(a = 6V9K)

OS does not perform any useful function by itself, it simply provides an environment within which other programs can do useful work.

DATE: / /
PAGE NO.:

# Operating Systems

**Theory :-**

| | |
|---|---|
| Internal Assessment | 10 |
| Common quiz | 10 |
| Mid-term | 30 |
| End-term | 50 |

**Lab :-**

| | |
|---|---|
| lab attendance | → 10 |
| mid-term | → 40 |
| End-term | → 50 |

## Ch-1 :- Introduction :-

Operating System :→ It manages computer's hardware.
A program that acts as an intermediary between a user of a computer and the computer hardware.

**Computer System structure**

Has 4 components

1. Hardware → Processor (CPU), memory, I/O devices
2. Operating System
3. Application programs → web browsers, compilers, databases
4. Users.

→ • OS → controls the hardware is a resource allocator → manages all resources

→ • OS is a control program manages & controls execution of programs to prevent errors and improper use of the computer

Teacher's Signature.....

Book → Operating System Concepts - 9th edition
silberschatz, galvin and gagne

DATE: / /
PAGE NO.:

OS is the

→ "one program running at all times on the computer is the Kernel."

→ Everything else is either
- a system program
- an application programm.

} along with kernel

Associated with OS but not the part of kernel

not associated with OS

Bootstrap program. (Initialize system, verify the components, transfer the control to OS)
→ stored within Computer hardware

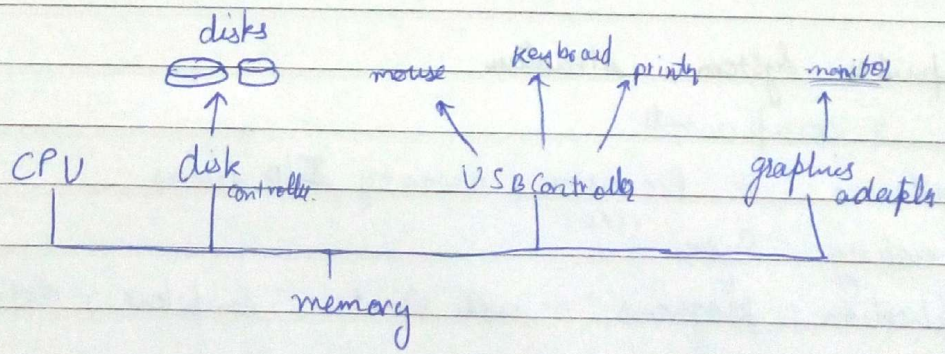
Locate the kernel and load it into memory.

When we start a system the program first executes is Bootstrap program.
- Stored in ROM or EPROM, generally known as firmware.

→ Uses
- initializes all aspects of systems (Eg hardware working verification)
- Loads operating system kernel and starts execution in memory.

now control goes to OS.

## Computer System Organization :-




## OS operation :-

→ I/O devices and the CPU can execute concurrently.

→ Device controller informs CPU that it has finished its operation by using an interrupt.

When interrupt is received by CPU then the current

Teacher's Signature.....

↑ Mobile OS include → core kernel + middleware → multimedia, graphics.
a set of software framework

DATE : / /
PAGE NO.:

execution going onn is stoped and CPU verifes the interrupt and do the required operation.

Storage device Hierarchy :→

registers
↓↑
cache
↑↓
main memory
↑ ↓
solid state disk → limited capacity but fast speed.
↑↓
hard disk (magnetic disk)
↑↓
optical disk
↑↓
magnetic tapes (and semiconductor memory have become faster and cheaper)

→ occurance of an event is usually signaled by an interrupt from either hardware or software.

→ Software may trigger an interrupt by executing a special operation called a system call.

→ The interrupt must transfer control to the appropriate interrupt service routine.

→ Solid State disks have several variants but in general are faster than magnetic disks and are non volatile.

* To ensure orderly access to the shared memory, a memory controller for CPU and device controller synchronizes access to memory.

