fixed.

pthread_mutex_t (pthread_mutex *ptr, pthread_mutex_attr_t * attr)

pthread_mutex_t my_mutex =

PTHREAD_MUTEX_INITIALIZER;

⇒ pthread_mutex_lock(pthread_mutex_t *ptr)

pthread_mutex_trylock(pthread_mutex_t *ptr)

⇒ If your process fail to acquire a lock using pthread mutex lock, then particular mutex blocks the call this system.

⇒ pthread_detach / pthread_self

↓

call will push the given value id in the detached state.

15/03/2022

* Purpose of pthread_once is to schedule & initialize code

⇒ pthread library provides a macro PTHREAD_ONCE_INIT

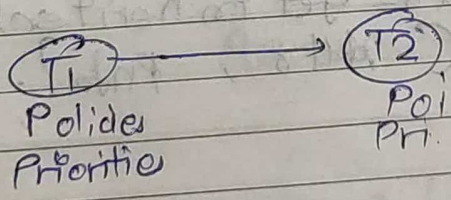
⇒ pthread_once executed checking for the arg. initialized or not. argument whether

If initialized will. schedule and execute the initialization code.

⇒ If not initialized will never schedule a function.

⇒ Pthread_once schedule and initialize a function of type which takes no arguments and no return

* pthread Scheduler Attributes



1) Inherit scheduler member
2) Priorities

→ When thread create new thread then newly created thread inherit some properties according to POSIX library.

shared region

→ There is a one memb. called as Inherit scheduler member that specifies whether a thread is inheritting the schedulers policies and prio. from its parent or managing on its own

Inherit Scheduler Member

PTHREAD_INHERIT_SCHED

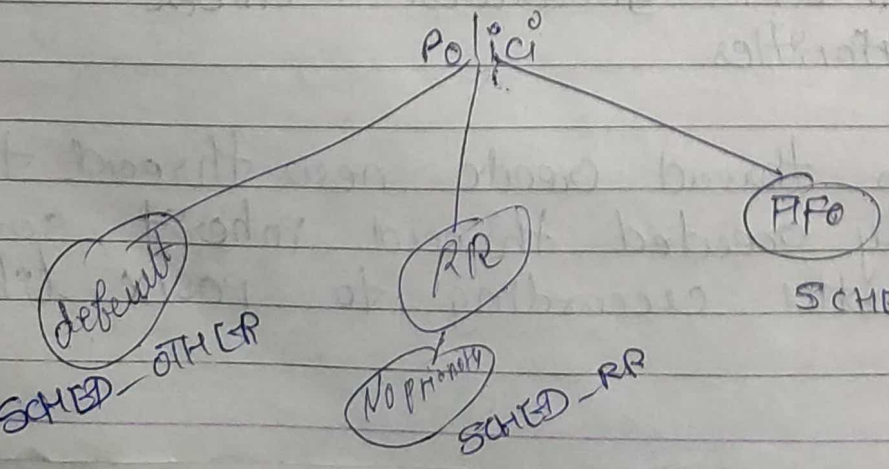
PTHREAD_EXPLICIT_SCHED

→ If a thread is having inherit SCHED then thread is having parent policies & priorities

→ If not it will not inherit the priorities.

* pthread_attr_getinheritsched (&attr, &no);

* Linux Scheduling priorities and policies :-

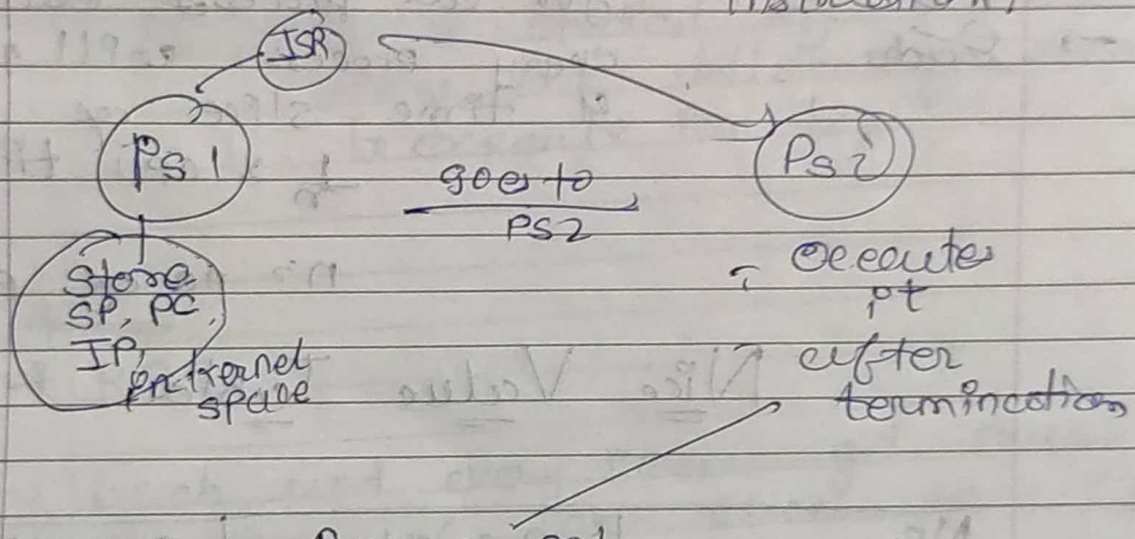


→ If policy default, all threads have same priority, all threads have same CPU slice time sharing.

→ RR - No issue of priority. All threads will get time slice in reverse order

* Context Switching :-

Stack pointer
Instruction pointer (points to next instruction)



* Preempting Scheduler

Kernel forcibly transferring the CPU from process without requesting the process

* Process Never leave a CPU on a interruption in Non-preempting scheduler if task get finished then server / Timer

(CFS)

→ Linux employees complete ~~for~~ Fair Scheduling

→ Each and every process proportion of CPU based on factor Nice value.

- gives weightages to the process with respect to CPU slice time.
- Nice values used for prioritise the process.
- If Nice value is same for all processes each & every process will get a time slice of

$\frac{1}{n} \times$ CPU time

n = no of process

→ ~~Each~~

Nice Value Table

Nice value	Priority	CPU slice time CPU slice
-20	highest	85 m sec
0	default	10 m sec
+19	lowest	5 m sec

Why a complete turn Scheduling?

→ Even lowest priority get a time slice.

* taskset command will assign a cpu core to process.

→ Utility tool by default comes with linux distribution.

taskset -c 1 ./nice &

Changing the CPU core

& - is a special bg symbol.

