

EXPERIMENT – 01

AIM: TO STUDY AND EXECUTE COMMANDS IN UNIX.

FILE RELATED COMMANDS:

1. **PWD COMMAND:** Prints the current working directory path.

```
(praveen@kali)-[~]  
$ pwd  
/home/praveen
```

2. **CD COMMAND:** Changes the current working directory.

```
(praveen@kali)-[~]  
$ cd Desktop
```

3. **LS COMMAND:** Lists files and directories in the current directory.

```
(praveen@kali)-[~/Desktop]  
$ ls -l  
total 16  
drwxr-xr-x 2 praveen praveen 4096 Sep 29 10:06 CET  
-rw-r--r-- 1 praveen praveen 558 Oct 4 2023 project_01_information_gathering_tool.py  
-rw-r--r-- 1 praveen praveen 1202 Oct 4 2023 project_02_network_and_port_scanner.py  
-rw-r--r-- 1 praveen praveen 328 Oct 6 2023 project_02_port_scanner.py
```

4. **RM COMMAND:** Removes files from the current directory.

```
(praveen@kali)-[~]  
$ rm short.py
```

5. **MV COMMAND:** Moves or renames files or directories.

```
(praveen@kali)-[~]  
$ mv cbit Desktop  
  
(praveen@kali)-[~]  
$ cd Desktop  
  
(praveen@kali)-[~/Desktop]  
$ ls -l  
total 20  
drwxr-xr-x 2 praveen praveen 4096 Sep 29 10:06 CET  
drwxr-xr-x 2 praveen praveen 4096 Sep 29 10:11 cbit  
-rw-r--r-- 1 praveen praveen 558 Oct 4 2023 project_01_information_gathering_tool.py  
-rw-r--r-- 1 praveen praveen 1202 Oct 4 2023 project_02_network_and_port_scanner.py  
-rw-r--r-- 1 praveen praveen 328 Oct 6 2023 project_02_port_scanner.py
```

6. **CAT COMMAND:** Concatenates and displays the contents of files.

```
(praveen@kali)-[~/Desktop]
└─$ cat >hello.txt
Good Mornning

(praveen@kali)-[~/Desktop]
└─$ cat hello.txt
Good Mornning

(praveen@kali)-[~/Desktop]
└─$ cat >hello.txt
I am a Student

(praveen@kali)-[~/Desktop]
└─$ cat hello.txt
I am a Student
```

7. **CMP COMMAND:** Compares two files byte by byte.

```
(praveen@kali)-[~/Desktop]
└─$ cmp hello.txt
cmp OS.txt
hello.txt - differ: byte 1, line 1
```

8. **CP COMMAND:** Copies files or directories.

```
(praveen@kali)-[~/Desktop]
└─$ cp OS.txt OS1.txt
```

9. **ECHO COMMAND:** Displays a string of text on the terminal.

```
(praveen@kali)-[~/Desktop]
└─$ echo "Hello"
Hello
```

10. **MKDIR COMMAND:** Creates a new directory.

```
(praveen@kali)-[~/Desktop]
└─$ mkdir sys.py
```

11. **PASTE COMMAND:** Merges lines of multiple files.

```
(praveen@kali)-[~/Desktop]
└─$ paste hello.txt OS.txt
I am a Student OS lab Internal is on Wednesday.
```

12. **RMDIR COMMAND:** Removes empty directories.

```
(praveen@kali)-[~/Desktop]
$ rmdir sys.py
```

13. **HEAD COMMAND:** Displays the first part of a file.

```
(praveen@kali)-[~/Desktop]
$ head states.txt
Andhra Pradesh
Arunachal Pradesh
Assam
Bihar
Chhattisgarh
Goa
Gujarat
Haryana
Himachal Pradesh
Jammu and Kashmir
```

14. **TAIL COMMAND:** Displays the last part of a file.

```
(praveen@kali)-[~/Desktop]
$ tail states.txt
Odisha
Punjab
Rajasthan
Sikkim
Tamil Nadu
Telangana
Tripura
Uttar Pradesh
Uttarakhand
West Bengal
```

15. **DATE COMMAND:** Displays or sets the system date and time.

```
(praveen@kali)-[~/Desktop]
$ date
Sun Sep 29 11:06:25 IST 2024
```

16. **GREP COMMAND:** Searches for patterns in files.

```
(praveen@kali)-[~/Desktop]
$ grep "operating" OS1.txt
An operating system brings powerful benefits to computer software and software development. Without an operating system, every application's
functionality of the underlying computer, such as disk storage, network interfaces and so on. Considering the vast array of underlying hardware, it is
impractical.
As long as each application accesses the same resources and services in the same way, that system software -- the operating system -- can be
developed to develop and debug an application, while ensuring that users can control, configure and manage the system hardware through a common a
```

17. **TOUCH COMMAND:** Changes file timestamps or creates an empty file.

```
(praveen@kali)-[~/Desktop]
$ touch OS1.txt
```

18. **CHMOD COMMAND:** Changes file permissions.

```
(praveen@kali)-[~/Desktop]
$ chmod 755 OS1.txt
```

19. **MAN COMMAND:** Displays the manual pages for commands.

```
(praveen@kali)-[~/Desktop]
$ man ls
```

```
LS(1) User Commands
NAME
  ls - list directory contents
SYNOPSIS
  ls [OPTION]... [FILE]...
DESCRIPTION
  List information about the FILES (the current directory by default). Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.
  Mandatory arguments to long options are mandatory for short options too.
  -a, --all
    do not ignore entries starting with .
  -A, --almost-all
    do not list implied . and ..
  --author
    with -l, print the author of each file
  -b, --escape
    print C-style escapes for nongraphic characters
  --block-size=SIZE
    with -l, scale sizes by SIZE when printing them; e.g., '--block-size=M'; see SIZE format below
  -B, --ignore-backups
    do not list implied entries ending with ~
  -c
    with -lt: sort by, and show, ctime (time of last modification of file status information); with -l: show ctime and sort by name; otherwise: sort by ctime, newest first
  -C
    list entries by columns
  --color[=WHEN]
    color the output WHEN; more info below
  -d, --directory
    list directories themselves, not their contents
  -D, --dired
    generate output designed for Emacs' dired mode
  -f
    list all entries in directory order
  -F, --classify[=WHEN]
    Manual page ls(1) line 1 (press h for help or q to quit)
```

20. **CLEAR COMMAND:** Clears the terminal screen.

```
root      5371      2  0 13:43 ?        00:00:00 [kworker/2:3H-ttm]
root      5372      2  0 13:43 ?        00:00:00 [kworker/2:4H-ttm]
root      5373      2  0 13:44 ?        00:00:00 [kworker/1:2-ata_sff]
praveen   5378    1299  3 13:44 ?        00:00:01 /usr/bin/eog /home/p
root      5405      2  0 13:44 ?        00:00:00 [kworker/u6:3-events]
praveen   5416    4144  0 13:44 pts/0    00:00:00 ps -ef

(praveen@kali)-[~]
$ clear
```

PROCESS RELATED COMMANDS:

- TOP COMMAND:** Displays a real-time view of active processes, including CPU and memory usage.

```
(praveen@kali)-[~]
$ top
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1642	praveen	20	0	3704540	260492	115072	S	10.3	6.5	0:23.56	gnome-shell
1396	praveen	20	0	407964	105360	64480	S	7.7	2.6	0:13.97	Xorg
3736	praveen	20	0	633380	57692	42212	R	7.7	1.4	0:00.52	gnome-terminal-
215	root	-51	0	0	0	0	S	2.6	0.0	0:00.45	irq/18-vmwgfx
2811	praveen	20	0	166520	6016	5376	S	2.6	0.2	0:00.14	sd_dummy
3781	praveen	20	0	12276	5248	3072	R	2.6	0.1	0:00.07	top
1	root	20	0	21080	12616	9288	S	0.0	0.3	0:01.50	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kthreadd
3	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_gp
4	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_par_gp
5	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	slub_flushwq
6	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	netns
10	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	mm_percpu_wq

- PS COMMAND:** Shows a snapshot of the current processes in the system.
ps: Displays processes running in the current shell.

