OS ⇒

↳ most important prog. that runs the machine.

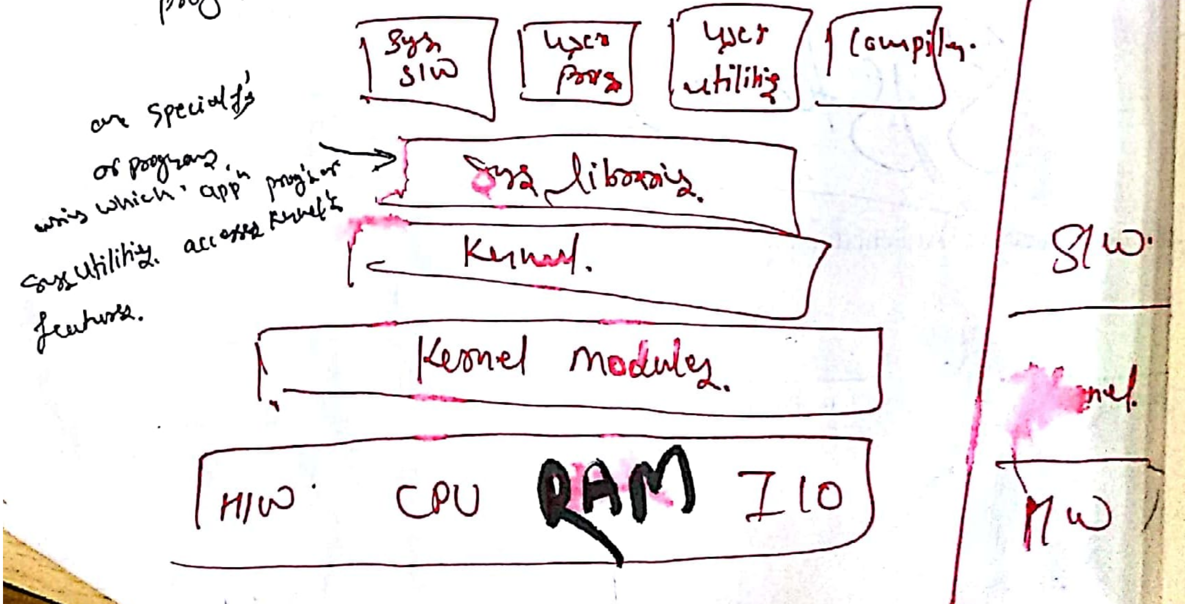
↳ Perform basic tasks., Such as

↳ i/p recognition (keyboard)
↳ sending o/p (Display)
↳ Keep track of files & directories indi...
↳ Control peripherals.
↳ like traffic COP.
↳ responsible for security.

Classification

↳ multiuser.
↳ Multiprocessing :- more CPU's
↳ Multitasking :- more than 1 prog.
↳ Multithreading :- parts of single prog. run concurrently.
↳ Real time.
⇒ instant response.

Utilities :- prog that provides the user most of the functionalities of an O.S.



Off-the-Shelf Operating Systems.

↳ Non-realtime. embedded OS :- use preemptive kernel but Strict deadline can't be met.

↳ Real time. Operating System :- Provides necessary f"ility for achieving σ-T Performance. through very low interrupt latency.

↳ Handheld / mobile. operating System :- Palm OS, Symbian OS windows CE

Commonalites of OS

↳ IDE :- To facilitate easy & fast development includes editor, compiler, debugger and also the necessary cross platform development tools.

↳ POSIX Compatibility :- POSIX 1003.1-2001 Standard Specifies the API. to achive portability of app".

↳ TCP/IP Support ⇒ including app" layer protocols such as. FTP, SMTP (Simple mail), HTTP (Hyper text transfer) is integrated along with O.S sw. f" call will be provided to access the n/w - related Services.

↳ Device Driver ⇒ for Commonly used devices such as Serial Port., Parallel Port, USB etc. A DDK (will) is also included for development of device drivers.

Portable Operating System Interface (POSIX)

↳ developed by IEEE.

(portability among)

before POSIX, every vendor has proprietary API for app dev.

↳ helps to access OS Objects & Services.