- forces the following command to execute in bg.

~~* top~~

* RR :-

→ Each and every process get equal proportion of CPU slice time in circular order.

⇒ If the process had timed given time slice CPU will move to next process

→ Each and every process use CPU with same amount of time.

→ No issue of priorities.

→ Implements starvation free process.

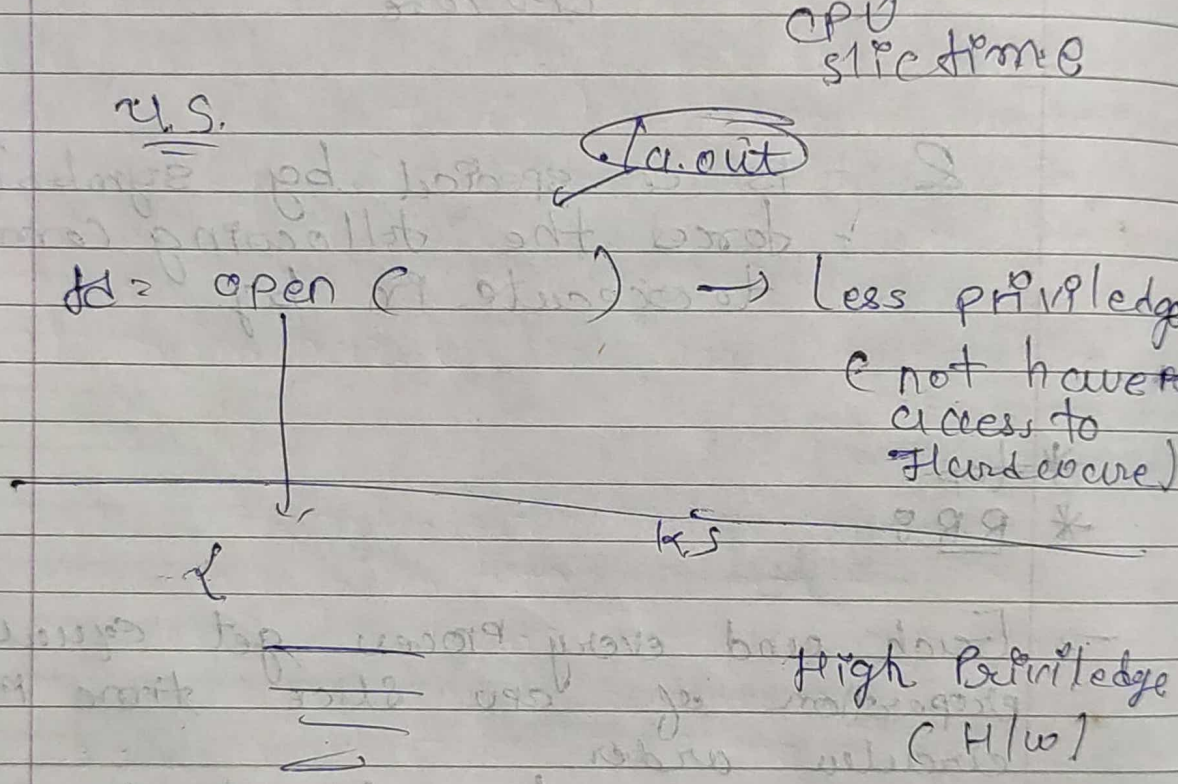
16/03/2022

Memory

* Process context, kernel context, Interrupt context

→ at any given point of time kernel will be executing either process, kernel, Interrupt context.

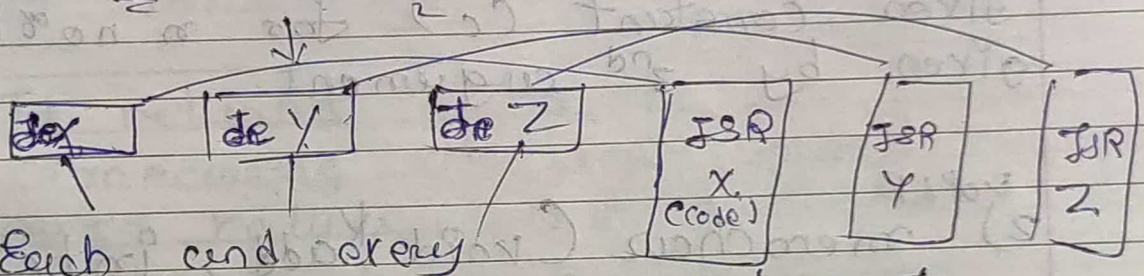
Context :- Piece of code
Instructions that are executing



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

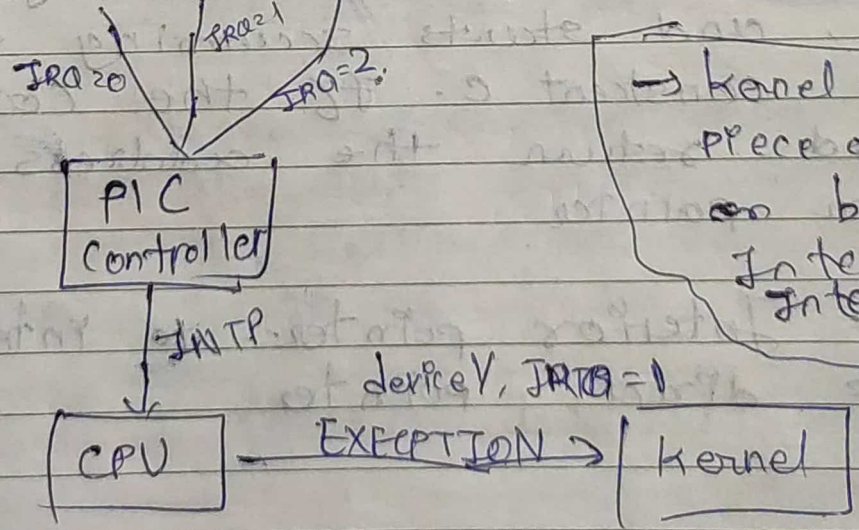
→ Kernel executing a piece of software (kernel service) on behalf of another kernel service.

* Interrupt context



→ Each and every device can generate interrupt.

→ Each and every device connected to pic controller through a physical core called as interrupt lines (IRQ)



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

* MMU (Memory Management Unit)

* Memory Manipulation calls -

going to particular address and changing data

kernel datatype int

void *
1) memset (void * add, int c, size_t nbytes);

when process executes memset - jumps to addr location provided by pointer argument and start set the data with given constant (c) for n no of bytes given by 3rd argument.

