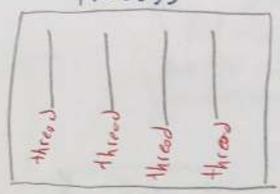
Process Management

Process:

A Process can be thought of as a Program in execution

thread:

A thread is the unit of exection within a Process. A Process can have anywhere from just one thread to many.



Program and Process
↓ class. ↓ object.

Process state:

Process state is defined by the current activity of it.

Process states:

New: the Process is being created.

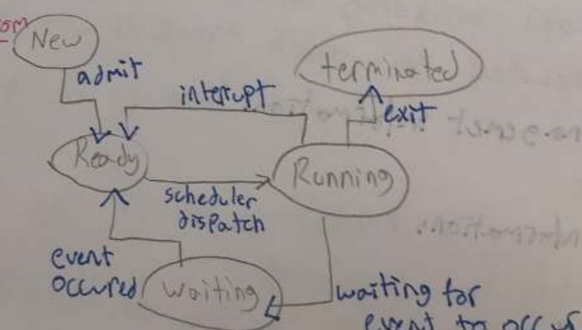
Running: Instructions are being executed

waiting: The Process is waiting for some event to occur. (I/o completion etc.

Ready: The Process is waiting to be assigned to a Processor.

Terminated: Process has finished execution.

Process state diagram



Process control blocks

Proceeses are represented as PCB (process control block) in the operating system.

Process state
Process number (id)
Program counter
Registers
Memory limits
list of open files
....

Process number (id): each Process's unique id. So operating system can identify it.

Process state: represents the process state.

Program counter: indicates the address of the next instruction that has to be executed for that particular Process.

CPU registers: the registers being used by the Process.

CPU scheduling information: it knows the priority of Processes and according to it, it determine the Processes will get executed. how much time it will take.

Memory managemet information: Memory being used by the process.

Accounting information: keeps a count of the used resources in the particular Process.

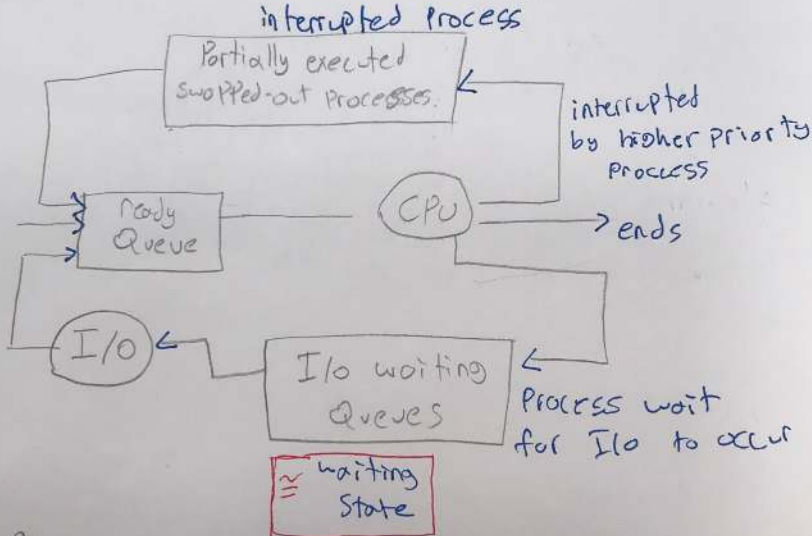
I/O status info: represents the I/O assigned to a Process.

Process Scheduling:

Selects a Process to be executed.

JOB QUEUE: the list of all the Processes in the system

READY QUEUE: the Process that are ready and waiting to be executed.



OS'de file descriptor var 3 modlu

* when process creates new process 2 scenarios are available

1- Parent continue executing concurrently with its children.

2- Parent waits until some or all children have terminated.

children can have some or all Parent resources.

Adres space possibilities

1- duplicate of Parent process (same program and data as Paret)

2- child Process has new Program loaded to it.

Process termination

* A Process terminates when it finishes executing its final statement and asks the operating system to delete it by using exit() system call

* Process may return status value to its Parent via wait() system call.

All resources of terminated Process are deallocated.

termination circumstances

Process can cause termination of another Process via system call.
Parent

InterProcess communication

Processes running could be independent or cooperating

independent: cannot affect or be affected by other Processes.

cooperating: they can affect or be affected.

