Semaphore :- — Binary Semaphore (…
— Counting " (

initialize	increment	decrement
down/(P)	up/(V)

⇒ Wait(S)
{
if (S==1)
S=0
else
Place Process in blocked State
}

Signal(S)
{
S++
}

(1.) Solve CS Problem :-

P1	P2	S
down(S)	down(S)	1
CS	CS	0
up(S)	up(S)	

} while(T) }

Binary Semaphore is use for CS Problem

(2.) Decide order of excution :-

P2 → P1 → P3

$S_1 = 0$, S_2 …

P1	P2	P3
Wait (S_1)	P2 (It is in CS)	Wait (S_2)
CS		CS

```
int count = 0
Void Producer()
{
  int item
  while (true)
{
  item = produce_item()
       if (count == N)
            sleep()
       insert_item(item)
       count = count + 1

  if (count == 1) wake up (consumer) }
       }
Void consumer()
  {
    int item;
    while (true)
    {
      if (count == 0) sleep()
      item = remove_item
      count = count - 1

      if (count = N-1)
           wake up (producer)
    }

}
```

=> Wake up waiting bit is set 0 then it is not call sleep()

P_1

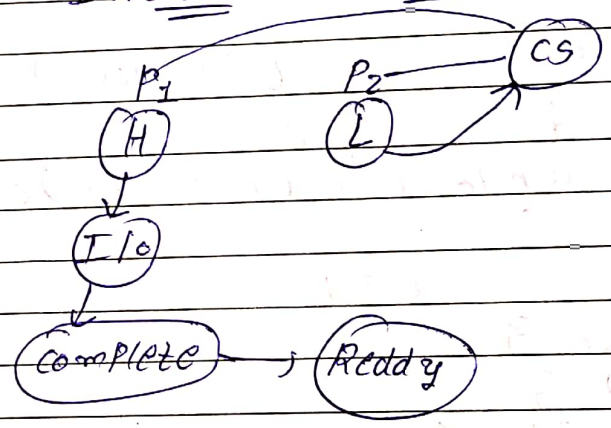
```
Void enter_CS(int X)
{
  While (Test&set(R,X));
}
CS
Void leave_CS(X)
{
  X = 0
}
```

P_2

```
Void enter_CS(X)
{
  While(Test&
    set(R,X));
}
CS
Void leave_CS(X)
{
  X = 0
}
```

(deadlock free)

Priority Inversion Problem:-



Edging => in this we increase priority of lower priority Processor.

Sleep() and Wake up()

(bounded buffer)

Producer and Consumer:-

P_0 P_1

While (TSL (&lock));

CR

lock = false

same

Initial lock = false

lock
~~false~~ True

lock
false

Enter - region :

TSL reg, lock

Cmp reg, #0

JNE Enter-region()

Return // use CR

jump → JNE

not equal to zero

Exit region :

mov lock, #0

(Acquire lock)

ACL {

While (FSL (L, 1))

{

L = 1

}

(fetch lock)

(read lock, increment by 1 and return old value of L)

(Solution for this is ;) (because when L=1 then also it is set L=1)

CRelease → RL (lock)

{

L = 0

}

L = 0 ⇒ available

P_1

ACL

CR

RL

P_2

ACL

CR

RL

Scanned by CamScanner

SHOT ON MI A2
MI DUAL CAMERA

P1

```
while(T)
{
  wants1 = T
  while (wants2 == T)
      CS
  wants1 = F
}
```

P2

```
while(T)
{
  wants2 = T
  while (wants1 == T);
      CS
  wants2 = F
}
```

P1

```
while (S1 == S2);
      CS
  S1 = S2
```

P2

```
while (S1 != S2);
      CS
  S2 = not(S1);
```

ME ✓

Progress X

TSL (Test & set lock)

TSL Rx, lock

Read lock and write into Rx and Return old value of lock.

TSL is multiprocessor System.

```
boolean TSL (boolean *target)
{
    boolean rv = *target
    *target = true
    return rv
}
```

Indivisibly

We can not context switching

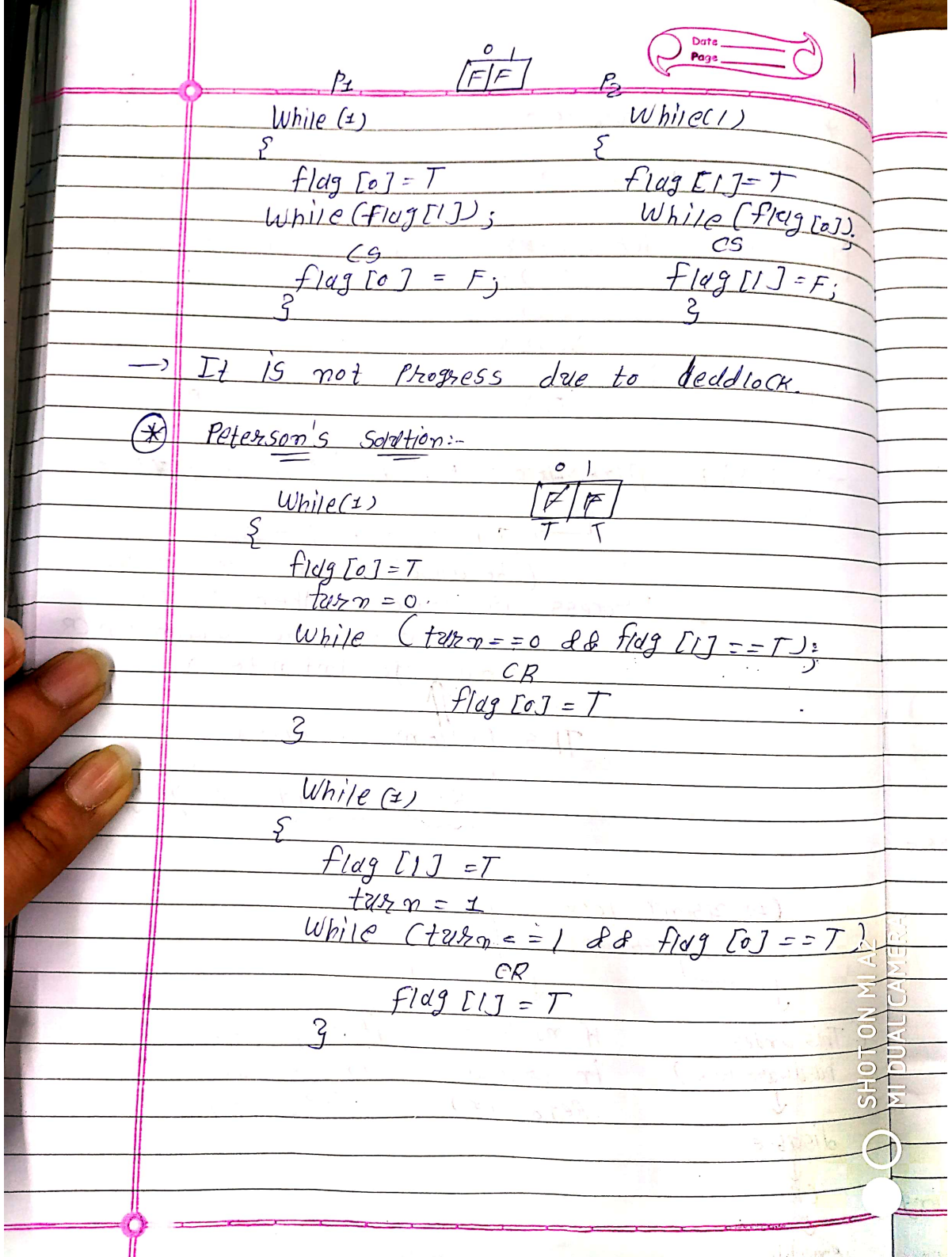
```
Void EnterRegion (int Process)
{
  int others
  others = 1 - Process
  interested [Process] = T
  turn = Process
  While (turn == Process && interested [others] == T);
}
Void leaveRegion()
{
  interested [Process] = F
}
```

P_0	P_1
EnterRegion()	EnterRegion()
CS	CS
leaveRegion()	leaveRegion()

P_0

Others = 1
interested [0] = T
turn = 0
While (turn == 0 && interested [1] == T);
CS
interested [0] = F
others = 0
interested [1] = T
turn = 1
While (turn == 1 && interested [0] == T);
CS
interested [1] = F

P_1



P1 | 0 1 | F F | P2

```
While (1)                    While (1)
{                            {
  flag [0] = T                 flag [1] = T
  While (flag [1]);            While (flag [0]);
        CS                           CS
  flag [0] = F;                flag [1] = F;
}                            }
```

→ It is not progress due to deadlock.

