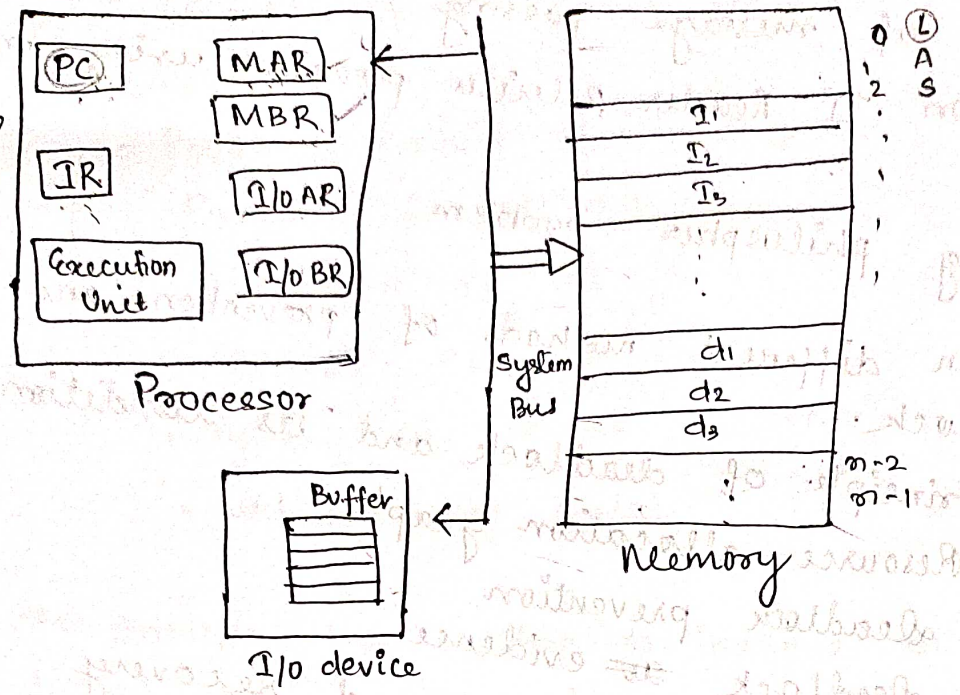
- Disk Scheduling
- Introduction to UNIX OS

Ch - 1

(Overview of Computer System)



PC → Stores the address of the next instruction to be fetch

IR → Stores the current data

MAR → It will save the address from where data is fetched. (Address of the data which should be modified) → from where Read/write operation will be performed.

MBR → Data

There are four main structural component of computer system are :-

i) Processor : It controls the operation of the computer and performs its data processing function.

ii) Main Memory : It stores data and programs, this memory is volatile means when the system is turned off the contain of memory will be lost

iii) I/O Modules or I/O device :- It helps in moving data between computer and its external

environment. The external environment consist of variety of devices including secondary memory devices, communication equipments and terminals.

System Bus :- It provides communication among processor, memory and I/O devices.

There are four internal register present inside the processor for exchanging the data with memory.

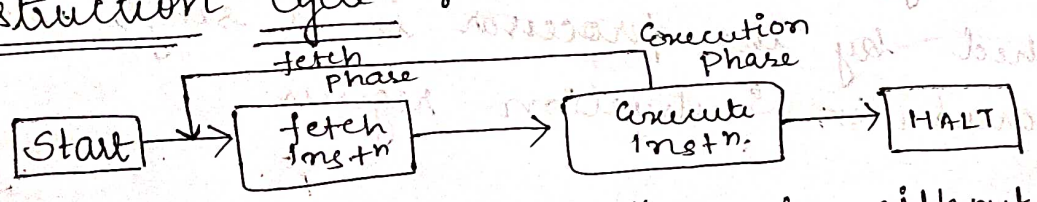
i) MAR (Memory Address Register) : It specifies the address in the memory for next Read/Write operation.

ii) MBR (Memory Buffer Register) : It contains data to be written into the memory.

iii) I/O AR : It specifies particular I/O device.

iv) I/O BR : It is for exchange of data between ~~memory~~ processor and I/O device.

Instruction Cycle :-



Instruction cycle without interrupt

Instruction processing consist of two steps :-

i) Processor Reads / fetches instruction from memory one at a time.

ii) Processor execute each instruction for program execution this process will be repeated till all the

instruction are executed.

Q What is Instruction cycle :-

→ The processing requires for a single instruction is called an instruction cycle.

i) fetch phase :- The processor fetches the instruction from the main memory.

ii) Execution phase :- The fetched instruction will be executed by the processor with the help of an execution unit.

HALT :- It occurs when the system is turned off or some external error postpones the execution of the instruction.

PC : (Program Counter)

It contains the address of the next instruction to be fetched

IR (Instruction Register) :-

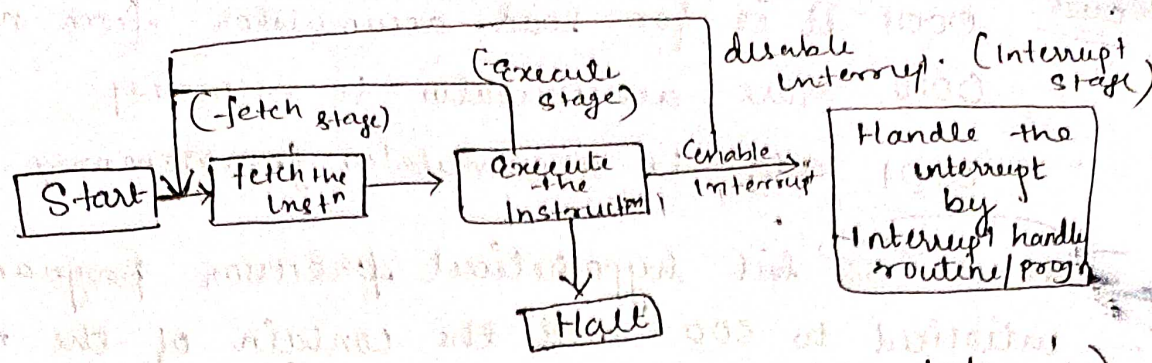
The data fetched by the processor in a single instn cycle which saved in instruction register.

Ques-1

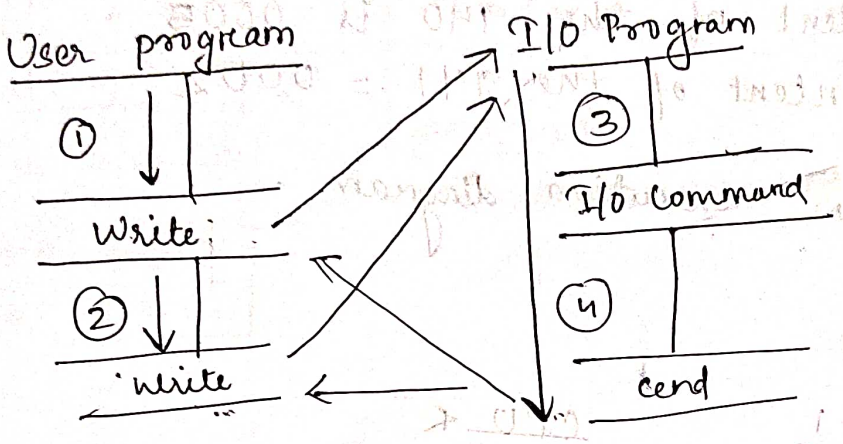
How to draw the ~~exec~~ instruction cycle if any operation is to be performed (like addition, subtraction etc) using memory Address Register, MBR, PC, IR with given opcode ?

Interrupt :-

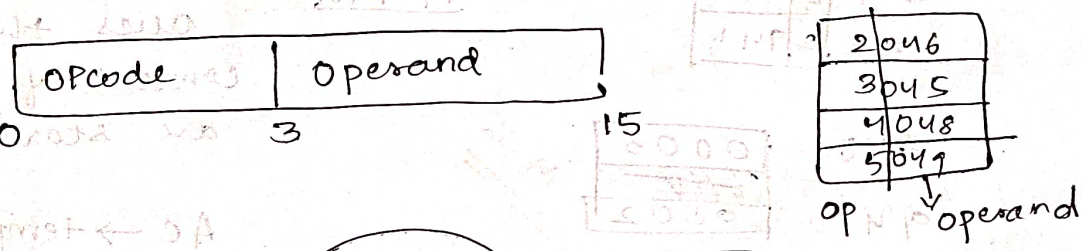
1) Program
2) Timer
3) I/O
4) Hardware

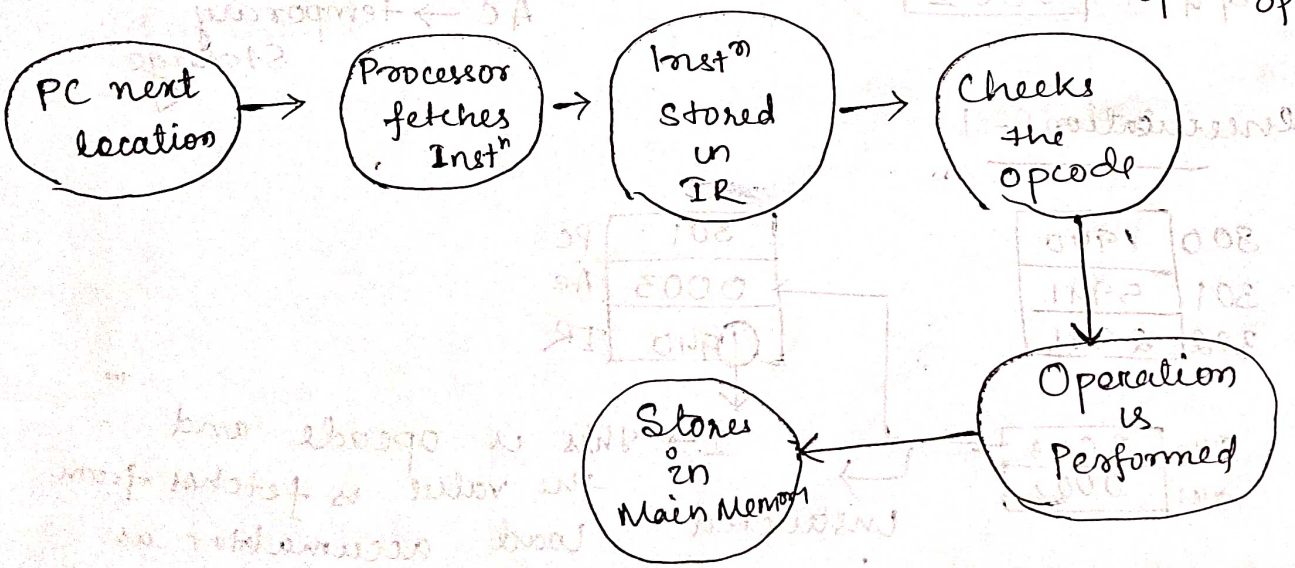


(Instruction Cycle with Interrupt)



(execution flow of Interrupt)





Ques

.0001 it is for load accumulator from memory

0010 Store accumulator to memory

0101 add to accumulator from memory

In a 32 bit hypothetical processor program counter is intialized to 300 add the contain of the memory word at address 940 to the content of memory word at address 941 and store the result in the later location. The content of the 940 is 0003

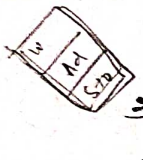
content of the 941 = 0002

Show the program execution diagram.

Ans) PC = 300

fetch-1

load
add
Store



300	1	940
301	5	941
302	2	941

940	0003
941	0002

M 940

CPU

300	PC
	AC
1940	IR

2 941 because it is mentioned in ques that the content of 940 & 941 are stored in 941

AC → temporary Storage

execution - 1

300	1940
301	5941
302	2941

940	0003
941	0002

Instructed

301	PC
0003	AC
1940	IR

1 → this is opcode and the value is fetched from load accumulator as it is instructed load AC we will load AC.

