4/4/22 OPERATING SYSTEM

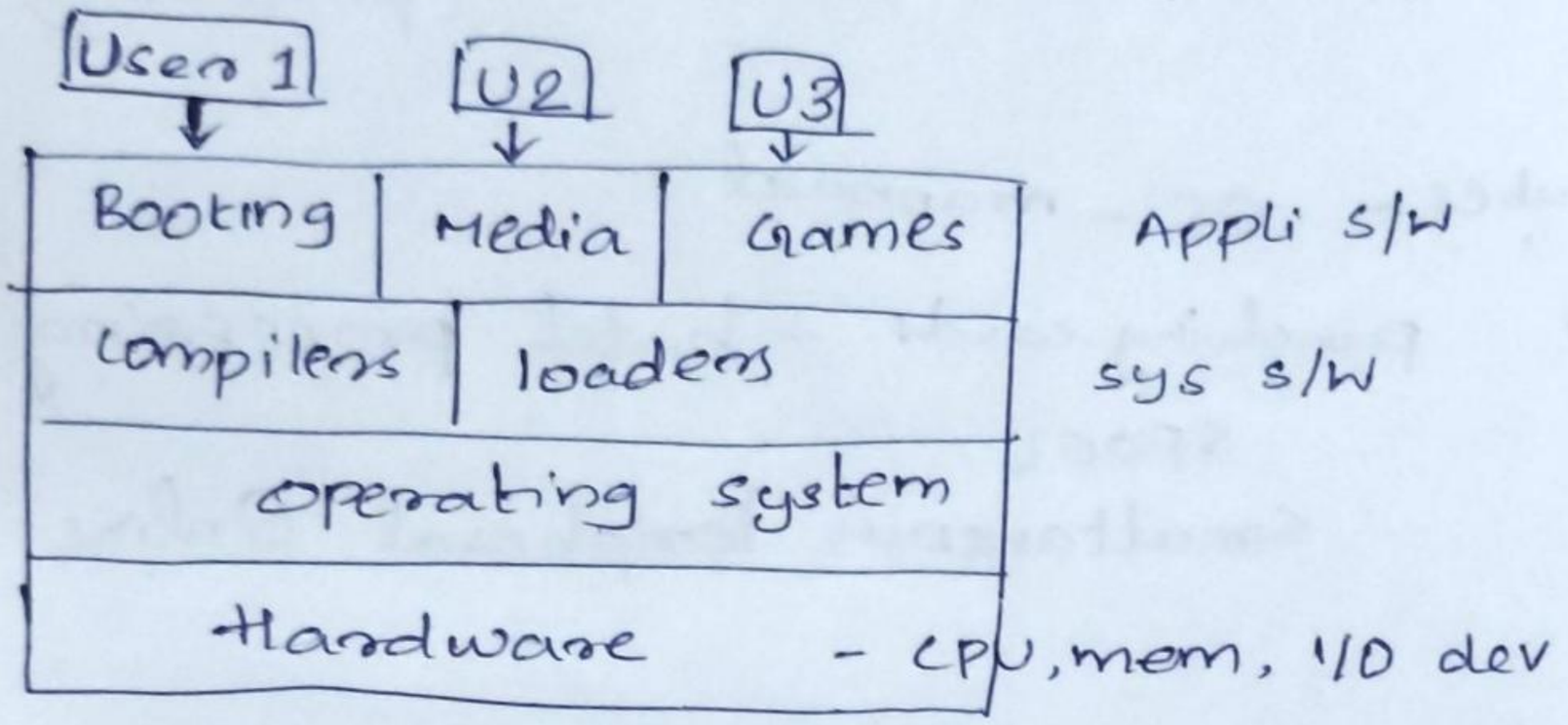
Software

1) Application software eg: MS word, MySQL

a) System s/w eg: Windowss, Linux, Android, compilers, Assembler

3) Utility s/w eg: Defragmentor

OS act as an interface b/w Hardware & User



Processor / processor management
Memory management
I/O or device "
file management
Concurrency management
RTOS (Real Time Operating System)

Two functions of OS

1) User Interface

2) Resource management

OS is machine dependent

5/4/22

OS → UI Extended machine (OS + Hardware)
Abstraction load any numbes of software (coverolap)

Resource manager
- manage resource allocation
- to decide which pogrm is cursently run in CPU
- ensure no conflict b/w datas/resources (protection)

OS involved as

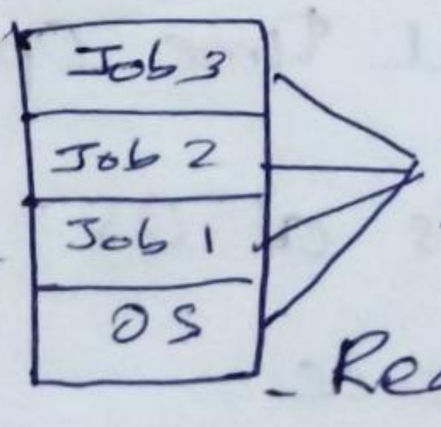
I gen - vacuum tubes - OS - manual.

