BASIC UNIX COMMANDS

Commands:

1. Command: pwd

   Description: Print the current working directory

   Output: | home | acecse 42 | t.

2. Command: date

   Description: display the current date

   Output: The Jan 25 14:58:24
   IST 2018

3. Command: Man

   Description: Give the entire manual for the specified Command

   Output: What manual page do you want?

4. Command: Bc

   Description: The calculator

   Output: bc 1.06.95

5. Command: Cal

   Description. The Cal produces a calendar of the month.

   Output: January 2018.

```
 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
```

6. Command: exit

   Description used to logout from the current user

   Output: logout
   There are stopped jobs

7. Command: tput clear

   Description: used to clear screen

   Output: [cse @ localhost r] $

8. Command: who

   Description: display who are all logged to the system.

9. Command: Who am i

   Description: display the text typed and system into.

   Output: 2018-01-25 14:16
   (192.168.40.129).

10. Command: echo text

    Description: display the text typed from the keyboard

    Output: thanu

11. Command: mkdir name

Description: create a new directory

Output: yam momo kovi cse.

12. Command:a. cd-dir. name.

Description: Change to directory

Output: [cse @ localhost cse] $

b. Command: cd

Description: Change to our home directory

Output: [cse @ localhost cse 1] $

c. Command: cd.

Description: Change to ponent directory

Output: [cse @ localhost N] $

d. Command: cd sub-dis-name.

Description. Change to subdirectory

Output: [cse @ localhost cse 1] $

13. Command: Cat > filename

Description: create a new file & type the text

Output: [cse @ localhost cse] $ ls Durga.

Hi……

14. Command: cat filename

Description: display all the content is file

Output: hello

15. Command: Vc filename

Description: View and modify the content of file

Output: hi... how are you?

16.Command: touch f1 f2 f3

Description: create empty files

Output: hi...

17. Command: cat old file 1 old file 2 > new file

Description Concatenate the file concatenate

Output: hi... hello ......

18. a. Command: Is

Description: list all the files in the directory

Output: a.out f1 f2 f3 w.c y.c

b. Command: Is-a

Description: list the content of the current directory in long term

Output: cse fileswp swp cse1 gname2 c.

c.Command: ls-al.

Description: list the content of the current directory is long term form with file

Output: -rw-r..r... |cse | ab20 cse | ab20 121 June 200 wo kstore drw xr-xr- 4 cse | ab20 cseab20 409 Max6 2012 Mozilla

d. Command: Is directory

Description: list the content of directory

Output: Sankar guna

e. Command: Is subdirectory

Description: list the content of subdirectory Output: cse cse1 cse2 cselab

f.Command: Is filename

Description: Check whether the file exit or not Output: f2

g. Command: Is character

Description: display the file which ends with the specified character Output: f1 f2 f3 f4 lab f5

h. Command: Is character *

Description: list the file start with specified character Output: cse lab20 new sab

i. Command: ls – ld

Description: list all the directories of user Output: 1 0 3 4 5 8 9

19. Command: my filename new filename

Description: rename file to new filename

Output: hi ... Sundar

20. Command: mv dirname new dirname

Description: rename directory to new directory name

Output: cse cse1 cse2 cse | ab20 f1 f2 21.

21.command: mv file.

Description: more file into existing directory

Output: Durga Lakshmi cseb

22. Command: Cp- r dir new dir

Description: copy file to new file

Output: Durga Lakshmi cse is new2

23. Command: cp file new file

Description: Copy file to new file

Output. [cse @ localhost cse] $ cat new2

24.Command: rm filename

Description: remove (delete file from directory) Output: bala f1 f4 sathish ragle

25. Command: rm-i filename

Description: ask the user if this wants to delete the specified Output: bala f1 f4 Rajkumar

26. Command: rmdir dirname Descriptions: remove the empty directory Output: bak sathish Rajkumar

27. Command: rm-r directory

Description: remove the directory & contents Output: bala Rajkumar is cse.

28. Command: head filename

Description: display the 1st 10 lines of file.

Output: aicha

Anand.

29. a. command: wc - w filename

Description: display the no. of character in a file

Output: 3 file

b. command: wc-l filename

Description: display the no. of lines in a file

Output: 3 fill

30. Command: cmp file1 file2

    Description: compare the content 2 files

    Output: bme hi

31. command diff file1 file2

    Description: compare the content of 2 files where they differ

    Output: 1,3c, 11

32. Command: find filename

    Description: to calculate the file in a dir

    Output: 1 file

33. Command: history

    Description: display the commands typed

    Output: 1067 exit1079 Cal 3 2064

34. Command: not-k filename

    Description: display the content in ascending order

    Output: abc bcd

35. Command ch mod-c filename

Description: change the mode of file

Output: -r-r-r ... I cse cse 25 Jan 2019:39 vl3

IMPLEMENTATION OF FILE MANAGEMENT SYSTEM - OPEN, READ, WRITE ON A FILE

PROGRAM :

```
#include<unisd.h>
#include<fcnt l.h>
#include<stri ng.h>
#include<stdi o.h>
int main()
{
int fd[2];
char buf1[23]="JUST A TEXT";
char buf2[100]; fd[0]=open("SAMPLE",O_RDWR);
fd[1]=open("SAMPLE",O_RDWR);
write(fd[0],buf1,strlen(buf1));
printf("\nENTER YOUR TEXT NOW:");
scanf("%s",&buf1);
write(fd[0],buf1,strlen(buf1)) write(1,buf2,read(fd[1],buf2,sizeof(buf 2)));
close(fd[0]);
close(fd[1])rintf("\n");
return 0;
}
```

OUTPUT :

```
ENTER YOUR TEXT NOW:
OPERATING SYSTEM
JUST A TEXT
OPERATING SYSTEM
```

IMPLEMENTATION OF FILE MANAGEMENT IN UNIX OPERATING SYSTEM USING OPEN(),CLOSE(),READ(),WRITE() SYSTEM CALLS.

PROGRAM:

```
#include<fcntl.h>
#include<stdio.h>
main()
{
int fp,exitstatus;
char chr='A';
int pid;
pid=fork();
if(pid==0)
{
fp=open("\t3.c",O_RDONLY);
printf("\nIN CHILD CHR IS%c\n",chr);
chr='B';
write(fp,&chr,2);
printf("\nCHILD PROCESSOR_ID:%d",pid);
printf("\nCHILD CHR AFTER CHANGE IS:%c",chr);
printf("\nCHILD TERMINATED");
close(fp);
}
else
{
wait(&exitstatus);
fp=open("\t3.c",O_RDONLY);
printf("\nPARENT PROCESSORID:%d",pid);
read(fp,&chr,2);
printf("\nCHR AFTER PARENT READ IS %c\n",chr);
close(fp)
```

OUTPUT:

```
IN CHILD CHR IS:A CHILD PROCESSOR_ID:0
CHILD CHR AFTER CHANGE IS:B
CHILD TERMINATED PARENT PROCESSOR ID:30450
CHR AFTER PARENT READ IS:A
```

IMPLEMENTATION OF FILE MANAGEMENT IN UNIX OPERATING SYSTEM TO READ A FILE IN REVERSE USING LSEEK() SYSTEM CALL.

