★ What is OS?

- system software
- acts as an interface between user and computer hardware

★ Functions -

- Memory Management
- Processor Management
- Device Mgmt
- File Mgmt
- Security
- Control over system performance
- Job accounting
- Error detecting aids
- Co-ordination between other softwares & users

★ System vs Application software

System software	Application software
General purpose software	Specific-purpose software
Written in low-level lang.	Written in high-level lang
Fast	slow
Used to run system	Used to do a single task
Small in size	Large in size
pre-installed	installed by user
doesn't interacts with user	interacts with user directly
complex to design	easy to design
Run independently	runs on system software
e.g. OS, disk drivers	text editors, browsers.

DATE: / /
PAGE NO.:

★ Basic co.

★ Components of OS
1. Process Management
2. File Management
3. Network Mgmt
4. Main memory
5. Secondary storage
6. I/O device
7. Security
8. Command Interpreter system

★ Interrupts
- an event that notifies the CPU to take immediate action.
- caused by components other than the CPU.
- Two types
  ① Hardware interrupts
    - caused by hardware devices like timers, mouse, keyboard, hard disks etc.
    - has number associated with each interrupt
    Two types
      A. Maskable interrupts - CPU can delay these interrupts when it receives higher priority interrupts
      B. Non-maskable interrupts - cannot be delayed, should be considered immediately.
  ② Software interrupts
    - caused by an executing program

When an interrupt occurs, CPU pauses executing ~~ex~~ program and executes the corresponding ISR. After handling interrupt, CPU executes the usual program.

DATE: / /
PAGE NO.:

★ System calls

- A programmatic way in which a computer program requests a service from the kernel of the operating system it is executing on.
- Allows a program to communicate with the kernel to access a resource such as memory or a hardware device
- When system call is made, the mode changes from user to kernel
- After completing the task, the mode changes from kernel mode to user mode.

★ User & kernel space

- seperation serves to provide memory protection & hardware protection from malicious or errant software behaviour
- Kernel space is reserved for running a privileged OS kernel, kernel extensions & most device drivers
- User space is the memory area where application software & some drivers execute.

★ User vs kernel mode

	User	Kernel
→	restricted mode / slave	privileged mode / master
→	application programs are executing & starts out	computer enters when accessing hardware resources.
→	process gets their own address space	processes get single address space
→	if an interrupt occurs, only one process fails	if an interrupt occurs the whole OS might fail.

★ File permission

User owner - u
group owner - g
others - o
All - a

Assign +
remove -
Overwrite =

① symbolic permissions

read : r , write : w , execute : x.

		user	group	others
Default	file	- r w -	- r - -	- r - -
	directory	r w x	r - x	r - x

★ Changing permission

chmod u+rx file/directory name

② Numeric permission (always overrider)

	Binary		
- - -	0 0 0	0	Default
r - -	1 0 0	4	
- w -	0 1 0	2	file 644
- - x	0 0 1	1	dir 755
r w x	1 1 1	7	

★ changing permissions

chmod 612 file/dir

u → rw
g → x
o → w

DATE: / /
PAGE NO.:

★ Changing owner

① User owner

chown u1 filename

↓

user name

only one

To make more than 1 user owner, make group of that users & then make group owner.

② Group owner

chgrp groupname filenameoname

★ Access Control List

① show permissions

get facl filename

② Adding permisions

① To add permissions for user

set facl -m "u:user:permissions" file/dir name

② To add permissions for group

set facl -m "g:group:permissions" file/dir name

-m → moder

★ What is shell -

- intermediator / interpreter between user & OS
- Takes i/p, translates in machine language & processes it on OS level
- for o/p, machine language code is sent to shell, shell translates it and sends it to user.

Types -

1. The csh shell - csh
2. The Bourne shell - sh
3. The korn shell - ksh
4. GNU Bourne-Again shell - BASH
5. Restricted shell

★ Shell programming

- Shell script - file that contains shell commands, variables & loops, decision & sequence
  - → shebang
    #!/bin/bash
    %./script → to run script

⇒ Command substitution

```
$LIST = `ls`        → backquotes
$echo $LIST

$(command)
$LIST = $(ls)
$echo $LIST
```

DATE: / /
PAGE NO.:

* Arithmetic Evaluation

① using let

```
$ let x = 10+2*7
```

② $[expression] or $((expression)) or "expr"

```
$echo "$((123+20))"

$echo "$[123*20]"

~~echo $(expr 123 +~~
echo $(expr $x+$y)
```