```
(praveen@kali)-[~]
$ ps
```

PID	TTY	TIME	CMD
4144	pts/0	00:00:00	zsh
4163	pts/0	00:00:00	ps

ps -aux: Shows detailed information about all processes for all users.

```
(praveen@kali)-[~]
$ ps -aux
```

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.1	0.3	21080	12616	?	Ss	11:53	0:01	/sbin/init sp
root	2	0.0	0.0	0	0	?	S	11:53	0:00	[kthreadd]
root	3	0.0	0.0	0	0	?	I<	11:53	0:00	[rcu_gp]
root	4	0.0	0.0	0	0	?	I<	11:53	0:00	[rcu_par_gp]
root	5	0.0	0.0	0	0	?	I<	11:53	0:00	[slub_flushwq]
root	6	0.0	0.0	0	0	?	I<	11:53	0:00	[netns]
root	10	0.0	0.0	0	0	?	I<	11:53	0:00	[mm_percpu_wq]
root	11	0.0	0.0	0	0	?	I	11:53	0:00	[rcu_tasks_kt]
root	12	0.0	0.0	0	0	?	I	11:53	0:00	[rcu_tasks_ru]
root	13	0.0	0.0	0	0	?	I	11:53	0:00	[rcu_tasks_tr]
root	14	0.0	0.0	0	0	?	S	11:53	0:00	[ksoftirqd/0]
root	15	0.0	0.0	0	0	?	I	11:53	0:00	[rcu_preempt]
root	16	0.0	0.0	0	0	?	S	11:53	0:00	[migration/0]
root	17	0.0	0.0	0	0	?	S	11:53	0:00	[idle_inject/
root	19	0.0	0.0	0	0	?	S	11:53	0:00	[cpuhp/0]
root	20	0.0	0.0	0	0	?	S	11:53	0:00	[cpuhp/1]
root	21	0.0	0.0	0	0	?	S	11:53	0:00	[idle_inject/
root	22	0.0	0.0	0	0	?	S	11:53	0:00	[migration/1]
root	23	0.0	0.0	0	0	?	S	11:53	0:00	[ksoftirqd/1]
root	26	0.0	0.0	0	0	?	S	11:53	0:00	[cpuhp/2]
root	27	0.0	0.0	0	0	?	S	11:53	0:00	[idle_inject/
root	28	0.0	0.0	0	0	?	S	11:53	0:00	[migration/2]

ps -ef: Lists all processes in a detailed format with additional fields.

```
(praveen@kali)-[~]
$ ps -ef
```

UID	PID	PPID	C	STIME	TTY	TIME	CMD
root	1	0	0	12:40	?	00:00:02	/sbin/init splash
root	2	0	0	12:40	?	00:00:00	[kthreadd]
root	3	2	0	12:40	?	00:00:00	[rcu_gp]
root	4	2	0	12:40	?	00:00:00	[rcu_par_gp]
root	5	2	0	12:40	?	00:00:00	[slub_flushwq]

3. **KILL COMMAND:** Sends a signal to terminate or control a process.

SIGKILL (-9): Forcefully terminates a process without cleanup.

```
(praveen@kali)-[~]
$ kill -9 1234
```

SIGTERM (-15): Gracefully terminates a process, allowing cleanup.

```
(praveen@kali)-[~]
$ kill -15 1432
```

4. **NICE COMMAND:** Starts a process with a modified priority (niceness value), where a higher value reduces its priority.

```
(praveen@kali)-[~]
$ nice -n 10 project1.py
nice: 'project1.py': No such file or directory
```

NETWORK RELATED COMMANDS:

1. **PING COMMAND:** Tests network connectivity by sending ICMP echo requests and measuring response times.

```
(praveen@kali)-[~]
$ ping facebook.com
PING facebook.com (163.70.140.35) 56(84) bytes of data.
64 bytes from edge-star-mini-shv-01-hyd1.facebook.com (163.70.140.35): icmp_seq=1 ttl=55 time=5.14 ms
64 bytes from edge-star-mini-shv-01-hyd1.facebook.com (163.70.140.35): icmp_seq=2 ttl=55 time=6.13 ms
64 bytes from edge-star-mini-shv-01-hyd1.facebook.com (163.70.140.35): icmp_seq=3 ttl=55 time=9.49 ms
64 bytes from edge-star-mini-shv-01-hyd1.facebook.com (163.70.140.35): icmp_seq=4 ttl=55 time=4.58 ms
64 bytes from edge-star-mini-shv-01-hyd1.facebook.com (163.70.140.35): icmp_seq=5 ttl=55 time=5.72 ms
64 bytes from edge-star-mini-shv-01-hyd1.facebook.com (163.70.140.35): icmp_seq=6 ttl=55 time=4.70 ms
64 bytes from edge-star-mini-shv-01-hyd1.facebook.com (163.70.140.35): icmp_seq=7 ttl=55 time=4.66 ms
64 bytes from edge-star-mini-shv-01-hyd1.facebook.com (163.70.140.35): icmp_seq=8 ttl=55 time=9.09 ms
64 bytes from edge-star-mini-shv-01-hyd1.facebook.com (163.70.140.35): icmp_seq=9 ttl=55 time=20.7 ms
64 bytes from edge-star-mini-shv-01-hyd1.facebook.com (163.70.140.35): icmp_seq=10 ttl=55 time=9.53 ms
64 bytes from edge-star-mini-shv-01-hyd1.facebook.com (163.70.140.35): icmp_seq=11 ttl=55 time=5.39 ms
64 bytes from edge-star-mini-shv-01-hyd1.facebook.com (163.70.140.35): icmp_seq=12 ttl=55 time=7.05 ms
64 bytes from edge-star-mini-shv-01-hyd1.facebook.com (163.70.140.35): icmp_seq=13 ttl=55 time=8.11 ms
64 bytes from edge-star-mini-shv-01-hyd1.facebook.com (163.70.140.35): icmp_seq=14 ttl=55 time=6.01 ms
64 bytes from edge-star-mini-shv-01-hyd1.facebook.com (163.70.140.35): icmp_seq=15 ttl=55 time=6.89 ms
```

2. **IFCONFIG COMMAND:** Displays and configures network interface parameters on a system.

```
(praveen@kali)-[~]
$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.1.8 netmask 255.255.255.0 broadcast 192.168.1.255
    inet6 fe80::a00:27ff:fec7:b004 prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:c7:b0:04 txqueuelen 1000 (Ethernet)
    RX packets 23672 bytes 21730678 (20.7 MiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 13771 bytes 4784402 (4.5 MiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 26 bytes 1540 (1.5 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 26 bytes 1540 (1.5 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

3. **NETSTAT COMMAND:** Displays active network connections, routing tables, and interface statistics.

```
(praveen@kali)-[~]
$ netstat -at
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 kali.bbrouter:37510     a23-65-124-99.dep:https ESTABLISHED
tcp        0      0 kali.bbrouter:53800     93.243.107.34.bc.:https ESTABLISHED
tcp        0      0 kali.bbrouter:47482     bom12s21-in-f5.1e:https ESTABLISHED
tcp        0      0 kali.bbrouter:60488     bom12s20-in-f10.1:https ESTABLISHED
```

4. **NSLOOKUP COMMAND:** Queries DNS to retrieve domain name and IP address information.

```
(praveen@kali)-[~]
$ nslookup www.cbit.ac.in
Server:          192.168.1.1
Address:         192.168.1.1#53

Non-authoritative answer:
Name:   www.cbit.ac.in
Address: 3.111.165.12
```

5. **TELNET COMMAND:** Establishes a connection to a remote server for communication or testing routes.

```
(praveen@kali)-[~]
$ telnet google.com 80
Trying 142.251.42.46...
Connected to google.com.
Escape character is '^]'.