II gen - transistor punching cards - batch processing.
SPOOL
Simultaneous Periphesal Online

III gen - IC multiprogramming

To manage this multiple progsams OS is ~~used~~ evolving
multiprogsamming - serially progsams are running
Time sharing - multiple programs run simultaneously.
~~Time is~~ with 1 processor
by switching the progsam in a spelific time. (micro

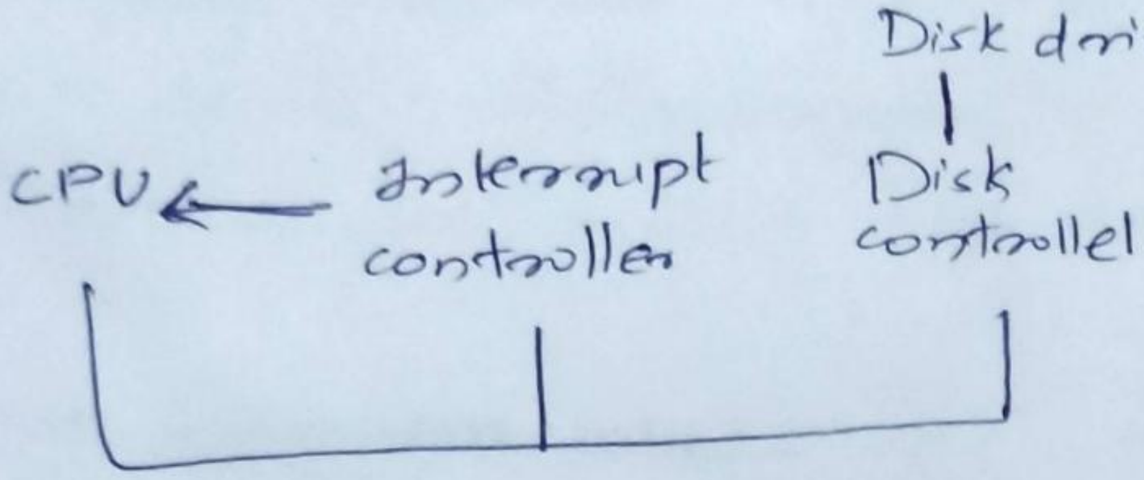
Multi processors - more than one cpu
Distributed sys - many comp at diff locn



IV gen Personal Computers

11/9/22

1 nsec Reg (1KB) depend up on word length
2 nsec Cache - 4 MB
10 nsec main mem - 512 - 2048 MB
10 ms Mag disk - 200 - 1000 GB
100 s mag tape - 400 - 8000 GB



Dispatcher - save ~~as~~ the current progm and pc value and reload next fn who want service.

OS Concepts

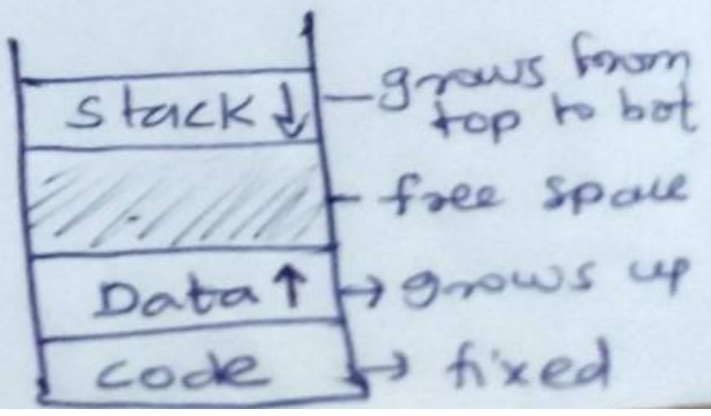
program - set of instructions transform in to desired operation.

Process - A program in execution

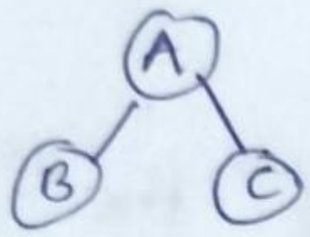
- Have process ID
- In main mem space is allocated to for storing data, instruc ~~called process~~

- Process Controll block → data, code, stack (subroutine) (i/p, intermediate data) } A process has

- store process ID
- mem location
- size of block
- I/O devices used
- files

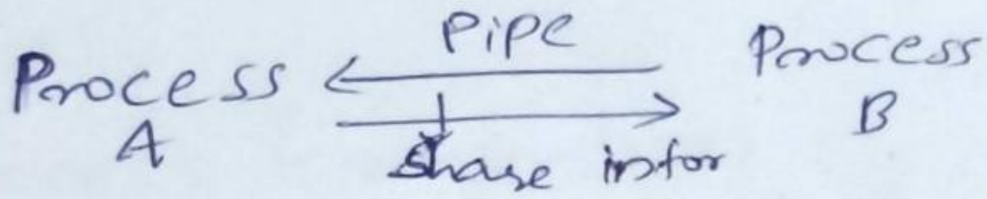


Address space

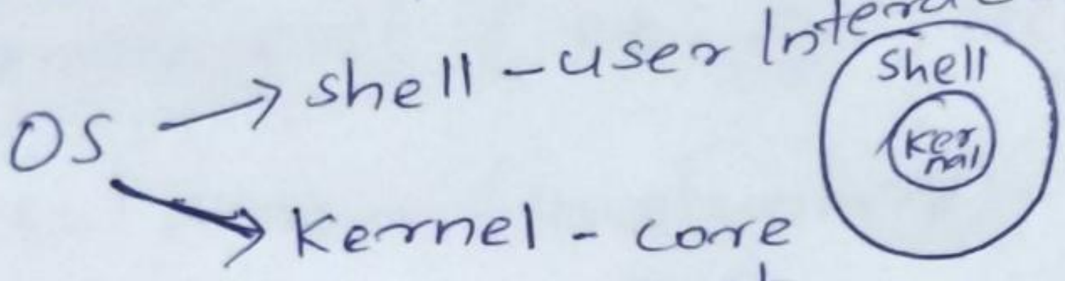


Here process A can create child process B & C
process create child using Fork() process

files
- root directory
- directory
- drives



Modes of operation



eg: dispatcher, file loading, I/O
↓
loading & unloading

system calls
- OS making calls

Pid = fork() create a child process identical to the parent

Pid = wail

OS structure

- A main program that invokes the requested service procedure
- A set of service procedure that carry out the system callels.
- A set of utility procedures that help the service procedures.