* for floating point arithmetics we used 'bc' basic calculator

```
$ echo "$x+$"y | bc
```

* Decision loops

```
if [expression];
then
        statements
elif [expression];
then
        statements
else
fi
```

* string comparisons

=	equal
!=	not equal
-n	length greater than zero (true or false)
-z	length is equal to zero (true or false)

* Number comparisons -

-eq
-ge returns true or false
-le
-ne
-gt
-lt

n1 comparison n2

* logical operators

!		NOT
-a	or &&	AND
-o	||	OR

* File operators -

-d - checks for directory
-f → file
-e → if exists
-r
-w
-x

-operator filename

DATE: / /
PAGE NO.:

* Case statement

```
case $ var in
        val 1 )
                statements ;;
        val 2 )
                statements ;;
        *)
                statements ;;
esac
```

→ multiple patterns can be listed by |

* while ~~express~~ loop

```
while expression
 do
        statements
 done
```

* Until loop

```
until [expression]
 do
        statements
done
```

* for loop

```
for var in list
   do
        statements
done
```

* break

```
while [condition]
do
   cmd -1
   break
   cmd -n
done
echo "done"
```

* continue

```
while [condition]
do
 cmd -1
 continue
 cmd -n
done
echo "done"
```

★ What is process

A process can be defined as
1. A program in execution
2. An instance of a running program
3. an entity, that can be assigned to & execute on a processor.

★ ⇒ preemptive scheduling

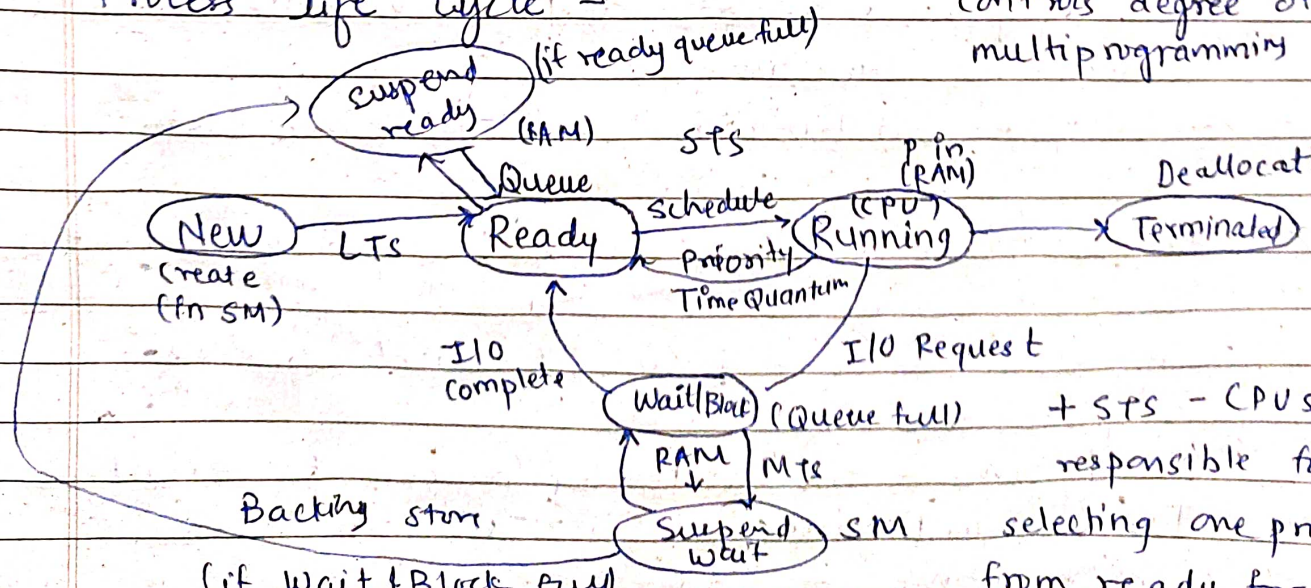
- works by dividing time slots
- process can be interrupted before completion.
- Maintaining queue & remaining time needs storage overhead
- e.g. priority scheduling, Round Robin

⇒ Non-preemptive scheduling

- process holds resource till process is terminated
- process is not interrupted until its life cycle is complete
- No overheads required.