(*) Peterson's solution:-

0	1
F	F
T	T

```
While (1)
{
  flag [0] = T
    turn = 0.
  While (turn == 0 && flag [1] == T);
                CR
        flag [0] = T
}

  While (1)
  {
    flag [1] = T
      turn = 1
    While (turn == 1 && flag [0] == T)
                  CR
          flag [1] = T
  }
```

* Strict Alteration:-

```
Turn = 1

While (T)                          While (T)
{                                  {
  While (turn != 0);                 While (turn != 1);
  CR ();                             CR ();
  turn = 1                           turn = 0;
  NonCR ();                          NonCR ();
}                                  }
(P1)                               (P2)
```

→ Strict Alteration Provide "Mutual Exclusion."

→ It is Not Progress.
(because when in CR No Process is present then P1 is not enter again in CR and go infinite)

This Problem is called "Busy waiting" or "Spin lock"

(1) using lock
(2) Interrupt

(This under hardware base)
↓
disable
↓
CR
↓
enable

if no Process in critical Region (1)

if any process in critical Region (0)

(1.) No 2 Processes may be simultaneously inside Critical Region (CR).
(2.) No assumption about CPU speed.
(3.) If No Process is in CR then it should not blocked.
(4.) No Process should wait forever to Enter its CR.

* Mutual Exclusion:-
Come Process is in CR So no other Process is allow to enter CR it is called Mutual Exclusion)

```
P1 ( )                P2 ( )
{                     {
(1) C = B - 1         (3.) D = 2 * B
(2.) B = 2 * C        (4.) B = D - 1
}                     }
```

B = 2

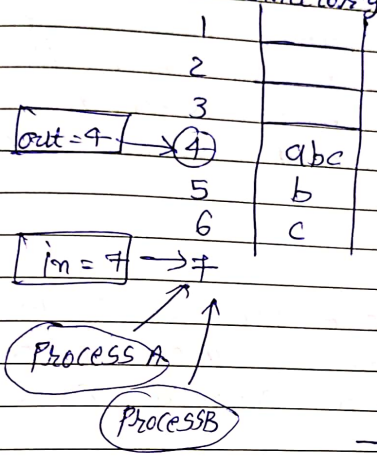
6 1,2,3,4 => 3
3,4,1,2 => 4
1,3,4,2 => 2
3,1,2,4 => 3
1,3,2,4 => 3
3,1,4,2 => 2

* Inter Process - communication *

(print file & delete file)

↓

Spooler directory (maintain file in a Queue)



Shared variable

in — next free slot

out — next file to be printed

Process A

↓

→ in => 7 (read file)

→ next - free slot = 7

↑

(local variable)

Process B

↓

→ in => 7 (read file)

→ next - free slot = 7

↓

next - free slot + 1

(*) **Race conditions:-** (Not consist between two Processes)

(*) **critical section:-** (one Process is in a critical region no one process is enter in this region)

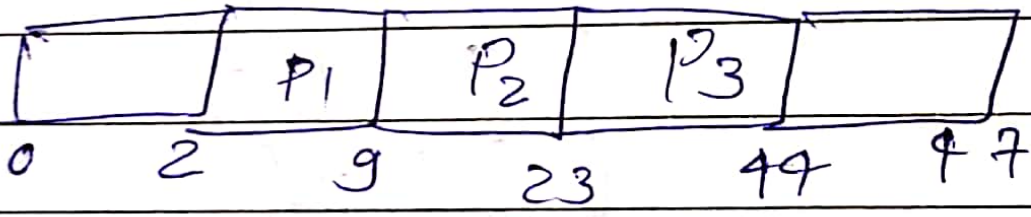
SRTF

	B.T.		
P_1	10	20%	I/O
P_2	20	70%	CPU
P_3	30	10%	I/O

I/O and CPU overlapping. A.T = 0

	B.T.	I/O	CPU	I/O
P_1	10	2	7	1
P_2	20	4	14	2
P_3	30	6	21	3

Gantt Chart



I/O 10

(preemtive)

Round Robin:

↳ Time Quantum

(TQ)

Time Quantum = 4

	A.T	B.T
A	0	3
B	2	6 2
C	4	4
D	6	5 1
E	8	2

A	B	C	D	B	E	D
0	3	7	11	15	17	19	20

Reddy queue

A	D	B	E	D

Multilevel queue Scheduling:

(galvin book)

(multilevel feedback queue)

- TQ = 2
- TQ = 4
- TQ = 8
- FCFS

	Process	Priority	A.T	B.T
A	1	2	0	4 3
B	2	4	1	2 1
C	3	6	2	3 2
D	4	10	3	5 3
E	5	8	4	1
F	6	12 (H.P.)	5	4
G	7	9	6	6

C = 4 - 4 = 0

D = 14 - 5 = 9

E = 2 - 2 = 0

A. W. T = $\frac{16}{5}$

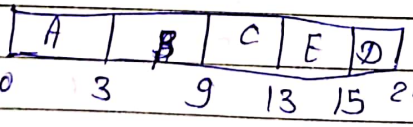
Priority scheduling :-

lower the Number → higher the Priority

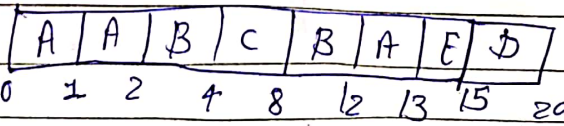
	A.T	B.T.	priority
A	0	~~8~~ 3	3
B	2	~~6~~ 4	2
C	4	4	1
D	6	5	5
E	8	2	4

Preemptive, Non Preemtive

Non Preemtive :-



Preemtive :-



FCFS:-

A	B	E	D	C
0	3	9	13	18	20

A = 3 − 0 = 3
B = 9 − 2 = 7
C = 13 − 4 = 9
D = 18 − 6 = 12
E = 20 − 8 = 12

A.T.T = 43/5

A = 3 − 3 = 0
B = 7 − 6 = 1
C = 9 − 4 = 5
D = 12 − 5 = 7
E = 12 − 2 = 10

A.W.T = 23/5

A = 3 - 0 = 3

B = 9 - 2 = 7

C = 11 - 4 = 7

D = 15 - 6 = 9

E = 11 - 8 = 3

C = 15 - 4 = 11

D = 20 - 6 = 14

A. T. T = $\frac{38}{5}$

Waiting time

A = 3 - 3 = 0

B = 7 - 6 = 1

E = 3 - 2 = 1

C = 11 - 4 = 7

D = 14 - 5 = 9

A. W. T = 18

Process	B.T	Arrival Time
P_1	3	0
P_2	3	0
P_3	20	0

P_1	P_2	P_3
0	3	6	26

⇒ Turn around Time

$P_1 = 3 - 0 = \boxed{3}$

$P_2 = 6 - 0 = \boxed{6}$

$P_3 = 26 - 0 = \boxed{26}$

Average Turn around Time ⇒ $35/3$

⇒ Waiting Time ⇒ $3 - 3 = \boxed{0}$

⇒ $6 - 3 = \boxed{3}$

⇒ $26 - 20 = \boxed{6}$

Average Waiting Time ⇒ $9/3 = 3$

(Non Premptive) ↓

(2.) SJF (Shortest Job first)

Pro. No.	B.T.	A.T
A	3	0
B	6	2
C	4	4
D	5	6
E	2	8

Gantt chart ↓

A	B	E	C	D
0	3	5	11	15	20

(*) Turn Around Time => Completion time − (Arrival time)
⇓ ⇓
different for different Process same for different Process

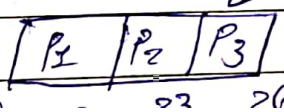
(*) Waiting Time => Turn Around Time − Service Time (Burst Time)
⇓
(Actual time / execution time)
⇓
same for different algorithm in one Process

(Non Preemptive)

(*) (1.) First come first serve (FCFS)
=> Convoy Effect

Arrival Time	Process	Burst Time
0	P_1	20
0	P_2	3

Gantt chart
⇓



(4.) Predictability (load balancing)

(5.) Throughput (maximum)

(6.) Processor utilization (Higher)

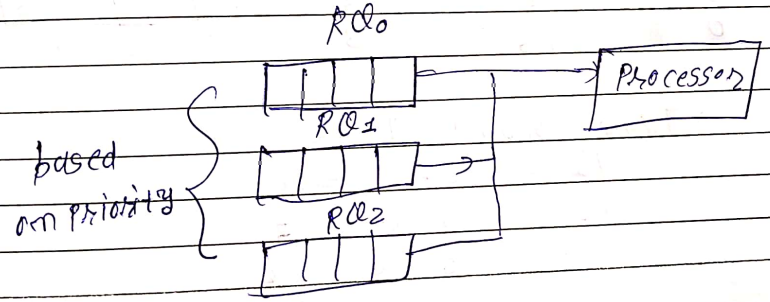
(7.) fairness (Everyone should give equal chance)

(8.) Enforcing priorities (Priority followed)

(9.) Balancing Resources

User Specific:- (1.), (2.), (3.), (4.)