```

6. **TRACEROUTE COMMAND:** Traces the path packets take to reach a network host, showing each hop.

```
(praveen@kali)-[~]
$ traceroute facebook.com
traceroute to facebook.com (163.70.140.35), 30 hops max, 60 byte packets
 1  EARTH-1010.bbrouter (192.168.1.1)  7.475 ms  7.430 ms  7.412 ms
 2  103.44.2.93 (103.44.2.93)  7.590 ms  7.698 ms  7.681 ms
 3  * * *
 4  * * *
 5  * * *
 6  ae1.pr02.hyd1.tfbnw.net (157.240.82.174)  9.853 ms  4.302 ms  4.257 ms
 7  po202.asw04.hyd1.tfbnw.net (129.134.96.248)  4.232 ms  4.213 ms po202.asw01.hyd1.tfbnw.net
 8  psw01.hyd1.tfbnw.net (129.134.115.158)  4.178 ms psw04.hyd1.tfbnw.net (129.134.115.155)  4
 9  157.240.38.83 (157.240.38.83)  7.156 ms 173.252.67.165 (173.252.67.165)  7.103 ms 173.252.6
```

STUDY ABOUT UNIX VI EDITORS AND ITS FEATURES

DESCRIPTION:

The **vi editor** is a powerful text editor that comes pre-installed on most UNIX and Linux systems. It is used primarily for editing plain text files and is known for being lightweight and efficient. The **vi** editor operates in three different modes, which make it versatile for various text manipulation tasks:

MODES OF VI EDITOR:

1. **Command Mode:** This is the default mode when vi is launched. It allows the user to navigate within the file, delete text, copy and paste, and switch to other modes.
2. **Insert Mode:** In this mode, users can input or edit text. It can be accessed from command mode by pressing i, I, a, A, etc.
3. **Last Line Mode (Ex Mode):** This mode is used to execute more complex commands, such as saving the file, quitting the editor, searching, and performing other file operations. It is accessed by typing : in command mode.

KEY FEATURES OF VI EDITOR:

1. **Lightweight and Fast:** vi is a very lightweight editor, making it efficient even on older systems.
2. **Cross-Platform:** Works on a wide range of UNIX-based systems and Linux distributions.
3. **No GUI Required:** vi is a terminal-based editor, making it useful for servers or environments without a graphical user interface (GUI).
4. **Syntax Highlighting:** In modern versions, such as vim (an extended version of vi), syntax highlighting is available for easier code editing.
5. **Search and Replace:** vi offers powerful search (/searchterm) and replace (:%s/old/new/g) capabilities across files.
6. **Multiple Buffers:** vi supports editing multiple files simultaneously by using buffers, allowing users to switch between them.

EXPERIMENT – 02

AIM: DEMONSTRATION OF FILE RELATED SYSTEM CALLS SUCH AS CREATE, OPEN, CLOSE, READ, WRITE, LSEEK.

DESCRIPTION:

1. Create: Creates a new file with the specified name. If the file exists, it will be truncated.

- **Syntax:** `int creat(const char *pathname, mode_t mode);`
- **Parameters:**
 - `pathname`: The name (path) of the file to be created.
 - `mode`: The file permissions (e.g., read, write, execute) for the new file.
- **Return:** Returns a file descriptor on success or -1 on failure.

2. Open: Opens an existing file and returns a file descriptor for subsequent operations.

- **Syntax:** `int open(const char *pathname, int flags, mode_t mode);`
- **Parameters:**
 - `pathname`: The name (path) of the file to be opened.
 - `flags`: Specifies how the file is to be opened (e.g., `O_RDONLY` for read-only, `O_WRONLY` for write-only).
 - `mode`: File permissions (used only when creating a new file).
- **Return:** Returns a file descriptor on success or -1 on failure.

3. Close: Closes an open file descriptor, freeing associated system resources.

- **Syntax:** `int close(int fd);`
- **Parameters:**
 - `fd`: The file descriptor of the file to close.
- **Return:** Returns 0 on success or -1 on failure.

4. Read: Reads data from a file descriptor into a buffer in memory.

- **Syntax:** `ssize_t read(int fd, void *buf, size_t count);`
- **Parameters:**
 - `fd`: The file descriptor from which to read.
 - `buf`: A buffer to store the data being read.
 - `count`: The number of bytes to read.
- **Return:** Returns the number of bytes read or -1 on failure.

5. Write: Writes data from a buffer in memory to a file.

- **Syntax:** `ssize_t write(int fd, const void *buf, size_t count);`
- **Parameters:**
 - `fd`: The file descriptor to which to write.
 - `buf`: A buffer containing the data to be written.
 - `count`: The number of bytes to write.
- **Return:** Returns the number of bytes written or -1 on failure.

6. Lseek: Changes the current file offset (position) for reading or writing.

- **Syntax:** off_t lseek(int fd, off_t offset, int whence);
- **Parameters:**
 - fd: The file descriptor of the file to reposition.
 - offset: The new position relative to the location specified by whence.
 - whence: Specifies the reference point for the offset (e.g., SEEK_SET, SEEK_CUR, SEEK_END).
- **Return:** Returns the new file offset or -1 on failure.

PROGRAMS:

Program 1: WAP to copy the content from one file to another without deleting the content of second file.

CODE:

```
Open ▾ program1.c
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>

void main() {
    char b[100];

    int fd = open("file1.txt", O_RDONLY);
    int fd1 = open("file3.txt", O_WRONLY | O_APPEND);

    int n = read(fd, b, 100);

    write(fd1, b, n);

    close(fd);
    close(fd1);
}
```

OUTPUT:

```
Open ▾ file1.txt
Hello, CBITians..

Open ▾ file3.txt
We are Legends
Hello, CBITians..
```

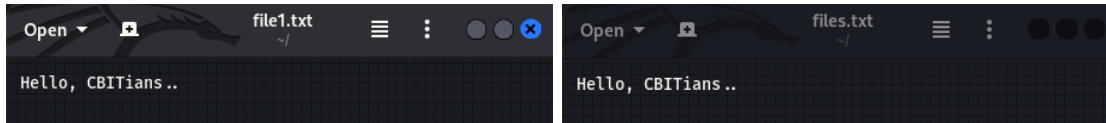
Program 2: WAP to read from one file and write into another file.

CODE:

```
Open ▾ program2.c
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>

void main() {
    char b[100];
    int fd = open("file1.txt", O_RDONLY);
    int fd1 = open("file2.txt", O_WRONLY | O_CREAT |
O_TRUNC, S_IRUSR | S_IWUSR);
    int n = read(fd, b, 100);
    write(fd1, b, n);
    close(fd);
    close(fd1);
}
```

OUTPUT:

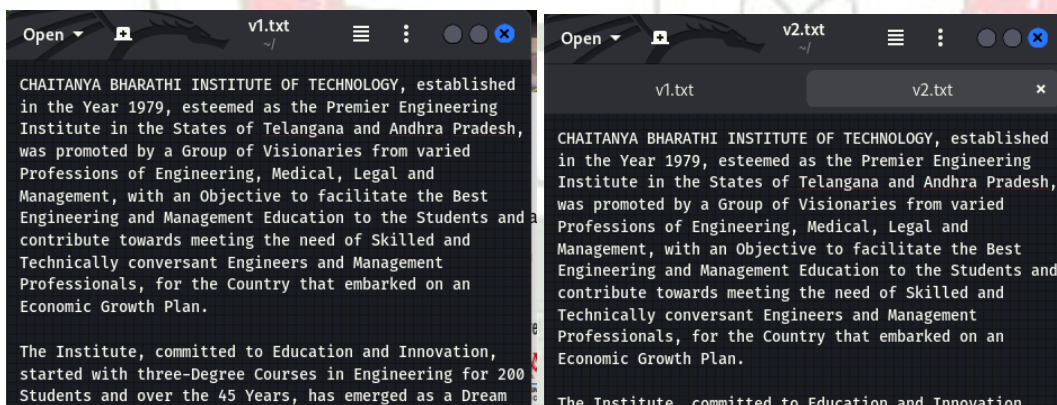


Program 3: WAP to copy the entire content from one file to another
(here, content is more than 100 characters).

