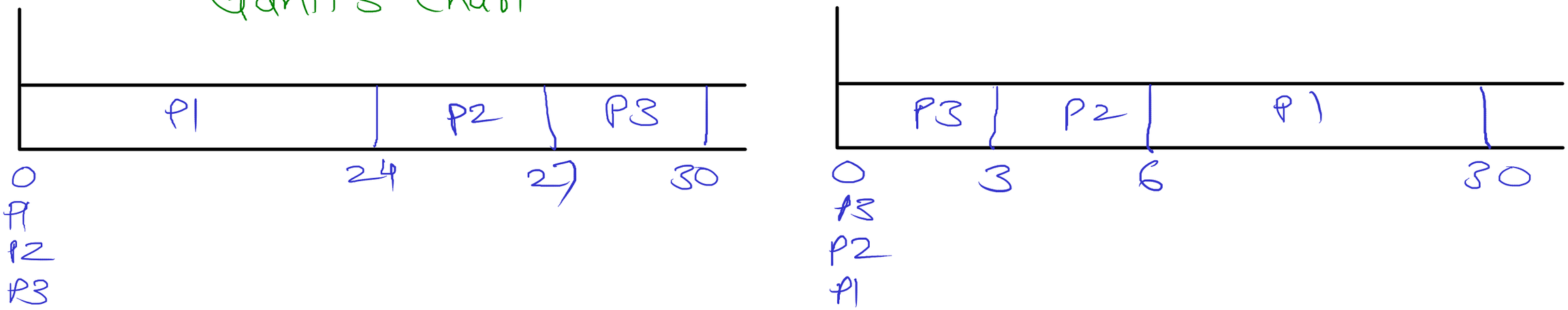
FCFS (First Come First Serve)

(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



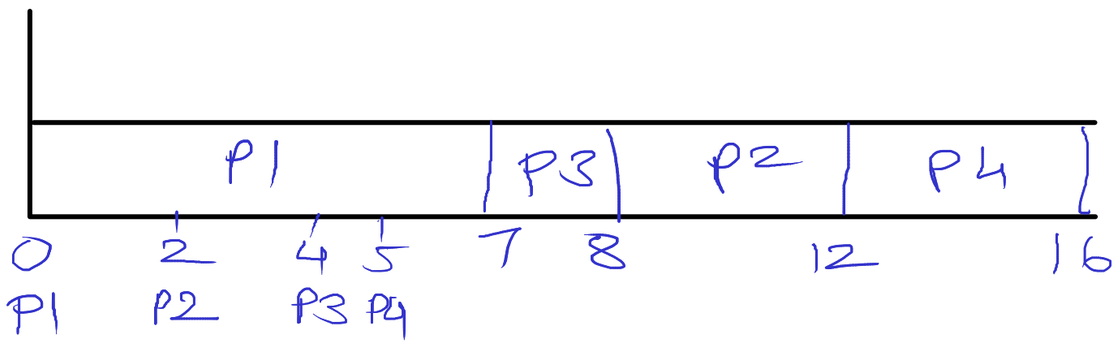
Convoy effect:

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

SJF (Shortest Job First)

(Non-Preemptive)

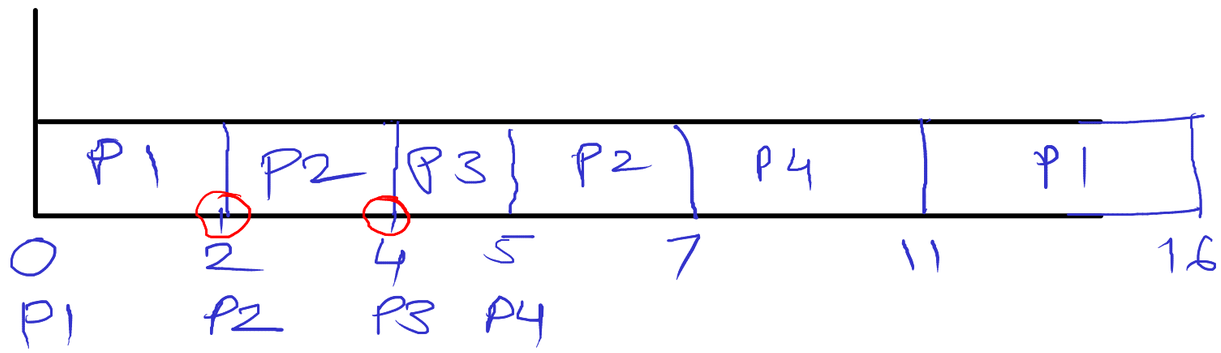
Process	Arrival	CPU Burst	WT	RT	TAT
P1	0	7	0	0	7
P2	2	4	6	6	10
P3	4	1	3	3	4
P4	5	4	7	7	11



(Preemptive)
(Shortest Remaining Time First)

Process	Arrival	CPU Burst
P1	0	7
P2	2	4
P3	4	1
P4	5	4

WT	RT	TAT
9	0	16
1	0	5
0	0	1
2	2	6
	0.5	



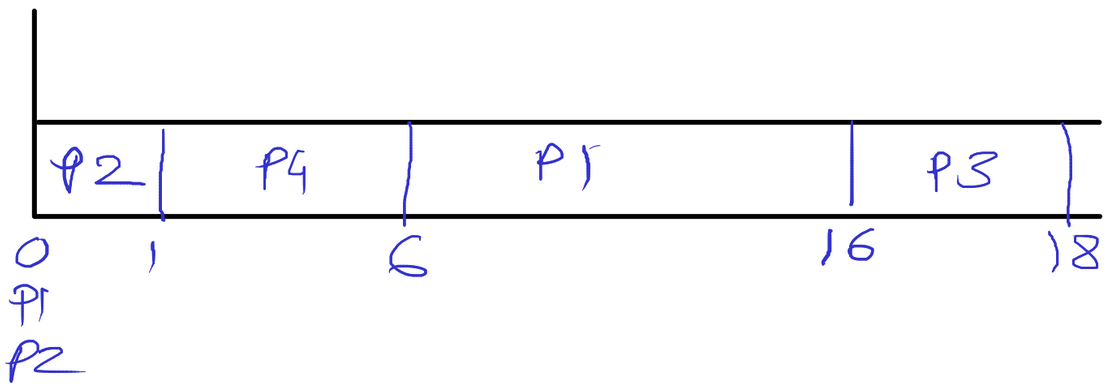
starvation :