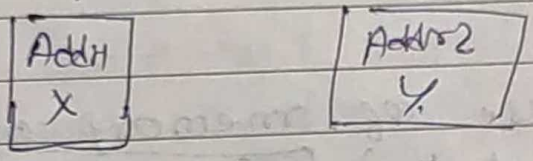
void *
2) memchar (void *add, int c, size_t nbytes);

The moment process executes memchar function jumps to the address location provided by pointer argu, and starts scanning for the given constant c. If the constant found, return the address to other pointer

→ ret : Interior pointer to interior address
Str : direct pointer

Pnt

3) memcmp
(void * addr1, void * addr2, size_t
n bytes)



- The moment process execute memcmp it jumps to two addr location, and start comparing the addresses byte by byte until it matching data.

→ It returns +1, 0, -1,
→ comparing the data based on ASCII values, and get the difference of ASCII.

4) void * memmove (void * dest,
void * src, size_t);

→ Process executes memmove, copies the data from source buffer to destination buff. for given n no of bytes

→ (If there is data, it will overwritten)

→ void * return address of destination

5) memcpy

* → slow, Reliable, generate guaranteed data

→ In case of memory overlapping (src, dest) memmove provides Reliable

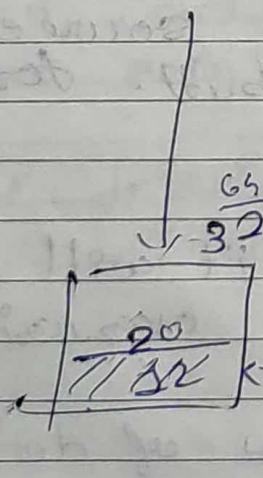
→ memcpy there is no temp. but.

* Alloca

void *calloca (size_t, nbytes);

The moment process executes a calloca memory call, allocates a memory from stack segment (Heap is full). returns pointer to the allocated region on success.

① malloc (20); malloc



Adv

- ① Minimum memory wastage - operation is faster to compare of malloc

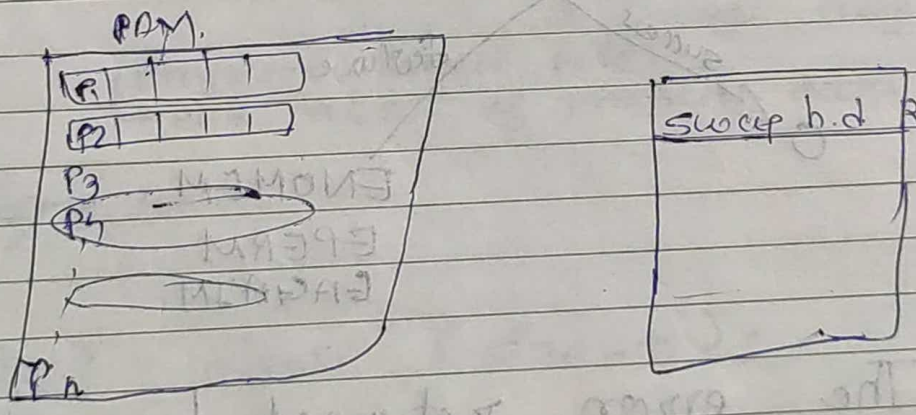
② alloc doesn't maintain pool of fixed block sizes and hence no memory fragmentation

③ Doesn't need to call free(). the memory automatically deallocated when function end.

Disadv

Repetitive usage of alloc & leads to stack overflow.

★ Swapping process :-



→ When RAM is running out to memory and OS wants to launch new memory then OS looks for inactive process in RAM and push them into Swap storage device (Our laptop. This called as swap out process.

→ When newly process done their job kernel get back the swap partition to RAM and is called swap in process.

* Memory locks:-

→ There could be a logn program and coe don't want kernel to swap my logn proces to my RAM.

→ Then we can apply memory locks

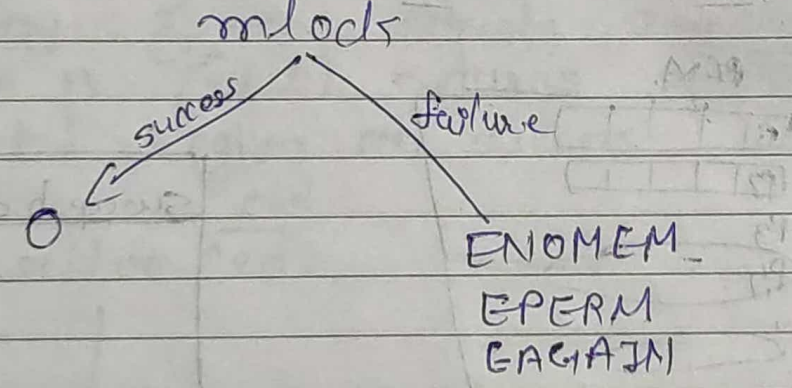
→ Types:-

int

1) mlock (void* addr, size_t);

2) int mlockall (int flags);

→

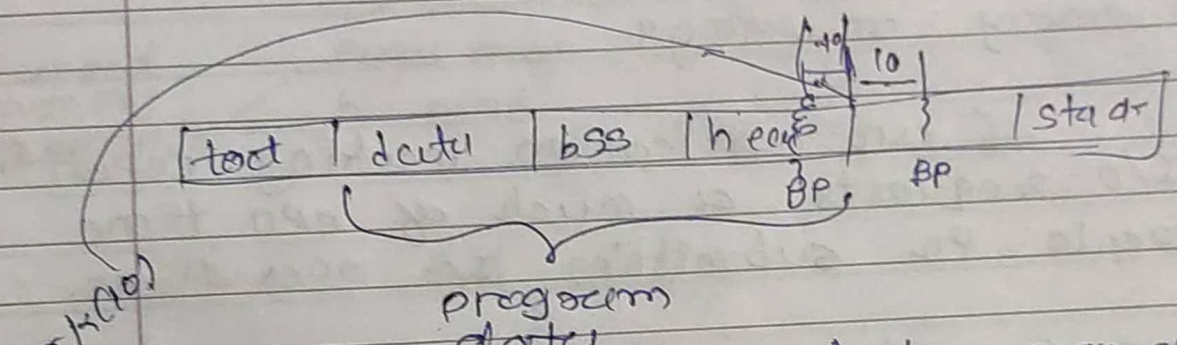


→ The error returned by mlock while trying to apply a memory lock

* EPERM :-
When no privileges to process, return a error called as EPERM.