F-2

300	1940
301	5941
302	2941

301	PC
0003	AC
5941	IR

940	0003
941	0002

0003
0002
0005

E-2

300	1940
301	5941
302	2941

302	PC
0005	AC
5941	IR

↳ we have to add to acc(AC) from memory

940	0003
941	0002

0003 →
+ 0002
0005

F-3

300	1940
301	5941
302	2941

302	PC
0005	AC
2941	IR

940	0003
941	0002

E-3

300	1940
301	5941
302	2941

303	PC
0005	AC
2941	IR

940	0003
941	0002

we have to store AC in memory i.e 941

940	0003
941	0005

LAB

UNIX OS

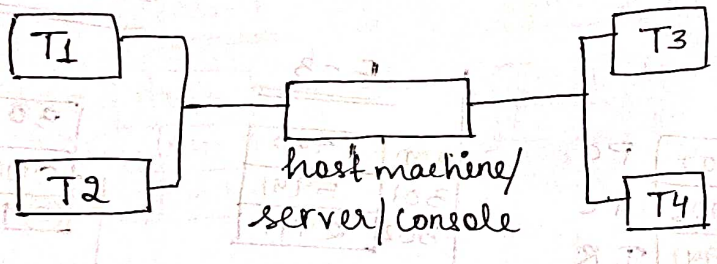
→ The UNIX OS is developed in 1970s at Bellab.
→ It is a set of programs that act as a link between the computer and the user.

Features of UNIX OS :-

* Multiuser Capability :-

→ The same computer resources can be accessable to many users not only same computers but are given different terminals to operate them.

→ A terminal is a keyboard and a monitor which acts like an input and output device for that user. All the terminals are connected to the host machine where resources are to be utilised.



A user at any terminal can use not only the computer but also other peripheral devices.

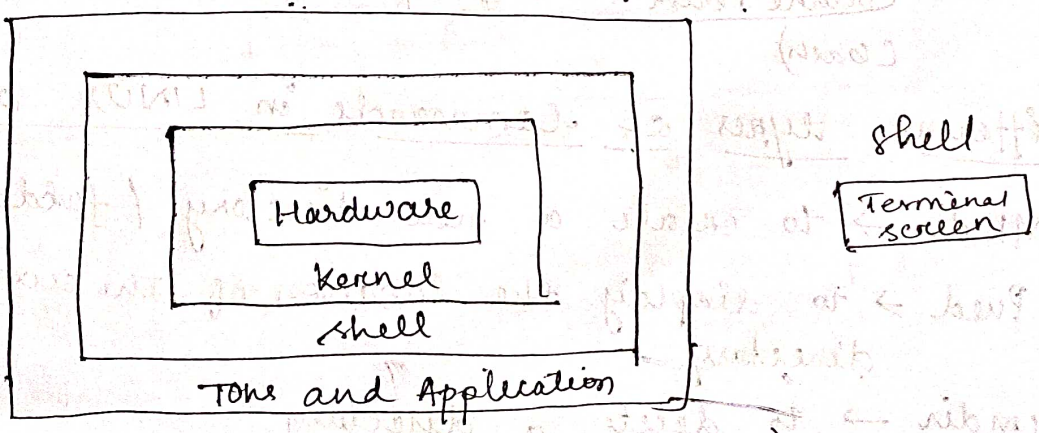
- The number of terminals that can be connected to the host machine depends on the number of ports in the controller car.
- There are three types of terminals
  i) Dumb terminal
     (Keyboard + Monitor)
  ii) Terminal elimination (K+M + Memory + Disk)
     (Connected through cable line)

iii) Dial - in terminal (K + M + Mc + D)
(connected through telephonic lines)

* Multitasking
It is a capable of carrying out multiple jobs at the same time
This is managed by dividing the CPU time intelligently among all the processes.

UNIX OS provides 3 types of security :-

1) Assigning user name and password to individual users confirming that not anybody can come and access your work.
2) At the file label there are read, write and execute permission to each file which decides who can access a particular file, who can modify it and who can execute it.
3) It encodes your file into an unreadable format so that someone succeeds in opening it your secret will be saved.
4) Portability : It can be portable to almost all computer system
5) Communication : The communication maybe within the network of a main computer or between two or more such computer networks.



(Architecture application)

Types of shell :-

1) Bourne shell speed + compatibility
2) C-shell Numeric + logical expression
3) Korn shell (Arrays + fun^n + string + manipulation)
4) Bourne-Again-Bourne (Arrow key)

Kernel :- It is the core of the UNIX OS which carries out most of the task like memory management, file management and task management etc.

Shell / Command intrepreter :-

It is a mediator which interprets the commands that we give and convey them to the kernel which ultimately execute them.

Shell	Path	Non-Rooted User prompt	Rooted User prompt
Bourne (sh)	/bin/sh and sbin/sh	$	#
C-shell (csh)	/bin/csh	%	#
Korn (ksh)	/bin/ksh	$	#
Bourne-again-Bourne (bash)	/bin/bash	bash-X.X x $	bash-X.XX#

Different types of commands in LINUX OS :-

Mkdir → to create a new directory / folder

Pwd → to display the location of the current working directory

rmdir → to delete a directory

ls :- it is used to display a list of content of a directory

cd : change directory

cp : copy the directory and its contents.

rm : to delete a file.

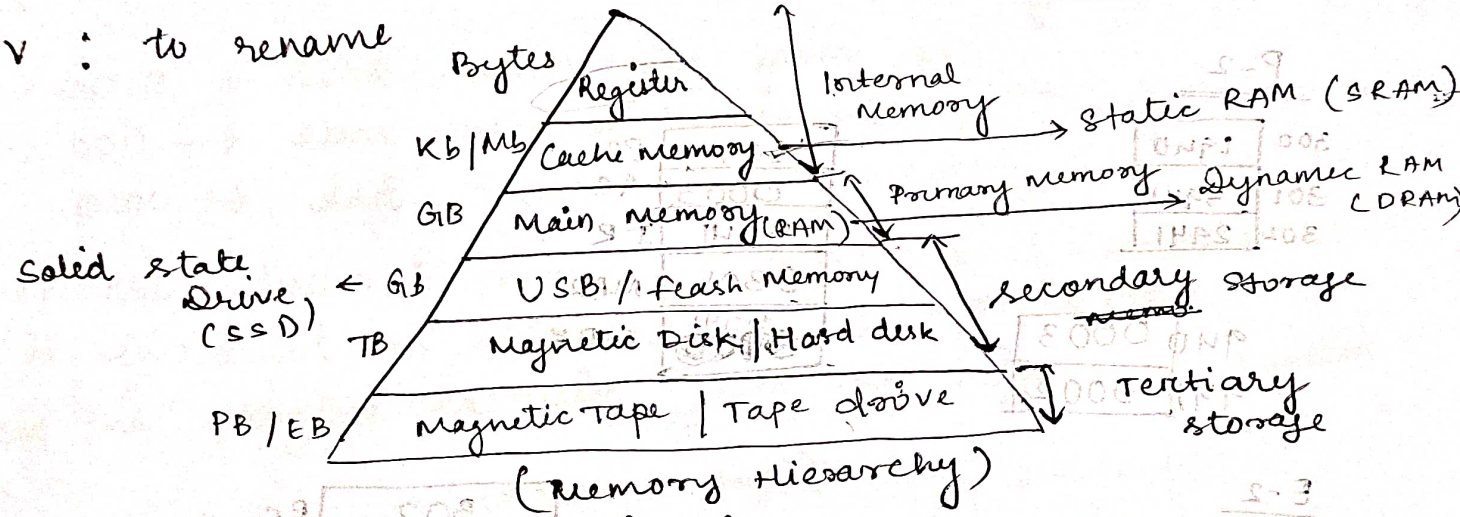
uname : it is to know the information about the OS

locate : to find a file from the database

touch : to create an empty file

cat : to concatenate the content of the multiple files

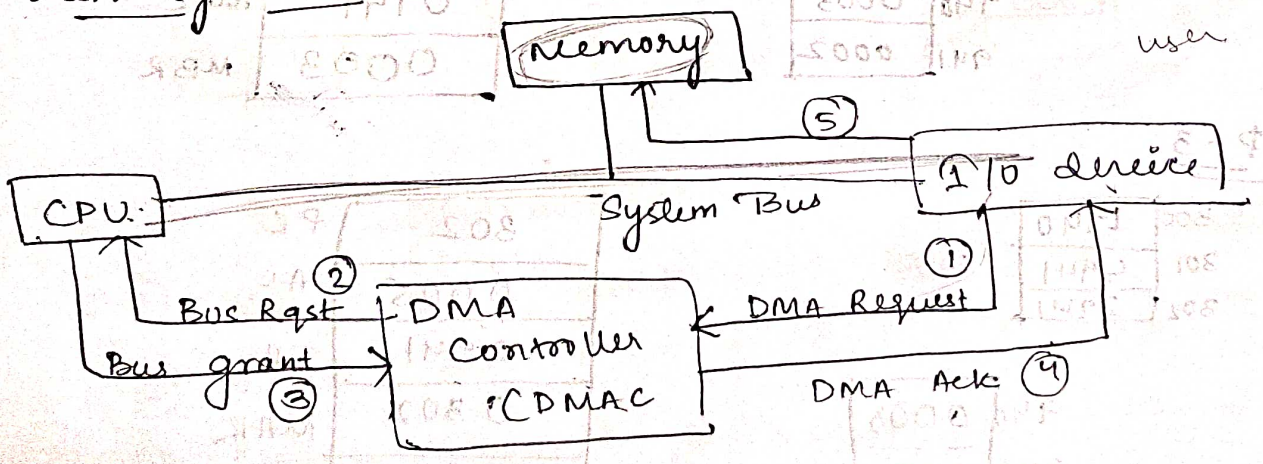
mv : to rename



when we move down size ↑

Access time ↓

Direct Memory Access :- (DMA)



- Same Question (940, 941) using MAR, MBR

Phase - 1

300	1940
301	5941
302	2941

300	PC
	AC
1940	IR
0300	MAR
1940	MBR

MAR - Stores the address i.e 300 as it is 3 bytes → 12 bits we need ~~for~~ 16 bits so add 0 i.e 0300

E-1

300	1940
301	5941
302	2941

940	0003
941	0002

301	PC
0003	AC
1940	IR
0940	MAR
0003	MBR

P-2

300	1940
301	5941
302	2941

940	0003
941	0002

301	PC
0003	AC
5941	IR
0301	MAR
5941	MBR

E-2

300	1940
301	5941
302	2941

940	0003
941	0002

302	PC
0005	AC → Add 2+3
5941	IR
0941	MAR
0003	MBR

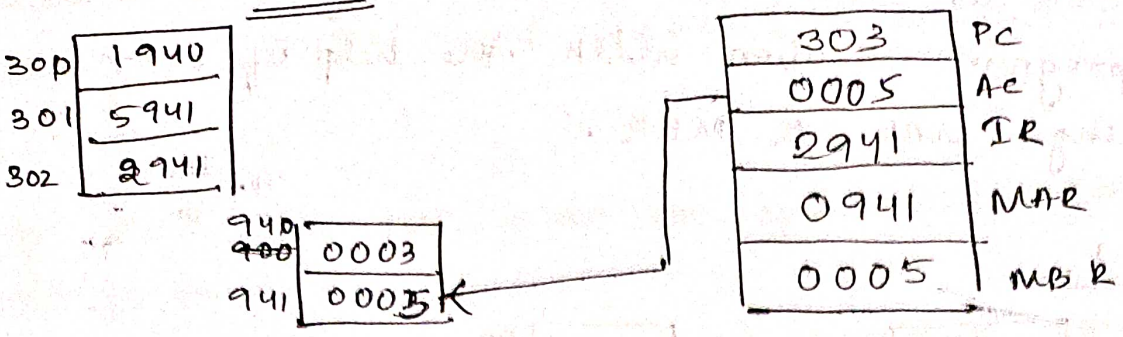
P-3

300	1940
301	5941
302	2941

940	0003
941	0002

302	PC
0005	AC
2941	IR
0302	MAR
2941	MBR

E - 3



→ The memory hierachy depends on size and access time when size increases access time and speed increase.

→ The register in memory hierachy is used the most.

Ques-1

0010 → Load accumulator from I/O

0011 → Store accumulator to memory content 351.

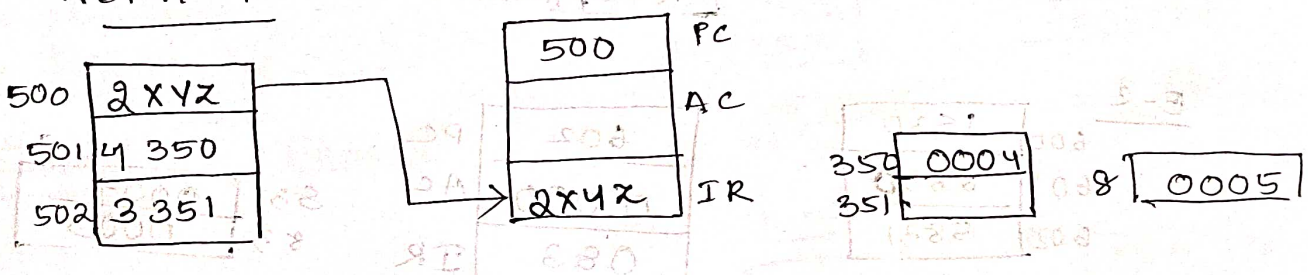
0100 → add accumulator to memory location 350.