System Specific:- (5.), (6.), (7.), (8.), (9.)
(oriented)



Starvation, edging

(1.) Non Preemptive (Complete or blocked process) ← We can take CPU

(2.) preemptive (we can take CPU in between)

term → Scheduler

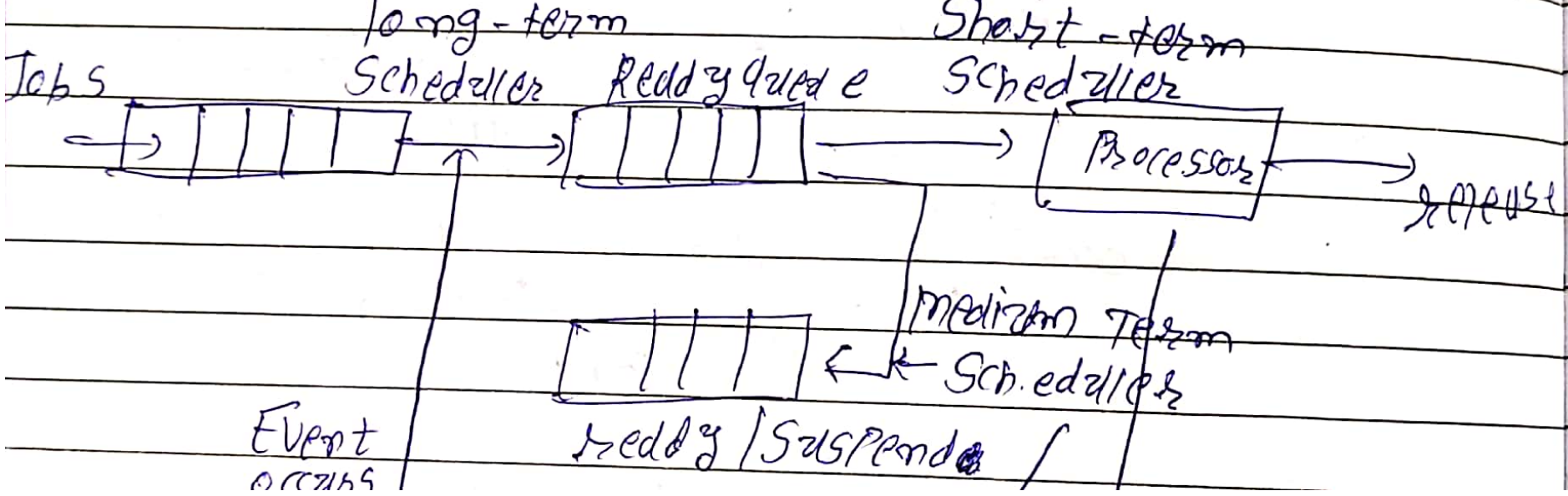
Ready / Suspended

Ready

Medium term reduce Degree of Multi-Programming

Use for Swapping

Short term ⟹ Ready → Running (dispatcher)



Suspend → Secondary memory

Suspend ularai → main memory

New → Process Not lodded In main memory

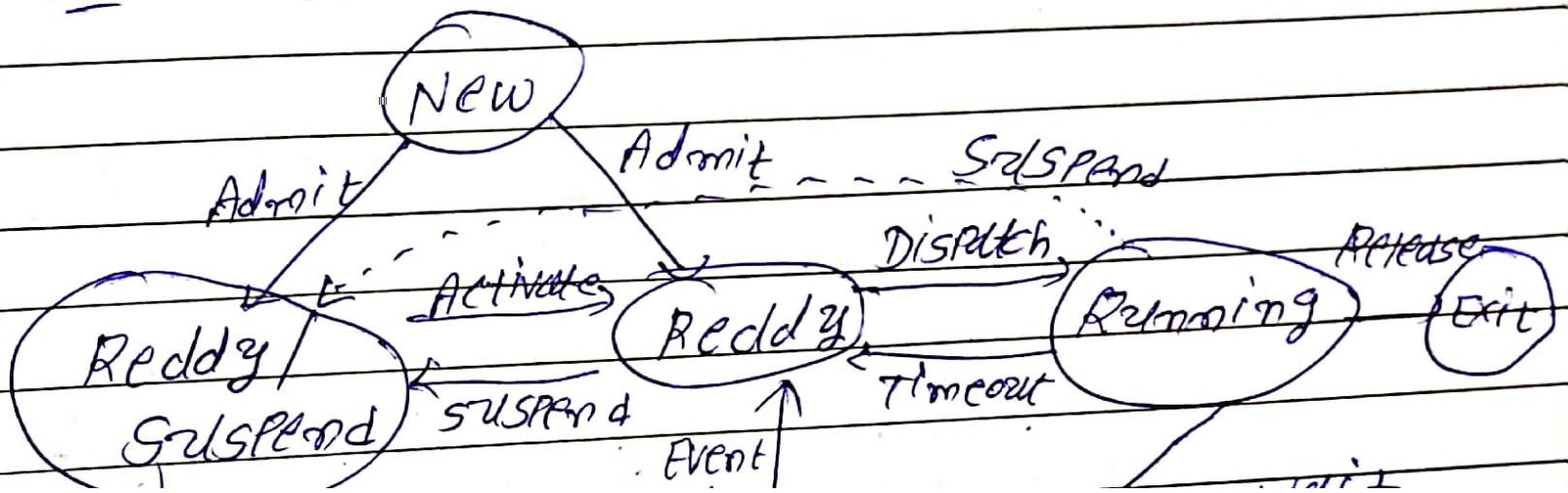
Reddy → " is " " " "

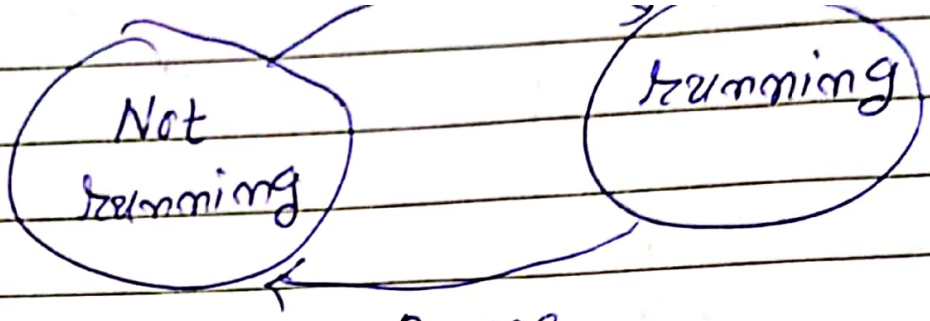
Suspended Processes:-

Swapping (OS Swaps one of the blocked Process out on disk into suspended mode)

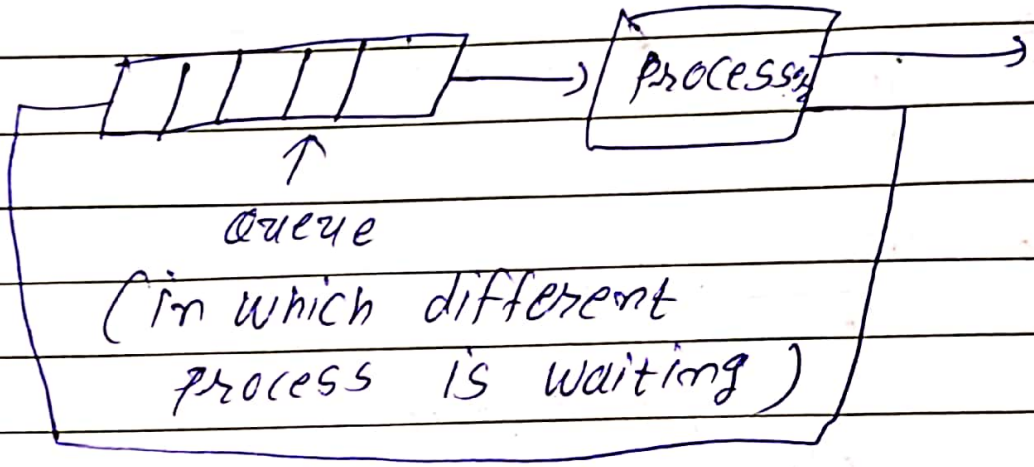
I.M.P.:

* Seven-State Model:-





One Process at a time (Unique Processor)



⇒ Preemption (Stop Process in between)

Privileged instruction → (it is use for os and kernel not for user)

M.P:-

// Five - State Model:-

* Process *

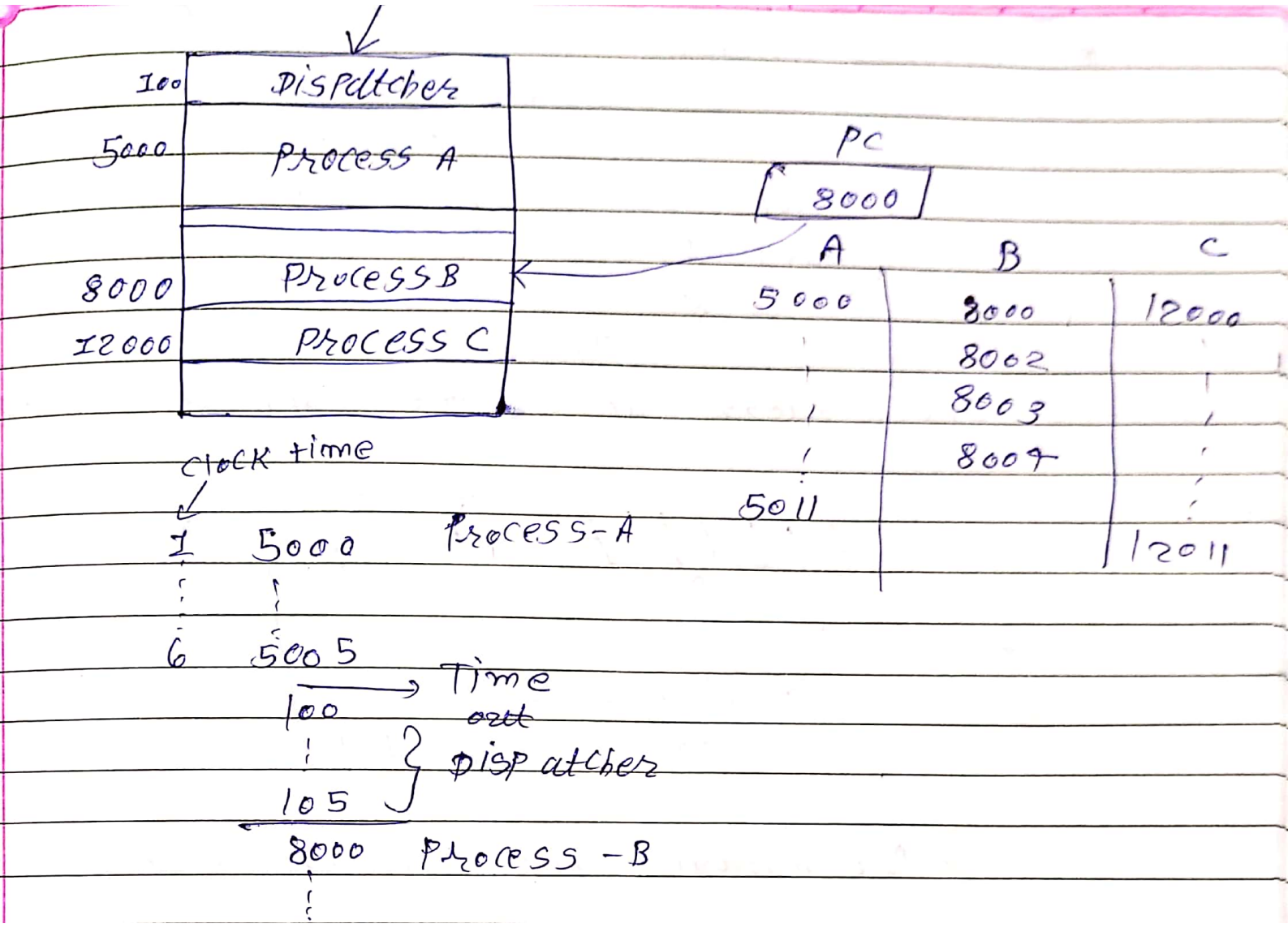
Process :- A Program in execution

- An instance of Program running on a computer
- unit of activity characterized by execution of sequence of instructions current state & associated resources

→ data
→ Program code

(*) Process control Block :- (PCB)

- Identifier (unique code for every Process)
- State
- priority
- Program counter
- Memory pointers



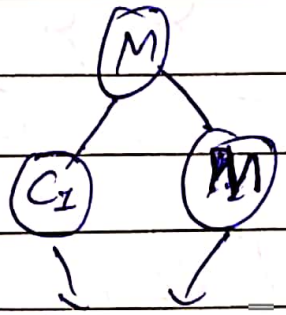
* execve or exec:- (replace our command with old one)
* argv and argc (command line)
  - argv → value
  - argc → count

Virtual Machine:-
- guest os
- host os

fork

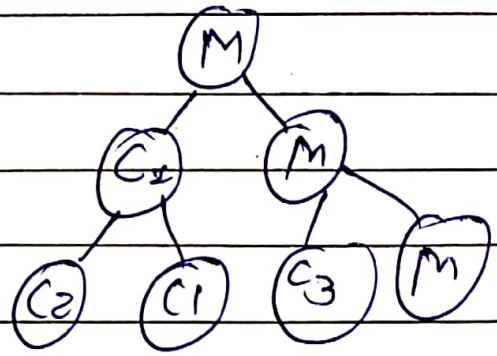
Child (0) | parent (Child ID)

```
main()
{
  fork()
  pf(Hello);
}
```



2 times Hello

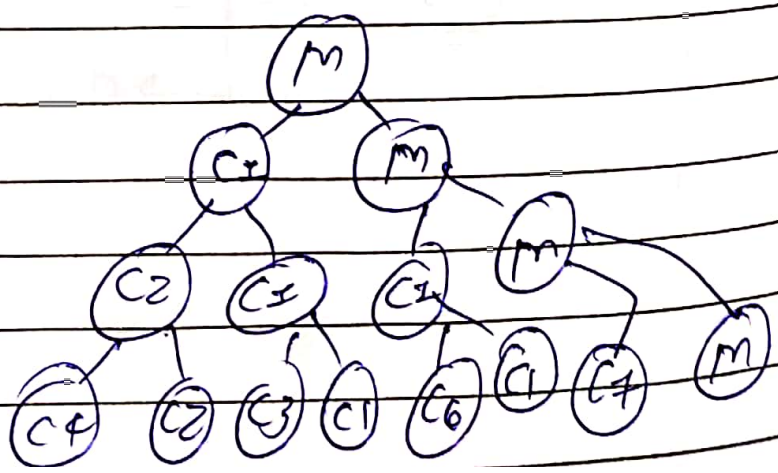
```
main()
{
  fork() }n=2
  fork()
  pf(Hello);
}
```



2^n = 4 times Hello

$2^n - 1$ = 4 - 1 = 3 child process

```
main()
{  fork()
   fork()
   fork()
   pf(Hello);
}
```

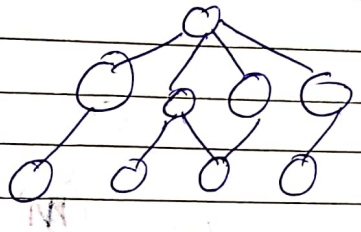


(interrupt)

→ TRAP – user mode to Kernal mode

Kernel Types:.