CODE:

```
Open ▾ file1.txt ~/  
Hello, CBITians..  
  
Open ▾ files.txt ~/  
Hello, CBITians..  
  
Open ▾ program3.c ~/  
#include <stdio.h>  
#include <fcntl.h>  
#include <unistd.h>  
#include <sys/types.h>  
#include <sys/stat.h>  
  
void main() {  
    char b[1000];  
    int fd = open("v1.txt", O_RDONLY);  
    int fd1 = open("v2.txt", O_WRONLY | O_CREAT | O_TRUNC,  
S_IRUSR | S_IWUSR);  
    int n = read(fd, b, 1000);  
    write(fd1, b, n);  
    close(fd);  
    close(fd1);  
}
```

OUTPUT:

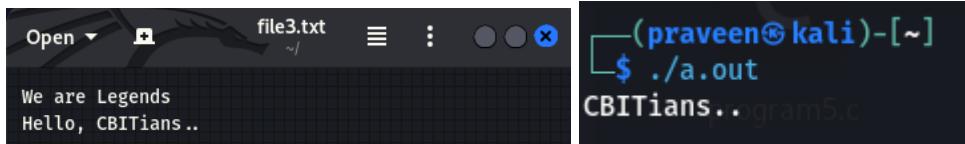


Program 4: WAP to print last 5 characters of a file onto the terminal screen.

CODE:

```
Open ▾ program4.c ~/  
#include <sys/types.h>  
#include <sys/stat.h>  
#include <fcntl.h>  
#include <unistd.h>  
#include <stdio.h>  
  
void main() {  
    char b[200];  
    int fd = open("file3.txt", O_RDWR);  
  
    lseek(fd, -11, SEEK_END);  
    int n = read(fd, b, 11);  
    write(1, b, n);  
  
    close(fd);  
}
```

OUTPUT:

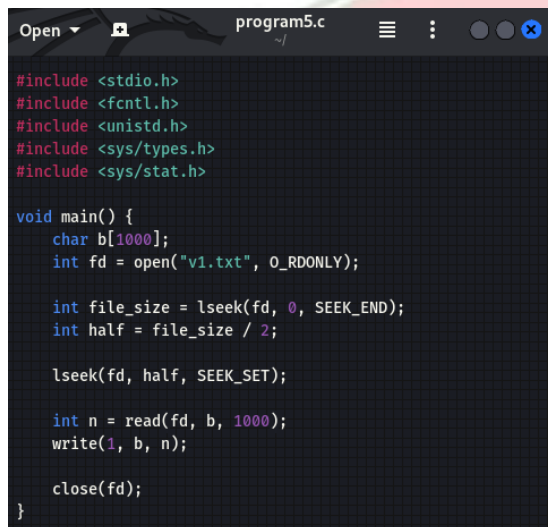


```
file3.txt
We are Legends
Hello, CBITians..

(praveen@kali)-[~]
$ ./a.out
CBITians..
```

Program 5: WAP to print second half of the file.

CODE:



```
program5.c
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>

void main() {
    char b[1000];
    int fd = open("v1.txt", O_RDONLY);

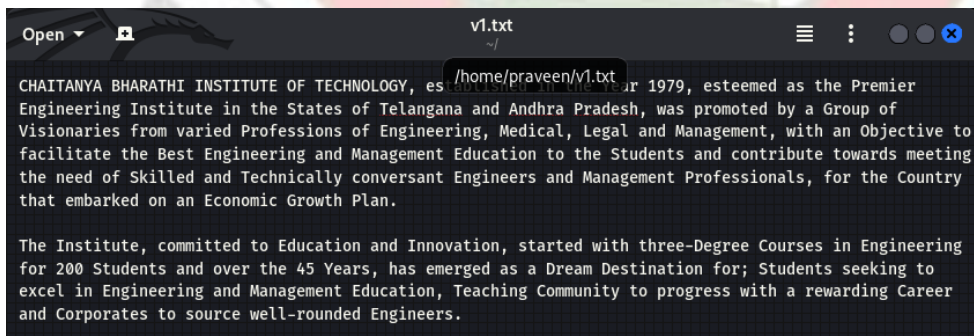
    int file_size = lseek(fd, 0, SEEK_END);
    int half = file_size / 2;

    lseek(fd, half, SEEK_SET);

    int n = read(fd, b, 1000);
    write(1, b, n);

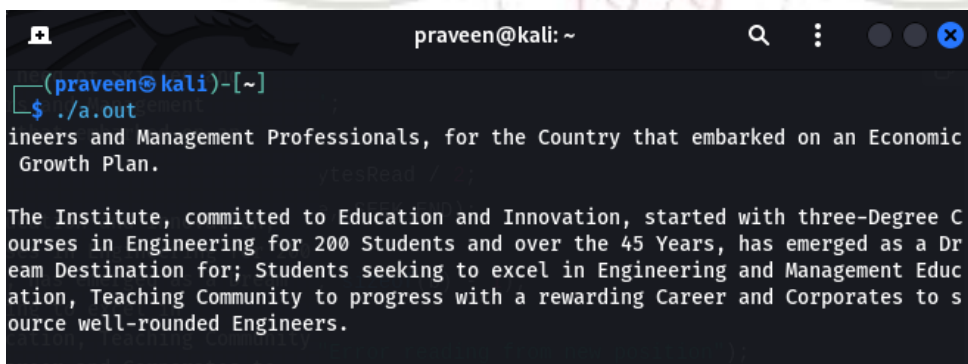
    close(fd);
}
```

OUTPUT:



```
v1.txt
CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY, established in 1979, esteemed as the Premier Engineering Institute in the States of Telangana and Andhra Pradesh, was promoted by a Group of Visionaries from varied Professions of Engineering, Medical, Legal and Management, with an Objective to facilitate the Best Engineering and Management Education to the Students and contribute towards meeting the need of Skilled and Technically conversant Engineers and Management Professionals, for the Country that embarked on an Economic Growth Plan.

The Institute, committed to Education and Innovation, started with three-Degree Courses in Engineering for 200 Students and over the 45 Years, has emerged as a Dream Destination for; Students seeking to excel in Engineering and Management Education, Teaching Community to progress with a rewarding Career and Corporates to source well-rounded Engineers.
```



```
praveen@kali: ~
(praveen@kali)-[~]
$ ./a.out
The Institute, committed to Education and Innovation, started with three-Degree Courses in Engineering for 200 Students and over the 45 Years, has emerged as a Dream Destination for; Students seeking to excel in Engineering and Management Education, Teaching Community to progress with a rewarding Career and Corporates to source well-rounded Engineers.
```


Program 6: WAP to print characters from 10th to 20th position of a file.

CODE:

```
Open ▾  program6.c  ~/  
  
#include <stdio.h>  
#include <fcntl.h>  
#include <unistd.h>  
#include <sys/types.h>  
#include <sys/stat.h>  
  
void main() {  
    char b[11];  
    int fd = open("v1.txt", O_RDONLY);  
  
    lseek(fd, 9, SEEK_SET);  
  
    int n = read(fd, b, 10);  
    write(1, b, n);  
  
    close(fd);  
}
```

OUTPUT:

```
Open ▾  v1.txt  ~/  
  
CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY, es /home/praveen/v1.txt  
Engineering Institute in the States of Telangana and Andhra Pradesh, was promoted by a Group of  
Visionaries from varied Professions of Engineering, Medical, Legal and Management, with an Objective to  
facilitate the Best Engineering and Management Education to the Students and contribute towards meeting  
the need of Skilled and Technically conversant Engineers and Management Professionals, for the Country  
that embarked on an Economic Growth Plan.  
  
The Institute, committed to Education and Innovation, started with three-Degree Courses in Engineering  
for 200 Students and over the 45 Years, has emerged as a Dream Destination for; Students seeking to  
excel in Engineering and Management Education, Teaching Community to progress with a rewarding Career  
and Corporates to source well-rounded Engineers.
```

```
(praveen@kali)-[~]  
$ cc program6.c  
(praveen@kali)-[~]  
$ ./a.out  
BHARATHI
```

EXPERIMENT – 03

AIM: *DEMONSTRATION OF PROCESS RELATED SYSTEM CALLS SUCH AS FORK, GETPID, GETPPID, WAIT, EXEC etc.*

DESCRIPTION:

Process-related system calls in Linux manage process creation, control, and synchronization:

1. fork()

- **Header Files:** #include <unistd.h>
- **Syntax:** pid_t fork(void);
- **Explanation:** Creates a new process by duplicating the calling process. The new process (child) gets a unique process ID, sharing most of the parent's resources. It returns 0 to the child process and the child's PID to the parent. On failure, it returns -1.