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

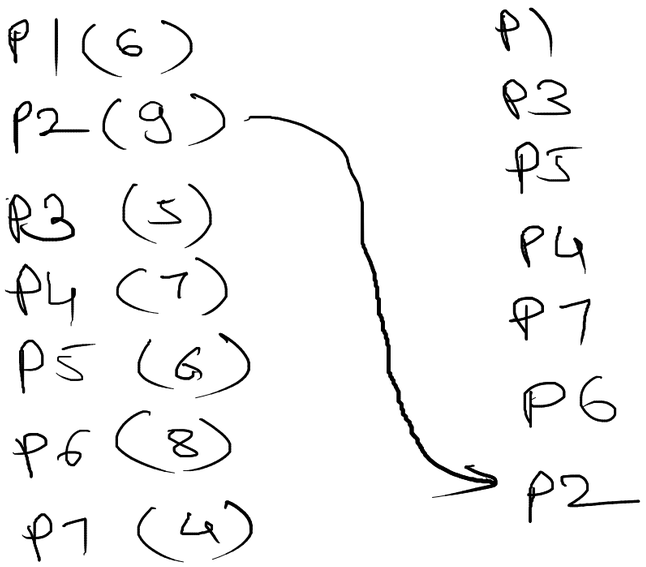
Priority

(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

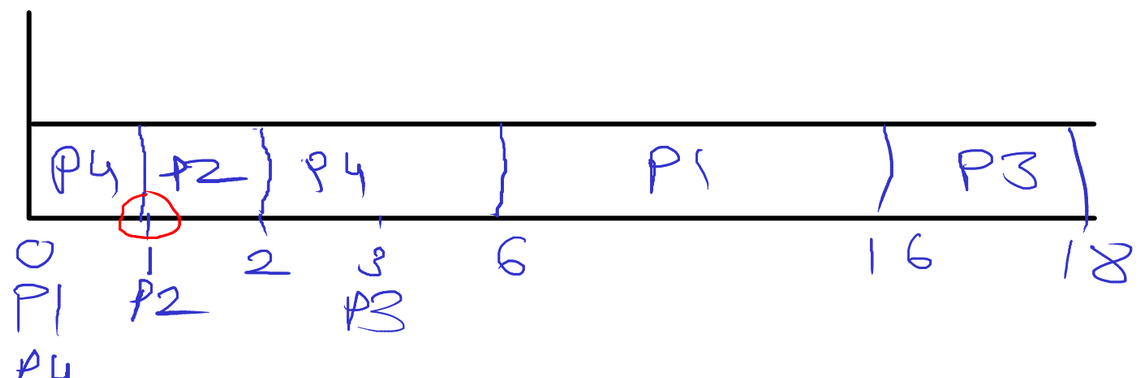


P1
P2
P3
P4



(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



P4

Starvation :—

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

Aging :—

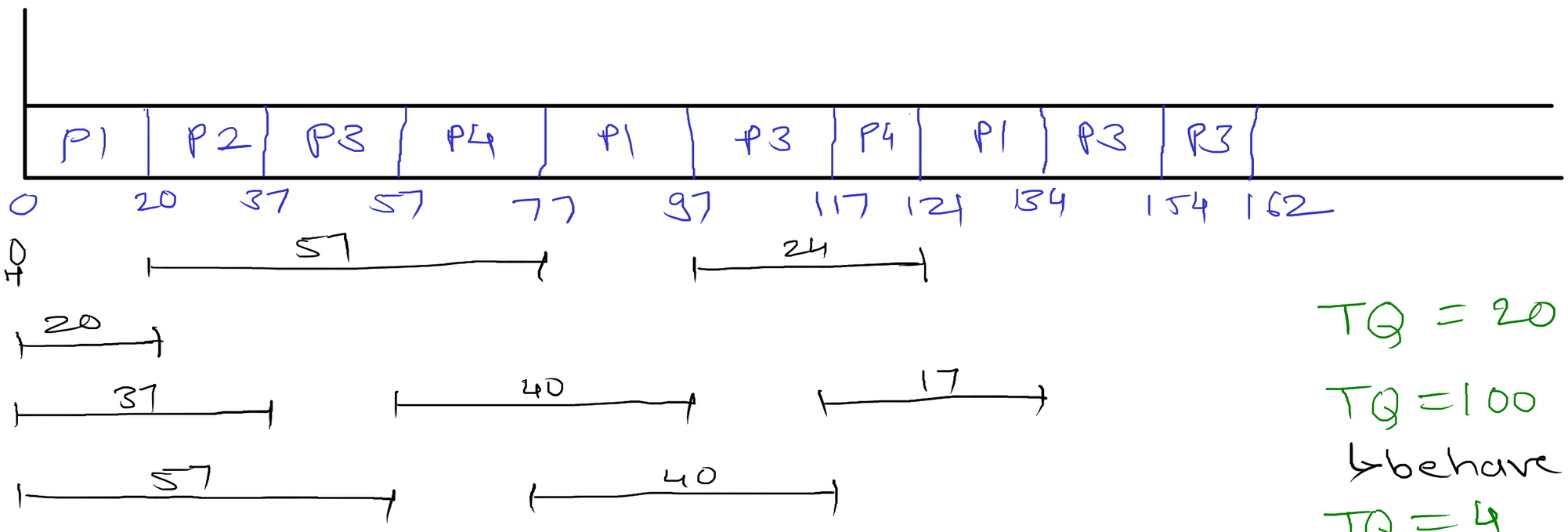
increase priority of process gradually.

