321 - Operating Systems - Section 04 - SZN -

Shakila.zaman@bracu.ac.bd
{80814}

Lecture 01

OB outline (we will follow)

Linux / Implementation. Case Study

OS Structure, Architecture

registrar
Processing
Secondary Storage (D:, E:, F:, G:)
Primary Storage (RAM, ROM)
main memory

process → secondary → processing → main memory
(every works when overs/finishes, frees from main memory)

advantages • can execute multiple tasks.

apps → base
OS - software
↓ base
hardware

- main memory - will be vacant or not - maintain by OS.
- how/what time it will run, when it will go - by OS.
- intermediary between hardware and software.

software 2 types,
• application,
• system.

OS → mac
↳ windows

✓ To Rise Up ~~Throughput~~ Throughput
✓ User Satisfaction (↑)

**/OS → Single user single processor eg. Globe ? satellite.

→ Batch OS (works are similar)

→ Multi-programming OS.

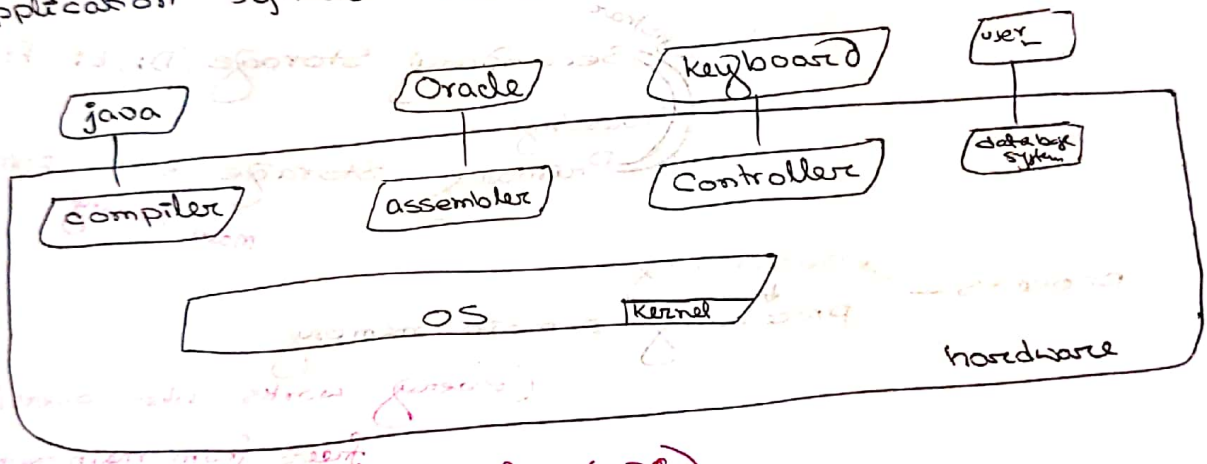
→ Multi-processing OS.

→ Time-Sharing OS / Multi-tasking OS

→ Multi-threading OS.

→ Distributed /.....

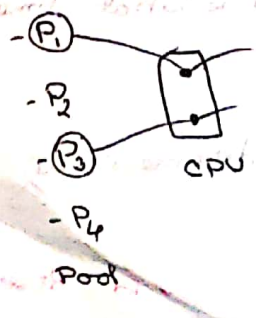
application software OS hardware.



application Program Interface (API)

- Batch OS → multiple work by the instructor, will known as same work/similar work.

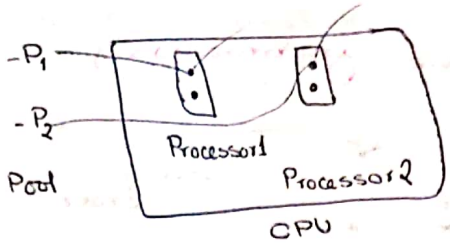
- Multi-programming OS →



At a time, single work executed.

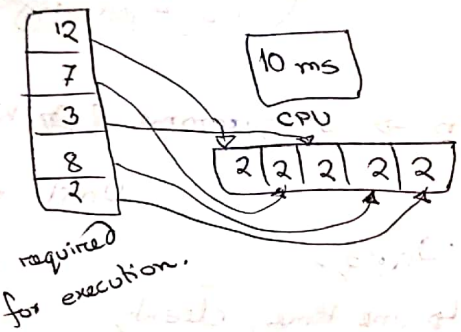
when P_1 over/finishes, then P_3 works.

- Multi-processing OS →



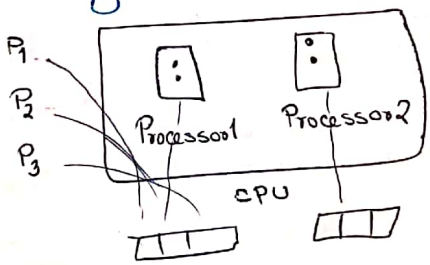
ex, multi-programming, forms multi-processing OS.

- Time-Sharing OS →

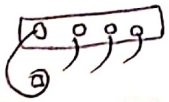


if P_1 takes lots of time, P_2's work will be started, so that no wastage of time occurs.
So, CPU will not stay ideal.

- Multi-tasking OS →



- Multi-threading OS →



no one is dependent on no one.
eg. Games.

Different methods (this must be Identical)
Process
Threading

o Distributed/... OS→

(need tobe synchronous)

eg. fb info of person.

Kernel
↳ Resource

integrated software
↳  all programs

portion of OS, is kernel.
↳ resource

ram → run → programme ⇒ Kernel

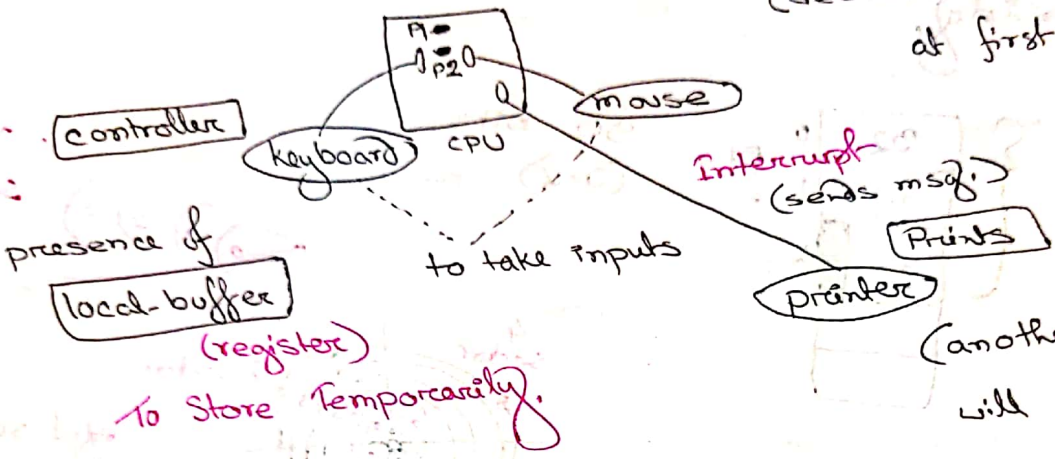
Until switched off, it runs.

Pair to Pair.
↳ one time client, another time server (it becomes) when/what need.

Lecture 01 intro 2

Computer System - Operation,

→ Multi-programming.

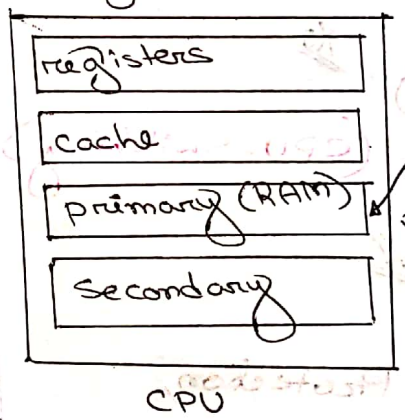


(dedicated for P1, at first priority)

To Store Temporarily.

(another process, it will provide service)

" Storage Structure "



Cache Memory

- regular use.
- frequently used by us, (data - then - stored)

small, costly, speedy than RAM.