PROGRAM

```
#include<fcntl.h>
#include<unistd.h>
int main(intargc,char**argy)
{
char buf; int size,fd;
fd=open(argy[1],O_RDONLY);
size=lseek(fd,-1,SEEK_END);
while(size-->=0)
{
read(fd,&buf,1);
write(STDOUT_FILENO,&buf,1);
lseek(fd,-2,SEEK_CUR);
}
}
```

OUTPUT:

```
[acecse5@localhost ~]$ vi file
SACHIN TENDULKAR
:wq
[acecse5@localhost ~]$ cc reverse.c
[acecse5@localhost ~]$ ./a.out
RAKLUDNET NIHCAS
```

IMPLEMENTATION OF DIRECTORY MANAGEMENT SYSTEM CALLS.

PROGRAM :

```
#include<stdio.h>
#include<dirent.h>
int main(int argc,char*argy[])
{
DIR *dir;
struct dirent *directory; dir=opendir(argy[1]);
while((directory=readdir(dir))!=NULL)
printf("%d%s\n",directory->d_ino,directory->d_name);
closedir(dir);
}
```

OUTPUT:

```
[acecse5@localhost ~]$ mkdir os
[acecse5@localhost ~]$ cd os
[acecse5@localhost gd]$ mkdir se
[acecse5@localhost gd]$ cd ..
[acecse5@localhost ~]$ cc dirmanage.c
[acecse5@localhost ~]$ ./a.out
5636321...
5636318...
5636326 f1
5636348 tm1
```

IMPLEMENTATION OF PROCESS MANAGEMENT CALLS - FORK(),WAIT(),EXECLP()

PROGRAM:

```
#include<stdio.h>
#include<unistd.h>
#include<sys/wait.h>
#include<sys/types.h>
#include<stdlib.h>
int main()
{
int pid;
pid=fork();
if(pid<0)
{
printf("\nFORK FAILED\n");
exit(-1);
}
else if(pid==0)
{
execlp("/bin/ls","is","-1",NULL);
}
else
{
wait(NULL);
printf("\nCHILD COMPLETE\n");
exit(0);
}
}
```

OUTPUT:

```
[acecse5@localhost ~]$ cc sysmanage.c
[acecse5@localhost ~]$ ./a.out
2a.c
filesyscalls.c
reverse.c
dirmanage.c
CHILD COMPLETE
```

IMPLEMENTATION OF PROCESS MANAGEMENT SYSTEM CALLS - FORK() , EXIT()

PROGRAM :

```
#include<stdio.h>
#include<sys/types.h>
int main(void)
{
pid_t pid;
printf("BEFORE FORK\n");
pid=fork();
if(pid>0)
{
sleep(1);
printf("PARENT_PID:%d RPID:%d,CHILD PID:%d \n",getpid(),getppid());
}
else if(pid==0)
printf("CHILD_PID:%d RPID:%d\n",getpid(),getppid()); else
{
printf("FORK ERROR\n");
}
printf("BOTH PROCESS CONTINUE FROM HERE\n");
}
```

OUTPUT:

```
[acecse5@localhost ~]$ cc processmanage.c
[acecse5@localhost ~]$ ./a.out
BEFORE FORK CHILD_PID:28283
RPID:28282
BOTH PROCESS CONTINUE FROM HERE PARENT_PID:28282
RPID:9144
CHILD PID:28282
BOTH PROCESS CONTINUE FROM HERE
```

IMPLEMENTATION OF PROCESS MANAGEMENT CALLS KILL(),SIGNAL()

PROGRAM:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/wait.h>
#include<signal.h>
pid_t pid;
int main(int agrc,char **argv)
{
int i,status;
void death_handler(int signo);
signal(SIGCHLD,death_handler);
signal(SIGALRM,death_handler);
switch(pid=fork())
{
case -1:printf("FORKERROR\n");
case 0:execvp(argv[1],&argv[1]);perror("exec");
          break;
default: alarm(5);
pause();
printf("PARENT DIES\n");
}
exit(1);
}
void death_handler(int signo)
{
int status; signal(signo,death_handler);
switch(signo)
{
case SIGCHLD:
waitpid(-1,&status,0);
printf("CHILD DIES:exit status%d\n",WEXITSTATUS(status));
break;
case SIGALRM:
if(kill(pid,SIGTERM)==0)
fprintf(stderr,"5 SECONDS OVER CHILD KILLED\n");
}
}
```

OUTPUT:

```
[acecse5@localhost ~]$ cc promanage.c
[acecse5@localhost ~]$ ./a.out
CHILD DIES:exit
status 0
PARENT DIES.
```

IMPLEMENTATION OF SCHEDULING ALGORITHM N (FCFS ALGORITHM)

PROGRAM :

```
#include<stdio.h>
struct fcfs
{
int pno;
int btime;
};
main()
{
int n,p,i,wait;
float avg,avg1,avg2,tc;
struct fcfs f[20];
printf("ENTER THE NUMBER OF PROCESS");
scanf("%d",&n);for(i=0;i<n;i++)
{
printf("\n ENTER THE PROCESS NUMBER");
scanf("%d",&f[i].pno); printf("ENTER THE BURSTTIME");
scanf("%d",&f[i].btime);
}
printf("THE FOLLOWING ARE THE PROCESS DETAILS\n");
printf("PROCESS TIME\t BURST TIME \tWAITING TIME \tTURN AROUND TIME\n");
avg=0; avg1=0; avg2=0; wait=0; tc=0;
for(i=0;i<n;i++)
{
printf("\t %d \t\t %d\t\t",f[i].pno,f[i].btime);
printf("\t %d \t\t%d \\t",wait,wait+f[i].btime);tc=tc+(wait+f[i].btime);
avg=avg+wait;
wait=wait+f[i].btime;
}
avg1=avg/n;avg2=tc/n;
printf("AVERAGE WAITING TIME FOR THE ACCESS IS %f\n",avg1);
printf("AVERAGE TURN AROUND TIME FOR THE PROCESS IS%f\n",avg2);
}
```

OUTPUT:

```
[acecse5@localhost ~]$ cc fcfs.c
[acecse5@localhost ~]$ ./a.out
ENTER THE NUMBER OF PROCESS 3
ENTER THE PROCESS NUMBER 1
ENTER THE BURST TIME 10
ENTER THE PROCESS NUMBER 2
ENTER THE BURST TIME 20
ENTER THE PROCESS NUMBER 3
ENTER THE BURST TIME 30
THE FOLLOWING ARE THE PROCESS DETAILS
PROCESS TIME   BURST TIME          WAITING TIME   TURN AROUND TIME
   1               10                   0               1
   2               20                   10              30
   3               30                   30              60
AVERAGE WAITING TIME FOR THE ACCESS IS 13.33
AVERAGE TURN AROUND TIME FOR THE PROCESS 33.33
```

IMPLEMENTATION OF SCHEDULING ALGORITHM (SHORTEST JOB FIRST)

PROGRAM:

```
#include<stdio.h>
struct s
{
int pno;
int btime;
}s[20];
main()
{
int i,j,n,wait,ptemp,ptemp1; floatavg,avg1,avg2,tc;
char ch;
printf("ENTER THE NUMBER OF PROCESS:");
scanf("%d",&n);
ptemp=0; ptemp1=0; wait=0;avg=0; avg1=0; avg2=0; tc=0;for(i=0;i<n;i++)
{
printf("\nENTER THE PROCESS NUMBER:");
scanf("%d",&s[i].pno); printf("\nENTER THE BURST TIME:");
scanf("%d",&s[i].btime);
}
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
if(s[i].btime<s[j].btime)
{
ptemp1=s[i].btime;
ptemp=s[i].pno;
s[i].btime=s[j].btime;
s[i].pno=s[j].pno;
s[i].btime=ptemp1;
s[j].pno=ptemp;
}
}
}
printf("\n THE FOLLOWING ARE THE PROCESS DETAILS:\n");
printf("\n PROCESS NO \t BURST TIME\tWAITING TIME\t TURN AROUND TIME\n");
for(i=0;i<n;i++)
{
printf("\n%d\t\t%d\t\t",s[i].pno,s[i].btime);
printf("%d\t\t%d\n",wait,wait+s[i].btime);
tc=tc+(wait+s[i].btime);
avg=avg+wait; wait=wait+s[i].btime;
}
avg1=avg/n;
avg2=tc/n;
printf("\n THE AVERAGE WAITING TIME OF PROCESS IS:%f",avg1);
printf("\n THE AVERAGE TURN AROUND TIME OF PROCESS IS:%f\n",avg2);
}
```

OUTPUT:

```
[acecse22@localhost expt4b]$cc sjf.c
[acecse222localhost expt4b]$./a.out
ENTER THE NUMBER OF PROCESS:3
ENTER THE PROCESS NUMBER:1
ENTER THE BURST TIME:2
ENTER THE PROCESS NUMBER:2
ENTER THE BURST TIME:3
ENTER THE PROCESS NUMBER:3
ENTER THE BURST TIME:3
THE FOLLOWING ARE THE PROCESS DETAILS:
PROCESS NO      BURST TIME                WAITING TIME     TURN AROUND TIME
     1              2                          0                   2
     2              3                          2                   5
     3               3                         5                   8
THE AVERAGE WAITING TIME OF PROCESS IS:2.3333
THE AVERAGE TURN AROUND TIME OF PROCESS IS:5.0000
```

IMPLEMENTATION OF SCHEDULING ALGORITHM (PRIORITY QUEUE SCHEDULING)

PROGRAM:

```
#include<stdio.h>
struct pr
{
int prino,pno;
int btime;
};
main()
{
btemp=0;
ptemp=0;
prtemp=0;
wait=0;
avg=int n,i,j,wait,ptemp;
float avg,avg1,avg2,tc;
int prtemp,btemp;
char c;
struct pr p[15];
printf("\nPRIORITY SCHEDULING\t\n");
printf("\nENTER THENUMBER OF PROCESS: ");
scanf("%d"0; avg1=0; avg2=0; tc=0;for(i=0;i<n;i++)
{
printf("\nENTER THE PROCESS NUMBER: ");
scanf("%d",&p[i].pno);
printf("\nENTER THE PRIORITY NUMBER: ");
scanf("%d",&p[i].prino);
printf("ENTER THE BURST TIME: ");
scanf("%d",&p[i].btime);
}
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
if(p[i].prino<p[j].prino)
{
prtemp=p[i].prino;
btemp=p[i].btime;
ptemp=p[i].pno;
p[i].prino=p[j].prino;
p[i].btime=p[j].btime;
p[i].pno=p[j].pno;
p[j].prino=prtemp;
p[j].btime=btemp;
p[j].pno=pte mp;
}
}
}
printf("THE FOLLOWING ARE THE PROCESS DETAILS: \n");
printf("PROCESS NO\tPRIORITY NO\tBURST TIME\t WAITING TIME \tCOMPLETE TIME\n");
for(i=0;i<n;i++)
{
printf("\n%d\t\t%d\t\t%d",p[i].pno,p[i].prino,p[i].btime);
printf("\t\t%d\t\t%d",wait,wait+p[i].btime);
tc=tc+(wait+p[i].btime);
avg=avg+wait;
wait=wait+p[i].btime;
}
avg1=avg/n;avg2=tc/n;
printf("\nAVERAGE WAITING TIME FOR THE PROCESS IS:%f\n",avg1);
printf("\nAVERAGE TURN AROUND TIME FOR THE PROCESS IS:%f\n",avg2);
}
```

OUTPUT:

```
PRIORITY SCHEDULING ENTER THE NUMBER OF PROCESS:4
ENTER THE PROCESS NUMBER:1
ENTER THE PRIORITY NUMBER:2
ENTER THE BURST TIME:1
ENTER THE PROCESS NUMBER:2
ENTER THE PRIORITY NUMBER:1
ENTER THE BURST TIME:3
ENTER THE PROCESS NUMBER:3
ENTER THE PRIORITY NUMBER:4
ENTER THE BURST TIME:6
ENTER THE PROCESS NUMBER:4
ENTER THE PRIORITY NUMBER:3
ENTER THE BURST TIME:2
THE FOLLOWING ARE THE PROCESS DETAILS:

PROCESS NO   PRIORITY NO   BURST TIME   WAITING TIME   COMPLETE TIME
     2             1            3             0               3
     1             2            21            3               24
     4             3            2             24              26
     3             4            6             26              32
AVERAGE TURN AROUND TIME FOR THE PROCESS IS:13.2555
```

IMPLEMENTATION OF SCHEDULING (ROUND ROBIN ALGORITHM)

PROGRAM:

```
#include<stdio.h>
struct rr
{
int pno,btime,com,wt,cal;
char st;
};
main()
{
int n,i,j,wait,temp,ta; float avg,avg1;
struct rr r[4];
printf("\n ENTER THE NUMBER OF PROCESS");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("\n ENTER THE PROCESS NUMBER:");
scanf("%d",&r[i].pno);
printf("\n ENTER THE BURST TIME:");
scanf("%d",&r[i].btime); r[i].st='a';r[i].wt=0;
r[i].com=0;
r[i].cal=0;
}
printf("\n THE FOLLOWING ARE THE PROCESSDETAILS");
printf("\n TIME QUANTUM=20");
printf("\n PROCESS TIME \t BURST TIME \t WAITING TIME \t COMPLETETIME");
avg=0;
wait=0;
temp=4;
for(j=0;j<n;j++)
{
while(r[j].st=='a')
{
for(i=0;i<n;i++)
{
if(r[i].st=='a')
if(r[i].btime>4)
{
r[i].wt=wait;
r[i].cal=r[i].cal+(wait-r[i].com);
r[i].com=wait+temp;
printf("\n%d\t\t%d\t\t%d\t\n",r[i].pno,temp,wait+temp);
wait=wait+temp;
r[i].btime=r[i].btime-temp;
if(r[i].btime<0)
{
r[i].st='p';
}
}
else
{

r[i].wt=wait; r[i].cal=r[i].cal+(wait-r[i].com);
r[i].com=wait+r[i].btime;
printf("\n %d \t\t %d \t\t",r[i].pno,r[i].btime);
printf("%d \t\t %d \t\t\n",wait,wait+r[i].btime);
wait=wait+r[i].btime;r[i].st='p';
}
}
}
}
for(i=0;i<n;i++)
{
avg=avg+r[i].cal;
avg1=avg1+r[i].com;
}
avg=avg/n; avg1=avg1/n;
printf("\n THE AVERAGE WAITING TIME IS %f ms",avg);
printf("\n THE AVERAGE TURN AROUND TIME IS is %f ms",avg1);
}
```

OUTPUT:

```
[acecse5@localhost ~]$ cc rr.c
[acecse5@localhost ~]$ ./a.out
ENTER THE NUMBER OF PROCESS:4
ENTER THE PROCESS NUMBER:1
ENTER THE BURST TIME:53
ENTER THE PROCESS NUMBER:2
ENTER THE BURST TIME:17
ENTER THE PROCESS NUMBER:3
ENTER THE BURST TIME:68
ENTER THE PROCESS NUMBER:4
ENTER THE BURST TIME:24
THE FOLLOWING ARE THE PROCESS DETAILS TIME QUANTUM=20

PROCESS TIME   BURST TIME    WAITING TIME    COMPLETE TIME
     1              53           81               134
     2              17           20               37
     3              68           94               162
     4              24           97               121
THE AVERAGE WAITING TIME IS:7.3
THE AVERAGE TURN AROUND TIME IS:118.
```

5A IPC MECHNASIUM-PIPES

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main() {
int pipefd[2];
char buffer[80];
// Create a pipe
if (pipe(pipefd) == -1)
{
perror("pipe");
exit(EXIT_FAILURE);
}
// Create a child process
pid_t pid = fork();
if (pid < 0) {
perror("fork");
exit(EXIT_FAILURE);
}
// Parent process (writer)
if (pid > 0)
{
close(pipefd[0]); // Close the reading end in the parent
const char* message = "Hello from the parent process!\n";
ssize_t bytes_written = write(pipefd[1], message, sizeof(message));
if (bytes_written == -1)
{
perror("write");
exit(EXIT_FAILURE);
}
printf("Parent: Sent message - %s\n", message);
}
// Child process (reader)
else
{
close(pipefd[1]); // Close the writing end in the child
ssize_t bytes_read = read(pipefd[0], buffer, sizeof(buffer));
if (bytes_read == -1)
{
perror("read");
exit(EXIT_FAILURE);
}
else if (bytes_read == 0)
{
printf("Child: End of pipe reached (no data).\n");
}
else
{
```

```
printf("Child: Received message - %s\n", buffer);
}
}
// Close the remaining pipe end (if not already closed)
close(pipefd[0]);
close(pipefd[1]);
return 0;
}
```

OUTPUT:

```
Parent: Sent message - Hello from the parent process!
Child: Received message - Hello from the parent process!
```

5B IPC -FIFO

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#define FIFO_NAME "myfifo"
int main()
{
char buffer[80];
int fd;
// Create the FIFO (if it doesn't exist)
mkfifo(FIFO_NAME, 0666); // Create with read/write permissions for all
// Decide if process will be a writer or reader based on command line args
if (argc == 2 && strcmp(argv[1], "writer") == 0)
{
// Writer process
fd = open(FIFO_NAME, O_WRONLY); // Open for writing only
if (fd == -1)
{
perror("open");
exit(EXIT_FAILURE);
}
const char* message = "Hello from the writer process!\n";
ssize_t bytes_written = write(fd, message, sizeof(message));
if (bytes_written == -1) {
perror("write");
exit(EXIT_FAILURE);
}
printf("Writer: Sent message - %s\n", message);
close(fd);
}
else if (argc == 2 && strcmp(argv[1], "reader") == 0)
{
// Reader process
fd = open(FIFO_NAME, O_RDONLY); // Open for reading only
if (fd == -1)
{
perror("open");
exit(EXIT_FAILURE);
}
ssize_t bytes_read = read(fd, buffer, sizeof(buffer));
if (bytes_read == -1)
{
perror("read");
exit(EXIT_FAILURE);
}
else if (bytes_read == 0)
{
```

```
printf("Reader: End of pipe reached (no data).\n");
}
else
{
printf("Reader: Received message - %s\n", buffer);
}
close(fd);
}
else
{
fprintf(stderr, "Usage: %s [writer|reader]\n", argv[0]);
exit(EXIT_FAILURE);
}
return 0;
}
```

OUTPUT:

Scenario 1: Running Writer First
Writer Terminal:

```
   Writer: Sent message - Hello from the writer process!
```

Reader Terminal (after running the writer):

```
Reader: Received message - Hello from the writer process
```

Scenario 2: Running Reader First
Reader Terminal:

```
Reader: End of pipe reached (no data).
```

Writer Terminal (after running the reader):

```
Writer: Sent message - Hello from the writer process!
```

5C MESSAGE QUEUE IPC

PROGRAM:

1. Message Structure Definition (common.h):

