*Operating System (OS) is an interface between computer user and computer hardware.*

***Why we use OS?***

1. For convinience.
   1. *Windows* is mainly used for this purpose.
2. Throughput
   1. Measure of how many unit of info. a system can process in given time.
   2. *Linux* is mainly used for this purpose.

***Functionalities of OS:-***

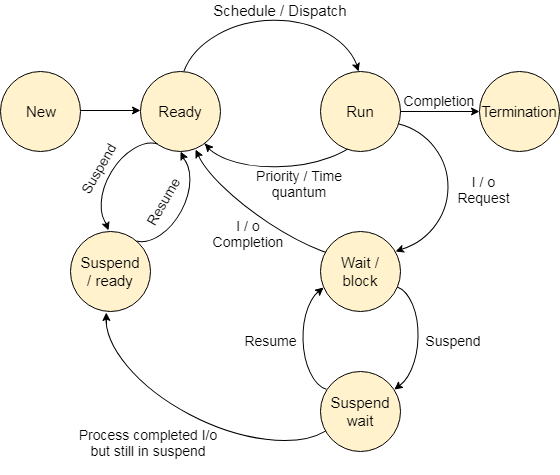
1. Resources management
2. Process management
3. Memory management (main memory ie, RAM)
4. Storage management (sec. memory ie, Hard disk)
5. Security

***Types of OS:-***

1. *Batch OS*
   1. Similer kind or batch of process given to computer.
   2. Processes were given in punch card or paper/magnetic tape to operator/directly to computer
   3. CPUs were not parallel functioning.
   4. CPUs becomes idle if process is given for I/O.
   5. Long time for results.
   6. Languages used in that time ie, 1960 were FORTRAN,etc,
2. *Multi-programmed OS*
   1. *Non-preemptive* ie, CPU will not do another process if the current one is not completed.
   2. If process is given for I/O, CPU will take another process.
   3. Idleness reduced.
3. *Multi-tasking/Time sharing OS*
   1. *Preemptive* ie, CPU will take another process if time for current process is over.
   2. Responsiveness increased.
4. *Real time OS*
   1. Processes have constraint on time.
   2. Two types:
      1. Hard – Constraints on time is too strict. Eg, missile system.
      2. Soft – Constraints on time is little loose. Eg, Youtube streaming.
5. *Distributed OS*
   1. Collection of independent systems, networked.
   2. Connects multiple computer through one communication channel.
6. *Clustured OS*
   1. A system where multiples systems are merged together.
7. *Embedded OS*
   1. A system which is designed to work on specific task. Eg, ATM, etc.

***Process State***

It is a model for users to understand how the process works.



1. **New**
   1. Here, new process is created.
   2. Stored in Secondary Memory (*SM*) as stable state.
2. **Ready**
   1. It’s in RAM.
   2. A queue.
   3. Process comes to ready state from new state with help of *LTS*(Long term scheduler) and *multiprogramming*.
   4. Now, the process is dispatched/scheduled with the help of *STS*(Short term scheduler).
3. **Running**
   1. Process is dispatched/scheduled to running state from Ready state.
   2. A CPU/CPUs.
   3. Process is still in RAM but processed by CPU.
   4. **Conditions**:
      1. Priority

When the high priority process comes the current process is sent to ready state.

* + 1. Time Quantum

Multi-tasking, when the given time for current task is over then the task is sent to ready state and next task comes for being execute.

1. **Termination**
   1. If the process is completed, then it is terminated.
   2. As resources in RAM are limited, the resources given to the process are deallocated after it is completed.
2. **Wait/Block**
   1. A RAM, Queue.
   2. Process comes from running state.
   3. When I/O requests for process comes.
   4. If I/O completed, process goes to ready state.
3. **Suspend Wait**
   1. A Storage memory.
   2. When RAM in wait/block state is being filling.
   3. *MTS*(Medium term scheduling) sends the process to suspent wait state.
   4. *If RAM is being cleared, process is sent back to wait/block state.*
4. **Suspend/Ready**
   1. Process comes here if:
      1. I/O is completed in suspend wait but there is no space in RAM.
      2. High priority process in ready state.
   2. Process then goes to ready state.
   3. *MTS* is working here.