* EAGAIN :-
If PS failed to supply memory locks for the specified address.

* MCL_CURRENT
↳ current PAS is locked.



* Sbrk :-

void * sbrk (size_t);

* brk :-

int brk(void * end_addr);

break point.

* my memcpy (void * dest, void * src, size_t n bytes)

* Memory mapping :-

Issue with I/O 0.5 M.S

If an application making repetitive I/O requests a much of CPU time spents in submitting I/O operations.

bcz mmap() is a POSIX memory function that maps given kernel file region or a device region or some random kernel memory into the calling process.

* mmap Syntax :-

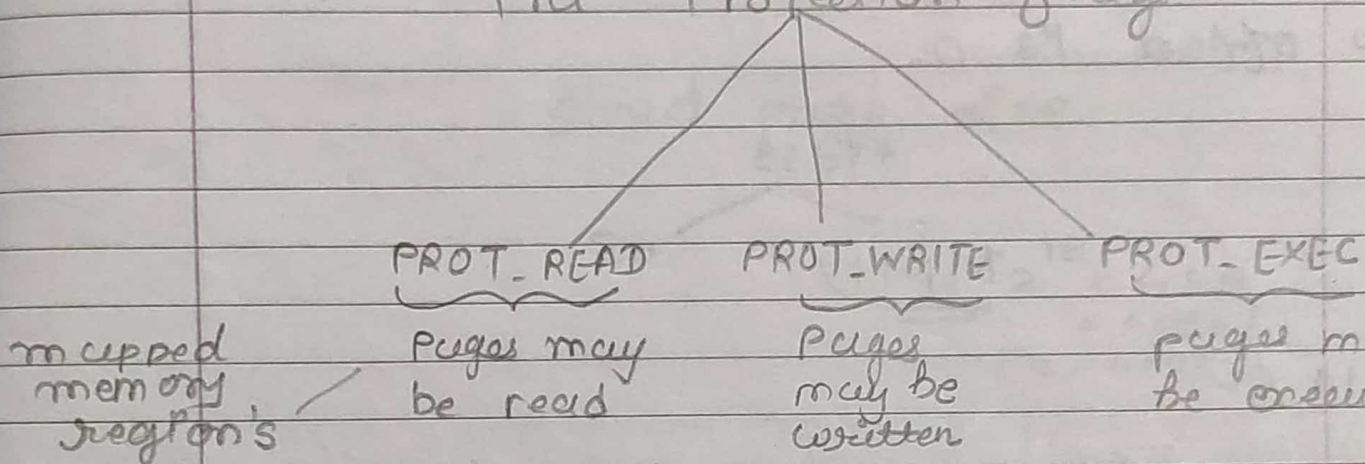
#include <sys/mman.h>

void * mmap (void * addr, size_t n bytes, int protection flags, int flags, int fd, off_t offset);

→ void *addr - coherent we have to map memory

→ size_t n bytes - No of bytes you want to map.

→ int Protection flag



→ int flags

↓

MAP_LOCKED;
MAP_SHARRED;
MAP_PRIVATE;
MAP_ANONYMOUS;
↳ (doesn't belong to any file)
& and fifth arg should -1

↳ MAP_LOCKED = applying lock to shared region

↳ MAP_SHARRED = Same region is shared b/w n no. of processes

↳ MAP_PRIVATE = create a private copy and changes will not reflect to other processes with which having shared region

⇒ First arg = 0 ⇒ It is recommended to use 0 to indicate asking kernel to map out a free process address space

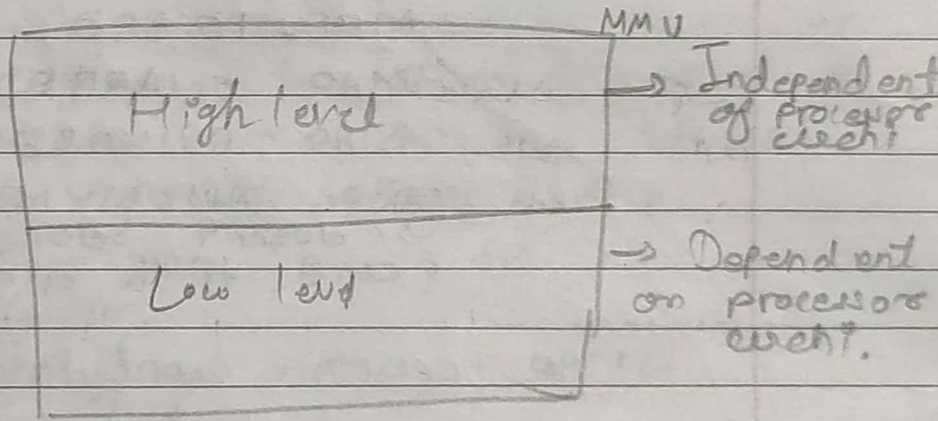
⇒ fd is -1 if file is not there

⇒ offset is 0.

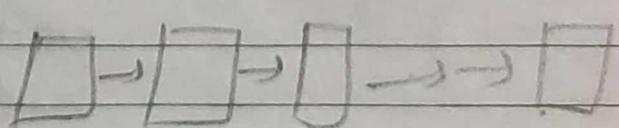
MMU

U.S.

K.S



→ When kernel booting starts, low level MMU also starts and creates lots of kernel memory data structure.

Ex
Struct page. 

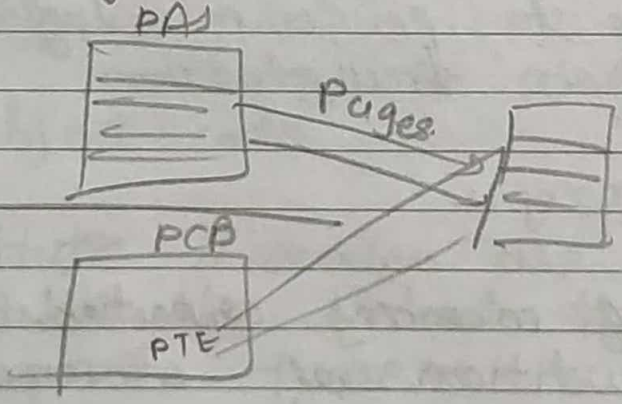
page size = 4kB
struct page size = 32 bit

→ Converting entire memory of RAM into list of struct pages.