```
#ifndef COMMON_H
#define COMMON_H
struct my_msg
{
long msg_type;
char some_text[100]; // Adjust size as needed
};
#endif
```

2. Sender Process (sender.c):

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include "common.h" // Include the header file
#define MAX_TEXT 100
int main()
{
int msgid;
struct my_msg some_data;
int running = 1;
key_t key = ftok("/tmp/myqueue", 65); // Replace with desired key generation method
// Get message queue ID (create if it doesn't exist)
msgid = msgget(key, 0666 | IPC_CREAT);
if (msgid == -1) {
perror("msgget");
exit(1);
}
while (running) {
printf("Enter message (or 'end' to quit): ");
fgets(some_data.some_text, MAX_TEXT, stdin);
some_data.some_text[strcspn(some_data.some_text, "\n")] = '\0'; // Remove trailing newline
// Check for end message
if (strncmp(some_data.some_text, "end", 3) == 0) {
running = 0;
} else {
some_data.msg_type = 1; // Set message type (e.g., 1 for request)
// Send message to queue
if (msgsnd(msgid, &some_data, strlen(some_data.some_text) + 1, 0) == -1) {
perror("msgsnd");
exit(1);
}

printf("Message sent!\n");
}
}
return 0;
}
```

3. Receiver Process (receiver.c):

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include "common.h" // Include the header file
#define MAX_TEXT 100
int main() {
int msgid;
struct my_msg some_data;
long int msg_to_rec = 0; // Any message type
key_t key = ftok("/tmp/myqueue", 65); // Replace with desired key generation method
// Get message queue ID (create if it doesn't exist)
msgid = msgget(key, 0666 | IPC_CREAT);
if (msgid == -1) {
perror("msgget");
exit(1);
}
while (1) {
// Receive message from queue
if (msgrcv(msgid, &some_data, MAX_TEXT, msg_to_rec, 0) == -1) {
perror("msgrcv");
exit(1);
}
printf("Received message: %s\n", some_data.some_text);
// Check for end message
if (strncmp(some_data.some_text, "end", 3) == 0) {
break;
}
}
return 0;
}
```

OUTPUT:

Sender Process:

```
Enter message (or 'end' to quit): Hello, world!
Message sent!
Enter message (or 'end' to quit): This is another message.
Message sent!
Enter message (or 'end' to quit): end
Message sent!
```

Receiver Process:

```
Received message: Hello, world!
Received message: This is another message.
Received message: end
```

5D SHARED MEMORY

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <fcntl.h>  // For O_CREAT and O_RDWR flags
#define SHM_SIZE 1024
#define SHM_KEY_FILE "shared_memory.txt"  // File for generating key
int create_shared_memory(key_t key, size_t size) {
int shmid;
// Create the shared memory segment (if it doesn't exist)
if ((shmid = shmget(key, size, IPC_CREAT | IPC_EXCL | 0666)) < 0) {
if (errno == EEXIST) {
// Segment already exists, attach to it
shmid = shmget(key, size, 0);
if (shmid < 0) {
perror("shmget");
exit(1);
}
} else {
perror("shmget");
exit(1);
}
}
return shmid;
}
void* attach_shared_memory(int shmid) {
void *data;
// Attach the shared memory segment to the process's address space
if ((data = (char *)shmat(shmid, NULL, 0)) == (char *)-1) {
perror("shmat");
exit(1);
}
return data;
}
void detach_shared_memory(void *data) {
// Detach the shared memory segment from the process's address space
if (shmdt(data) < 0) {
perror("shmdt");
exit(1);
}
}
int main(int argc, char *argv[]) {
key_t key = ftok(SHM_KEY_FILE, 'X');  // Generate unique key using filename and char
int shmid;
char *data;
int mode = 0;  // 0 for reader, 1 for writer
if (argc > 1) {
mode = (strcmp(argv[1], "-w") == 0) ? 1 : 0;
}
// Create or attach to the shared memory segment
shmid = create_shared_memory(key, SHM_SIZE);
// Attach the shared memory segment
data = (char *)attach_shared_memory(shmid);
if (mode == 1) {  // Writer process
printf("Writer process attached to shared memory segment.\n");
// Write data to the shared memory
strcpy(data, "Hello from the writer process!");
printf("Writer process wrote data to shared memory: %s\n", data);
} else {  // Reader process
printf("Reader process attached to shared memory segment.\n");

// Read data from the shared memory
printf("Reader process read data from shared memory: %s\n", data);
}
// Detach from the shared memory segment
detach_shared_memory(data);
return 0;
}
```

OUTPUT:

Writer Process Output:

```
Writer process attached to shared memory segment.
Writer process wrote data to shared memory: Hello from the writer process!
```

Reader Process Output:

```
Reader process attached to shared memory segment.
Reader process read data from shared memory: Hello from the writer process!
```

6A PAGING MEMORY TECH

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
main()
{
int ms, ps, nop, np, rempages, i, j, x, y, pa, offset;
 int s[10], fno[10][20];
clrscr();
printf("\nEnter the memory size -- ");
scanf("%d",&ms);
printf("\nEnter the page size -- ");
scanf("%d",&ps);
nop = ms/ps;
printf("\nThe no. of pages available in memory are -- %d ",nop);
printf("\nEnter number of processes -- ");
 scanf("%d",&np);
rempages = nop;
for(i=1;i<=np;i++)
{
printf("\nEnter no. of pages required for p[%d]-- ",i);
 scanf("%d",&s[i]);
if(s[i] >rempages)
{
printf("\nMemory is Full");
break;
}
rempages = rempages - s[i];
printf("\nEnter pagetable for p[%d] --- ",i);
 for(j=0;j<s[i];j++)
scanf("%d",&fno[i][j]);
}
printf("\nEnter Logical Address to find Physical Address ");
printf("\nEnter process no. and pagenumber and offset -- ");
scanf("%d %d %d",&x,&y, &offset);
if(x>np || y>=s[i] || offset>=ps)
printf("\nInvalid Process or Page Number or offset");
else
{
pa=fno[x][y]*ps+offset;
printf("\nThe Physical Address is -- %d",pa);
}
getch();
}
```

OUTPUT:

```
Enter the memory size – 1000 Enter the page size -- 100
The no. of pages available in memory are -- 10
Enter number of processes -- 3
Enter no. of pages required for p[1]-- 4
Enter pagetable for p[1] --- 8 6
9
5
Enter no. of pages required for p[2]-- 5
Enter pagetable for p[2] --- 1 4 5 7 3
Enter no. of pages required for p[3]-- 5
Memory is Full
Enter Logical Address to find Physical Address Enter process no. and pagenumber and offset  -- 2
3
60
The Physical Address is --  760
```

6B SEGMENTATION

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
struct list
{
int seg;
int base;
int limit;
struct list *next;
} *p;
void insert(struct list *q,int base,int limit,int seg)
{
if(p==NULL)
{
p=malloc(sizeof(Struct list));
p->limit=limit;
p->base=base;
p->seg=seg;
p->next=NULL;
}
else
{
while(q->next!=NULL)
{
Q=q->next;
Printf(“yes”)
}
q->next=malloc(sizeof(Struct list));
q->next ->limit=limit;
q->next ->base=base;
q->next ->seg=seg;
q->next ->next=NULL;
}
}
int find(struct list *q,int seg)
{
while(q->seg!=seg)
{
q=q->next;
}
return q->limit;
}
int search(struct list *q,int seg)
{
while(q->seg!=seg)
{
q=q->next;
}
return q->base;
}
```

```
main()
{
p=NULL;
int seg,offset,limit,base,c,s,physical;
printf(“Enter segment table/n”);
printf(“Enter -1 as segment value for termination\n”);
do
{
printf(“Enter segment number”);
scanf(“%d”,&seg);
if(seg!=-1)
{
printf(“Enter base value:”);
scanf(“%d”,&base);
printf(“Enter value for limit:”);
scanf(“%d”,&limit);
insert(p,base,lmit,seg);
}
}
while(seg!=-1)
printf(“Enter offset:”);
scanf(“%d”,&offset);
printf(“Enter bsegmentation number:”);
scanf(“%d”,&seg);
c=find(p,seg);
s=search(p,seg);
if(offset<c)
{
physical=s+offset;
printf(“Address in physical memory %d\n”,physical);
}
else
{
printf(“error”);
}
}
```

OUTPUT:

```
[examuser56@localhost ~]$ cc seg.c
[examuser56@localhost ~]$ ./a.out
Enter segment table
Enter -1 as segmentation value for termination
Enter segment number:1
Enter base value:2000
Enter value for limit:100
Enter segment number:2
Enter base value:2500
Enter value for limit:100
Enter segmentation number:-1
Enter offset:90
Enter segment number:2
Address in physical memory 2590
[examuser56@localhost ~]$ ./a.out
Enter segment table
Enter -1 as segmentation value for termination
Enter segment number:1
Enter base value:2000
Enter value for limit:100
Enter segment number:2
Enter base value:2500
Enter value for limit:100
Enter segmentation number:-1
Enter offset:90
Enter segment number:1
Address in physical memory 2090
```

IMPLEMENTATION OF PAGE REPLACEMENT ALGORITHM..

PROGRAM :

```
#include<stdio.h>
int n,pg[30], fr [10];
voidfifo();
void optimal();
void lru();
void main()
{
int i,ch;
printf("\n Enter total number of pages :”);
scanf("%d",&n);
printf("\n Enter sequence:");
for (i=0;i<n;i++)
scanf("%d",&pg[i]);
do
{
printf(" MENU \n");
printf("\n 1) FIFO");
printf("\n 2)OPTIMAL ");
printf("\n 3)LRU");
printf("\n 4)EXIT");
printf("\n ENTER YOUR CHOICE: ");
scanf("%d",&ch); switch (ch)
{
case 1: fifo();
break;
case 2:optimal();
break;
case 3:lru();
break;
}
}
while (ch !=4);
getchar();
}
void fifo()
{
int i, f,r,s,count, flag,num,psize: f=0;r=0; s=0;
flag=0; count=0:
printf("\n Enter size of page frame:"); scanf("%d",&psize);
for (i=0;i<psize; i++)
{
fr[i]=-1;
}
while (s<n)
{
flag=0; num=pg[s];
for (i=0;i<psize;i++)
{
```

```
if(num==fr[i])

{
s++;
flag=1; break;
}
}
if (flag==0)
{
if (r<psize )
{
fr[r]=pg[s];
r++;
s++;
count++;
}
else
{
if(f<psize)
{
fr [f]=pg[s];
s++;
f++;
count++;
}
else f=0;
}
}
printf("\n");
for (i=0;i<psize; i++)
{
printf("%d \t", fr[i]);
}
}
printf("\n page fault = %d",count); getchar();
}
void optimal()
{
int count [20],i , j , k , fault , f , flag , temp , current , c , dist ,max , m , cnt , p , x; fault = 0;
dist=0; k=0;
printf("\n Enter Frame sizes :”); scanf("%d",&f);for (i=0;i<f;i++)
{
count[i] = 0; fr[i]=-I;
}
for (i=0;i<n;i++)
{
flag=0; temp=pg[i]; for(j=0; j<f;j++)
{
if(temp==fr[j])
{
flag=1; break;
}
}
```

```
if((flag==0) && (K<f)
}
{

fault++; fr[k]=temp; k++;
}
else if((flag==0)&& (k==f))
{
fault++;
for (cnt=0;cnt<f;cnt++)
{
current=fr [cnt]; for (c=i;c<n;c++)
{
if (current !=pg[c]) count[cnt]++;else break;
}
}
max=0;
for (m=0;m<f:m++)
{
if (count[m]>max)
{
max=count[m]; p=m;
}
}
fr[p]=temp;
}
printf("\n");
for (x=0; x<f;x++)
{
printf("%d\t", fr[x]);
}
}
printf("\n Total number of fault=%d",fault); getchar();
}
void lru()
{
int count[10], i , j , k , fault, flag , tem , curren , c , dist , max ,m ,cnt , p , x ; fault=0;
dist=0; k=0;
printf("\n Enter frame size:"); scanf("%d",&f);for (i=0;i<f;i++)
{
count[i]=0; fr[i]=-1;
}
for (i=0;i<n;i++)
{
flag=0; temp=pg[i]; for(j=0;j<f;j++)
{
if(temp==fr[j])
{
flag=1; break;
}
}
if((flag== 0)&&(k<f))
{
```

```
fault++; fr[k]=temp; k++;
}
else if((flag==0)&&(k==f})
{
fault++;

for (cnt=0;cnt<f;cnt++)
{
current=fr [cnt]; for (c=i;c>0;c--)
{
if (current !=pg[c]) count[cnt]++;else break;
}
}
max=0;
for (m=0;m<f;m++)
{
if (count[m]>max)
{
max=count[m]; p=m;
}
}
fr [p]=temp;
}
printf("\n");
for (x=0; x<f;x++)
{
printf("%d\t", fr[x]);
}
}
printf("\n Total number of faults = %d",fault); getchar();
}
```

OUTPUT:

```
Enter total number of pages:10

Enter sequence:7 0 1 2 0 3 0 4 2 3

MENU
FIFO
OPTIMAL
LRU
EXIT
ENTER YOUR CHOICE: 1
Enter size of page frame:3
7 -1 -1
7 0 -1
7 0 1
2 0 1
2 0 1
2 3 1
2 3 0
2 3 0
4 3 0
4 2 0
page fault = 9

MENU
1)FIFO
2)OPTIMAL
3)LRU
4)EXIT
ENTER YOUR CHOICE: 2
Enter size of page frame:3
7 -1 -1
7 0 -1
7 0 1
2 0 1
2 0 1
2 0 3
2 0 3
2 0 4
2 0 4
2 0 3
total number of faults = 8

MENU
1)FIFO
2)OPTIMAL
3)LRU
4)EXIT
ENTER YOUR CHOICE: 3
Enter size of page frame:3
7 -1 -1
```

```
7 0 -1
7 0 1
2 0 1
2 0 1
3 0 1
3 0 1
3 0 4
3 0 4
2 0 3
total number of faults = 9

MENU
1)FIFO
2)OPTIMAL
3)LRU
4)EXIT
ENTER YOUR CHOICE:4
```

IMPLEMENTATION OF SIMULATION OF DEADLOCK AVOIDANCE AND PREVENTION ALGORITHM.