→ Monolithic Kernel
→ Micro Kernel



Main Procedure

Source Procedure
utility Procedure

[Model of Monolithic kernel]

layered System

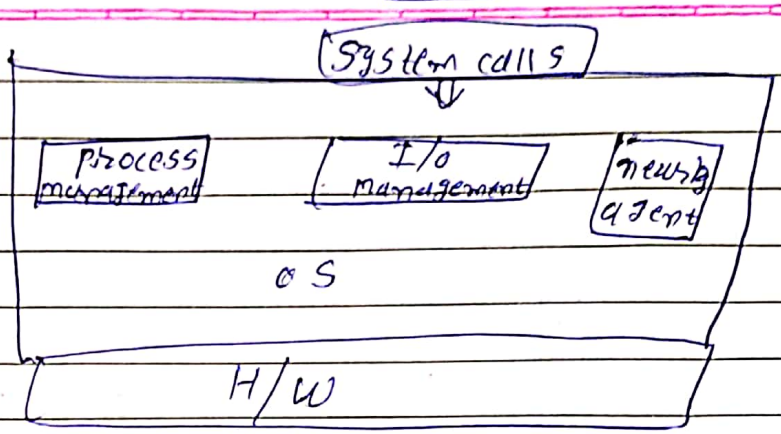
Operator
user Programs
operator Communication
Memory M.gmt
Multi Programming

[The System]

OS → OS as extended machine (Top to down / bottom to top)
→ OS as resource manager (Abstraction)
↓
Memory, CPU, Cycles

OS

GUI



(*) services of os:-

=> Program execution:-

=> I/O management:-

=> filesystem manipulation:- (file management)

=> Error detection:-

=> resource allocation:-
(CPU cycle, memory)

=> Security:-

System Software | application Softwares

→ os is stored in ROM, but while booting in RAM.

Simultaneously work →
- Multiprocessing → એક વખતે કેટલા કામો થાય
- multitasking → યુઝરને લાગે કામો થાય છે પણ fast switching હોય
- multiprogramming → In memory many program
  → CPU utilizing good

Application software:- word, excel
System software:- OS, drivers

OS is initially stored in ROM, but boots(loads) in R

Generations:-
Vacuum tube
Transistor
IC (multiprogramming)
GUI
Super computer

provided by hardware
mode bit 0 ← supervisor mode
OS runs in Kernal mode
applications runs in user mode } dual mode
Shell - command line interface (interpreter) mode bit 1

Kernal is the core of operating system.

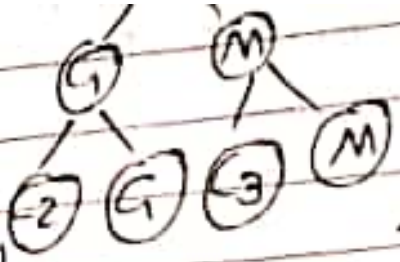
Kernal + protection/security/abstraction = OS

Top to down } fun. to provide abstraction
down to top } to manage the hardware

Resource manager:- time multiplexed
space multiplexed

multiple programming - CPU utilization is more
multi processing -
Multi tasking - threading

main()
{
fork();
fork();
printf("Hello");
}



print : 2^n

4- times Hello

execve() or exec():
replacing current process

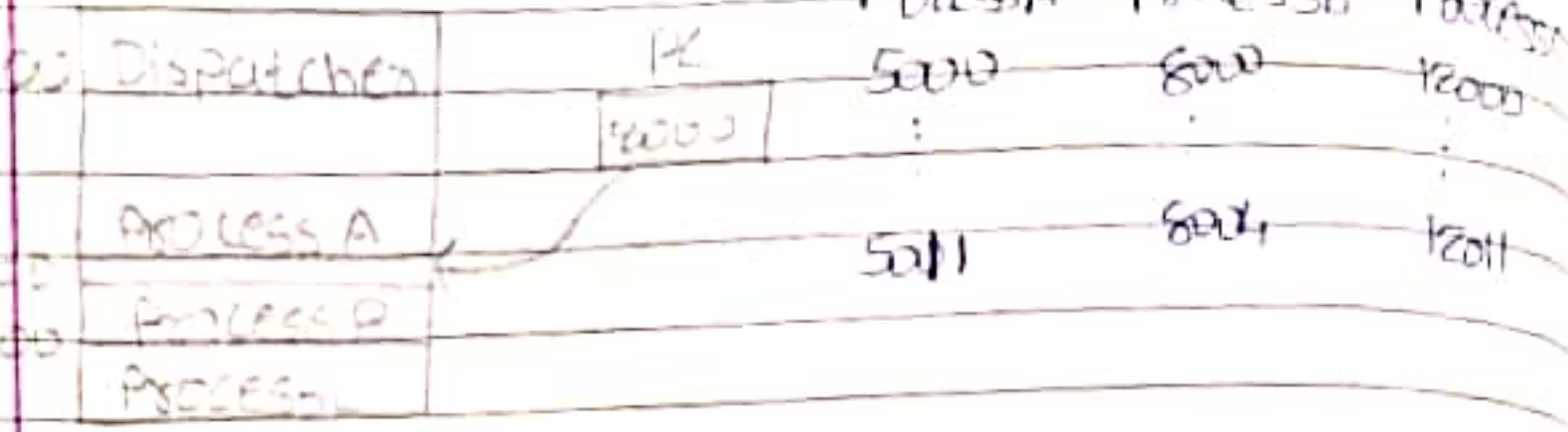
64 bit handles more data then 32 bit.

Virtual machine:- one more OS
host & guest OS
layer :- Hypervisor

OS
- OS as extended Machine (top to down, bottom to top)
- OS as resource Manager (Memory, CPU, cycles)
  provides services (process, service, I/O mgmt.)

History

OS zoo



5005 → Time out

100
:
105
} dispatcher program

program counter ⇒ 8000

8000
:

8004 → Time out
100
:
105

12000
:

12005 → Time out

Galvin

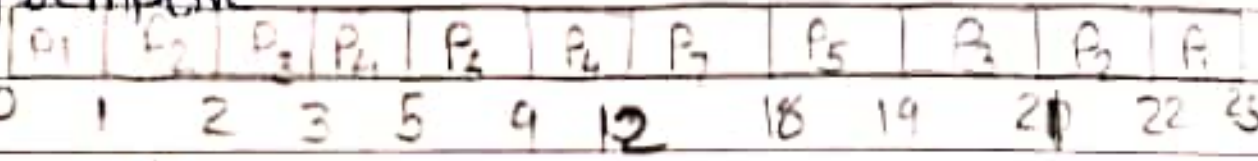
Multilevel feedback queue:-

TQ=2
TQ=4
TQ=8
FCFS

multiple queues
higher burst time then process go in below queue

Priority	A.T.	B.T.
2	0	4
4	1	2
6	2	3
10(HP)	3	5
8	4	1
12	5	4
9	6	6

- Preemptive



B.T.	I/O	CPU	I/O
10	2	7	1
20	4	14	2
30	6	21	3

20% I/O
70% CPU
10% I/O

* Producer - Consumer problem: Cbounded buffer pr
pointer, client - server, barcode reader

```
#define N 10
int count = 0
void producer()
{
    int item;
    while (true)
    {
        item = produce_item()
        if (count == N) sleep()
        insert_item (item)
        count = count + 1
        if (count == 1) wakeup (consumer)
    }
}
void consumer()
{
    int item
    while (true)
    {
```

Q) dispatcher / scheduler selects next process

* Type's of schedulers:-
1. long term (new → ready) (or ready / block)
2. short term (
3. Medium term (secondary to main, vice versa swapping clock / I/O

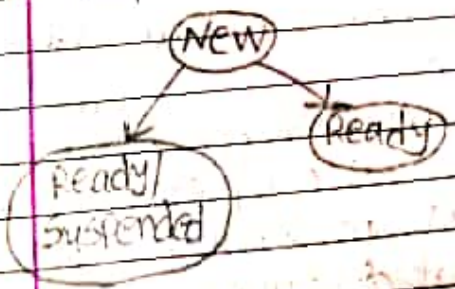
Q) Process may be interrupted by fault in memory or

Q) Structure of a process:-
int pid
long state


linked list of processes

* Types of processor scheduling:-
long term
medium term
short term
} system software (Varies based on frequency)

Q) long term → Job scheduler can control
Degree of multiprogramming
If processor is idle then create



CPU bound :- process uses CPU
I/O bound :- process uses I/O
Generally mixing

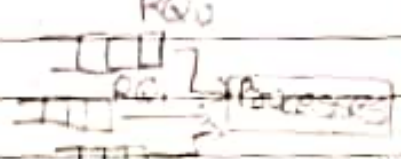
User :- Response
Deadline
Turn around time
Predictability

System oriented :- Throughput
Processor utilization
Fairness
Enforcing priority
Balancing resources

Batch system :- Throughput - no. of jobs completed per hour
Turn around time

Real time system :- Deadlines
Predictability

Interactive system :- response time



Queues are divided according to priority queues (0-n)

Starvation :- Edging
After some time give more priority to lower priority process

Nonpreemptive :- Either blocked / complete
preemptive :- Take process from CPU
priority
system clock

Q→ Busy waiting wastes CPU cycles.

