⊛ Linux Internals ⊛

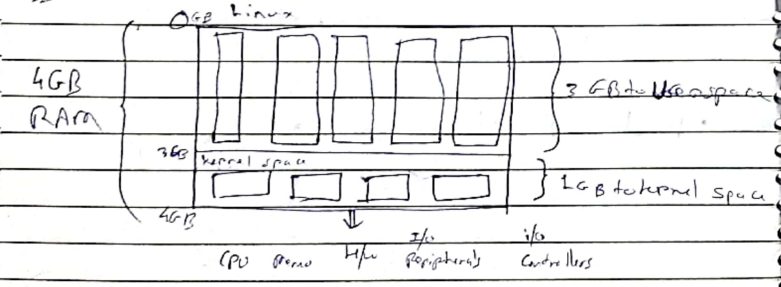
⊛ Intro to Linux

- Open source Unix like OS.
- Based on Linux kernel.
- Developed by Linus Torvalds

⊛ Advantages

1. Free Open source
2. Portable, Secure
3. Scalable / Modular
4. Runs 24*7 w/o rebooting.
5. Very short debugging time
6. Suitable for programmers.

⊛ Linux kernel Architecture



⊛ Kernel :- ① Core of Operating System

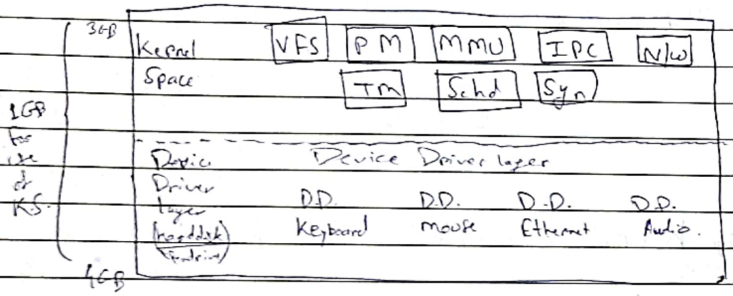
ⓐ Pure s/w code that talks to H/w.

ⓑ Kernel is 1st piece of s/w code that loads in RAM & Runs until system shutdown.

⇒ Kernel Services & its Architecture

1. File System Services
2. I/O devices services
3. Multiprocessing services
4. Multi-threading services.
5. Memory mapping & allocation services.
6. Inter process Communication.
7. Signal management.
8. Schedular services.
9. Synchronization
10. Network programming.
11. Device Drivers.

• Architecture :-



Intro to Services.

① File system.
:- Provides services so that programmer can open, read, write, modify, close existing file & create file.

② I/O device services.
:- With this we can read, write, open, create, modify, device operations.

③ Multiprocessing Services.
:- Can schedule, execute, terminate single or multiple process.

④ Multithreading Services.
:- Used for launch, schedule, execute, terminate single/multiple threads.

⑤ Memory management
:- Used for management of kernel space.

⑥ Inter Process Communication.
:- To communicate with diff count processes running on same machine.

⑦ Signal Management
:- Used to send/receive signals & used them with the processes.

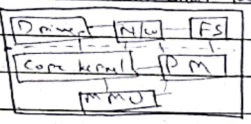
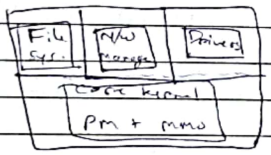
⑧ Schedular Services
:- Used to schedule process/tasks according to priority
① Round Robin ② CFS.

⑨ Synchronization
:- Used to synchronise diff. process using/sharing same resources
① Semaphore ② Mutex ③ Spinlock

(10) Network programming
: Used to communicate with devices over network. Also known as socket programming

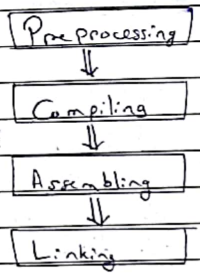
(11) Device Drivers.
: Used to talk on H/w on request of application
it has 2 interface → interface with Appln & H/w.

(*) Types of kernel
:

Monolithic	Microlithic
1) All services built into single image to have same addr loc".	1) All service are built seperately & have diff addr location.
2) 	2) 
3) OS size → large.	3) OS size → small
4) Difficult Debugging	4) Simple debugging
5) Complex Design	5) Complex design
6) More resource req.	6) less resources req.
7) Faster execn	7) Slower exec"

3) Developement tools & Utilities.

(1) Stages of Compilation.
:



- e ⇒ .i
- s ⇒ .s
- c ⇒ .o

(2) Types of Compilation.
:

Native	Cross
1) Process of creating executing file for local (same) machine or architecture	1) Process of creating executable file that runs on different/specific CPU architecture
e.g. $gcc one.c -o out	e.g. $arm-linux-gnueabi-gcc 1.c -o out
	gnueabi ⇒ G not unix executable application binary interface.

③ errors, bugs, debugging, failure, fault

:- Errors :- mistake in programming code.

bugs :- if programmer accepts defect, it is bug

debugging :- Processing of fixing bug.