Accessability → Sequential access (magnetic Tap)
→ Random access

magnetic tap,

CD

→ serially it goes,

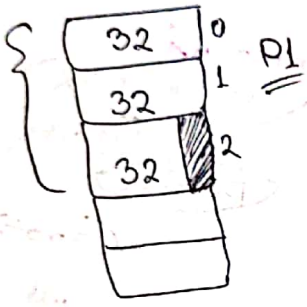
eg. (after one, two will come next),

1
2

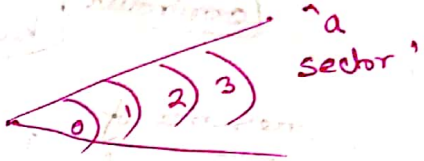
Random Access,

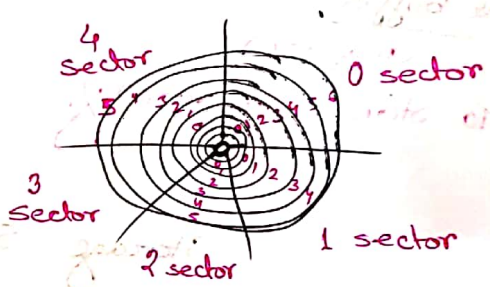
Disk D

Addressing,



P1 = 90 bits (0-2)





eg. 1 sector trace 1

multi-programming ✗

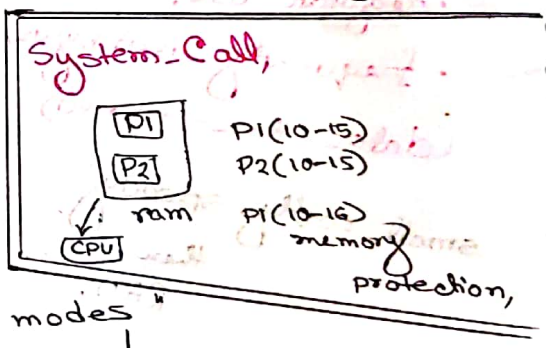
vs

(CPU-switching)

time-sharing ✓

layer-added - storage (high)

Hardware Protection,



hardware → Two modes

user mode (1) — Kernel mode (0)

error in user (system-call) → Kernel

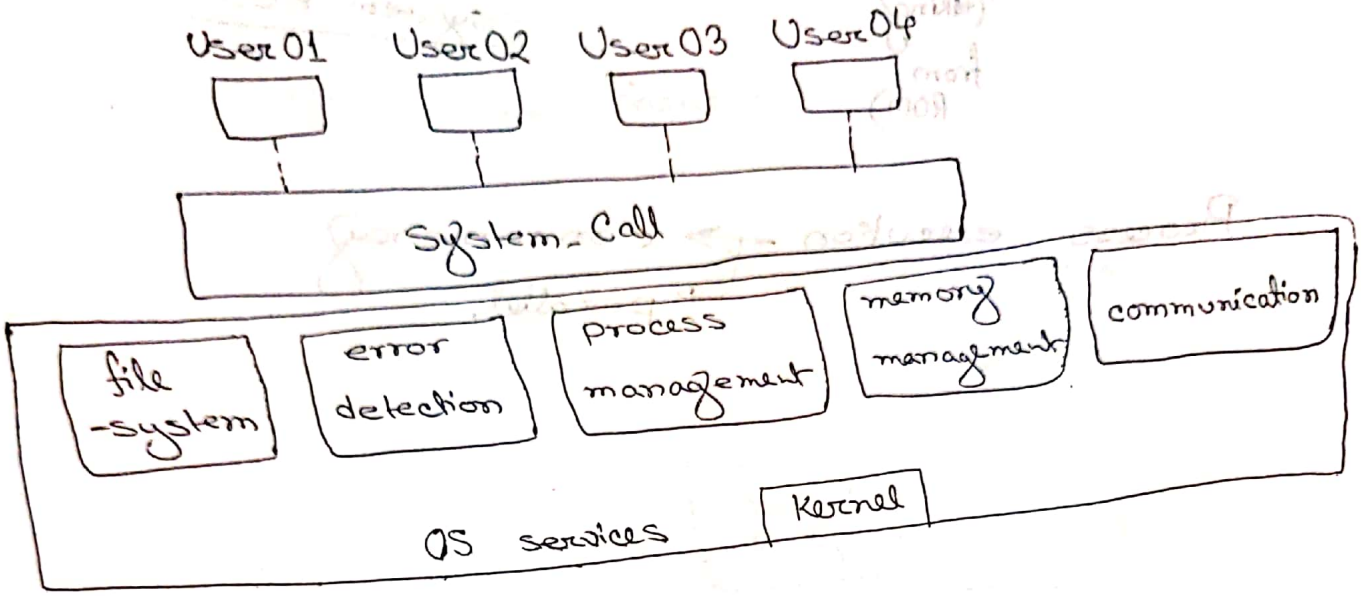
keyboard (controller)

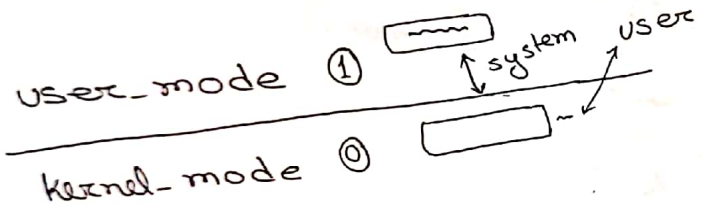
(controlling of OS)

(read)

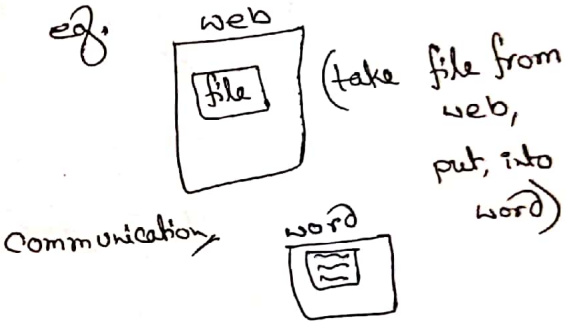
Chapter 02

OS - Structures,

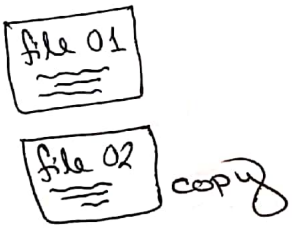




eg.



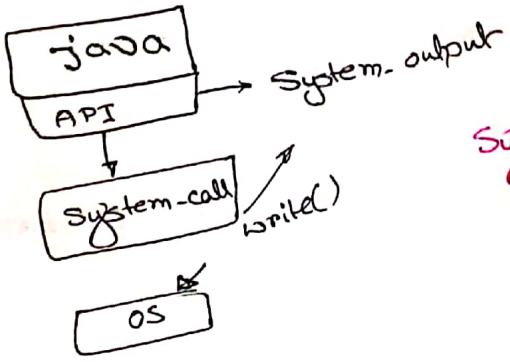
for eg.



system_call,
- create
- write
- error (exit, eboot/ terminate)

[API-system call-OS]

API (application programing interface)



System_Call(user mode ① goes ⓪ kernel mode)

ROM → RAM → hardware
(taking from ROM) (runs)

system boot

Process execution → concurrency.
→ parallel.