Monolythic

Semice

System Calls

Utilities

layered

Each layer has function.

e - THE OS

layer 0 - processor allocation & multiprogramming

layer 1 - Memory & drum management

2 - Operator-process communication

3 - I/O management

4 - user programs

5 - ~~Oper~~ The Operator

It is a hierarchial structure based on function.

Microkernals
- handle interrupts, processes, sheeduling, IPC

22/4/22

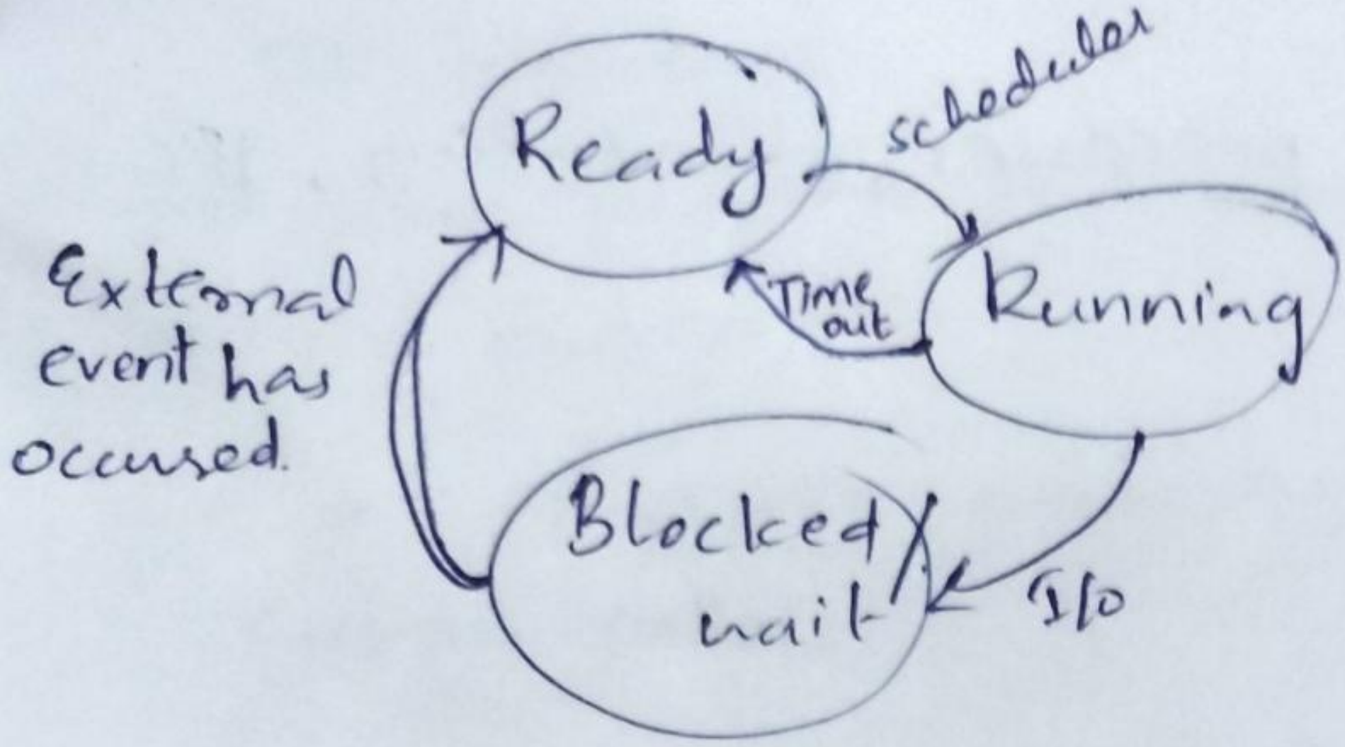
Process and Threads

26/4/22

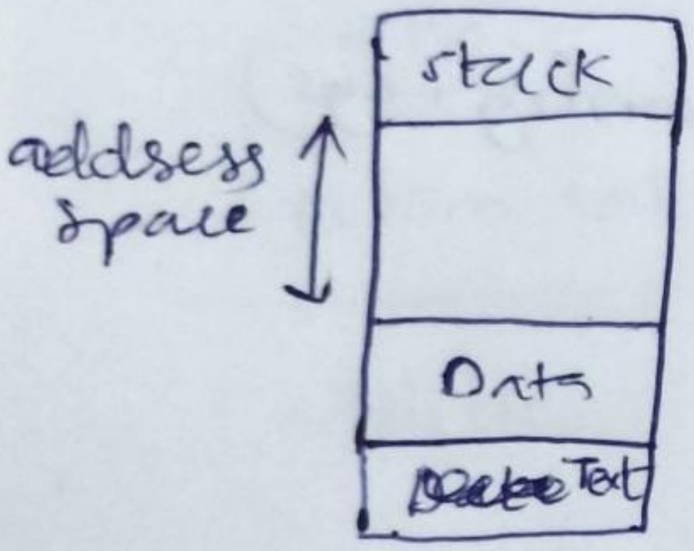
Process creation
- system initialization
- divide the process and execute
- user req to create a new process.
- Initiation of batch job. (autoexec.bat, config.sys)