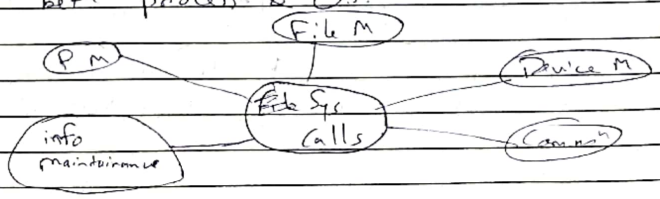
fault :- Occured due to some conditions & leads to sys failure.

Failure :- inability of the system sw to perform task function

① OS Breakdown ② Sys h/w Breakdown

④ System calls.

:- Procedure of providing interface / comm^n betn process & OS.



• Need of system call

:- Access to priveleged instructions

:- Resource Management

:- IPC

:- Process Control

:- Error & exception handelling

:- Security & protection

⑤ Command line arguments :-

:- Syntax

↳ int main (int argc, char *argv[])

argc : counts no. of arguments

argv : stores arguments given in terminal

Terminal :-

ex. :- $ ./one two three four

here, argc = 4,

*argv[4] = {"./one", "two", "three", "four"}

- Linker

:-

Types	
Static	dynamic
$ gcc -static one.c -o st-out	$ gcc one.c -o out

6) Libraries

:- A group of precompiled object files.

: Used to avoid rewriting of code.

- Need

:- (1) Reusability (4) Abstraction & Encapsulation

(2) Modularity & maintainability.

(3) Efficiency & performance.

(5) Community & Colaboration

- libraries types

Static (Compilation time)

Dynamic (runtime).

Source code
↓
preprocessor
↓
compiler
↓
linker

static linker → Static library

dynamic linker → Dynamic library

(1) Static library

:- needed files → 1.c 2.c 3.c file.h.

Steps

(i) $ gcc -c 2.c 3.c → 2.o 3.o

(ii) $ ar rcs lib-static.a 2.o 3.o ⇒ lib-static.a

(iii) $ gcc -c 1.c ⇒ 1.o

(iv) $ gcc 1.o -o out -L. lib-static

(v) $ ./out

here, ar ⇒ archive tool

rcs ⇒ replace / create / symbols

from / variables

② Dynamic Library

- needed files → 1.c 2.c 3.c fib.h

Steps :

(i) $ gcc -c 2.c 3.c -fpic ⇒ 2.o 3.o

(ii) $ gcc *.c -shared -o lib_dynamic.so ⇒ lib_dynamic.so

(iii) $ gcc -c -l.c ⇒ 1.o

(iv) $ gcc 1.o -o out -L. lib_dynamic.so

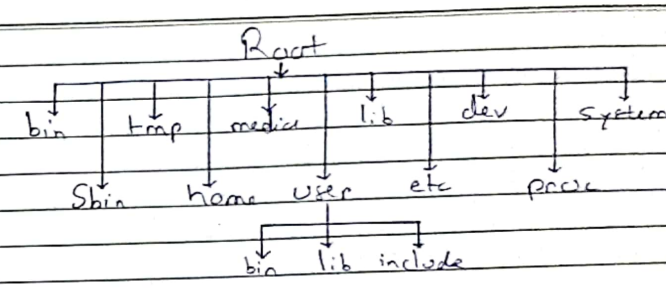
(v) $ sudo cp lib_dynamic.so /usr/lib

(vi) $ ./out

③ Working with files (F.S)

:- R.F.S. :- service that manipulates blocks memory more efficiently in order to organise files & directories in proper structured manner in memory

- Service when all file system are attached during kernel bootup time (Mount Point)



(i) Bin :- Contains lower level system utilities (LLU)

:- LLU :- programs that produce graphical o/p

- eg. bash, cp, ls, ps.
- ~~config~~files of commands used on terminal
- executables of commands used on terminal

(ii) Sbin :- Contains super system utilities (SSU)

:- SSU :- used by both admin & user

- eg. ip, halt
- executables of SSU

(iii) tmp :- contains temporary files

- removes temporary data when no longer used

4) home :- Each usr has personal file space for storage.
: This file space is named after logged in usr.

5) User is supported for Userspace.
(i) usr/lib :- libraries for higher level services.
: libraries of 3rd party applications.
(ii) usr/include :- all header file locations.
(iii) usr/bin :- Contains high level utilities for system

6) media :- Subdirectories created within this folder, when we insert removable media file.

7) lib :- Contains essential shared libraries for system boot.
:- Used for low level services. & kernel boot up time.

8) etc :- Contains files used in system administration.
:- Contains all sys config info.
:- cat /etc/passwd. , cat /etc/shells.

9) dev :- All device files are saved here. (special device files)

10) proc :- it is logical file sys. & not permanent
- Created during kernel boot up time & destroyed during shut down
- Creates lots of data structure for kernel & store them in file system.

11) System :- Virtual file system for modern for linux distributions to store & allows modification of devices connected to system.