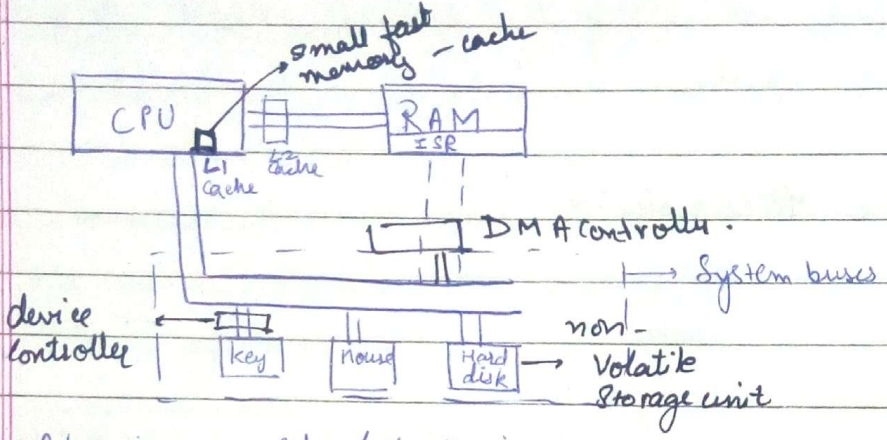
* The occurrence of an event is usually signaled by an interrupt from either the hardware or the software.

hardware → signal → CPU → interrupt

software → system call → interrupt

Teacher's Signature

3.01.11 Computer Architecture




→ A bus is a set of parallel wires

→ Device controller controls the comm. b/w device driver and device

Q. Why do we Compile the program?

To convert the source code to machine code.

→ C codes are machine independent but these codes have to be somehow converted to such a language that the particular CPU understands.

→ ISA :→ Instruction Set Architecture.

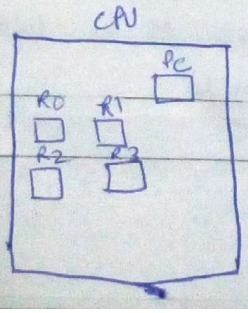
→ Assembly language — Machine language (CPU dependent)

| Assembly language | Machine language (CPU dependent) |
|---|---|
| ADD | 00 |
| SUB | 01 |
| INR | 10 |
| IN | 11 |

↓ then after conversion understood by CPU

→ A CPU has a lot many registers.

→ PC :→ Program Counter → It always store the address for the next program instruction that is to be executed.




Teacher's Signature.....

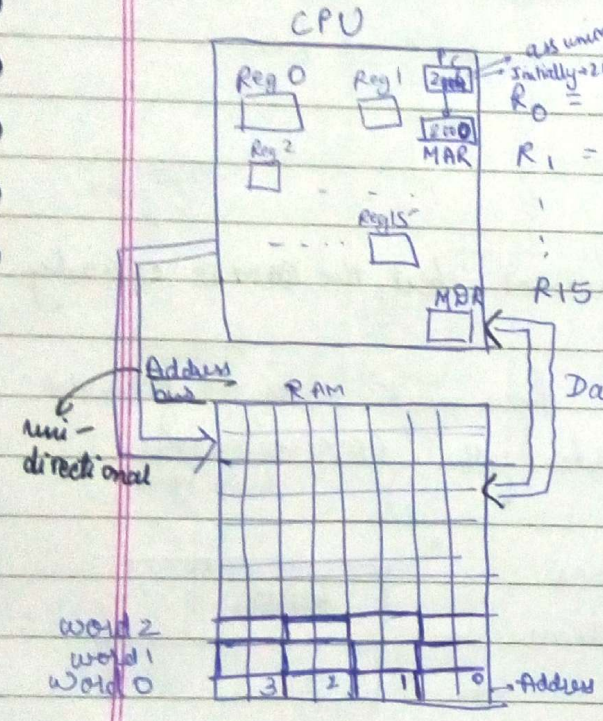
Example :-

```
void main ( )        → high level language
{
    scanf ("%d, %d", &y, &z);
    x = y + z;
}
```

Assembly language code.
↳ has to be converted into Machine code.

```
IN  Reg 1, Port 0      11 000 1 000
IN  Reg 2, Port 0      11 001 0 000
ADD Reg 1, Reg 2, Reg 3.  00 0001 0010 0011
```

Port 0 → Port 7
000 → 111




assuming that my all instructions are of 4 bits.

$R_0 = 0000$

$R_1 = 0001$

$R15 = 1111$

Reading – loading.
Writing – storing into the memory.

Word size – 32 bits

Q. What does 32 bit processor means?

It means it can process 32 bits of data.
Range of the numbers increases with bits
For eg :→ In 8 bit processor
We can just deal with 1111 1111 = 255 numbers (unsigned numbers)

→ If we have a 32 bit processor then the registers should also be of able to store 32 bit data.

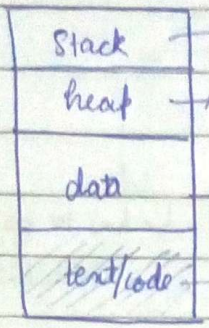
→ Fabrication technology → A 14

→ Each bite having an address → Byte-addressable unit.
Word - addressable memory

→ MAR → Memory address register

Teacher's Signature.....

→ All the instructions for a particular function are stored sequentially.




→ In 32 bits processer → It can address to $2^{32}$ locations. address can be ~~add~~ handled.

Range of address → 0 to $2^{32}-1$

$2^{32}$ – 4 GB

$2^{20}$ → 4 MB

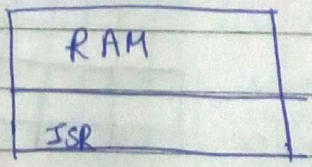
→ Why its like 32-bit processor but the RAM is 1GB only not necessary to have 4GB.

Physical RAM

It depends on the supplies how much RAM is provided. But it is possible to upgrade the RAM to 4 GB.

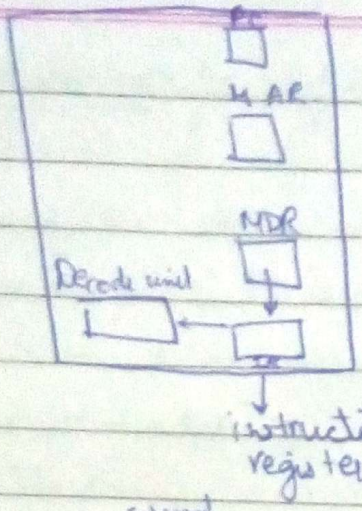
→ MDR → Memory data register

→ ISR → Interupt Service routine




→ If large amount of data has to be transferred b/w RAM and hard disk, then the processor initializes the DMA and DMA transfers data to the hard disk using bus

Teacher's Signature.....

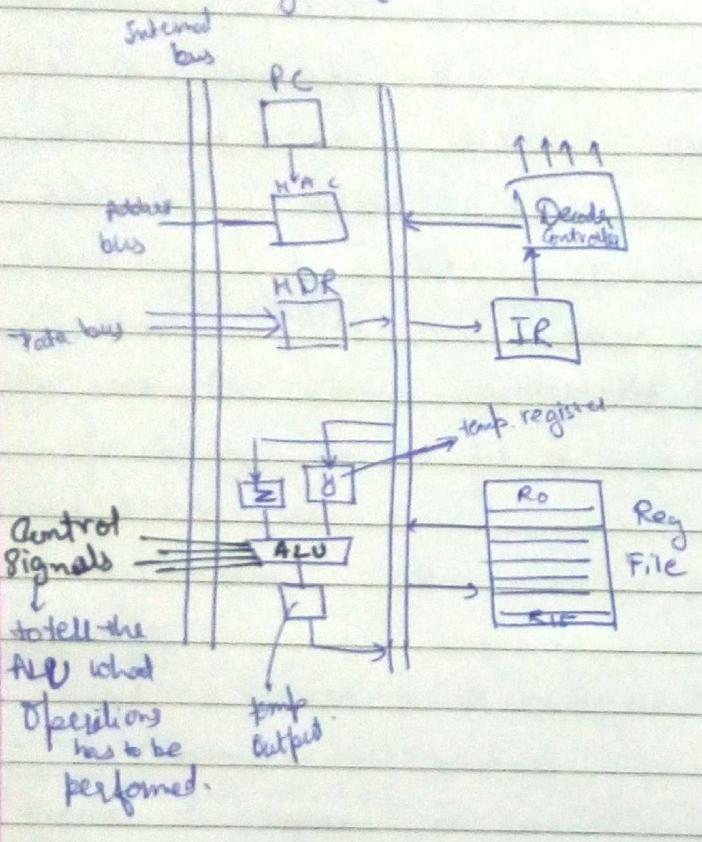



Opcode → operational code

1100 0 111 000

Opcode operands

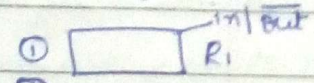
↳ once identified then the source associated signals will be activated.




Example :→

Fetch

Add Reg1, Reg2, Reg3




② $R_2$ is Activated

③ operation performed using ALU

④ $R_3$ is Activated and data is stored in $R_3$.

data path

→ For each instruction we have micro routine and for that we have micro instructions.

→ Fetch → Instruction brought into MDR is called fetch

* Fetch - Decode - Execute cycle

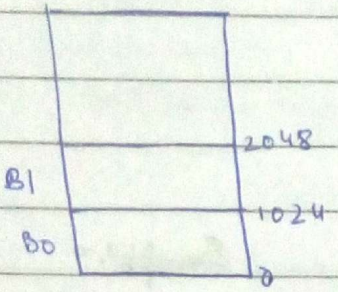
For all computer it is performed to execute any of the instruction.

→ The caches work on the principle of locality.

→ Special locality →

A CPU checks wheather a address is this in the CPU or not, if it is this than it is a hit otherwise it is a miss.

Eg:→




If we want 2000 address then the whole B1 is taken to the cache.

→ Main memory access takes millisec

→ Cache memory access takes nanosec.

↳ Decrease the time of fetch cycle.

5/01/18

DATE : / /
PAGE NO. :

## Multiprogramming (Batch System)

→ a the time of wait it switched.

needed for efficiency

→ Single user cannot keep CPU and i/o devices busy at all time.

→ One job selected and run via job scheduling

→ When it has to wait (for i/o for example), OS switches to another job.

the switching predefined

→ Timesharing (multitasking) is logical extension in which CPU switches job so frequently that users can interact with each job while it is running. creating interactive computing

→ Response time should be < 1 second.

→ Each user has atleast one program executing in memory ⇒ process

→ If several jobs ready to run at the same time ⇒ CPU scheduling

• After every time slice CPU switches its job.

→ If process don't fit in memory, swapping moves them in and out to run

moving to the secondary memory (hard disk)

→ Virtual memory allows execution of processes not completely in memory.

⇒ terminal → a monitor as well as a keyboard.

Unix → (command line execution)

⇒ Process → program in execution.

⇒ Swapping → moving a process from RAM (memory) to disk (virtual memory) and bringing a process from disk to RAM

### Memory layout of any type of multiprogramming system:→

- operating system
- job 1
- " 2
- " 3
- " 4

512M → job doesn't interfere each other is called memory protection

Teacher's Signature.....

DATE : / /
PAGE NO. :

## Interrupt driven

→ hardware interrupt by one of the devices
→ Software interrupt (eg: division by zero)
→ Request for operating System service
→ Other process problems include infinite loop, process modifying each other or the O.S.