★ Process life cycle -

+ LTS - Job scheduler
- controls degree of multiprogramming



+ STS - CPU scheduler responsible for selecting one process from ready for scheduling on the running state

MTS - swapping process from MM to SM & vice versa. may have effect on performance

LTS → Long Term Scheduler

★ Scheduling Algorithms.

① FCFS
- Jobs are executed on first come, first serve basis
- based on FIFO queue

② SJF
- Best approach to minimize waiting time
- processor should know in advance how much time will process take to execute

③ Priority Based
- Process with highest priority is executed first
- if same priority then executed on FCFS basis

④ Round Robin.
- Each process is provided a fix time to execute, called as quantum
- Context switching is used to save states.

★ Arrival Time - time at which process arrives in ready queue
Completion Time - Time at which process completes the execution
Burst time - time taken by process for execution
Turn around time - CT - AT
Waiting time - TAT - BT

★ fork

→ used to create new process i.e. child process

- runs concurrently with the process that makes the fork() call i.e. parent process.
- child process uses same resources as that of parent process
- fork() returns
  - -1 (negative value if creation of child process was unsuccessful
  - 0 returned to the newly created child process
  - positive value (pid) → to parent or caller

- no. of processes created $= 2^n$
  n = no. of time fork call is made

★ waitpid vs. wait

	waitpid	wait (can avoid zombie process)
-	can be made non-blocking call by WNOHANG option	blocking call
-	waits for a specific child equal to pid	waits for any child process
-	waits only for terminated child	waits for terminated or a signaled child

return value
- -1 no child process
- PID child terminated
- 0 child exists but not terminated

~~returns -1 if child process has terminated~~

(same for wait) returns nothing -1, PID if non of the child is terminated. caller block

★ exec system call.

- used to execute a file which is residing in an active process
- when exec is called previously executable file is replaced & new file is executed.
- entire content of the process is replaced with ~~so~~ a new program
- new program is loaded into same process space.
- not PID is changed.
- but the data, code, stack, heap etc. of process are changed & replaced.

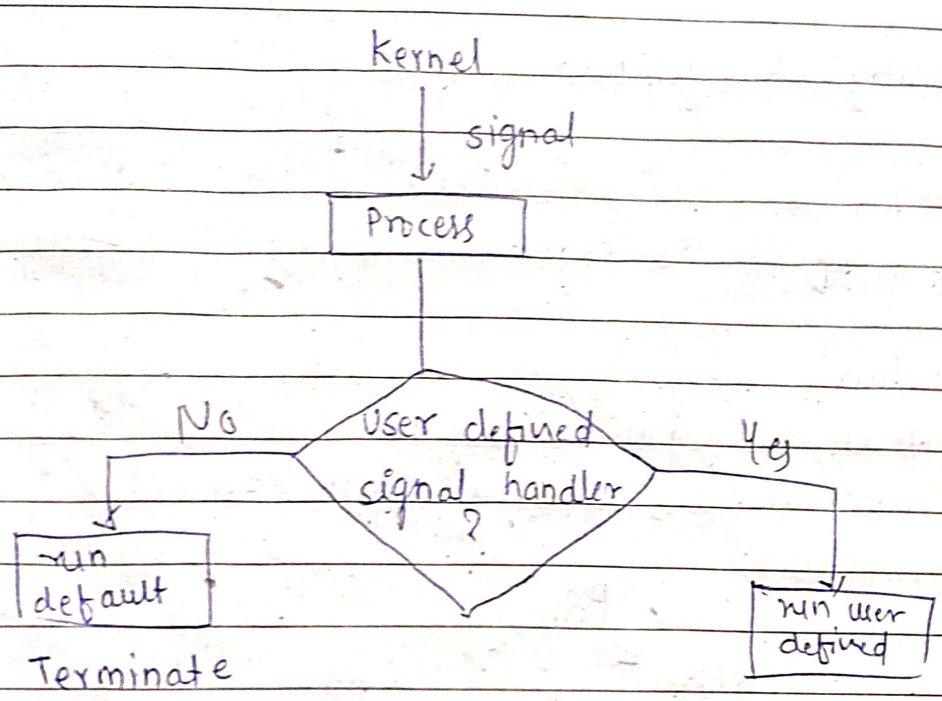
★ Signals

- limited form of ipc {no data is shared}
- notification sent to a process in order to notify ~~if~~ it of an event that has occurred
- signals can be delivered synchronously or asynchronously.