- Sys calls for file input/output

(*) I/O devices services.
:- #include <fcntl.h> ← library

Basic I/O calls ⟹ open(), read(), write(), close(), create().

Called as Common api programming interrupts

Types of files

- **Normal**
  - Text (.c, .txt, .s)
  - Binary (.o, .out, .mp3)
- **Special**
  - (i) Pipes
  - (ii) Fifos
- **Device**
  - (i) Char/Block device files

(i) `int open (const char* pathname, int flags, mode_t mode)`

- Success → F.D.
- Failure → -1
- pathname → path of file.
- flags → O_RDONLY, O_WRONLY, O_RDWR, O_CREAT
- mode → Permission, 3 octal numbers like 777

- inode :- data structure that holds metadata about all files in sys
- superblocks :- collection of metadata used to shows properties of files.

(ii) `int creat(const char* pathname, mode_t mode);`

- success → fd
- failure → -1

(iii) `ssize_t read(int fd, void *ptr, size_t nbytes);`

- Success → no. of bytes read
- failure → -1

(iv) `ssize_t write(int fd, void *ptr, size_t nbytes);`

- success → no. of bytes written
- failure → -1

- Current file offset position
  - :- non-negative no. measures from begining of file.
  - :- initially zero :- needed in r/w operations.

(v) `off_t lseek (int fd, off_t nbytes, int whence);`

:- interpretation of lseek depends on 3rd arg

• int whence → (1) SEEK_SET : Repositon to zero

(2) SEEK_CUR : from current pos.

(3) SEEK_END : from end of the file.

• without pf & sc

FD table	
0	KB
1	Display/terminal
2	error device
⋮	⋮
1023	

printf → write(1, char *[], sizeof char*())

Scanf → read(0, char *[], sizeof char*())

(vi) int fdup(int fd);

(vii) int dup2(int fd1, int fd2);

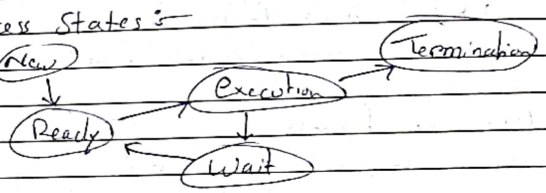
(viii) Redirect Operator
: $./out > text.txt

(ix) chmod ___ : Change file mode / permissions
3 octal digits
उदाहरण

#include<stdio.h>

8) Linux Process.
: The program / binary during its execution is called as process

• Process States :-



• Processes Queues

Ready Queue
: linked list of Process waiting for CPU slice time for execn
① $ps -elf : shows running Processes
② $ps -af : shows running process from terminal
: Ready Q. numbers depends on core of CPU
③ $pstree ④ $top

Wait Queue
: linked list of process sus... by some condition.
: no. of Wait Q. depend on ne... of resources.

- Process Identifiers (PID)
  - (i) PID = 0 :- Core kernal Process
  - (ii) PID = 1 :- init process :- ppid = 0.

PAS :- Process Address Space
PCB :- Process Control Board.

- Need of Process
  - (1) Resource allocation (2) Multiprogramming & Multitasking
  - (3) Process creation & termination (4) Scheduling (5) ipc
  - (6) Synch (7) Fault & error handling

- Process Creation
  - (i) pid_t fork();

Parent	Child
returns pid of child	returns zero

fork calls once & returns twice.
no. of processes in fork = 2^n
n = no of times fork is called.

- The primary segment of program before fork() sys call gets copied to child as it was.
- But changes after fork sys call cannot be seen in memory segment of parent & child.

(ii) vfork():-
- creates parent & child
- executes child first & then executes parent
- while child is executing, parent is in Block state

(iii) at exit(); <stdlib.h>
- 1st execution of main function is completed
- then it does execute functions in at exit fun^ in reverse order of their calls.

- Process Termination
  - (i) return 0 (ii) exit (0) (iii) Send signal for eg → ctrl C
  - (i), (ii) → Normal Term^
  - (iii) → Abnormal term^

<sys/types.h>
<sys/wait.h>

- Types of process.
  :- (1) <u>Orphan Process</u>
    :- Child executes without any parent in existance is Orphan process.
  (2) <u>Zombie Process</u>
    :- Parent is busy in doing task & failed to fetch child exit code, then Child becomes zombie state / process.
  (3) <u>Daemon process</u>
    :- Process continuously runs in background
    :- Provide services & functionalities.
    :- Does not get controlled from terminal.

(iv) wait (&stat); ⇒ pid of child.
:- waits for child to terminate & stores its exit code in 2nd byte of stat

	exit code of child		
0	1	2	3

:- WEXITSTATUS(stat); → shows exit code of child

<unistd.h>

(v) execl(path of prog., " ", " ", " ", NULL);
  (passed commands / args)

(vi) execlp(cmd name, cmd, cmd, 0);

(vii) execv(path of prog, cmds);

(viii) execvp(~~path~~ cmd name, cmds);

(ix) waitpid(child pid, & stat, 0);
  :- parent blocked until child term.