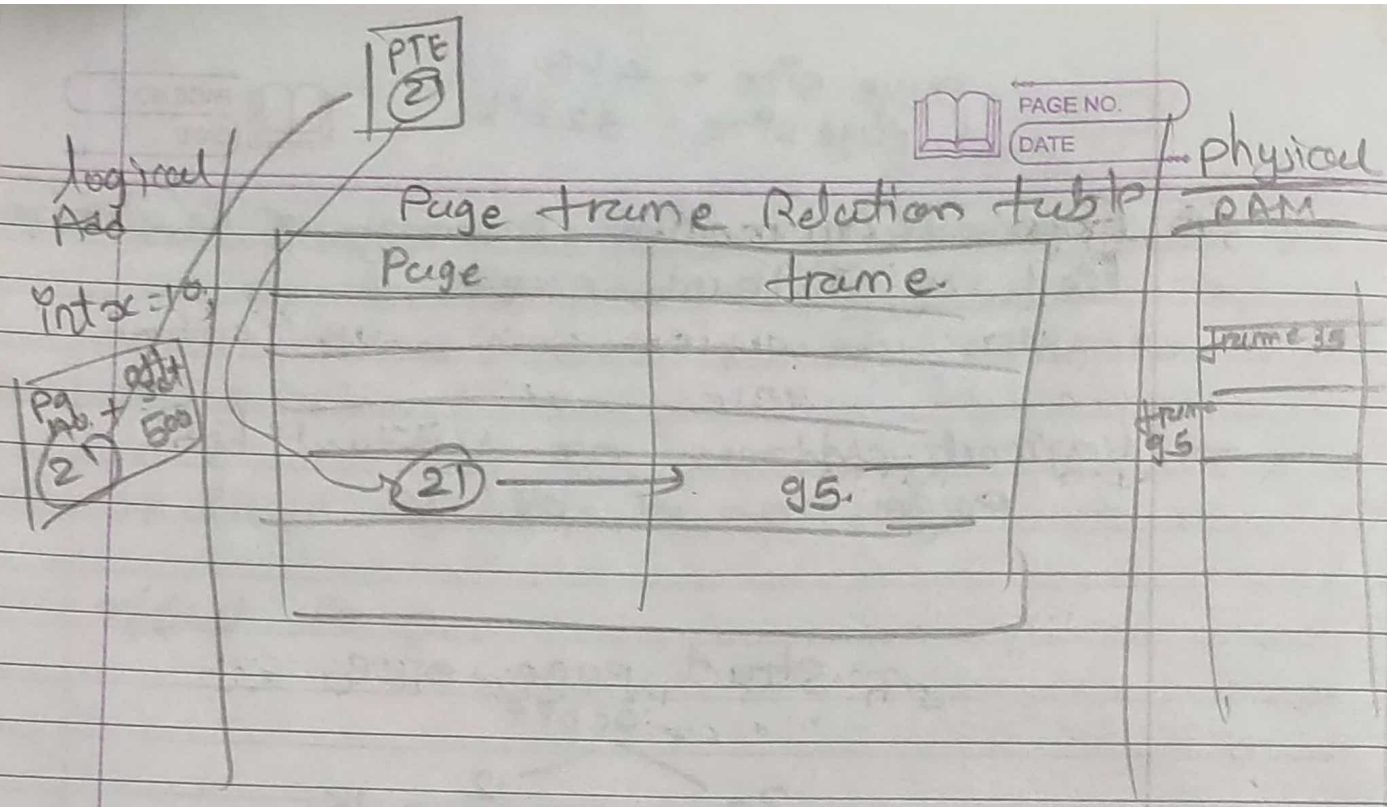
→ Logical address or Virtual add = page no. + offset

struct page size
32 bit
20 | 12
2^{12} = 4kB = 4096 [0-4095]

→ Each and every process maintain one page table entry PTE inside PCB of process.
→ If refrence to PTE lost, kernel will never no pts all process pages are in memory.
→ PTE identifies these particular belongs to x particular process



→ Inside kernel there is dynamic data structure called as page frame allocation table



→ Processor will take pg no and look for a matching frame no in page frame relation table. Now fetches respective frame no. to page no.

→ Now takes the pg no and adds to offset. Now one can get Physical Address of integer variable x.

→ For each and every I/O request operation processor have to perform logical to physical address translation.

* Virtual Memory

→ A section of memory created temporarily on scsi partition of storage device.

→ Linux uses RAM and the virtual memory and virtual addresses are assigned to processes.

* Free Command

provide amount of available virtual memory and used by your system.

* Vmstat :-
Complete virtual memory stattics reporter

* cat /proc/meminfo

b = Uninterruptable Process
r = runnable Process
si = swap in
so = swap out
bi = blocks received from blocked device
bo = blocks sent to blocked device
in = Interrupt
cs = Context switching

* Library :-
Library is a group of precompiled object code.

* Static libraries are one which are statically linked to prog at compile time.

* Dynamic libraries are which come dynamically linked to pro executable file at run time

→ In linux static libraries has extension .a

Steps

1) Generate object files.

2)

$ ar

ar is linux commandline tool called as archive. That is used to create static library

$ ar - replace. create, symbol

* -L. → linking the library lib_st.a from 'current directory

Compilation at 4 diff stages

Preprocessor — one.c → one.i
(remove comments, identifiers #, define and cat)

(.i)

Compiler

(.s)

Assembler

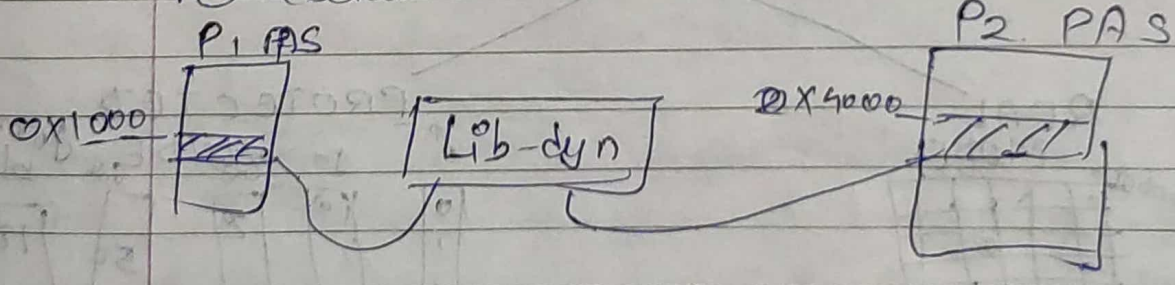
* -fpic.

Position Independent code

* Dynamic files have one more extension as .so. They are also called as shared library.

