Operating System

Single-user operating system → DOS

Multi-user OS → Windows

Program: Collection of instructions

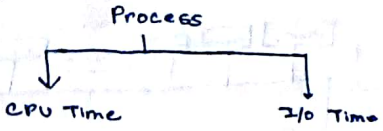
Process: Collection of Programs

Functions

OS → Memory Management
→ File Management
→ Storage Management
→ Protection and Security

Scheduling Algo:

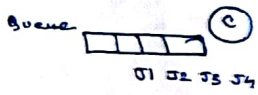
1) FIFO
2) Shortest job first



Types of OS

1. Single user OS → DOS (Not Used)
2. Batch OS (Not Used)
3. Multiprogramming / Time sharing
4. Multitasking -
5. Multiprocessing
6. Real Time OS etc.

2)



Batch OS

- Job which enters 1st is executed first.
- Not Efficient and not interactive

3)
- CPU never sits idle.
- CPU time is shared between various processes.
- Extension of the Batch OS

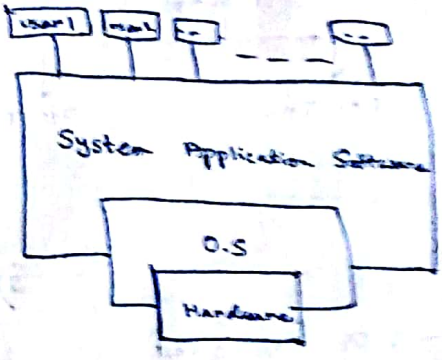
4) pre-emptive

5)

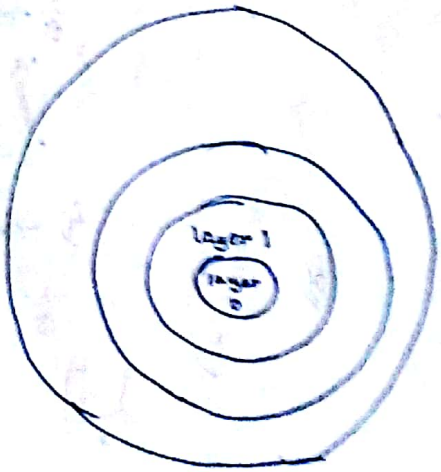
- The jobs are independent
- less costly than multi-programm
- Increases reliability, if one processor doesn't function, then others take that

6) Used in military operations or weather forecast etc.

- Booting
  - Cold Booting
  - Warm Booting
- Bootstrap Loader



OS



1) Core part / Kernel
2) Shell / Command Interpreter

mode → 0 ⟶ Kernel Mode

mode → 1 ⟶ User Mode

Types of System Calls

	Windows	Unix
1. Process Control	CreateProcess() exitProcess() WaitForSingleObject()	fork() exit() wait()
2. File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
3. Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
4. Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
5. Communication	CreatePipe()	pipe()
6. Protection	SetFileSecurity() InitializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

Lect 07

Process: A program in execution.

- Heap → Allocated memory during run-time
- Stack → Func Variables

- Process Control Block (PCB)
- Context Switch
- Scheduler
- Swapping

Scheduling Criteria

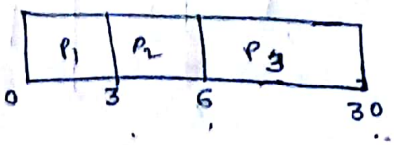
1) Throughput
2) Response Time
3) Turnaround Time
4) Waiting Time

1. FCFS (First Come First Serve)
2. SJF (Shortest Job First) → Min Avg. Waiting Time
3. RR (Round Robin)
4. Multiple Feedback queue etc.

FCFS

Process	Burst Time
P_1	3
P_2	3
P_3	24

Gantt Chart



1) Non-preemptive scheduling
2) Non-preemptive scheduling.

Avg Waiting Time

$$T = \frac{0+3+6}{3} = 3$$

(P_1, P_2, P_3)

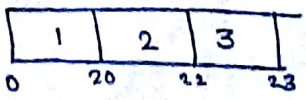
1. Throughput
2. Response Time
3. Turn Around Time
4. Waiting Time

P. No.	Arrival Time AT	Burst Time BT	Completion Time CT	(CT - AT) TAT	waiting time (TAT - BT) WT
P_1	0	4	4	4	0
P_2	1	3	7	6	3
P_3	2	1	8	6	5
P_4	3	2	10	7	5
P_5	4	5	15	11	6

First Come First Serve (FCFS) → Non-preemptive

Convoy Effect (Disadvantage)