RR (Round Robin)
(Preemptive)

Process	CPU Burst
P1	53
P2	17
P3	68
P4	24

53, 13, 0
0
48, 28, 8, 0
4, 0

	WT	RT	TAT
P1	0+57+24	0	134
P2	20	20	37
P3	37+40+17	37	162
P4	57+40	57	125



TQ = 20

TQ = 100
↳ behave like FCFS

TQ = 4
↳ CPU overhead will increase

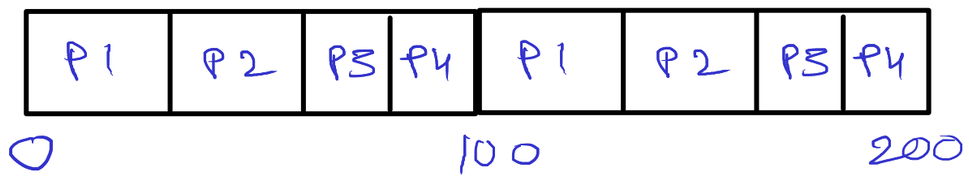
Fair Share

- 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

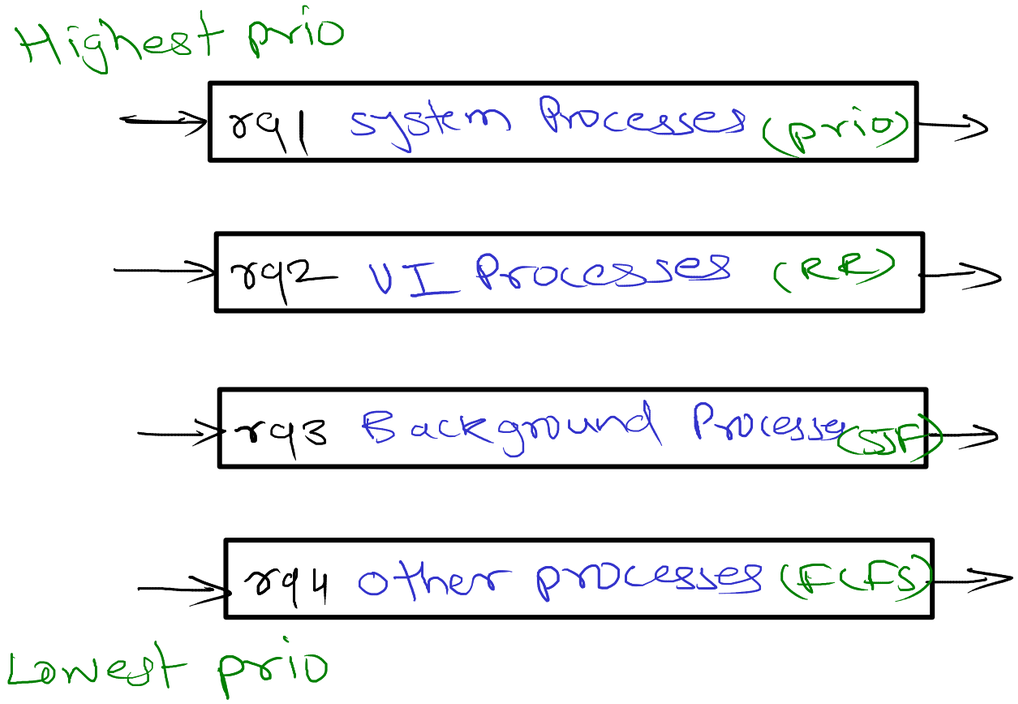
Process	Nice Value
P1	10
P2	10
P3	10
P4	10

Epoch - 100

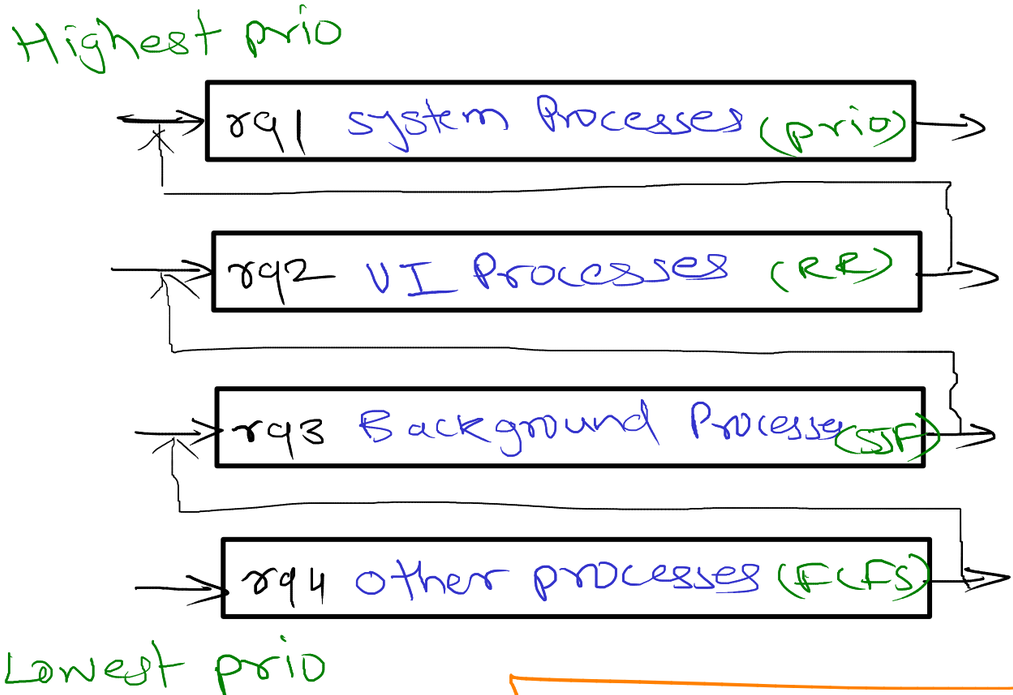
Process	Nice Value
P1	5
P2	5
P3	10
P4	10