* fpic -

→ Dynamic libc uses a flag called as PIC. Dynamic libc should be relocatable



→ This means the dynamic lib should ready to load at any address of Address space

→ A process needs a dynamic library. In such a way that dynamic library appears as a part of executable program

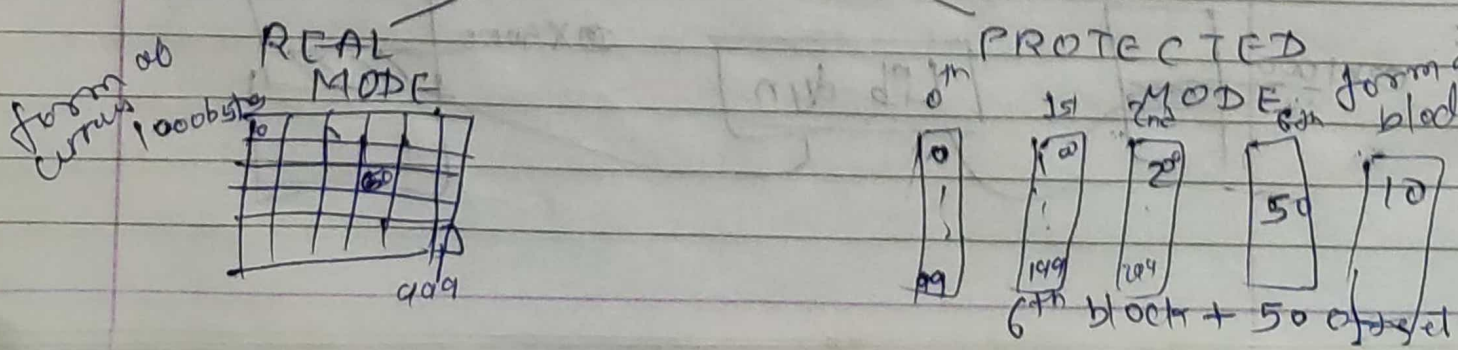
→ Later if another executable file requires dynamic lib, the instruction of a library should merge with memory of executable program.

→ -shared is used to generate a dynamic lib.

* ldd

→ command line tool ldd, will provide the dependencies required for the executable file

* PROCESSOR



→ It all depends on processor how it is looking at memory
→ based on that os will form policy to manage memory
→ Processor can look into memory in Real mode

→ Once the system is power on, during execution of bios code and boot loader code processor in Real mode, looking at memory ~~Real~~ Mode as array of byte.

→ Once the kernel boot up starts processor shifted to protected mode and looking memory as set of blocks. (Named list of pages). And this process called as Processor initialization

→ Kernel is creating some illusion and making CPU to look memory as set of blocks.

* Debugging / GDB (GNU Debugger)

→ Command line tool
→ Open source
→ Works with executable files
→ It is powerful debugging tool, support C, C++, python, Fortran, Pascal, photon..

Logical Address = page + offset
physical Address = frame + offset.

PAGE NO.
DATE

→ gdb applying -g (giving symbols)
symbols = variables, functions & expected file

1) List
It will give 10 lines of code

2) break :-
gdb use break command to stop command execution at user line of choice.

3) print, print locals.
will print the variables value

* Ways to run command line

1) $gdb --args ./Ex. arg1 arg2 arg3

2) $ gdb ./Ex
(gdb) r arg1 arg2 arg3

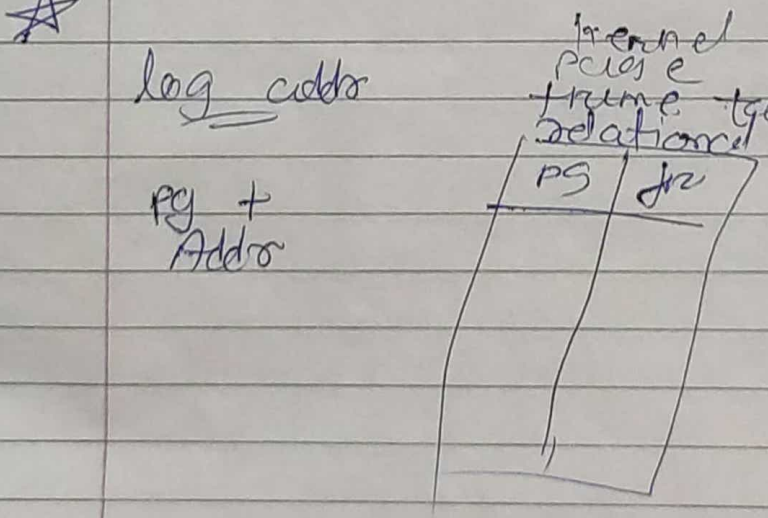
3) (gdb) set args

* Segmentation Fault
=
If you observe real time app. debug example most of the times bugs happen to be from users point of view.
ex accidently done dereferencing NULL pointer

→ Dereferencing uninitialized pointer
→ Dereferencing pointer which is free up
→ In accessing memory beyond the lower boundry region and beyond upper boundry region like int a[5];

a[-1] = X
a[6] = X.

* When a process created PAS is allocated and program is executed to use only the allocated region. When a program makes a violation the segmentation fault occure.



→ CPU will get the physical logical address, and will map to specific physical address with the help of kernel frame relational table.

→ Logical addresses are invalid memory addresses the CPU giving un fruits to get frame no.

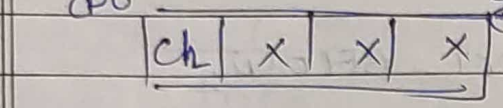
→ The MMU Processor throw Page fault error to the kernel kernel will check for the process segment area and then send a signal called SIGSEGV to respective process

→ On reception of signal the process will abort the operation with Segmentation fault.

22/03/2022

→ GDB command
* p type

* Memory Padding :-



if you try to fetch memory for char.

* __attribute__((aligned(2)));

Requesting attribute - aligned specifies please provide min alignment for my variables.

* gdb fails to ~~trap~~ trace out ~~down~~ heap memory violation.

* Electric fence :-
Electric fence is a library, which has got it's own implementation of malloc and calloc memory allocation calls.
→ Using Efence when requesting dynamic mem allocat. using malloc and calloc will allocate for specified amount of memory.
→ Standard c library malloc and calloc allocating more no. of bytes than actually requested for.

→ E-fence is configured to report heap memory violation either for upper or lower boundry, but not both at a time.

terminal

```
$ export EF_PROTECT_BELOW=1
```

and in code

```
extern int EF_PROTECT_BELOW;
```

→ Configuring e-fence to report lower boundry regions

→ GDB doesn't checks for memory violations

* Debugger tool

sudo apt-get install valgrind

→ Valgrind is a run time tool used to trace down heap memory violations it's also called as heap memory profiling tool.