P.NO.	AT	BT	CT	TAT	WT
1	0	20	20	20	0
2	1	2	22	21	19
3	2	1	23	21	20

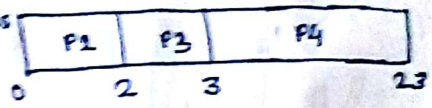


$$AWT = \frac{0+19+20}{3} = \frac{39}{3}$$

$= 13$ ms

P.NO.	AT	BT	CT	TAT	WT
P1	1	20	23	22	2
P2	0	2	2	2	0
P3	0	1	3	3	2

$\therefore$ AWT $= \frac{2+0+2}{3} = \frac{4}{3} = 1.33$ ms



Shortest ~~Job~~ first (SJF)

SJF
- non-preemptive
- preemptive (SRTF)

Non-preemptive

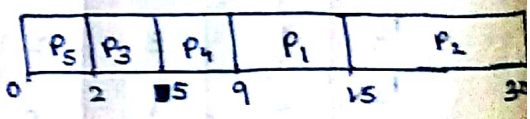
(CT-AT) (TAT-BT)

P.NO.	AT	BT	CT	TAT	WT
P_1	0	6	15	15	9
P_2	0	15	30	30	15
P_3	0	3	5	5	2
P_4	0	4	9	9	5
P_5	0	2	2	2	0

$\therefore$ AWT $= \frac{9+15+2+5+0}{5}$

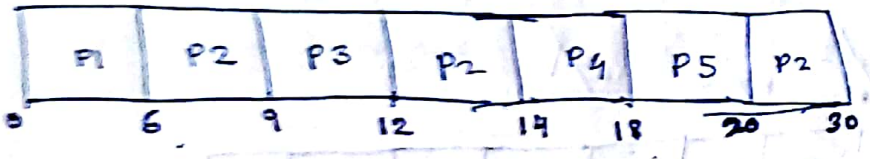
$= \frac{31}{5}$

$= 6.2$ ms



Shortest Remaining Time First (SRTF)

Process ID	Arrival Time	Burst Time	Priority
P1	0	6	1
P2	3	15	1
P3	9	3	1
P4	14	4	1
P5	17	2	1



	AT	BT	CT	TAT	WT	Response Time
P_1	0	6	6	6	0	0
P_2	3	15	30	27	12	6-3=3
P_3	9	3	12	3	0	0
P_4	14	4	18	4	0	0
P_5	17	2	20	3	1	(18-17)=1

- Use SRTF To Schedule —

	AT	BT	CT	TAT	WT	RT
P_1	0	7	19	19	12	0
P_2	1	5	13	12	7	0
P_3	2	3	6	4	1	0
P_4	3	1	4	1	0	0
P_5	4	2	9	5	3	3 (7-4)
P_6	5	1	7	2	1	1 (6-5)

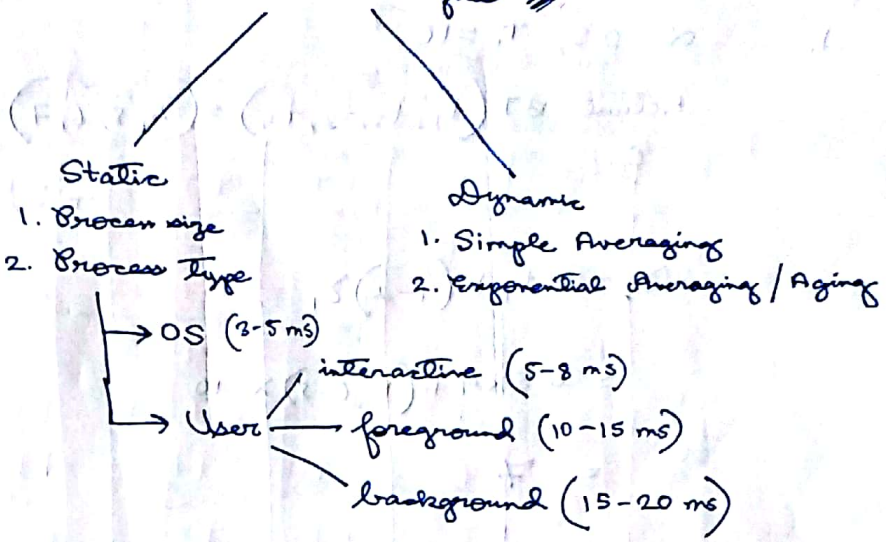
$P_1 = 6$
$P_1 = 5$

$P_1 = 6$ $P_3 = 2$ $P_6 = 1$
$P_2 = 4$ $P_5 = 2$

- Advantages —
  1) more CPU time occupied
  2) Lowers Turnaround and Waiting Time