An external device is named as light & it contains 0005. P.C is initialised to 500, content of 350 is 0004. Draw the program execution with the help of instruction cycle.

(In next page)

Ans)

fetch - 1



Q

P.C is initialised to 600

① 0001 – Load accumulator from memory ~~location 831~~

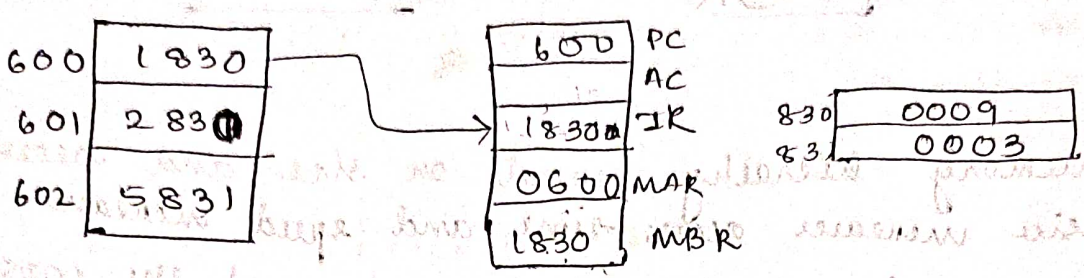
② 0010 – subtract from AC of memory location 831

③ 0101 – Store AC to memory location 831

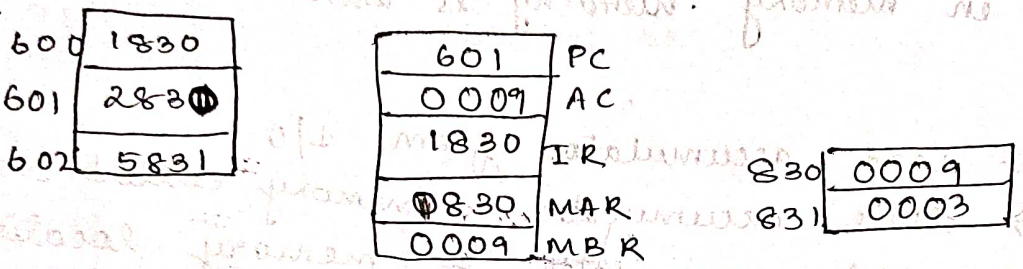
Content of 830 is 0009 and content of 831 is 0003. Draw the program execution with the help of instrn. cycle including MAR & MBR.

(Ans)

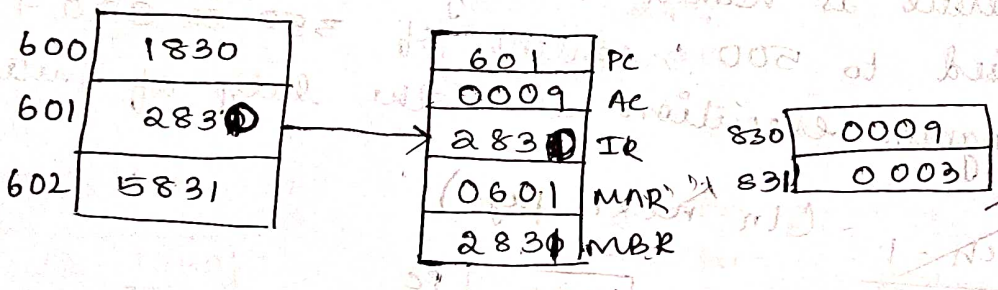
F-1



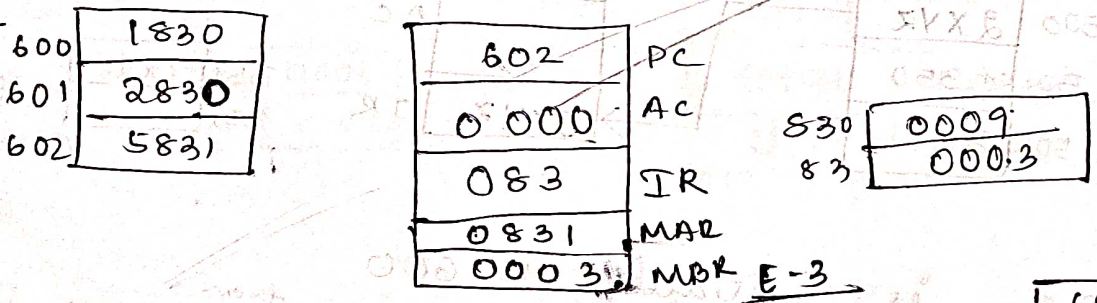
E-1



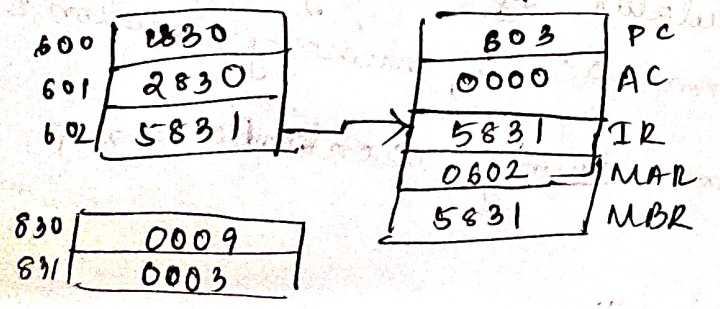
F-2



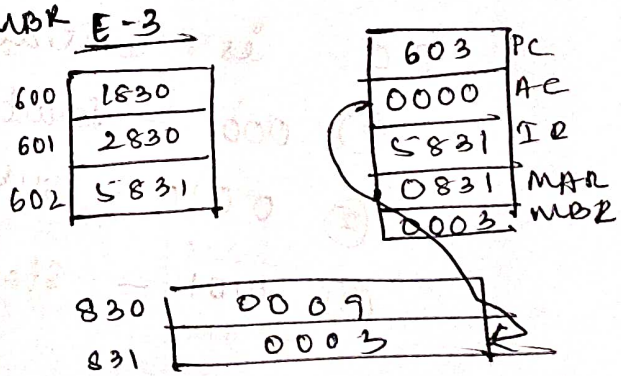
E-2



F-3



E-3

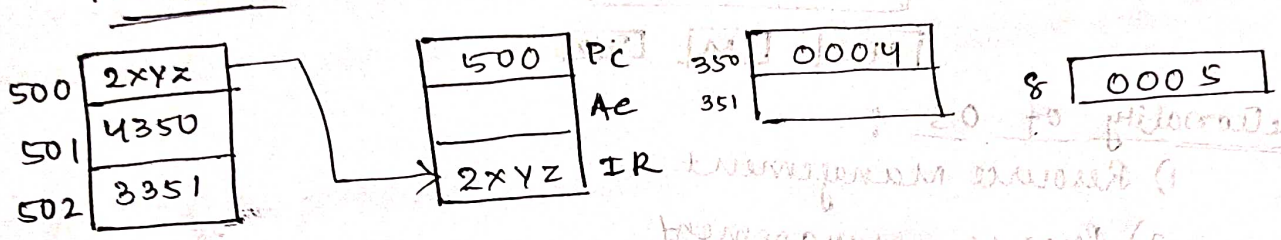


Q write the commands to create an another directory dir2 in directory dos_assgn1 and make dir2 as the current working directory

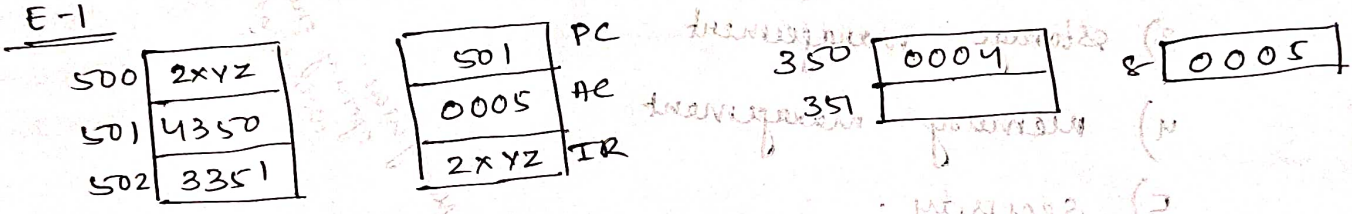
Q To delete the dir2 when dos_regdno will be the current working directory.

Ans) cd dir2, cd.., cd.., rmdir doassign1/dir2

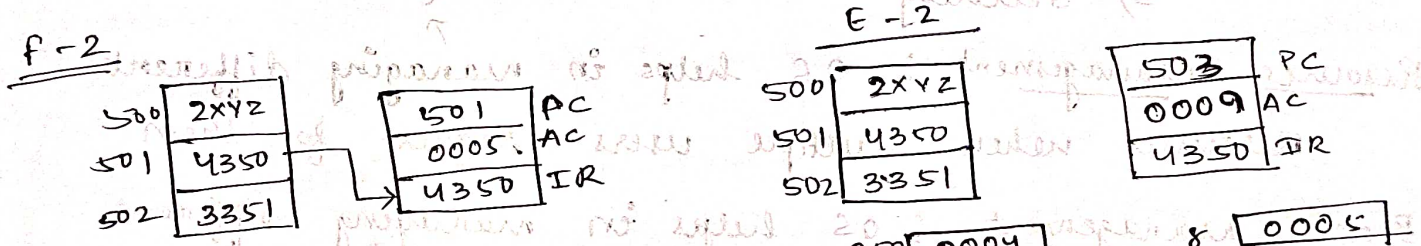
Q-1 Fetch-1



E-1

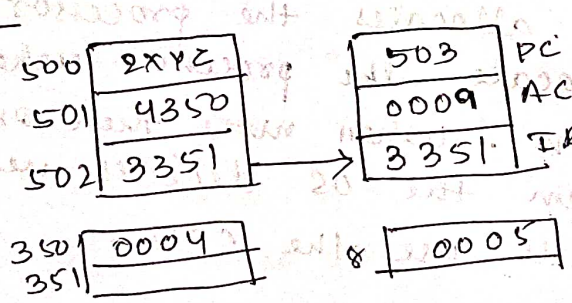


F-2

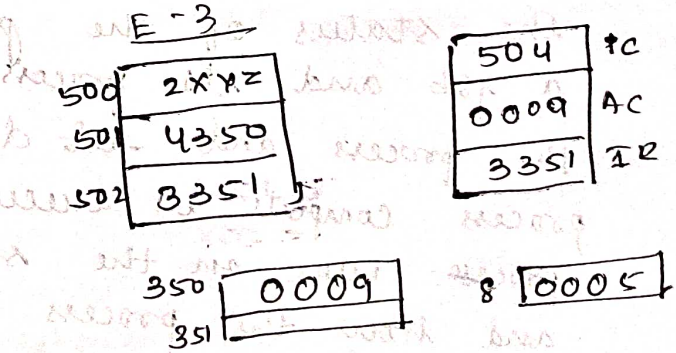


E-2

F-3



E-3

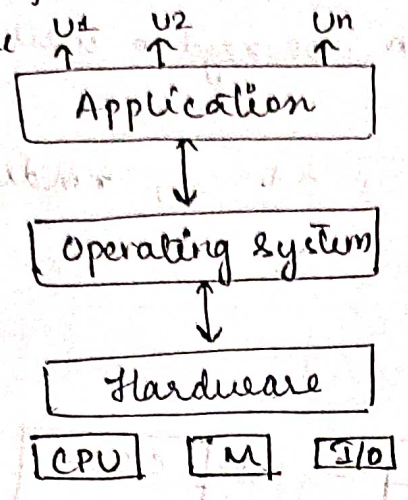


Operating System

Operating System Overview:

OS is the interface between user and hardware.

OS is the software inside the computer



Two factors:

1) Convinience
2) Throughput. (No. of tasks executed per unit time)

functionality of OS:

1) Resource Management
2) Process management
3) storage management
4) Memory management
5) security.