✱ any Process that shares data is cooperating Process

IPC: (inter process communication)

Processlerin haberleşmesi için 2 yöntem var.

1- **Shared memory**: they use same region of memory. thus read/write will affect Directly.

2- **Message Passing** communication occur in form of messages between cooperating processes.

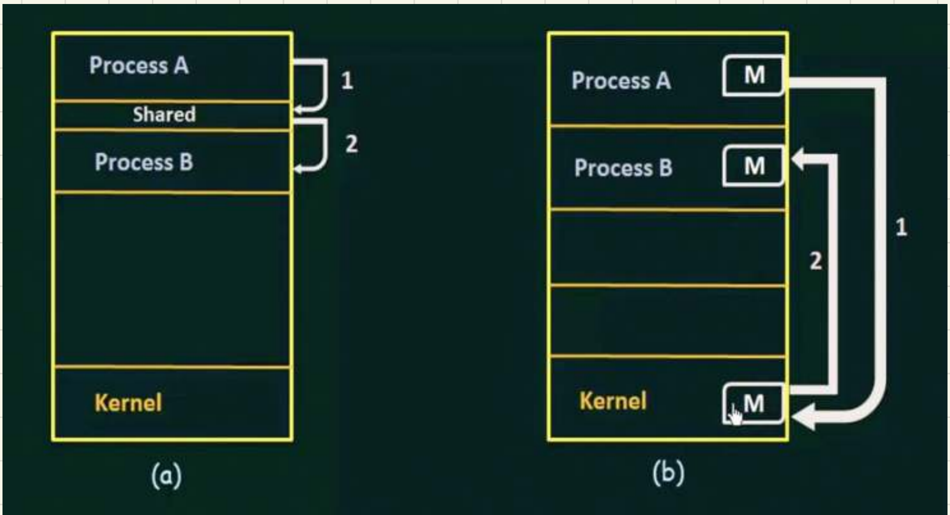


Fig: Communications models, (a) Shared memory, (b) Message Passing.

Shared memory:

- Communicating Processes needs to establish a region of shared memory.
- it resides in the address space of the process that creating shared-memory segment.
- other processes have to attach that region to their own adress space.

Message Passing systems

- allow Processes to communicate without sharing same adres space.
  - * useful in distributed enviroments.

2 functionalties:

1- send message
2- recieve message

* messages could be fixed or variable size.

fixed-size:

System-level implementation is easy. But it makes use of them harder
(Programming with them)

variable size:

system-level implementation harder. make use of them is easier

if processes P and q wants to communicate using messages they
have to establish a communication link.
exchanges messages via send() / recieve() operations

Naming:

Processes that wants to communicate must have a way to refer to each other. they
can use Direct or indirect communications.

Direct:

explicitly name recipient or sender of the communication.

send (P, messag) send a message to Process P.
P → recipient

recieve(Q, message) recieve message from Process Q.
Q → sender.

Properties:

- link established with excatly 2 processes
- each pair of processes have only 1 link.

Indirect Communications

messages are sent to and recieved from mailbox, or Ports

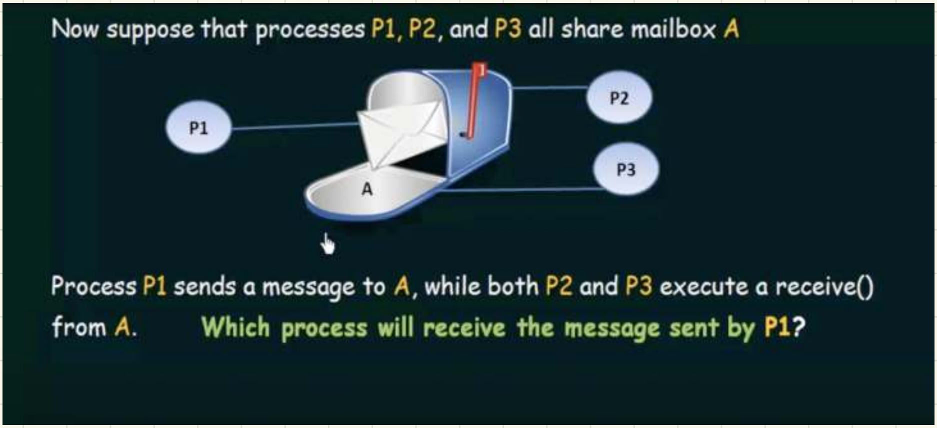
mailbox:

- messages are placed and removed.
- each mailbox has unique ID.
- 2 Process must have shared mailbox to communicate.
  - send (A, message) → send to mailbox A
  - recieve(A, message) → retreve from mailbox A

A → mailbox identifier.