- Disadvantage —
  1) Starvation Problem

Prediction Techniques



1. Simple Averaging:

- Given n processes $(P_1, \ldots, P_n)$
- Let t_i be the actual Burst time of process P_i.
- Let T_i be the predicted Burst Time.

$$T_{n+1} = \frac{1}{n} \sum_{i=1}^{n} t_i$$

2. Exponential Averaging:

α → Smoothening factor

$0 \leq \alpha \leq 1$

$$\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n \quad \ldots\ldots (1)$$

$$\tau_n = \alpha t_{n-1} + (1-\alpha)\tau_{n-1} \quad \ldots (2)$$

Putting (2) in (1),

$$\tau_{n+1} = \alpha t_n + (1-\alpha)\alpha t_{n-1} + (1-\alpha)^2 \tau_{n-1}$$

$$= \alpha t_n + (1-\alpha)\alpha t_{n-1} + (1-\alpha)^2 \alpha t_{n-2} + (1-\alpha)^3 \tau_{n-2}$$

~~$\alpha = 0.5, T_0 = 10$~~

$\alpha = 0.5, \quad T_1 = 10$

Actual BT $(t_1, t_2, t_3, t_4) = (4, 8, 6, 7)$

τ_5

$T_2 = \alpha t_1 + (1-\alpha)\tau_1$

$\tau_2 = \alpha t_1 + (1-\alpha)\tau_1$

$= 0.5 \times 4 + (1-0.5) \times 10$

$= 7$

$\tau_3 = 0.5 \times 8 + (1-0.5) \times 7$

$= 7.5$

$\tau_4 = 0.5 \times 6 + (1-0.5) * 7.5$

$= 6.75$

$\tau_5 = 0.5 * 7 + (1-0.5) * 6.675$

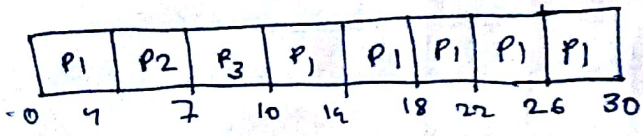
$= 6.875$

Round Robin Algorithm (pre-emptive)

Time quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

Gantt Chart —

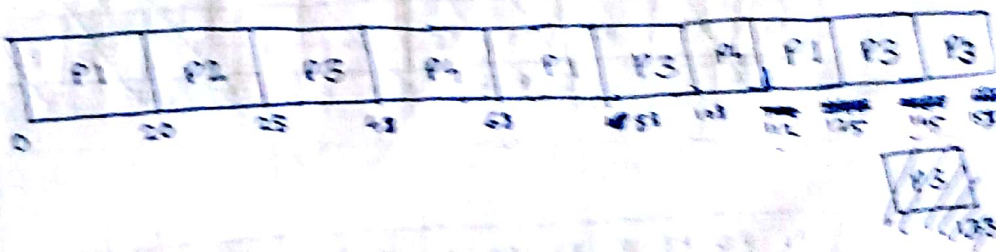


	CT	WT
P1	30	(10-0)=10
P2	7	(3-0)=3
P3	10	(13-2)=11

Example

Time quantum = 20

	AT	BT
P1	0	53
P2	0	8
P3	0	68
P4	0	24



Waiting Time

$P_1 = (0-0) + (68-20) + (112-88) = 72$

$P_2 = (20-0) = 20$

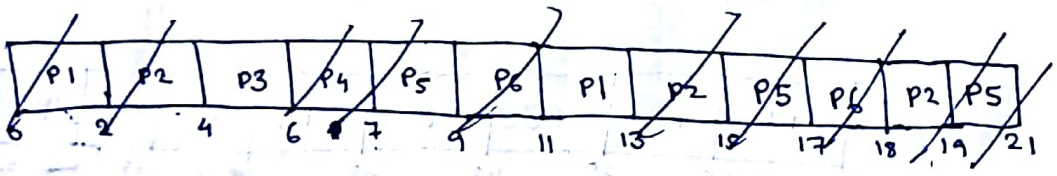
$P_3 = (28-0) + (88-48) + (125-108) = 85$

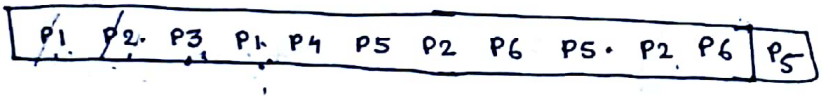
$P_4 = (48-0) + (108-68) = 88$

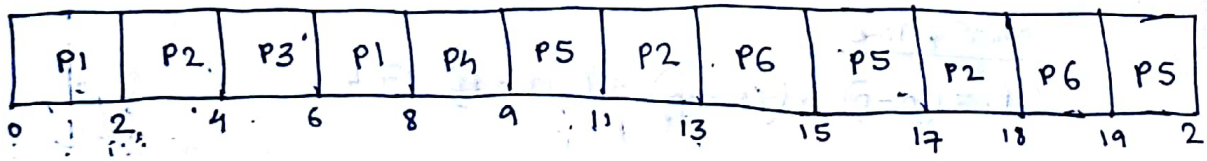
2.