9) Thread Management

:- It is parallel <u>context of an execution</u>
  (set of instructions executing)

- pthread library :- posix Threads
  :- IEEE Standards implemented.
  :- Common api's for threads.

<pthread.h>

pthread library #include<pthread.h>

- datatypes
  - ↓ pthread_varname
  - pthread_t : for thd id
  - pthread_attr_t : for thd attr
- functions
  - ↓ pthread_fun^n name

(i) pthread_create(pthread_t *ptr, pthread_attr_t *attr, void * (fun^n name), void *args)

arg1 ⇒ tid variable
arg2 ⇒ attribute variable
arg3 ⇒ fun^n name ptr
arg4 ⇒ args to fun^n.

(ii) pthread_join(tid, ret_val);
↑ NULL

(iii) pthread_self(); ⇒ Gives thread id of self.

- Thread termination.
  :- When process terminates, thread also terminates.

(iv) pthread_cancel(tid); Used in anywhere

(v) pthread_exit(&status); Used in thread fun

- pthread_once_t = PTHREAD_INIT_ONCE
  :- Used to execute only once

(vi) pthread_once(once variable, once init fun^n);

- Pthread attributes.
  :- Used for :- Stack Management
  :- Thd synch
  :- Thd scheduling

:- Default stack size of thd is 8 mb. & can be changed by using attributes.

:- pthread_attr_t attr;

(vii) pthread_attr_init(&attr);

To destroy attr

(viii) pthread_attr_destroy (&attr);

(ix) pthread_attr_getstacksize (&attr, stacksize);
→ to get stack size.

(x) pthread_attr_getstackaddr (&attr, stacksize);
→ to get stack address.

(xi) pthread_attr_setstack (&attr, newstackaddr, newstacksize);

- Detachable thread

(xii) pthread_attr_setdetachstate (&attr, PTHREAD_CREATE_DETACHED);
↳ Detachable attribute.

also

(xiii) pthread_detach (tid);

10) ~~Memory Management~~

Process	Thread
1) Process is a program in execution	1) Thread is a segment of a process.
2) Process takes more time to terminate	2) Thread takes less time to terminate
3) Process takes more time for creation	3) Thread takes less time for creation.
4) Requires more time for context switching (state switching)	4) Requires less time for context switching.
5) Process is less efficient in terms of communication	5) Thread is more efficient in terms of communication
6) Multiprogramming holds concept of multiprocess	6) We don't need multiprogramming for multi threads.
7) Process is isolated	7) Thread shares memory.
8) It is heavyweight	8) It is ~~multi~~ lightweight.
9) If one process is blocked it will not affect others	9) If user level thread is blocked then all user level threads are blocked
10) Process don't share data with each other	10) Threads shares data with each other.
11) sys call used to invoke Process	11) Api are used to invoke threads

- System call :- Allows program to request service from kernel.
- API :- Set of protocol, routines, fun' allows exchang data among various applications & devices.

10) Memory Management. (1) Need.
:- (i) Memory allocation & deallocation
(ii) Memory protection.
(iii) Virtual memory management.
(iv) Memory fragmentation.
(v) Memory paging & swapping.

(2) Memory Partitioning
:- (1) Paging
:- Process segments are divided into parts & each part is called page. (each 4KB)
:- Page servia is provided by Low level MMU.
:- Ram is also divided into equal size of units, each unit is called Page frame.
:- Kernel maintains dynamic D.S. called pagetable

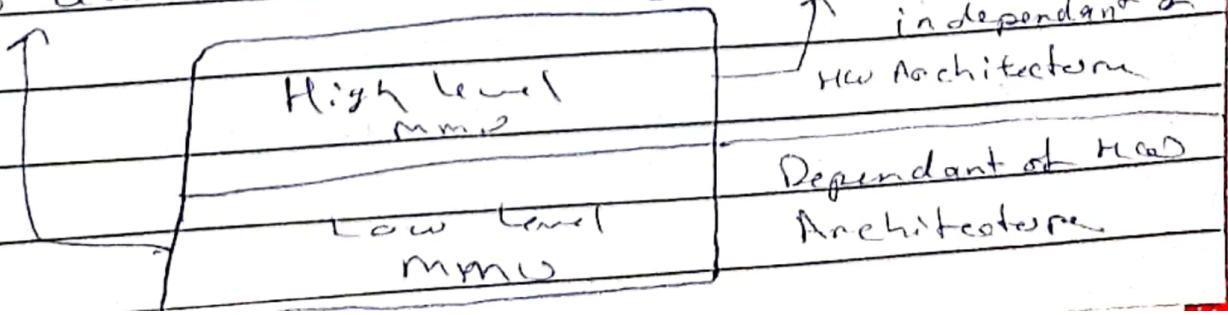
(ii) Segmentation.

- Each program has it's own program memory known as program segment.
- Each Program segment contains (1) Text (2) data (3) bss (4) Heap (5) Stack.