***System Call***

It means conversion from user mode to kernal mode.

1. File related
   1. open(), close(), write(), read(), create file, etc.
2. Device related
   1. Read, write, reposition, ioctl, fcntl.
3. Information
   1. get pid, attributes, get system time and data.
4. Process control
   1. Load, execute, fork, abort, wait, signal, allocate, etc.
5. Communication
   1. pipe(), create/delete connection, shmget().

**fork()**

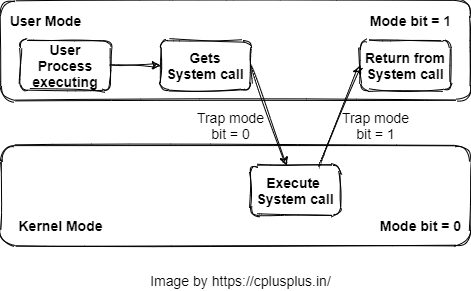
* Function to create child process.
* A system call to create child process.
* +1(positive) for parent and 0 for child.
* No. of forks, 2n and child process, 2n – 1.

P

/ \

* + - * + C1 P
        + / \ / \
        + C2 C1 C3 P

***User Mode vs Kernal Mode***



* Computer is directly in on user mode.
* CPU can switch between user and kernal.
* We can switch to kernal by system call.
* Calls such as read(), starts Trap and we switched to kernal.
* Then we return back to user mode.
* We have to use pc in kernal mode that’s why they’re dual mode.

***Threads***

* A thread is a single sequential flow of control within a program.
* The real excitement surrounding threads is not about a single sequential thread.
* It's about the use of multiple threads running at the same time and performing different tasks in a single program.

|  |  |
| --- | --- |
| **Process** | **Threads** |
| 1. System calls involved. | 1. No system calls. |
| 1. Os treats different processes differently. | 1. All user level threads treated as single task by os. |
| 1. Different process have different copy of codes, data, etc. | 1. Different threads share same copy. |
| 1. Content switching slower. | 1. Content switching faster. |
| 1. Blocking a process won’t block another process. | 1. Blocking a process will block entire process as it uses same data. |
| 1. Independent. | 1. Interdependent. |

|  |  |
| --- | --- |
| **User Level Thread** | **Kernal Level Thread** |
| Managed by user level library. | Manages by OS. |
| Faster. | Slower. |
| Context switching faster. | Slower due coz OS is working itself.  Process>KLT>ULT |
| Blocking a ULT will block entire process as it uses same data. | Blocking a KLT won’t affect others. |