↳ IEEE POSIX 1003.13 Stand. App" Environment Profile
or POSIX R-T App" Support. API for EI.

↳ The Concepts of thread become popular. Cuz of this Standard.

↳ This Standard gives various C lang. f^n calls & libraries that need to be implemented by OS.

↳ POSIX for Small System POSIX Profile

↳ POSIX Standard for large multi-processor based sys

Differences in OS ⇒

↳ COTS OS differs in the following aspects:

↳ Support for processors ⇒ OS code has:-

Why every OS can't support all processor.

(a) processor-independent code. e.g context Switching i.e content of CPU has to store in memory. need A.S.L.

(b.) processor-dependent code

Commercial OS Supports X86/64, MIPS, PowerPC, Sparc, ARM

(b.) Footprint ⇒ memory occupied by. Kernel differs from OS to OS.

→ The RTOS Kernel is in KB's. OS. Vendor Specifies minimy Hardware requirement. & OS.

(c.) Java Enviournmt :- Some vendors provide a Java V. M (J.V.M) Supports on theirs

(d.) Board Support Packages :- Vendors Supply. h/w boards. built around different processor with as pretedonts the h/w BSP Speed up S/w development.

(e.) Scheduling Algorithms :- OS Support's roundrobin. FIFO etc.. where as OS supports Priority based preemtive Scheduling.

(f.) Priority Inheritance :- Few like few dislikes fell that designers should be given flexibility in app dev.

(g.) Maximum no. of tasks :- differs from OS to OS. Some Supports only 64 task, Some claim "unlimited" e.g if 32-bit integer is used for taskID, the no. of tasks is limited to 2^{32}.

(h.) Assigning task priorities :- In some OS each task has a unique priority & Priority is fixed.

In Others Priority can be changed dynamically during executing time.

7. POSIX Support :- few vendors claim POSIX Support. "small System Portable" doesn't mean full blown POSIX Support.

8. Licensing terms - prices of OS differ
   - Under free G.P.L.
   - On line Payment.
   - Pay for Service.

Embedded O.S

- Embedded NT → for SBC
  use in ATM, Ticket booking bill payment Service etc.

NT = new tech.

Req for Embedded NT

- OS without any N/w Support is
  9 MB of RAM & 8MB of Prog. (Cylinder)

if appn is a single Win32 appn with n/w Capability.
min RAM req is 16 MB RAM & 16 MB of Prog memory.

App development using Embedded NT is easy.

IDE → Visual Studio env. (VC++ or VBasic)

IIS 3.0 (Internet Info Service (web server)) is included in Embedded NT.

dependencies

- Windows NT 4.0 Service Resource kit.
- Windows NT 4.0
  - Service Pack 4.0 or later
  - IE
  - Visual Studio

Create headless System.

↳ Null VGA
↳ Null Keyboard Drive.
↳ Null Mouse.

↳ Debugging a headless System. is a problem coz lacky i/p devices.
↳ Use. Console administration Componed. using Serial interface.

↳ Create. a target sys that boots from. Drive (CDRom)

↳ Create. a bootable Partition. on the hard disk.

→ Window XP Embedded (edition):

Successor to Embedded NT

↳ Preemptive multitasking OS. & uses window32 app's & drive.

app" :- STB, IrDA Compliances., Net meeting.

9 RAM + 8 Rom. ⇒ dev. env. Visual Std

Embedded Linux ⇒ GNU GPL. license.

→ available freely.
→ LynuxWorks., Monta Vista., Redhat, Timesys etc.

→ Open Source.
→ Source code for - dd., n/w slw, Protocol slw
speech/image slw
→ Support Posix
→ No royalty.
→ Availability of Many People with expertise in Unix/Linux prog.

Embedded Linux Consortium. Platform Specification

⇒ ELC defines three environments. Types are,

↳ Minimal Sys environment.

↳ applicable for ES with 1 cpu & associated memory.

↳ only 1 process with 1 or more Linux task is working.

↳ OS foot print will be small.

Intermediate-Sys environment.

↳ 1 or more cpu

↳ file can be built in flash memory device.

↳ Support multiple process, asynchronous I/O operation & support dynamic (linking COW) libraries.