A communication link in this scheme has the following properties:

- A link is established between a pair of processes only if both members of the pair have a shared mailbox.
- A link may be associated with more than two processes.
- Between each pair of communicating processes, there may be a number of different links, with each link corresponding to one mailbox.



Çözüm:

- let 2 processes communicate at most.
- allow one process at a time to execute recieve() operation.
- allow system to choose who will recieve, notify sender who recieved.

Synchronous or asynchronous communication:

- message passing:

  - Blocking (synchronous)
  - non blocking (asynchronous)

Blocking-send

sending process is blocked until message is recieved by recieving process or by mailbox

Non-blocking-send

sending process sends message and resume operation.

Blocking-recieve:

reciver blocks until a message is available

non-blocking recive

recive message or null.

Buffering:

Queue of messages attached to communication link.

3 types of buffer

1- zero capacity . no messages are queued sender must wait for reciever.

2- Bounded capacity: finite length of n. if full sender must wait.

3. Unbounded capacity: infinite length. sender never waits.

Sockets:

- socket is defined as an end point for communication
- A pair of processes communicating over a network employ a pair of sockets
  - one for each Process.
- A socket is defined by an IP adress concatenated with a Port number.

RPC (remote Procedure calls)

- communicating 2 Processes on different systems.
- it similiar to IPC mechanism.
- we need to use message-based scheme
- data have to be well structured as it will travel over the internet
- when client invoke remote Procedure the RPC will call the appropiate stub. and Provide the Params that where Provided to the remote Procedure. and the stub will locate Port on the server and marshall the Parameters.

marshalling Parameters: Packaging the Parameters into a form that can be transmitted over a network.

- the stub then transmit a message to the server using message passing
- similar stub on the server side recieves this message and invokes the procedure on the server.

issues		solution
• differences in data representation on the client and server machine • Big endian - little endian	→	RPC systems define a machine independent represantation of data. One such external data representation. (XDR)
local procedure calls fail only in extreme circumstances. RPC can fail, or be duplicated and executed more than once, as a result of common network errors	→	using Ack make sure that RPC is sent exactly once. client keep sending untill get Ack.
how does a client know the port numbers on the server?	→	Port binding can be done by rendezvous mechanism. OS provides a rendezvous (or matchmaker) service to connect client and servers.

$$\frac{1}{0.4 + \frac{0.6}{2}} = \frac{1}{0.4+0.3} \quad \frac{1}{0.7} \quad \frac{10}{7}$$

CPU Scheduling

its in the basis of multiprogrammed operating systems.

* objective of multiprogramming is to have some process running at all times, to maximize CPU utilization.
* in single processor computers when a processes is waiting for I/O it will hold CPU and will waste time. So; with multiprogramming we try to use wasted time productively:
  * Several processes are kept in memory at one time.
* when a process has to wait, the operating system takes the CPU away from waiting process and gives it to another process.
* **CPU Burst:** Process is being executed in the CPU.
* **I/O Burst:** CPU is waiting for I/O for further execution.

final CPU burst ends with system request to terminate execution.

⋮

load store
add store
read from file } CPU burst

wait for I/O } I/O burst

store increment index
write to file } CPU burst

wait for I/O } I/O burst.

⋮

②

Preemptive and Non-Preemptive schedling

* CPU scheduler: selects the next Process that will get executed by the CPU.
* Dispatcher: gives control of the CPU to the Process that was selected by the CPU scheduler.

Dispatch latency: the time it takes to stop one Process and start another running.

CPU Scheduling's 4 circumstances:

1- when a Process switches from running state to waiting state. (wait for I/O)

2- when a process switches from the running state to the ready state (ex: interrupt occured)

3- when a Process switches from waiting state to the ready State. (completion of I/O)

4- terminates.

- in Situations 1 and 4 the CPU doesn't take decision and it must select other Process, since ether its terminated or in the waiting state. (non Preemptive)
- in 2 and 3 when CPU is back shall it continue executing its last Process? or which descion it must take. (Preemptive)

non-Preemptive: the CPU will never be taken out from Process until it finish execution or it goes to waiting State.