Multi Level Ready Queue



Multi Level Feedback Ready Queue



man 7 sched

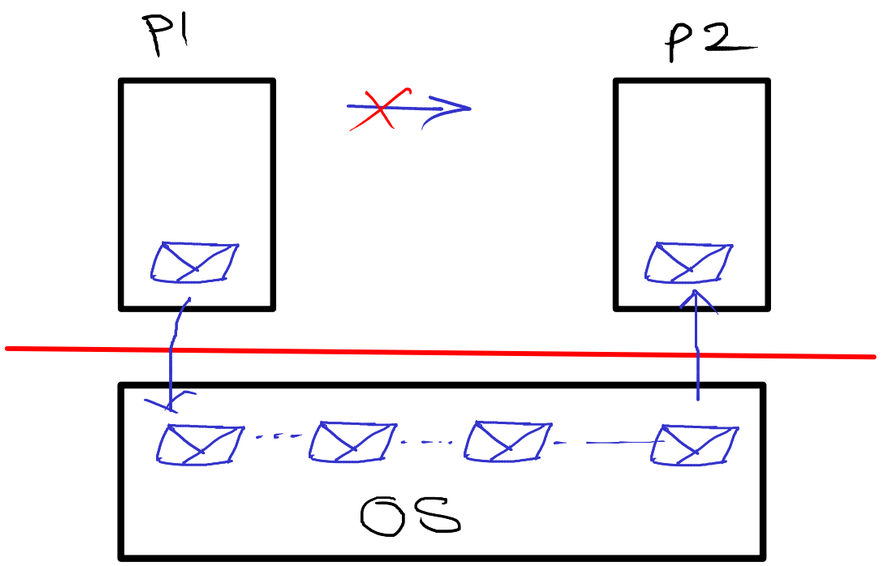
Linux scheduling policies

- Non Real time policies
  - 1) SCHED_OTHER
  - 2) SCHED_BATCH
  - 3) SCHED_IDLE

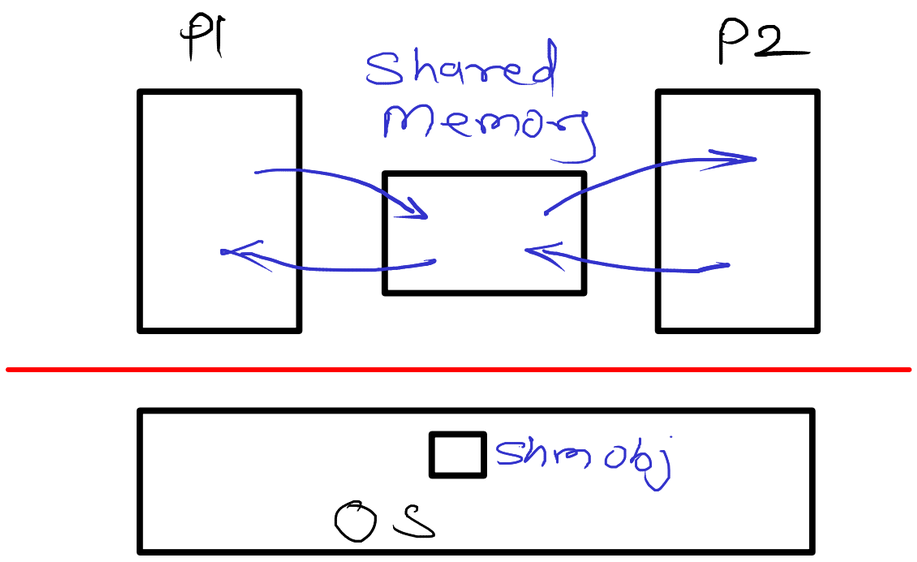
  } CFS
- Real time policies
  - 4) SCHED_FIFO — FCFS
  - 5) SCHED_RR — RR

IPC (Inter Process Communication)

