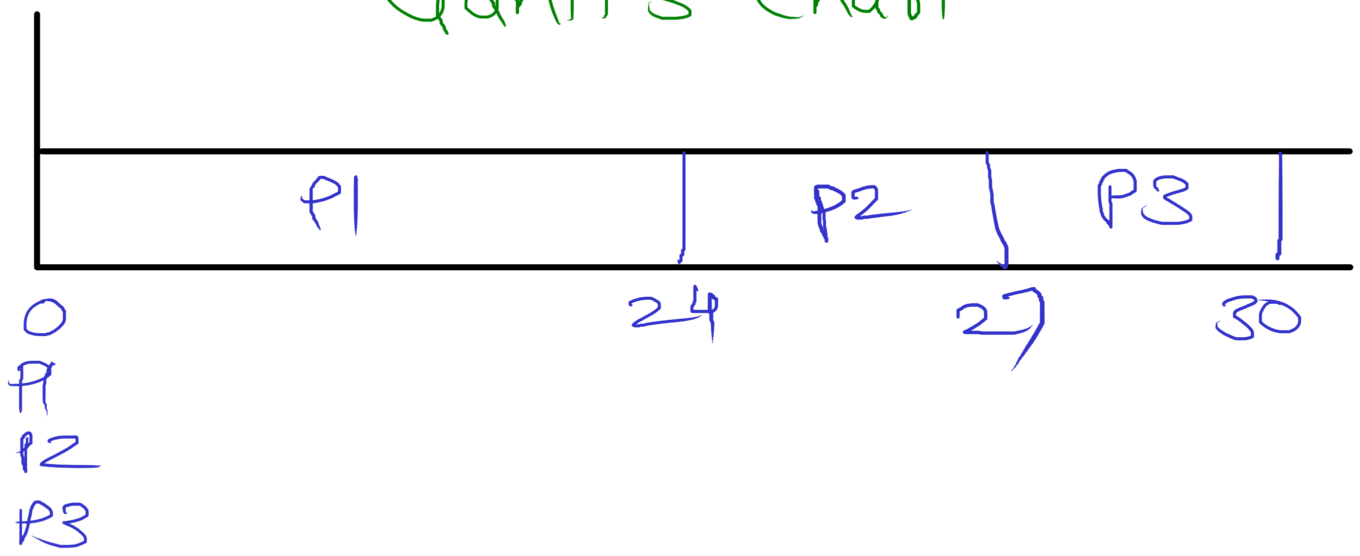
FCFS (First Come First Serve)

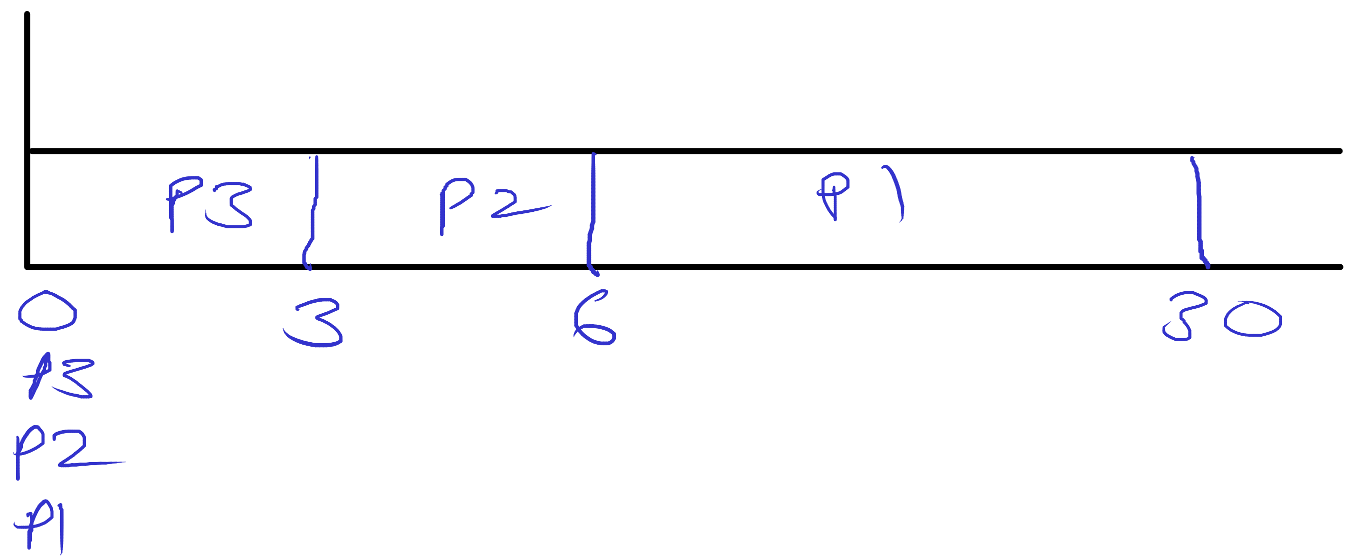
(Non - Preemptive)

Process	Arrival	CPU Burst	WT	RT	TAT
P1	0	24	0	0	24
P2	0	3	24	24	27
P3	0	3	27	27	30

Process	Arrival	CPU Burst	WT	RT	TAT
P3	0	3	0	0	3
P2	0	3	3	3	6
P1	0	24	6	6	30

Gantt's chart



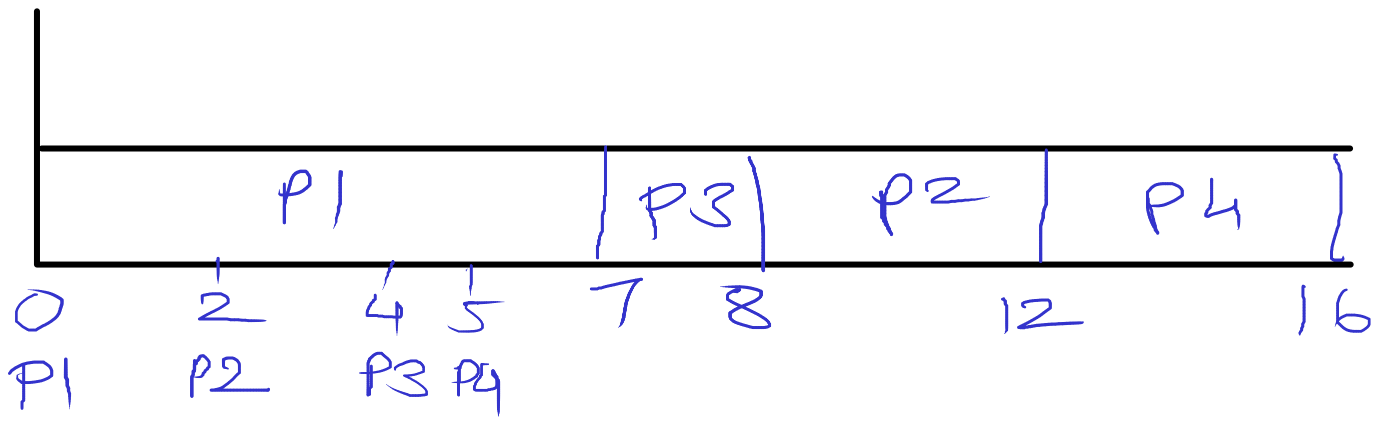


Convoy effect :
due to arrival of longer process early,
all other processes has to wait for longer time.