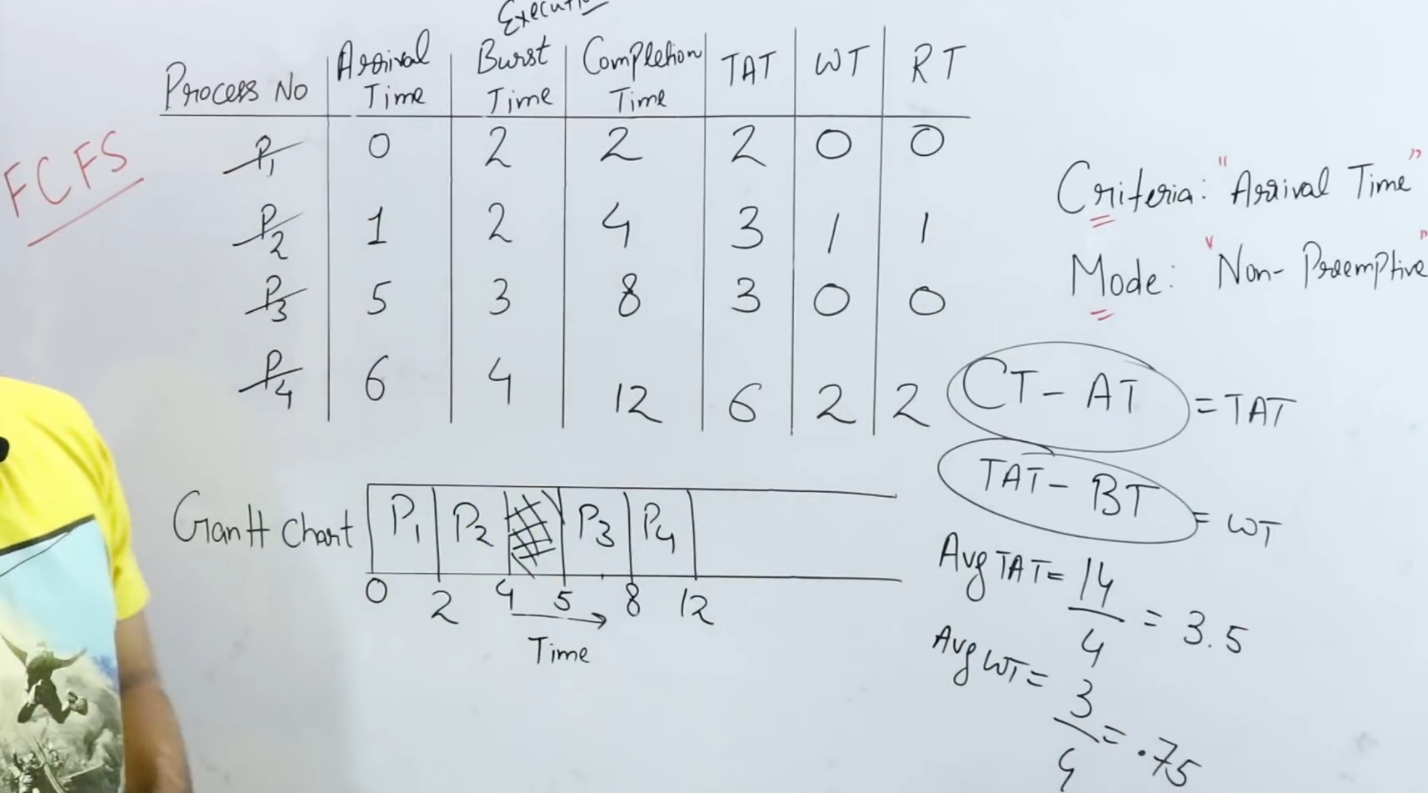
***Scheduling Algorithm***

1. It schedules processes from ready queue(RAM) to running queue(CPU) to do the processing.
2. It has 2 types:
   1. Pre-emptive (*processes can come back to RAM*)
      1. SRTF(Short remaining time first)
      2. LRTF(Long remaining time first)
      3. Round robin
      4. Priority Based
   2. Non pre-emptive
      1. FCFS(First come first serve)
      2. SJF(Shortest job first)
      3. LJF(Longest job first)
      4. HRRN (Highest response ration next)
      5. Multilevel queue

***CPU Scheduling Terminologies:-***

* 1. *Arrival time*
     1. Time in which process enter ready queue/state.
  2. *Burst time*
     1. Time required by process to get executed on CPU.
  3. *Completion time*
     1. Time in which process completes its execution.
  4. *Turn around time*
     1. (Completion time – arrival time)
  5. *Waiting time*
     1. (Turn around time – burst time)
  6. *Response time*
     1. (Time in which process gets to CPU – arrival time)

Numericals:

1. *FCFS*
2. *SJF*

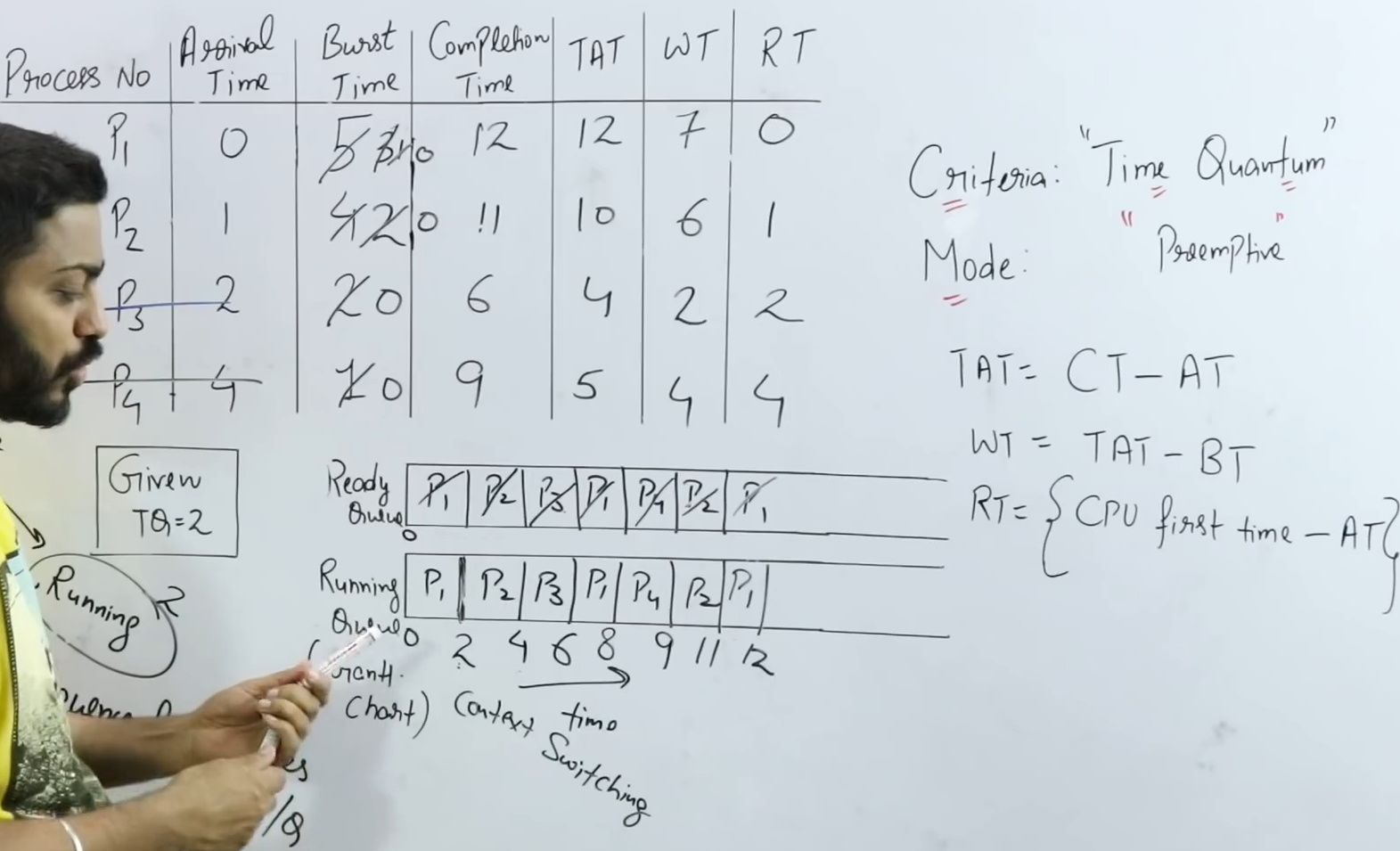
**

* RT = WT

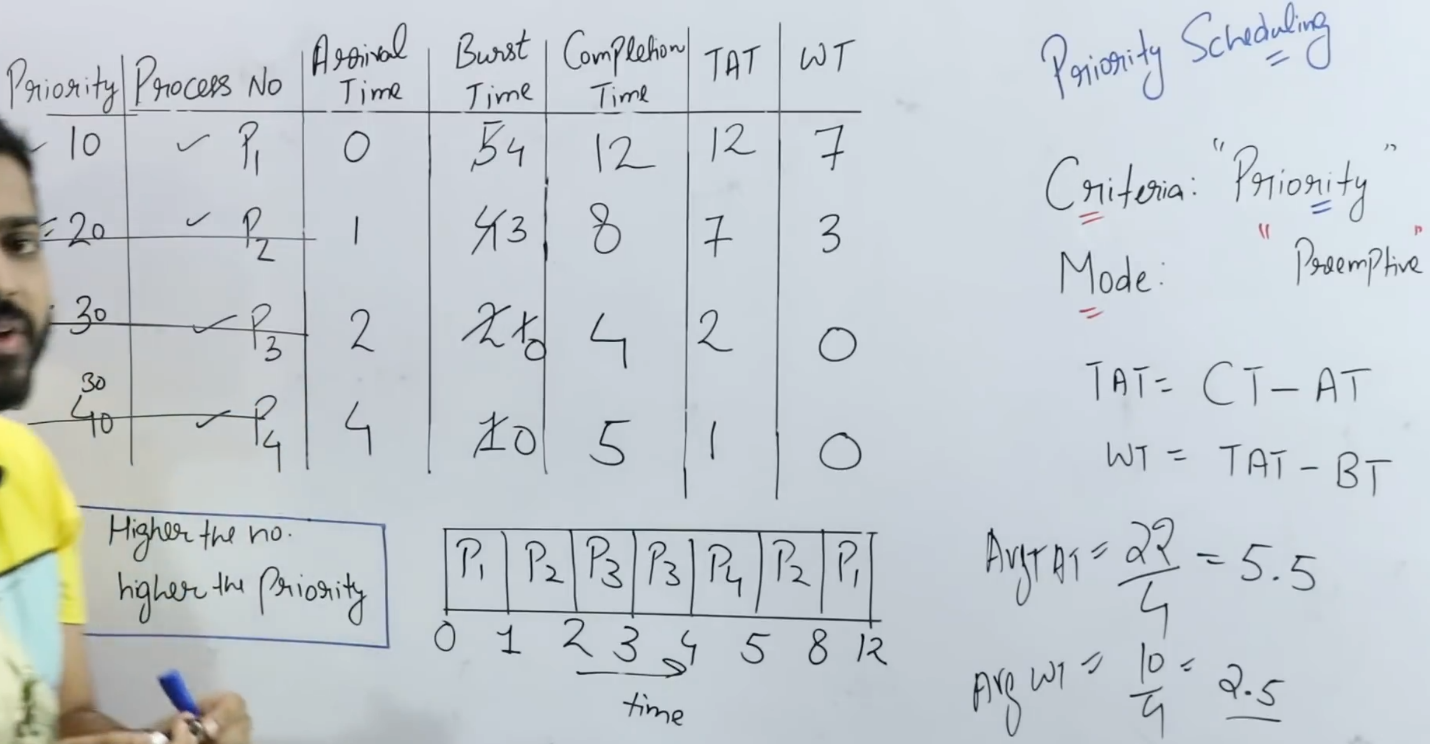
1. *SRTF*



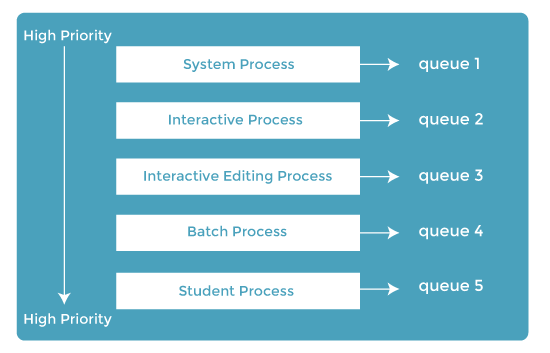
1. *RR(Round robin)*



1. *Priority Scheduling*

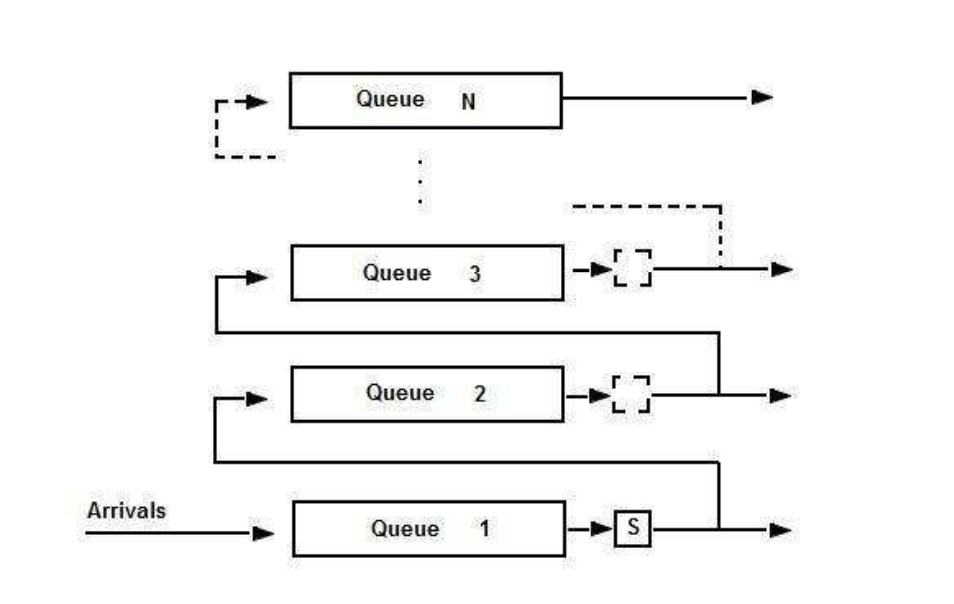
**

***Multilevel Queue Scheduling***

**

* If the processes are different, why use the same queue(ready queue).
* We’ll use different queue for different kind of process.
* It has some anomalies:
  + If high priority p1 is running when will low priority p2 will run, etc…

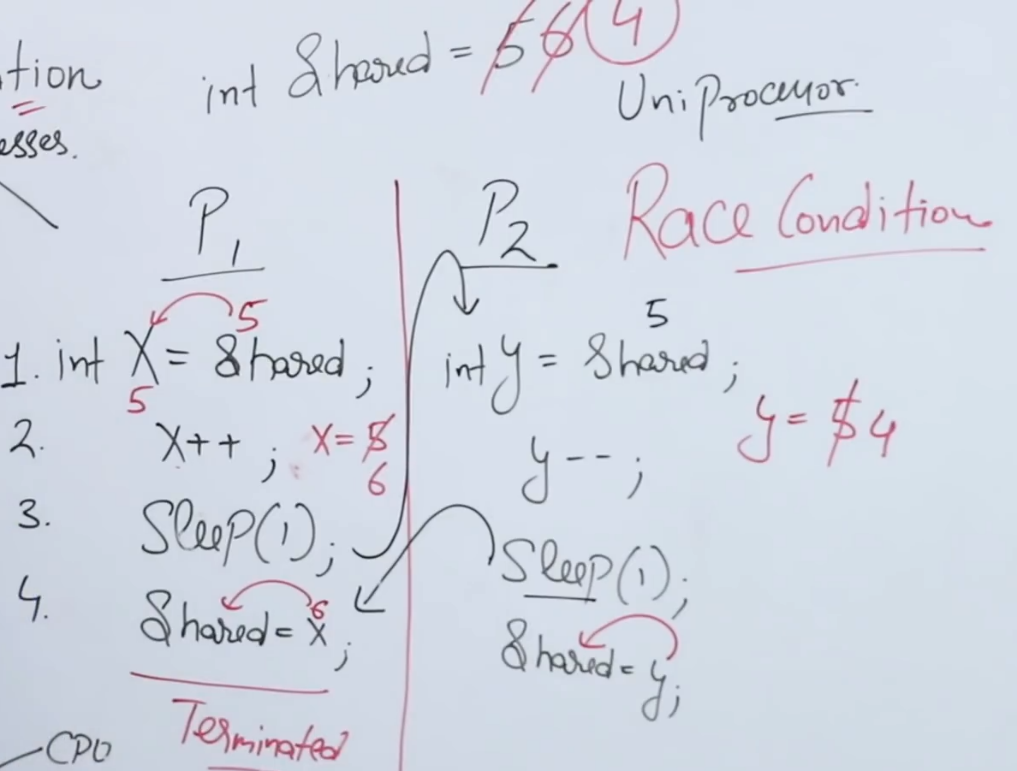
***Multilevel Feedback Queue Scheduling***

******

* Processes can move between the queue.
* Useful for the low priotity queue.
* As low priority process can give feedback by upgrading them to different queue according to their time quantum.
* High priority doesn’t have feedback.

***Process Synchronization***

* It is a process in which processes who share same memory are manages in OS.
* Processes have 2 types:
  + Cooperative process:
    - Share memory, variable, buffer, code, etc.
    - Can have effect on one another.
  + Independent process
* When a process 1 sleeps or stops for some time, process 2 starts running, It’s *context switching.*



* When we run p1, shared space is 5 while 6 in p2.
* Above condition is called *Race condition*.
* To stop this synchronization is important.