Lecture 02

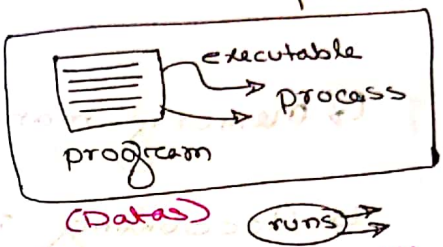
Processes, Chapter 04

Program vs Process,

software ~~ program

↓

process



VLC Player

Process 01

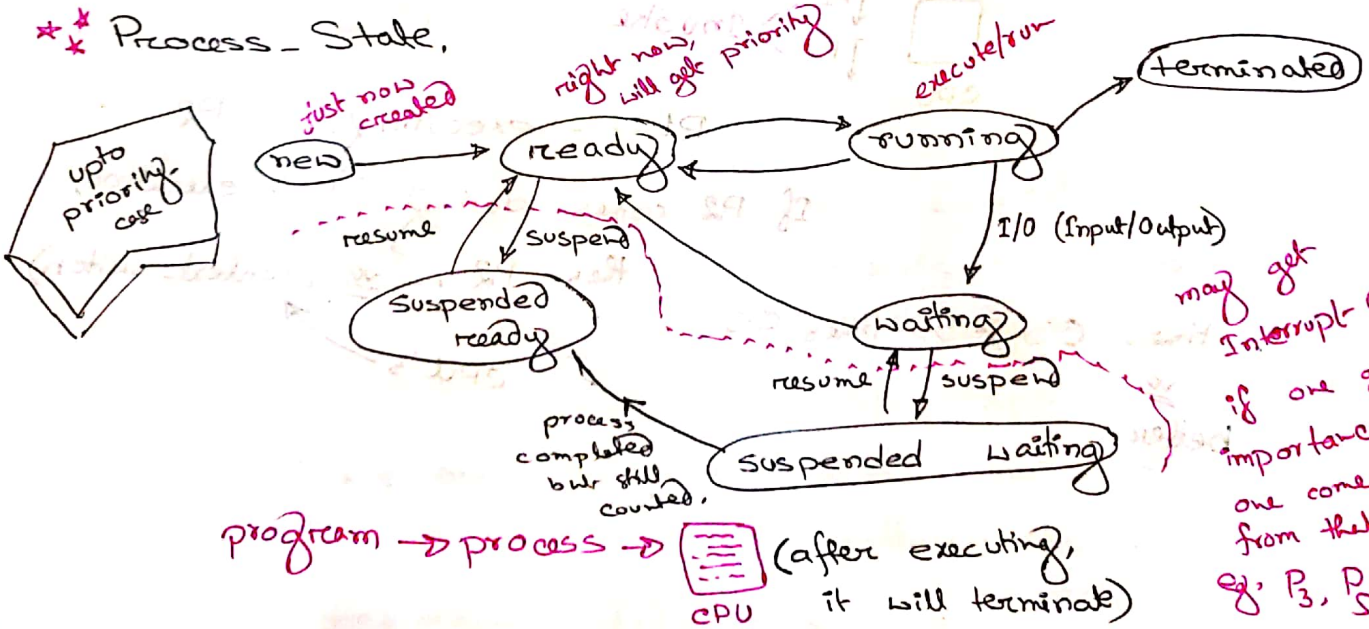
Program Counter — when which will execute, line by line, checks instructions.

Stack — local, global variables.

Data Section — Data, code

heap — Dynamically allocated variable.

Process_State,



program → process → CPU (after executing, it will terminate)

PCB (Process Control Block)

↳ Process State.

↳ Program Counter.

↳ CPU registers.

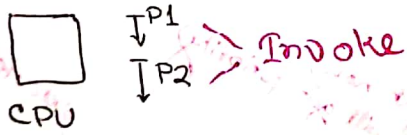
↳ CPU scheduling info. (based on priority)
(which algorith is used.)

↳ Memory-management info.

↳ Accounting info.

↳ I/O status info.

** Context-Switch
(CPU switch Process to Process)

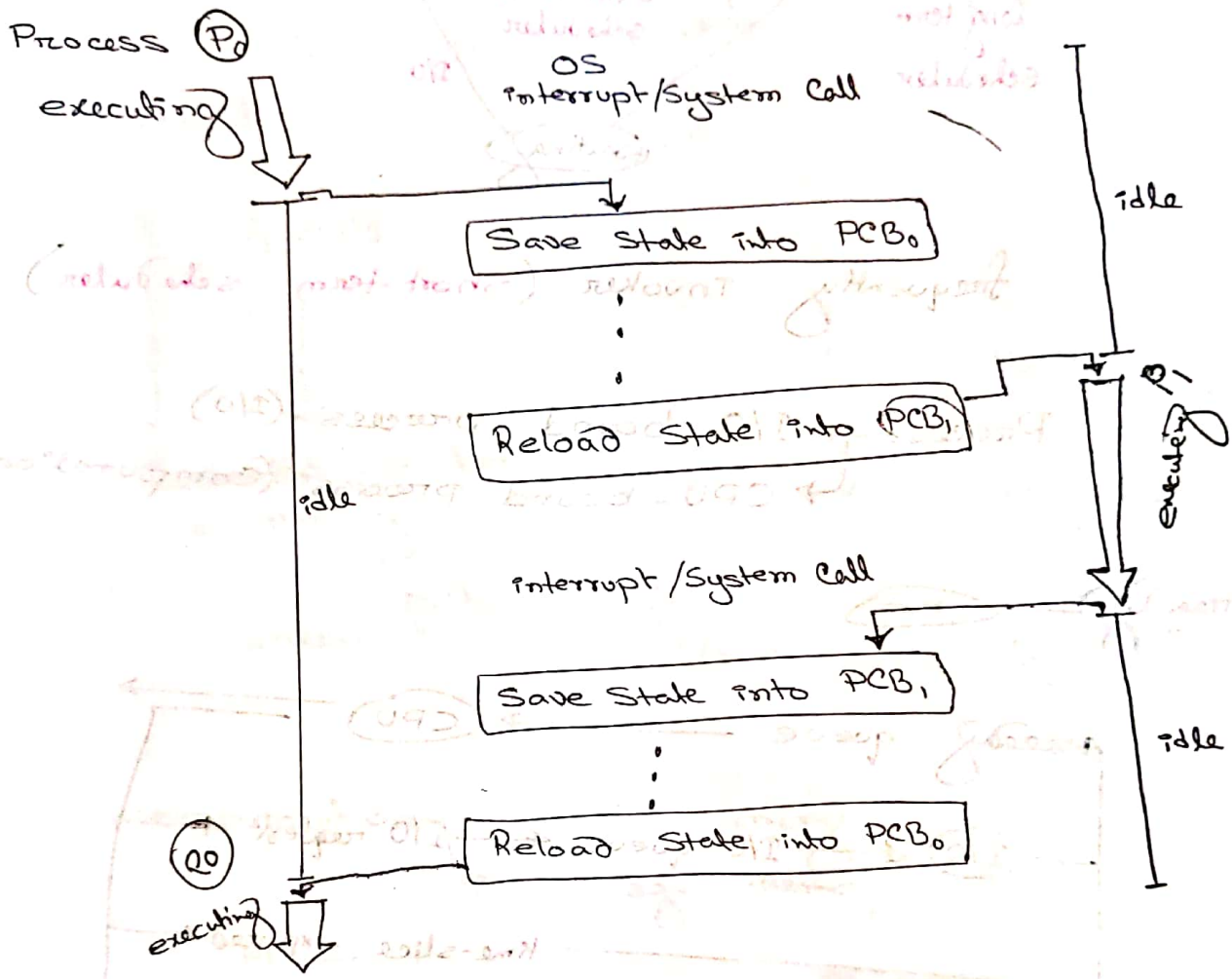


P1 ---- executing ---- P2

If P2 comes during P1's execution,
then P2 -- ≠ (context-switch)
CPU's

5. times C.S > 50 times C.S
≠
better.

Process State → Running
↓
Ready (P1)

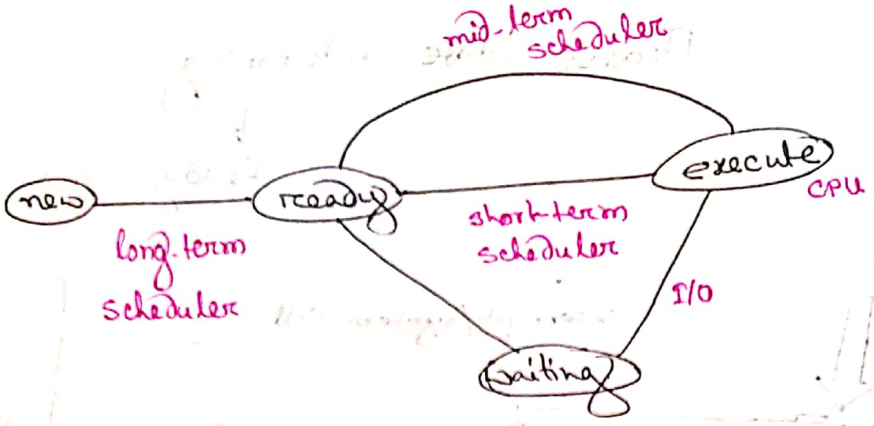


Schedulers,

Long-term scheduler → (job scheduler)
→ controls degree of multi-programming.
→ seconds/mins.