Time Quantum = 2

PNo	AT	BT	CT	TAT	WT
1	0	4	8	8	4
2	1	5	18	17	12
3	2	2	6	4	2
4	3	1	9	6	5
5	4	6	21	17	11
6	6	3	19	13	10







$$\therefore \text{Avg. Turnaround Time} = \frac{8+17+4+6+17+13}{6}$$

$$= \frac{53}{6} = 8.8$$

$$\therefore \text{Avg. WT} = \frac{4+12+2+5+11+10}{6}$$

$$= \frac{44}{6} = 7.3$$

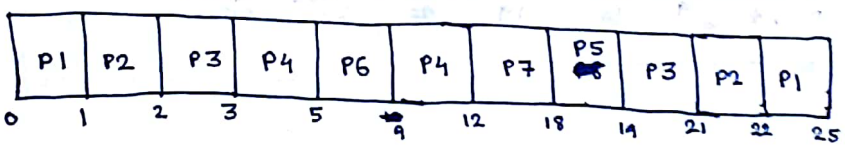
Priority Scheduling (Pre-emptive)

• Priority 8 > Priority 6
(Higher no → higher priority)

PNo	Priority	AT	BT	CT	TAT	WT	RT
1	2	0	4^3	25	25	21	0
2	4	1	2^1	22	21	19	0
3	6	2	3^2	21	19	16	0
4	10	3	5^{3^0}	12	9	4	0
5	8	4	1^0	19	15	14	14
6	12	5	4^0	9	4	0	0
7	9	6	6^0	18	12	6	6

P1 P2 P3 P5

Gantt Chart



$$\text{Throughput} = \frac{\text{Total No. of Process}}{\text{Total Time}}$$

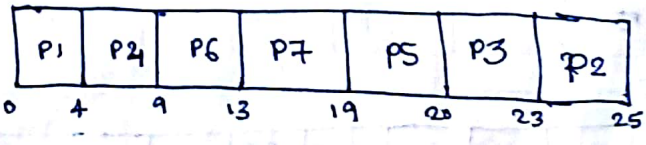
$$= \frac{7}{25}$$

Priority Scheduling (Non-Pre-emptive)

PNo	Priority	AT	BT	CT	TAT	WT	RT
1	2	0	4	4	4	0	0
2	4	1	2	25	24	22	22
3	6	2	3	23	21	18	18
4	10	3	5	9	6	1	1
5	8	4	1	20	16	15	15
6	12	5	4	13	8	4	4
7	9	6	6	19	13	7	7

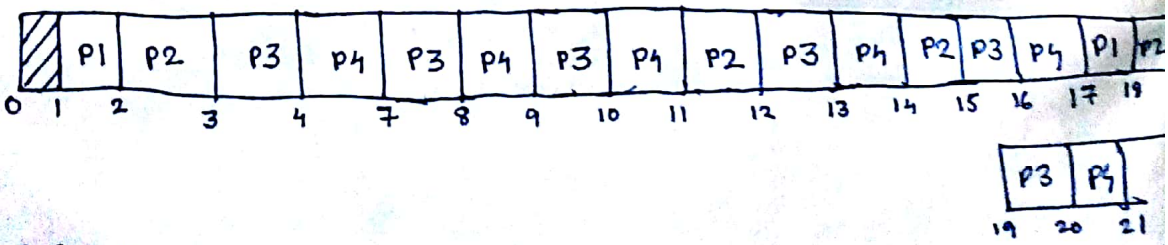
Same

Gantt Chart



Longest Remaining Time First (Pre-emptive)

PNo	AT	BT	CT	TAT	WT	RT
1	1	2	18	17	15	0
2	2	4	19	17	13	0
3	3	6	20	17	11	0
4	4	8	21	17	9	0

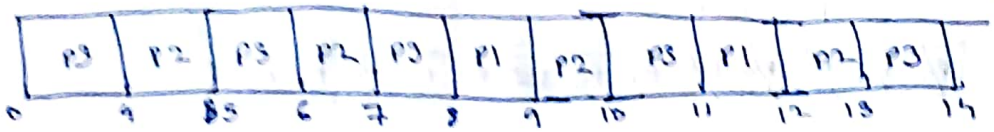


Advantage:

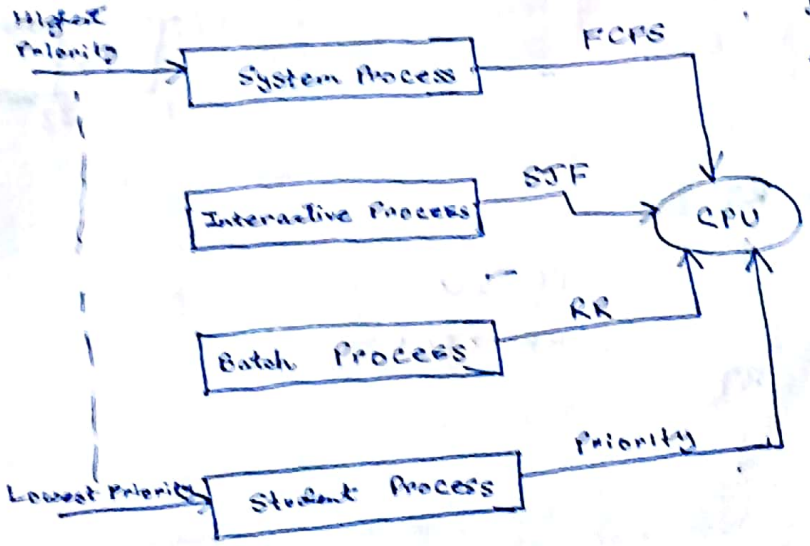
All process end at the same time.

Explain:

PNo	AT	BT	CT	TAT	WT	RT
1	0	2	12	12	10	8
2	0	4	13	13	9	4
3	0	8	14	14	6	0



Multi Level Queue Scheduling



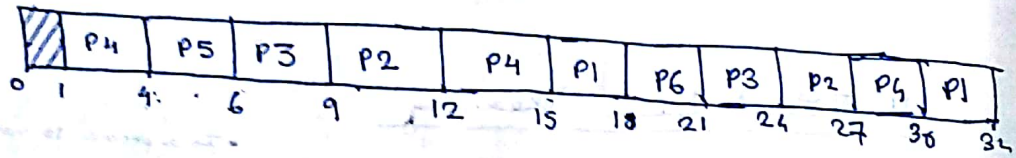
- If a process is not executed for a long period of time due to the presence of higher priority processes, then its priority is increased such that it can get executed

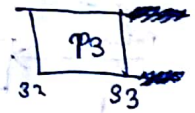
Round Robin Example

TQ = 3

PNo	AT	BT
1	5	5
2	4	6
3	3	7
4	1	9
5	2	2
6	6	3

P4 P5 P3 P2 P4 P1 P6 P3 P2 P4 P1 P3





TQ ↑ CT ↓ RT ↑

TQ ≅ ∞

RR → FCFS

TQ ↓ CT ↑ RT ↓

Threads

heavyweight → process is called heavyweight

Process control block (PCB) → Process stores info about the process/ instructions.

- Thread is a lightweight process.

Definition of Thread: Read from ppt.

- A thread is created within a process, it cannot exist outside a process.

Threads VS Processes

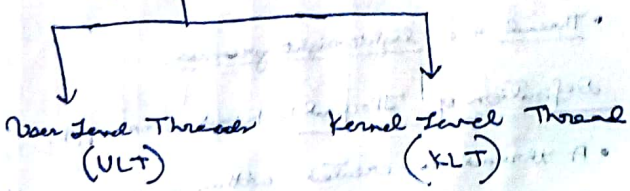
1) A thread has no data segment or heap.	1) A process has code/data/heap and other segments
2) Thread is lightweight, taking lesser resources.	2) Process is heavyweight or resource intensive.
3) Inexpensive creation	3) Expensive creation

Read from ppt.

- Single Threaded and multithreaded model → PPT
- Benefits of Threads → PPT

- Thread Synchronization → PPT
- Relationships between ULT States and Process States → PPT

Types of Threads



- Advantages and Inconveniences of ULT → PPT
- More than 1 thread within a process cannot run simultaneously

- Advantages of Inconveniences of KLT → PPT

Multithreading Models :

— Many to many
— Many to one
— one to one

- Difference between ULT and KLT → PPT
- Relationship between Threads and Processes → PPT

Process Synchronization

- Producer - Consumer Problem
- Race Condition

a = 10

```
P()
{
    R(a)
    a = a+1
    w(a)
}
```

P1	P2
P1 : 11	10
P2 : 12	11

B=2

```
P1()
{
    ① C = B - 1;
    ② B = 2 * C;
}
```

```
P2()
{
    ③ D = 2 * B;
    ④ B = D - 1;
}
```

C1	C2	C3	C4	C5	C6
1, 2, 3, 4	3, 4, 1, 2	1, 3, 4, 2	3, 1, 2, 4	1, 3, 2, 4	3, 1, 4, 2
C = 2-1 = 1	D = 4				
B = 2×1 = 2	B = 3				
D = 2×2 = 4	C = 2				
B = 4-1 = 3	B = 4				
B = 3	B = 4	B = 2	B = 3	B = 3	B = 2

Critical Section

```
do
{
    Enter C.S
      CS
    exit CS
    remainder S
} while();
```

Solution to Critical Section

Requirements —

1. Mutual Exclusion
2. Progress
3. Bounded Waiting

First Attempts — Busy Waiting

A laundry list of proposals to achieve mutual exclusion—

- Disabling interrupts
- Lock variable
- strict alternation
- Peterson's Solution
- TSL .....

For 2 processes —

Case 1: Strict Alternation → Satisfies Mutual Exclusion property
→ Does not satisfy progress (Strict Alternation occurs)
→ NO Bounded Waiting

P0

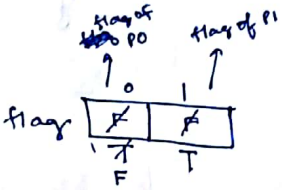
turn = 0

```
while (TRUE){
    while (turn != 0);
    critical-region();
    turn = 1;
    non-critical-region();
}
```

P1

```
while (TRUE){
    while (turn != 1);
    critical-region();
    turn = 0;
    non-critical-region();
}
```

Case 2:



P0

```
while(1)
{
    flag[0] = T;
    while (flag[1]);
    critical section
    flag[0] = F;
}
```

P1

```
while(1)
{
    flag[1] = T;
    while (flag[0]);
    critical section
    flag[1] = F;
}
```

— Satisfies Mutual Exclusion

— Supporting Progress. But both gets stuck in an infinite loop, if context switch occurs after flag[0] = T in P0. Then the loop in P1 continues.

Problems with case 2 —

- Employs busy-waiting while for the cr, a process spins.
- Progress condition is not satisfied properly. It may lead to deadlock.

Case 3: Petersen's Algorithm

- Follows Mutual Exclusion
- Progress Followed (No Deadlock like previous case)
- Follows Bounded Waiting

P0

```
while(1)
{
    flag[0] = T;
    turn = 1;
    while(turn == 1 && flag[1] == T);
    critical section
    flag[0] = F;
}
```

P1

```
while
{
    flag[1] = T;
    turn = 0;
    while(turn == 0 && flag[0] == T)
    critical section,
    flag[1] = F;
}
```

turn = 0 / 1

flag

0	1
F	F
T	F

- Explain how Petersen's Solution solve all the 3 criteria.

Semaphores

For n processes we need to use Semaphore.

Semaphore is of two Types — 1) Binary
2) Counting

Semaphore is an integer variable which is used to synchronise processes.

Pi

```
do {
    entry Section
    Critical Region
    exit CR_Section
    Remainder Section
} while(True);
```

Two operation — 1) ~~wait~~, wait, 2) ~~signal~~ signal

semaphore variable (S) is generally initialized by 1.

S = 1

$P_1, \ldots\ldots, P_n$

```
wait(S)
{
  while(S<=0);
  S = S-1;
}
```

```
signal(S)
{
  S = S+1;
}
```

P_i

```
do{
    wait(S)
    Critical Region
    signal(S)
    Remainder Section
} while(True);
```

$P_1, P_2, P_3, P_4, \ldots\ldots P_n$

$S = 0$

Operations of Semaphore:

- Solve Critical Section Problem
- Perform order of execution of the process
- Manage the resources

Question

Shared variable = x

x = 0

Processes → w, x, y, z

w, x → x = x + 1

y, z → x = x - 2

What will be the maximum value of x if all 4 process execute exactly once.

Ans: +1 +1 -2 -2 = -2 (Ans)

s = 1

x = 0

W	X	Y	Z
wait(s)	wait(s)	wait(s)	wait(s)
R(x)	R(x)	R(x)	R(x)
x = x + 1	x = x + 1	x = x - 2	x = x - 2
w(x)	w(x)	w(x)	w(x)
signal(s)	signal(s)	signal(s)	signal(s)

s = ~~1~~ 0, ~~1~~, 1

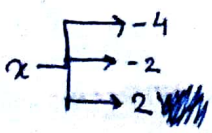
x = 0, ~~-1~~, -1

- If s = 2, find the maximum possible value of x.

s = ~~1~~, ~~1~~, 2 → W

Ans: 2

x = ~~0~~ 0, -2,



...ing Classical Problems using Semaphores:

...he Producers Consumer ~~Pro~~ Problem / Bounded Buffer

```
void producer ()
{
    while(T)
    {
        produce();
        wait(E);
        wait(S);
        append();
        signal(S)
        signal(F)
    }
}
```

- Along with s, 2 other semaphores have been used.

E(empty) is initialized with the size of the buffer.

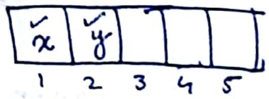
F(Full) is initialized with 0.

~~the size~~

S = 1

E = n

F = 0



E=3
S=0
S=1

E=4
S=0
S=1

```
void consumer()
{
    while(T)
    {
        wait(F)
        wait(S)
        take()
        signal(S)
        signal(E)
        use()
    }
}
```

S = ~~1~~ ~~0~~ ~~1~~ ~~0~~ 1

E = ~~5~~ ~~4~~ ~~3~~ 4

F = ~~1~~ ~~0~~ ~~1~~ 2

Scanned by CamScanner

:

Reader-Writer Problem

Writer

- Only one writer allowed to write at a time
- At time of writing, reading is not allowed

Reader

- Multiple readers can read simultaneously.
- At time of reading writing is not allowed.

One reader, one writer

Writer

```
wait(mutex)
write here
signal(mutex)
```

Reader

```
wait(mutex)
Read here
signal(mutex)
```

mutex = 1

Multiple reader, single writer

Writer

```
wait(wrmutex)
write here
signal(wrmutex)
```

Reader

```
wait(mutex)
readers ++
if(readers==1)
    wait(wrmutex)
signal(mutex)
Read here
wait(mutex)
readers--.
if(readers==0)
    signal(wrmutex)
signal(mutex)
```

mutex = 1

Dining - Philosophers Problem

Solution 1:

Take left fork first and then take second fork

```
do {
    wait (chopstick [i]);
    wait (chopstick [(i+1) % 5]);
    // critical Section
    eat
    signal (chopstick [i]);
    signal (chopstick [(i+1) % 5]);
    think;
} while (true);
```

Solution 2:

Lefty

```
do {
    wait (chopstick [i]);
    wait (chopstick [(i+1) % 5]);
    // critical section
    eat
    signal (chopstick [i]);
    signal (chopstick [(i+1) % 5]);
    think.
} while (true);
```

Righty

```
do {
    wait (chopstick [(i+1) % 5]);
    wait (chopstick [i]);
    .
    .
    .
```

Solution 3:

Use of Arbitrator

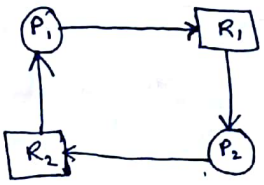
```
do {
    wait(mutex)
    wait(chopstick[i]);
    wait(chopstick[(i+1)%5]);
    signal(mutex)
    // critical section
    eat
    signal(chopstick[i]);
    signal(chopstick[(i+1)%5]);
    think
} while(true);
```

Solution 4: Tannenbaum's Solution

Deadlock

Necessary Conditions for Deadlock:

- **Mutual Exclusion**
  - at least one resource must be held in non-sharable mode
- **Hold and Wait**
  - a process is holding a resource and waiting for others
- **No Preemption**
  - only the process can release its held resource
- **Circular Weight**
  - $\{P_0, P_1, \ldots P_n\}$
  - P_i is waiting for the resource held by P_{i+1}.
  - P_n is waiting for the resource held by P_0.



Dealing with Deadlock —

1) Deadlock Prevention
2) Deadlock Avoidance
3) Deadlock Detection
4) Deadlock Recovery

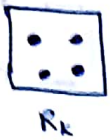
- **Eliminating Circular Wait:**

P_i's ~~request~~ grant for R_j will be accepted only if $R_i < R_j$.

grant not accepted → F(typewriter) = 1
can request successfully → F(DVD) = 5
F(Printer) = 12

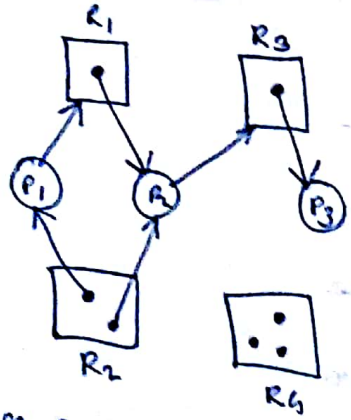
Resource Allocation Graph

- $P_i \rightarrow R_j$ → Request Edge
- $R_j \rightarrow P_i$ → Allocation Edge / Assignment Edge



- The dots represent the number of instances of the resource.