Messaging Passing Model

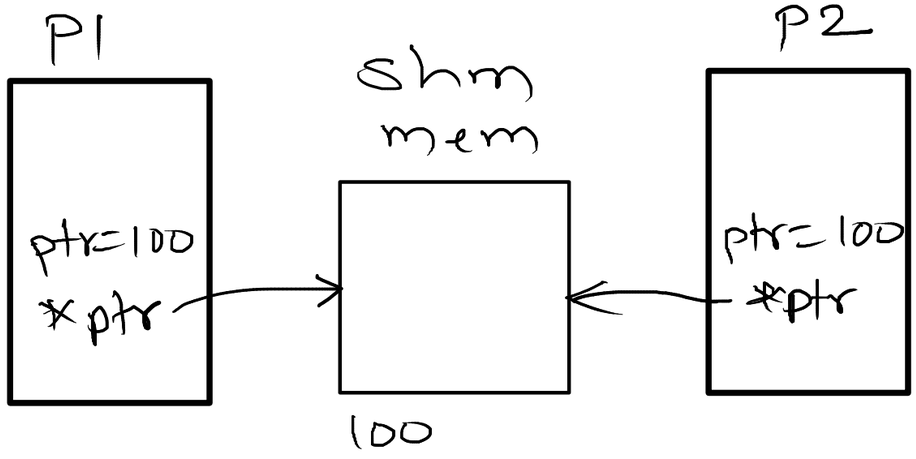


Shared Memory Model



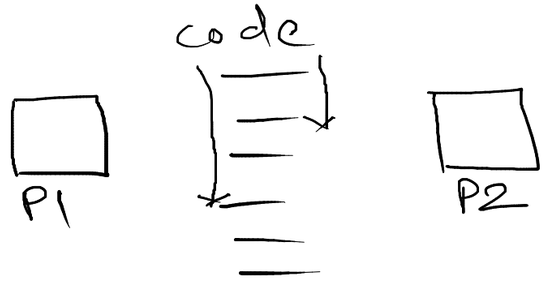
1) Message queue
2) Signal
3) Pipe
4) Socket

5) Shared Memory (Fastest)



Peterson's problem :—

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



Critical section

piece of code which two processes can execute at a time

Solution for above two problem is synchronization

— There are two types of sync. mechanisms

1) Semaphore
2) Mutex

Semaphore :

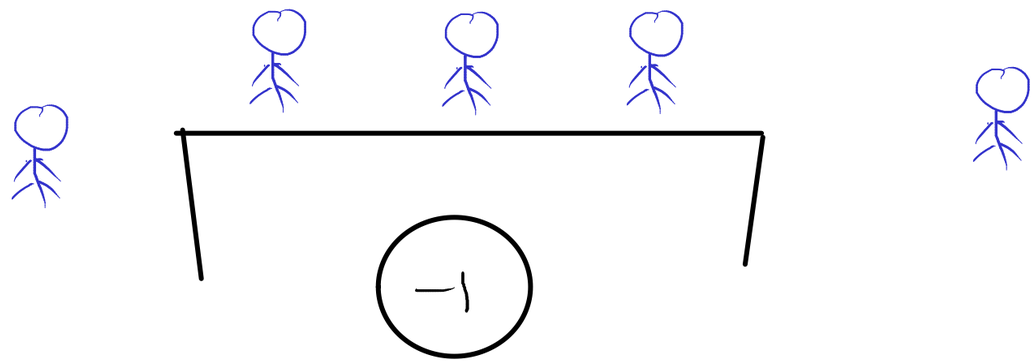
- internally a counter

Operations :

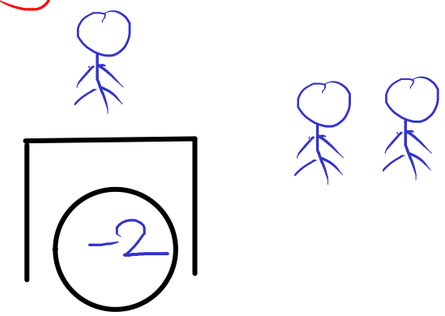
1> DEC
   i> decrement counter
   ii> if counter < 0, then
      block the current process

2> INC
   i> increment counter
   ii> if one or more processes are blocked on counter then wake up one.

Counting Semaphore

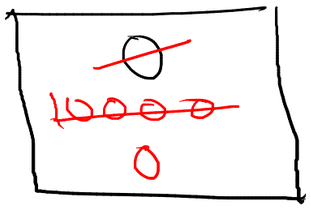


Binary semaphore



Mutex : (Mutual Exclusion)

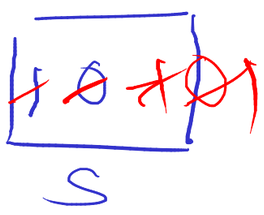
- similar to binary semaphore
- Mutex operations - lock or unlock
- the process who will lock mutex, will become owner of mutex.
- only owner can unlock the mutex



s = sem_init(1);

```
void deposit( )
{
    sem_wait(s)
    balance += 10000;
    sem_post(s)
}
```

```
void withdraw( )
{
    sem_wait(s);
    balance -= 10000;
    sem_post(s);
}
```



sem_ctl(s);

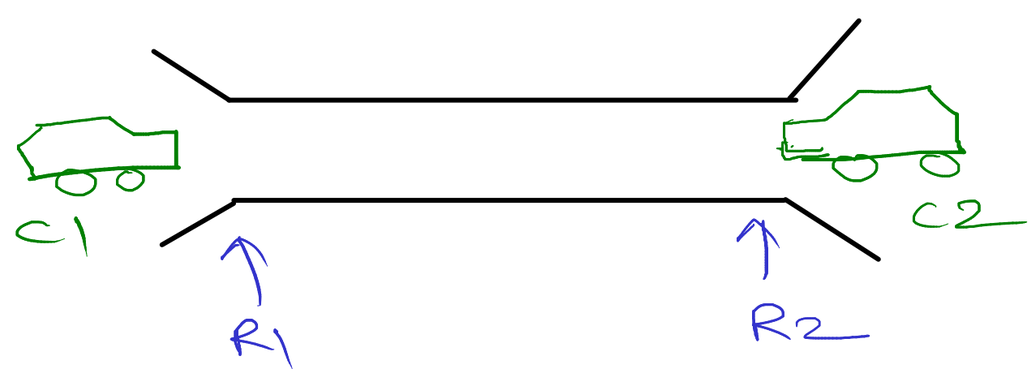
pthread_mutex_create( )
pthread_mutex_lock( )
pthread_mutex_unlock( )
pthread_mutex_destroy( )

pthread - library

Deadlock :

- indefinite waiting for resource
- there are four conditions and when all four condition are true at same time, deadlock

1) Mutual Exclusion
2) No preemption (Resource)
3) Hold & wait
4) Circular wait



Prevention

- while writing OS, one condition / 4 is always kept false.