2. getpid()

- **Header Files:** #include <unistd.h>
- **Syntax:** pid_t getpid(void);
- **Explanation:** Returns the process ID (PID) of the calling process. Useful for identifying the process, especially in multi-process applications.

3. getppid()

- **Header Files:** #include <unistd.h>
- **Syntax:** pid_t getppid(void);
- **Explanation:** Returns the parent process ID (PPID) of the calling process, useful for tracking process lineage.

4. exec() Family

- **Header Files:** #include <unistd.h>
- **Syntax:** int execve(const char *pathname, char *const argv[], char *const envp[]);
- **Explanation:** Replaces the current process image with a new process image specified by the pathname. The argv argument holds pointers to the command's arguments, and envp holds pointers to environment variables. If successful, exec does not return; otherwise, it returns -1.

5. wait()

- **Header Files:** #include <sys/types.h>, #include <sys/wait.h>
- **Syntax:** pid_t wait(int *status);
- **Explanation:** Suspends execution of the calling process until one of its child processes terminates. It returns the PID of the terminated child and stores the child's termination status in the variable pointed to by status. Returns -1 if no child processes exist.

1. Program to demonstrate fork system call

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>

int main() {
    int a = 2;
    pid_t pid;
    pid = fork();

    printf("%d\n", pid);

    if (pid < 0) {
        printf("FORK FAILED\n");
    } else if (pid == 0) {
        printf("CHILD PROCESS \t a is: ");
        printf("%d\n", ++a);
    } else {
        printf("PARENT PROCESS \t a is: ");
        printf("%d\n", --a);
    }

    printf("EXITING WITH X = %d\n", a);

    return 0;
}
```

OUTPUT:

```
9675
PARENT PROCESS    a is: 1
EXITING WITH X = 1
0
CHILD PROCESS     a is: 3
EXITING WITH X = 3
```

2. Program to demonstrate getpid(),getppid() system calls

Without wait()

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>

int main() {
    int a = 2;
    pid_t pid;
    pid = fork();

    printf("%d\n", pid);

    if (pid < 0) {
        printf("FORK FAILED\n");
    } else if (pid == 0) {
        printf("CHILD PROCESS \t a is: %d\n", ++a);
        printf("I AM THE CHILD AND MY PROCESS ID IS %d\n", getpid());
        printf("I AM THE CHILD AND MY PARENT'S PROCESS ID IS %d\n", getppid());
    } else {
        printf("PARENT PROCESS \t a is: %d\n", --a);
        printf("I AM THE PARENT AND MY PROCESS ID IS %d\n", getpid());
        printf("I AM THE PARENT AND MY CHILD'S PROCESS ID IS %d\n", pid);
    }

    printf("EXITING WITH X = %d\n", a);

    return 0;
}
```

OUTPUT:

```
9785
PARENT PROCESS  a is: 1
I AM THE PARENT AND MY PROCESS ID IS 9784
I AM THE PARENT AND MY CHILD'S PROCESS ID IS 9785
EXITING WITH X = 1
0
CHILD PROCESS  a is: 3
I AM THE CHILD AND MY PROCESS ID IS 9785
I AM THE CHILD AND MY PARENT'S PROCESS ID IS 1299
EXITING WITH X = 3
```

With Wait()

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

int main() {
    pid_t pid = fork();

    if (pid < 0) { // Error in creating a child
        printf("Child cannot be created\n");
    }
    else if (pid > 0) { // Parent section
        printf("Process id of child = %d\n", pid);
        printf("Process id of parent = %d\n", getpid());

        int status;
        pid_t child_pid = wait(&status); // Parent waits for the child

        if (child_pid > 0) {
            if (WIFEXITED(status)) {
                printf("Child process (PID: %d) exited with status %d\n", child_pid, WEXITSTATUS(status));
            } else {
                printf("Child process (PID: %d) did not terminate normally\n", child_pid);
            }
        } else {
            perror("wait failed");
        }
    }
    else { // Child section
        sleep(1); // Simulate some work
        printf("Process id of child = %d\n", getpid());
        printf("Process id of parent = %d\n", getppid());
    }

    return 0;
}
```

OUTPUT:

```
Process id of child = 9850
Process id of parent = 9849
Process id of child = 9850
Process id of parent = 9849
Child process (PID: 9850) exited with status 6
```

3. Write a Program to demonstrate the concept of Orphan and Zombie Process

ORPHAN CODE:

```
#include <stdio.h>
#include <unistd.h>

int main() {
    pid_t pid = fork();

    if (pid < 0) { // Error in creating a child
        printf("Child cannot be created\n");
    }
    else if (pid > 0) { // Parent section
        printf("Process id of child = %d\n", pid);
        printf("Process id of parent = %d\n", getpid());
    }
    else { // Child section
        sleep(1); // Simulate delay to show the child becoming an orphan
        printf("Process id of child = %d\n", getpid());
        printf("Process id of parent = %d\n", getppid()); // Shows parent's process ID
    }

    return 0;
}
```

Sleep():

```
#include <stdio.h>
#include <unistd.h>

int main() {
    pid_t pid = fork();

    if (pid < 0) { // Error in creating a child
        printf("Child cannot be created\n");
    }
    else if (pid > 0) { // Parent section
        sleep(1); // Sleep to ensure child doesn't become orphan
        printf("Process id of child = %d\n", pid); // pid holds child's PID
        printf("Process id of parent = %d\n", getpid());
    }
    else { // Child section
        printf("Process id of child = %d\n", getpid());
        printf("Process id of parent = %d\n", getppid());
    }

    return 0;
}
```


OUTPUT:

```
(praveen@kali)-[~]  
$ ./a.out  
Process id of child = 9915  
Process id of parent = 9914  
  
(praveen@kali)-[~]  
$ Process id of child = 9915  
Process id of parent = 1299
```

OUTPUT:

```
Process id of child = 10012  
Process id of parent = 10011  
Process id of child = 10012  
Process id of parent = 10011
```

4. Program to demonstrate exec System Call

```
#include <stdio.h>  
#include <unistd.h>  
#include <stdlib.h>  
#include <sys/types.h>  
#include <sys/wait.h>  
  
int main(int argc, char *argv[]) {  
    int pid = fork();  
  
    if (pid < 0) {  
        printf("Fork failed\n");  
        return 1;  
    }  
    else if (pid == 0) {  
        execl("/bin/ls", "ls", NULL);  
        perror("execl failed");  
        exit(1);  
    }  
    else {  
        wait(NULL);  
        printf("Child process complete\n");  
    }  
  