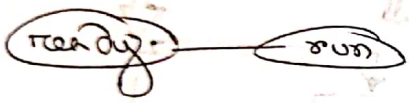
Short-term scheduler → (CPU scheduler)
→ milli-seconds.

Mid-term scheduler → swap-in
→ swap-out.



frequently invoker (Short-term scheduler)

Process → I/O-bound process (I/O)
→ CPU-bound process (computations)



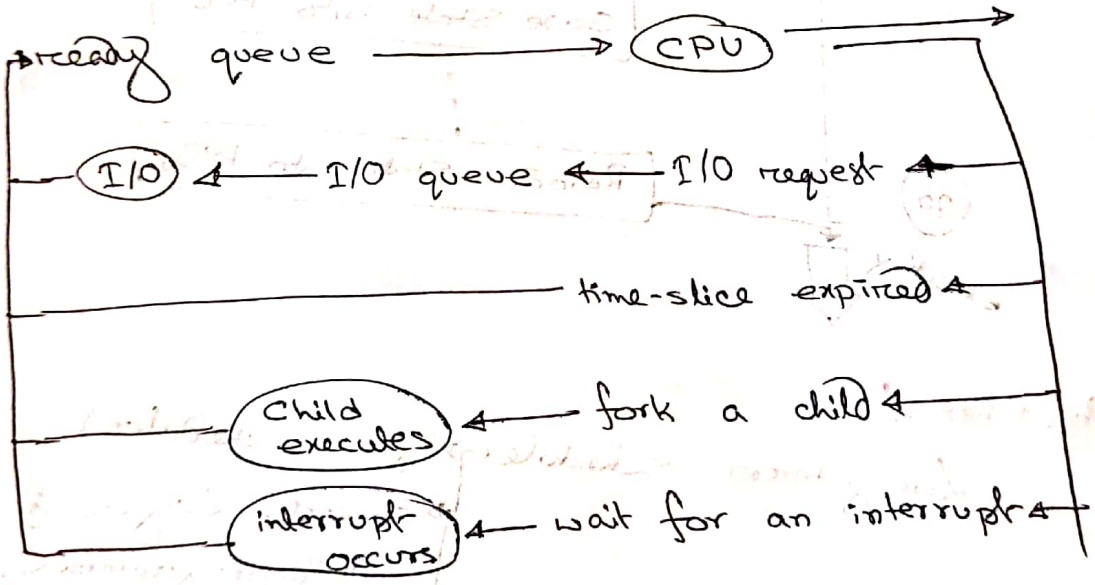
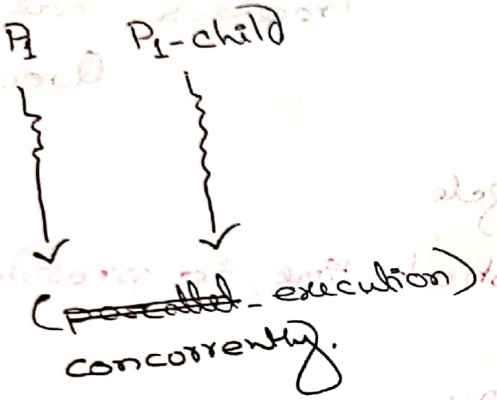
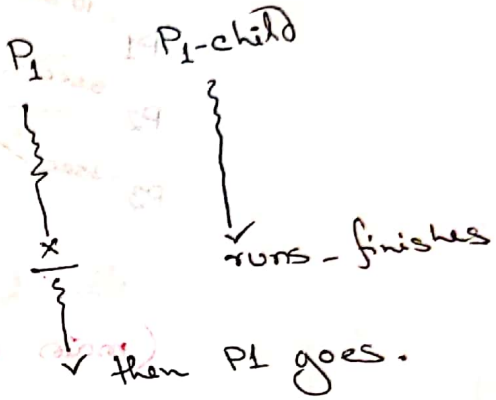


figure- Queuing Diagram

fork()

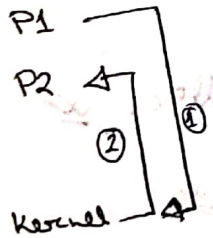
* information Share → partial (subset of P1)
  → no ...
  → full
* execution





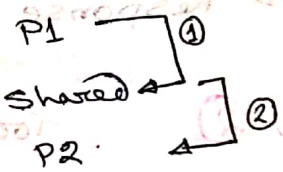
process ⇒ Cascading (kill parent, kill child)
P1-child
P2-child.

** Intercommection → Shared memory
→ message memory



"message memory"

eg. to get result/sum.
→ Short Process.
→ Involve of Kernel.
→ Slower.

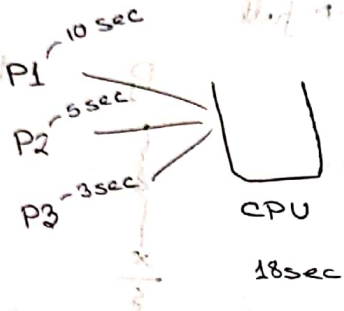


"Shared memory"

eg. to get big result, big file.
→ Big Process.
→ Faster.
→ Absence of Kernel.

Lecture 04

First - come, first - served (FCFS) scheduling.

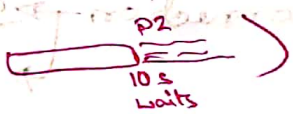


Processes are in Queues.

(main aim, in short time, to provide service)

Scheduling criteria

1. CPU Utilization (Keep CPU, as busy as possible)
2. Throughput (in per sec/execution time, how much process can be executed)
3. Turn around time
4. Waiting time (P1 10s, P2 5s)



5. Response time

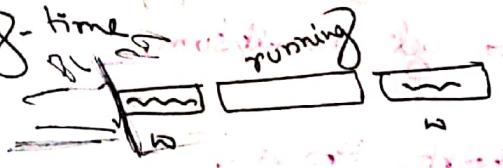
① , ② Too much is better.

③ , ④ , ⑤ Less is better

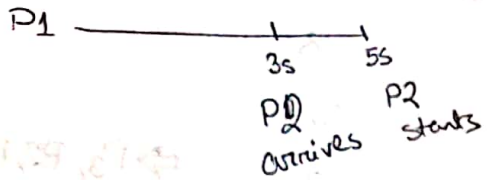
⇨ Response_time

ready → first got the CPU.

⇨ Waiting_time

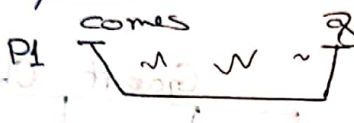


arrived, got the CPU
R_time (0)



→ waiting-time of P2 is 2s.