Process Termination
* Event cause when,
- Normal exit (voluntary)
- error exit (eg: $\frac{a}{0}$, security problm, absence of file)
- fatal error (involuntary) - (hack - other process terminate this process)
- killed by another process (involuntary)

Process States
1) Ready state - processes are in queue, not executing.
scheduler :- selection of process
2) Running state :- when scheduler select it from Ready Queue
3) Blocked/wait :- when proces req an i/o devices it state changed to blocked/wait state



Process table / process control block



decisions are made from the entries.

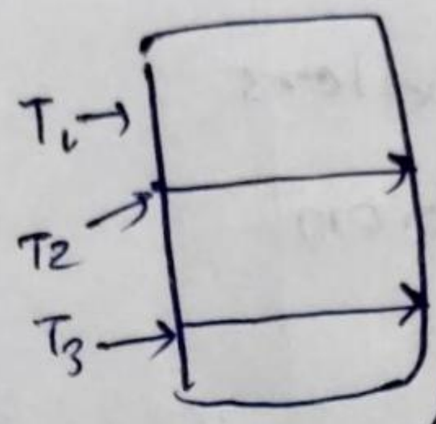
3 levels

25/4/22 Process → have its own address space

A proces is splnt in to 3 threads

keyboard — display — disk

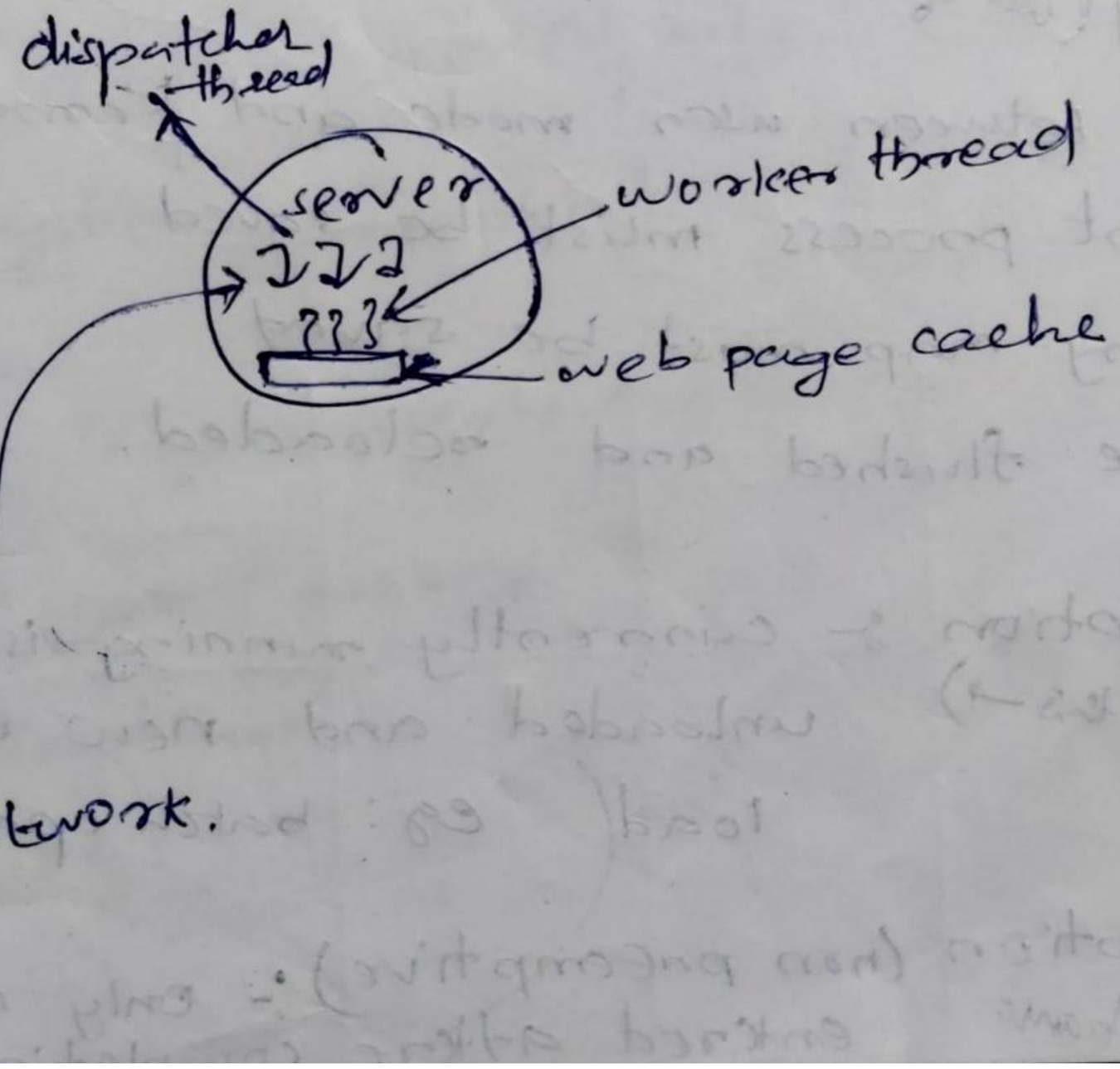
each process done by each thread. Threads sharing the address space



(sharing)
-but value or data or stack segment, (PC val) will be different.

dispatcher thread

assign different works for worker thread



Per process items

- Address space
- Global variable
- Open file
- Child process
- Pending alarms
- signals & signal handlers
- Accounting information

Per thread items

- PC
- registers
- stack
- state

Scheduling

- what to be run next
- Do cpu run to its end or switch b/w diff jobs.