    return 0;  
}
```

OUTPUT:

Desktop	Public	file1.txt	getpid_withwait.c	program4.c	v2.txt
Documents	Templates	file3.txt	orphan.c	program5.c	
Downloads	Videos	files.txt	program1.c	program6.c	
Music	a.out	fork.c	program2.c	sleep.c	
Pictures	exec.c	getpid.c	program3.c	v1.txt	
Child process complete					

EXPERIMENT – 04

AIM: *Implement CPU Scheduling Algorithms (a) Round Robin (b) SJF (c) FCFS.*

DESCRIPTION:

Round Robin (RR) scheduling allocates CPU time to each process in a circular order. Each process gets a fixed time quantum to execute before being moved to the back of the queue. This method ensures fairness and responsiveness but can lead to higher average turnaround times if the quantum is not well-tuned.

Shortest Job First (SJF) scheduling selects the process with the smallest execution time for the CPU next. It can be either preemptive (Shortest Remaining Time First) or non-preemptive. SJF minimizes average waiting time and turnaround time but may lead to process starvation if short jobs keep arriving.

First Come First Serve (FCFS) scheduling processes tasks in the order they arrive. The first process in the queue is executed to completion before moving to the next. While simple and easy to implement, FCFS can lead to the "convoy effect," where longer processes delay the execution of shorter ones.

ALGORITHM:

1. FCFS Scheduling

1. Input the processes along with their burst time (bt).
2. Find waiting time (wt) for all processes.
3. As first process that comes need not to wait so
 - a. waiting time for process 1 will be 0 i.e. $wt[0] = 0$.
4. Find waiting time for all other processes i.e. for
 - a. process $i \rightarrow wt[i] = bt[i-1] + wt[i-1]$.
5. Find turnaround time = waiting_time + burst_time for all processes.
6. Find average waiting time = $\text{total_waiting_time} / \text{no_of_processes}$.
7. Similarly, find average turnaround time = $\text{total_turn_around_time} / \text{no_of_processes}$.

2. SJF Scheduling

1. Sort all processes according to the arrival time.
2. Select the process that has the minimum arrival time and minimum burst time.
3. After the completion of the process, create a pool of processes that arrive afterward until the completion of the previous process.
4. Select the process among the pool that has the minimum burst time.

5. Completion Time: Time at which the process completes its execution.
6. Turnaround Time: The time difference between completion time and arrival time.
 - $\text{Turnaround Time} = \text{Completion Time} - \text{Arrival Time}$
7. Waiting Time (W.T): The time difference between turnaround time and burst time.
 - $\text{Waiting Time} = \text{Turnaround Time} - \text{Burst Time}$

3. RR Scheduling

1. Initialize the queue.
2. Set the time quantum.
3. Enqueue all processes with their burst times.
4. While the queue is not empty:
 - $\text{current_process} = \text{Dequeue}()$
 - If $\text{burst_time}[\text{current_process}] \leq \text{time_quantum}$:
 - Execute current_process for burst_time[current_process].
 - Remove current_process from the queue.
 - Else:
 - Execute current_process for time_quantum.
 - Update $\text{burst_time}[\text{current_process}] = \text{burst_time}[\text{current_process}] - \text{time_quantum}$
 - Enqueue current_process at the end of the queue.
5. End while.

PROGRAMS:

1. FCFS

```
import matplotlib.pyplot as plt

print("FIRST COME FIRST SERVE SCHEDULING")
n = int(input("Enter number of processes: "))
processes = {}

for i in range(n):
    key = "P" + str(i + 1)
    arrival_time = int(input(f"Enter arrival time of process {key}: "))
    burst_time = int(input(f"Enter burst time of process {key}: "))
    processes[key] = [arrival_time, burst_time]

processes = dict(sorted(processes.items(), key=lambda item: item[1][0]))

ET = []
TAT = []
WT = []

for i in range(len(processes)):
    if i == 0:
        ET.append(processes[list(processes.keys())[i]][0] + processes[list(processes.keys())[i]][1])
    else:
        ET.append(ET[i - 1] + processes[list(processes.keys())[i]][1])

TAT = [ET[i] - processes[list(processes.keys())[i]][0] for i in range(len(processes))]
WT = [TAT[i] - processes[list(processes.keys())[i]][1] for i in range(len(processes))]

RT = []
for i in range(len(processes)):
    if i == 0:
        RT.append(0)
    else:
        RT.append(ET[i - 1] - processes[list(processes.keys())[i]][0])

avg_WT = sum(WT) / n
avg_TAT = sum(TAT) / n
total_time = ET[-1]
throughput = n / total_time

print("Process | Arrival | Burst | Exit | Turn Around | Wait | Response")
for i in range(n):
    process_key = list(processes.keys())[i]
    print(f" {process_key} | {processes[process_key][0]} | {processes[process_key][1]} | {ET[i]} | {TAT[i]} | {WT[i]} | {RT[i]}")
```

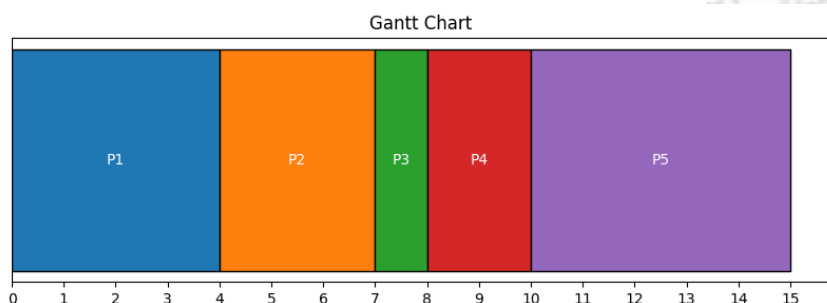
```
print("Average Waiting Time:", avg_WT)
print("Average Turnaround Time:", avg_TAT)
print("Throughput:", throughput, "processes/unit time")

plt.figure(figsize=(10, 3))
current_time = 0
for i in range(n):
    plt.barh(y=0, width=processes[list(processes.keys())[i]][1], left=current_time, edgecolor='black')
    plt.text(x=current_time + processes[list(processes.keys())[i]][1] / 2,
            y=0,
            s=list(processes.keys())[i],
            ha='center',
            va='center',
            color='white')
    current_time += processes[list(processes.keys())[i]][1]

plt.yticks([])
plt.xticks(range(current_time + 1))
plt.xlabel("Time")
plt.title("Gantt Chart")
plt.show()
```

OUTPUT:

```
FIRST COME FIRST SERVE SCHEDULING
Enter number of processes: 5
Enter arrival time of process P1: 0
Enter burst time of process P1: 4
Enter arrival time of process P2: 1
Enter burst time of process P2: 3
Enter arrival time of process P3: 2
Enter burst time of process P3: 1
Enter arrival time of process P4: 3
Enter burst time of process P4: 2
Enter arrival time of process P5: 4
Enter burst time of process P5: 5
Process | Arrival | Burst | Exit | Turn Around | Wait | Response
P1      | 0       | 4     | 4     | 4           | 0     | 0
P2      | 1       | 3     | 7     | 6           | 3     | 3
P3      | 2       | 1     | 8     | 6           | 5     | 5
P4      | 3       | 2     | 10    | 7           | 5     | 5
P5      | 4       | 5     | 15    | 11          | 6     | 6
Average Waiting Time: 3.8
Average Turnaround Time: 6.8
Throughput: 0.3333333333333333 processes/unit time
```



2. SJF

```
import matplotlib.pyplot as plt

def sjf_scheduling():
    print("SHORTEST JOB FIRST SCHEDULING")
    n = int(input("Enter number of processes: "))
    processes = []

    for i in range(n):
        process_id = i + 1
        arrival_time = int(input(f"Enter arrival time of process P(process_id): "))
        burst_time = int(input(f"Enter burst time of process P(process_id): "))
        processes.append([process_id, arrival_time, burst_time])

    processes.sort(key=lambda x: x[1])
    waiting_time = [0] * n
    turnaround_time = [0] * n
    completion_time = [0] * n
    response_time = [0] * n

    total_waiting_time = 0
    total_turnaround_time = 0
    total_response_time = 0
    current_time = 0
    completed_processes = 0
    completed = [False] * n
    gantt_chart = []
    gantt_times = []
```