Pointers, function address, function variables are stored here	Stack
Dynamic memory is allocated here	Heap
Uninitialized static & global variables are stored here	BSS
Initialized global & static variables are stored here	Data
All program in text format is stored here	Text

- Types of MMU Units.

(i) Low level MMU (ii) High level MMU.



<String.h>

(i) LL MMU :- Create to initialize kernel memory management data structure (Page)

(ii) HL MMU :- (a) Slab allocation
(b) fragment allocation
↳ vmalloc();

- Memory Allocation Services

Automatic Memory Allocation (AMA)	Dynamic Memory Allocation (DMA)	Static Memory Allocation (SMA)

(*) Memory manipulation calls.
:-

(i) void *memset(addr, value, size);
:- jumps to given address, modifies its value according to given value & size

(ii) void *memchr(addr, value, size);
:- Returns address of particular character to be found in given size.

(iii) int memcmp(addr1, addr2, size);
:- Compares values of two diff address location upto given size.

(iv) void *memcpy(addr1, addr2, size)
destination | source
:- Copies values of addr2 to addr1 upto given size

(v) void *memmove(addr1, addr2, size);
:- moves values of addr2 to addr1 upto given size.

<sys/mman.h>

(A) Anonymous Memory mapping.

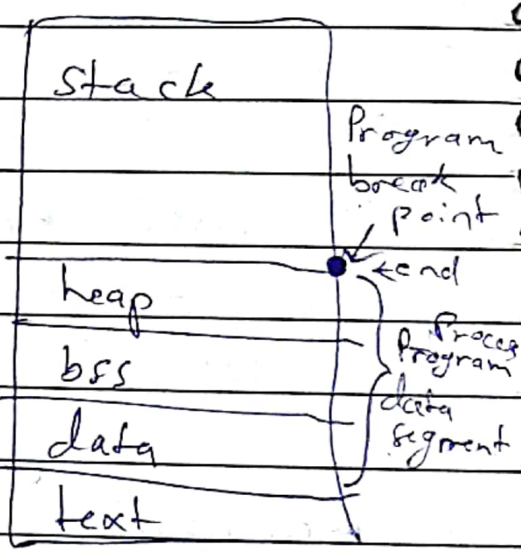
(vi) void *mmap(void *addr, size_t len, int protect, int flags, int fd, off_t offset);
:- int protect :- Read write protection
:- it is a posix fun.
:- Maps kernel/file memo. region on device memory region from kernel space to process Address space (PAS). (It is in User space).

0 -1
S f

(vii) int munmap (addr, size);
:- memory unmapping (i.e freeing allocated memory).

(viii) ~~int brk(void *end_data_segment);~~
S=0 f=-1

(viii) void *sbrk (int value);
:- function used to increase or decrease program break point by given no. of value. returns addr



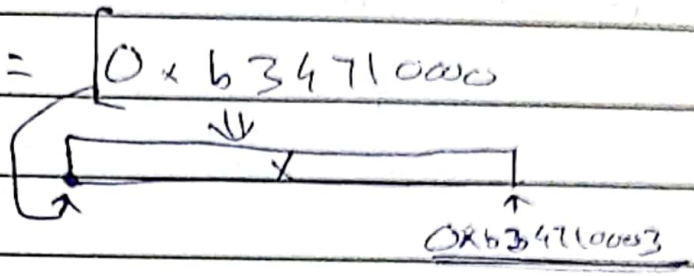
(ix) int brk(void *addr);
:- Used to increase program data segment ~~by~~ given to given address, does not returns addr.

(*) Types of Address.

Virtual	Physical
logical address ⇒ does not actually present in Harddisk	Real address ⇒ Actually present in harddisk.

(1) Virtual Address.

int x → &x = 0x63471000



Virtual Address = Page no. + Offset.

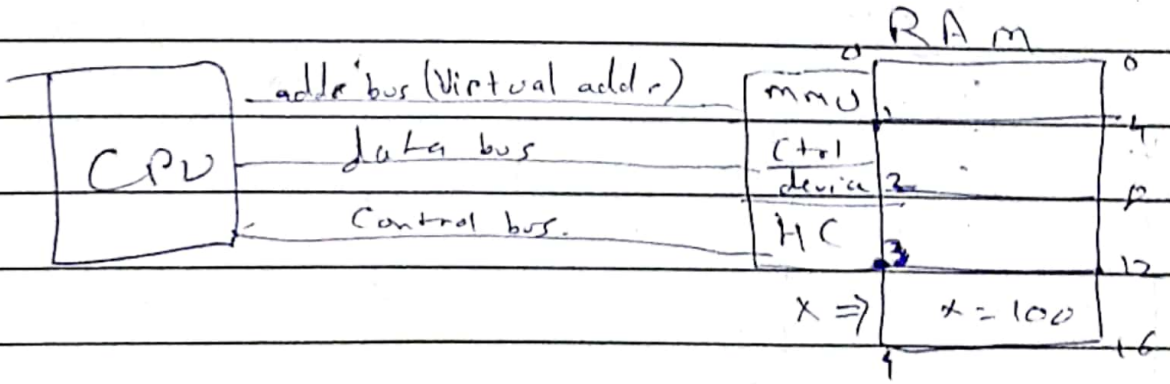
= Pg. no. * 4kB + Offset no.

eg. :- (125 x 4kB) + 500;

(2) Physical Address.