SJF (Shortest Job First)

(Non-Preemptive)

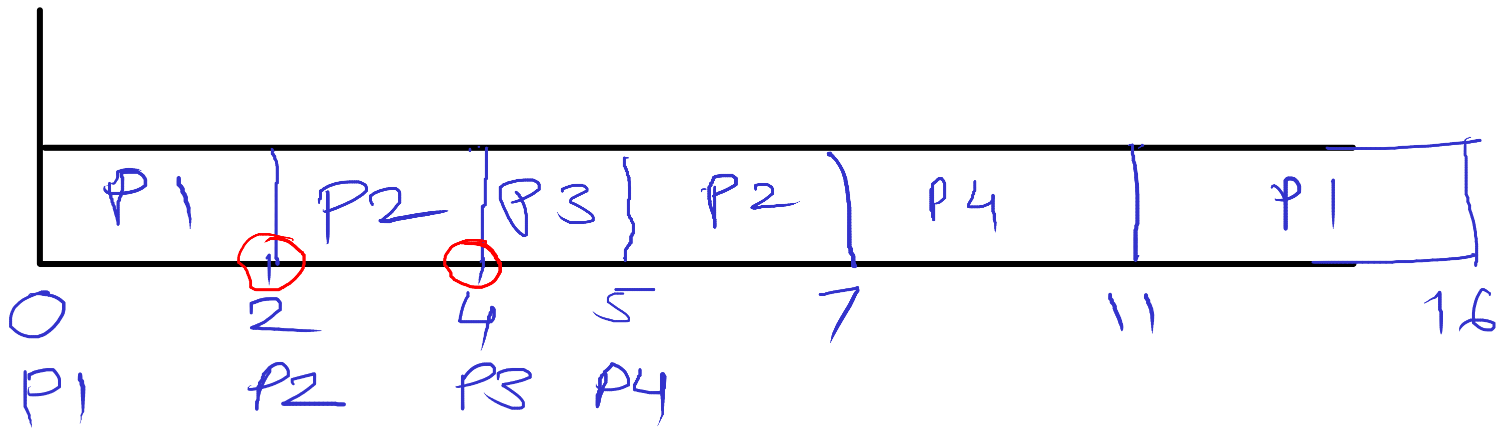
Process	Arrival	CPU Burst	WT	RT	TAT
P1	0	7	0	0	7
P2	2	4	6	6	10
P3	4	1	3	3	4
P4	5	4	7	7	11



(Preemptive)

(Shortest Remaining Time First)

Process	Arrival	CPU Burst		WT	RT	TAT
P1	0	7	5	9	0	16
P2	2	4	2	1	0	5
P3	4	1		0	0	1
P4	5	4	4	2	2	6
					0.5	



starvation :

due to longer CPU time (duration), process will not get enough CPU time for execution.

Priority

(Non Preemptive)

Process	Arrival	CPU Burst	Priority	WT	RT	TAT
P1	0	10	3	6	6	16
P2	0	1	1 (H)	0	0	1
P3	0	2	4 (L)	16	16	18
P4	0	5	2	1	1	6

P2	P4	P1	P3

0, 1, 6, 16, 18

0: P1, P2, P3, P4

(Preemptive)

Process	Arrival	CPU Burst	Priority	WT	RT	TAT
P1	0	10	3	6	6	16
P2	1	1	1	0	0	1
P3	3	2	4	13	13	15
P4	0	5	2	1	0	6

4

P4	P2	P4	P1	P3

0, 1, 2, 3, 6, 16, 18

0: P1, P4
1: P2
3: P3

P1 (6)
P2 (9)
P3 (5)
P4 (7)
P5 (6)
P6 (8)
P7 (4)

P1
P3
P5
P4
P7
P6
P2

Starvation :— due to low priority of process, not getting enough CPU time for execution.

Aging :— increase priority of process gradually.

RR (Round Robin)

(Preemptive)