Switching (uses of threads)

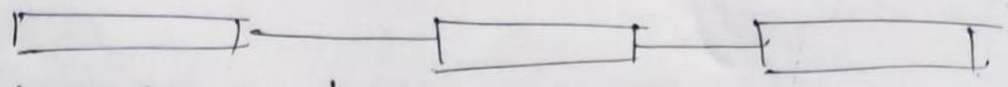
- Switch between user mode and kernal mode
- Current process must be saved
- Memory map must be saved
- Cache flushed and reloaded.

preemption :- currently running is to be unloaded and new one will load (eg: batch opn
(which do this →)

non preemption (non preemptive) :- only new one will load eg: bank entered after completing current

process_behaviour

a)



long CPU burst

b)

short CPU burst

→ when CPU get faster, processes are getting more I/O bounded

→ Goals for diff system

Batch system

- Throughput
- Turnaround time - (time to take complete a process)
- CPU utilization - keep CPU busy.

Interactive system (preemption)

- Response time.
- proportionality - user's expectation

Real-time sys (preemption)

- Meeting deadlines
- Predictability

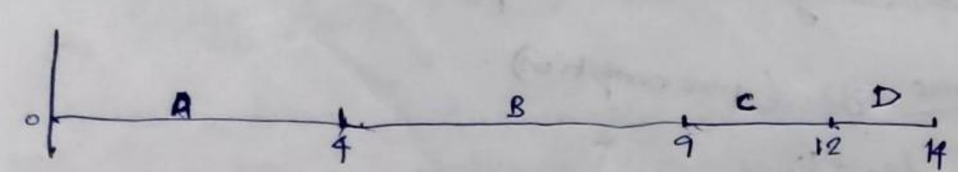
Scheduling in Batch system

- first come - first-served
  - Arrival time
- Shortest job first
  - CPU burst time

process	Arrival time	CPU Burst	start-time	end	Turnaround
A	0	4	0	4	4
B	1	5	4	9	8
C	2	3	9	12	10
D	3	2	12	14	11

33/4 = 8.25

FCFS



26/4

Batch systems - Non preemptive (preempt at same programs) — If one process → I/O all process in batch go for I/O

FCFS

SJF - Non-preemptive

SJN - Preemptive

Process	Arrival Time	CPU Burst	start Time	end time	Turn around	waiting time
A	0	4	0	4	4	0
B	1	5	9	14	13	8
C	2	3	6	9	7	4
D	3	2	4	6	3	1
					6.75	3.7

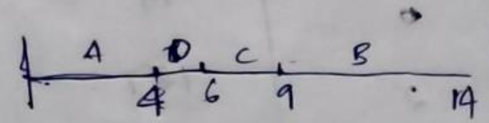
wait time = start - arrival

A: 0, B: 3, C: 7, D: 9

19/4 = 4.75

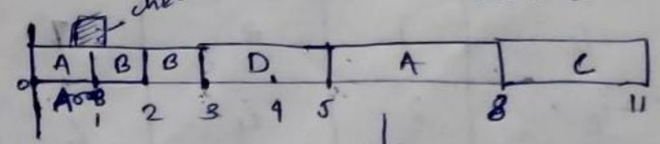
Process	Arri time	burst	start Time	End time	Turn time	waiting time
A	0	4	0	8	8	0
B	1	2	1	3	2	0
C	2	3	8	11	9	6
D	3	2	3	5	2	0
					21/4 = 5.25	6/4 = 1.5

SJF



SJN

checking → total time = end time + checking time



save context of A
kernal turn on

25/4/22

Interactive system
- Response time
- proportionality * User expectations

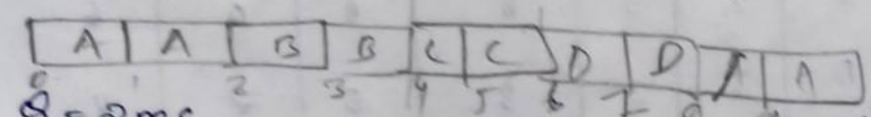
Types

1) Round Robin
- Preemptive

Time Quantum

Q=1ms

Process	CPU Burst	start time	end time	wait time	Turn around
A	5	0	14	0	14
B	7-5	1	17	1	17
C	2-0	2	7	2	7
D	3-1	3	11	3	11
				1.5	12.25

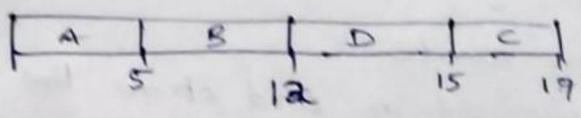


Q=2ms

Process	CPU burst	start time	end time	wait time	Turn around
A 0	3-5	0	14	0	14
B 1	5-7	2	13	2	12
C 2	2-0	4	6	4	6
D 3	1-3	6	15	6/3	13/12.5

- priority (static assign, dynamic assign prio)

- (Non-premtive)



HLW – Round Robin

- Multiple Q

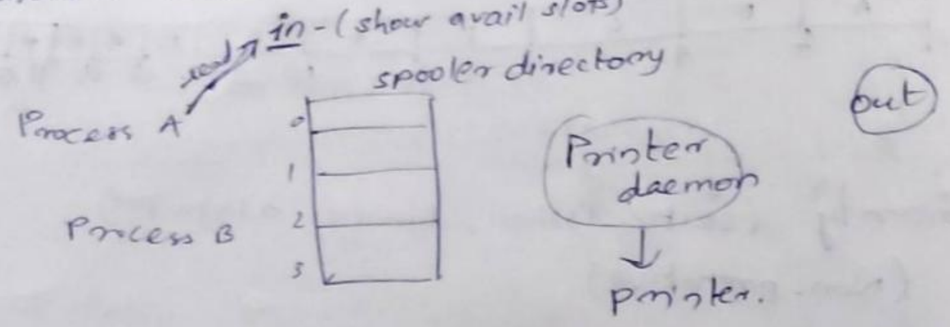
CTSS
- only one process in memory in a time
- swapping is req from disk to mem
- Highest prio will be given to I/O bound

Q-2
- large compute time

Guaranteed
If there are n process,
$\frac{1}{n}$ cycles of CPU

Check
no of cpu cycles obtained / no of cpu cycles entitled

Interprocess Communication (IPC)



in will increment after write
process read the value of in and store the data in to spooler dir. after write value of in ↑
printer read the value of out and print.

Race condition

- 2 or more process compete to access same buffers / reading/writing some shared data
- can't predict when timer will goes off and switching takes place.
- situation which problems arrive is critical section

read (in) storing in spooler dir write (++in)	Critical region part of pogrm where shared mem is accesed

solution :

Mutual exclusion

prohibit more than one process from reading and writing the shared data at the same time.

4 conditions to provide mutual exclusion

1) No 2 process simultaneously in critical region.
2) No assumptions made about speeds or number of CPUs
3) No process running outside its critical region may block another process.
4) No processor must wait forever to enter its critical region.

Critical section

do
    entry sec
    exit sec

To ensure only 1 process is inside the critical reg and in mutual exclusion.

when a process attempt to enter, it get blocked until previous process complete.

Mutual Exclusion

+) Disable Interrupt

→ User interrupt (processes) is not good soln

→ After entering critical seg, disable all interrupt

2) Lock variable

lock - 0 no contest
lock - 1 critical seg is occupied

before entering critical seg made lock to 1
and while leaving make it as 0.

Here mutual exclusion got break.

Strict alternation

One process is not in critical region blocks another process to enter (make busy waiting)

Spin lock

A lock using busy waiting

Peterson's solution to acheive mutual exclusion

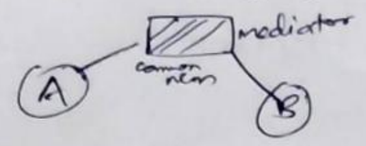
Problems of mutual excl in busy waiting

- not proper soln
- No priority sys (High prio process wait so much to enter low prio process to come out of critical first)

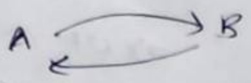
21/22 Interprocess communication

- Synchonisation

1) Shared Memory System



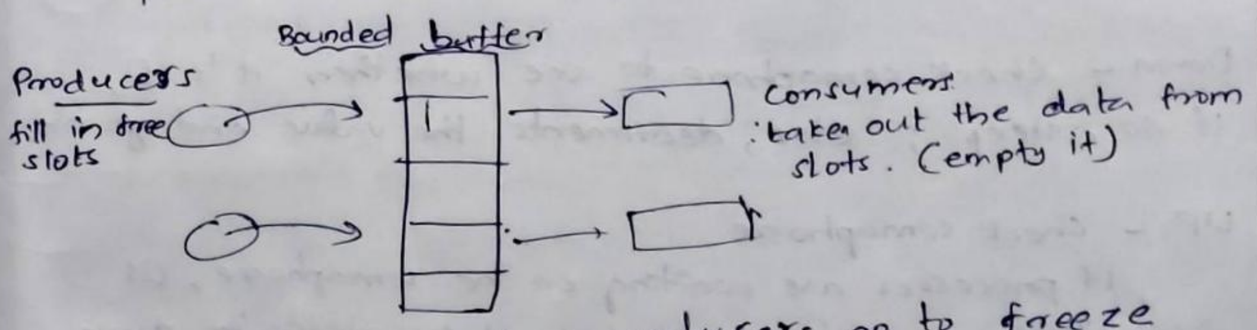
2) Message passing System



Block instead of busy waiting

1. Wakeup sleep

produce consumers problm



After buffer gets full producers go to freeze
when all slots empty consumer get forced

count : no of empty slots / filled slots

* High prio get out of CPU waiting to execute
* low prio inside critical
so both cannot run.

Semafore

-primitive data type

2 opn on semafore

1) DOWN — WAIT — P
2) UP — SIGNAL — V

Atomic Action :- A single, indivisible action
DOWN & UP opn are atomic action, no interpose in b/w the opn takes place.

Down - Check semaphore to see whether it's 0, if so, sleep ; else; decrements the value and go on.

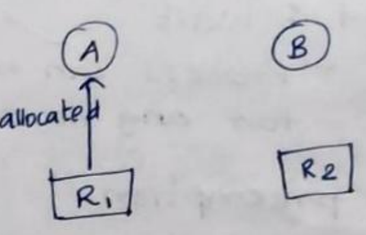
UP - Check semaphore
If processes are waiting on the semaphore, Os will chose on to proceed, and completes its down
- Consider as a sign of number of resources.
- If semphor is 0, incsement.

Binary semaphore (0/1)
Counting semaphore 0 → N (positive)
└ 1) full semaphore - no of filled slots in buffer
2) Empty semaphore - no of empty slots in buffer

mutex - binary val (0/1)
- during down, if N=1 ~~enter criti~~, decrement → critical region (IF N=0 ; no enter to critical reg)
- during up, if N=0, increment → critical reg

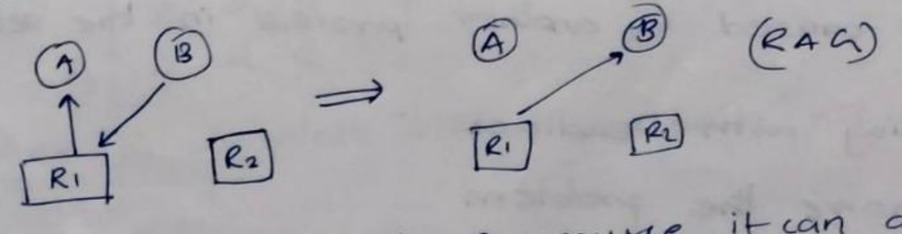
Module - 3
DeadLock

- Processes rep by circle (A)
- Resources rep by [R1]



1. Request ↓
2. If Available ↑ Aquire
3. Use Resource
4. Release

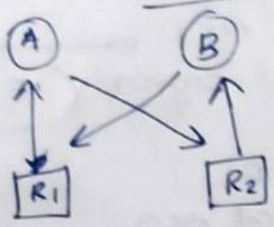
once resource is allocated to a process it won't give to another, ~~dont~~ that time another process request for resource, indicated by [Non preemptable]



If Process A want 2 resource it can aquire
A process can hold more than 1 resource if necessary.

Resource allocation Graph (RAG)

deadlock



Conditions for deadlock

1) Mutual exclusion
   - Resources cannot be shared

2) Hold & wait
   - Process can hold a resource and can request for any no of resource.

3) No preemption
   - Once process is allocated cannot externally remove the allocation.

4) Circular wait

Processes are said to be in deadlock if each process in the set is waiting for an event to occur which is caused by another process in the set.

For Dealing with Deadlocks :

1) Just ignore the problem
2) Detection and Recovery. Let deadlocks occur detect them, take action.
3. Dynamic avoidance by careful resource allocation.
4. Prevention {~~and~~ prevent even 1 condition that results deadlock.}

① Detection from resource allocation graph if a process get repeated they are is deadlock.

② E (4 tapes, 2 plotter, 3 scanners, 1 CDROM) → A = (2 1 0 0)

current allocation matrix

$$C = \begin{matrix} P_1 \\ P_2 \\ P_3 \end{matrix} \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

Request Matrix

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix} \begin{matrix} X \\ X \\ \checkmark \end{matrix}$$

can't allocate processes ~~chances of deadlock~~

chance allocation table

$$C = \begin{matrix} \\ \\ P_3 \end{matrix} \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 2 & 2 & 2 & 0 \end{bmatrix} \qquad A = [0, 0, 0, 0]$$

After P_3 complete it will give back all resources it aquired.

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad A = [2, 2, 2, 0]$$

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{matrix} X \\ \\ \checkmark \end{matrix}$$

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 3 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

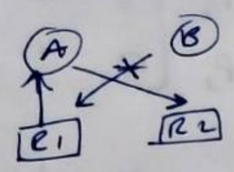
$A = [4, 2, 2, 1]$

$$R = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \checkmark$$

$$C = \begin{bmatrix} 2 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \xrightarrow{r} A =$$

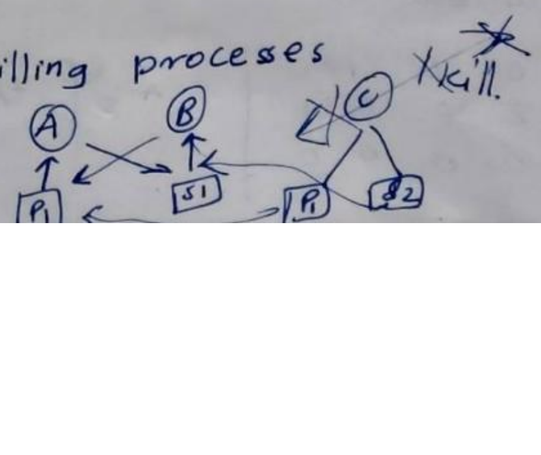
Recovery

- Recovery through preemption
- ~~Recovery t~~



- Recovery through rollback
  1. A is allocated R_1
  2. B is allocated R_2
  3. A req for R_2
  4. B req for R_1 (X) (reason for DL) X

- Recovery through killing processes



20/5/22

Banker's Algorithm – Deadlock avoidance

1) Safe 2) Resource Allocation (Handle resource of many types)

↓

check whether the allocation leads to deadlock in future.