:- Frame no. x 4kB + Offset

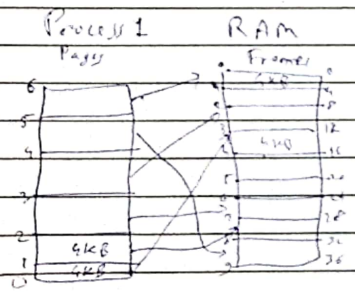
for e.g. 3 kB + 500 = 3 x 4096 + 500



(v) int mlock (addr, size);
- Applied lock on specific address for specific no. of bytes → S = 0 ⇒ F = -1.

(vi) void mlockall (int flags);
- Complete pages of process are locked.

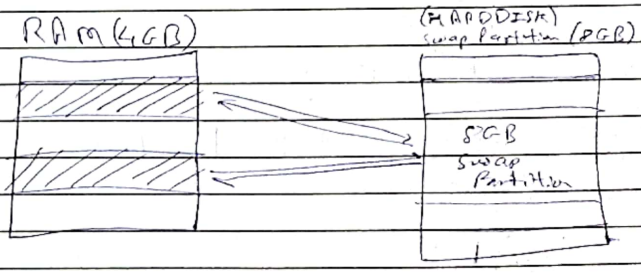
(*) Page tables
- Memory segment of process divided into parts, each part called as Page.
- RAM also divided into parts, each part called as Frame.
- Relation of page & frame stored in Page Table.



Page	Frame
0	4
1	8
2	7
3	2
4	9
5	1
6	0

(*) Memory Swapping & its need
- When sys runs out of RAM space, it uses swapping technique.
- It uses swapping partition to transfer inactive memory pages/processes from RAM to Hard disk.
- Swap partition must be twice as of RAM size.

• Need :- (1) Increase memo. capacity (2) Manage memo. pressure
(3) Balancing memory Allocation
(4) Support large application.
(5) Flexibility & adaptability.



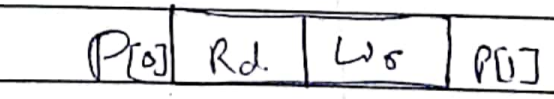
11) Inter Process Communication (IPC)

:- Communication betn process ~~running~~ on same/different machine is IPC.

Need :- ① Process Co-ordination
② Concurrent Programming
③ Cli-ferv Comm^n.
④ Inter Process Co-ordination.
⑤ Process isolation & Security.
⑥ Distributed Computing.

(*) Types of IPC

① PIPE
② FIFO
③ MsgQ.
④ Shared Memory.

① PIPE :- 

:- Permits Unidirectional data transfer.

$ ls | less

Pipe symbol :- Pipe comm^n betn ls & less

$ cat /proc/sys/fs/pipe-max-size

↳ shows max. size of pipe => 1048576

↳ 1MB

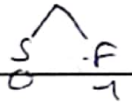
$sudo su

$ echo 4096 > /proc/sys/fs/pipe-max-size

↳ Update max pipe size to 4096 => 4096

<unistd.h>

(1) int pipe (int arr[2]); => Use Before fork();



=> Use read() to read data

=> Use write() to write data

(2) FIFO (named Pipe)

- Files used for Unidirectional process.
- FIFO are internally files with names.
- Connection Oriented.

$ mkfifo "fifoname" => newfifo.

Terminal-1	Terminal-2
$ cat > newfifo	$ cat < newfifo
Work as writer	Work as reader.

Here we write on terminal 1, will be displayed on terminal 2.

<sys/types.h>

(ii) int mkfifo(const char *pathname, mode_t mode)
→ makes fifo in given path directory.

(*) client/server appln with fifo.
→ (1) make 2 prog files (1 for cli & 1 for serv.
(2) Make & open both fifo's in both files
(3) Use one for cli to serv & other for serv to cli.
(4) Use read & write sys calls for comm^n.

(*) PIPE	FIFO
(1) Unnamed ipc object	(1) Named ipc Object
(2) Can be used only for related processes	(2) Can be used for related process as well as unrelated processes.
(3) Creates Kernel objects in K.S.	(3) Creates files in file system.
(4) Gets lost after process termination	(4) Stays still even after process termination
(5) in pipe (read()) are blocking calls	(5) in fifo, both open & read are blocking calls.
(6) parent ⇌ child	(6) client ⇌ server

<sys/types.h>
<sys/ipc.h>
<sys/msg.h>

(3) Message Queues

- Full duplex ipc.
- Linked list of messages stored in K.S.
- Identified by msg Q-ID.
- MsgQ is composed of data field & type field:

type ⇐ long int value that decides data

data ⇒ Actual data to be send.