```

while completed_processes < n:
    available_processes = [p for p in processes if p[1] <= current_time and not completed[p[0] - 1]]

    if available_processes:
        available_processes.sort(key=lambda x: x[2])
        process = available_processes[0]
        process_id = process[0]
        arrival_time = process[1]
        burst_time = process[2]

        if response_time[process_id - 1] == 0:
            response_time[process_id - 1] = current_time - arrival_time
            start_time = current_time
            completion_time[process_id - 1] = start_time + burst_time
            waiting_time[process_id - 1] = start_time - arrival_time
            turnaround_time[process_id - 1] = waiting_time[process_id - 1] + burst_time
            current_time += burst_time
            completed_processes += 1
            completed[process_id - 1] = True
            gantt_chart.append(f"P{process_id}")
            gantt_times.append((start_time, current_time))
        else:
            gantt_chart.append("IDLE")
            gantt_times.append((current_time, current_time + 1))
            current_time += 1

    avg_waiting_time = sum(waiting_time) / n
    avg_turnaround_time = sum(turnaround_time) / n
    avg_response_time = sum(response_time) / n
    total_time = completion_time[-1]
    throughput = n / total_time if total_time > 0 else 0

    print("\nProcess | Arrival Time | Burst Time | Waiting Time | Turnaround Time")
    for i in range(n):
        print(f" P{processes[i][0]} | {processes[i][1]} | {processes[i][2]} | {waiting_time[i]} | {turnaround_time[i]}")

    print(f"\nAverage Waiting Time: {avg_waiting_time:.2f}")
    print(f"Average Turnaround Time: {avg_turnaround_time:.2f}")
    print(f"Average Response Time: {avg_response_time:.2f}")
    print(f"Throughput: {throughput:.2f} processes/unit time")
    print("\nGantt Chart:")

    plt.figure(figsize=(10, 3))
    for i in range(len(gantt_chart)):
        if gantt_chart[i] != "IDLE":
            plt.barh(y=0, width=gantt_times[i][1] - gantt_times[i][0], left=gantt_times[i][0], edgecolor='black')
            plt.text(x=gantt_times[i][0] + (gantt_times[i][1] - gantt_times[i][0]) / 2,
                    y=0, s=gantt_chart[i], ha='center', va='center', color='white')

    plt.yticks([])
    plt.xticks(range(int(current_time) + 1))
    plt.xlabel("Time")
    plt.title("Gantt Chart")
    plt.show()

sjf_scheduling()

```

OUTPUT:

SHORTEST JOB FIRST SCHEDULING

```

Enter number of processes: 5
Enter arrival time of process P1: 1
Enter burst time of process P1: 7
Enter arrival time of process P2: 2
Enter burst time of process P2: 5
Enter arrival time of process P3: 3
Enter burst time of process P3: 1
Enter arrival time of process P4: 4
Enter burst time of process P4: 2
Enter arrival time of process P5: 5
Enter burst time of process P5: 8

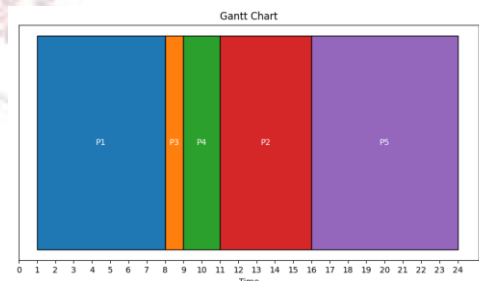
```

Process	Arrival Time	Burst Time	Waiting Time	Turnaround Time
P1	1	7	0	7
P2	2	5	9	14
P3	3	1	5	6
P4	4	2	5	7
P5	5	8	11	19

```

Average Waiting Time: 6.00
Average Turnaround Time: 10.60
Average Response Time: 6.00
Throughput: 0.21 processes/unit time

```



3. RR

```
import matplotlib.pyplot as plt
from collections import deque

def round_robin(processes, burst_time, arrival_time, quantum):
    n = len(processes)
    remaining_burst_time = burst_time[:]
    waiting_time = [0] * n
    turnaround_time = [0] * n
    response_time = [0] * n
    gantt_chart = []
    t = 0
    process_queue = deque()
    process_start_time = [-1] * n

    while True:
        for i in range(n):
            if arrival_time[i] <= t and remaining_burst_time[i] > 0 and i not in process_queue:
                process_queue.append(i)

        if process_queue:
            i = process_queue.popleft()
            if process_start_time[i] == -1:
                process_start_time[i] = t
            start_time = t
            burst = min(quantum, remaining_burst_time[i])
            t += burst
            remaining_burst_time[i] -= burst

            if remaining_burst_time[i] == 0:
                waiting_time[i] = t - burst_time[i] - arrival_time[i]
                turnaround_time[i] = t - arrival_time[i]
            else:
                process_queue.append(i)
            gantt_chart.append((processes[i], start_time, t))
        else:
            next_arrival = min([arrival_time[i] for i in range(n) if arrival_time[i] > t], default=None)
            if next_arrival is not None:
                t = next_arrival
            else:
                break

    avg_waiting_time = sum(waiting_time) / n
    avg_turnaround_time = sum(turnaround_time) / n
    avg_response_time = sum(response_time) / n
    total_time = max(t, max([arrival_time[i] + burst_time[i] for i in range(n)]))
    throughput = n / total_time if total_time > 0 else 0

    print("Gantt Chart:")
    plt.figure(figsize=(10, 3))
    for process, start, end in gantt_chart:
        plt.barh(y=0, width=end - start, left=start, edgecolor='black')
        plt.text(x=(start + (end - start) / 2, y=0, s=f"P{process}", ha='center', va='center', color='white'))

    plt.yticks([])
    plt.xticks(range(int(t) + 1))
    plt.xlabel("Time")
    plt.title("Gantt Chart")
    plt.show()

    print("\nProcess\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time\tResponse Time")
    for i in range(n):
        print(f"P{processes[i]}\t{arrival_time[i]}\t{burst_time[i]}\t{waiting_time[i]}\t{turnaround_time[i]}\t{response_time[i]}")

    print(f"\nAverage Waiting Time: {avg_waiting_time:.2f}")
    print(f"Average Turnaround Time: {avg_turnaround_time:.2f}")
    print(f"Average Response Time: {avg_response_time:.2f}")
    print(f"Throughput: {throughput:.2f} processes/unit time")

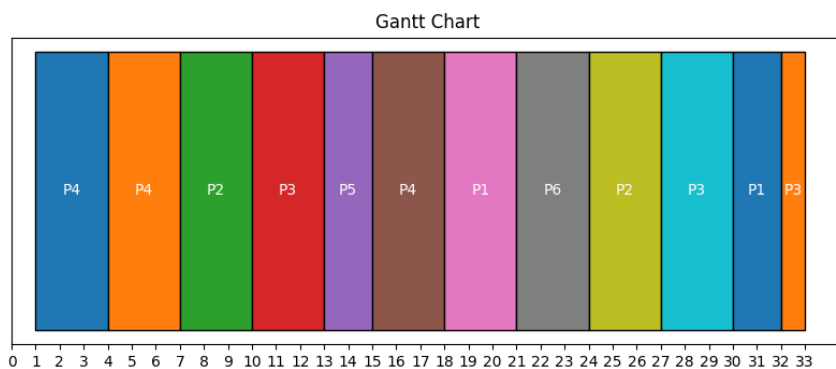
    n = int(input("Enter the number of processes: "))
    processes = []
    burst_time = []
    arrival_time = []

    for i in range(n):
        process_id = i + 1
        processes.append(process_id)
        bt = int(input(f"Enter the burst time for process P{process_id}: "))
        burst_time.append(bt)
        at = int(input(f"Enter the arrival time for process P{process_id}: "))
        arrival_time.append(at)

    quantum = int(input("Enter the time quantum: "))
    round_robin(processes, burst_time, arrival_time, quantum)
```

OUTPUT:

```
Enter the number of processes: 6
Enter the burst time for process P1: 5
Enter the arrival time for process P1: 5
Enter the burst time for process P2: 6
Enter the arrival time for process P2: 4
Enter the burst time for process P3: 7
Enter the arrival time for process P3: 3
Enter the burst time for process P4: 9
Enter the arrival time for process P4: 1
Enter the burst time for process P5: 2
Enter the arrival time for process P5: 2
Enter the burst time for process P6: 3
Enter the arrival time for process P6: 6
Enter the time quantum: 3
```



Process	Arrival Time	Burst Time	Waiting Time	Turnaround Time	Response Time
P1	5	5	22	27	22
P2	4	6	17	23	17
P3	3	7	23	30	23
P4	1	9	8	17	8
P5	2	2	11	13	11
P6	6	3	15	18	15

Average Waiting Time: 16.00
Average Turnaround Time: 21.33
Average Response Time: 16.00
Throughput: 0.18 processes/unit time

OUTPUT ANALYSIS:

1. First Come First Serve (FCFS)

Time Complexity: $O(n)$

Processes are executed in the order they arrive.

Space Complexity: $O(n)$

Requires space to store process information, such as arrival time, burst time, completion time, waiting time, and turnaround time (TAT).

2. Shortest Job First (SJF)

Time Complexity: $O(n \log n)$

Sorting processes by burst time takes $O(n \log n)$.

Space Complexity: $O(n)$

Space is needed to store process details and the sorted list.

3. Round Robin (RR)

Time Complexity: $O(n * q)$

Each process may be processed multiple times (up to n times), and each time quantum qqq may require $O(n)$ operations. Thus, the complexity is $O(n * q)$, where qqq is the time quantum.

Space Complexity: $O(n)$

Uses a queue to manage processes, requiring space proportional to the number of processes.

EXPERIMENT – 05

AIM: DEMONSTRATION OF IPC MECHANISMS SUCH AS PIPE and SHARED MEMORY.

DESCRIPTION:

Inter-Process Communication (IPC) mechanisms, like **pipes** and **sharedmemory**, enable data exchange between processes.

- **Pipes** provide a unidirectional communication channel, allowing one process to write and another to read data sequentially.
- **Shared memory** enables multiple processes to access common memory space, facilitating fast data exchange by minimizing copying.

Pipe()

pipe(2)	System Calls Manual	pipe(2)	DESCRIPTION
NAME	pipe, pipe2 - create pipe		pipe() creates a pipe, a unidirectional data channel that can be used for interprocess communication. The array <code>pipefd</code> is used to return two file descriptors referring to the ends of the pipe. <code>pipefd[0]</code> refers to the read end of the pipe. <code>pipefd[1]</code> refers to the write end of the pipe. Data written to the write end of the pipe is buffered by the kernel until it is read from the read end of the pipe. For further details, see pipe(7) .
LIBRARY	Standard C library (<code>libc</code> , <code>-lc</code>)		
SYNOPSIS	<pre>#include <unistd.h> int pipe(int pipefd[2]); #define _GNU_SOURCE /* See feature_test_macros(7) */ #include <fcntl.h> /* Definition of *_ constants */ #include <unistd.h> int pipe2(int pipefd[2], int flags); /* On Alpha, IA-64, MIPS, SuperH, and SPARC/SPARC64, pipe() has the following prototype; see VERSIONS */ #include <unistd.h> struct fd_pair { long fd[2]; }; struct fd_pair pipe(void);</pre>		If <code>flags</code> is 0, then pipe2() is the same as pipe() . The following values can be bitwise ORed in <code>flags</code> to obtain different behavior:
		O_CLOEXEC	Set the close-on-exec (<code>FD_CLOEXEC</code>) flag on the two new file descriptors. See the description of the same flag in open(2) for reasons why this may be useful.
		O_DIRECT (since Linux 3.4)	Create a pipe that performs I/O in "packet" mode. Each write(2) to the pipe is dealt with as a separate packet, and read(2) s from the pipe will read one packet at a time. Note the following points:
			<ul style="list-style-type: none">Writes of greater than <code>PIPE_BUF</code> bytes (see pipe(7)) will be split into multiple packets. The constant <code>PIPE_BUF</code> is defined in <code><limits.h></code>.

Shared Memory

shmget(2)	System Calls Manual	shmget(2)	SHMOP(2)	System Calls Manual	SHMOP(2)
NAME	shmget - allocates a System V shared memory segment		NAME	shmat, shmdt - System V shared memory operations	
LIBRARY	Standard C library (<code>libc</code> , <code>-lc</code>)		LIBRARY	Standard C library (<code>libc</code> , <code>-lc</code>)	
SYNOPSIS	<pre>#include <sys/shm.h> int shmget(key_t key, size_t size, int shmflg);</pre>		SYNOPSIS	<pre>#include <sys/shm.h> void *shmat(int shmid, const void *_Nullable shmaddr, int shmflg); int shmdt(const void *shmaddr);</pre>	
DESCRIPTION	<p><code>shmget()</code> returns the identifier of the System V shared memory segment associated with the value of the argument <code>key</code>. It may be used either to obtain the identifier of a previously created shared memory segment (when <code>shmflg</code> is zero and <code>key</code> does not have the value <code>IPC_PRIVATE</code>), or to create a new set.</p> <p>A new shared memory segment, with size equal to the value of <code>size</code> rounded up to a multiple of <code>PAGE_SIZE</code>, is created if <code>key</code> has the value <code>IPC_PRIVATE</code> or <code>key</code> isn't <code>IPC_PRIVATE</code>, no shared memory segment corresponding to <code>key</code> exists, and <code>IPC_CREAT</code> is specified in <code>shmflg</code>.</p>		DESCRIPTION	<p>shmat() <code>shmat()</code> attaches the System V shared memory segment identified by <code>shmid</code> to the address space of the calling process. The attaching address is specified by <code>shmaddr</code> with one of the following criteria:</p> <ul style="list-style-type: none"> • If <code>shmaddr</code> is NULL, the system chooses a suitable (unused) page-aligned address to attach the segment. 	

shmdt()
shmdt() detaches the shared memory segment located at the address specified by **shmaddr** from the address space of the calling process. The to-be-detached segment must be currently attached with **shmaddr** equal to the value returned by the attaching **shmat()** call.

On a successful **shmdt()** call, the system updates the members of the **shmid_ds** structure associated with the shared memory segment as follows:

- **shm_dtime** is set to the current time.
- **shm_lpid** is set to the process-ID of the calling process.
- **shm_nattch** is decremented by one. If it becomes 0 and the segment is marked for deletion, the segment is deleted.

1. Program to Demonstrate Ordinary Pipe IPC Mechanism

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main() {
    int fd[2];
    if (pipe(fd) == -1) {
        perror("Pipe creation failed");
        exit(EXIT_FAILURE);
    }
    // Fork a child process
    pid_t pid = fork();
    if (pid == -1) {
        perror("Fork failed");
        exit(EXIT_FAILURE);
    }
    if (pid == 0) {
        // Child process
        close(fd[0]); // Close the read end of the pipe
        char message[] = "Hello from the child process!\n";
        write(fd[1], message, sizeof(message));
        close(fd[1]); // Close the write end of the pipe in the child
    } else {
        // Parent process
        close(fd[1]); // Close the write end of the pipe
        char buffer[100];
        read(fd[0], buffer, sizeof(buffer));
        printf("Parent received: %s", buffer);
        close(fd[0]); // Close the read end of the pipe in the parent
    }
    return 0;
}
```

OUTPUT:

```
cbit@cbit-VirtualBox:~$ gedit pipec.c
cbit@cbit-VirtualBox:~$ cc pipec.c
cbit@cbit-VirtualBox:~$ ./a.out
Parent received: Hello from the child process!
cbit@cbit-VirtualBox:~$
```

2. Program to Demonstrate Shared Memory IPC Mechanism.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/shm.h>
#include <sys/ipc.h>
#define SHM_SIZE 1024 // Size of the shared memory
int main() {
    key_t key = ftok("shmfile", 65); // Generate unique key
    int shmid = shmget(key, SHM_SIZE, 0666 | IPC_CREAT); // Create shared memory
    if (shmid == -1) {
        perror("shmget failed");
        exit(1);
    }
    char *shared_memory = (char*) shmat(shmid, NULL, 0); // Attach to shared memory
    if (fork() == 0) { // Child Process
        sleep(1); // Ensuring parent writes first
        printf("Child reads: %s\n", shared_memory); // Read data from shared memory
        shmdt(shared_memory); // Detach from shared memory
    } else { // Parent Process
        strcpy(shared_memory, "Hello from parent!"); // Write data to shared memory
        printf("Parent wrote to shared memory.\n");
        wait(NULL); // Wait for child to finish
        shmctl(shmid, IPC_RMID, NULL); // Destroy shared memory
    }
    return 0;
}
```

OUTPUT:

Parent Process: Creates a shared memory segment and writes data to it.

Child Process: Reads the data from the shared memory.

```
Parent wrote to shared memory.
Child reads: Hello from parent!
```