PROGRAM :

```
#include<stdio.h>
void main()
{
int cLm[7][5],req[7][5], alloc[7][5],rsrc[5].avail[5],comp[7]; int first ,p,r,I,j,prc,count,t
count=0;
for (i=0;i<7;i++) comp [i]=0;
printf ("Enter the number of process:");
scanf("%d",&p);
printf("\n enter the no of resources:");
scanf("%d",&r);
printf("\n enter the claim for each process:");
for (i=1;i<p;i++)
{
printf("\n for process %d \n",i); for(j=1;j<=r; j++)
{
scanf("%d", &clm[i][j]);
}
}
printf("\n enter the allocation of each process:"); for (i=1;i<=p;i++)
{
printf("\n for process %d \n",i); for(j=1;j<=r; j++)
{
scanf("%d",&alloc[i][j]);
}
}
printf("\n enter total no of each resources : “);
for(j=1;j<=r; j++)
scanf("%d",&rsrc[j]);
for(j=11j<=r; j++)
{
int total=0;
avail[j]=0;
for (i=1;1<=p; i++)
{
total+=alloc[i][j];
}
avail[j]=rsrc[j]-total; do
{
for(i=1;i<=p;i++)
{
for(j=1;j<=r; j++)
{
req[i][j]=clm[i][j]-alloc[i][j];
}
}
printf("\n available resouces is: ");
for(j=1;j<=r; j++)
{
printf("%d", avail[j]);
printf("\n claim matrix: \t\t allocation matrix : \n “);
for (i=1;i<=p;i++)
{

for(j=1;j<=r; j++)
{
printf("%d",clm[i][j]);
}
printf("\t\t\t");
for(j=1;j<=r; j++)
{
printf("%d",alloc[i][j]);
}
printf("\n");
}
prc=0;
for (i=1;i<=p;i++)
{
if(comp[i]==0)
{
prc=1;
for (j=1;<=r; j++)
{
if(avail[j]==0)
{
pre=0; break;
}
}
}
if (prc !=0)
break;
}
if(prc !=0)
{
printf("\n process %d runs to completion “ ,prc);
count++;
for(j=1;j<=r; j++)
{
avail[j]+=alloc[pre][j]; alloc[prc][j]=0;clm[prc][j]=0;
comp[prc]=1;
}
}
}
While (count !=p && prc !=0);
if(count==p)printf("\n the system is in safe state !!”);
else
printf("\n the system is in unsafe state !! “);
}
```

OUTPUT:

```
Enter the number of process : 4
enter the no of resources:3
enter the claim for each process:
for process 1
1

2

3

for process 2

0

1

1

for process 3
4

3

1

enter the allocation of each process : for process 1

6     0     0

for process 2

2     0     0

for process 3

2     1     1

for process 4

0     0     2
enter the total no of resources : 10  5  7 available resources is 3     4     2



claim matrix   allocation matrix



3

1
```

```
1


0


process 1 runs to the completion        available resources is : 5        4        2 claim matrixallocation
matrix

0
1


1


0
process 2 runs to the completion        available resources is : 7        5        3 claim matrixallocation
matrix

0

0


1


0


process 3 runs to the completion        available resources is : 7        5        5 claim matrixallocation
matrix

0

0


0


0


process 3 runs to the completion the system is in safe state!!
```

IMPLEMENTATION OF THE CONSUMER PROBLEM USING SEMAPHORE

```
#include<stdio.h>
#include<stdlib.h>
typedef int semaphore;
semaphore mutex=1;
semaphore full=0;
semaphore empty=0;
main ()
int i,opt, size;
int buffer[20]={0};
printf("\n enter buffer capacity:");
scanf("%d",&size);
empty=size:
do
{
printf("\n menu: \n 1.producer \n 2.consumer \n 3,quit");
printf("\nenter the option:");
scanf("%d",&opt);
switch (opt)
{
case 1:
if (empty !=0)
{
empty-- ;
mutex-- ;
printf ("\nenter the items :”);
scanf("%d",&buffer [mutex]); mutex++;
full++;
printf("\n the item in buffer are:"); for (i=0:i<mutex; i++) printf(“%d” , buffer[i]);
printf("\the buffer size is %\t", mutex); mutex++;
}
else


printf("\n buffer overflow \n"); break;
case 2:
if (empty !=size)
{
full--; mutex-=2;
printf("\n the consumer Item is %d “ ,buffer[mutex-full]);
mutex++;
empty++;
for(i=0:i <mutex; i++)
buffer[i]=buffer [i+1];
printf("\n new item in buffer are:");
for (i=0; i <mutex-1;i++)
printf("%d",buffer[i]);
printf("\n the buffer size is %d” , mutex-1);
}
else
printf(“\n buffer empty \n);
break;
case3:
exit(0);
break;
default:
printf( "\n enter a valid option “);
break;
}
}
while(opt!=3);
return 0;}
```

OUTPUT :

```
enter the buffer capacity :
1 menu
1. producer
2.consumer
3. quit
enter the option : 1
enter the item: 5
the item in buffer are : 5
the buffer size is
1menu
1.producer
2.consumer
3. quit
enter the option : 1
buffer overflow
menu
1.producer
2.consumer
3. quit
enter the option:2
consumer item is : 5
the row item in the buffer are : 0
the buffer size is 0
menu
1.producer
2.consumer
3. quit
enter the option:2
buffer is empty
menu
1.producer
```

```
2.consumer
3. quit
enter the options : 3
```

EX.NO:

CASE STUDY: LINUX USING VMWARE

WHAT IS VMWARE?

VMware is a virtualization software suite that allows you to create and run virtual machines (VMs) on a host computer. Each VM acts as a separate computer system with its own operating system (OS), applications, and files. This enables you to run multiple operating systems, including Linux, on a single physical machine.

WHY USE LINUX IN A VM?

There are several reasons why you might want to use Linux in a VM:

- **Safe Experimentation:** You can try out Linux without affecting your main operating system. This is a great way to learn about Linux without risk.
- **Running Specific Applications:** You can use Linux to run applications that are not available for your main operating system.
- **Development and Testing:** Developers can create and test applications in a Linux environment that is isolated from their main system.
- **Security Research:** Security professionals can use VMs to create controlled environments for testing security tools and vulnerabilities.