① synchronous - if signal is called internally i.e. by process itself.
e.g. division by zero SIGFPE

② asynchronous - result of some ~~a synchronous~~ external event. eg. ctrl+c sigint
ctrl+z sisstp.

interrupt - comm in CPU & kernel
signal - comm in process & kernel



Terminate
Dump
Ignore
Stop
continue

★ SIGSTOP & SIGKILL cannot be ignored.

★ sending signals

$ kill -signal pid

★ trapping
$ trap command signals

Used to ① clean up temporary files
② Ignore signals

① $ trap " rm $WORK DIR / work1-$$ &
$WORK DIR / datacut $$; exit" 1, 2

② $ trap " " 1 2 3 15.

★ Memory Management

(1)

★ Types of memories -

① Primary memory
- known as main memory
- communicated directly with CPU

Primary

RAM	ROM
① D.RAM	PROM
② SRAM	EPROM
	EEPROM

② Secondary memory
- also known as auxiliary memory
  e.g. pendrive
  HDD etc.

★ Memory management techniques

- Contiguous
  - Static
    - Fixed Partition
  - Dynamic
    - Variable Partition
- non-contiguous
  - ① Paging
  - ② Multilevel Paging
  - ③ Inverted paging
  - ④ Segmentation
  - ⑤ segmented paging.

* Fixed partitioning

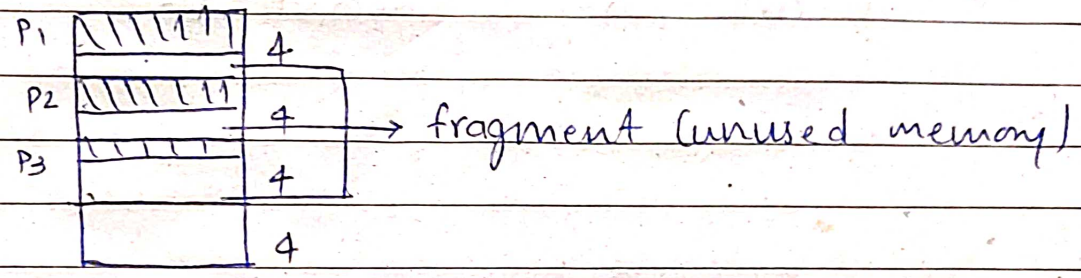
* Internal Fragmentation.

- occurs when fixed size memory blocks are allocated to the process
- free space is created when memory assigned is larger than memory requested

$P_1 = 3$

$P_2 = 2$

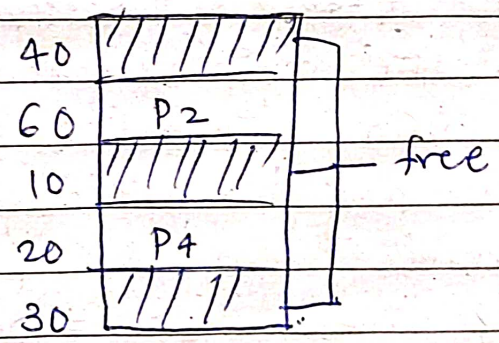
$P_3 = 1$



* External fragmentation

- occurs when variable size memory space is allocated
- memory cannot be allocated as it is non contiguous.

40	P1
60	P2
10	P3
20	P4
50	P5



P5 comes.