→ Valgrind is standalone debugger used at runtime

→ When process makes use of standard library malloc and calloc calls valgrind reports why this process has failed. You can use it with or without debug symbol info

* Linux threads has its own default policies called as
SCHED_OTHERS
↳ Default scheduling policy priority is dynamic that can be changed by your system.

* FIFO (SCHED_FIFO)
↳ In this threads will have a fixed priorities ranging from 0 — 99.
1 — 99.
↓ lowest ↓ highest

↳ A thread with priority 1 in ~~stat~~ SCHED_FIFO policy will execute first as compare to default policy SCHED.
↳

* RR (Round Robin) Scheduling policy.
→ Each and every thread will have equal priorities & will execute in circular order.
→ Each thread will use resource for same no. of time.

pthread_getschedparam() function going to particular thread id and reading the policy & storing the same in second argument, and

Vidhya
PAGE NO.:
DATE:

* x/b byte
printing ASCII value present at that particular address.

* x/s byte
will print string until you reach at NULL character

* set
Instead of recompiling the code, changes can be made from gdb comm and to modify value of existing variable.

23/02/2022

* Makefiles :-
makefile

makefile is called as program building tool in Linux and operating sys.

→ It is set of commands similar to terminal commands. but the difference is it have organized text instruction.

→ makefiles have got variables, single, Mu target tu

→ One of is we have on system is kernel build makefile

Rules

↳ Makefile should start with M. without any extension. All the source file and makefile should be in one same dir.

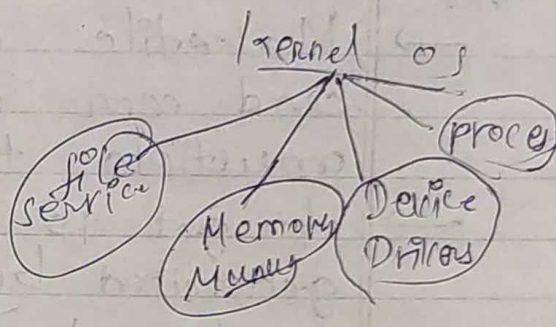
↳ Makefile starts with target name :

↳ next instru. should with tab space

K.S

Kernel build makefile

Processor Architecture makefile



→ Entire linux kernel operating system is built into a makefile called as kernel build makefile
/lib/modules/<kernel version>/build/makefile.

1) Write a makefile.

* apple

→ Make is a linux utility tool used to generate applications
→ Make tool by default comes with linux distribution
→ Make reduce work load of recompile to

Vidhya
PAGE NO.:
DATE:

→ Make file goes to current directory and starts execution of makefile.
→ gcc find directly called by makefile. In turn develop executable.
→ The primary objective of maketool is to break down large object source code into smaller piece and asses small piece of code whether they need recompilation or not.

→ Makefile contains make variable and can contain single target or multiple target.
→ target is the filename which is generated by makefile program.
→ a target can be either executable or target can be object file or it may be a action such as clean.

Makefile

Vidhya
PAGE NO.:
DATE:

(target) (sub target)
* cppln : main.o meen.o sub.o.
$(CC)

→ In this makefile variable and multiple target. Here created object files as a target
→ If I want to recompile any executable this make will reduce workload of recompiling all file, because make going to compile only modified file

System Calls

System Function

system (fork, Excel, exit)

```
#include <stdlib.h>

int system(const char *command)
```

(can be any shell command)

error case

- Error :- child not found
- Error :- too little argu (empty comm)

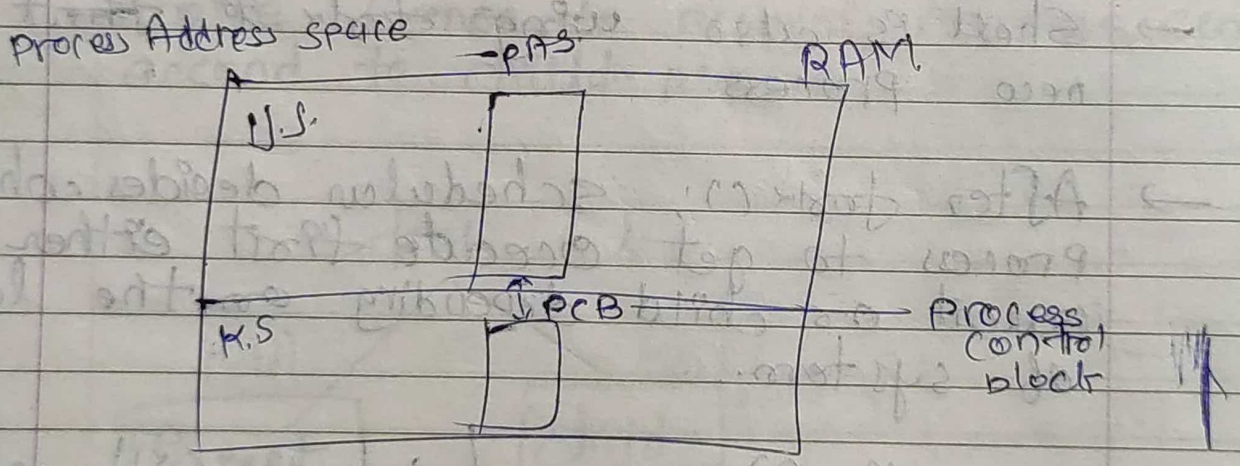
on success → return value of system call 0 and 1 will talk about the shell availability. When 0, shell is doing some command (shell is busy). When NULL given as command, return 1 means shell is available

→ Scheduler ~~ted~~ choose one best process based on priority (Ranking system) and get executed

→ wait queue L.L of process that have been suspended (waiting) due to some events (like scanf, while) (might be waiting for some data.)

→ RQ — Per CPU in machine | W.Q — Per resource in machine

depends on hardware



→ PCB contains all the process current PAS.

↔ PCB going to main in

pid,
PS - Process states,
PP PO's
PC Reg.
SP Reg.
IP reg.