Preemptive: CPU may be taken away while its executing.

Scheduling Criteria

1- CPU utilization
2- Throughput
3- turnaround time
4- waiting time
5- response time

1- CPU utilization:

we want to keep the CPU as busy as possible. theortically from 0 to 100 percent. in practice 40% - 90%

2- throughput:

the measure of processes gets completed per time unit

3- turnaround time:

the time from submission of the process to its complition.

4- waiting time:

waiting time is the sum of the periods spent waiting in ready queue. CPU scheduling algorithm affects waiting time of a process.

5- response time

In an interactive system. turnaround time may not fit well. so response time is from the time submission is requested till it get first response. the turnaround time is generally limited by the speed of the output device.

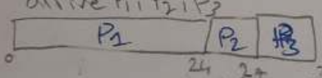
Scheduling algorithms

FCFS (first come, first serve):

- who ever come first takes CPU first. uses FIFO Queue.
- when a process wants to execute it enters the Queues and when it start running it gets out.

Process	Burst time
P1	24
P2	3
P3	3

Arrive time = 0
arrive P_1, P_2, P_3



waiting time

$P_1 = 0$
$P_2 = 24$
$P_3 = 27$

$avg = \frac{(0+24+27)}{3} = 17$

if Processes arives P_2, P_3, P_1:

P_2	P_3	P_1
0 – 3	3 – 6	6 – 30

waiting time

$P_2 = 0$

$P_3 = 3$

$P_1 = 6$

$avg \left(\frac{0+3+6}{3}\right) = 3$

* The FCFS scheduling algorithm is **non Preemptive**.

convoy effect: if Process with higher burst times arrives before the Process with smaller burst time.

ex)

Process ID	Arrival time	Burst time
P_1	4	5
P_2	6	4
P_3	0	3
P_4	6	2
P_5	5	4

given table calculate average waiting time and average turnaround time. using $\boxed{FCFS}$.

P_3	idle	P_1	P_5	P_2	P_4
0 – 3	3 – 4	4 – 9	9 – 13	13 – 17	17 – 19

2

Turn around = completion time - arrival time

waiting time = turnaround time - Burst time

Process ID	completion time	turnaround	waiting time
P_1	9	9-4=5	5-5=0
P_2	17	17-6=11	11-4=7
P_3	3	3-0=3	3-3=0
P_4	19	19-6=13.	13-2=11
P_5	13	13-5=8	8-4=4

SJF (shortest-job-first scheduling)

• when CPU is available it is assigned to the Process that has the smallest next CPU burst. if 2 Processes have same length FCFS is used.

SJF can be preemptive or non preemptive.

ex)

Process ID	Burst time
P1	6
P2	8
P3	7
P4	3

draw gantt chart using SJF (non-preemptive).

P4	P1	P3	P2
0 – 3	3 – 9	9 – 16	16 – 24

waiting time

P4 ~ 0

P1 = 3

P3 ~ 9

P4 ~ 16

$\boxed{avg = 7}$

ex)

Process ID	Arrival time	Burst time
P1	0	8
P2	1	4
P3	2	9
P4	3	5

draw gantt and calculate average time using SJF (Preemptive).

P1	P2	P2	P4	P1	P3
0 – 1	1 – 2	2 – 5	5 – 10	10 – 17	17 – 26

7

turn = 17 - 0

17 - 1 - 7

P1 : 7

P3 : 9

P4 : 5

waiting time => total waiting time - total executed - Arrival time.

$P_1 = 10 - 1 - 0 = 9$

$P_2 = 1 - 0 - 1 = 0$

$P_3 = 17 - 0 - 2 = 15$

$P_4 = 5 - 3 = 2$

AVG = 6.5

* Preemptive SJF also called shortest remaining time first scheduling

* it is impossible to know the next CPU request of a Process.

Q/ using Preemptive SJF calculate average waiting time

Process ID	Arrival time	Burst time
P1	0	12
P2	2	4
P3	3	6
P4	8	5

calculate average time?