$ ipcs -q → to show msgQ on system

```
struct msgq{
        long m_type;
        char data[100];
};
```

} MsgQ structure.

(iii) int msgget(key_t key, int msgflags);
→ creates msg Q.. & returns MsgQ id.

(iv) int msgsnd(int msgid, void *ptr, size_t nbytes, int flags);
S 0 F -1
ptr of struct

(v) int msgrcv(int msgid, void *buff, size_t nbytes, long type, int flags);
S 0 F -1

$ ipcrm -q msgid
$ ipcrm -Q key
} To remove MsgQ from system.

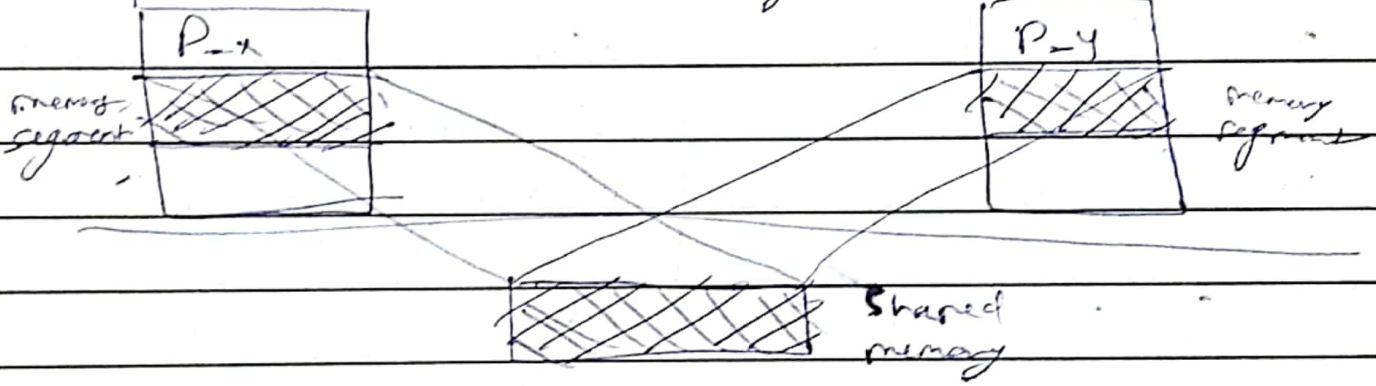
(vi) int msgctl(msgid, cmd, &buf);
IPC_STAT ← cmd → IPC_RMID

(*) FIFO	MsgQ.
(1) Conn^n Oriented	(1) Conn^n less oriented
(2) Half duplex	(2) Full duplex
(3) Uses basic I/O calls	(3) Uses Seperate Sys calls.
(4) Stream data transfer.	(4) Packet data transfer
(5) First in first out	(5) Any order associated wt type

<sys/types.h>
<sys/ipc.h>
<sys/shm.h>

(4) Shared memory

- Fastest IPC : Allows 2/more process to share same memory.
- Fastest because no sys calls are recieved.
- Attached to Process Address Space (PAS).
- No need to move one memory loc^n to another.



(vii) int shmget (key_t key, size_t size, int flags);

shmid = -1

(viii) void * shmat (shmid, addr, flags) ; <0 F-1

↳ Creates shmemory with given id & address

- After every attachment shared memory segment structure get updated.

$ ipcs -m => shows shared memories

$ ipcrm -m id } => removes shm.
$ ipcrm -m key

(ix) int shmctl(shmid, cmd, struct shmid_ds)

0 / -1 — IPC_STAT / IPC_RMID

(*) Msg Q	Shared Memory
(1) Slower	(1) faster
(2) Inherited synchronization by default	(2) Synchronization is not present by default.
(3) Secure	(3) not enough secure
(4) Get to know when message is arrived	(4) Do not gets notified when shared memory is updated/modified.

12) Synchronization

- Used when shared memory is in use.
- helps to maintain consistency & integrity when multiple process/threads uses same shared resources.

Types

- Semaphore
- MUTEX
- Spinlocks.

- Need of Synchronization
  - (1) Race Condition
  - (2) Data inconsistancy
  - (3) Deadlocks
  - (4) Starvation

(1) Semaphores :- Synchronization primitive
Used to control access over shared resources

(a) Initialize semaphore (create & initialize)

(i) int shmget (key, no of semaphore, IPC Flags);
semid ← S
↳ to create semaphore

(ii) semctl (semid, index of sem in its array, cmd, 0);
cmd ← F
↳ to control semaphore

(iii) semOp (semid, struct sembuf * buf, no of semaphore);
F → 0

(*) struct sembuf semop
{ semop.sem_num = 0
semop.sem_op = -1
semop.sem_flag = 0 }

- Race condition :- two or more process/threads try to access some shared resource at a time.
- Critical section :- Code/set of instructions that is dealing with accessing of shared resource.