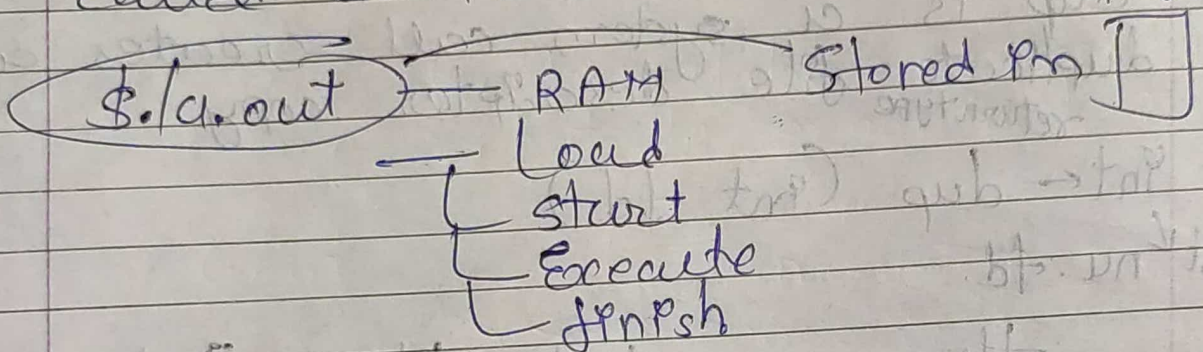
Memory
Threads

Kernel Services

1] file system
2] Process Management

Process Management

→ A program which is in execution called a process



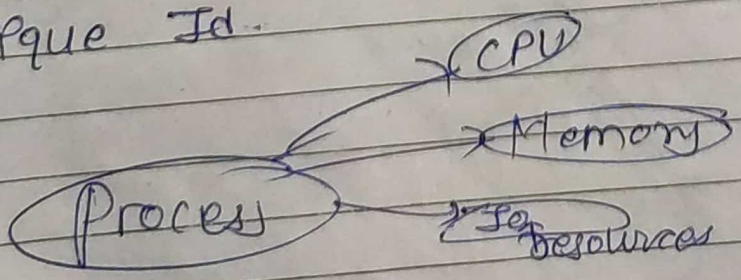
→ Linux can run multiple programs, multiple processes simultaneously / Concurrently.

→ In order to perform multitasking oper. at any given point of time ten of apps running on machine

Command $ PS - dsc

PS = Process state

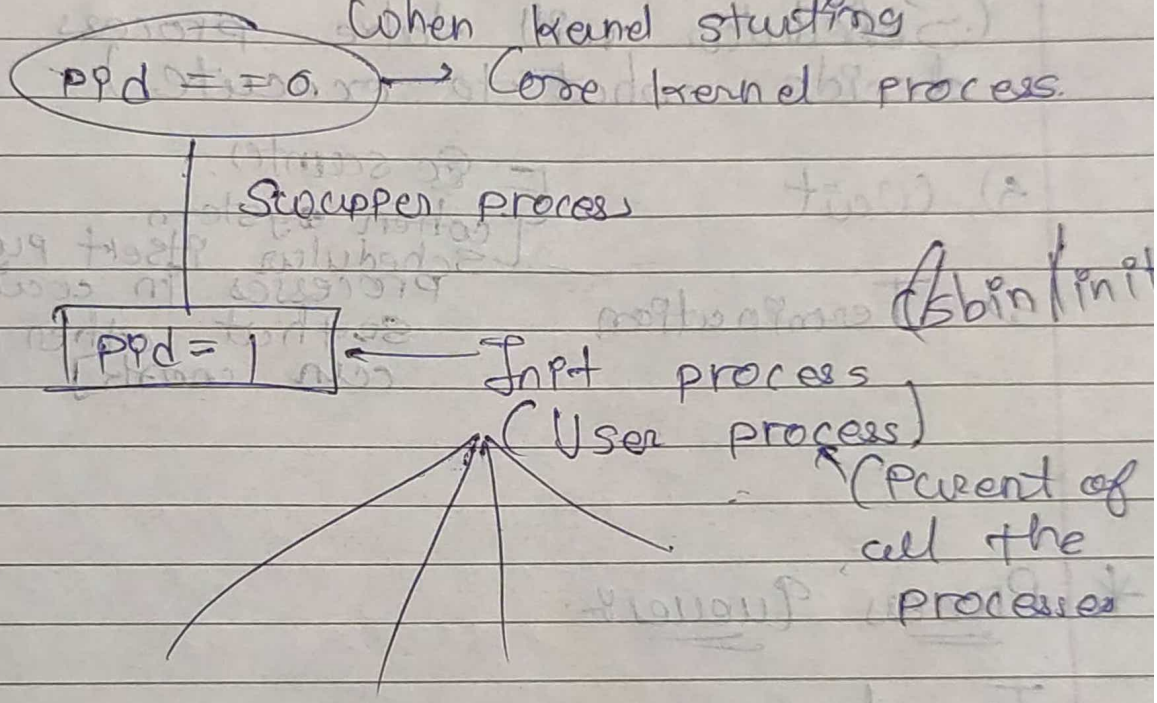
→ Each and every process determined by Unique Id.



→ Processes are isolated with each other, they run in independent environm

PID - Process Identification no
→ Each and every process is allocated with unique identifier by OS.

Same PID will allocated to other process after some time-outs.



⇒ getPid() → returns the pid of current process

⇒ getppid() → returns parent proc pid

⇒ Ctrl+C ⇒ kill signal (terminate)
1) generate hardware interrupt
2) notified by the kernel.
3) kernel sends termination signal to proc
4) On the success of call.

PAGE NO.
DATE

* Process States:-

1) New (newly created process)
↓
2) Ready
↓
3) Execution
Running state
on CPU

(Scheduler will take process and decide what to execute)

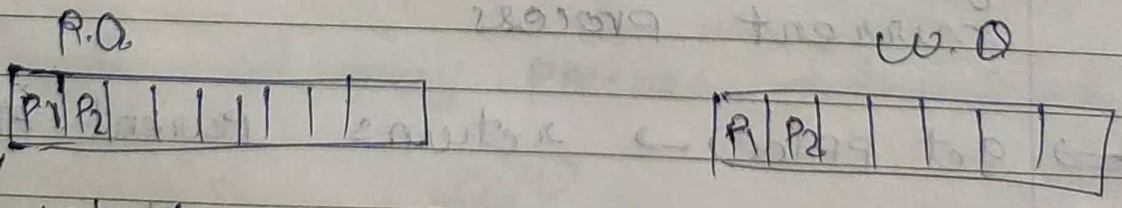
4) Wait

5) Termination

[I/O system call
Scheduler insert puts processes in wait state so that other system can work]

* Process Queues:-

Two types
1) Ready queue
2) wait queue



→ Linked list of processes that are waiting for execution

Ready to get