Process	CPU Burst	
P1	53	33, 13
P2	17	
P3	68	48, 28, 8
P4	24	4

	WT	RT	TAT
P1	0+57+24	0	134
P2	20	20	37
P3	37+40+17	37	162
P4	57+40	57	121

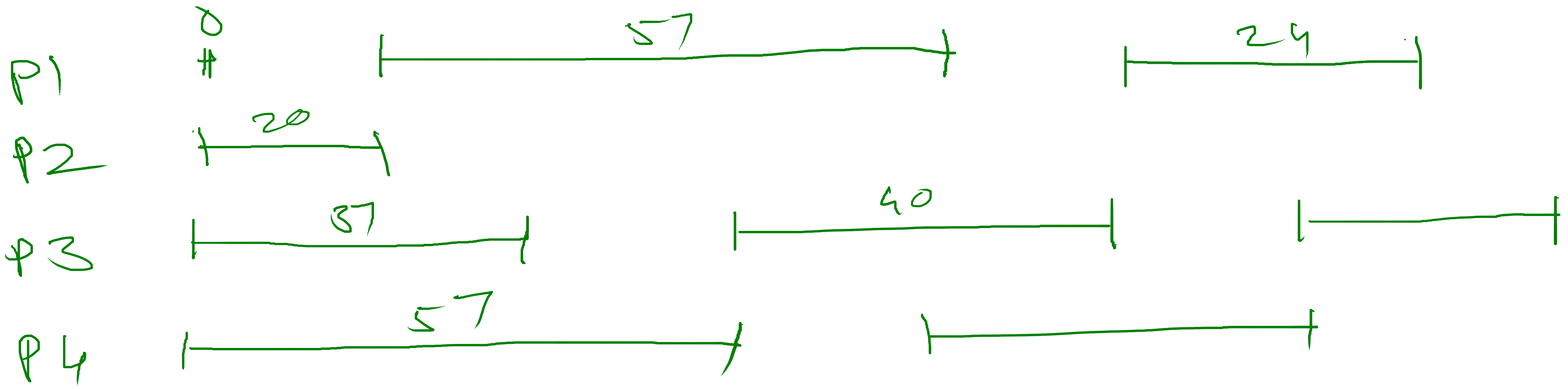
TQ = 20

TQ = 100
↳ behave like FCFS

TQ = 3
↳ CPU overhead will increase

P1	P2	P3	P4	P1	P3	P4	P1	P3	P3

0, 20, 37, 57, 77, 97, 117, 121, 134, 154, 162



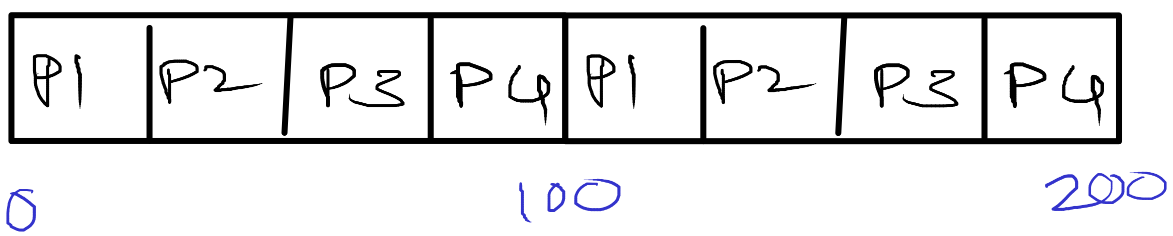
Fair Share

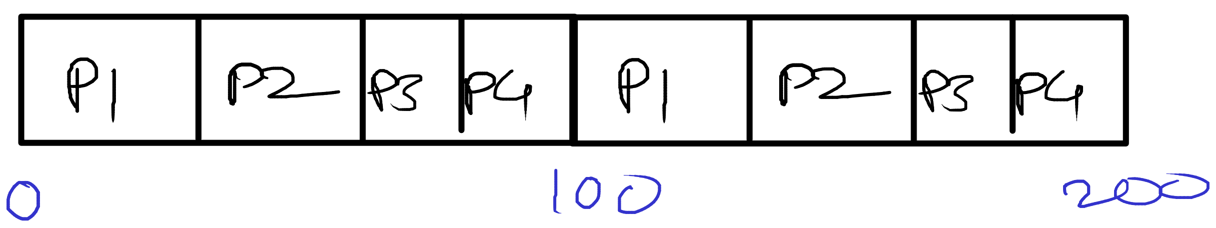
- CPU time is divided into time slices (epoch)
- some share of each epoch is given to the processes which are in ready queue.
- share is given to the process on the basis of their priority
- priority of every process is decided by its nice value
- nice values range ---> -20 to +19 (40 values)
  - * -20 - highest priority
  - * +19 - lowest priority

Process	Nice Value
P1	10
P2	10
P3	10
P4	10

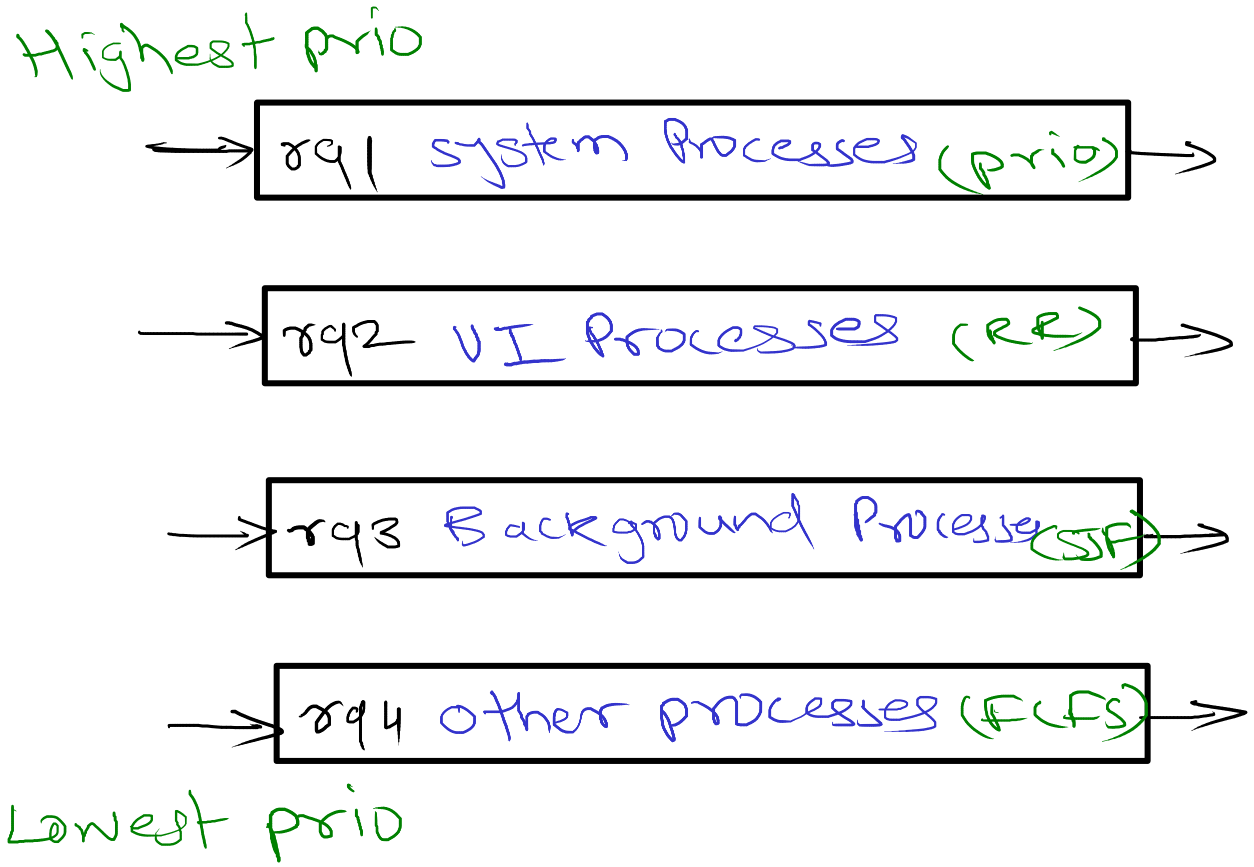
Epoch - 100

Process	Nice Value
P1	5
P2	5
P3	10
P4	10

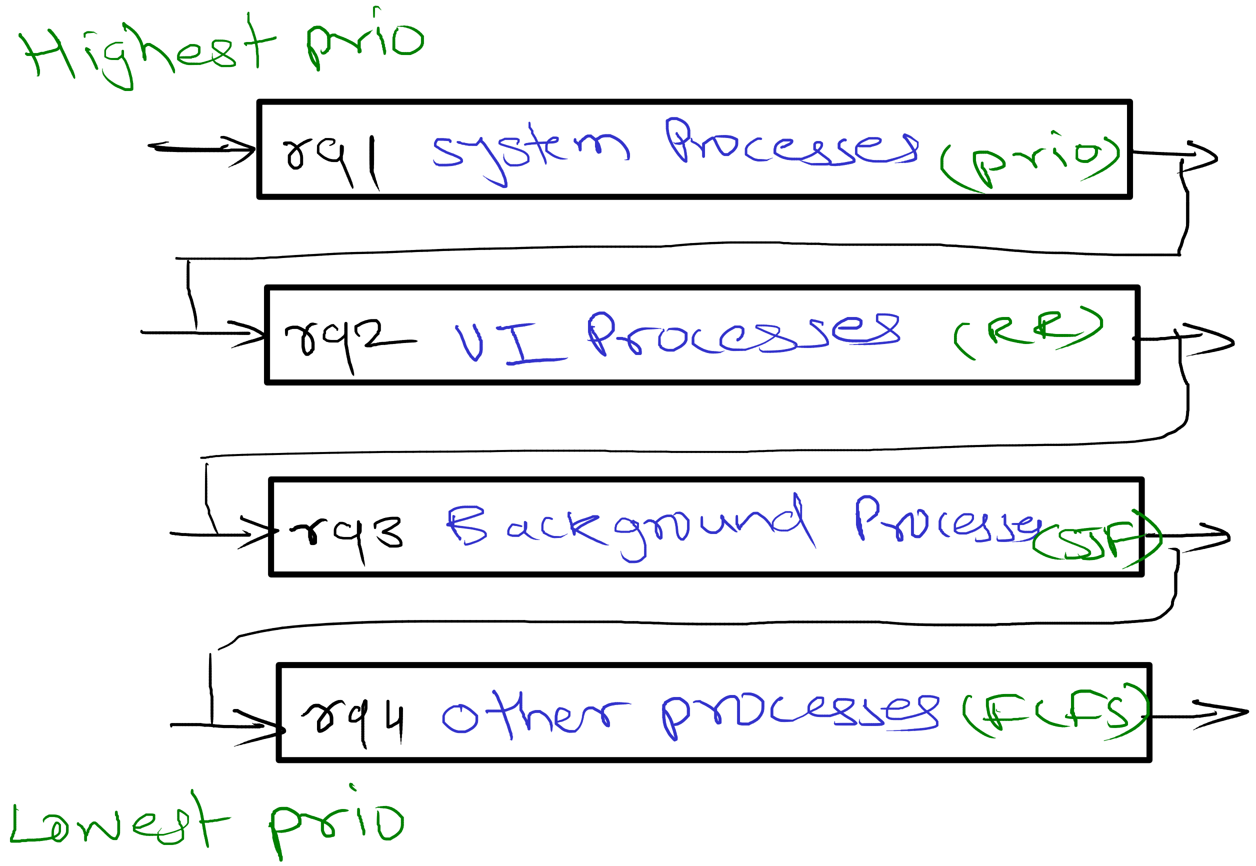




Multi Level Ready Queue



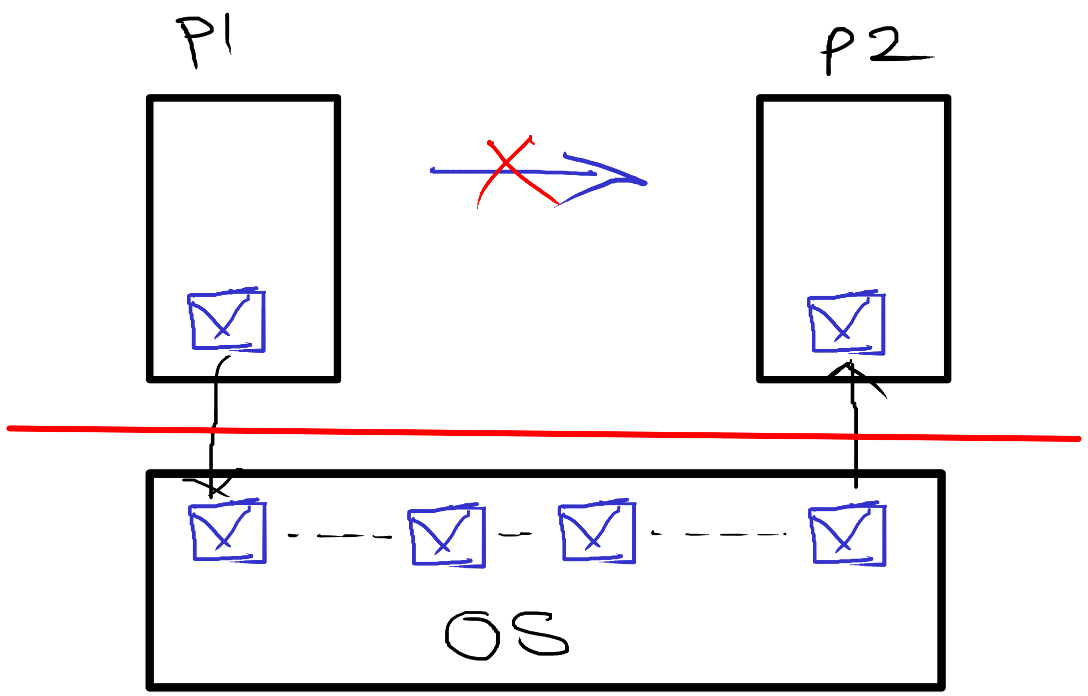
Multi Level Feedback Ready Queue

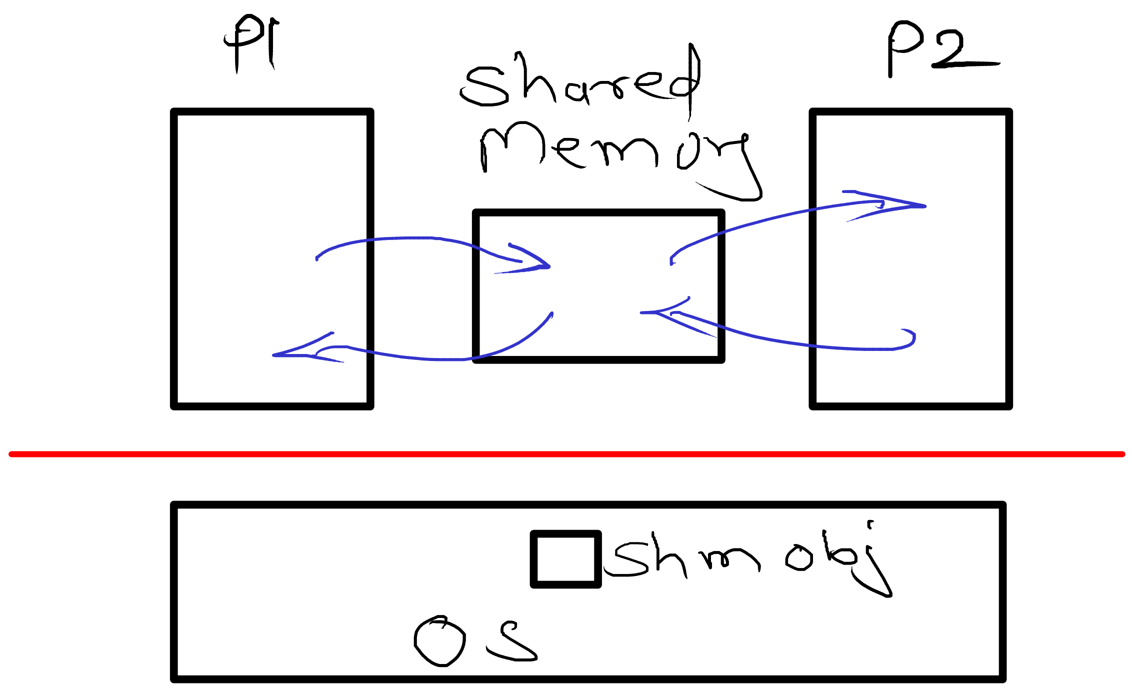


IPC (Inter Process Communication)

Message Passing Model

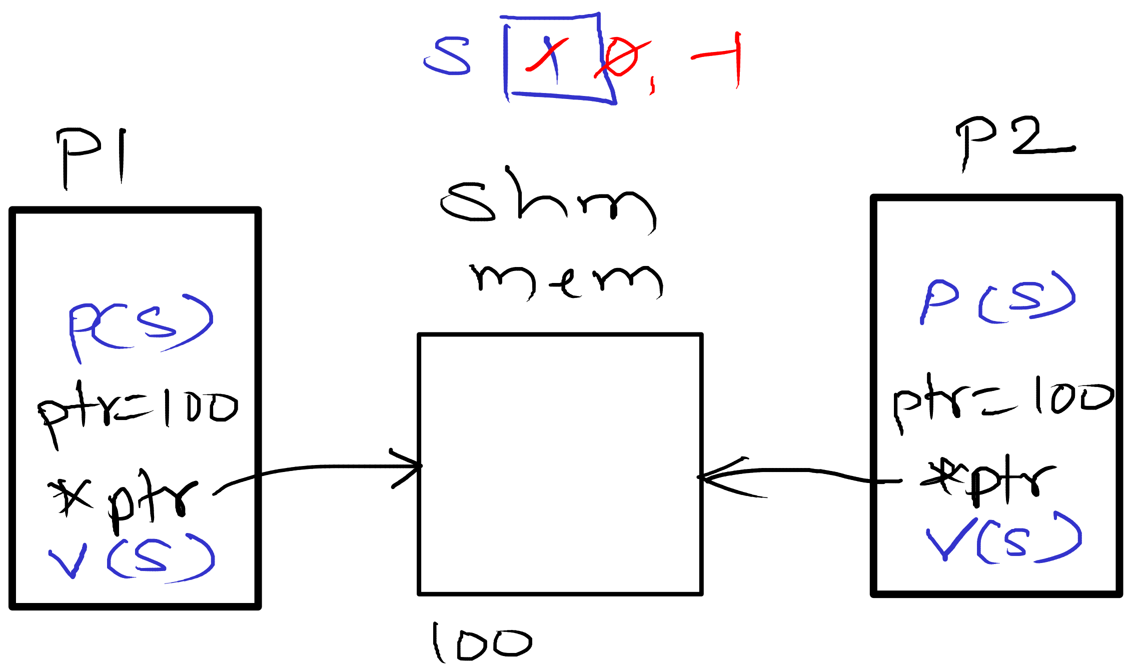
Shared Memory Model

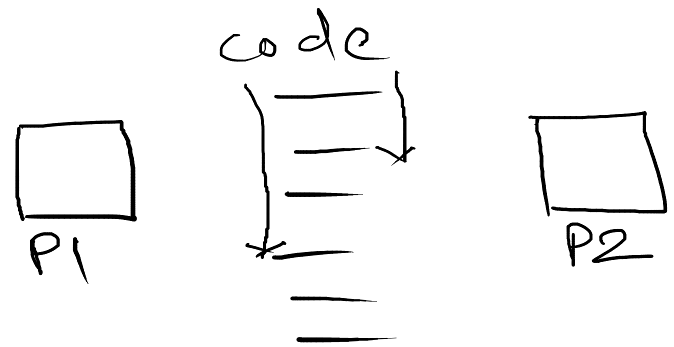




1) Signal
2) Message queue
3) Pipe
4) Socket

5) Shared Memory (Fastest IPC)





Critical section

piece of code which two processes can execute at a time

Peterson's problem :-

two or more processes want to access same variable at same time, this will give you wrong result (Data inconstancy).

Solution for above two problem is synchronization

- There are two types of sync. mechanisms
  1) Semaphore
  2) Mutex

Semaphore :

- internally it is a counter

Operations:

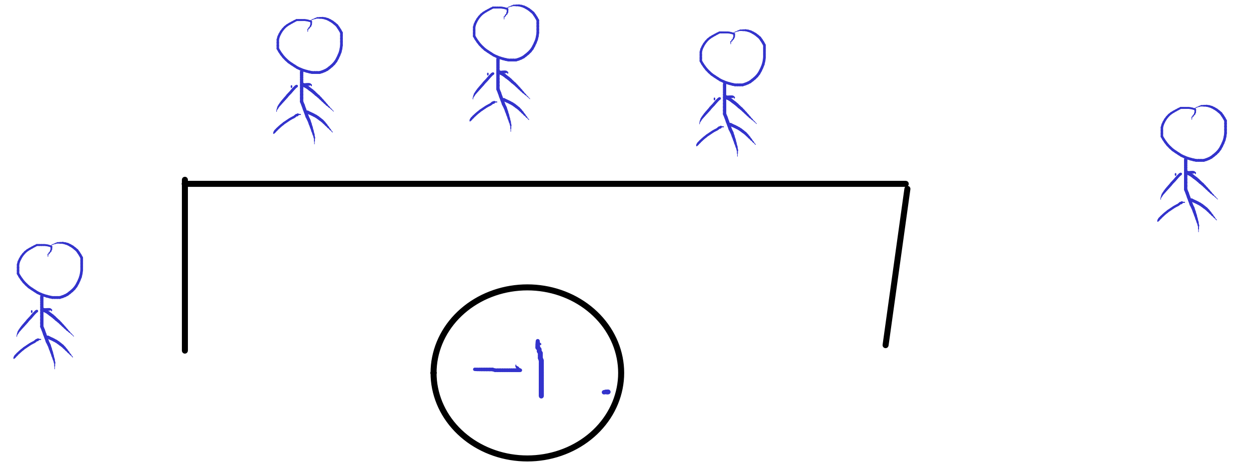
1) Dec op / wait() / p():

- dec semaphore counter
- if counter < 0, then block current process

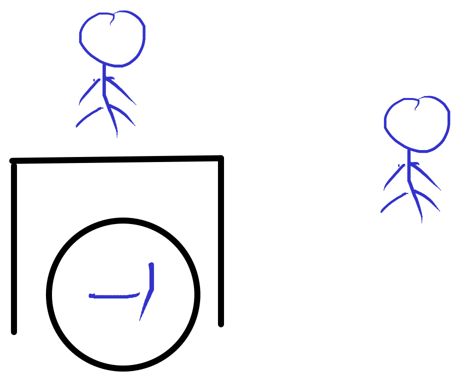
2) Inc op / post() / v():

- inc semaphore counter
- if any process is blocked on semaphore, wakeup it

Counting Semaphore



Binary Semaphore

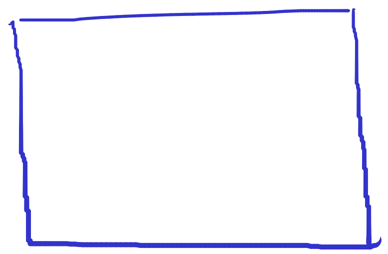


Mutex : (Mutual Exclusion)

- similar to binary semaphore
- operations - lock() / unlock()
- the process which will lock the mutex, will become owner of that process
- only owner can unlock the mutex

	Semaphore	Mutex
create –	sem_init()	pthread_mutex_create()
dec –	sem_wait()	pthread_mutex_lock()
inc –	sem_post()	pthread_mutex_unlock()
destroy –	sem_ctl()	pthread_mutex_destroy()

pthread — thread library



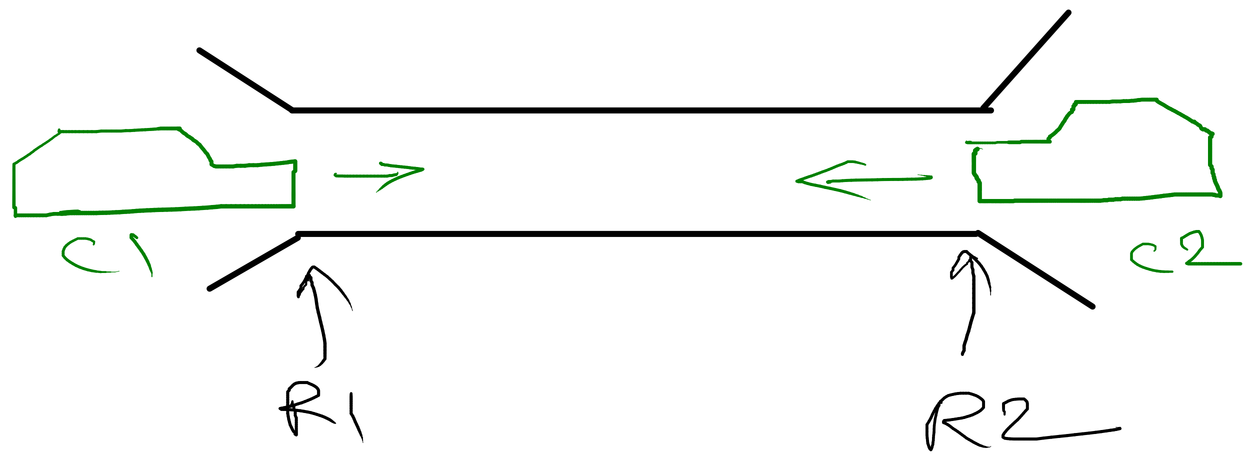
balance

```
s = sem_init();

void inc( ){
    sem_wait();
    balance++;
    sem_post();
}

void decc( ){
    sem_wait()
    balance--;
    sem_post()
}

sem_destroy();
```

Deadlock :

- indefinte waiting for resource
- deadlock will occur when all 4 conditions will hold true.

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



Preventions :

- while writing OS, one condition is kept always false

Avoidance :

- Resource allocation graph
- Bankers algorithm
- safe state algorithm

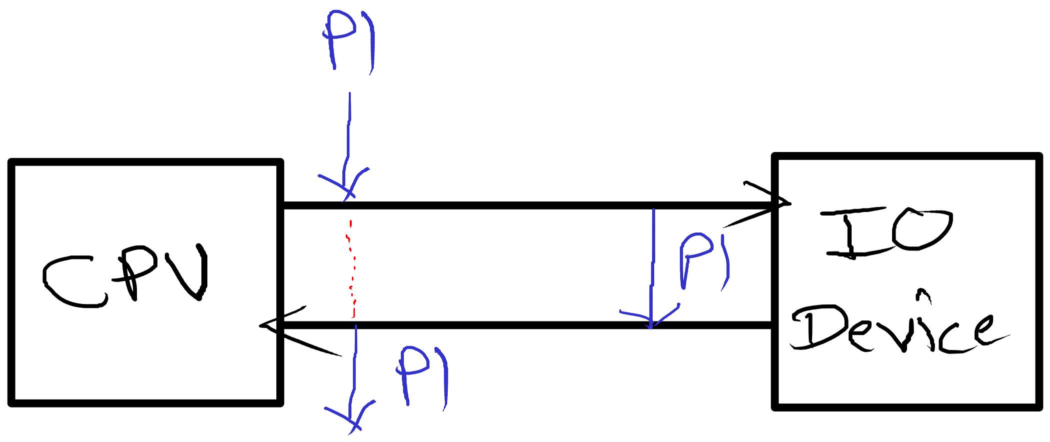
Recovery :

- resource preemption
- forceful termination

IO Management

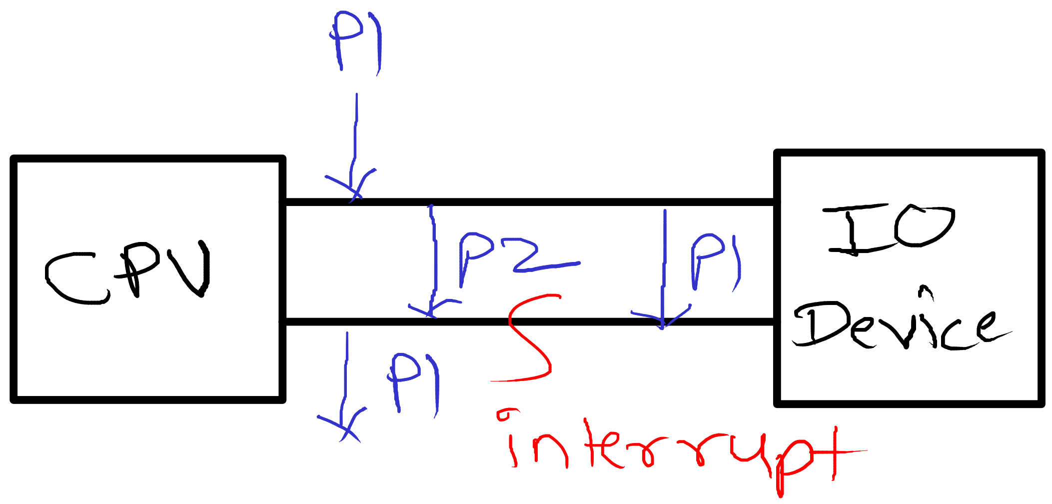
Synchronous IO

Hardware Techinque - Polling

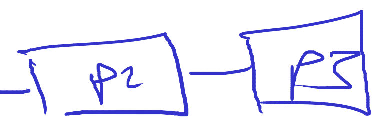


Asynchronous IO

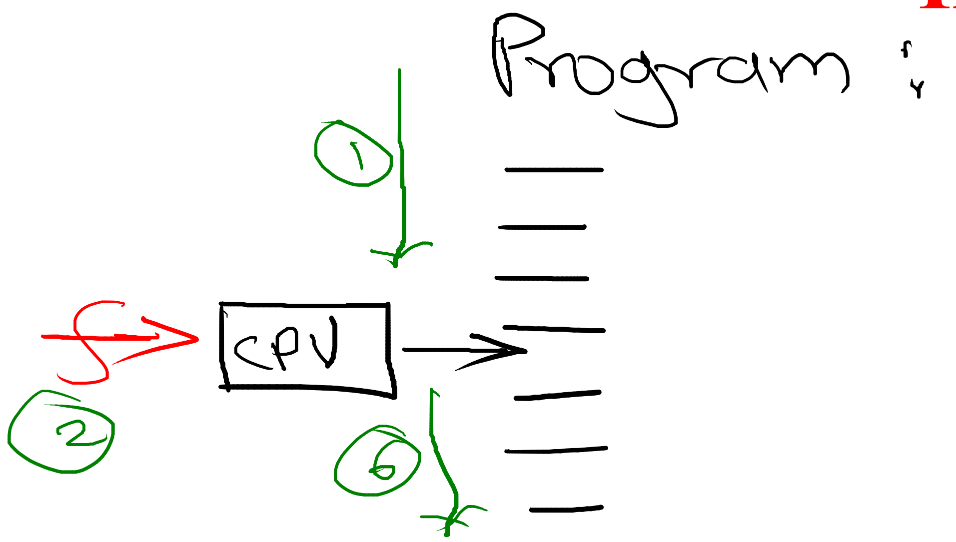
Hardware Techinque - Interrupt



Device	Status (busy/Idle)	Waiting queue
Printer	busy	*
Keyboard		
⋮		

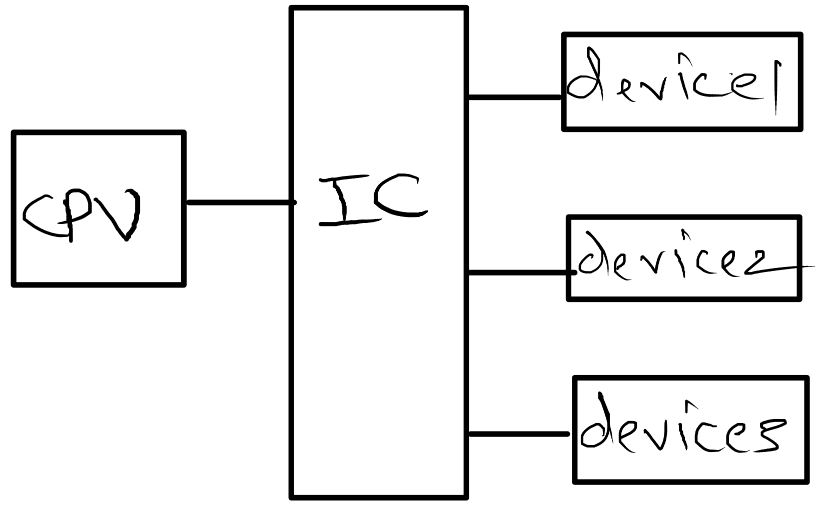


Interrupt Service

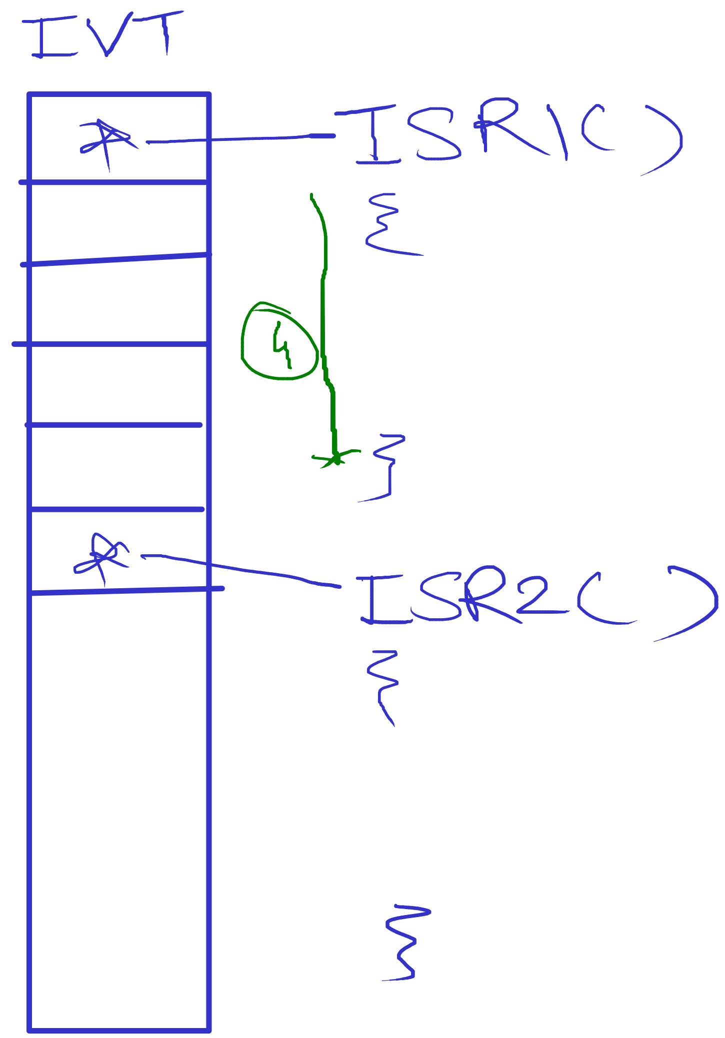


ARM7 – VIC
– NVIC

x86 – 8259
– apic



```
interrupt_handler( )
{
    1) save execution context
       of current process
    2) find ISR from IVT for
       corresponding interrupt.
    3) call ISR
    4) restore execution context
       of paused process.
}
```



Dual Mode Operation

Program:

1

2

6

user mode

interrupt_handler() {

3

5

}

ISR() {

4

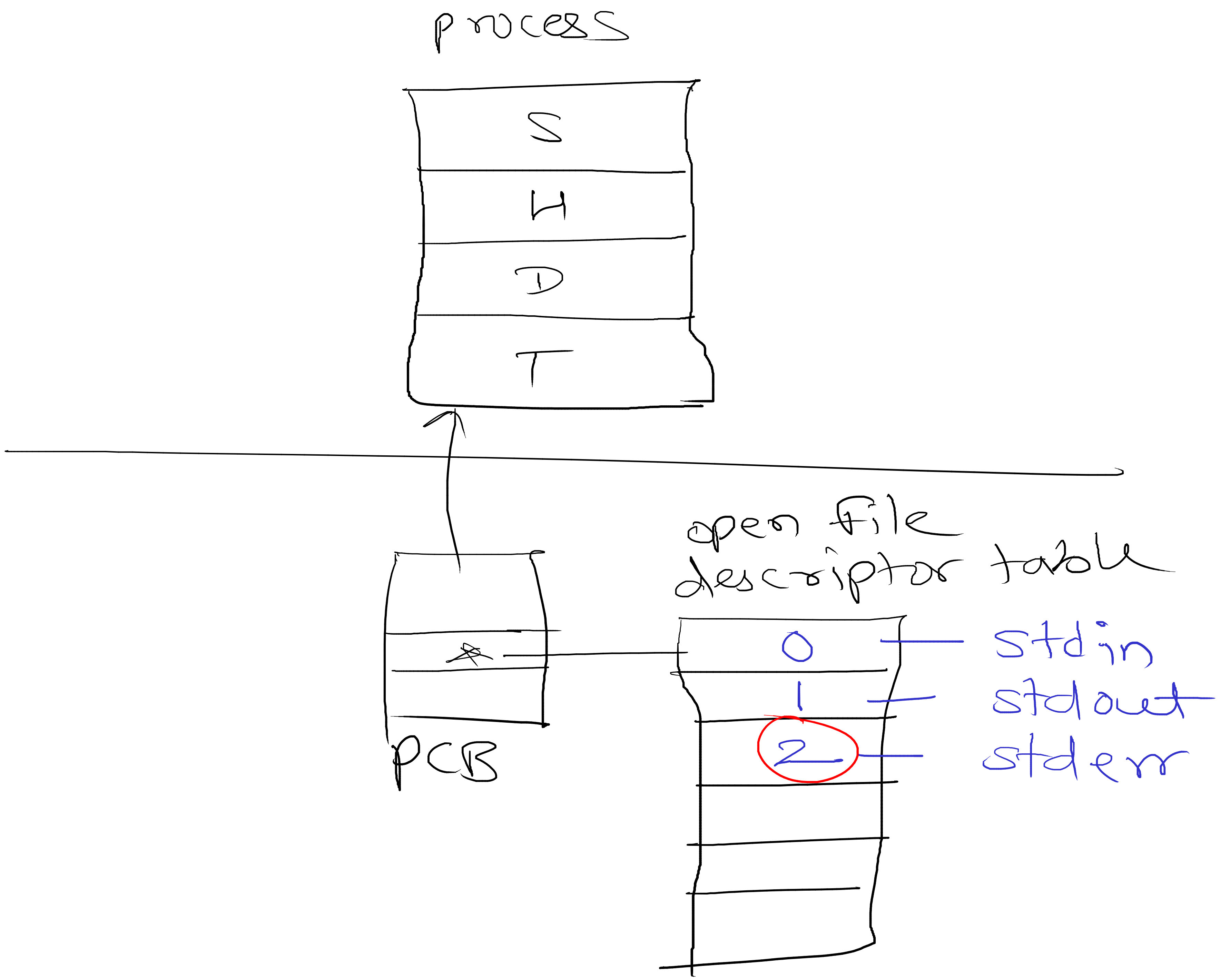
}

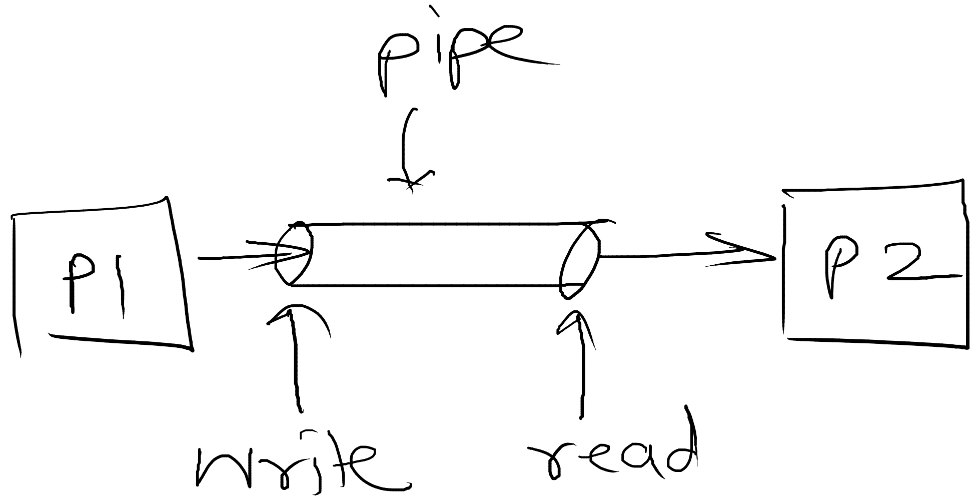
kernel mode

CPU mode

0 – kernel mode / priviledged

1 – user mode / unpriviledged


process
S
H
D
T
PCB
open file descriptor table
0 stdin
1 stdout
2 stderr


pipe
P1
P2
write
read

sort

uniq