Available memory $40 + 10 + 30 = 80$

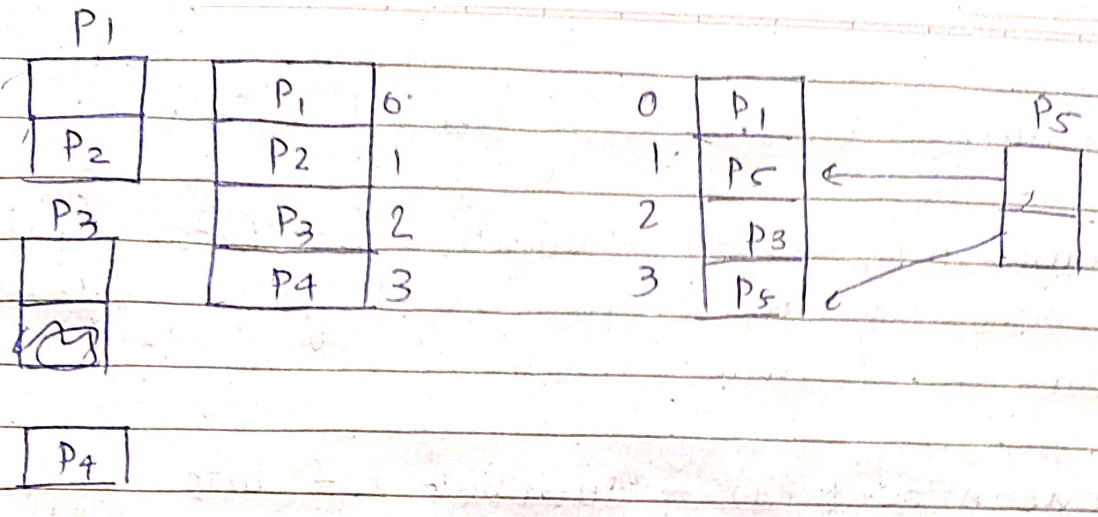
But still cannot be allocated as it is non-contiguous.

	Internal	External
→ block size	fixed	variable
→ happens when	req. memory > < available space	when process is removed
→ solution	best-fit block	compaction & paging
→ occurs with	paging & fixed partitioning	segmentation & dynamic partition
→ occurs in	worst fit	best fit & first fit

☆ Paging

- Paging is a storage mechanism that allows OS to retrieve processes from secondary storage into the main memory in form of pages.
- The main memory is divided into small fixed-size blocks of physical memory called frames.

  frame size = page size — To avoid external frag.
- used for faster access of data.
- one page of the process is stored in one frame
- can be stored at different locations but priority to contiguous frames or holes



★ Number of pages per process = $\frac{\text{Process size}}{\text{Page size}}$

Number of frames = $\frac{\text{Main memory size}}{\text{frame size}}$

★ Memory management unit
- maps logical address to physical address.

★ Logical address is divided in to

Page number(p) - Number of bits required to represent the pages in logical address space or page number

Page offset (d) - Number of bits required to represent particular word in a page or page size of logical address space or word number of a page or page offset.

★ Physical address is divided into

Frame number (f) - Number of bits required to represent the frame of Physical address space or frame number.

Frame offset (d) - Number of bits required to represent particular word in a frame or frame size of Physical Address space or word number of a frame or frame offset.

★ Page table

Enable / Disable

Frame No.	Invalid (0) / Valid (1)	Protection RWX	Reference 0/1	Caching	Dirty

← mandatory field → ← optional fields →

① Frame no - no. of frame in which page is residing.

(1) (0) (1)

② Valid / Invalid - tells whether page is present or not.

③ Protection - permissions on data in page read, write, execute

(0/1)

④ Reference - tell whether page was brought into main memory previously or not

⑤ Caching - data static ~~en~~ & used frequently - enable
else disable

⑥ Dirty bit / Modify → 1 - data modified in main memory.
needs to change in hard disk