Q→ Priority inversion problem:-

```
P1                                    P2
void enter_CS(X)                      void enter_CS(X)
{                                     {
  while(Test&Set(R,X));                 while(Test&Set(R,X));
}                                     }
  CS                                    CS
void leave_CS(X)                      void leave_CS(X)
{                                     {
  X=0                                   X=0
}                                     }
```

Q→ This is deadlock free

Q→ Priority - inversion problem:
using sleep (blocked) and wake-up in place of busy waiting

```
P1                    P2
Higher pri.           lower pri
   ↓                     ↓
I/O (blocked)           CS
   ↓
Ready (can not enter in CS)
```

Q→ lower priority can not schedule.

Date
Page

Q→ I/O devices:- driver communicate with controller
↑ ↑
kernal mode has registers
access hardware

procedure call:- user & OS (user mode)
system call:- OS & hardware (kernal mode)

Q→ TRAP instructions:- provides transaction b/w user mode to system mode

Q→ Interrupt (disturbance) → Hardware
→ Software

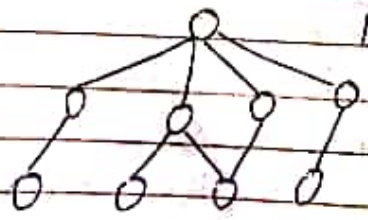
Q→ System call:- stack
process fork():- return 0 id to child
Mgmt. exit() & child id to parent (copy of present)

read(), write(), open(), close(), seek():- file Mgmt

mkdir(), rmdir(), link():- Directory Mgmt

Q→ Kernal types:-
↳ Monolithic Kernal
↳ Micro Kernal

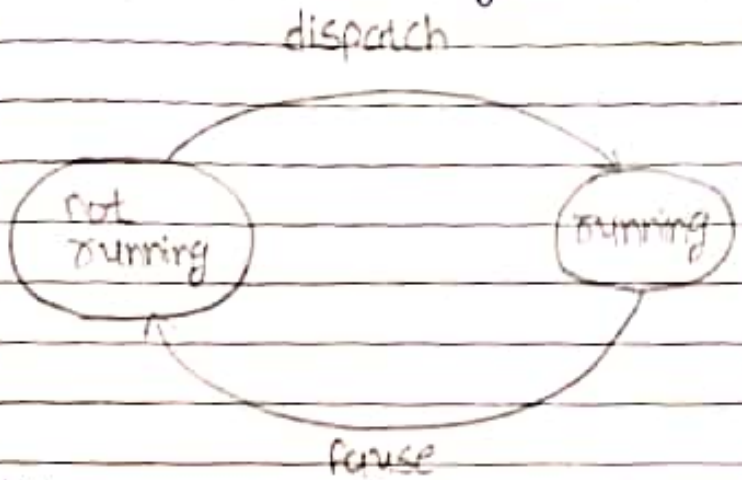
Model of Monolithic:- Kernal



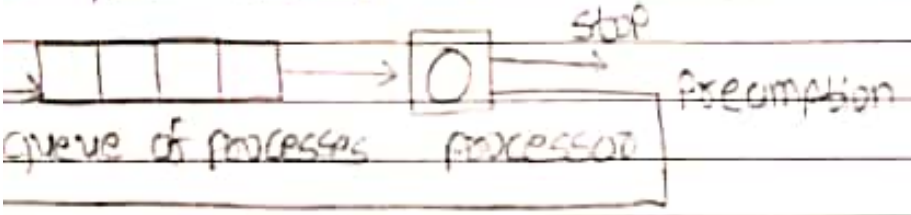
Main procedure
service procedure
Utility procedure Clock up

Process states:-

Two state:- running or not running

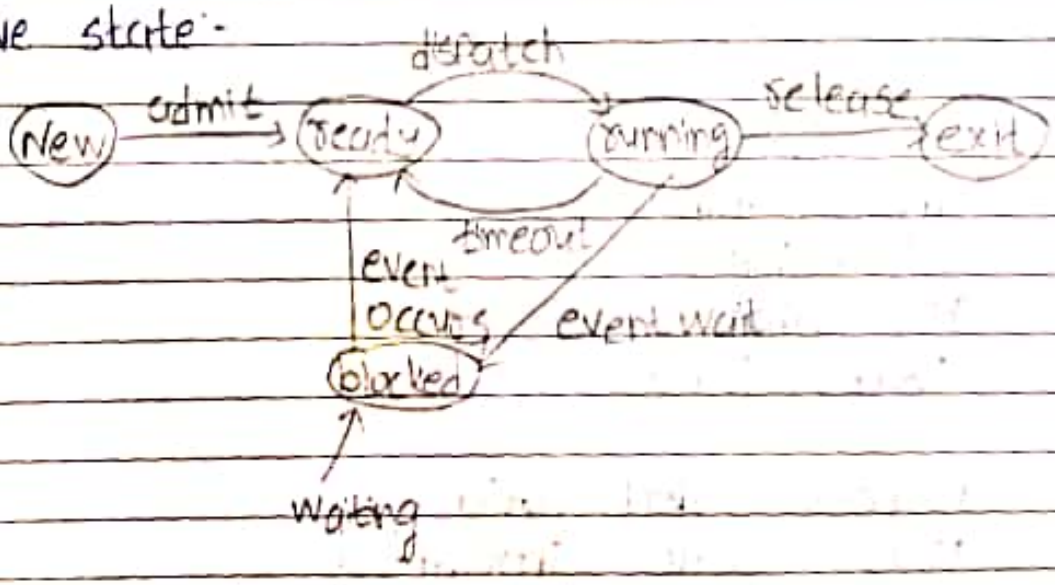


Uniprocessor system - one process will execute at a time



privileged instruction - only for kernel

Five state:-



New → ready (main memory)

Every state process is in main memory

Ready → Exit, blocked → Exit : if parent process terminate then child process automatically.

context switching is not possible during this (Indivisible)

```
boolean TSL (boolean *target)
{
    boolean rv = *target
    *target = true
    return rv (return old value)
}
while (TSL(&lock));        lock is false initially
```

OR

```
lock = false
```

```
enter_region:                exit_region:
    TSL Rx, lock                 mov lock, #0
    cmp Rx, #0
    JNE enter_region()
    Return //use CR
```

JNE :- jump not equal to 0, cmp :- compare

```
AQL  ← acquired lock
{
    while (FAL(L, 1))  SOL :;
    {
        L = 1
    }
}
RL ← release lock
{
    L = 0
}
intion L = 0
```

P_1	P_2
AQL	AQL
Use CS	Use CS
RL	RL

context switch and in AS
after while context
then RL, context switch
Not allowed to ente

1. No 2 processes may be simultaneously inside CR.
2. There should be no assumption about CPU speed.
3. If no process is in CR then it should not block it.
4. No process should wait forever to enter in CR.

Q) Mutual Exclusion:- If any process is in critical section then other processes should not to be allowed to access it.

Ex

```
P1()          P2()          B=2
{             {
C=B-1         D=2*B
B=2*C         B=D-1
}             }
```

These processes run concurrently then one after another.

sequence:- 1,2,3,4 ⇒ B=3
3,4,1,2 ⇒ B=4
1,3,4,2 ⇒ B=2
3,1,2,4 ⇒ B=
1,3,2,4 ⇒
3,1,4,2 ⇒

2 never execute before 1

Q) Solution to avoid race condition.