Dual mode operations allows OS to protect itself and other system components

→ User mode and Kernel mode
(has restricted privilages) (process initiated by OS are executed in kernel mode)

→ Mode bit
Set when there is a switch of mode.

→ System call → A special fxn provided by os to the program
↳ changes mode to Kernel, return from call resets it to user.

→ Virtual machine manages mode (VMM) mode for guest VMs
① supported by hardware
② supported by software eg → java.
A special mode of execution using hardware or software we can run another O.S in one O.S.
→ More authorities than user mode and less authorities than kernel mode.

Teacher's Signature.....

8/01/18

DATE : / /
PAGE NO. :

# O.S Services :-

- **File System manipulation :-**
main storage device is structure and that is known as fileSystem.
Programs need to read and write files and directories, create and delete them, search them, list file Information, permission management.

- **Communication :-** Between process.
(Interprocess communication)

- **Error-detection :-** Takes care of hardware or software level error.

**Resource Managment and keeping track of them :-**

(1) **Resource allocation :-** CPU cycles, main memory, i/o devices, file storage.
(most complicated and important and biggest part of OS.)
Manages Processing devices, memory devices etc.

(2) **Accounting :-** Keeps track of which user is using which resource in how much amount on which processor etc.

(3) **Protection and Security :-**
The multiple running programms should not effect each other, so that the addresses are preserved and the memory is preserved.
Security from intrend (From other programs.) and extrend threats (attack through network on the System.)

Teacher's Signature.....

SHREE
DATE : / /
PAGE NO. :

## System Calls :-

→ Accessed through some interfaces and functions provided by the language like C and C++.
→ mostly accessed by programs via a high level API rather than direct System call use.
→ Three most common APIs are Win 32 API for Windows.

System call sequence to copy the contents of one file to another file.

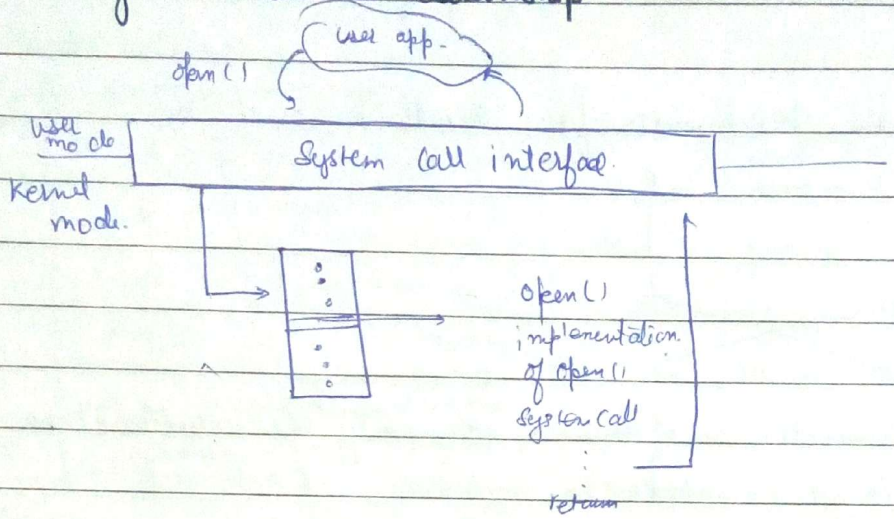
Example System call Sequence. (done by OS in Kernel mode)

→ Acquire input file name.
→ Write prompt to screen
→ Accept input
→ Acquire output file name
→ Write prompt to screen.
→ Accept input
→ open the input file
  If file ✗ abort
→ Create o/p file
→ if file exists, abort
→ Loop
→ Read from input file
→ Write to o/p file
→ Until read fails
→ close o/p file
→ Write Completion message to screen
→ Terminate normally

Teacher's Signature.....

SHREE
DATE : / /
PAGE NO. :

# API- System Call- OS Relationship




## ⇒ Process Management

Deals with the every activity associated with a Process

**Process :→**

Program written in any language which is in execution is called Process (running Program)

Program → Passive entity

Process → Active entity.

The running system may require CPU, memory, I/p devices these are needs are managed by Process management.

## ⇒ PM Activities

1) Suspending (temp. taking this CPU from process) and resuming processes

If a process requires any i/p from keyboard then the process is suspended and when it receives i/o required it resumes its old process.

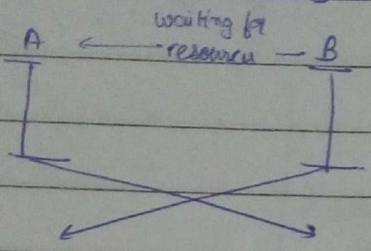
2) Synchronization among the process.

For separate o/p (sequential) of multiple programs running at the same time we need synchronization.

Teacher's Signature.....

3) Process Communication.

4) Provides Mechanism for deadlock handling.




→ If we terminate one of them than only the other will be able to get executed

→ Some cordinating process does not goes into deadlock, this is called presention from deadlock.

## Memory management :→

→ mainly deals with the maintaince of main memory.

→ Its job is to Keeping track of all memory occupied by different process.

→ Allocation and dellocation of memory space to different process

→ Some time its moves some data to virtual memory from main memory

→ Storage management → Deals with storage in disk

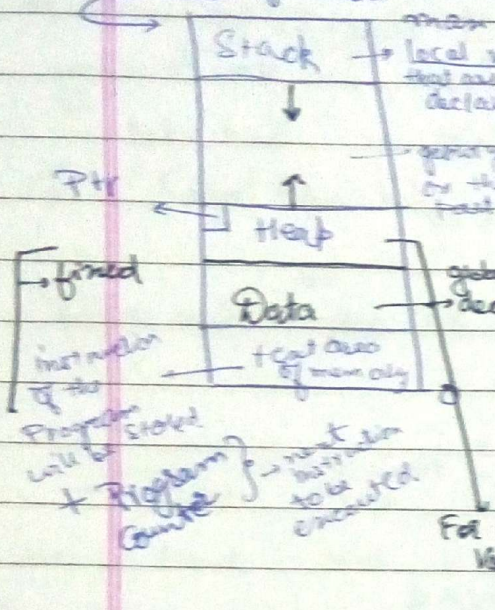
## I/O SubSystems :→

Deals with the various input, ~~the~~ output devices.

10/01/18

→ For the execution of program, first the program gets loaded into memory

Structure of a Process in memory




(when a process gets loaded, the storage 2 most important stuffs are:→

① The source code

Different States in which process go:→

① New:→ global variable will be assigned, program is loaded, PC will be at initial state only. (takes few milli sec)

② Ready:→ this process is ready for execution, if sufficient resources are available.

③ Running:→ assigns a processor to this process
fetches instructions, retrieve the value of PC
executes them.

④ Terminated:→ Two situation of termination.
① Successful completion
② Sometimes because of some error.
In both the cases the memory spaces will be freed.

After running State:→
Waiting / Suspended → goes to Ready State
This might be situation that from running state the Process goes to waiting state (Eg. → get input from keyboard)
— handed over to particular device driver and process does some other execution

Teacher's Signature

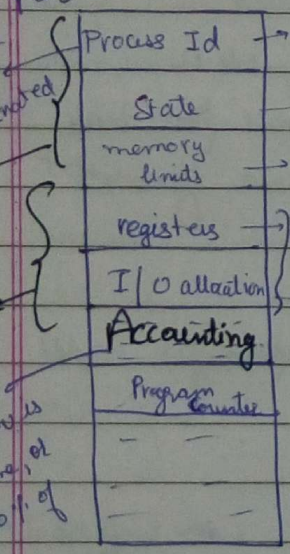
CH-3

Kernel maints all these informations, this is access to only some informations.

Sometimes the program goes directly from running to waiting state.

This may be due to some interrupt. whenever a process during running a process, encounters an interrupt it goes to ready state for analysing what type of interrupt it is, and what needs to be done.

→ During the initialization O.S makes a data structure called PCB (Process Control Block)




CPU Scheduling info, memory allocation info, etc all these informations are stored in PCB.

- After assiging the memory space, PCB is prepared.
- After execution PCB is deleted and memory is freed.
- If we grow the memory area beyond the limits, the program gets terminated.

→ CPU Scheduling info :- Includes a process priority, pointers to scheduling queues, and any other scheduling parameters.

Teacher's Signature.....

fig 1.8, 1.5 - multitasking, context switch

12/01/18 **System Calls**

DATE : / /
PAGE NO. :

① <Linux / sched.h> → similar to PCB. ← header file.

Structure name → task_struct sched → PCB corresponding structure.

Created as unsigned integer. ← pid_t pid ;
long State

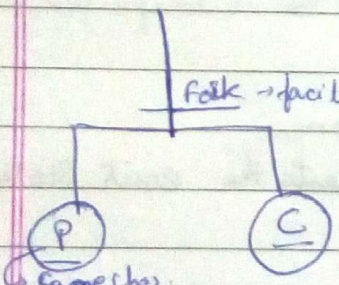
unsigned int time-slice ;
↳ depending on this value, Processor time to each event is allocated

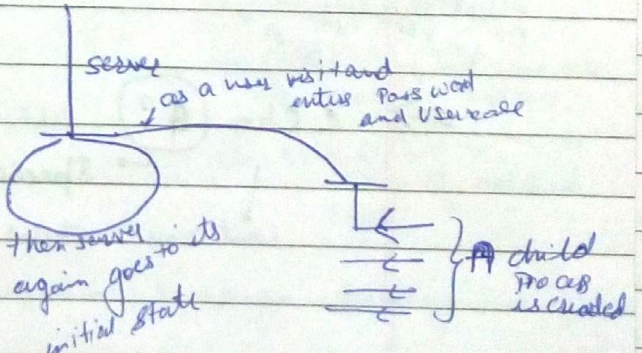
② <sys/types.h> ↳ necessary

→ struct task_struct sched Parent ; → PCB of Parent process.
" " Child ;
↳ list

→ Struct Files_list *F ; → Contains multiple value, point to the list of files, opened for this process.
Struct mm_struct *m
↳ have information about the memory of the process, max address, min address

→ 1 Parent can create multiple child process.




Eg :→

```
int main()
{   pid_t pid ;
    pid = fork() ;   <0 → their is some error, sys was unable to report new Process
    if (pid <0)      successful {=0 >0
        → fprintf (stderr , " Error ") ;
    if ( pid == 0 ) //child
```

String of files or o/p

Teacher's Signature.....

./a.out < i.txt

./a.out > i.txt → o/p of this program should be redirected to this file.

> i.txt 2 err.txt

↳ all the error message should go to err.txt.

```
{
  ==
  ==
}
```

```
if (pid > 0)        // Two process when Parent executes in this if, & child in (pid == 0)
{                   for (i = 0; i <= 10; i++)
  ==                    Printf ("in parent");
}
return 0;
```

→ Child can invoke some external process using System calls. exece (" ↓ ")

LS command

→ System call by Parent → wait ()

↳ Parent is voluntarily requesting for a halt, it will wait until the the child finishes.

exit(0) i.e. successful exit → 0, errors → 1

⇒ echo $? Write after the program.

↳ Special variable, Represents the exit Status

Contains Status of last command

## Interprocess Communication :⇒

Multiple processes in execution, if required can communicate with each other.

→ Child and parent have common address space.