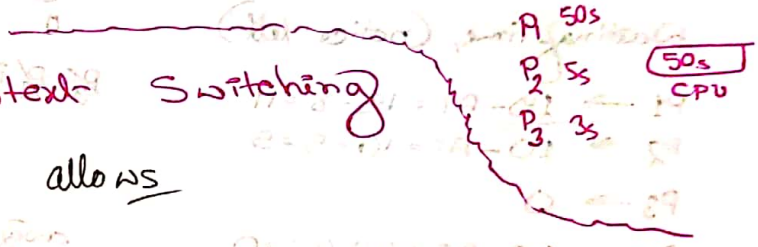
Turnaround time,



Non-Preemptive, XO does not allows

Context Switching

Preemptive, ✓✓ allows



3 maths must

FCFS (non-preemptive) does not allow context switching

example 01

Process	Burst-time (execution-time)
P1	24
P2	3
P3	3

⇒ P1, P2, P3

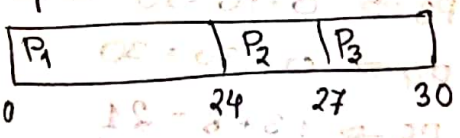
solution:

W.T, (notice left)

P1 → 0
P2 → 24
P3 → 27

Avg → 0+24+27 = 17

Gantt Chart



R.T, (same as W.T)

P1 → 0
P2 → 24
P3 → 27

avg = 17,

T.T, (notice right)

P1 → 24
P2 → 27
P3 → 30

avg → 24+27+30 = 27

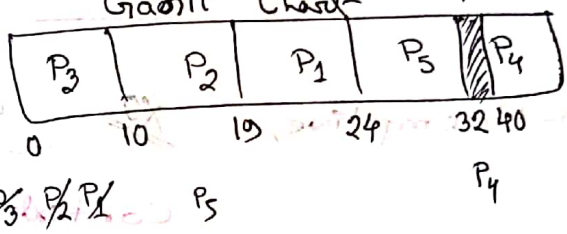
example 02

Process	Burst-time	Arrival-time
P1	5	8
P2	9	5
P3	10	0
P4	30	40
P5	8	11

⇨ P3, P2, P1, P5, P4

Solution:

Gantt Chart



P3 P2 P1 P5 P4

Waiting-time, (notice left)

P1 → 19 - AT = 19 - 8 = 11
P2 → 10 - AT = 10 - 5 = 5
P3 → 0
P4 → 40 - AT = 40 - 40 = 0
P5 → 24 - AT = 24 - 11 = 13

avg. w.t = 5.8

Response-time, (same as w.t)

avg. r.t = 5.8,

Turn around - time, (notice right) { w.t + b.time , or right side - a.time }

P1 → 11 + 5 = 16
P2 → 5 + 9 = 14
P3 → 0 + 10 = 10
P4 → 0 + 30 = 30
P5 → 13 + 8 = 21

avg. t.t = 18.2

Convoy - effect, P1 = 400s, P2 = 10s P3 = 100s

Lec 04

CPU scheduling.

"Shortest-Job-first (SJF) scheduling."

- non-preemptive.
- preemptive.

** SJF,

Process	Burst Time (execution time)
P_1	6
P_2	8
P_3	7
P_4	3

$\Rightarrow P_4 \rightarrow P_1 \Rightarrow P_3 \Rightarrow P_2$

SJF scheduling Chart,

1	P_4	1	P_1	1	P_3	1	P_2	1
0		3		9		16		24

W.T (watch left)

$P_1 \rightarrow 3$

$P_2 \rightarrow 16$

$P_3 \rightarrow 9$

$P_4 \rightarrow 0$

0+3+16+9=

	BT	A.T
P_1	50	70
P_2	120	0
P_3	200	300
P_4	60	30/25
P_5	90	120
P_6	30	0

(burst time kom jeita seita nibo) SJF

non preem

| P_2 | P_1 | P_4 | P_5 | P_3 |
0 120 170 230 320 520

P_1 P_4 P_5 arrived at 120
50 60 90

firstly need to arrive;

P_1	120	0
P_4	60	0

program

same b.T, consider arrival time

Preemptive (Shortest - remaining - time - first)

P_1	15	70
P_2	120	0
P_3	200	300
P_4	60	25
P_5	90	120
P_6	30-25 = 5	0

P_2, P_6
120 30

| P_6 | P_6 | P_4 | P_1 | P_4 | P_2 |
0 25 30 70 85 105 120

next arrive time

(30 এর আগে আছে, P_6)

(next a.t.)

P_2, P_6, P_4
120 5 60
-15 -40
= 105 = 20
-90
= 15

P_5 90

P_1 15

| P_5 | P_2 | P_2 | P_3 |
120 210 300 315 515

CS → 2ta ase

last slot (left) - AT.

W.T

$P_1 \rightarrow 70 - AT = 70 - 70 = 0$

$P_2 \rightarrow$ 300 - 15 - 90 - A.T (paisilo paisilo age.)

$P_3 \rightarrow 315 - AT =$

$P_4 \rightarrow 85 - 40 - AT -$

$P_5 \rightarrow 120 - AT$

$P_6 \rightarrow 25 - 25 - AT = 0.$

Always work on last slot.

last slot.

last slot (left) - age (which big work) - A.T = T.T

T.T

$P_4 = 20 + 60 = 80.$ (u.t + b.T)

right ~~left~~ - a.t.

105 - 25 = 80

R.T

P_2 = Jokhon first CPU te Pase.

105 - AT = 105

$P_4 = 30 - AT = 5$

Process	BT	① AT	② Priority
P_1	10	5	8
P_2	8	3	3
P_3	6	8	0
P_4	5	4	5
P_5	9	10	4
P_6	15	3	8

Priority ← Non-preemp. / preemp.

(process के type-उसे)

Priority (non-pr) scheduling → highest no will highest priority.

P_6	P_1	P_4	P_5	P_2	P_3
0 3	16	21	30	38	44

P_6, P_2, P_4, P_1, P_3, P_5

preemptive SJF (Shortest-remaining time-first)

	BT	AT	Priority
P_1	7	5	8
P_2	8	3	3
P_3	6	8	0
P_4	5	4	5
P_5	9	10	4
P_6	3	3	8

LT

$P_1 = 10 - 2 - 2 - AT =$

(Pr) Shortest-clock.

idle	P_6	P_6	P_6	P_1	P_1	P_1	P_4	P_5	P_3	P_2
0 3	4	5	6	8		10	13	18	27	35 40

P_2 8 — P_6 3-1 =2 -1 =1 — P_4 5 — P_1 7 -2 =5 -2 =3 — P_3 6 — P_5 9

(321)

Round Robin CPU scheduling Algo. (Preemptive)

(Lighter kato sec e execute korbe tar jonno)

Process	B.T	A.T	Quant time
P_1	12	5	
P_2	8	0	4
P_3	10	10	
P_4	6	6	
P_5	8	12	

Timesharing R.R Algor...

* maintain the queue#

Maintain Queue, | P_2 P_2 P_1 P_4 P_3 P_5, P_1, P_4, P_3, P_5, P_1, P_3, P_5

| P_2 | P_2 | P_1 | P_4 | P_3 | P_5 | P_1 | P_4 | P_3 | P_5 |
0 4 8 12 16 20 24 28 30 34 38

| P_1 | P_3 | P_5 |
41 43 44

38−4−4−5 = 25

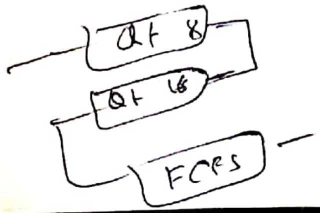
P_1 → wt → 8 − 5 = 3 (AT)   38−8−8

T.T → wt + bt / right − at = 41 − 5 = 36.

R.t → 8 − 5 = 3 (AT)

AT.	0	5	6	10	12
	P_2	P_1	P_4	P_3	P_5

irregular magnitude (multi-feed back algo.)



