Aim - To implement general purpose utility commands in linux.

CAL OUTPUT

1)

January 2016

Su	Mo	Tu	We	Th	Fr	Sa
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

2)

2000

JANUARY

S	M	T	W	T	F	S
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

FEBRUARY

S	M	T	W	T	F	S
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28					

MARCH

S	M	T	W	T	F	S
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

APRIL

S	M	T	W	TH	F	S
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30						

MAY

S	M	T	W	TH	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

JUNE

S	M	T	W	TH	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	

JULY

S	M	T	W	TH	F	S
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

AUGUST

S	M	T	W	TH	F	S
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

SEPTEMBER

S	M	T	W	TH	F	S
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

OCTOBER

S	M	T	W	TH	F	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

NOVEMBER

S	M	T	W	TH	F	S
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

DECEMBER

S	M	T	W	TH	F	S
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

Date :

EXPERIMENT NO 1 (A)

Aim -

To implement general purpose utility commands in linux.

Commands -

cal - Display a calender

Syntax - cal [-smjy13] [[day] month] year]

Options -

-1	Display single month output. (This is default)
-3	Display prev / current / next month output.
-s	Display Sunday as the first day of the week
-m	Display Monday as the first day of the week
-j	Display Julian dates (days one-based, numbered from January 1)
-y	Display a calender for the current year.
-V	Display version information and exit.

Example 1 computer@computer-desktop:~$ cal

Example 2 To view the calender of a specific year, provide the year as an argument.
computer@computer-desktop:~$ cal 2000

Example 3 To view the calendar of a specific month, provide the month and the year as argument.
computer@computer-desktop:~$ cal 1 1990

Page No.

3)

JANUARY 1990

S	M	T	W	TH	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

4)

JULY

S	M	T	W	TH	F	S
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

AUGUST

S	M	T	W	TH	F	S
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

SEPTEMBER

S	M	T	W	TH	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

DATE OUTPUT

Tuesday 13 September 2022 2:39:29 PM IST

BC OUTPUT

```
computer@computer-desktop: ~$ bc
bc 1.06.95
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006 Free
    Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type 'warranty'.
1 + 10
11
5 * 3
15
2^10
1024
10^(5+4)
90
10^5+4
54
```

Date :

Example 4 The option -3 displays the calendar of the specified month along with its previous & next month.

computer@computer-desktop : ~$ cal -3 8 1947

date - Print or set the system date and time

Syntax date

Ex computer@computer-desktop: ~$ date

bc - Basic calculator
- An arbitrary precision calculator language
- use quit to exit.

Example computer@computer-desktop : ~$ bc

echo - Display a line of text.

Ex - 1) computer@computer-desktop : $ echo "Fleiss Linux Resources"

Example - 2) computer@computer-desktop : $ echo -e "Fleiss \vLinux \Resources"

Example 3) computer@computer-desktop : $ os = linux

computer@computer-desktop : $ echo "I use $ os"

printf - Format and print data

Example computer@computer-desktop : $ printf "Fleiss \vLinux \vResources \n"

Page No.

ECHO OUTPUT

1) Eleiss linux Resources
2) Eleiss
   Linux
   Resources
3) I use linux

PRINTF OUTPUT

Eleiss
Linux
Resources

SEQ OUTPUT

1)
1
2
3
4
5
6
7
8
9
10

2)
1
3
5
7
9
11
13
15
17
19

3)
1
4
7
10
13
16
19

FACTOR OUTPUT

1) 100 : 2 2 5 5
2) 200 : 2 2 2 5 5

Date :

seq - Print a sequence of numbers

Syntax
seq [OPTION] ... LAST
seq [OPTION] ... FIRST LAST
seq [OPTION] ... FIRST INCREMENT LAST

Description - Print numbers from FIRST to LAST, in steps of INCREMENT.
- If FIRST or INCREMENT is omitted, it defaults to 1.

Example
1 - seq 1 10
2 - seq 1 2 20
3 - seq 1 3 20

Factor - Factor numbers

Syntax Factor [NUMBER]

Example computer@computer-desktop: ~$ Factor 100
Example computer@computer-desktop: ~$ Factor 200

mkdir - Short form of "make directory", mkdir is used to create directories on a file system.

Syntax mkdir [option] Directory

Example mkdir OS-PRACT

Page No.

CLEAR OUTPUT

```
computer@computer: ~
(base) computer@computer: ~$
```

LS OUTPUT

```
computer@computer: ~
(base) computer@computer: ~$ ls

aa.jpg          Documents       NetBeansProjects
abc.c           Downloads       ns-allinone-2.35
anaconda3       Music           ns_simple.tcl
Android         Pictures        snap
Desktop         pt              Videos

(base) computer@computer: ~$
```

Date :

cd - The cd command, which stands for "change directory", changes the shell's current working directory

cd ~ : Returns you to your login directory.

Example: cd OS_PRACT

clear - clear the screen

Syntax: clear

ls - lists the contents of a directory

Syntax: ls [-l] [-a]

-l → list your files in long format, which contains a lots of useful information

-a → list all files

Example: ls

cp - cp stand for copy
- Copies files and directories.
- If the command contain two files names, then it copy the content of 1st file to the 2nd file. If the 2nd file doesn't exist, then first it creates one and content is copied to it. But if it existed then it is simply overwritten without any warning.

Page No.

MAN OUTPUT

DATE(1) User Commands DATE(1)

NAME

date - print or set the system date and time

SYNOPSIS

date [OPTION]... [+FORMAT]

date [-u|--utc|--universal] [MMDDhhmm[[CC]YY][.ss]]

DISCRIPTION

Display the current time in the given FORMAT, or set the system date.

CAT OUTPUT

1) Unix
Linux
Solaris
AIX
Linux
HPUX

2) 20
19
5
49
200

Date :

Syntax cp Source - Filename Destination - filename

Example computer@computer-desktop:~$ cpa.txt b.txt

rm - The rm command removes (deletes) files or directories

Syntax rm Filename

Example computer@computer-desktop:~$ rm OS-PRACT

man - The reference manual. Id displays the documentation for a command.

Syntax man command-name

Example man date

cat

- The cat (short for "concatenate") command is one of the most frequently used command in Linux / Unix like OS.
Syntax — cat Filename1 Filename2

- cat command allows us to create single or multiple files, view contain of file, concatenate files and redirect o/p in terminal or files.
Syntax - cat filename.

Page No.

3)
20
19
5
49
200
25
18
5
48
200

VI OUTPUT

computer@computer: ~

```
#include<stdio.h>
main()
{
printf("HI");
}
~
~
~
~
~
~
:wq
```

Date :

Example - 1) cat Hello.txt
2) cat Data.txt
3) cat File1.txt File2.txt

vi - Create a new file if it already does not exist, otherwise opens existing file.
- To come in insert mode you simply type i or press insert key. To get out of insert mode, press the Esc key, which will put you back into command mode.
- The command to save the content of the editor is :w. The command to quit out of vi is :q

Syntax vi filename

Example computer@computer:~$ vi hello.c

sort - sort command is used to sort a file, arranging the records in a particular order. By default, the sort command sorts files assuming the content are ascii. Using options in sort command, it can also be used to sort numerically

Syntax $ sort Filename

- All records are sorted alphabetically

Page No

REMOVE DUPLICATION OUTPUT

1)
AIX
HPUX
Linux
Linux
Solaris
Unix

2)
AIX
HPUX
Linux
Solaris
Unix

3)
5
19
20
49
200

4)
200
49
20
19
5

5)
5
5
18
19
20
26
48
49
200
200

6)
5
18
19
20
25
48
49
200

CUT COMMAND OUTPUT

1)
a
P
S

2)
cat
CP
LS

Date :

Remove duplication - Sort removes the duplicates using the -u option.

Syntax : $ sort -u filename

-n option can sort the decimal no. as well.

Syntax $ sort -n filename.

'r' option does a reverse sort.

Syntax $ sort -nr filename.

Example
1) sort Hello.txt
2) sort -u Hello.txt
3) sort -n Data.txt
4) sort -nr Data.txt
5) sort -n file1.txt file2.txt
6) sort -nu file1.txt file2.txt

CUT Commands - Linux command cut is used for text processing. You can use this command to extract portion of text from a file by selecting columns.

1) Select Column of Character → To extract only a desired column from a file use -c option.
2) Select column of Character using Range → Range of characters can also be extracted from a file by specifying start and end position delimited with '-'.

Page No.

GREP COMMAND OUTPUT

Linux

PWD OUTPUT

output : /home/computer

Date :

Example 1) To display 2nd character from each line of a file text.txt

$ cut -c2 test.txt

Example 2) To following example extract first 3 character of each line from a file called test.txt

cut -c1 test.txt

Grep Commands - Grep stands for Global search for Regular Expression and Print

- Grep is frequently used command in Unix (or Linux)
- Most of u use grep just for finding the words in a file

Syntax: grep " literal_string" Filename

Output: $ grep Linux Hello.txt

pwd - The pwd command allows you to print the current working directory on your terminal.

Example: pwd

diff, comm, and cmp - The diff, comm and cmp commands compare differences and are some of the most useful Linux commands you must know.

Page No.

IFCONFIG OUTPUT

```
computer@computer: ~
(base) computer@computer: ~ $ ifconfig
enO2: Flags = 4163<UP, BROADCAST, RUNNING, MULTICAST> mtu 1500
      inet 192.168.176.31 netmask 255.255.255.0
           broadcast 192.168.176.255
      inet6 fe80::6033:e273:9df8:2148 prefixlen 64
           scopeid 0x20<link>
      ether e0:be:03:69:58:2b txqueuelen 1000 (Ethernet)
      RX packets 379050 bytes 408025694 (408.0 MB)
      RX errors 0 dropped 0 overruns 0 frame 0
      TX packets 228533 bytes 47051235 (47.0 MB)
      TX errors 0 dropped 0 overruns 0 carrier 0
           collisions 0
      devices interrupt 19 memory 0x4f900000-4f920000
```

Conclusion -

Thus the general purpose utility command in Linux is executed successfully

Date :

Syntax
a) diff <file 1> <file 2>
b) cmp <file 1> <file 2>
c) comm <file 1> <file 2>

ifconfig - The ifconfig command will give you the list of all the network interfaces along with the IP address, MAC addresses and other information about the interface.

Syntax ifconfig

Example ifconfig

Conclusion

Thus the general purpose utility command in Linux is executed successfully.

(A) 08/09/22

Page No.

EXPERIMENT NO 1(B)

Aim

Write a program to implement following system call

1) fork()
2) wait()
3) sleep()
4) write()
5) read()

Date :

EXPERIMENT NO 1 (B)

Aim

Write a program to implement following system call.

1) fork()
2) wait()
3) sleep()
4) write()
5) read()

Theory

A system call is used by application (user) program to request service from the O.S.

fork()

- fork system call use for creates a new process, which is called child process, which runs concurrently with process (which) process called system call fork) and this process is called parent process.
- After a new child process created, both processes will execute the next instruction following the fork() system call.
- A child process use same pc(program counter), same CPU register, same open files which use in parent progress.
- It take no parameter and return integer value

Page No

Date :

- Below are different values returned by fork():
  a) Negative value - creation of child process was unsuccessful.
  b) Zero - Returned to the newly created child process.
  c) Positive value - Returned to parent or caller. The value contain process ID of newly created child process.
- The child process can start execution before the parent process or vice-versa.

getpid()

- The getpid system call returns the process ID of the calling process.

getppid()

- The getppid system call returns parent process ID of the calling process.

wait()

- wait() call blocks the calling process until one of its child processes exits or a signal is received.
- After child process terminates, parent continues its execution after wait system call instruction.

sleep()

- The sleep system call is used to take a time value as a parameter, specifying the

Page No.

Date :

minimum amount of time that the process is to sleep before resuming execution

Syntax - sleep(time);

exit()

- The exist system call is used to terminates a process either normally or abnormally.

write()

- write() can be used to write to any file in the system
- Syntax :

```
#include <unistd.h>
write(int fd, data, size);
```

- The first parameter (fd) is the file descriptor where you want to write
- The data that is to be written is specified in the second parameter.
- Finally, the third parameter is the total byte that are to be written.

read()

- The use of read() system call is to read from a file descriptor.
- Syntax -

```
#include<unistd.h>
read(int fd, buffer, size);
```

Page No.

Output 1 -

Hello World

Output 2 -

Hello World
Hello World

Date :

- The first parameter is the file descriptor.
- The second parameter is the buffer where the read data will be saved.
- Lastly, the third parameter is the no of bytes that you want to read.

Procedure for Execution

1. Open gedit
2. Type the program.
3. Save the file with .c extension
4. Compile the program using gcc Filename.c
5. Run the program by typing ./a.out

Program 1

```
#include<stdio.h>
int main()
 {
    printf(" Hello world\n");
    return (0);
 }
```

Program 2

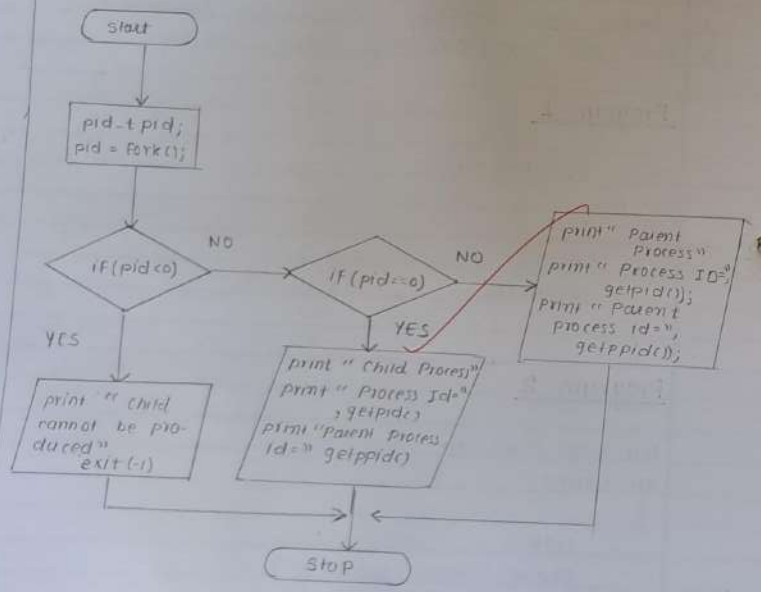
```
#include <stdio.h>
int main()
{
    fork();
    printf(" Hello world\n");
    return(0);
}
```

Page No.

Output 3 -

Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World

Flow Chart (4)



Date :

Program 3

```
#include<stdio.h>
int main()
{
    fork();
    fork();
    fork();
    printf("Hello world\n");
    return(0);
}
```

Program 4

```
#include<stdio.h>
#include<sys/types.h>
int main()
{
    pid_t pid;
    pid = fork();
    if(pid<0)
    {
        printf("\n Child cannot be produced");
        exit(-1);
    }
    else if(pid==0)
    {
        printf("\n Child Process");
        printf("\n Process ID = %d \n",getpid());
        printf("\n Parent Process id = %d \n", getppid());
    }
```

Page No.

Output 4 -

```
Parent process
Process ID = 10450
Parent process ID = 10443
Child Process
Process ID = 10451
Parent process ID = 10450
```

Output 5

```
I am Child process
I am Parent process
```

Date :

```
    else
    {
        printF("\n Parent Process");
        printf("\n Process ID = %d \n", getpid());
        printF("\n Parent Process id = %d \n", getppid());
    }
    return 0;
}
```

Program 5

```
#include<stdio.h>
#include<sys/types.h>
main()
{
    pid_t pid;
    pid = fork();
    if (pid<0)
    {
        printF("\n Child cannot be produced \n");
        exit(-1);
    }
    else if(pid==0)
    {
        printFC("\n I am Child process \n");
    }
    else
    {
        wait();
        printf("\n I am parent process \n");
    }
}
```

Page No

Owpw 6

I am Parent Process
I am Child Process

Date :

Program 6

```
#include<stdio.h>
#include<sys/types.h>
main()
{
    pid_t pid;
    pid = fork();
    if(pid<0)
    {
        printf("\n Child cannot be produced\n");
        exit(-1);
    }
    else if(pid==0)
    {
        wait();
        printf("\n I am Child Process\n");
    }
    else
    {
        printf("\n I am Parent Process\n");
    }
}
```

Program 7

```
#include<stdio.h>
#include<sys/types.h>
main()
```

Page No.

Output 7 -

I am Parent Process
I am Child Process

Owpw 8 -

hello

```
{
    Pid_t pid;
    pid = fork();
    if (pid < 0)
    {
        printf("\n Child cannot be produced \n");
        exit(-1)
    }
    else if (pid == 0)
    {
        sleep(1);
        printf(" I am Child Process \n");
    }
    else
    {
        printf("\n I am Parent class\n");
    }
}
```

Program 8

```
#include<unistd.h>
int main()
{
    write (1, "hello \n", 6);
    // 1 is the file descriptor, "hello \n" is
       the data, 6 is the count of character
       in data
}
```

Output 9 -

```
Enter some message
Welcome to OS
Welcome to OS.
```

Conclusion

Thus the fork(), wait(), sleep(), read() and write() system call is executed successfully.

Date :

Program 9

```
#include<unistd.h>
int main()
{
    char buff[20];
    write(1, "Enter some message", 18);
    read(0, buff, 15);   // read 10 bytes, store in
                            buffer (buff)
    write(1, buff, 15);   // print 10 bytes from the
                            buffer on the screen
}
```

Conclusion

Thus, the fork(), wait(), sleep(), read(), and write() system call is executed successfully.

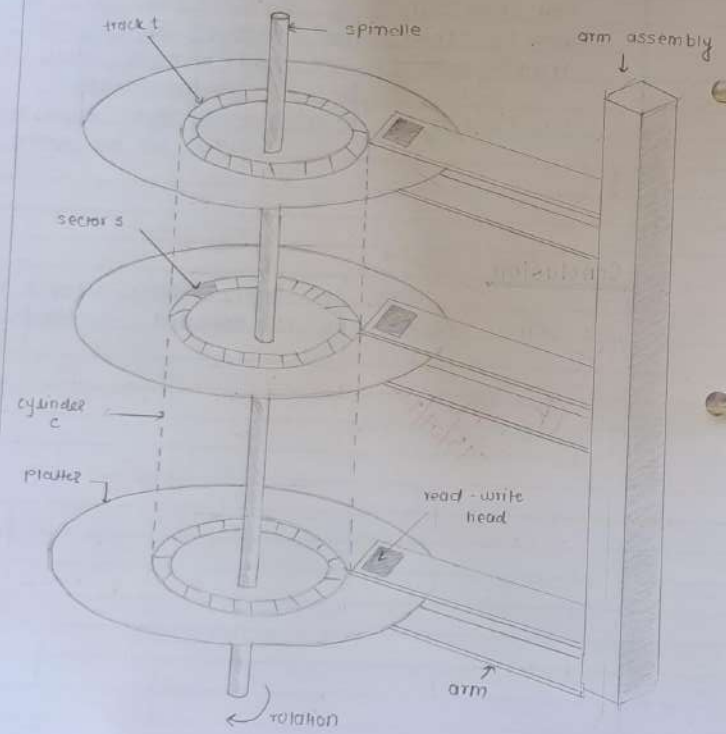
15/09/22

Page No.

EXPERIMENT 2

Aim -

To write a 'C' program to implement First Come First Serve (FCFS) Disk Scheduling Algorithm.



Date :

EXPERIMENT NO - 2

Aim

To write a 'C' program to implement First come First serve (FCFS) Disk Scheduling Algorithm

Theory

- FCFS is the simplest disk scheduling algorithm
- It is the simplest Disk scheduling algorithm It services the IO request in the order in which they arrive

- **Advantage**

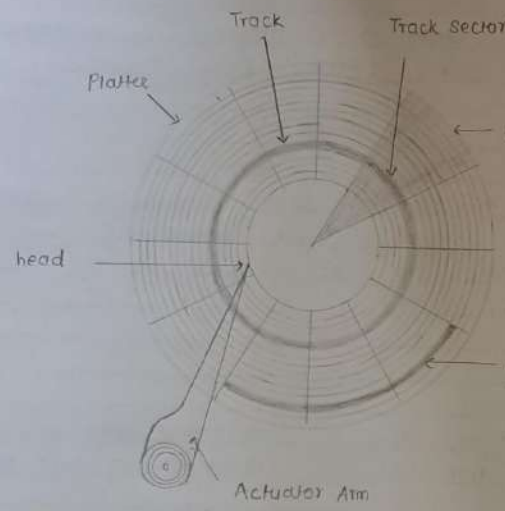
- It is simple, easy to understand and implement.
- It does not cause starvation to any request.

- **Disadvantage**

It results in increased total seek time.
It is inefficient.

- **Seek Time**

- A disk is divided into many circular tracks
- Seek Time is defined as the time required by the read/ write head to move desired track



Date :

Rotational latency

- The disk is divided into many circular tracks, and these track are further divided into blocks knows as sectors.
- The time required by the read/ write head to rotate to the request sector from the current position is called Rotational latency

Procedure for Execution

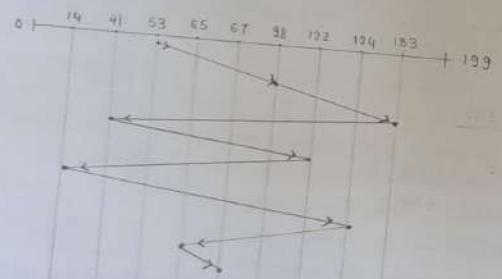
1. Open gedit
2. Type the program
3. Save the file with .c extension
4. Compile the program using gcc filename.c
5. Run the program by typing ./a.out.

Program

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
   int RQ[100], i,n, TotalHeadMovement=0, initial;
   printf(" Enter the number OF Request\n");
   scanf(" %d", &n);
   printf(" Enter the Request sequence\n");
   for( i=0; i<n ; i++)
    scanf(" %d", &RQ[i]);
```

Page No.

FCFS



Date :

```
printf(" Enter initial head position\n");
scanf(" %d", &initHead);
for(i=0; i<n; i++)
{
    TotalHeadMovement = TotalHeadMovement + abs
                        (RQ[i] - initHead);
    initHead = RQ[i];
}
printf(" Total head movement is %d", TotalHead-
                                     Movement);
return 0;
}
```

Numerical

Consider a disk queue with request for I/O to blocks on cylinder 98, 183, 41, 122, 14, 124, 65, 67. The FCFS scheduling algorithm is used. The head is initially at cylinder no. 53. The cylinder are numbered from 0 to 199 The total head movement (in number of cylinder) incurred while servicing the request is

Seek Time = |53-98| + |98-183| + |183-41| + |41-122| + |122-14| + |14-124| + |124-65| + |65-67|

= 632

A.S.T = 79

Page No.

Date :

To write a 'C' program to implement First Come First Serve (FCFS) CPU Scheduling Algorithm.

- The process which arrives first in the ready queue is firstly assigned the CPU.
- In case of a tie, process with smaller process id is executed first.
- It is always non-preemptive in nature.

- It is simple and easy to understand.
- It can be easily implemented using queue data structure.
- It does not lead to starvation.

- It does not consider the priority or burst time of the processes.

It is the time of arrival of a process in its ready state (before its execution).

Page No.

Date :

It is the exact time when a process complete i. the execution part.

WT refers to the total time that a process spends while waiting in a ready queue before reaching the CPU.

WT = TAT - BT

- TAT refers to the total time interval present between the time of process submission and the time of its completion.
- The difference betn the time of completion and the time of arrival is known as the Turn Around Time of the process.

TAT = CT - AT OR TAT = BT + WT

1. Open gedit.
2. Type the program.
3. Save the file with .c extension
4. Compile the program using gcc filename.c
5. Run the program by typing ./a.out

Date :

```
#include <stdio.h>
struct process
{
    int pid;
    int btime;
    int ttime;
    int wtime;
} p[10];

int main()
{
    int i,j,k,n,ttur,twat;
    float awat,atur;
    printf(" Enter the no of process: ");
    scanf(" %d", &n);
    for(i=0; i<n; i++)
    {
        printf(" Burst time for process p%d (in ms): ", (i+1));
        scanf(" %d", &p[i].btime);
    } p[i].pid = i+1;
    p[0].wtime = 0
    for(i=0; i<n; i++)
    {
        p[i+1].wtime = p[i].wtime + p[i].btime;
        p[i].ttime = p[i].wtime + p[i].btime;
    }
    ttur = twat = 0
    for(i=0; i<n; i++)
```

Date :

```
    {
        ttur t= P[i].ttime;
        twat= P[i].wtime;
    }
    awat= (float) twat/n;
    atur = (float) ttur/n;
    printF(" \n FCFS Scheduling \n\n");
    For( i=0; i<28 ; i++)
    printF("-");
    printf(" \n Process B-Time T-Time W-Time \n");
    for(i=0; i<28; i++)
    printF("-");
    For( i=0; i<n; i++)
    printF(" P%od \t %od \t %od \t %od", P[i].pid, P[i].btime,
            P[i].ttime, P[i].wtime);
    printF("\n");
    For(i=0 ; i<28, i++)
    printF("-");
    printF("\n\n GANTT Chart\n");
    printf("-");
    For(i=0 ; i<(P[n-1].ttime + 2*n); i++)
    printF("-");
    printF("\n");
    printF("1");
    For(i=0; i<n; i++)
    {
        k= P[i].btime/2;
        For(j=0; j<k ; j++)
        printF(" ");
        printF(" P%od", P[i].pid);
```

Page No.

```
        for(j=k+1; j<p[i].bTime; j++)
        printf("  ");
        printf("|");
    }
    printf("\n");
    printf("-");
    for(i=0; i<(p[n-1].tTime + 2*n); i++)
    printf("-");
    printf("\n");
    printf("0");
    for(i=0; i<n; i++)
    {
        for(j=0; j<p[i].bTime; j++)
        printf("  ");
        printf("%2d", p[i].tTime);
    }
    printf("\nAverage waiting time: %5.2f ms", awat);
    printf("\nAverage turn around time: %5.2f ms", attr);
    return 0;
}
```

Consider the processes P1, P2, P3, P4 given in the below table, arrives for execution in the same order, with Arrival Time 0, and given Burst Time. Find the average waiting Time using FCFS scheduling algorithm.

Date :

Process	Burst Time
P_1	21
P_2	3
P_3	6
P_4	2