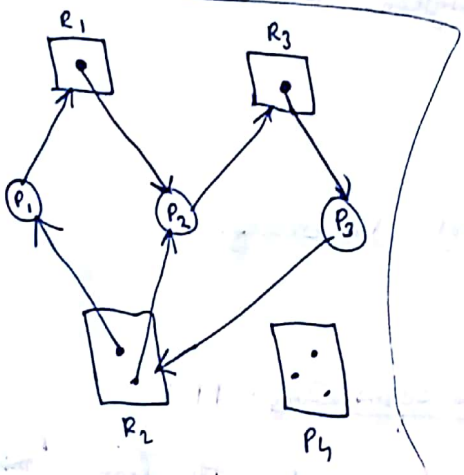
Example Graph:



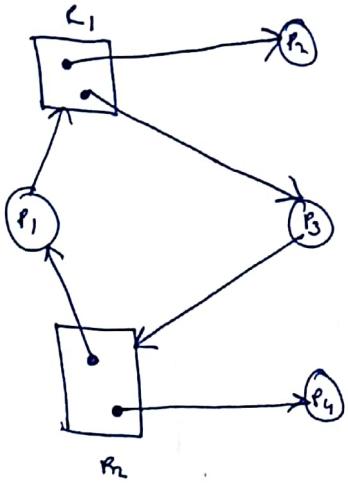
~~Deadlock~~

- A cycle ~~in a block graph~~ in the graph signifies that our system may or may not be in deadlock

Graph with Deadlock:

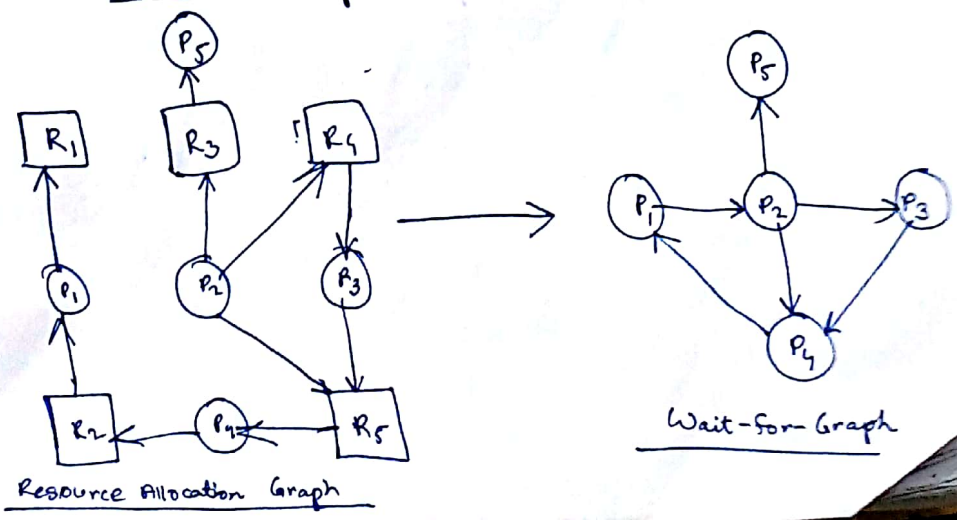


Graph without deadlock:

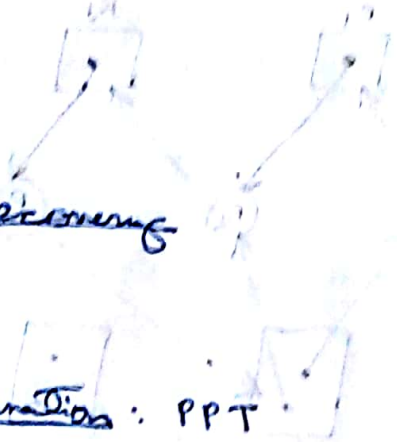


Deadlock Detection

Wait-for-Graph



- If there is cycle in the wait-for-graph, then the system is in deadlock.



Deadlock Recovery

- PPT

Process Termination : PPT

- Factors on which the term 'minimum cost' depends — PPT