↳ Support for Sec object is also provided.

Full System Environment.

↳ 1 or more cpu, Sec. storage n/w support & user interface.

↳ Supports full multipurpose linux.

↳ Support for user interface & n/w support.

RTOS → ARM, MIPS, PowerPC
→ multiple scheduling Algo
→ upto 10 65K tasks are supported.
and each task can have 65K threads

↳ qNX Neutrino → min memory req: 64K Kern. ROM
& 32K Kernel RAM

↳ Vx Works → Support preemptive & R-R scheduling
25% priority dec. Algo
Mars pathfinder.

↳ Micro C/OS-II ⇒ Supports 64 tasks out of
with 8 are sys task.
unique priority for each task.
R-R not supported.

↳ RT Linux ⇒ 4 MB min. memo req^n.

TASK and Task Scheduler ⇒

→ The program in execution. is known as - Process.

→ A Prog can have number of processes.

→ Every process has its own address space.

→ Thread use address space of process.

diff b/w process and thread :-

when CPU switches from a process to another the current info. needs to be saved in Process Descriptor and load the information of a new process.

Switch to 1 thread to another is simple.

→ A task is simply a set of instructions loaded into the memory.

Threads can themselves split into two or more simultaneously tasks.

Process :- a instance of Comp. Prog. that is being executed. It Contains the program code and its current activity. ("Data Organization Concept")

Thread ⇒ A thread of execution results from a fork of a Comp. Prog. into two or more. Concurrently running task. ("Scheduling Concept")

A Thread is Contained inside a process

e.g. Auto Spell check & auto save of file while writing

Task ⇒ Set of Prog instructions that are ~~loaded~~ loaded in memory.

↳ embedded S/W Consist of no. of tasks which include
↳ OS tasks ↳ App^n-specific tasks

↳ each task is implemented as infinite loop.

task objects Consist of.

↳ a. its name ↳ c. a priority
↳ b. a unique ID ↳ d. a stack
& ↳ e. Task Control Block (TCB)
↳ Contains all info related to task

Kernel task :-

① Startup task, which exe. when OS Start.
② Exception handling task
③ logging task → message log.
④ Idle task, → lowest priority to ensure that CPU is not idle.

↳ In Single CPU :- task share to share CPU ~~in Time Div.~~ in time in Disciplined manner.

Therefore each task has to be assigned a priority and a mechanism for deciding which task will get CPU time next, has to be worked with.

This is known as task scheduling.

The object that does task Scheduling is known as Task Scheduler

✓ task has to share resources such as :-
- ↳ CPU ~~reg~~ reg. ↳ external memory.
- ↳ I/O devices.

✗ task should not corrupt the data of another task.

↳ issues to keep in mind while Scheduling

(a.) Many f^n tasks, may make call to f^n's.

reentrant f^n ⇒ f^n used by 1 or more than task without data corruption.

non-reentrant f^n ⇒ data is corrupted when more than 1 task calls the f^n. e.g global variables.

```
int x;
void fun2 (int x)
{
return (x+x);
}
```

→ This f^n is non-reentrant as it is global var to make it interrupt use local var.

Critical Section of code:- portion of code should not be interrupted while running this.

↳ Interrupt's are disable

Shared Resources :- Every task or resources such as Serial Port, keyboard, display or memory loc?

if two or many task shared same resource, They should maintain discipline.

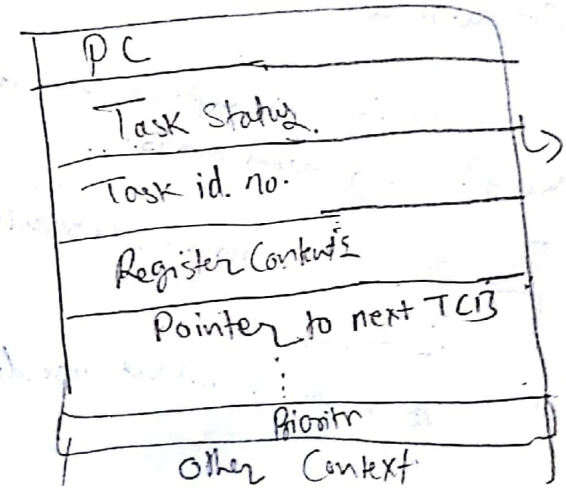
eg semaphore & mutexes.

⇒ Task need to Communicate data amongst themselves.

e.g. A task wires data into an array & another task read data from that array.

Intertask Communication needs to be achived Through special mechanism. Such as mailboxes, message queues, pipes, event registers and Signals.

Task Control Block:



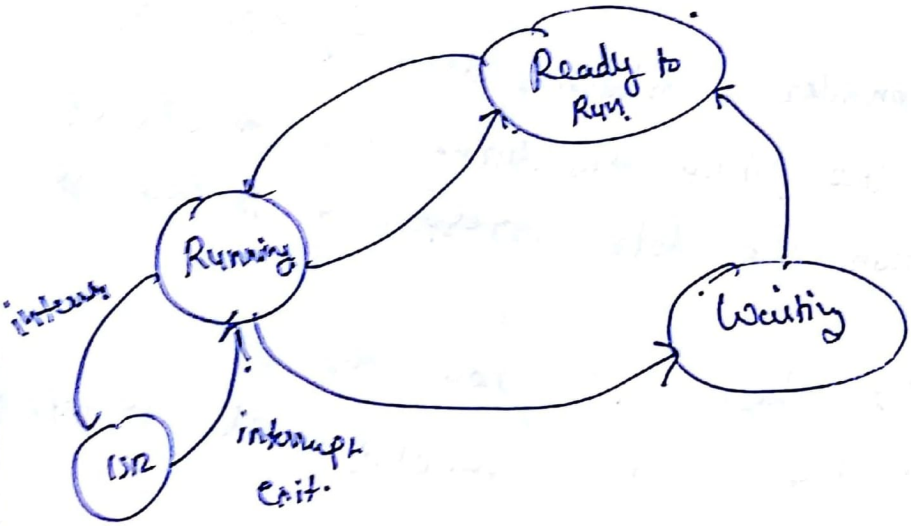
↳ a context (e.g. PC & register Context)

↳ an identification String

↳ a status, such as ready ↳ executing or Blocked

↳ a priority (if applicable)

Task States



Running State ⇒ if it is being executed by the CPU

Waiting State ⇒ waiting for another event to occur.

e.g. Slow data transfer. (get data from serial port)

Ready to Run ⇒ if it is waiting in a queue for CPU time

Task Stack ⇒ every task has a stack that stores local variables, fn parameters, return addrs and CPU Registers during an interrupt.

Context Switching ⇒

↳ Consider a situation when the CPU is executing a low priority task, during which time the CPU register contain the data corresponding to this task.

↳ The CPU now has to take out the L.P.T in other words (preempt) and execute the H.P.T

↳ The state of the CPU registers when a task is to be pre-empted is called Context.

↳ Saving the contents of CPU reg & then loading the new task is called "Context Switching"

Scheduling Algorithms ⇒

which task to run? use Scheduling

↳ FIFO.

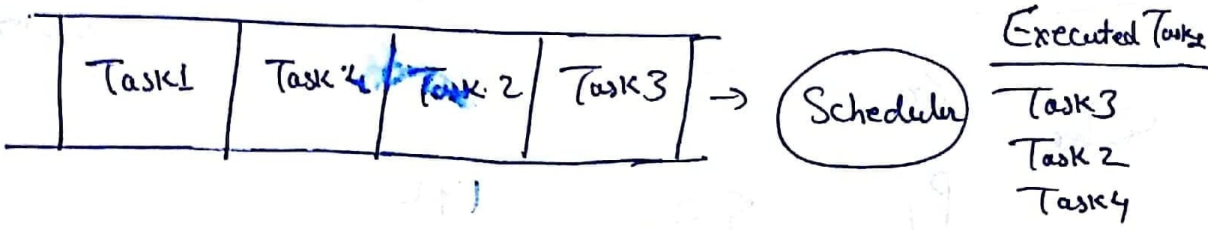
↳ Round-robin

↳ Round-Robin with priority.

↳ Shortest Job first

↳ Non-preemptive multitasking.

↳ Preemptive multitasking.

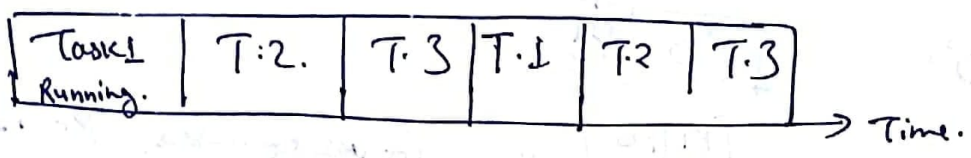


↳ not well suited for most appⁿ bcoz it is difficult to estimate the amount of time a task has to wait for being executed.

↳ not time criticality.

↳ small no. of tasks.

Round-Robin Algo



→ Certain time for each task waiting in queue.

→ Time slice allocated to each task is called quantum.
Control to next task if.

(a) The current task is done within the time slice.

(b) Current task (T_i) has no work to do.

(c) Current task has Completed its allocated Time slice.

⇒ Note No priorities in among the task.

→ Not good for time critical operations.

e.g digital multimeter, dMware oven.

	Arrival time	Execution Time
P_0	0	250
P_1	50	170
P_2	130	75
P_3	190	100

0 [P0]
50 [P0 | P1]
130

Round Robin with Priority.

quantum time is 100 ms.

P0	0	250
P1	50	170
P2	130	75
P3	190	100
P4	210	130
P5	350	50

Execute. Time

0	P0	
50	P0 | P1	
100	P1 | P0	→ Quantum time. 100 ms expires. P0 is forced out
130	P1 | P0 | P2	P2 arrives.
190	P1 | P0 | P2 | P3	→ P3 arrives
200	P0 | P2 | P3 | P1 (↳ preempted)	Next quantum. P0 get processed
210	P0 | P2 | P3 | P1 | P4	→ P4 arrives
300	P2 (executed) | P3 | P1 | P4 | P0	→ Next quantum
350	P2 | P3 | P1 | P4 | P5	
375	P3 | P1 | P4 | P0 | P5	→ P2 done.

475 ⇒ P1 | P4 P0 P5 → P3 done

575 ⇒ P4 | P0 | P5

675 ⇒ P0 | P5 | P4 | → Next quantum. P4 forced out

695 ⇒ P5 | P4. P0

745 ⇒ P4.

775 ⇒ P4 get completed.

Throughput ⇒ No. of. process that are completed per time unit

Turnaround time ⇒ time b/w Submission & Completion.

Waiting time ⇒ Scheduling affects only waiting time.

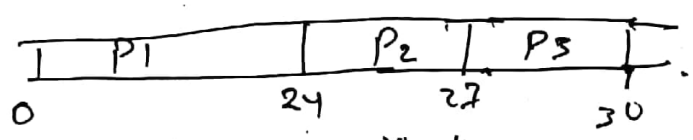
Response time ⇒ Time b/w Submission & first response

burst time ⇒ amount of time process uses the processor or time req to complete execution of process or task

---

FIFO

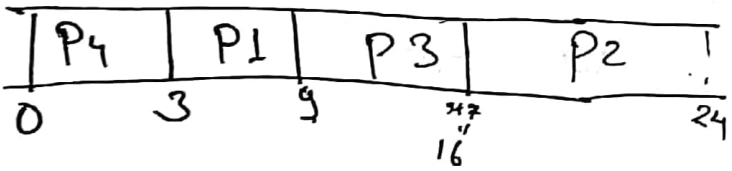
Process	Burst time	Arrival
P1	24	0
P2	3	0
P3	3	0



Average waiting time: (0+24+27)/3 = 17

Shoot Job first.

Process	Burst time	Arrival
P_1	6	
P_2	8	
P_3	7	
P_4	3	



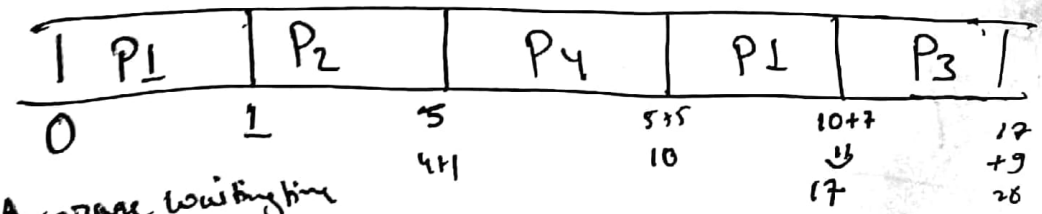
Average waiting time. $(0+3+9+16)/4 = 7$

with FCFS $(0+6+(6+8)+(6+8+7))/4 = 10.25$

SJF with Preemption

(Shortest Remaining time first)

Process	Burst time	Arrival
P_1	8	0
P_2	4	1
P_3	9	2
P_4	5	3



Average waiting time

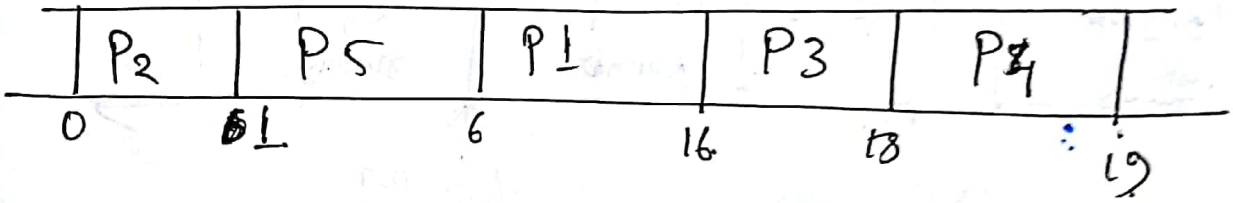
$\Rightarrow (10-1) + (1-1) + (17-2) + (5-3)/4 = 6.5$

with SSF $\Rightarrow (0+4+(4+5)+(4+5+8))/4 = 7.75$

Priority. Scheduling Algo

Process	Burst time	Arrival	Priority
P1	10	0	3
P2	1	0	1
P3	2	0	4
P4	1	6	5
P5	5	0	2

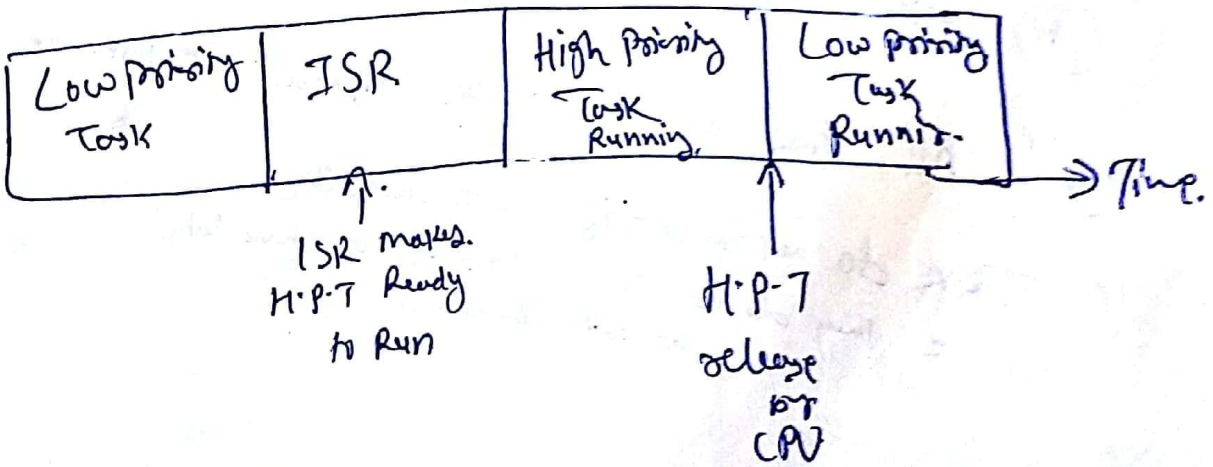
Gantt chart



Average waiting time: $(0+1+6+16+18)/5 = 8.2$

NOW - Preemptive Multitasking (Co-operative multitasking)