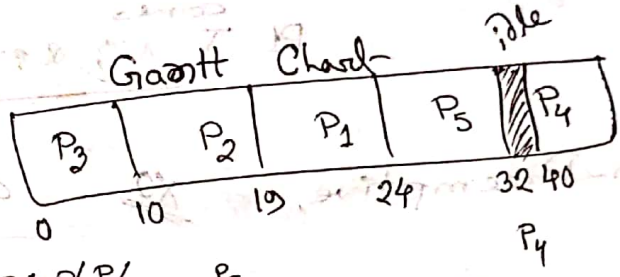
Example 02

Process	Burst-time	Arrival-time
P1	5	8
P2	9	5
P3	10	0
P4	30	40
P5	8	11

⇨ P3, P2, P1, P5, P4

Solution:

Gantt Chart



Waiting-time, (notice left)

P1 → 19 - AT = 19 - 8 = 11
P2 → 10 - AT = 10 - 5 = 5
P3 → 0
P4 → 40 - AT = 40 - 40 = 0
P5 → 24 - AT = 24 - 11 = 13

avg. w.t = 5.8

Response-time, (same as w.t)

avg. r.t = 5.8,

Turn around - time, (notice right)

P1 → 11 + 5 = 16
P2 → 5 + 9 = 14
P3 → 0 + 10 = 10
P4 → 0 + 30 = 30
P5 → 13 + 8 = 21

w.t + b.time
or,
right side - a.time

avg. t.t = 18.2

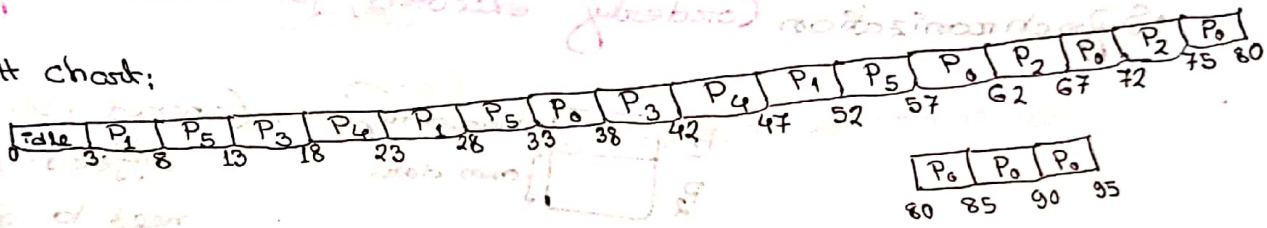
Convoy - effect, P1 = 400s, P2 = 10s, P3 = 100s

example of Round Robin

Process	Burst Time	Arrival Time	Priority	Quant
P_0	30	15	3	
P_1	15	3	6	
P_2	8	40	4	5
P_3	9	7	3	
P_4	10	8	1	
P_5	20	3	5	

maintain Queue: ~~P_1~~ 15, ~~P_5~~ 20, ~~P_3~~ 9, ~~P_4~~ 10, ~~P_2~~ 10, P_5 5, P_0 30, ~~P_5~~ 4, ~~P_4~~ 5, ~~P_1~~ 5, ~~P_5~~ 5, P_0 25, P_2 8, P_0 20, ~~P_2~~ 3, P_0 15, P_0 10, P_0 5, ~~P_0~~ 0

Gantt chart:



So, waiting time (left - work done - at)

$P_0 = 90 - 5 - 5 - 5 - 5 - 5 - 5 - 15 = 45$

$P_1 = 47 - 5 - 5 - 3 = 34$

$P_2 = 72 - 5 - 40 = 27$

$P_3 = 38 - 5 - 7 = 26$

$P_4 = 42 - 5 - 8 = 29$

$P_5 = 52 - 5 - 5 - 3 = 39$

So, average wt = 33.33 sec.

and, turnaround time (right - at)

$P_0 = 95 - 15 = 80$

$P_1 = 52 - 3 = 49$

$P_2 = $ ~~72 - 40 = 32~~ $75 - 40 = 35$

$P_3 = 42 - 7 = 35$, $P_4 = 39$, $P_5 = 54$

So, average tt = 48.67 sec.

response time (First left - at)

$P_0 = 33 - 15 = 18$

$P_1 = 3 - 3 = 0$

$P_2 = 62 - 40 = 22$

$P_3 = 13 - 7 = 6$

$P_4 = 18 - 10 = 8$

$P_5 = $ ~~52~~ $8 - 3 = 5$

So, average rt = 9.83 sec.

Moreover,

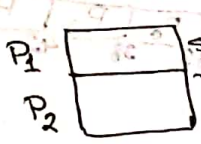
there are some more problem, → Priority Scheduling,
Shortest Job first,

$$\text{Throughput} = \frac{\text{no. of process}}{\text{total CPU time}} = \frac{6}{95} = 0.0632 \text{ sec.}$$

Lecture 05

Process Synchronization

- Synchronization (orderly executed, for shared region)



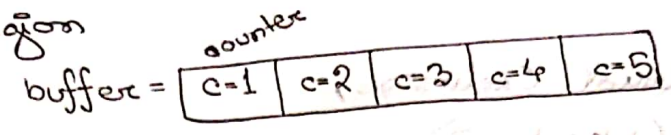
(shared code, different way, we need to give protection)

(other process can't enter)

- Data Inconsistancy (same data, multiple data shared, consistancy ✗)
- Race Condition (same variable, two or three, use or executed)
- Critical Section Problem (also called Shared Region Problem)

⊙ Producer_Consumer Problem

⊙ Shared Region



Case-Producer:

→ buffer-size == counter-size

→ c=c++;

(space is for consumer)

(will produce getting vacant)

Presence of consumer,

→ c==0 ; 0=sleep.

→ c--; (consumed).

// buffer (in) = nextProducer ;

Critical Section

do {

entry section (Yes/No).

critical section

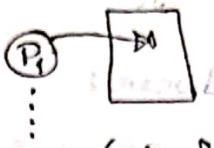
exit section

remainder section (own data)

} while (true);

Solution of Critical Section Problem,

- Mutual Exclusive,
  (obligatory)



if another enters,
synchronization XO

(at first it will finish its task, then other will enter)

- Progressive,
  (independently runs) (checks whether interested or not)

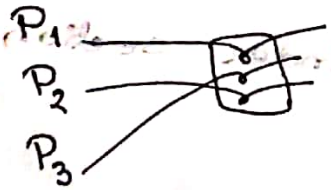
if interested, then that process runs—

eg. P_1 --- P_{100} -- P_1

One enters, other cannot enter during that period of time.

interested -- shared region

- Bounded Waiting,
  (waiting time to be made as "fixed")



aging
(after few moments, (after--after) duration
will introduce priority--)

For solving - critical problem -

Peterson's solution

or

Two Process solution,

(also called)

Flag | F | F |

turn 0/1

i/j

P_0

```
do{
    Flag[0] = TRUE
    turn = 1;
while(Flag[1] && turn==1);
    //critical section
    Flag[0] = FALSE;
    //remainder section
}
While (TRUE);
```

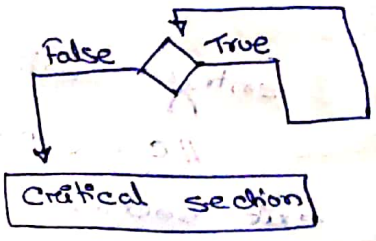
P_1

```
do{
    Flag[1] = TRUE
    turn = 0
while (Flag[0] && turn=0);
    //critical section
    Flag[1] = FALSE;
    //remainder section
}
While (TRUE);
```

00 → 0
01 → 0
10 → 0
11 → 1

Flag → indicates ← Interested are ready or not-,... (True) {boolean-type}

turn → which process will go.... (0/1)



F && T = F

T && T = T

Supports → ✓, ✓, ✗

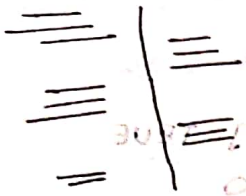
eg-

critical section = 5 lines

for instruction (each) two seconds

after ~~5 lines~~ 6 seconds, another can enter (context-switching)

soln:



Critical Section Problem's Solution;

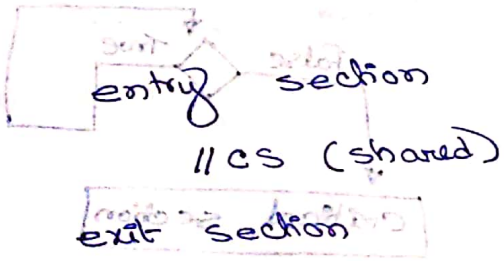
* One soln → Peterson's Soln or Two Process Soln.
* Another soln → Semaphore.