Many users opening one website at time

Resource Management: OS helps in managing different resources when multiple users request for them

Process Management: OS helps in managing different task when an arrive simultaneously. It determines the status of the processor and the processes; selects a job and its processor. allocates the processor to the process and ~~det~~ deallocate the processor when the process complete its execution. when more then one process runs on the system the OS decides when and how the process will use the CPU.

Storage management : OS allows the user to maximize the utilisation of the storage device while protecting the data integrity. (Hardware (secondary M)) (permanently save)

Memory management : It is the management of main memory. A primary memory, whatever program is executed it has to present in the main memory. when program complete its execution the memory region is ~~relised~~ released and can be used by other programs.

Therefore, there can be more than one program present at a time. So, memory management is required & OS handles this.

Security : OS provide security by giving authentication through login and employes a firewall

- window's uses kerberos security protocol

Evolution of Operating System :

→ 1st generation (1945 - 1955) → vacuum tubes, Plug Boards

→ 2nd generation (1955 - 1965) → Batch OS (Non preemptive) ← After one complete another process works

→ 3rd ~~Gen~~ generation (1965 - 1980) (preemptive)

- Multiprogramming OS (Reduce CPU idleness)
- Multitasking { Time sharing
  ↳ to reduce waiting time.

→ 4th generation (1980 - Now)

→ GUI (Graphical User Interface)

- CN (More use of Computer Network)
- Real time OS. [Hard Realtime OS / Soft Realtime OS]
- Distributed OS
- Clustered OS
- Embedded OS

Evolution of Operating system :

PCB → (Process Control Block) : Used to track the process execution status

Process ID
Process State
Program Cont
Priority
CPU scheduling info
I/O related info
Accounting
Memory Mgt info
CPU Reg

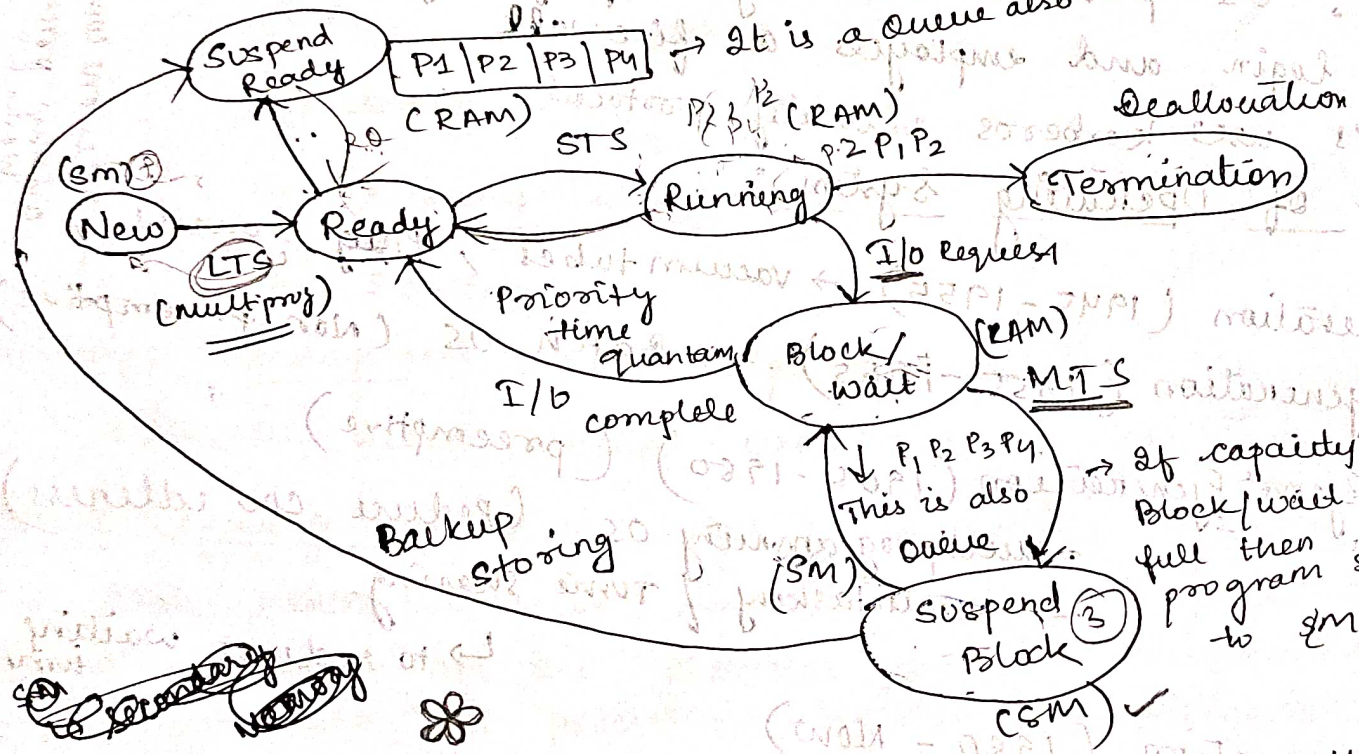
STS → Short term schedule

LTS → Long term schedule

SM → secondary Memory

→ All are queue

MTS → Medium Term schedule



Nonpreemptive → No process will jump in b/w

New : New is the first ~~stage~~ state of the process state cycle. when process creation is taking place, the process is in new state

Ready : when process creation gets completed the process comes into the ready state during this state the process is loaded into main memory and it is placed in Queue called as Ready Queue

Running state: whenever a CPU is allocated to a process from Ready Queue, the state of the process changes from Ready to Running.

Terminated / Completed : when the entire set of instⁿ is executed and the process is completed then the state of the process changes from running to terminated.

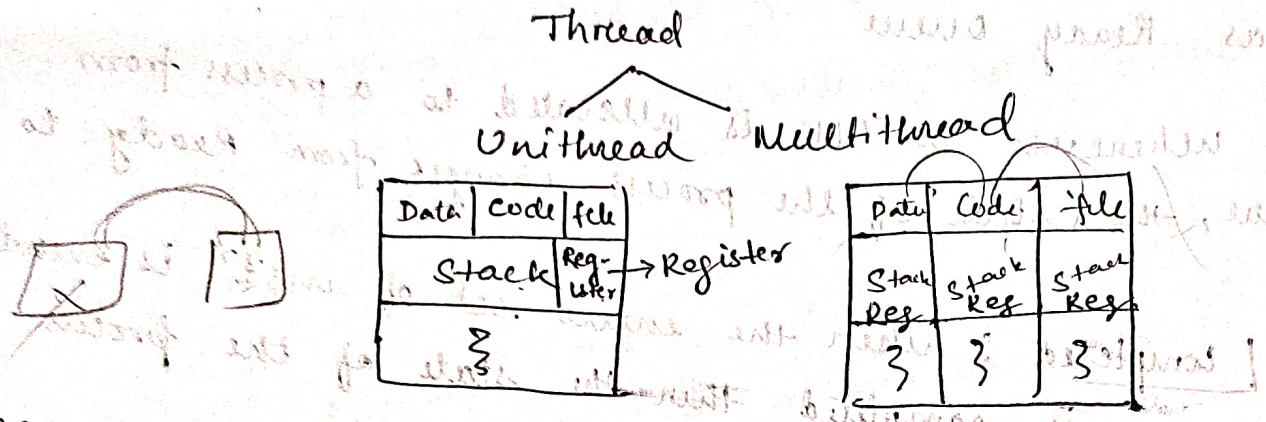
Block / wait : when the process is executing the instructions the process might require some I/O operation which CPU cannot execute. During such condition process is bought back to the main memory and the state is change to block or wait state. In this case process is placed in a Queue called as wait Queue in the main memory.

Suspend wait : whenever the process that is in waiting for / wait state or block state in main memory, swap out to the secondary memory due to main memory being completly filled, the state of the process is changed to suspend wait or suspend Block.

Suspend Ready : whenever the main memory is filled the process which is in ready state is swapped out from main

memory to secondary memory. So, the change of the state is called suspend Ready state. Once the main memory has the enough space for the process, the process will be brought back to the main memory and in the ready state.

Thread :



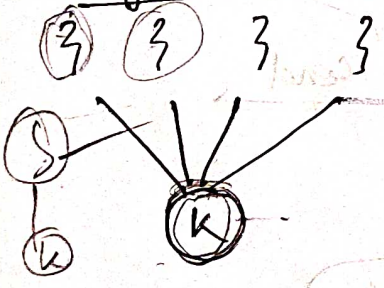
Difference between Process and Thread

Process	Thread
→ System calls are involved in process.	→ There is no system calls involved in process
→ OS treates different process differently	→ All the user level threads treated as the single task for OS.
→ Different process have different copies of data file and code.	→ Thread shares same copy of data file & code.
→ Context switching is slower	→ Context switching is faster.
→ Blocking a process will not block the another (Independent)	→ Blocking a thread will block entire process.

User level thread	Kernel level thread
→ It is implemented by users or API	→ It is implemented by OS.
→ implementation of user threads is easy	→ implementation of kernel state is complicated.
→ context switching time is Less.	→ Context switching time is more.
→ Context switching require no hardware support.	→ Context switching require hardware support.
→ If one user level perform blocking ~~performance~~ operation then the entire process will be blocked.	→ If one kernel thread perform blocking operation then another thread can continue the execution.
→ Uselevel thread are more portable than kernel level thread.	→ Kernel level thread are less portable than user level thread

Multithreading Model :

Many - to - one :-

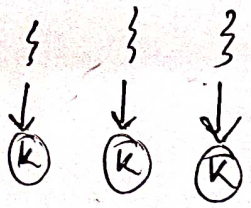


→ It maps many user level threads to kernel level threads.

→ Thread management is done by thread library in user space.

disadvantage :- the entire process will be blocked if a thread makes a blocking system call.

One - to - one :-



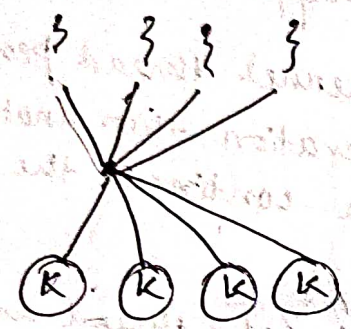
→ It maps one user level thread to one kernel level thread.

→ It provides more concurency than many-to one model because another thread will

run when a thread will make a blocking system call.

Limitation :- Creating a user thread requires creating a corresponding kernel thread. but the overhead of creating kernel threads can burden the performance of an application.

Many-to-Many :-

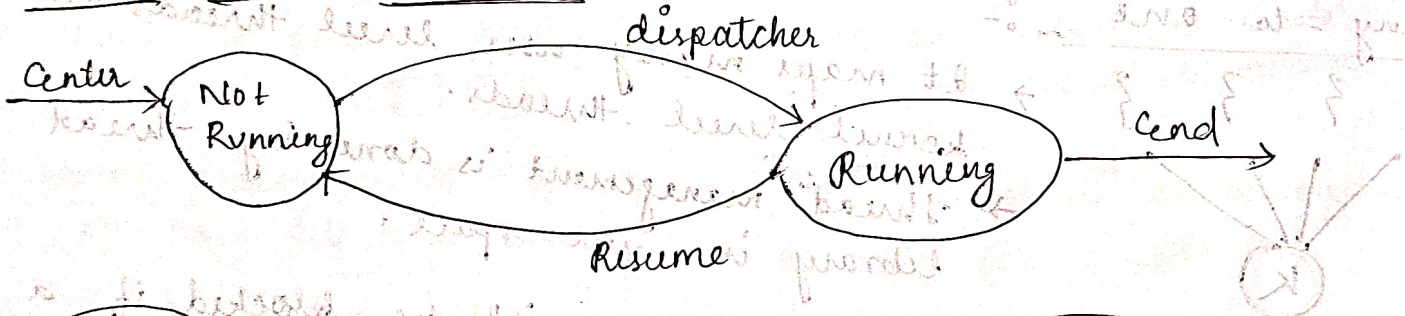


→ It multiplexes many user level threads to equal or smaller no. of kernel level threads.

→ Developers can create as many user thread as necessary & the corresponding kernel threads can run in parallel on a multiprocessor

→ when a thread performs ~~wt~~ a blocking system call the kernel can schedule another thread for execution.

2 state Process Model :-



New
fab queue

Block
device Q
wait Q
Block Q

Ready
Ready Queue

Response Time :-

The point of the time when the process gets the CPU for the first time - arrival time.

First Come First serve : (FCFS)

P1 P2 P3

P1 P2

Mode : Non-preemptive

Basis : arrival time.

Ques

P. No	AT	BT
P1	0	2
P2	1	3
P3	2	4
P4	4	2

Calculate the completion time, turn around time, waiting time, Response time, average turn around time, avg waiting time for each process using FCFS.

RQ

compare AT compare AT

P1	P2	P3	P4

Ans)

P1	P2	P3	P4

0 (0+2) = 2 (2+3) = 5 (4+5) = 9 9+2 = 11

GC = 11

Gantt chart

CT	(CT-AT) TAT	(TAT-BT) WT	(GC-AT) RT
2	2	0	0
5	4	1	1
9	7	3	3
11	7	5	5

Q-2

P.No	AT	BT
P_1	2	1
P_2	4	3
P_3	6	7
P_4	7	5

RQ

~~P_1~~	~~P_2~~	~~P_3~~	P_4

GC

//	P_1	//	P_2	P_3	P_4

0 2 3 4 7 14 19

CT	TAT	WT	RT
3	1	0	0
7	3	0	0
14	8	1	1
19	12	7	7

Q-3

P.No	AT	BT
P_1	1	2
P_2	2	4
P_3	4	6
P_4	3	3
P_5	7	1

A

RQ

~~P_1~~	~~P_2~~	~~P_4~~	~~P_3~~	~~P_5~~

1

	P_1	P_2	P_4	P_3	P_5

0 1 3 7 10 16 17

P1 AT

P1 BT + prev val

CT	Turn Around T	WT	RT
0	0	0	0
3	2	0	0
7	5	1	1
16	12	6	6
10	7	4	4
17	10	9	9

Q-4

P.No	AT	BT
P_1	1	2
P_2	0	3
P_3	1	2
P_4	5	1
P_5	3	4
P_6	5	2

Ans) RQ

~~P_2~~	~~P_1~~	~~P_3~~	~~P_5~~	~~P_4~~	P_6

P_2	P_1	P_3	P_5	P_4	P_6

0 1+2 = 3 5 7 11 12 14

gant chart

CT	TAT	WT	RT
5	4	2	2
3	3	0	0
7	6	4	4
12	7	6	6
12	8	4	4
14	9	7	7

Shortest job first (SJF) :- (Non-preemptive)

Ques-3
↳ same Ques

RQ

~~P1~~	~~P2~~	~~P4~~	P3	~~P5~~

Add BT here
If BT same
Check
AT

Gantt chart

	P1	P4	P2	P5	P3

0 1 3 6 10 11 17

Q-2

RQ

~~P2~~	~~P1~~	~~P3~~	~~P4~~	~~P5~~	~~P6~~

GC

P2	P1	P4	P3	P6	P5

0 3 5 6 8 10 14

BT of P2

	AT	CT	TAT	WT	RT
P2	0	5	4	2	2
P1	3	3	3	0	0
P4	5	8	7	5	5
P3	6	6	1	0	0
P6	8	14	11	7	7
P5	10	10	15	3	3

Q1

P.No	AT	BT
P_1	1	2
P_2	2	2
P_3	4	3
P_4	0	3
P_5	6	1

Q-1

Ans

RQ

~~P_4~~	~~P_1~~	~~P_2~~	~~P_3~~	P_5

GC

P_4	P_1	P_2	P_3	P_5

0 3 5 7 10 11

	CT	TAT	WT	RT
	5	4	2	2
	7	5	3	3
	10	6	3	3
	3	3	0	0
	11	5	4	4

2

P.No	AT	BT
P_1	1	5
P_2	0	6
P_3	1	1
P_4	2	2
P_5	5	3
P_6		

SJF

RQ

~~P_4~~	~~P_1~~	~~P_2~~	P_3	~~P_5~~

GC

P_4	P_1	P_2	P_5	P_3

0 3 5 7 8 11

CT	TAT	WT	RT
5	4	2	2
7	5	3	3
11	7	4	4
3	3	0	0
8	2	1	1

Q-2

	AT	BT
P_1	1	5
P_2	0	6
P_3	1	1
P_4	2	2
P_5	5	3
P_6	2	4

RQ

~~P_2~~	~~P_1~~	~~P_3~~	~~P_4~~	P_5	~~P_6~~

GC

P_2	P_1	P_3	P_4	P_6	P_5

0 6 11 12 14 18 21

CT	TAT	WT	RT
11	10	5	5
6	6	0	0
12	11	10	10
14	12	10	10
21	16	13	13
18	16	12	12

SJF

RQ

~~P_2~~	~~P_1~~	~~P_3~~	~~P_4~~	~~P_6~~	~~P_5~~

GC

P_2	P_3	P_4	P_5	P_6	P_1

0 6 7 9 12 16 21

CT	TAT	WT	RT
21	20	15	15
6	6	0	0
7	6	5	5
9	7	5	5
12	7	4	4
6	14	10	10

CT	TAT	WT	RT
21	5	0	15
6	7	1	0
7	8	7	5
9	10	8	5
12	11	8	4
26	19	15	10

Q-3

PN	AT	BT
P_1	2	4
P_2	3	2
P_3	5	4
P_4	6	5
P_5	7	6

RQ

~~P1~~	~~P2~~	~~P3~~	~~P4~~	P5

GC

	P_1	P_2	P_3	P_4	P_5

0 2 6 8 12 17 23

CT	TAT	WT	RT
6	4	0	0
8	5	3	3
12	7	3	3
17	11	6	6
23	16	10	10

SJF

RQ

~~P1~~	~~P2~~	~~P3~~	~~P4~~	P5

GC

	P1	P2	P3	P4	P5

0 2 6 8 12 17 23

CT	TAT	WT	RT
6	4	0	0
8	5	3	3
12	7	3	3
17	11	6	6
23	16	10	10

Shortest Remaining Time first :- (preemptive)

Ques

P No	AT	BT
P_1	1	~~5~~
P_2	0	~~6~~ ~~5~~
P_3	1	~~1~~ 0
P_4	2	~~2~~ 0
P_5	5	~~3~~
P_6	2	~~4~~ ~~3~~ 0

RQ

~~P2~~	~~P1~~	~~P3~~	~~P4~~	~~P6~~	~~P5~~

GC

P_2	P_3	P_4	P_6	P_6	P_5	P_2	P_1

0 1 2 4 5 8 11 16 21

CT	TAT	WT	RT
21	20	15	15
16	16	10	10
2	1	0	0
4	2	0	0
11	6	3	3
8	6	2	2

Q-3

PNO	AT	BT
P1	2	~~4~~ ~~3~~ ~~2~~ 0
P2	3	~~2~~ 0
P3	5	4
P4	6	5
P5	7	6

RQ

~~P1~~	~~P2~~	~~P3~~	~~P4~~	~~P5~~

GC

////	P1	P2	P1	P1	P1	P3	P4	P5

0 2 3 5 6 7 8 12 17 23

CT	TAT	WT	RT
8	6	2	0
5	2	0	0
12	7	3	3
17	11	6	6
23	16	10	16

Q-4

PNo	AT	BT	CT	TAT	WT	RT
P_1	2	4	10	8	4	4
P_2	1	2	3	2	0	0
P_3	3	4	14	11	7	7
P_4	5	1	15	10	9	9
P_5	1	3	6	5	2	2

RQ

~~P_2~~	~~P_5~~	~~P_3~~	~~P_1~~	P_4

GC

////	P_2	P_5	P_1	P_3	P_4

0 1 3 6 10 14 15

P No	AT	BT
P1	2	4
P2	1	~~2~~ 0
P3	3	4
P4	5	~~1~~ 0
P5	1	~~3~~ 1

RQ

~~P2~~	~~P5~~	P2	P3	P4

GC

//	P2	P5	P4	P1	P3

0 1 3 6 7 11 15

SRTF

RQ

~~P2~~	~~P5~~	~~P1~~	~~P3~~	~~P4~~

GC

//	P2	P2	P5	P5	P4	P1	P3

0 1 2 3 5 6 7 11 15

PRIORITY :-

Mode : Non - Preemptive / preemptive

Criteria : Priority

Higher no = Higher P

Q-1

P.No	AT	BT	Priority
P_1	0	2	2
P_2	2	3	4
P_3	2	1	6
P_4	3	2	1

Higher no = ~~Lower P~~ Higher P

RQ

P_1	P_2	P_3	P_4

GC

P1	P3	P2	P4

0 2 3 6 8

Higher no = lower P

RQ

P_1	P_2	P_3	P_4

GC

P1	P2	P4	P3

0 2 5 7 8

If priority is same then we will compare AT.

Q-2

P.No	AT	BT	P
P_1	1	2	9
P_2	3	3	1
P_3	5	2	7
P_4	0	1	6
P_5	7	2	4
P_6	3	6	3

a) Higher no = high P

RQ

P_4	P_1	P_2	P_6	P_3	P_5

GC

P_4	P_1	P_6	P_3	P_5	P_2

0 1 3 9 11 13 16

b) Lower no = higher P

RQ

P_4	P_1	P_2	P_6	P_3	P_5

GC

P_4	P_1	P_2	P_6	P_5	P_3

0 1 3 6 12 14 16

Ques

P.No	AT	BT	P
P1	1	~~2~~ 1	9
P2	3	~~3~~ 2	1
P3	5	~~2~~ 0	7
P4	0	10	6
P5	7	2	4
P6	3	~~8~~ 4	3

FCFS

RQ

~~P4~~	~~P1~~	~~P2~~	~~P6~~	~~P3~~	~~P5~~

GC

P4	P1	P2	P6	P3	P5

0 1 3 6 12 14 16

SJF

RQ

~~P4~~	~~P1~~	~~P2~~	P6	~~P3~~	~~P5~~

0

GC

P4	P1	P2	P3	P5	P6

0 1 3 6 8 10 16

SRTF

RQ

~~P4~~	~~P1~~	~~P2~~	~~P6~~	~~P5~~

GC

P4	P1	P2	P6	P5	P2	P6

0 1 3 5 7 9 10 14

RQ

~~P4~~	~~P1~~	P2	P6	P3	P5

GC

P4	P1	P2	P2	P3	P2	P5	P6

0 1 3 4 5 7 8 16

RQ

~~P4~~	~~P1~~	~~P2~~	~~P6~~	~~P3~~	~~P5~~

BT

1 — 2
2 — ~~1~~
~~3~~ — ~~1~~
4 — 1
5 — 2
~~6~~ — 6

GC

P4	P1	P2	P2	P3	P3	P6	P5

0 1 3 5 6 7 8 10 16

Priority with preemptive :-

RQ

~~P4~~	~~P1~~	P2	~~P6~~	~~P3~~	~~P5~~

GC

P4	P1	P6	P3	P3	P5	P6	P2

0 1 3 5 6 7 9 13 16

GC

P4	P1	P6	P3	P3	P5	P6	P2

0 1 3 5 6 7 9 13 16

Round - Robin Algorthim :-

Mode : Preemptive

Criteria : time quantum

P.No	AT	BT
P1	0	2
P2	2	3
P3	1	1
P4	3	2
P5	5	1

TQ = 2 unit

RQ

~~P1~~	~~P3~~	~~P2~~	~~P4~~	~~P5~~	~~P2~~	~~P5~~

GC

P1	P3	P2	P4	P5	P2	P5

0 2 3 5 7 9 10 12

Q-2

TQ = 2

P.No	AT	BT
P1	1	2
P2	2	~~4~~ ~~2~~
P3	3	~~6~~ ~~4~~ ~~2~~
P4	4	~~8~~ ~~6~~ 4

RQ

~~P1~~	~~P2~~	~~P3~~	~~P4~~	~~P2~~	~~P3~~	~~P4~~	~~P3~~	~~P4~~	~~P4~~

GC

	P1	P2	P3	P4	P2	P3	P4	P3	P4	P4

0 1 3 5 7 9 11 13 15 17 19 21

Q-3

TQ = 3

P.No	AT	BT
P1	1	2
P2	3	3
P3	5	2
P4	0	1
P5	7	2
P6	3	~~6~~ 3

RQ

~~P4~~	~~P1~~	~~P2~~	~~P6~~	~~P3~~	~~P5~~	~~P6~~

GC

P4	P1	P2	P6	P3	P5	P6

0 1 3 6 9 11 13 16

Ques

P.No	AT	BT
P_1	0	~~8~~ 1
P_2	1	~~6~~ 2
P_3	2	3
P_4	3	1
P_5	4	~~5~~ 1
P_6	6	4

TQ = 4

RQ

~~P1~~	~~P2~~	~~P3~~	~~P4~~	~~P5~~	~~P1~~	~~P6~~	~~P2~~	P5

GC

P_1	P_2	P_3	P_4	P_5	P_1	P_6	P_2	P_5

0 4 8 11 12 16 17 21 23 24

Ques

P.No	AT	BT
P_1	0	~~4~~ 2
P_2	0	~~5~~ 1
P_3	0	~~5~~ 3
~~P_4~~		

TQ = 2

RQ

~~P1~~	~~P2~~	~~P3~~	~~P1~~	~~P2~~	~~P3~~	P3

GC

P_1	P_2	P_3	P_1	P_2	P_3	P_3

0 2 4 6 8 9 11 12

Highest Response Ratio Next (HRRN):

Mode – Non-preemptive

Criteria : Response Ratio

Ques

P.No	AT	BT
P1	0	3
P2	2	5
P3	4	4
P4	6	1
P5	8	2

$$RR = \frac{W+S}{S}$$

W = Waiting time

S = Burst time

W = Current time - Arrival time

RQ

P1	P2	P3	P4	P5

0

GC

P1	P2	P4	P3	P5

0 3 8 7 13 15

Ut-1 R.R

P3 = 8 − 4 = 4 = 4/4

P4 = 2 = 3

P5 = 0 = $\frac{0+2}{2}$ = 1

Ut-2

P3 = $\frac{5+4}{4}$ = 9/4 2.25 = 2.25

P5 = $\frac{1+2}{2}$ = 3/2 1.3 = 1.3

High = Hy P

Ques

P.No	AT	BT	P
P1	0	3	4
P2	2	5	3
P3	4	4	2
P4	6	5	1
P5	8	2	6

TQ = 3

RQ

~~P1~~	~~P2~~	~~P3~~	P4	~~P5~~

GC

P1	P2	P3	P5	P4

0 3 9 13 15 20

Ut-1

$P3 = \frac{5+4}{4} = 2.5$

$P4 = \frac{3+5}{5} = 8/5 = 1.6$

$P5 = \frac{1+2}{2} = 3/2 = 1.5$

Ut-2

$P4 = \frac{7+5}{5} = 12/5 = 2.4$

$P5 = \frac{5+2}{2} = 7/2 = 3.5$

FCFS

RQ

~~P1~~	~~P2~~	~~P3~~	~~P4~~	~~P5~~

GC

P1	P2	P3	P4	P5

0 3 9 13 18 20

SJF

RQ

~~P1~~	~~P2~~	~~P3~~	~~P4~~	~~P5~~

0

GC

P1	P2	P5	P3	P4

0 3 9 11 15 20

SR

RQ

~~P1~~	~~P2~~	~~P3~~	P4	~~P5~~

GC

~~P1~~	~~P2~~	~~P2~~	~~P2~~	~~P5~~	~~P3~~	~~P4~~

0 2 4 6 8 10 14

P1	P2	P2	P3	P3	P5	P2	P4

0 2 3 4 6 8 10 15 20

HW

RQ

~~P1~~	~~P2~~	~~P3~~	~~P4~~	~~P5~~

GC

P1	P2	P5	P3	P4	

0 3 9 11 15 20

Round Robin

RQ

~~P1~~	~~P2~~	~~P3~~	P4	~~P2~~	~~P5~~	P3

GC

~~P1~~	~~P2~~	~~P2~~	~~P3~~

0 3 6 9 12

~~P1~~

0

Round-Robin

RQ

~~P1~~	~~P2~~	~~P3~~	~~P4~~	~~P2~~	~~P5~~	P3	P4

GC

P1	P2	P3	P4	P2	P5	P3	P4

0 3 6 9 12 15 17 18 20

Ques

	AT	BT	
P1	0	3	4
P2	2	5 4 3 2 1	3
P3	4	4	2
P4	6	1	1
P5	8	2	6

4 (2)

SRTF

RQ

P1	P2	P3	P4	P5

GC

P1	P1	P2	P2	P4	P2	P2	P5	P3

0 2 3 4 6 7 8 9 11 13

FCFS

RQ

P1	P2	P3	P4	P5

GC

P1	P2	P3	P4	P5

0 3 8 12 13 15

SJF

RQ

P1	P2	P3	P4	P5

GC

P1	P2	P4	P5	P3

0 3 8 9 11 15

SRTF

RQ

P1	P2	P3	P4	P5

GC

P1	P1	P2	P2	P4	P2	P2	P5	P3

0 2 3 4 6 7 8 9 11 15

H.P RQ

P1	P2	P3	P4	P5

G.C

P1	P2	P5	P3	P4

0 3 8 10 14 15

Round

RQ

P1	P2	P3	P4	P2	P5	P3

G.C

P1	P2	P3	P4	P2	P5	P3

0 3 6 9 10 12 14 15

Multilevel Queue Scheduling :-

features of Multilevel Queue Scheduling :-

→ In this algorithm process are divided into multiple queue based on their priorities.

→ Each queue has different priority level.

→ Higher priority processes are placed in queues with higher priority levels while lower priority processes are placed in queues with lower priority level.

Priority assign :

→ priorities are assigned to the processes based on their type, characteristics & importance

Preemption :

In this algorithm a higher priority process can preempte a lower priority process

efficient ~~arrogo~~ allocation of CPU time :-

→ This algorithm ensures that the processes with higher priority level are executed in a timely manner.

Eg

P.No	AT	BT	Q.No
P1	0	~~4~~ 2	1
P2	0	~~3~~ 1	1
P3	0	~~5~~ 2	2
P4	10	3	1

Q1 uses Round Robin.
Q2 uses FCFS
Priority of Q1 > Q2
TQ for Round Robin is 2 unit

Ans)

RQ: Q1 | Q2

Q1: ~~P1~~ | ~~P2~~ | ~~P1~~ | ~~P2~~ | ~~P4~~

Q2: P3

first we will check whose priority is greater then we will execute.

(BT jobs are arrived with in when)

GC

P1	P2	P1	P2	P3	P4	P4	P3

0 2 4 6 7 10 12 13 15

P.No	AT	BT	Q.No
P1	0	~~4~~ 2	1
P2	0	~~5~~ 1	2
P3	9	~~5~~ 2	2
P4	8	2	1
P5	12	3	1
P6	15	1	2
P7	17	~~5~~ 3	1

Q1 use R.R
Q2 uses FCFS
TQ = 2 unit
Q1 > Q2

RQ: Q1 | Q2

Q1: ~~P1~~ | ~~P4~~ | ~~P4~~ | ~~P4~~ | ~~P5~~ | ~~P5~~ | ~~P7~~

Q2: ~~P2~~ | ~~P3~~ | P6

GC

P1	P1	P2	P4	P2	P3	P5	P5	P3	P7	P7	P7	P6

0 2 4 8 10 11 12 14 15 17 19 21 22 23

Different types of process :-

i) System process : CPU itself has its own process to run which is generally termed as system process.

ii) Interactive process or foreground process :- It is a type of process in which there should be same type of interaction.

iii) Batched process or Background process : This process collects the programs and data together in the form of a batch before the processing starts.

Multilevel Feedback Queue Scheduling :-

features :- (Allow process to move b/w queues)

→ Multiple Queues : Similar to multilevel queue scheduling this algorithm divides process into multiple queues based on their priority level. but unlike multilevel queue scheduling process can move between queues on their behaviour and needs.

→ Priority adjusted dynamically : If the priority of a process can be adjusted dynamically based on its behaviour such as how much CPU time it has used or how often it has been blocked.

→ Time slicing : Each queue is assigning time quantum or time slice which determines how much CPU time a process in that queue is granted to use before it is preempted & move to a lower priority queue

→ feedback mechanism : this algorthim uses a feedback mechanism to adjust a priority of a process based on its behaviour over time.

→ Preemption :- Preemption is allowed in this algorthim which means a higher priority process can preempt a lower priority process to get the CPU time it needs.

Advantages of multilevel feedback Queue scheduling :-

→ It is more flexible

→ It permits different processes to move between different queues.

→ It prevents starvation by moving a process that waits too long for the lower priority queue to the higher priority queue.

Ques

Consider a system that has a CPU bound process which requires a BT of 40 sec. The multilevel feedback queue ~~schedu~~ scheduling is used and queue quantum time is 2 sec. in each level it is incremented by 5 sec then how many times the process will be ~~interputed~~ interputed and in which queue the process will terminate the execution.

Ans)

Queue T.Q = 2 sec

BT = 40 sec

2	38
7	31
12	19
17	2
.	0

Q1 [P1] 2

Q2 [P1] 7

Q3 [P1] 12

Q4 [P1] 17

Q5 [P1] 2

Ques

Consider three processes P1, P2 & P3 having AT 0, 1 & 3 and BT 5, 7 & 4 the completion ~~of~~ order of three process under the policy FCFS, Round Robin having TQ = 2 units

i) FCFS : P1, P2, P3

Round Robin : P1, P2, P3

ii) FCFS : P1, P3, P2

R.R : P1, P3, P2

iii) FCFS : P1, P2, P3

R.R : P1, P3, P2

iv) FCFS : P1, P3, P2

R.R P1, P2, P3

Ans)

P.No	AT	BT
P1	0	~~5~~ ~~3~~ 1
P2	1	~~7~~ ~~5~~ 3
P3	3	~~4~~ 2

TQ = 2 units

FCFS

RQ

P1	P2	P3

GC

P1	P2	P3

0 5 12 16

R.R

~~P1~~	~~P2~~	~~P3~~	~~P1~~	~~P2~~	~~P3~~	~~P1~~	~~P2~~

P1	P2	P3	P1	P2	P3	P1	P2	P2

0 2 4 6 8 10 12 13 15 16

RQ

R.R

~~P1~~	~~P2~~	~~P1~~	~~P3~~	~~P2~~	~~P1~~	~~P3~~	P2

P1	P2	P1	P3	P2	P1	P3	P2	P2

0 2 4 6 8 10 11 13 15 16

Ques

Consider a set of n process with known run times R1, R2 ... Rn to be run on a uniprocessor machine. which of the following processor scheduling algorthim results in maximum throughput.

i) Round Robin

ii) shortest job first.

iii) Heghest Response Ratio Next

iv) first come first serve.

Ques

P.No	AT	BT	Priority
P1	2	~~4~~ 3	9
P2	1	~~2~~ 0	2
P3	4	3	3
P4	6	1	1
P5	7	~~5~~ 4	7
P6	12	3	5

Heigh No = Lower P

TO = 3

FCFS

RQ

~~P2~~	~~P1~~	~~P3~~	~~P4~~	~~P5~~	~~P6~~

P1

GC

P2	P1	P3	P4	P5	P6

0 1 3 7 10 11 15 19

SJF

RQ

~~P2~~	~~P1~~	~~P3~~	~~P4~~	~~P5~~	P6

GC

	P2	P1	P3	P4	P5	P6

0 1 3 7 10 11 16 19

SRTF

RQ

~~P2~~	~~P1~~	~~P3~~	~~P4~~	P5	~~P6~~

GC

	P2	P2	P1	P1	P1	P4	P3	P5	P6	P5

0 1 2 3 4 6 7 8 11 12 15 19

Priority

Note : cal -y → calender of whole year
cal 07 2000 → calender of 7 month 2000 year
cal 2020 → calender of 2020
cal -3 → calender of after 3 months
cal -A 2 →
cal -B 2 →

3) date + "%d-%m-%y" → 19-10-23
date + "%d-%m-%Y" → 19-10-2023
date + "%D" → 10/19/23
date + "%T" → 12:47:20
date + "%a%b%d%m%y" → Thu Oct 19-10-23
date + "%A %B %D %m %y" → Thursday October 19-10-2023

5) echo "filename - $(basename dteal.sh)"
echo "linecount - $(wc -l < dteal.sh)"
echo "wordcount - $(wc -w < dteal.sh)"
echo "charcount - $(wc -m < dteal.sh)"

6) echo "Total No. of argument : $#"
echo "first arguments : $1"
echo "second argument : $2"
echo -e "List of argument : $@"

7) echo "first $1 line of $3:"
head -n $1 $3
echo "last $2 line of $3 :"
tail -n $2 $3

Ans

	AT	BT
P1	1	1
P2	3	2
P3	0	3
P4	5	4
P5	2	5

RQ

~~P3~~	~~P1~~	~~P5~~	~~P2~~	P4

GC

P3	P1	P2	P5	P4

0 3 4 6 11 15

$P5 = \frac{4+5}{5} = 9/5 = 1.8$

$P4 = \frac{2+4}{4} = 6/4 = 1.5$

Lt-1

$P1 = \frac{2+1}{1} = 3$

$P5 = \frac{1+5}{5} = 6/5$

$P2 = 1$

Lt-2

$P5 = \frac{2+5}{5}$

$= 7/5 = 1.4$

$P2 = \frac{1+2}{2} = 3/2$

$= 1.5$

System call

→ It is a method for a computer program to request a service from the kernel of the OS. on which it is running.

→ A system call is a method of interacting with OS through high level programming language.

Types of System call :-

→ **Process Control :**

- end, abort
- load, execute
- Create Process, Terminate Process
- get Process attributes
- set process attributes
- wait for time
- Allocate & free memory

→ **File Manipulation :**

- Create a file
- Delete the file
- Opening & closing of file
- Read, write & reposition
- get file attributes
- set file attributes

→ **Device Manipulation :-**

- Request device
- Release device
- Read, write, reposition
- Get device attributes
- set device attributes
- logically attach or detach device

→ Information Maintance :

- Get Time or date
- Set Time or date
- Get system data
- Set system data
- Get process, file or device attributes
- Set process, file or device attributes

→ Communication :

- Create, delete, communication connection
- Set, receive messages
- Transform status information
- Attach or detach remote devices

Specific System Call :-

i) Wait() : when a process needs to wait for another process to complete its execution wait system call is used. or it is invoked.

ii) exec() : This system call is invoked when executable file in the context of an already running process replaces the older executable file. (Replace the centire process with same ID)

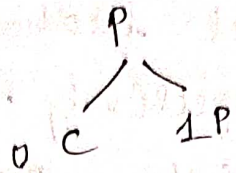
iii) Kill() : It is used by the OS to send a termination signal to a process that is executing.

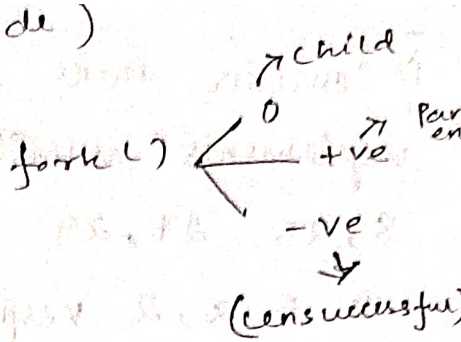
iv) exit() : It is used to terminate the program execution

v) fork() : fork system call is used to create a process that are the clone or copy of existing process.

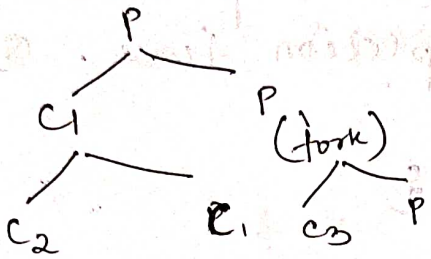
fork (will create a child class in a code)

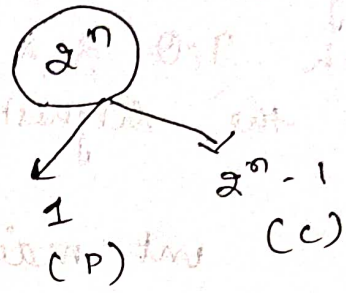
```
int main () {
  fork ();
  printf ("Hello world");
}
```





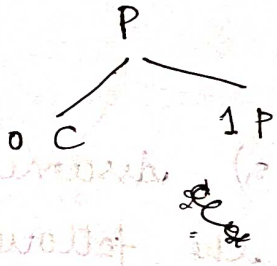
If 2 fork will there in a code





ques

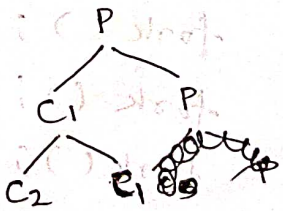
```
int main () {
Pid_t P = fork ();
  If (P = = -1)
  printf ("Error occured");
  else if (P = = 0)
      Pf ("This is child process");
  else
      Pf ("This is parent process"),
  return 0;
}
```



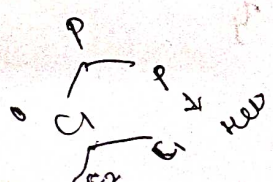
O/P : It's upto compiler a possible output
- This is C.P
- This is P.P.

ques

```
int main () {
  if (fork () = = 0)
    If (fork ())
    P.f ("Hello work");
    Exit (0)
}
```



O/P : Hello world



1) Consider there are 5 processes arriving in the system that implements multilevel feedback queue scheduling at 5, 14, 8, 25, 27, 29 respectively. There burst times are 2, 5, 2, 2 respectively. There are 3 ready queues ready queue 1 & ready queue 2 implement R.R with T.Q 2 & 5. RQ3 implements FCFS. which process has the highest completion time.

$Q_1 > Q_2 > Q_3$

2)

```
int main() {
    fork();
    fork()
    pf("DOS");
    fork();
}
```

3) Assuming that system call forks never fails consider the following C programs P1 & P2 executed on a Linux system.

P1

```
int main() {
    fork();
    fork();
    fork();
    pf("Happy \n");
}
```

P2

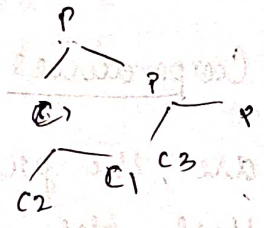
```
int main() {
    fork();
    pf("Happy \n");
    fork();
    pf("Happy \n")
    fork();
    pf("Happy \n"
}
```

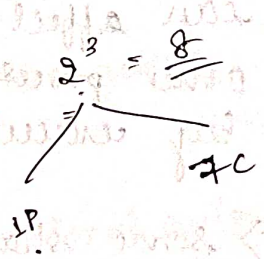
Statement 1: P1 displays Happy 8 times

S2 : P2 displays Happy 12 times

op1 : Both st1 & st2 true
op2 : Both statements are false
op3 : Stat1 is correct but st2 is false.
op4 : st1 is false & st2 is true.

4)
```
int main () {
    if (fork() && (!fork())) {
        if (fork() || fork())
        {
            fork();
        }
    }
    Pf ("2");
}
```





$2^3 = 8$

~~Synchronization~~ - Process

Synchronization

common
same variable
" resources
" memory
" code

Cooperative (common) | Independent

- Process synchronization is the condition of the execution of multiple processes to ensure that they access the shared resources in a controlled & predictable manner.

→ It aims to resolve the problem of race condition & other Synchronization issue in a concurrent system.

Diff^n between Cooperative & independent.

Cooperatives	Independent
- Those are the processes which can affect the execution of other process or can be effected by execution of other process	- Those are the process which cannot affect or cannot be affected by execution of other process
→ some shared state with other processes	→ No shared state b/w the processes.
→ It is non-deterministic which means it may have different result each time.	→ It is deterministic which means input alone determines the result.
→ It maybe irreproducable which means difficult to debug and the taste timing affects the results.	→ stop and start without adverse effect.
Eg: Two Programs sharing single terminal	→ Eg: Isolated program that calculate factorial sequences.

Q. Why synchronization is needed?

Ans) Let there are two process P1 & P2 (cooperative process) sharing the same variable named as shared and that variable is initialised to 5 if we want to execute cooperative processes in a uniprocessing & multiprogramming environment then

~~5~~ 4 → Global variable

int shared = ~~5~~

Uniprocessor

P1	P2
Local variable ← x = shared	y = shared y = 5
x=5 x++;	y--; y = 4
x=6 sleep (2);	sleep (3); → Pause
(Pause the P1 for 2 sec) shared = x;	shared = y;
shared = 6 → Terminate	shared = 4

CPU will run the P2 process

CS = Context switch

CS

CS

CPU → P2

after execution of 2nd line x++ it will switch to next program due to sleep() → this help to switch into new process CS → context switch.

→ If P1 is executed first followed by P2 then we are getting the output of the shared variable as 4.

→ similarly, if we reverse the order of execution the final output of the shared variable is 6. ~~So~~,

→ So, the output variable depends on the order of execution of the programs; this condition is called Race Condition.

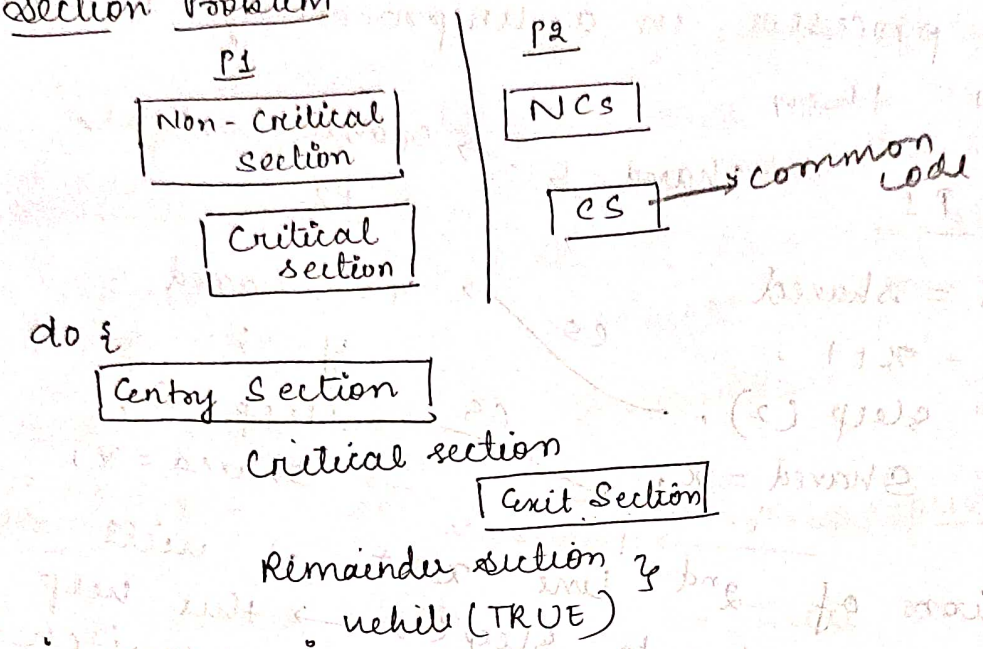
→ we need synchronization to avoid this Race condition.

Race Condition:

- Race condition occurs when multiple threads read & write the same variable which means they have ~~a~~ access to some shared data and they try to change it at the same time.

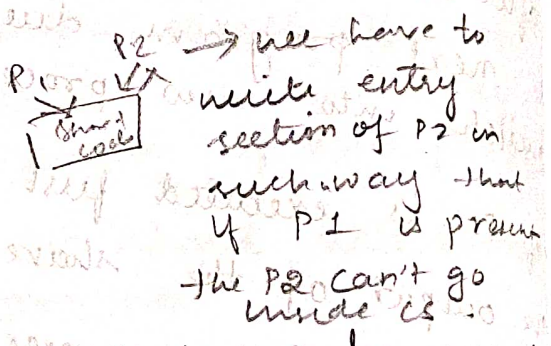
- In such a scenario threads are racing each other to modify the data.
- This condition is called Race Condition.

Critical Section Problem



Synchronisation Mechanism :-

1) Mutual exclusion } Primary
2) Progress
3) Bounded wait → secondary



Critical Section :

→ A critical section is a code segment that can be accessed by only one process at a time.

→ It contains shared variables that need to be synchronized to maintain the consistency of data variables.

wait () signal () :

→ The wait() system call mainly handles the entry to the critical section while the signal() system call handles the exit from the CS.

→ If we remove the critical section, we cannot guarantee the consistency of the end outcome after all the processes finished executing simultaneously.

Synchronisation Mechanism

→ for the process to be synchronized they should satisfy the following 3 condition.

i) Mutual exclusion:

If a process is executing in its critical section than no other process is allowed to execute in critical section.

ii) Progress:

If no process is executing in cs and other processes are waiting outside the cs than only those processes that are not executing in their Remaining section can participate in deciding which will enter the cs next & the selection cannot be postponed indefinitely.

iii) Bounded wait

A Bound must exist on the no. of times that other processes are allowed to enter their cs after a process has made a request to enter its cs.

Software solution to critical section Problem:-

- Dekker's Algorthim
- Peterson's Algorthim

Hardware solution to cs Problem:

- Compare & Swap
- exchange.

Dekker's Algorithm :-

Process Pi

```
do {
  flag [i] = true;
  while ( flag [j]) →
  {
    if (turn == j) turn = j
  }
    flag [i] = false;
  while (turn == j) ; → true
  if false flag [i] = true;
  }
  Critical section -
    turn == j;
    flag [i] = false;
  Remainder section,
}
while (true);
```

↓ then process Pj ceneeds

Process Pj

```
do {
  flag [j] = true;
  while (flag [i])
  {
    if (turn == i)
    {
      flag [j] = false;
      while (turn == i)
        flag [j] = true;
    }
  }
  Critical section.
  turn = = i;
  flag [j] = false;
  Remainder section
} while (true);
```

C1 : Pi → intrested
Pj → NI (not intrest)

C2 → Pi = NI
Pj = i

C3 : Pi – Intrested
Pj – intrested

→ It is a software based solution to the critical section problem.
→ Kits ~~two~~ ~~two~~ takes two process Pi & Pj

iii) This solution requires two data types to be shared by two process.
i) int-turn (it indicates whose turn it is to ~~enter~~ ii) enter its critical section.

Boolean flag :-

- It is a array variable having two variables with true or false & it is used to indicate if a process is ready to ~~enter~~ its critical section.
- It is restricted to only two processes.

Peterson's Algorithm :-

Process Pi

```
do {
    flag [i] = true;
    turn = j ;  — 5 sec
    while (flag[j] && turn == j) ;
    Critical section
    flag [i] = false ;
    Remainder section
} while (true) ;
```

Process Pj

```
do {
    flag [j] = true;
    turn = i;  — 6 sec
    while (flag [i] && turn == i);
    critical section
    flag [j] = false;
    Remainder section
} while (true);
```

C1: Pi - I
Pj - NI

C2: Pi - NI
Pj - I

C3: Pi - I
Pj - I
(in case 3 while loop will execute until it will false)

Compare and swap :-

int compare-and-swap (int * lock, int expected-val, int new-val)

= 0 1 e = 0 v = 1

{ int original-val = *lock; 1

if (*lock == expected-val) { 0 = 0

*lock = new-val; // lock = 1

(lock 1) } return original-val; → return-0

}

eg(1): initially lock = 0; # check for mutual exclusion

(2) while (compare-and-swap(&lock, 0, 1) != 0);

↳ false

(3) critical section. → Now P0 in CS

(4) Lock = 0;

(5) Remainder section.

P1 request to enter into CS when while loop - true then next statement cannot execute

Lock = 0

↳ C.S is empty

(if lock = 1 then P0 can enter into C.S)

~~(P0 execute instr (1) & instr (2)~~

~~Soln explanation~~

→ Assume two process are running P0 & P1. P0 executes instruction 1 and instruction 2. So it matches with the second parameter so it will assign the lock variable as 1

→ The whole fun' returns the main, initial value of lock which is 0 so it satisfied and execute the instruction (3) & enters the critical section.

→ At the same time if P1 wants to enter the critical section it will again execute the instr 1 & 2 but while now lock is updated to 1

It is not matched with the second parameter so it does not assign a new value to lock variable.

→ It will return the original value as 1 so it is not satisfied the condition & stock inside the instruction. This

is called busy waiting.

→ Now process P_1 cannot enter to the critical section because process P_0 has not completely executed the code inside the critical section.

Semaphore :-

- semaphore is a shared variable which is used to solve the critical section problem and to achieve process synchronization in the multiprocessing environment. (restrict access to shared resource)
- A semaphore is denoted by 'S' which uses two atmoic operation when a process wants to access to shared resou then it had to take permission from S.
  - i) ~~wait~~ wait operation (wait()) also called down function (P)
  - ii) signal operation (signal()) also called up func (V)
- wait operation of this semaphore is the entry section of code
- signal operation of this semaphore is the exit section of code.

Semaphore
- Binary (0, 1)
- counting ($-\infty, \infty$)

wait() for binary semaphore :-

wait()

when $S = 1$ $T = 0$

$S = 0$, CS is occupied

$S = 1$, CS is empty

```
P (semaphore S)
{
while(S <= 0);
S--;
}
```

if True then it is waiting → false

→ S-- = S = 0 ↳ it means CS is occupied ($P_0 \rightarrow CS$)

P_1 rqst and stucked (After P_0 exit) ($P_1 \rightarrow CS$) $S = 1$

(Critical sec)

signal() for binary semaphore :-

signal()

```
P (semaphore S)
{
S++;
}
```

(S)(+)

(P)

11001100

00

Binary Semaphore :

- The value of Binary semaphore can range btw 0 & 1.
- This binary semaphore is called MUTEX.
- when S is ~~less than~~ equal to 0 (S = 0) the critical section (CS) is occupied.
- If a process wants to enter a critical section it will execute the wait operation and can use the resource in the critical section.
- when that process completes its execution in critical section it will release the resource and the value of S will be incremented to inform that another process can request for the critical section.

Counting semaphore :-

- The value of counting semaphore can range over an unrestricted domain [$-\infty$ to ∞]
- It is used to control access to a resource that has multiple instances.

wait() for counting semaphore :-

wait()

If S = 4

P1

P (semaphore S)
{
S = S - 1; → S = ~~4~~ ~~3~~ ~~2~~ ~~1~~ ~~0~~ -1 -2
if (S < 0) → ~~false~~ ~~true~~ true
{
Put process in suspended list;
}
else
return;
}

P1 rqst to go CS
P2 rqst
P3 rqst
P4 rqst
P5 rqst → P5 will go to block list
P1 → CS
→ P0 → in CS
P2 → CS
P3 → CS
P4 → CS
P6 rqst

↳ P5 (Block list)
P6 (Block)

Signal() for counting semaphore :-

```
P (semaphore S)
{
    S = S + 1;
    If (S <= 0) {
        Select a process from suspended list & wake up();
    }
}
```

when P5 completed execution then P6 exit

$S = \cancel{1}\ \cancel{0}\ 1$

P5 → RQ
P6

True

↓ it means it will select from block list & put in RQ.

when S = 1, CS is empty, now P5 & P6 which is in RQ can rqst to go CS by executing wait() funⁿ.

→ If S = 5 then process can enter into the CS.

→ If S = -5 then 5 process are in suspend list or block list

Ques If initial value of semaphore is 15. & you perform 5P operation followed by 6V operations. what is the final value of semaphore.

Ans) Initial → S = 15

15 - 5 = 10

10 + 6 = 16

final value of semaphore is 16

Ques (11)

00110011

S = 1

T = 0

```
while (true) {
    wait (S);
    print ("1");
    print ("1");
    signal (T);
```

```
while (true) {
    wait (T);
    print ("0");
    print ("0");
    signal (S);
```

S	T
1	0
0	1
1	1
0	0

00

CS → 0

Ques (Imp)

P

```
while (1);
{
Printf('0');
Printf('0');
}
```

Q

```
while (1);
{
printf('1');
Prentf('1');
}
```

Let 'S' & 'T' are two binary semaphore then what is the initial value of S & T. So that the output will be 00110011 .... Utilise the wait() & signal() operation in proper place.

Ans) There are 4 possibilities

S	T
0	0
0	1
0	0
1	1

This case is not possible

It means as is occupied

As both are empty

for process P we use 'S' semaphore

for process Q we use 'T' semaphore.

S = 1 T = 0

```
while (1);
{
wait(S);
Printf('0');
Printf('0');
~~Signal(S)~~
}
Signal(T);
```

↳ T value incremented

T = 1

```
while (1);
{
wait(T);
Printf('1');
Printf('1');
}
Signal(S);
```

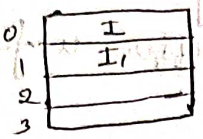
Classical Problem of Synchronization :-

i) Bounded Buffer / Producer Consumer Problem

Producer :

```
int count = 0;
void producer ()
{
  int I;
  while (true)
  {
    produce_item (I)
    while (count == n);
      Buffer [in] = I;
      in = (in + 1) mod n;
      count = count + 1;
```

Buffer



Consumer :

```
void consumer ()
{
  int C;
  while (true) {
    while (count == 0);
      C = Buffer (out);
      out = (out + 1) mod n;
      Count = Count - 1;
```

→ Producer-Consumer problem is a classical problem of synchronisation where there are two cooperative processes. One is producer process & second is consumer process.

→ The producer process produces one-by-one item & stores them in a buffer of size n.

→ The consumer process consumes one item from the buffer which the producer produce.

→ In this synchronization problem one shared variable is there which is called count. It determines the no. of items present in the buffer.

→ Producer process has one local variable that is (in) which is initialised to 0.

→ Similarly consumer process has one local variable i.e (out) which is also initialised to 0.

→ (In) identifies the next available location in the buffer

Solving the Bounded Buffer Problem using

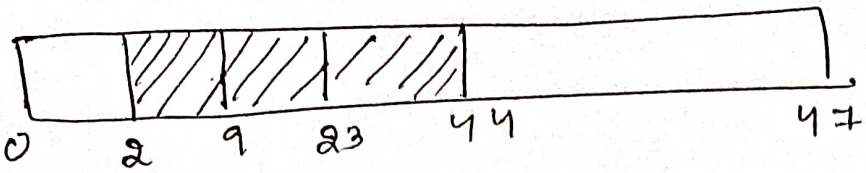
Explanation of Bounded Buffer problem :-

→ Identify the local & global Variable (shared variable)
→ Initialise the local & global variable
→ Check whether the Buffer is full in case the producer produces an item
→ Check whether the Buffer is empty in case the consumer consumes an item
→ Check for the Synchronisation problem if one of the process gets preempted
→ Solve the Synchronisation problem using semaphore.

Ques Consider three process all arriving at time 0 with execution of time 10, 20, 30 units respectively. each process spends the first 20% of execution time ~~during I~~ doing I/O, the next 70% of time doing computation & the last 10% of the time doing I/O again. The OS uses ~~sorted~~ shortest remaining time first algorthem & ~~se~~ schedule a new process either when the running process gets block or when the running process finishes its compute bus. Assume that all the I/O operation can be overlap as much as possible for what percentage of the CPU remains ideal.

Sol^n

	P_1	P_2	P_3
	10 (I/O)	20 (CPU)	30 (I/O)
20% of 10	2	4	6
70% of 10	7	14	21
10% of 10	1	2	3

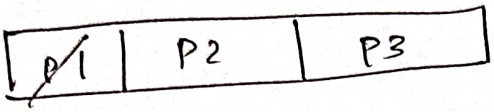


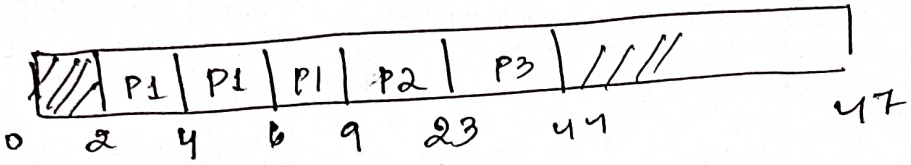
$2 + 3 = 5$

$\frac{5}{47} \times 100 = 10.6\%$ (Ans)

	Total	I/O	CPU
P1	10	2	~~8~~
P2	20	4	~~16~~
P3	30	6	~~24~~

RR





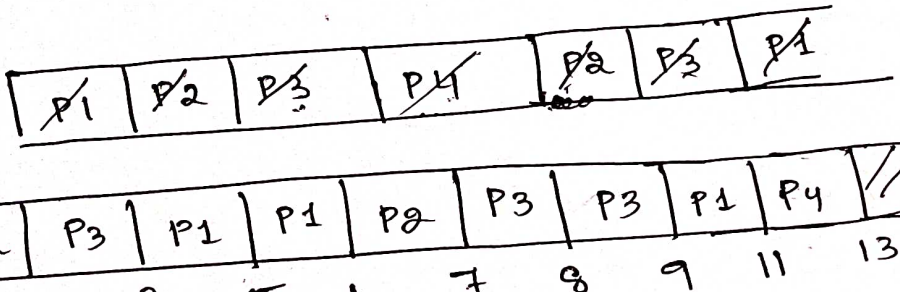
Ques Process no P1 to P4 A.T 0, 0, 2, 5 BT

	AT	CPU	I/O	CPU
P1	0	~~3~~ 0	~~2~~ 0	~~2~~ 0
P2	0	~~2~~ 0	~~4~~ ~~3~~ 0	~~1~~ 0
P3	2	~~1~~ 0	~~3~~ ~~1~~ 0	~~2~~ 0
P4	5	~~2~~ 0	2	~~1~~ 0

Always we see CPU time 1st

calculate the Average waiting time & Avg time

Ans) RR



→ 1st compare CPU time