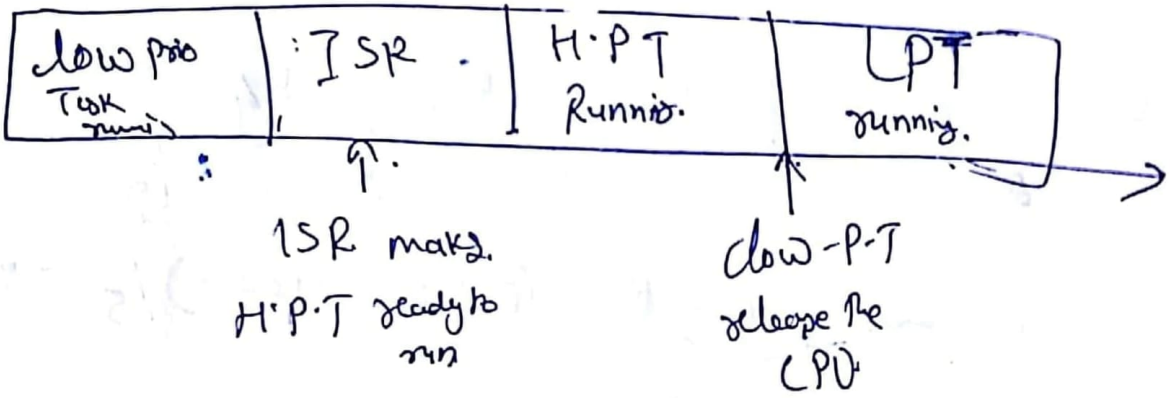
⇒ The task cooperate with each other to get their share of CPU time. Hence, each task chose to release the CPU & give control to another task. on its own.



Preemptive Multitasking

⇒ if a lower priority task is presently running and higher priority task is Ready to Run, the running task is preempted and higher priority task is executed.

↳ All RTOS have preemptive multitasking



RATE Monotonic Analysis

Priority — Static
↘ dynamic

RMA assumptions :-

(a) Highest priority task run first. i.e. priority based preemptive multitasking.

(b) All task run at regular interval i.e task are periodic

(c) Task do not synchronize with each other (i.e they do not share resources or share data) not valid assumption.

In RMA priority is proportional to freq of execution

if i^{th} task has an execution period of T_i & C_i is its execution time, C_i/T_i gives % age of CPU time required for i^{th} task.

$$\sum (C_i/T_i) \leq U(n) = n(2^{1/n} - 1)$$

$U(n)$ ↓ utilization factor. n → no. of task.

n	U(n)
1	1.
2	0.8
3	0.72
:	:
infinity.	0.65

if no. of tasks is ∞ then about 70% of CPU time is utilised.

Task management f^n calls.

- Create task.
- Delete task.
- Suspend task.
- Resume task

- Change priority
- Query task

TCB / PCB

next process in que → Pointer | Process state

- Process no.
- Prog. Counter
- registers
- memo. limits

list of open files.

- CPU scheduling info
- Accounting info
- I/O Status info

Semaphores ⇒ H/W or S/W flag. Variable whose value indicates the status of a common resource.

It's purpose is to lock the resource being used.

Binary Semaphores.

↳ 2 methods (up, down/lock/unlock)

↳ Binary semaphore take only 2 values (0/1).

They are used to acquire locks.

when resource is available process in charge set semaphore 1

Counting Semaphores

↳ use to allocate resources from a pool of identical resources.

Edsger Wybe Dijkstra

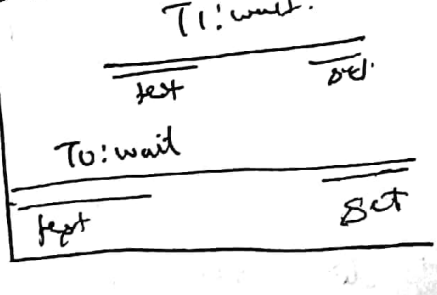
→ Semaphore is a Boolean variable or integer → that can accessed only through two atomic operations.