Semaphore:

* Int variable
* binary semaphore → 

another name; (Mutex Semaphore)

* counting semaphore



//RS

Two Operation, S: wait() and signal

(entry) (exit section)

binary

```
s.wait() ..... s.wait(){
                   while (s <= 0); //no op
  //CS               s--;
s.signal() ..... }
                 s.signal(){
                   s++;
                 }
```

// first always will get as One(1). //

// False → enter
True → Loop //

Counting eg. [1] Restaurant ... [40] booked seats shown.

```
wait(semaphore s){
  while (s==0); // (waits > 0) //
```

(Readers-Writers Prob)

* Reading
* Writing

☆ Incase of Reading,
lots of readers,
another reader
allowed.
will restrict writer,

** Continuously busy waiting → spinlock.

busy-waiting (will check _ vacant or not, can enter or not)

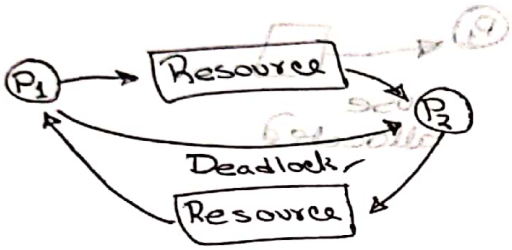
if call of interrupt more

↳ CPU cycling (becoming (damaged))

Lecture 07

Deadlock

own resource – used by another one --



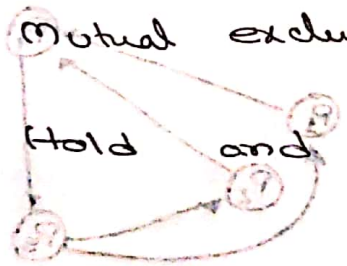
if,
process = 5
Printer = 5
(then much better)

resource – requirements (↑)

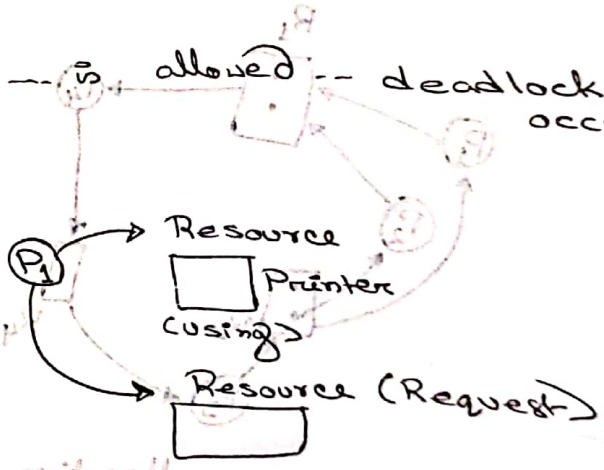
→ for need of concurrent resource, (run)
→ waits for each other.
→ after execution, cannot terminate --.

Characterization

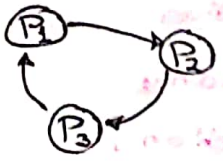
- Mutual exclusive
- Hold and Wait



--- is allowed -- deadlock occur.



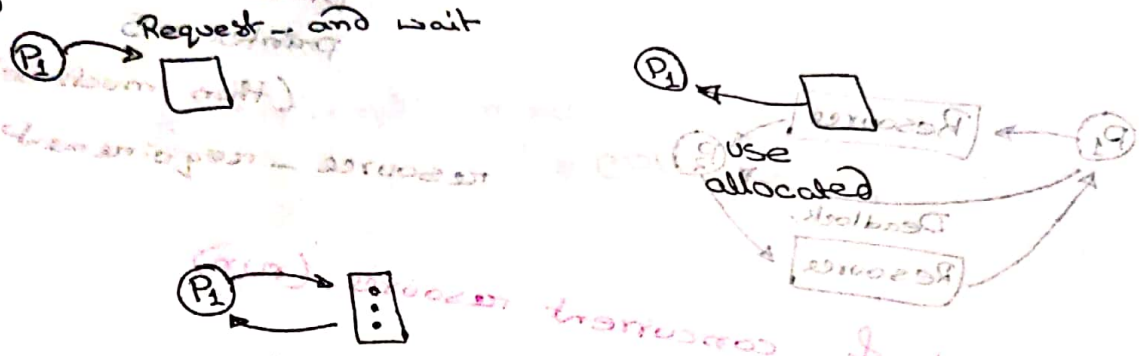
- No Preemption
- Circular Wait



Symbols,

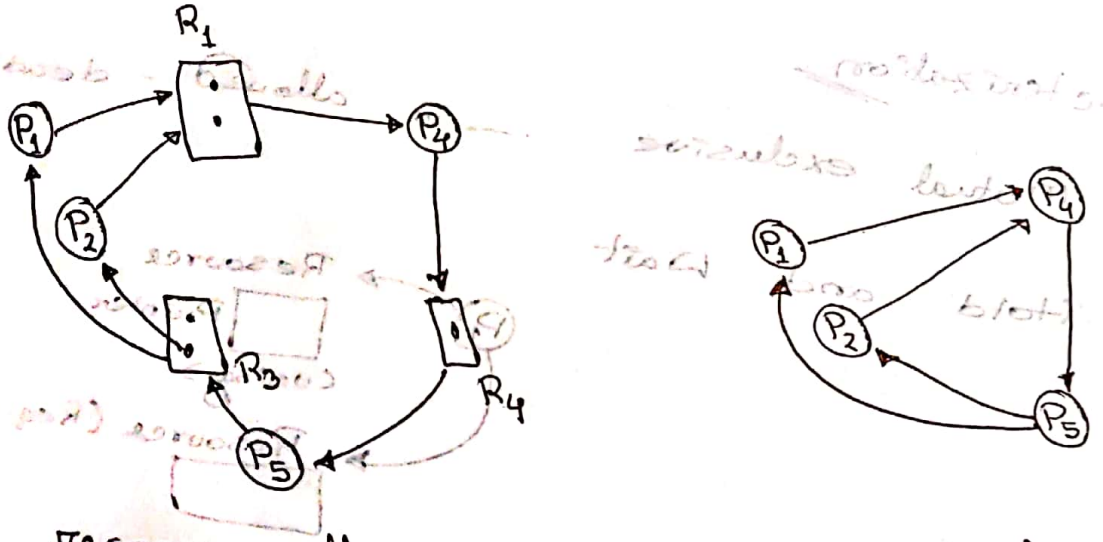
Resource Process Instances, (types)

eg.



after all works done, Release.

eg.

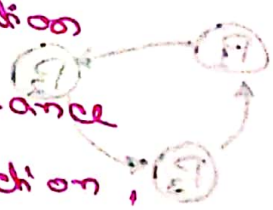


order

P_1
P_2
P_4
P_5

resource allocation graph wait for graph.

→ Prevention
→ Avoidance
→ Detection



Banker's Algorithm

For Quiz 02

CSE 321

→ available (free still now) {instances - total value of a type from allocation}

→ max (number of resources highest needed)

→ allocation (already using)

→ need (number which are more needed) {max - allocation}

instances = available + total.

only if, condition becomes "True",

available = previous available + allocation.

eg. 5 processes P_0 through P_4,

3 resource type, A (10 instances), B (5 instances), C (7 instances).

	allocation	max	Available	Need
p0	0 1 0	7 5 3	3 3 2	7 4 3
p1	2 0 0	3 2 2		1 2 2
p2	3 0 2	9 0 2		6 0 0
p3	2 1 1	2 2 2		0 1 1
p4	0 0 2	4 3 3		4 3 1

at first, P_0,

check, need ≤ available

7 4 3 ≤ 3 3 2

So, P_0 (F)

then, P_1,

check, need ≤ available

1 2 2 ≤ 3 3 2

So, P_1 (T)

So, available = prev. available + allocation

3 3 2 + 2 0 0

= 5 3 2

like this way, Safe Sequence can be obtained.