P_1 : 10
P_2 : ~~4~~ ~~2~~ 0
P_3 : ~~6~~ 4
P_4 : 5

P1	P2	P3	P4	P1
0 – 2	2 – 6	6 – 12	12 – 17	17 – 27

waiting time: total - burst - arr

$P_1 = 17 - 2 - 0 = 15$
$P_2 = 2 - 0 - 2 = 0$
$P_3 = 6 - 0 - 3 = 3$
$P_4 = 12 - 0 - 8 = 4$

$22/4 = 5.5$

~~Scheduling~~

Priority scheduling

- each Process has a Priority. the Process with the highest Priority gets executed. if equal use FCFS.
- Priority Scheduling can be Preemptive or non-Preemptive.

Preemptive Priority scheduling:

the algorithm will Preempt the CPU if the Priority of the newly arrived Process is higher than the running one.

Non Preemptive:

new Process will have to wait the Process to finish.

Problem: Starvation may occur. (Small Priority Processes may never execute.

Solution: Aging: as time Progresses increase the Priority of Process.

ex) consider Processes arrives at same time=0. small value of Priority means the higher. use priority scheduling. ②

Process ID	Burst time	Priority
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2

waiting time

$P_1 = 6$
$P_2 = 0$
$P_3 = 16$
$P_4 = 18$
$P_5 = 1$

Avg $= \frac{41}{5} = 8.2$

| P2 | P5 | P1 | P3 | P4 |
0 1 6 16 18 19

ex)

0 is highest priority. Calculate average time using Preemptive Priority scheduling algorithm.

Process ID	Arrival time	Burst time	Priority
P1	0	11	2
P2	5	28	0
P3	12	2	3
P4	2	10	1
P5	9	16	4

~~2P1: 11 9~~
~~1P4: 10 7~~
~~0P2: 28~~
4P5:
~~3P3~~

| P1 | P4 | P2 | P4 | P1 | P3 | P5 |
0 2 5 33 40 49 51 67

waiting time = total - arrive - burst

$P_1 = 40 - 2 - 0 = 38$
$P_2 = 5 - 0 - 5 = 0$
$P_3 = 49 - 0 - 12 = 37$
$P_4 = 33 - 3 - 2 = 28$
$P_5 = 51 - 0 - 9 = 42$
$60 + 75 = \frac{145}{5} = 29$

Round - Robin algorithm

* round-robin Scheduling algorithm is designed espically for timesharing systems.
* each Process gets a small unit of CPU time (time quantum). then the Process is added to the end of the ready queue.
* ready Queue is treated as a circular Queue.

P.S.: if we choose a large time quantum RR will be FCFS.

ex)
arrive time 0, time quantum = 4. Calculate the following:

Process ID	Burst time
P1	24
P2	3
P3	3

Gantt chart:

P1	P2	P3	P1	P1	P1	P1	P1
0	4	7	10	14	18	22	26

30

turnaround: Completion - arrival

P1: ~~24 26~~ ~~14~~ ~~18~~ ~~30~~

waiting: turn around time - Burst time

Process ID	Completion time	turnaround time	waiting time
P1	30	30-0 = 30	30-24 = 6
P2	7	7	7-3 = 4
P3	10	10	10-3 = 7

Avg wait = $\frac{17}{3}$ = 5.6

method 2:

a) RR time quant = 2, calculate average turnaround and wait time.

Process ID	Arrive time	Burst time
P1	0	5
P2	1	3
P3	2	1
P4	3	2
P5	4	3

Queue

~~P5/1~~, ~~P1/1~~, ~~P2/1~~, ~~P5/3~~, ~~P4/2~~, ~~P1/3~~, ~~P3/1~~, ~~P2/3~~, ~~P1/5~~

Queue ←

P1	P2	P3	P1	P4	P5	P2	P1	P5
0	2	4	5	7	9	11	12	13

14

Process	completion	turnaround	waiting
1	13	13	8
2	12	11	8
3	5	3	2
4	9	6	4
5	14	10	

Multilevel Queue.

* Ready Queue is Partitioned into seperate Queues.
  - foreground (interactive)
  - background (Batch)

They have
- Different response time.
- Different sched-ling needs

* Process assigned Permanetly to a queue.
* foreground Processes may have higher Priority than background.

important: each Queue have different algorithm

- foreground - RR
- Background - FCFS

* scheduling of Queues. Fixed Priority sched-ling. (serve all from foreground then from background.). Possibilty of starvation

* time-slice: each Queue gets a certain amount of CPU time. which it can schedule among its Processes.

example of priority Process Queue: