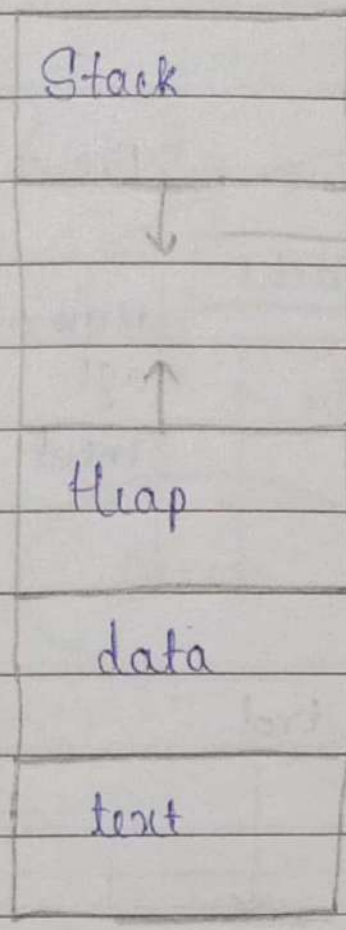
Operating Systems.

Address

Physical/ Virtual



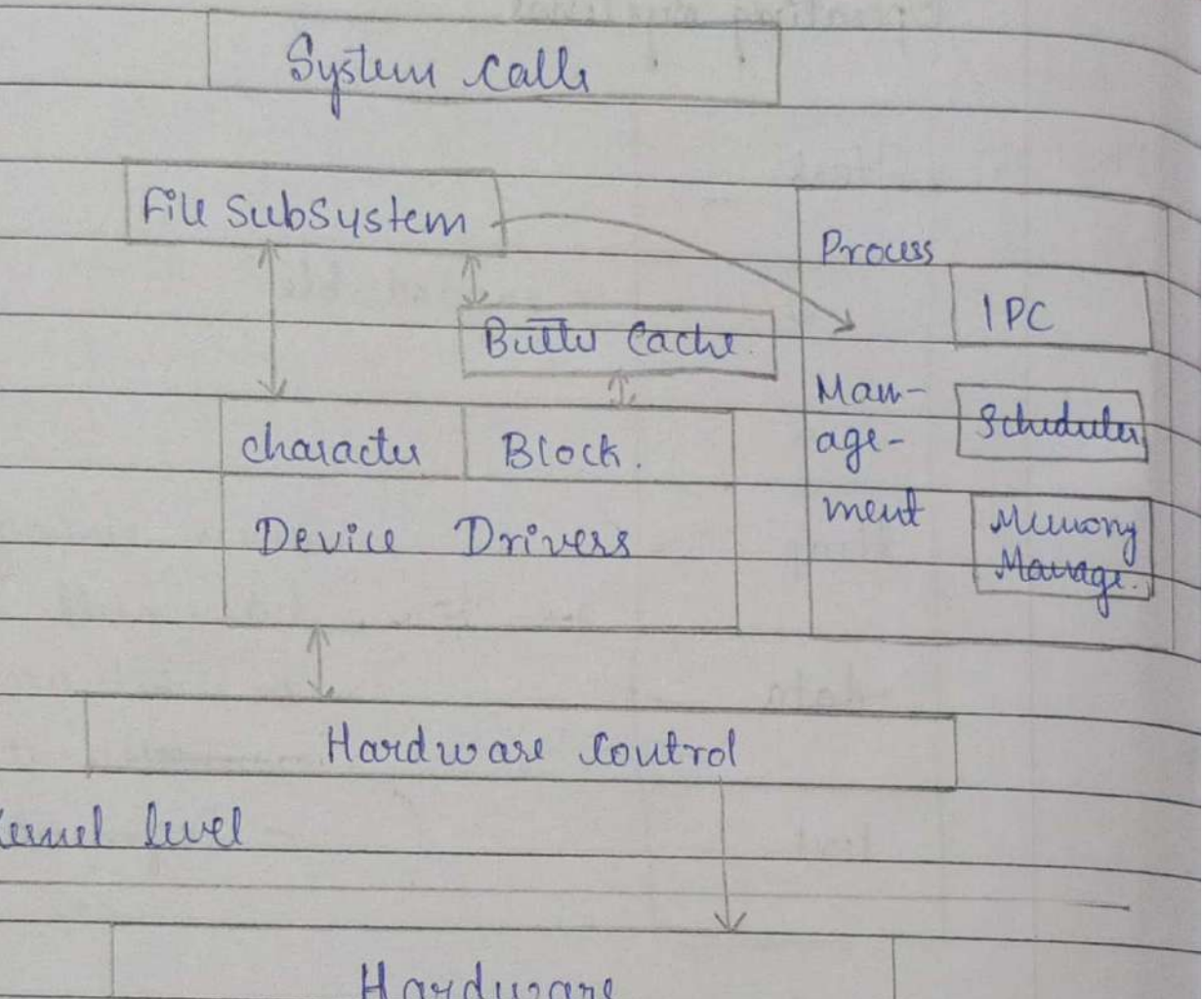
when string doesn't have null character or while accessing the non-allocated memory - segmentation fault.

Kernal acts a bridge b/w user & hardware
mode bit differs the user and kernal
0 - kernal code
1 - user code

Functions of OS

* Accounting
* process management
* device management
* Resource management
* Memory management
* file management

Block Diagram:



Process Management:

There are 5 states of process:

* New State

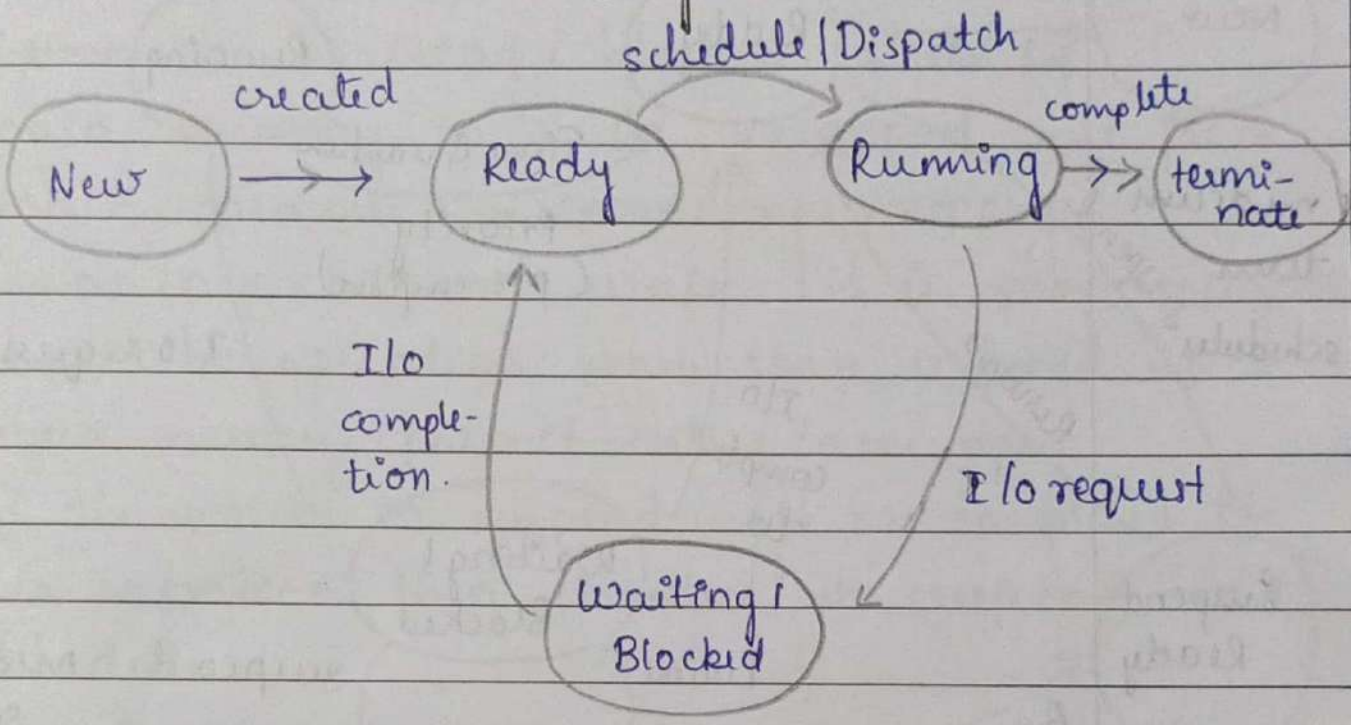
  Initiated for execution by user not by CPU

* Ready State

  All required resources are allocated it is maintained in queue. The scheduler will start the execution

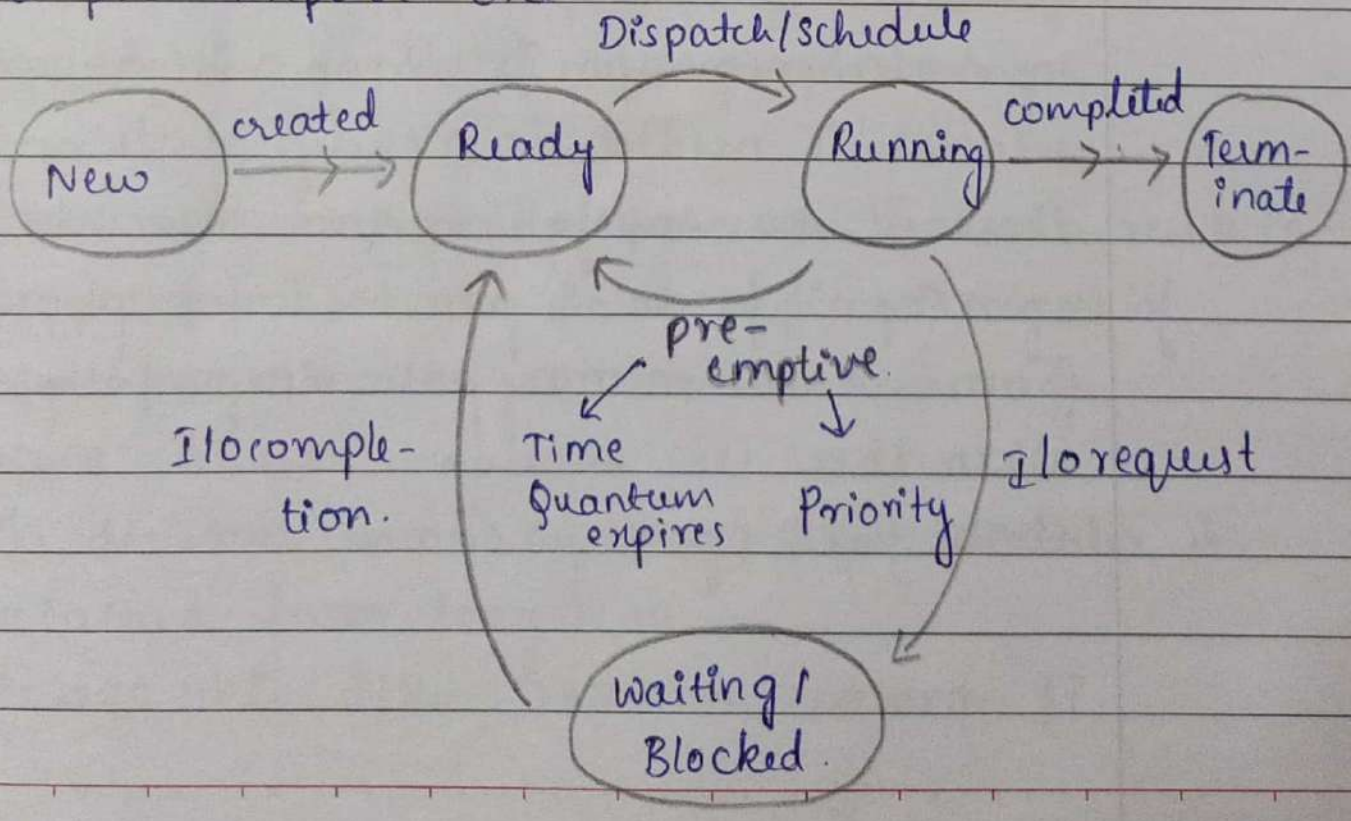
Uses and Kernel are modes of processor.
CPU works in user and kernel mode so that it can fulfill its goals i.e convenience and efficiency
Whenever uses applications are running then there is no much use of CPU (MS word etc) then CPU enters the user mode.
In kernel mode highly priviliged functions run
Kernel mode - system mode, priviliged mode, supervisor mode.

Process Transition State Diagram.



This is for non-pre-emption.
The 5 state diagram.

* The pre-emptive one



Schedulers:

The decision made by short term scheduler is retained for short time

The long-term scheduler picks the process from new state and places in ready queue. This decides the degree of multi-programming.

Maximum number of processes that can reside in ready queue at any point of time → degree of multi-programming

The decision made by long-term scheduler is retained from long time

+ Process Control Block

Pointers	Process state
Process number	
Program counter	
Registers	
Memory limits	
List of open files	

Process	Time. sec
P_1	5
P_2	4
P_3	3
P_4	4

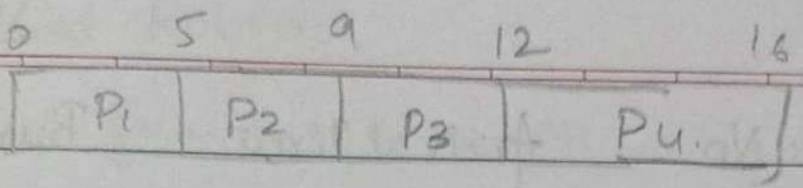
First Come First Serve 0 4 9 12

FCFS = $(P_1 + P_2 + P_3 + P_4)$

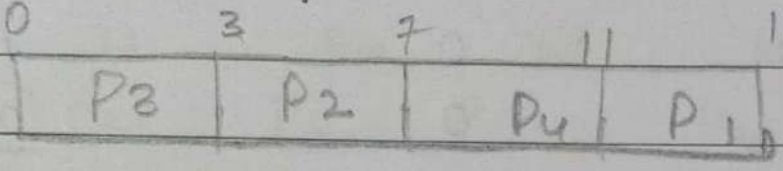
$$= \frac{}{4}$$

$= 6.5$.

PAGE NO.:
DATE: / /

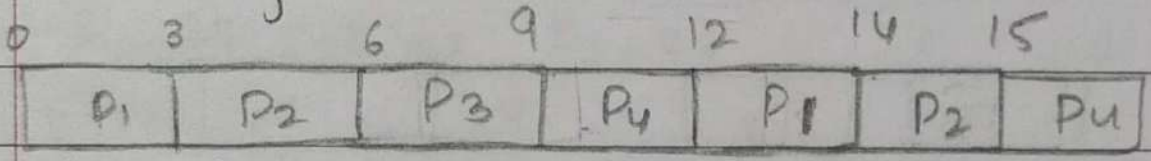


Shortest Job first



$= \frac{21}{4} = 5.25.s$

Time Quantum:- 3 sec.



Round Robin

$P_1 = 0 + 9$

$P_2 = 3 + 14 - 6 = 3 + 8 = 11$

$P_3 = 6$

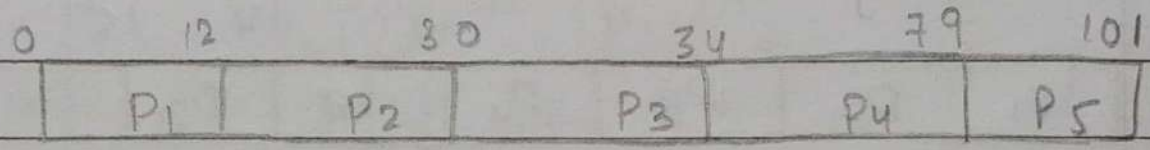
$P_4 = 9 + 15 - 12 = 9 + 3 = 12.$

Waiting $= \frac{38}{4} = 9.5$ // Context Switching PCB overwritten

Turn Around time:

Process No	Arrival Time	Burst time (execution)					
P1	0	12	7	2	-	-	
P2	0	18	13	8	3	-	
P3	0	04 - 4	-	-	-		
P4	0	45	40	35	30	25	20
P5	0	22	17	12	7	2	

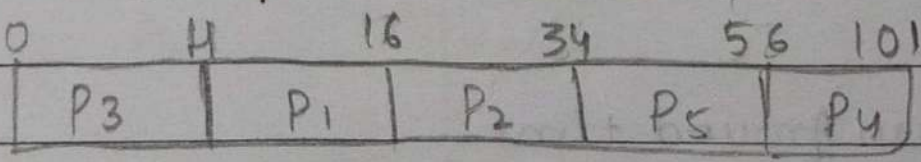
First Come First Serve:



$$FCFS = \frac{P1 + P2 + P3 + P4 + P5}{5}$$

$$= \frac{0 + 12 + 30 + 34 + 79}{5} = 31$$

Shortest Job first:

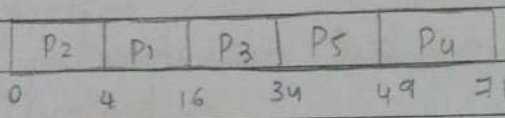


$$SJF = \frac{0 + 4 + 16 + 34 + 56}{5} = 22$$

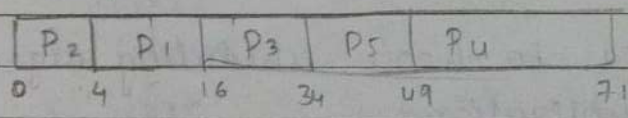
Q1)

Process	Arrival	Priority	Burst time	CT
P_1	0	3	12 (11)	16
P_2 X	0	2	04	04
P_3 X	5	1	18	34
P_4	10 ✓	6	22	71
P_5	25	4	15.	49
			71	

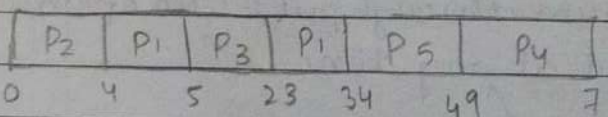
⇒ Apply SJF (non-preemptive).



⇒ Priority (non-preemptive)



⇒ Priority (pre-emptive)



TAT	WT
16	04
04	00
29	11
61	39
24	09
	$\frac{63}{5} = 12.6$

CT	TAT	WT
34	34	22
4	4	0
23	18	0
71	61	39
49	24	9
		$\frac{70}{5} = 14$

Response time

P_1	4	4	4
P_2	0	0	0
P_3	16	16	5
P_4	49	49	34
P_5	34	34	49
	$\frac{\Sigma}{5} = 20.6$	$\frac{\Sigma}{5} = 20.6$	$\frac{92}{5} = 18.4$
	↓	↓	
	SJF nonpre-emptive	priority non-preemptive	priority pre-emptive.

priority

Process	Arrival	Priority	BT	CT	TAT	WT	RT_1	RT_2
P_1 ✓	0	3	12	34	34	22	4	4
P_2 ✓	0	2	04	04	04	00	0	0
P_3 ✓	5	1	18	23	18	00	5	16
P_4 ✓	10	6	22	61	51	29	39	39
P_5 ✓	25	4	05	39	14	09	34	34
						60	82	= 18.6
						5 =	5	
							= 16.4	non-preemptive

P_2	P_1	P_3	P_1	P_5	P_4

0 4 5 23 34 39 61

↓

preemptive

priority non-preemptive

P_2	P_1	P_3	P_5	P_4

0 4 16 34 39 61

Exec functions:-

Exec (it is used to create but the text or source code will replaced by the program defined in the call).

fork
↓
call
↓
child

Variants:
exec l(list of variable)
exec p(path variable) → if some other location
exec v(vector (base add of arg vector), path)
exec e(ipecific environment)

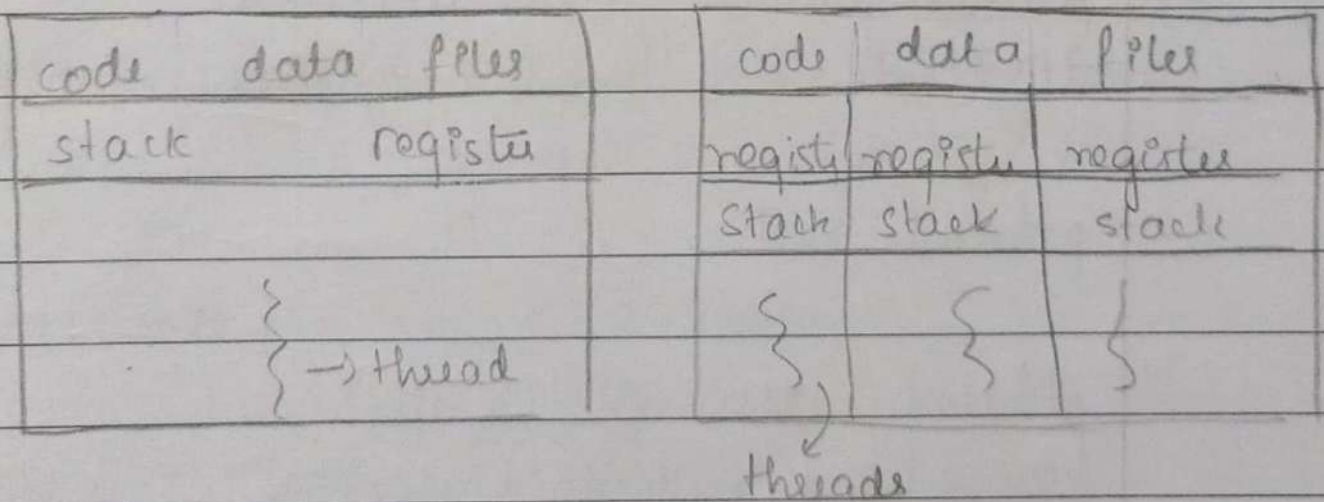
l - list of individual variable
path variable - where all software path are stored (environment variable)
echo $path - list of path.
l ↔ v

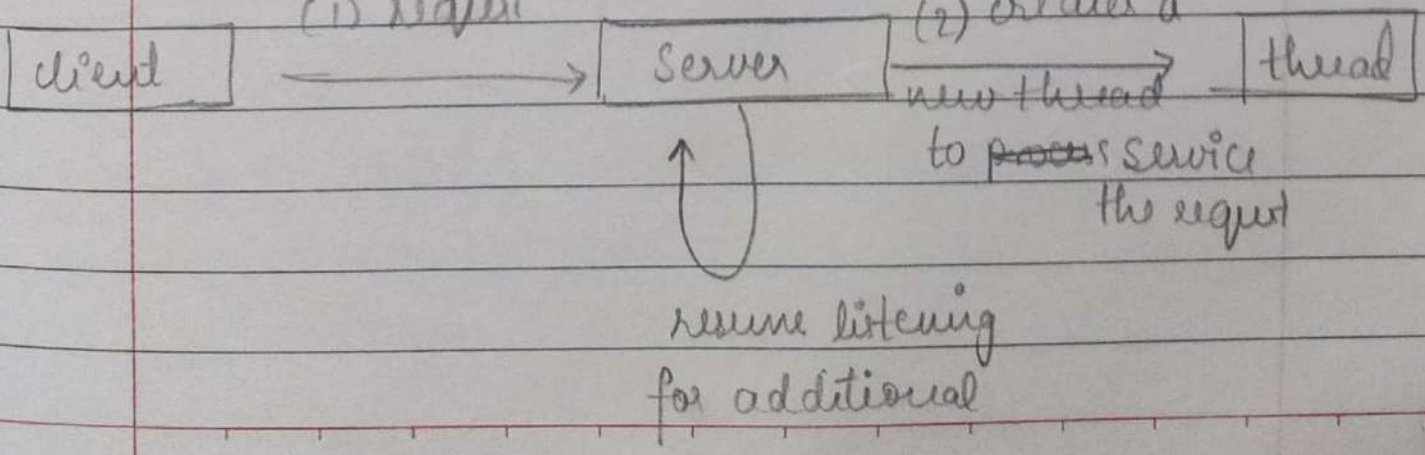
Threads → register set
→ program counter
→ stack
→ thread id
uniquely

↓

code section, data section and other resource like open files and signals

It is a basic unit of CPU utilization.
It is a light weight process.





DATE : / / 6

fork()	vfork()
→ Diff add space	→ Same add space
→ Simultaneous exe	→ Child suspends parent until it executes
→ Changes are not reflected	→ Change are reflected
→ Copy-on write	→ No copy on write.

Working of wait

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>

Int main()
{
    pid_t cpid;
    if (fork==0)
        exit(0);

    else
    {   cpid = wait(NULL);
        pf(" Parent pid = %d \n ", getpid());
        pf("child pid = %d \n", cpid);
    }

    return 0;
}
```

Deadlock Avoidance

PAGE NO.:
DATE: / /

Deadlock avoidance

(i) Ensure the system is always in safe state

Safe state → it is a state where we allocate the resources to the processes in such a way that there is no deadlock created

i.e. the system should allocate a resource to the process that requested for it only if its gonna stay in safe state after allocation even if the resource is available.

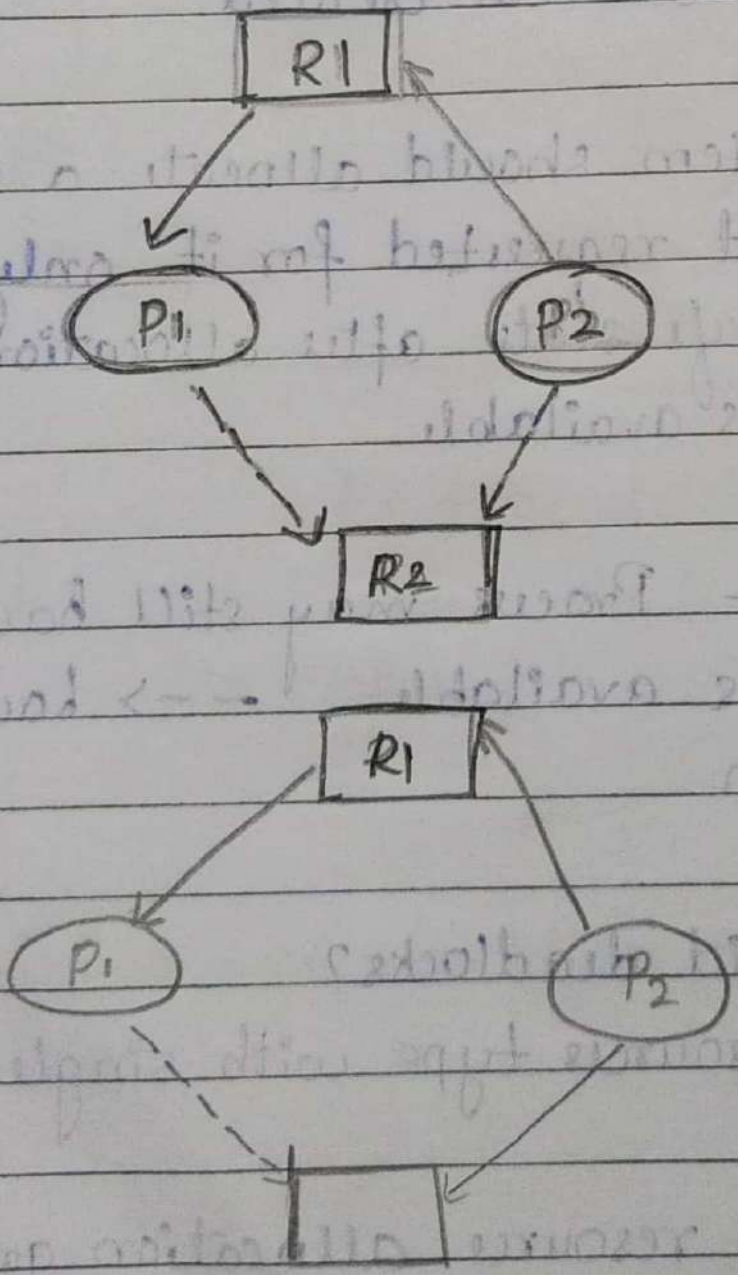
Drawback:- Process may still have to wait if the resource is available --> Lower Resource utilization.

How to avoid deadlocks?
→ For resources type with single instance

Using the resource allocation graph:
(i) Types Of Edges:-
- Assignment edge → Resources already allocated to a process

- Request Edge: Resources being requested by processes.
- Claims: Resources may be requested

NOTE: Claim edge may become a request edge later



DATE : / /

Resource - Allocation Graph Algorithm:

Grant resource only when following condition is met

→ Request edge $P_i \rightarrow R_j$ to assignment edge $R_j \rightarrow P_i$ does not create a cycle

→ Uses cycle detection algorithm

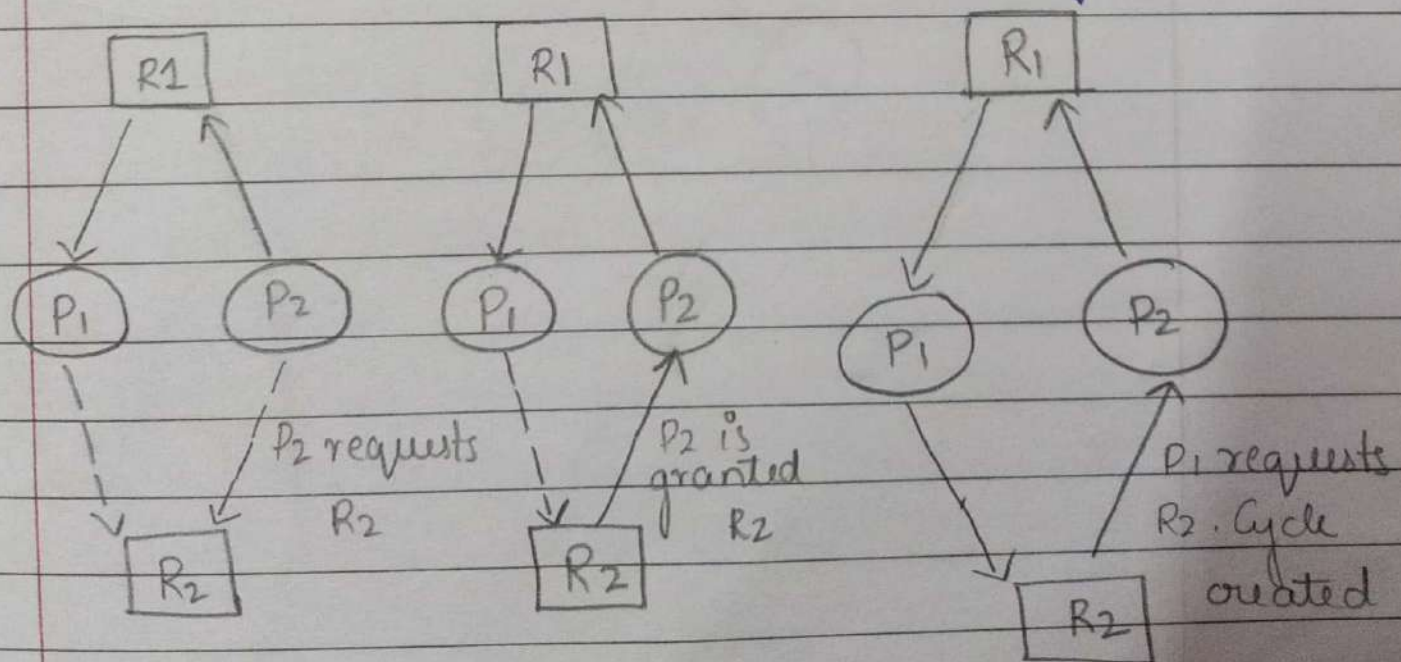
→ No cycle exists means no potential deadlock Allocate the resource

Ex → P_2 requests R_2

→ If this is granted it will create a cycle

- P_1 requests R_2 and P_2 requests R_1

→ P_1's request for R_2 will not be granted



DATE : / /

Overview of Banker's Algorithm:

Resource Allocation Algorithm
→ Does not work for system with multiple instances of same resource type

Bankers Algorithm
→ Suitable for systems with multiple instances of the same resource type
→ Suitable in a banking environment
Ensures bank must manage cash in such a way that it satisfies the needs of all customers

Threads

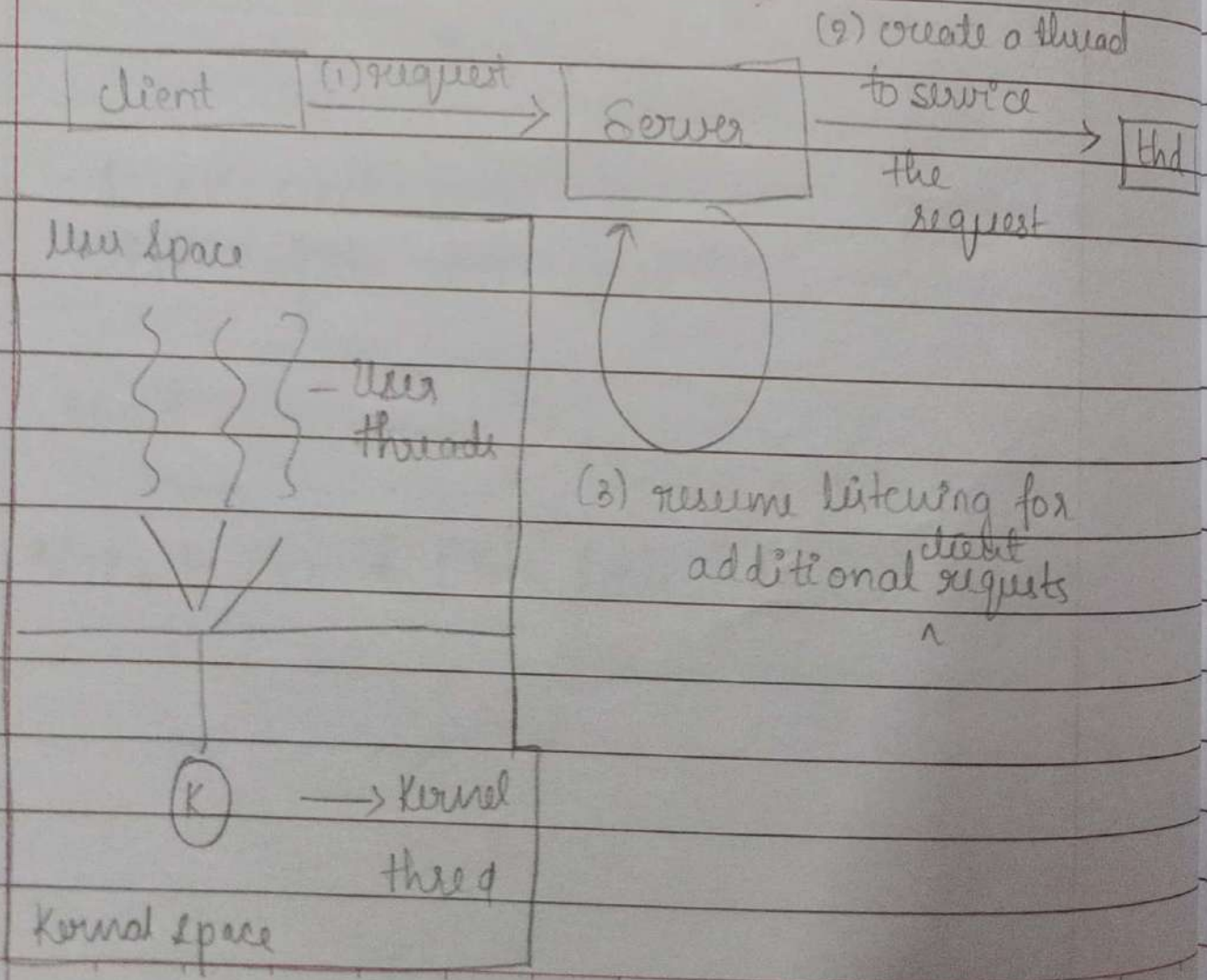
→ basic unit of CPU utilization

Process

→ basic unit of work.

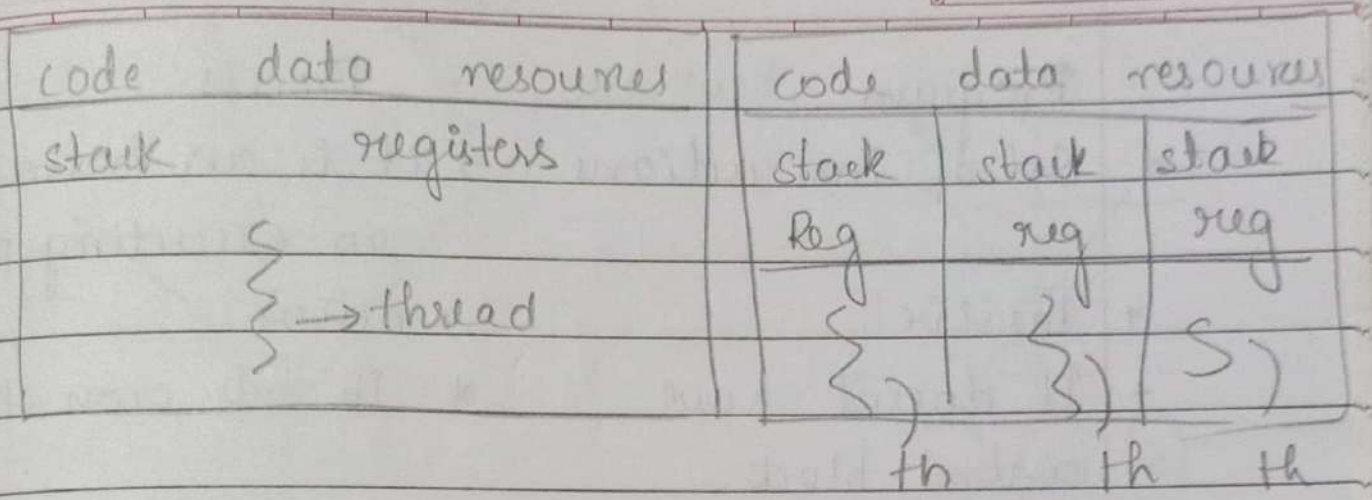
Thread → stack
→ ID
→ PC
→ Register set

In common → code / text, data, open files
signals



PAGE NO.:

DATE : / /



Users-threads	Kernel-threads
* Easier and faster to create than kernel-level threads	* slower than users-level threads, slower to create & manage
* Can run on any OS	* Cannot run on any OS
* No kernel mode privileges required for thread-switching	
* Entire process is blocked if one user-level thread perform blocking operations	* if one kernel thread perform blocking op'n then another thread can cont
* implemented by users	*implemented by OS
* OS doesn't recognize	* OS recognizes
* context switch time is less	* more

Program	Process
* Set of instructions	* it is an instance of an executing program
* Passive	* active
* It doesn't have control block	* it has own PCB
* It is a file residing on disk	* program loaded in main memory

Process	thread
* heavyweight process	* lightweight process.
* program in execution	* Segment of process
* unit of work	* unit of CPU utilization
* more time to terminate	* less.
* more time to create	* less.
* " " " CS	* less.
* less efficient in terms of communication	* more efficient
* more resources	* less.

Codes for chapter-2

DATE : / /

// working of wait

```
#include <stdio.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <unistd.h>
int main()
{   if (fork()==0)
        pf("HC: hello from child\n");
    else
    {  pf("HP: hello from parent\n");
       wait(null);
       pf("CT: child has terminated\n");
    }
    pf("Bye\n");
    return 0;
}
```

o/p: HP
HC
CT
Bye

// wait() working

```
#include < stdio.h>
# include <sys/ wait.h>
# include <unistd.h>
# include <stdlib.h>
int main()
{
    pid_t cpid;
    if (fork() ==0)
        exit(0);
    else
        cpid= wait (NULL);
    pf(" Parent PID = %d \n", getpid());
    pf(" child PID = %d\n", cpid );
    return 0;
}
```

o/p: Parent PID = 1234
child PID = = 4678

The cat command is used
→ Display, Copy, Combine, create text files.

* Create a file
  cat > filename
  /* Enter the info then press ctrl+d to come to the command prompt */

* View file
  cat filename.

* Combine two file.
  cat filename1 filename2 > newfilename

* Deleting file
  rm filename.
  // No confirmation

* Moving files.
* mv <filename> <newfile location).
  /* Permission denied : It needs super user permission */.

* sudo mv filename location.
  /* Then it asks for password */
  Password is retained for 15 min / terminal

* ls ⇒ list all files in current directory

* ls -R
⇒ listing subdirectories

* ls -al
⇒ getting detailed information abt files

File type & Access Permission | Memory Block | Owner/ Creator | User Group

File Size (in bytes) | Date and time of creation

File/directory name.

* Hidden items in linux begin with "." period symbol.
To view hidden files.
ls -a command.

Creating and viewing files

* Renaming filename.
mv filename newfilename.

* Creating directory
mkdir songs
/* Blue color - directory
White color - files.
This creates a sub-directory in present working directory (usually home)

* Create directory at new location.
mkdir </path/new_directory_name>

* Creating multiple directory
mkdir <dir1> <dir2> <dir3>

* Remove directory
rmdir <directory_name>
/* No sub-directories or files in the directory to be deleted */

* Renaming directory
mv <direct_name> <new_dire_name>

The 'Man' Command
Man stands for manual
The refernce book for Linux.

=> man <command>
This command gives manual on that command

The History Command.
Shows commands you have used in past

Clear command
=> clear

Copy pasting Command
Ctrl + C
Ctrl + Shift + V
- OR -
Shift + Insert
- OR -
Paste option

* pwd
present working directory
Absolute path - Compute path from root

Green - executable file.
↓
In windows (batch files).
sh - shlllscript.

* ls
list file
(with color distaiction)

* ls - l
long list of file.

drwr - directory file // Blue
- regular file.

other user.
-rw-[r]--r
read write
creator

link
count
/ \
Soft hard

User group others

* ls -a
hidden files.
display all including hidden files
.
..

* mkdir folder.

* cd directory name.
change directory

. - current diretory
.. - parent directory

* ls -la

* rm -f

* To create file
touch filename

* print "content" > file1.
Redirecting the content to file.

* echo "content" > file.
writing to file.
* cat file
display.

* vi filename.

* cp - copy the file.
mv - mov the file

cp src dist
mv src dest

* ps - lists all process that are executing

* gedit &

* ps -aux

* wc - lines, bytes

* chmod 777 filename.
// for permission change rwx
000 - 111

PAGE NO.:
DATE: / /

Allocated

	T1	T2
P1	1	1
P2	1	1
P3	2	1

Max

	T1	T2
P1	2	3
P2	3	2
P3	4	4

Available

T1	T2
1	1

T1 = total = 5 units
T2 = total = 4 units

Work = Available = (1, 1)

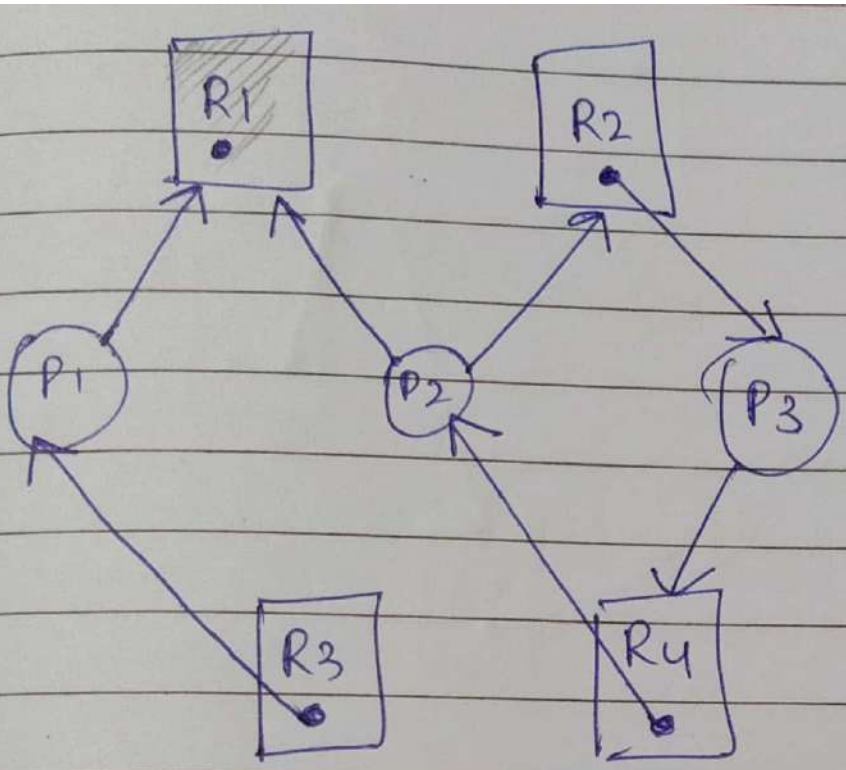
finish

F	F	F
0	1	2

Need

	T1	T2
P1	1	2
P2	2	1
P3	2	3

No unsafe



→ Non-preemptive resources → met

→ Mutual exclusion - met

→ P2 is holding R4 and requesting for R1 & R2 so hold and wait condition

P1 has R3 request R1

P3 has R2 request R4

→ P3 → P2 , P2 → P3

deadlock can exist 2-process deadlock