Detection Algorithm (for whom, deadlock is occuring)
{It's also can be done by banker's algo.}

Now, if "Request" comes,

check: { req ≤ need
       { req ≤ available

available = available - req

allocation = allocation + req

need = need - req

Suppose, above math, ---new, ---(3,3,0) req by P_4 --?

	req	allocation	max	available	need
P_4	(3,3,0)	(002)	(433)	(332)	(431)

at first,

req ≤ need = 330 ≤ 431
(True)

req ≤ available = 330 ≤ 332
(True)

available = available - req = (332) − (330)
= (002)

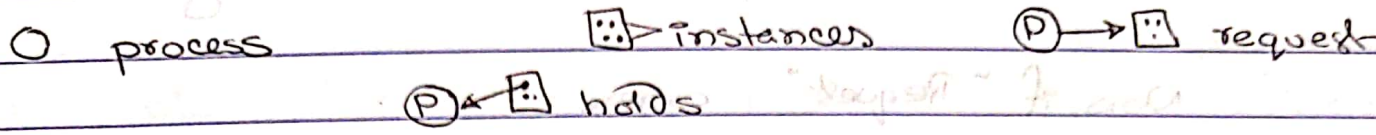
allocation = allocation + req = (002) + (330)
= 332

need = need - req = (431) − (330)
= (101)

// safe sequence?
// request?

deadlock characteristic → mutual exclusive, hold & wait, no preemption, circular wait. {all must be present}

~~Contiguous Allocation~~ →



Deadlock → Prevention
→ Avoidance
→ Detection

Memory Management

17th July, Wednesday

Non-Contiguous Allocation vs Contiguous Allocation

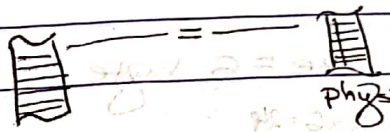
↳ Paging ↳ base, limit

(space free from) (next space)

we can put our file, any where, we will find any free-space

(generated by CPU) (storing in hard ware)

convert logical address into physical address.



[bit numbers must be equal]

page 0 → 1
page 1 → 4
page 2 → 3

0	1
1	4
2	3

1	page 0
3	page 2
4	page 1

logical memory — page table — physical memory.

* suppose page ~~size~~ = 3 bytes
* physical memory = ~~27~~ bytes, 27/3 = 9 frames.

Again, page = 4 byte

logical address = 4×4 = 16

Formula = (frame number × size) + offset

page-number	offset
0	0
	1
	2
1	3
	4
	5
2	6
	7
	8

$(2 \times 4) + 2 = 10$

offset / number

0	12	m
1	13	n
2	14	o
3	15	p

Page 3

logical memory

Page table

0	1
1	2
3	2

0	
4	
8	m
	n
	o
	p
12	

physical memory,

example,

4 pages, page size = 6 byte, physical memory = 36 byte, $36/6 = 6$ frames

$4 \times 6 = 24$ (0-23)

(10, 3) — physical address, (0-35)

paging table

0	2
1	4
2	0
3	1

Formula:

{(frame no × size) + offset}

for 3, $(2 \times 6) + 3 = 15$

for 10, $(4 \times 6) + 4 = 28$

offset	numbers		
0	0	a	page 0
1	1	b	
2	2	c	
(3)	(3)	d	
4	4	e	
5	5	f	
0	6	g	page 1
1	7	h	
2	8	i	
3	9	j	
(4)	(10)	k	
5	11	l	
0	12	m	page 2
1	13	n	
2	14	o	
3	15	p	
4	16	q	
5	17	r	
	18		
	19		
	20		

frame 0	0	
	1	
	2	
	3	
	4	
	5	
	6	
frame 1	7	
	8	
	9	
	10	
	11	
frame 2	12	a
	13	b
	14	c
	(15)	d
	16	e
	17	f
	18	
frame 3	19	
	20	
	21	
	22	
	23	

frame 4	24	e
	25	f
	26	g
	27	h
	(28)	i
	29	j

1400 / 1000 00

1899

** Segmentation (how gives user-view of memory, {user representation}) with help of contiguous. (continuous allocation (smooth)) {we need fitting}

segment table

Segment	limit	base
0	1000	1400
3	1100	3200

segment 0
segment 1
segment 2
segment 3

logical memory space

1400 ⋮ 2399 } segment 0

3200 ⋮ 4299 } segment 3

physical memory

example#

segment	base	limit	
0	219	600	(219 – 818)
1	2300	14	(2300 – 2313)
2	90	100	(90 – 189)
3	1327	500	(1327 – 1826)
4	900	200	(900 – 1099)

{logical memory space}

(segment) (logical address)

(a) 0, 430 (valid)
(219+430)
=649

(b) 1, 10 (valid)
(2300+10)
=(2310)

(c) 2, 500 (invalid)

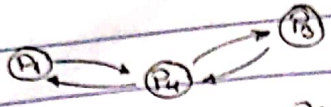
(d) 3, 400 (valid)
(1327+400)
= 1727

(e) 4, 300 (invalid)
(900+300)
= 1199

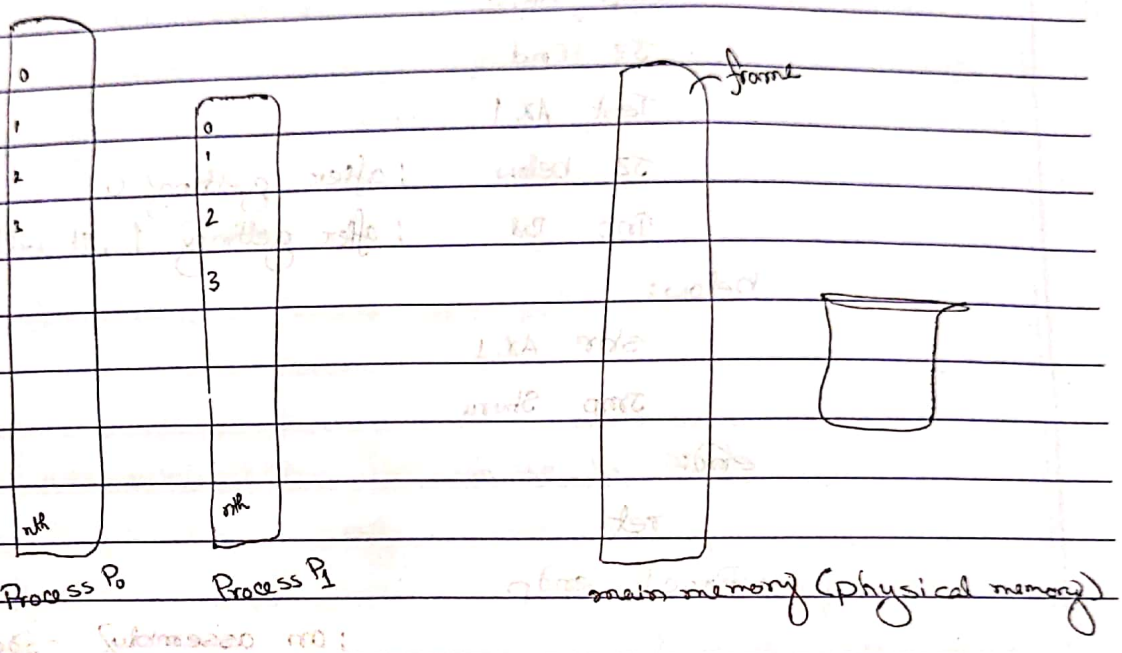
Fragmentation:
i. external (contiguous)
ii. internal (non-contiguous)

8
4
6
13 ✓
2

Operating System



Virtual Memory (logical addressing)



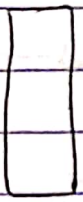
demand paging (currently working page runs into main memory)

→ checks whether it is present in main memory or not.
→ if we didn't find, it known as Page fault.
→ take from secondary memory. (back storage)

fig. steps in handling a Page fault.

call filup, take from back store
(page replacement)
{requirement}

First In First Out Algo.



main memory
3 frame

which comes first, we will replace it by our new one.

3	2	4	0	9	3	0	4	5	2
3	3	3	0	0	0		4	4	4
	2	2	2	9	9		9	5	5
		4	4	4	3		3	3	2
✓	✓	✓	✓	✓	✓		✓	✓	✓

in total (nine)

page fault (less) good ✓

Optimal Page Replacement (Recent future)

"upcoming recent who can come"

"who comes from far"

3	2	4	0	9	3	0	4	5	2
3	3	3	3	3			3	5	5
	2	2	0	0			0	0	2
		4	4	9			4	4	4

in total (eight)

Least Recently Used (LRU)

Optimal →

← LRU

3	2	4	0	9	3	0	4	5	2
3	3	3	0	0	0		4	4	4
	2	2	2	9	9		9	5	5
		4	4	4	3		3	3	2

page fault (nine)