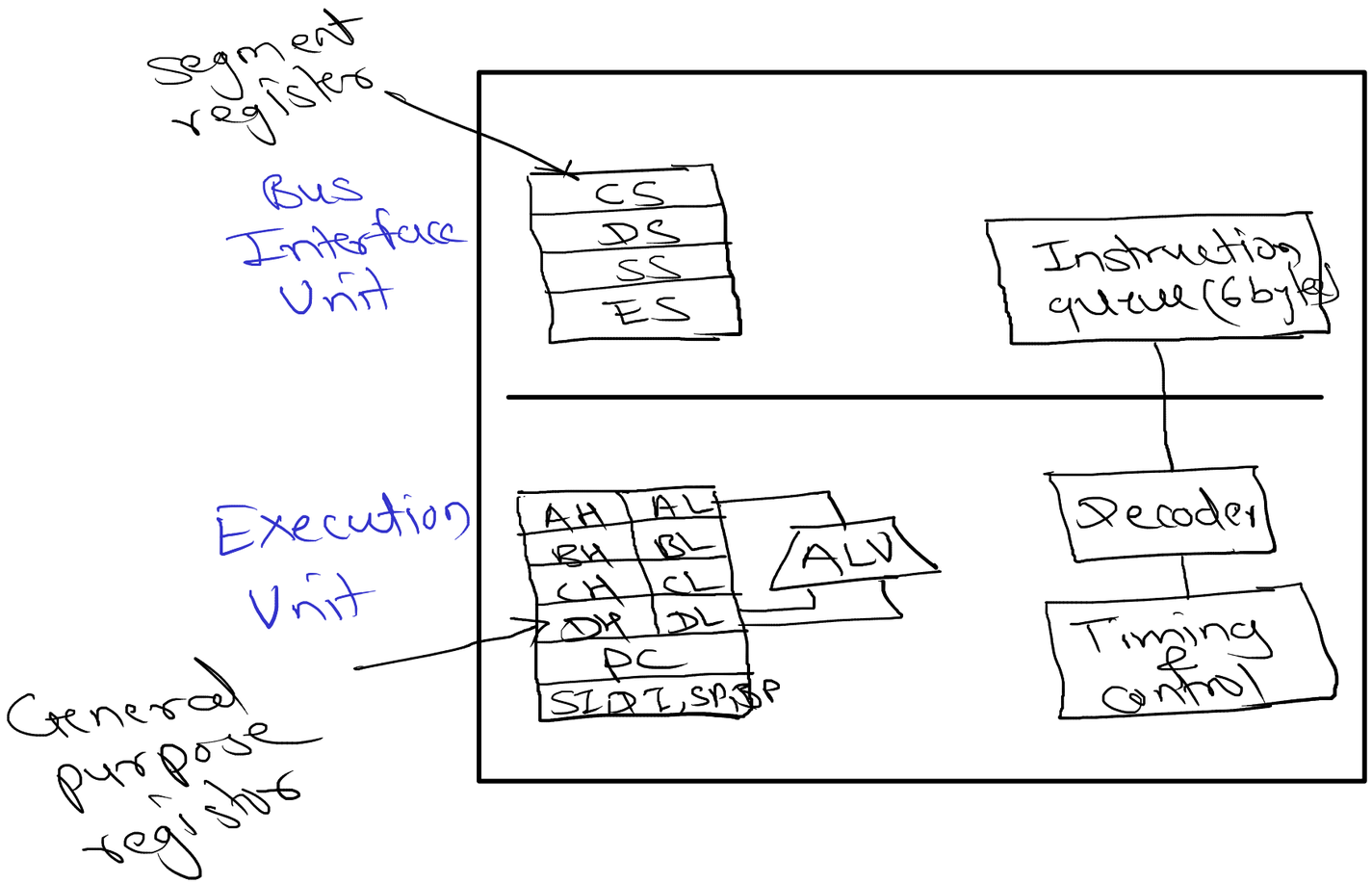
Avidance

1) Resource allocation graph
2) Banker's Algorithm
3) Safe state Algorithm

Recovery

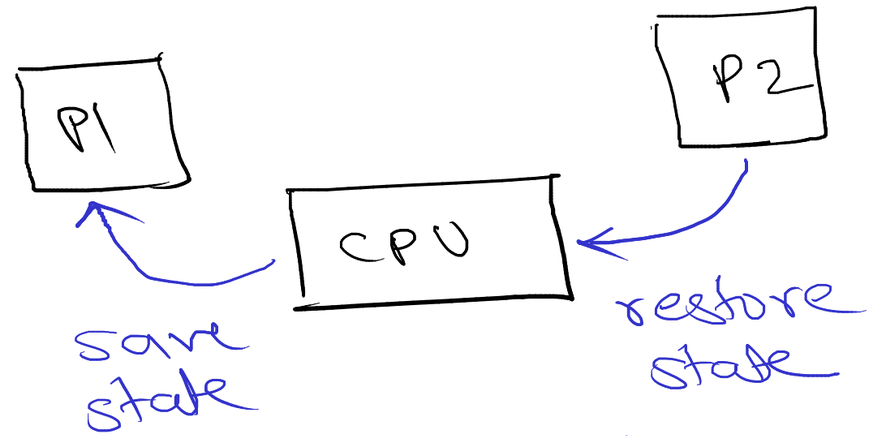
1) resource preemption
2) process termination