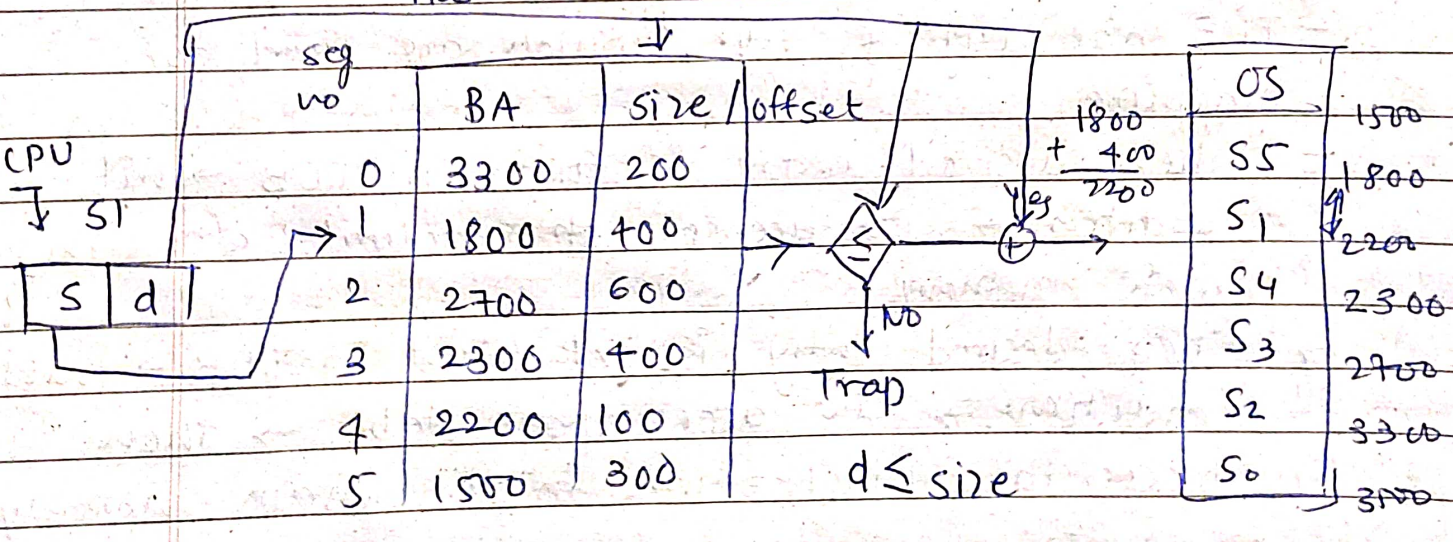
★ Translational look aside buffer.
- memory cache
- used to reduce Effective access time
- time taken to access TLB is less than time taken to access main memory
- TLB is faster and smaller than the main memory & cheaper & bigger than the register.
- follows concept of locality of reference which means it contains only the entries of pages that are frequently used.
- TLB has tags & keys, which are used for mapping.
- when desired entry is found in TLB → hit condition → CPU accesses the actual location in main memory
→ when desired entry is not found in TLB → miss condition → CPU accesses page table → then access the actual frame in the main memory

★ Shared pages
- pages with common code shared between multiple processes
- not all pages are shareable
- read-only pages are shareable - eg. program text

- data pages are not shareable.

* re-entrant code
- reusable routine that multiple programs can invoke.
- non-self modifying code.

* Segmentation
- divides the program/process into segments
- segments are saved in main memory
- segments are of variable sizes.
- more closer to user perspective
- CPU generates logical addres → MMU (using segment table) convert it into physical address → segment's location.
- segment table
  has two entities Base address & size/limit



★ Most process do not need all their pages.

DATE: / /
PAGE NO.:

★ Virtual memory
- storage scheme that provides illusion of having very large main memory
- done by treating part of secondary memory as the main memory
- User can load the bigges size processes.

Adv.
- increased degree of multiprogramming
- large application can be run with less RAM

disadv.
- slower system due to swapping
- lesser hard disk space
- more time in switching between applications.

★ Demand paging
- method of virtual memory management
- pages which are least used, get stored in the secondary memory
- A page is copied to the main memory when its demand is made or page fault occurs.

★ Belady's anomaly
- phenomenon where increasing frame size leads to increase in page faults for a given memory access pattern.
- commonly seen in
  - FIFO
  - second change algorithm
  - Random page replacement algorithm

- Can never occur in
  + LRU
  + optimal
  as they are stack-base

★ Page replacement algorithms

① FIFO → in this algorithm, a queue is maintained. The page which is assigned the frame first will be replaced first. In other words, the page which resides at the rare end of the queue will be replaced on the every page fault.

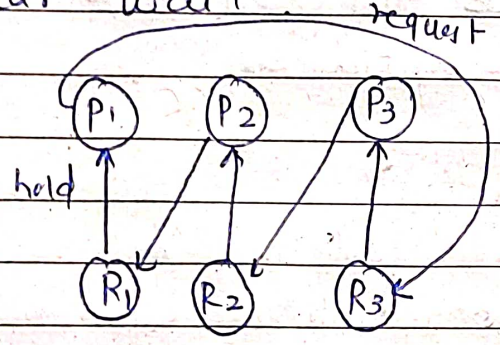
② LRU - least recently used : This algorithm replaces the page which has not been referred for a long time. This algorithm is just opposite to the optimal page replacement algorithm. In this, we look at the past instead of staring at future

③ Optimal Page replacement algorithm - This algorithms replaces the page which will not be referred for so long in future. Although it can not be practically implementable but it can be used as a benchmark. Other algorithms are compared to this in terms of optimality.