Need matrix = Total requt - Current Allocation

- Before process start we know no of resources.

process req <0 0 1 0> less than available matrix allocation is possible

	Allocation				Need matrix				Available
A	3	0	1	1	1	1	0	0	[0 0 2 0]
B	0	1	0	0	0	1	1	2	
C	1	1	1	0	3	1	0	0	
D	1	1	0	1	0	0	1	0	
E	1	0	0	0	1	1	1	0	

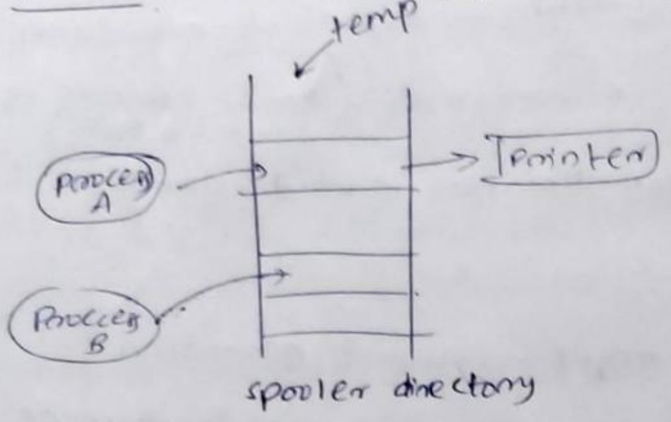
If one process is not fullfilled, it will lead to deadlock

Deadlock Prevention

Avoid any of 4 condition

1) Mutual exclusion
2) Hold & wait
3) no preemption
4) Circular wait

Spooling for mutual Exch buffer



If both processor wants to access printer, first they store their stuff in to a temp buffer called spooler dir, whenever resource get free the printing takes place

2) for hold & wait

↳ After acquire all resource, start the process

↳ only when the process a particular resource it acquire that resource.

3) NO preemption

↳ spooling

4) attack Circular wait

↳ Ask for a resource in a particular order

eg: reading → printing etc. (numbering)

Banker's Algorithm

let n be the no of process in system

let m be the no of resources in system

Available : is a vector of length m

Max : n x m matrix

Allocation : n x m matrix

Need : n x m matrix

Safety Algorithm

To find if a system is safe or not

1. Let work and finish be vectors of length m and n respectively

Initialize work = Available and finish [i] = false for i = 0, 1, 2, ... n-1

2. find an index i such that both

a) finish [i] == false

b) Need$_i$ ≤ work

If no such i exit go to step 4

3. work = work + Allocation

finish [i] = true

go to step 2

4. If finish [i] == true for all i

then system is in safe state.

Resource Allocation Algorithm

Let Request : request vector for process P_i.
If $Request_i[j] = k$, then process P_i wants k instances of resource type R_j

When a request for resources is made by a process the following action are taken.

1) If $Request_i \leq Need_i$; go to step 5
   else raise error condition since process has exceeded maximum claim.

2) If $Request \leq Available$ go to step 3
   else P_i must wait, since resources are not available.

3) Have the system pretend to have allocated the requested resources to the process P_i by modifying

$Available = Available - Request_i$
$Allocation = Allocation + Request_i$
$Need_i = Need_i - Request_i$

If the resulting Resource allocation is <u>Safe</u> then the transaction is approved and process P_i is allocated its resources.

2

Process	current allocation		Max	Allocation	Available	
	R_1	R_2	R_1	R_2	R_1	R_2
P_1	7	2	9	5		
P_2	1	3	2	6	2	1
P_3	1	1	2	2		
P_4	3	0	5	0		

Requirement matrix of P1

R_1	R_2
2	3
1	3
1.	.1
2	0

Allocat (1, 1) to P_3
After P_3 completes
Available (3, 2)
~~Process is safe, continue~~
~~it~~.

Allocation possible in P_4 but can allocate to P_1, P_2 so it is unsafe.

? Process	Allocation			Max			Available	Request		
P_0	0	1	0	7	5	3	3 3 2	7	4	3
P_1	2	0	0	3	2	2		1	2	2 ✓
P_2	3	0	2	9	0	2		6	0	0
P_3	2	1	1	2	2	2		0	1	1 ✓ ✓
P_4	0	0	2	4	3	3		4	3	1 ✓

Allocate to P1

P_1 3 2 2 Available (2, 1, 0)

After P_1 (5, 3, 2)

Allocate to P_3

P_3 2 2 2 Available (~~1, 2, 3~~) (5, 2, 1)

After P_3 (7, 4, 3)

So this can allocate to any of the remaining so this system is in safe.

? Can request (3; 3, 0) by P_4 be granted?

	Allocation		
P_0	0	1	0
P_1	2	0	0
P_2	3	0	2
P_3	2	1	1
P_4	3	3	2

Available

3 3 2

request is

3 3 0

A = 0, 0, 2

If request of P_4 is granted can't allocate to any other P. So request will be rejected

? can request (0, 2, 0) by P2 be granted?

P_2 3 0 2 changed to 3, 2, 2

Available is 3, 3, 2

request is 0, 2, 0

Now,

A = 3, 1, 2

first check if resource is asked (in max) only then allocate. If they not ask no need to allocate for non requested resources.