1. Strict Alternation:- Turn=1

```
while(1)
{
    While(turn!=0);   <- True
    CR();
    turn=1;
    Non(CR);
```

Q→ 200 types:- mainframe (batch, transaction, time sharing)
server os
Multiprocess os (parallel computers)
personal computer os (gui)
Real-time os (hard / soft)
Embedded os
Smartcard os

multiprogram	multitasking	multiprocessing
CPU utilization is more (less ideal)	continuous switch (process switching in threads)	parallel process (simultaneously execution)

Q→ Server os - web server, print server, file server

Q→ Real-time - multimedia (soft), missile launching (hard)

Q→ privilleged instruction - used by kernal mode only
system clock setting
Not accessed in user mode

driver :- provides communication b/w user & os (device)

MMU - memory management unit (virtual to main mem)

Q→ Memory hierarchy
Registers
Cache
Main memory
Magnetic disk } secondary
Magnetic tape } storage

top-down: access time increases, memory capacity increases

Textbook 2.3.2.4

Test & set lock :- instruction
(instruction based hardware solution)

```
while (T)
{
    wants1 = T
    while (wants2 == T);
        CS
    wants1 = F
}

while(T)
{
   wants2 = T
    while (wants1 == T);
     CS
      wants2 = F
}
```

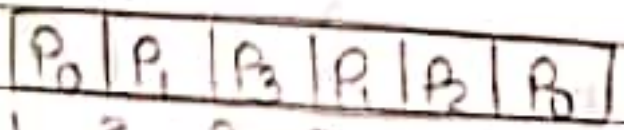
while($S_1 == S_2$); CS $S_1 = S_2$

P_2 :- while ($S_1 != S_2$) CS $S_2 = (not\ S_1)$

Mutual exclusion ✓
Progress ✗ (Not same process runs twice)

TSL (Test and set lock) :- hardware solution
Rx :- register
lock :- value available in memory (0 or 1) of lock
Reads value of lock in register and returns old value
Context switching
Interruption is not applicable for multiprocessors.
TSL is used for multiprocess solution.

Priority	burstime	arrivaltime	endtime	turnaroundtime	waiting
4	3	1	11	10	7
2	2	2	8	6	4
3	1	4	9	5	4
1	4	3	7	4	0
				25	15

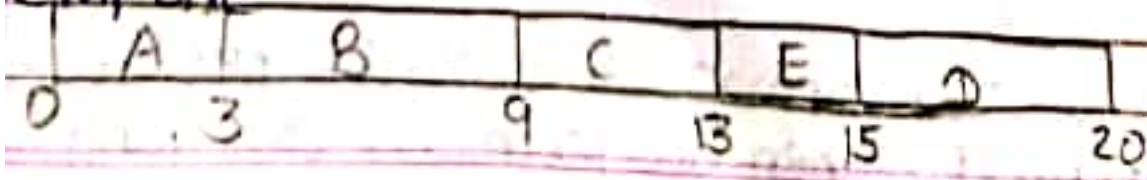


Priority scheduling (non-pacamptive and pemptive)

A.T.	B.T	Priority	Endtime
0	3	1	3
2	6	4	2
4	4		1
6	5		5
8	2		4

lower value higher priority

emptive



If process A :- in 6

next free slot:=6

then process B:- in 6

next free slot+=1

If shared variable is safe then

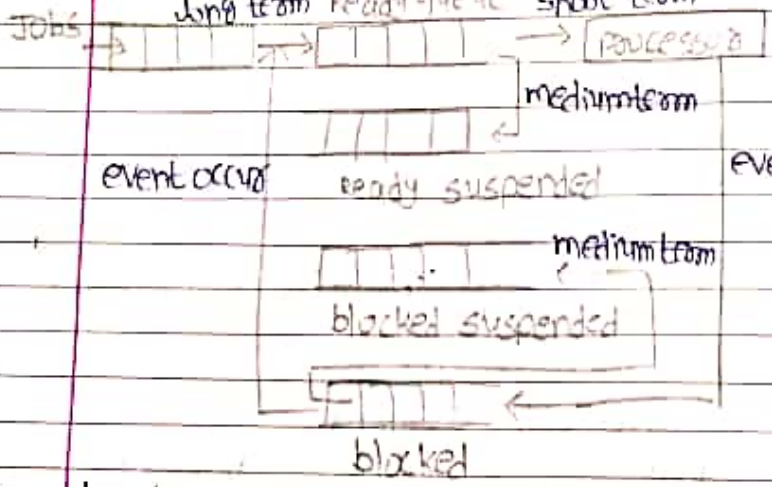
Race condition:- Two or more processes are communicating with each other and final result depends on which process will run

Critical section:- shared data which is accessed by many programs. So, that region should be accessed by any one process at a time using synchronization.

region

Q→ medium term:- swapping process (main to secondary)
Medium term scheduler can reduce degree of multiprogramming
because it swap out processes.

Q→ short term:- CPU scheduler (dispatcher)
Ready → Run
dispatcher selects processes from queue according to scheduling algorithm



freq long term < medium term < short term

* Scheduling criterias:- (only for short term)
user system

min 1. Turn around time:- process total time (submission to completion)
min 2. Response time:- multiple processes and first response
3. Deadlines:- criteria under
4. Predictability:- load balancing, process should complete in amount time
max 5. Throughput:- no. of processes completed per unit time
max 6. Processor utilization:- CPU should not be idle
7. Fairness:- fair to users, not discrimination
8. Enforcing priority:- priority should follow
9. Balancing resources:- not process over utilized

Q) Binary semaphore for 2 processes is used to s
Critical section problem.

Q) Order of execution of some process uses bin
semaphore.

Q) Busy waiting is not present in semaphor

Q) Decide order of execution: $P_2 \rightarrow P_1 \rightarrow P_3$
Two semaphores are used.
- $S_1 = 0$, $S_2 = 0$ ← Not any condition for
-

P_1	P_2	P_3
wait(S_1)	P_2 (CS)	wait(S_2)
CS	signal(S_1)	CS
signal(S_2)		

Q) Context switching is not possible in wait()
and signal() are executing. (atomic/indivisib

Q) Wakeup waiting bit is used to check whether particular consumer is actually sleep or not.

* Semaphore:- N process solution
initialization, increment, decrement

Q) Application:- To solve critical section problem

Q) functions:- wait(), signal()

1. binary semaphore ← binary
2. counting semaphore ← integer value

Q) Busy waiting is not but locking() & unlocking()

1. Binary semaphore fun:-

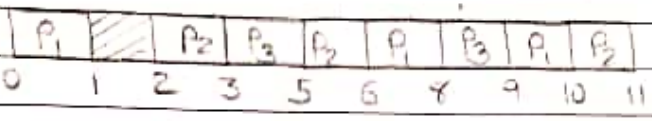
Q) Initial value should be initialized.

```
down(P)                    up(V)
wait(S)                    signal(S)
{                          {
  if(S==1)                   S++
    S=0                    }
  else
    //place process in
      blocked state
}
```

Q) Solve CS problem:- assume S = 1

```
P1                         P2
down(S)                    down(S)
CS                         S
up(S)                      up(S)
```

	A.T	Priority	b.T	Ib	CPV
P_1	0	2	1	5	3
P_2	2	3	3	3	1
P_3	3	1(high)	2	3	1



P2:- While (1)
{
step 1 flag[1] = T
while (flag[0]);
Critical section
flag[1] = F
}

Q→ Mutual exclusion (only one process in CS)

Q→ If at first step process context switch then deadlock occurs. Both process are waiting for critical turn

Q→ Peterson (both strict alteration)

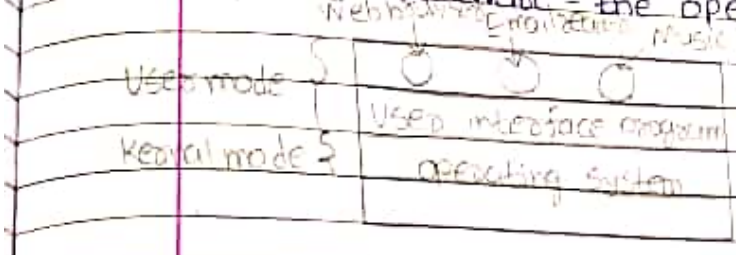
P1:-while (1)
{
flag[0] = T
turn = 0
while(turn == 0 && flag[1] == T);
CS
flag[0] = F
}

F	F

Mutual exclusion ✓
No deadlock (indefinite blocking)
For one cycle

P2:- While (1)
{
flag[1] = T
turn = 1
while (turn == 1 && flag[0] = T);
CS
flag[1] = F
}

Q) Managing one or more processors, main memory, disks, printers, various input/output devices requires a layer of software - the operating system.



Q) Kernal is the first program of operating system that is loaded into the main memory to start the working of the system. It basically translates the commands entered by the user in a way to make the computer understand that what has user requested. Acts as a bridge b/w application software and hardware.

Q) Kernal takes care of the memory, process, task and disk management.