- Properties of Semaphore
  :- (1) Optimised for Non contension cases.
  (2) no, iss ues with process priorities.
  (3) Cannot be used with interrupt

Types of Semaphore

Binary	Counting
Semaphore with value 0 or 1	Semaphore with value more than one

(2) Mutex
:- It is like semaphore with usage count.
:- Only one process/thread aquire mutex at a time
:- Process/thread cannot terminate while holding lock
:- Cannot be used with interrupt.

(a) Declare mutex

∴ pthread_mutex_t mymutex;

(b) Initialize mutex variable

(iv) ∴ pthread_mutex_init (&mymutex, &mutexattributes);

(c) Initialization → Dynamic

→ Static

→ mymutex = PTHREAD_MUTEX_INITIALIZER

(d) Locking

(v) ∴ pthread_mutex_lock(&mymutex) ← Gets Blocked if lock is not available

(vi) pthread_mutex_trylock(&mymutex) ← terminates if lock is not available

(e) Unlock

∴ (vii) pthread_mutex_unlock (&mymutex);

(3) Spinlocks

∴ Spinlock is a synchronizing technique which can be integrated with ISR Interrupt Service Routine

(1) Declare Spin lock

: pthread_spinlock_t slock;

(2) Initialize spinlock

(viii) : pthread_spin_init(&slock, PTHREAD_PROCESS_PRIVATE)

~~pthread_spinlock~~ shared betn thd of same process

PTHREAD_PROCESS_SHARED

↳ Shared betn proces (shared in memory).

(3) Locking

: (ix) pthread_spin_lock(&slock)

(4) Unlocking

: (x) pthread_spin_unlock(&slock)

(5) Destroy

: pthread_spin_destroy(&slock);

- Deadlock : Neither process/thread/ISR moves further.

: A has x & B has y

A req Y & B req. X B.cl Both are aquired by A&B resp. Hence Neither holds lock & does not release & both goes in forever waiting

(*)

Binary Semaphore	Mutex
1) Faster	1) slower
2) signalling mechanism.	2) Locking mechanism.
3) Operates on priority	3) Does not operates on priority.
4) Sem. value is changed according to wait() & signal operations	4) Mutex value can be modified with only lock & unlock.
5) Multiple threads can aquire semaphore	5) Only one thread can aquire Mutex at a time
6) no ownership	6) Ownership.

(*)

Mutex / spinlocks	Deadlocks Mutex
mutex => 1) Takes turns to share resource process =>	spinlocks => 1) Use loop mechanism to share resource
2) Temporary prevent thd from moving	2) Blocks thread for extended amt of time
3) Useful for time limit critical section	3) Useful for crucial extended areas
4) no use of context Switching	4) Use of context Switching
5) Disable preemption	5) Support preemption.
6) may not sleep while waiting for lock	6) may sleep while waiting for lock.

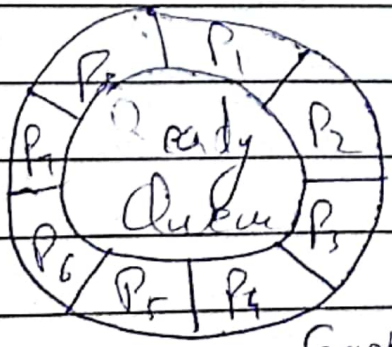
13) Scheduling Policy.

:- Used to maximise multiprogramming & multiprocessing by maximizing CPU Usage.

- Need :-
  1. Maximize CPU Usage.
  2. Recource Allocation.
  3. Multiprogramming & multitasking
  4. Process Prioritazation
  5. Interactive Responsiveness
  6. Real time system & Load balancing.
  7. Power management.

- Types of Scheduler

Round Robin



Each process uses slice time as per their turn

Complete fair Scheduler

(i) Slice time depend on no. of process

(ii) if n process are ready, then each process will get (100/n) % of CPU time

										A	B
A	1	2	3	4	6	8					
B	1	2	3	4						B	4
C	1	2	3	4	6	8	12	16		C	16
D	1	2	3	4						D	4

- slice time :- small unit of time used by process is called CPU slice time (from 10ms - 100ms)

- nice value :- it is a value that defines priority or weightage of a process. ~~Lower the no~~
  - lower the no. higher priority.
  - Ranges from (-20 to +19)
  - Default value is 0.
  - used in CFS scheduling.
  - if 10 process have same nice value
  - $\frac{1}{n} \times$ cpu slice time $\Rightarrow \frac{1}{10} \times 100 \Rightarrow 10$ ms
  - ∴ Each process gets 10 ms each slot.

- $ taskset -c 1 ./nice
  - ↳ set particular cpu core with a process.

- & => When used, runs process in Background.

Round Robin	CFS.
① Gives equal amount of slice time to each process.	① Gives slice time according to nice value
② uses slice time & Queue	② Uses nice value
③ Simple to implement	③ complex to implement
④ Less efficient	④ more efficient.