CPU Architecture



Context switching

Execution context:
- values of CPU registors



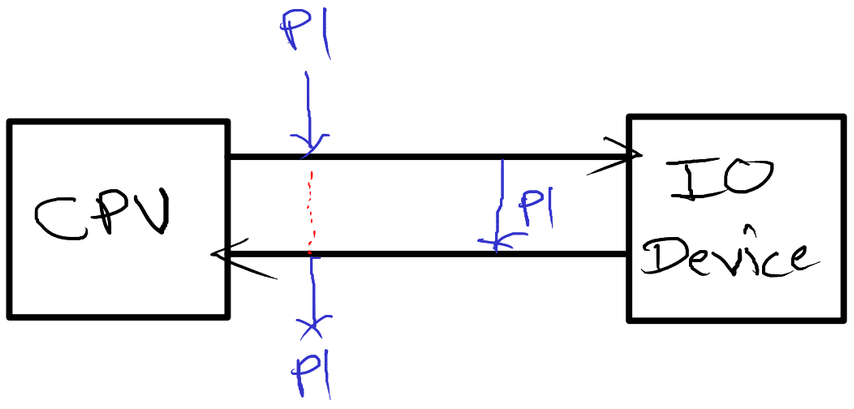
CPU Dispatcher

Dispatcher latency

IO Management

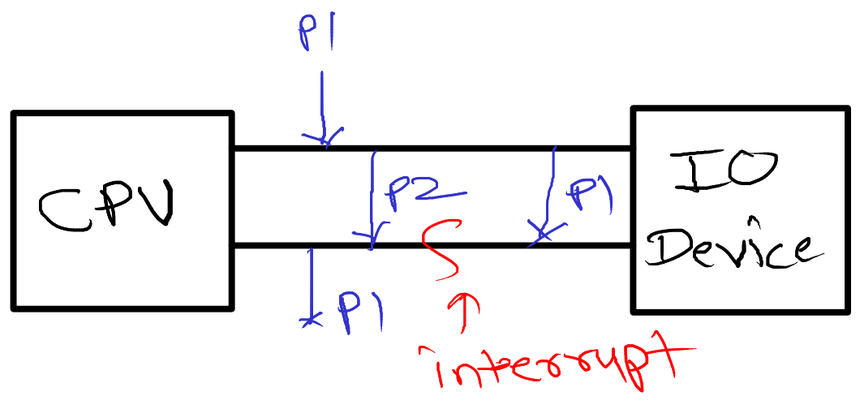
Synchronous IO

Hardware Technique - Polling



Asynchronous IO

Hardware Technique - Interrupt

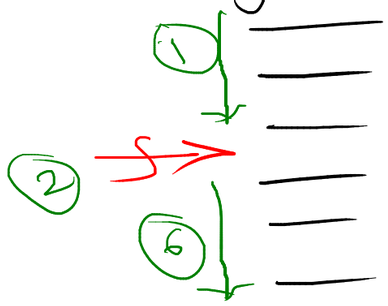


IO Device Table

Device	Status (IDLE/ Busy)	Waiting queue
Printer Mouse Keyboard : (	Busy	* → P1 → P2

Interrupt Service (Delivery)

Program:

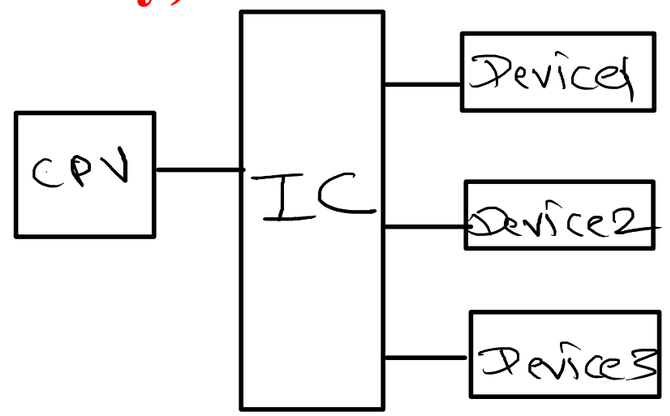


ARM7 – VIC

– NVIC

x86 – 8259

Modern – apic



---

interrupt_handler()
{

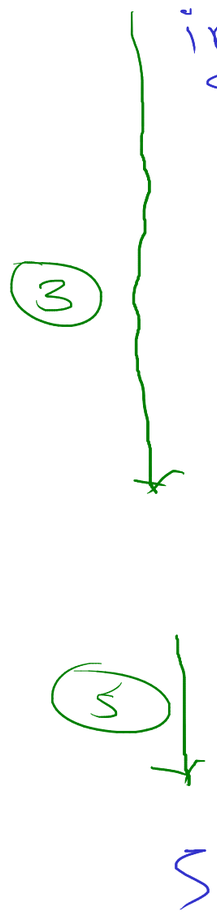
1) Save execution context of current process in its PCB