(1) wait – P(S) P → proberen → to test (Dutch)

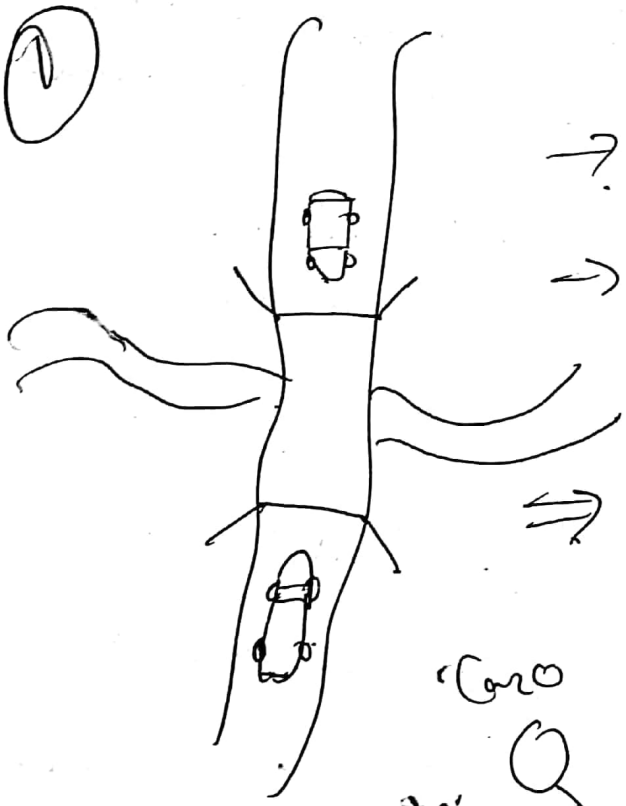
(2) Signal – V(S) Verhogen → V → to increment.

wait → test the value of semaphore, if it is true set it to false. work with test set

Signal → set the value to false.

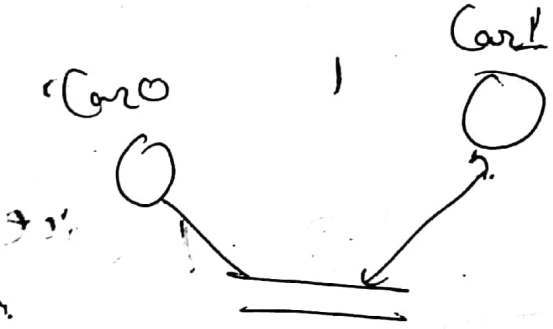


(1)



→ Bridge ⇒ Shared resource.

→ 1 car (process) at a time



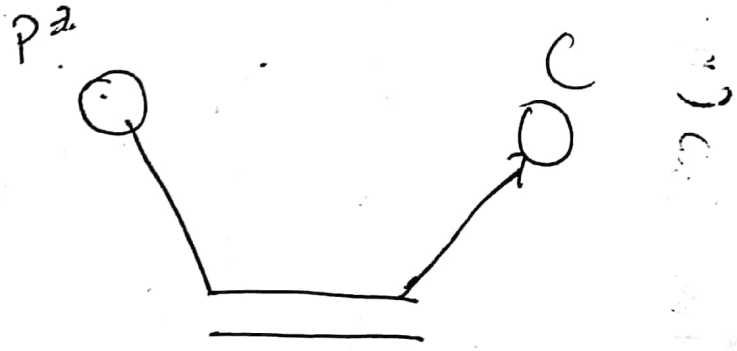
Critical Section.

Soln

⇒ placing a rock on the edge of the bridge

⇒ pick the rock, drive across, put the rock back.
if no rock then wait

(2) producer - Consumer model



P0 ⇒ don't write if Buffer is full.

C ⇒ not take data. when buffer is empty.

Variable count is provided. to measure no. of item in buffer

++ when incremented

-- when removed.

OS :- Controls & Coordinates the use of H/W among the different processes & app^n.

↳ Resource Utilization
↳ Resource allocation.
↳ Process Management
↳ File Management
↳ I/O management
↳ Device Management.

```
#include<stdio.h>
#include<conio.h>
double square root(double n)
{ return pow(n,.5);
}
int main()
{ double n, result;
  printf("enter no.\n");
  scanf("%lf",&n);
  result = Squareroot(n);
  printf("%lf", result);
  return 1;
}
```

long int = 8 byte.
d. double int 8 byte.
float = 4 byte.