DATE: / /
PAGE NO.:

★ Deadlock → if two or more processes waiting for an event to occur, that will never occur. (rarely occurred)

⇒ Necessary conditions of deadlock.
- Mutual Exclusion - only one process using resource at given time
- No-pre emption
- Hold & wait - One process holds one resource & waits for another resource
- Circular wait



★ Deadlock Prevention & ~~adv~~ avoidance

① Deadlock ignorance (ostrich method)
- ignore the deadlock
- used by ~~winod~~ windows / linux
- deadlock not ~~had~~ handled as more code will affect the system performance

② Deadlock prevention
→ try to remove all or any of following
- Mutual Exclusion
- No pre emption
- Hold & wait
- circular wait

③ Deadlock avoidance (Bankers algo)
- checking for occurance of deadlock when resource is allocate.
- allocation of can be safe (no deadlock) & unsafe

④ Deadlock detection & recovery
- after deadlock detection processes are killed one by one is till deadlock is resolved
- Resource pre-emption.

★ Critical section
- the segment of code where processes access shared resources, such as common variables & files & perform write operations on them.

• No two processes can execute in critical section at same time.

★ Semaphore
- integer variables
- used to solve critical section problem
- has to atomic operations wait () & signal ~~fro~~ for process synchronization.
- ① wait

```
wait(s)
{
    while (s<=0); // no operation.

    s--
}
```

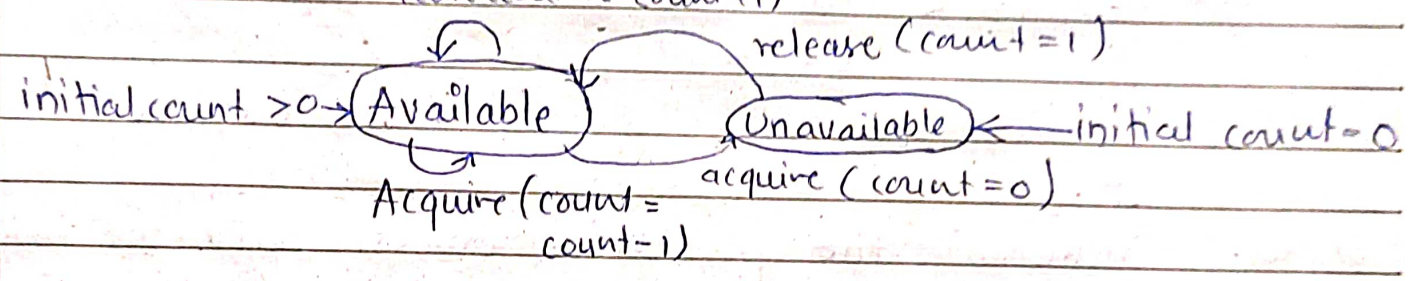
② Signal

```
signal (s)
{
    s++
}
```

- Types of semaphore

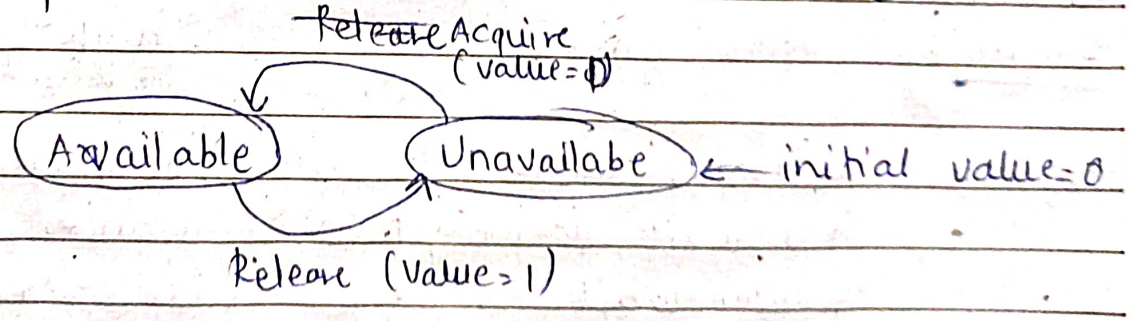
① Counting semaphores -

semaphore count = no. of ~~re~~ available resources



② Binary semaphore

- has value as 0 & 1
- wait operation works only if S = 1
- signal operation succeeds when ~~sem~~ S = 0



* Mutex

- mutex → an object providing Mutual Exclusion between threads.
- ensures that only one thread has access to critical section or data by using operations like a lock & unlock.

- thread having the lock of mutex can use the critical section while other threads must wait till the lock is released
- is a special Binary semaphore that synchronizes the access to shared resources like memory or I/O
- tries to solve 'Producer Consumer Problem'. key is either with producer or consumer

★ Deadlock vs. starvation

	Deadlock	Starvation
Occurs when	none of the process in the set is able to move ahead	process waits for indefinite period of time to get the resource it requires.
A.K.A	Circular waiting	lived lock.
resources blocked by	other process	high priority process.
conditions.	mutual exclusion, hold & wait, no-preemption, circular wait	Uncontrolled mgmt of resources, process priorities, use of random selection
process that can continue	None	all except victim process.
External intervention	Required	Not required.

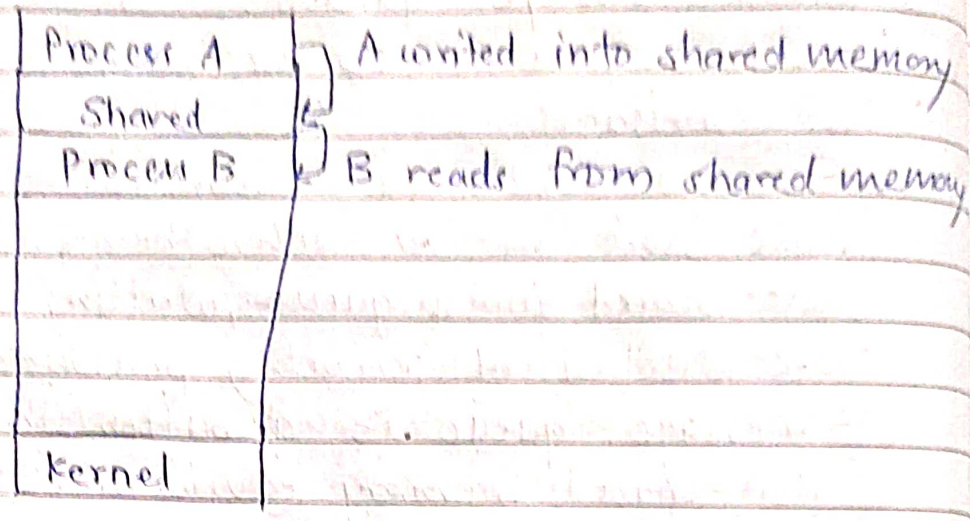
DATE: / /
PAGE NO.:

✱ IPC

① using message queues.
- allows processes to exchange ~~of~~ in the form of messages.
- asynchronous communication by sending messages ~~are store in~~ to each other where the messages are stored in a queue, waiting to be processed & are deleted after being processed.
- message queue is a buffer that is used in non-shared memory environments, where tasks communicate by passing messages.
- Tasks share a common buffer pool.
- unbounded FIFO queue that is protected from concurrent access by different threads.
- many tasks can write into the queue, but only one can read messages at a time.
- • msgget() → creates new queue / existing queue is opened.
  • msgsnd() → new messages are added to the end of a queue.
  • msgrcv() → messages are fetched
  • msgctl() → perform control operation on the message queue.

② using shared memory -
- Shared memory is a memory shared between two or more process
- fasted ipc mechanism
- in shared memory processes can set up structures

& others may read/write on them.
- Semaphores can be used when synchronization is required



- shmget() - create the shared memory segment
  shmat() - used to attach the shared segment with the process's address space.
  shmdt() - detach process
  shmctl() - change ownership of a shared memory. only original owner can do it.

★ Using pipes.
- mechanism by which the output of one process is directed into the input of another process.
- can be viewed as an ordinary file, the system actually manages it as FIFO queue.
- pipe is created using the pipe system call.
- A pipe descriptor has an array that stores two pointers, one for i/p end & other for o/p end.
- operates in one direction only.
- cannot send message to multiple processes.
- read end read irrespective of anything. so insecure.

PAGE NO.:

★ using FIFO
- bi-directional
- ~~atu~~ aka named pipes
- mknod(1) → creates named pipe
  arguments are pathname, mode & dev
  returns zero on success & -1 in case of failure
- mkfifo() → creates a FIFO special file used for named pipe
  - arguments are ~~path~~ file name & mode.
  returns zero on success & -1 in case of failure.