Q) Kernal is an interface between software & Hardware. whereas OS is in between user & hardware.

Q) Top to down:- provide application programmers abstract resources
Down to top:- manage these hardware resources

Q) 1st generation:- Vacuume tube
2nd generation:- transistor, batch system, mainframes
3rd generation:- ICs, multiprogramming
4th generation:- Personal computers

Q) multiprocessing refers to the hardware (i.e. CPU units) rather than the software.

ptive

A	A	B	C	B	A	E	D

0 1 2 4 8 12 13 15 20

Starvation problem in priority based algorithm (solution AG)

Round Robbin algorithm:-
↳ Time quantum (Response time)
If time quantum is more then FCFS
If time quantum is less then context switching

AT	B.T
0	3
2	6
4	4
6	5
8	2

time quantum = 4

A	B	E	D

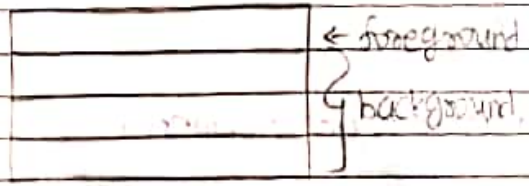
Ready queue

A	B	C	D	B	E	D

0 3 7 11 15 17 19 20

galvin

Multilevel queue scheduling:-
foreground:- processes executes first
background:-



Process

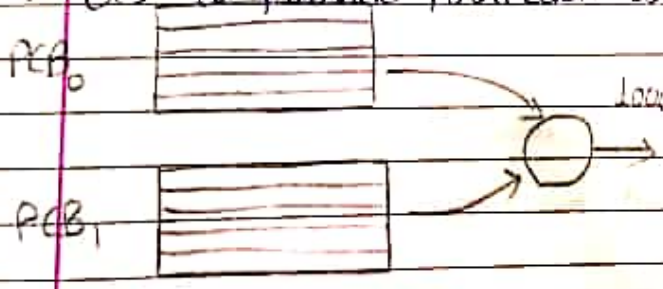
Stalling

Q) a program in execution
Every process has different id
An instance of program running on a computer
Unit of activity characterised by execution of sequence of instructions, current state & associated resource

program code
Data

Q) Process control block:-
- ↳ Identifier
- ↳ state (running, ready)
- ↳ priority (not necessary to execute in priority)
- ↳ program counter (store address of next instruction)
- ↳ Memory pointers
- ↳ context data (stores values in registers)
- ↳ I/O states information (list of open file, I/O devices)
- ↳ Accounting information (CPU time that process has used)

Q) OS to provide protection of above data for each process



Q) Context switching:- saving information for process which is running and loaded for next process By dispatcher

(begin)

```
void enter_region(int process)        process no. 0
{                                                 1
    int other
    other = 1 - process
    interested(process) = T
     turn = process
     while (turn == process && interested[othe
}
leave-
void region()
{
    interested[process] = F
}
```

P0 P1

enter_region()

First come first serve:- (FCFS) Non preemptive alg.
Convoy effect (large burst time)

Gantt Chart:-

Pro No.	BT
P_1	20
P_2	3
P_3	3

Turn around time (all processes arrive at 0)

P_1 $20-0 = 20$

P_2 $23-0 = 23$

P_3 $26-0 = 26$

avg. turn around time: $69/3 = 23$

Waiting time

P_1 $20-20 = 0$

P_2 $23-3 = 20$

P_3 $26-3 = 23$

```
while (T)
{
    while (turn != 1);
    CR();
    turn = 0;
    NonCR();
}
```

True

At last turn=0

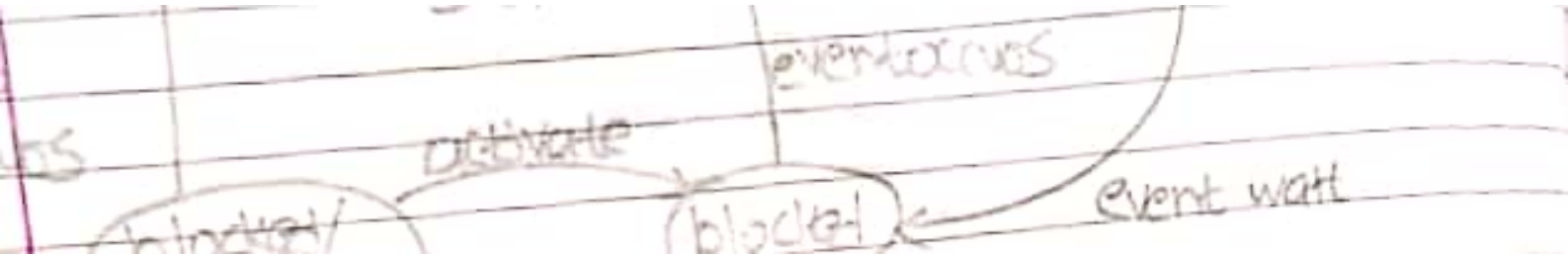
If only A is running then after first execution it can not enter second time. Problem of progression.

Strict alteration between only not more

Busy waiting/spin lock:- simply testing the condition in while loop only running.

Using lock (software solution)
Interrupt related to hardware

In lock based mutual exclusion not follow

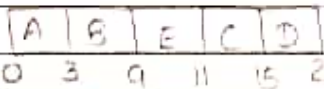


Pro No.	B.T.	FT	TAT	WT
P_1	3	3	3	0
P_2	3	6	6	3
P_3	20	26	26	6

Q) If first process has more burst time then turn around time and avg. waiting time increases for all. (avg)

2. SJF (shortest job first scheduling):- Nonpreemptive on burst time

Pro No	BT	AT
A	3	0
B	6	2
C	4	4
D	5	6
E	2	8



avg. turn around time = 38/5

avg. waiting time = 18/5

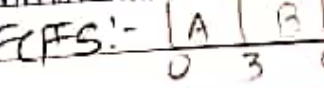
Pro No.	B.T.	A.T	F.T.	T.A.T	W.T
A	3	0	3	3	0
B	6	2	9	7	1
C	4	4	15	11	7
D	5	6	20	14	9
E	2	8	11	3	1
				38	18

FCFS

FT	TAT	WT

Q) Minimum arrival time if same minimum burst time

FCFS:-

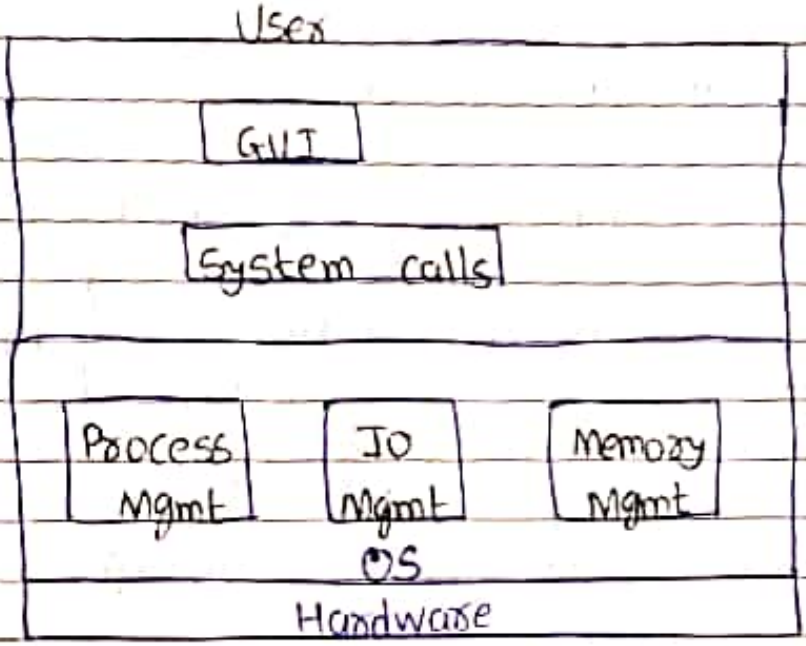


Q) Problem:- predictability for shortest job
starvation (if all small jobs are coming)

3. Shortest remaining time first:- preemptive
snatch CPU when one process is executing

operating system:- provides interface services
resource mangement
process management

OS communicates between hardware & OS and provides abstraction of HW.



Main memory :- speed
shell script :- scheduling & testing algo.

Services by OS:-
↳ program execution
↳ I/O operations
↳ File system manipulation
↳ Error detection
↳ resource allocation (CPU cycle, memory)
↳ security (authentication)