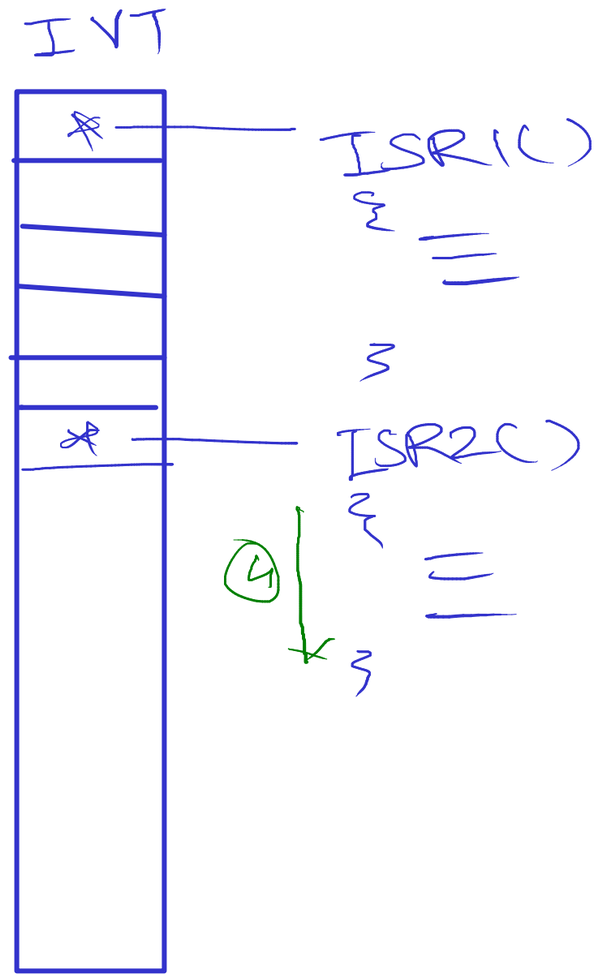
2) Find address of ISR from IVT for interrupt

3) call ISR

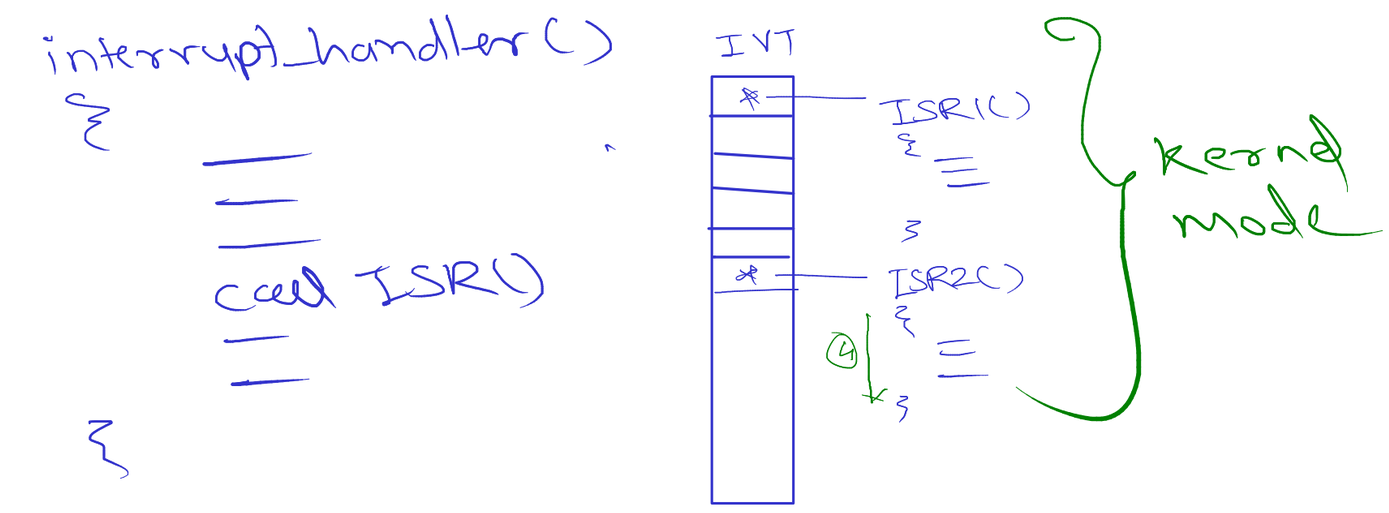
4) Restore execution context of paused process

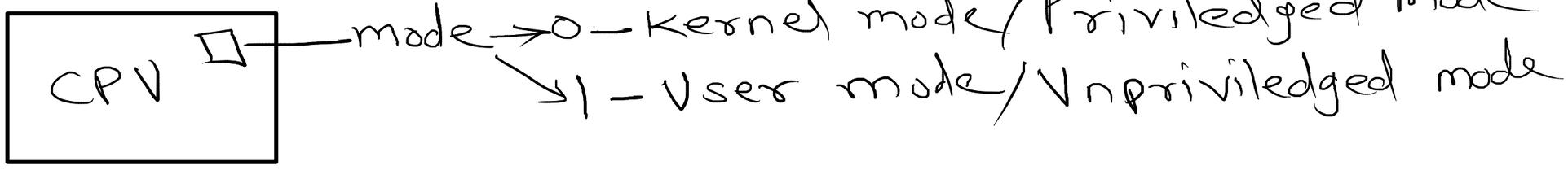
}



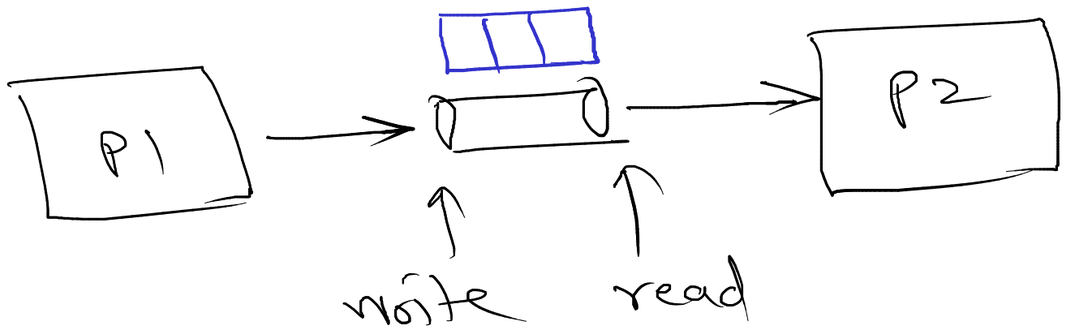


Dual Mode Operation





Pipe



sort

uniq