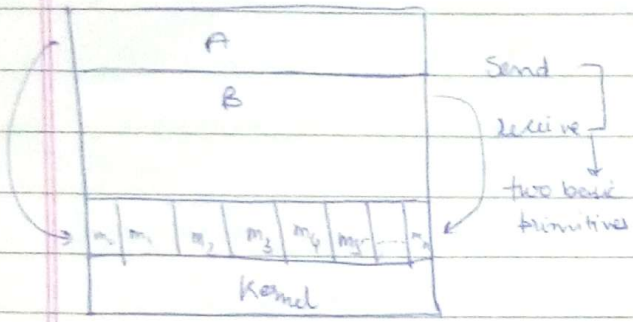
Teacher's Signature.....

Need for interprocess communication :→

① Information Sharing
② Computation speedup
③ Modularity
④ Convenience

For communication there are two most popular techniques

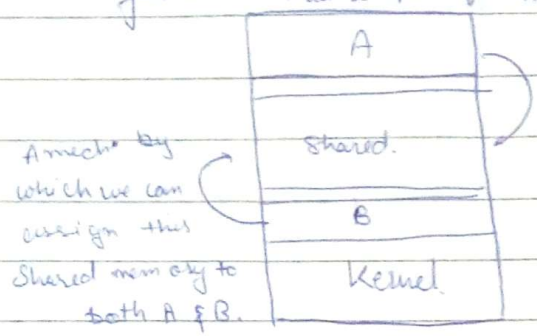
(1) Message passing




(2) Shared Variable / memory

faster.

Address space of the Process




→ Information Shared using a numbered message

→ We need to establish two unidirectional channels

→ Mainly required when multiple Process are cooperating

Cooperative process → Can effect each other operation

Independent operation → doesn't effect each other.

Difference.

| Shared memory | Message passing |
|---|---|
| ① both Running on the same system | Implemented for ① distributive Systems also, b/w multiple systems. |
| ② Direct memory access using Common Pool | ② established with the help of external layers. |

→ If both the process are on the same system message passing can be done using shared variables

Producer - Consumer :→

→ Once the producer produces then only the consumer starts.

→ We need to have some space where the can communicate.

```
#define bs 10        ← buffer size.
struct {
    ____
    ____
    ____
} item;
```

Teacher's Signature.....

DATE : / /
PAGE NO. :

# Bounded Buffer.

circular buffer

Item buffer [bs];

region of memory shared

↳ is shared by many processes

↳ whenever there is some information in it, consumer can consume it.

int in = 0; ↳ represent the position where next will be produced

int out = 0; ↳ From where the item is given to consumer.

→ When in = out the array is empty.

→ When the list is full, Producer will be having no work, when list is empty the producer will work consumer will wait

Producer will wait for some space in buffer to be empty. for some entries in the list

Producer:

While ((in+1) % bs) == out) // wait;

↳ if not equal, means no item in the list.

buffer [in] = nextitem;

in = (in+1) % bs;

Consumer:→

While (in == out) // wait

↳ no item, so consumer has no activity to perform.

Consumed_item = buffer [out]

out = (out+1) % bs

Position to the point from where the next item has to be taken.

→ For unbounded buffer we need a lot of memory space.

Teacher's Signature.....

A communication link must exist

→ message passing is implemented using a queue.
Can go beyond Producer - consumer as it is done among multiple systems. Here we need send and receive type of fns.

⇒ **Direct** or **Indirect**

If we specify a receiver.

Send(A, msg)
receive(B, msg)

Send and receive message without any more information

send(msg)
receive(msg)

→ Direct or indirect communication
→ Synchro or asynchro comm.
→ Automatic or explicit buffering.

17/01/18

**Threads :-** (thread ID, p.c, a register set and a stack)

→ If we perform multiple processes using Multiple processing is called heavy weight process.
→ Whenever a process makes a subprocess, some values are duplicated.
→ Increasing memory footprint of our program due to subprocess
heavy weight coz it copies its data at multiple places.
Multiple process model memory footprint will be much more than Multi thread model
↳ less memory for same no. of users or processes.
→ Extra redundant data is removed in case of multithreading
→ Multithreading model helps us in improving the performance.
↳ supported by libraries

fork, wait, exit } → for multi process

Teacher's Signature.....