HOW TO SET UP LINUX IN A VM:

1. **Download and Install VMware:** There are two main options for VMware software:
    - **VMware Workstation Player (free):** This is a free, non-commercial version for personal use. It's a good option for basic VM creation and use.
    - **VMware Workstation Pro (paid):** This paid version offers more advanced features, such as nested virtualization and support for more processors and memory.
2. **Download a Linux Distribution (ISO):** Choose a Linux distribution (distro) that suits your needs. Popular choices include Ubuntu, Mint, Fedora, and Debian. You can download the ISO image file from the distro's website.
3. **Create a New VM:** Open VMware and follow the on-screen instructions to create a new VM. You'll need to specify the amount of RAM, storage space, and processor cores to allocate to the VM.
4. **Install Linux on the VM:** During VM creation, point VMware to the downloaded Linux ISO file. This will boot the VM from the ISO and allow you to install Linux onto the virtual hard disk.

5. **Start Using Linux:** Once the installation is complete, you can power on the VM and start using Linux. You'll have a separate desktop environment and access to all the features of the chosen Linux distribution.

ADDITIONAL CONSIDERATIONS:

- **Hardware Requirements:** Ensure your host computer has enough RAM and processing power to run both the host operating system and the Linux VM smoothly.
- **Network Connectivity:** You can configure network settings for the VM to allow internet access or isolate it from your main network.
- **VMware Tools:** Install VMware Tools within the VM for improved performance and features like shared folders and seamless mouse/keyboard integration.

INSTALLATION PROCESS FOR RUNNING LINUX IN VMWARE

1. Download and Install VMware Workstation Player:

- Head to the official VMware Workstation Player download page: VMware Workstation Player download [invalid URL removed]
- Choose the free "VMware Workstation Player" option for personal use.
- Download the installer file (.exe for Windows) and follow the on-screen instructions to complete the installation.

2. Download a Linux Distribution (ISO):

- Visit the website of your chosen Linux distribution. Popular choices include:
  - Ubuntu: https://ubuntu.com/download/desktop
  - Mint: https://www.linuxmint.com/download.php
  - Fedora: https://www.fedoraproject.org/workstation/download/
  - Debian: https://www.debian.org/distrib/
- Locate the download section and choose the appropriate version for your needs (often a 64-bit version).
- Download the ISO image file.

3. Create a New Virtual Machine (VM):

- Open VMware Workstation Player.
- Click on "Create a New Virtual Machine" in the welcome screen.
- Select "Typical (recommended)" and click "Next."

4. Specify Installer disc image file:

- Choose the option "I will install the operating system later."
- Click "Next."

- In the "Guest operating system" section, select "Linux" and choose the version closest to your downloaded ISO (e.g., Ubuntu 64-bit).
- Click "Next."

5. Allocate Resources:

- Specify a name for your virtual machine (e.g., "My Linux VM").
- Decide on the amount of RAM to allocate to the VM. A good starting point is 2 GB, but you can adjust based on your system resources and desired performance.
- Choose the amount of disk space for the virtual hard disk. 20 GB is a reasonable starting point, but you can allocate more if needed.
- Click "Next."

6. Customize Hardware (Optional):

- This step allows you to fine-tune hardware settings for the VM. You can leave most settings at default for basic use.
- Click "Next" if you don't need to modify anything.

7. Connect Virtual Disc:

- Select "Split virtual disk into multiple files" (recommended for easier management).
- Choose a location to store the virtual disk files.
- Click "Next."

8. Review Settings and Finish:

- Review the summary of your VM configuration.
- If everything looks good, click "Finish" to create the VM.

9. Install Linux on the VM:

- Locate the downloaded Linux ISO file on your host machine.
- Power on the newly created VM.
- During the boot process, you'll be prompted to choose a boot device. Select the downloaded Linux ISO file (it might be listed as "CD/DVD Drive").
- Follow the on-screen instructions to install the chosen Linux distribution onto the VM's virtual hard disk. The installation process will vary slightly depending on the distro you selected.

10. Start Using Linux:

- Once the installation is complete, you'll be prompted to restart the VM.
- The VM will boot into the installed Linux distribution, and you'll have access to a dedicated Linux desktop environment.

BOOTING AND RUNNING LINUX:

1. **Power on the VM:** In VMware Workstation Player, locate your Linux VM and click the "Play virtual machine" button (or right-click and choose "Power on").

2. **Login:** The VM will boot up, displaying the chosen Linux distribution's startup process. Once it reaches the login screen, enter your username and password (created during installation) to log in.
3. **Desktop Environment:** You'll be greeted by the Linux desktop environment, which will vary depending on the distro you installed. Explore the menus, applications, and features available in your chosen Linux distribution.

BASIC OPERATIONS:

- **File Management:** Use the file manager (e.g., Nautilus in Ubuntu) to navigate directories, view files, and manage storage within the VM.
- **Applications:** Launch applications installed on your Linux VM from the desktop menu or search bar. Some distros come pre-installed with essential tools like web browsers, text editors, and office suites.
- **Terminal:** The terminal is a command-line interface that allows you to interact with the Linux system directly using text commands. You can find it in the applications menu or by searching for "Terminal."
- **Network Connectivity:** If you configured network access during installation, you should be able to browse the internet and access network resources from within the Linux VM.

INTERACTION WITH HOST MACHINE:

- **Shared Folders:** With VMware Tools installed, you can configure shared folders to access files and folders between your host machine (Windows/macOS) and the Linux VM. This allows for easy transfer of data between the two environments.
- **Copy-Paste:** You can also enable copy-paste functionality between the host and guest OS with VMware Tools, allowing you to seamlessly copy and paste text or files.

PROS:

- Safe experimentation with Linux without affecting your main system.
- Run applications unavailable on your primary OS.
- Isolated environment for development, testing, and security research.
- Excellent platform for learning Linux in a controlled setting.
- Potentially resource-efficient compared to other operating systems.
- High degree of customization for a personalized Linux experience.

CONS:

- Virtualization software adds overhead, potentially impacting performance.
- Learning curve associated with using Linux, especially for beginners.
- Limited access to host machine hardware compared to native installation.
- Importance of maintaining good security practices within the VM.

- Reliance on host machine resources can affect VM performance.

CONCLUSION:

Whether using Linux in a VM is right for you depends on your specific goals. If you prioritize safe exploration, running specific applications, or creating isolated development environments, the advantages outweigh the drawbacks. However, if you require maximum performance, have limited host system resources, or prefer direct hardware access, a native Linux installation might be a better choice. Ultimately, VMware VMs provide a flexible and convenient way to experience Linux on your existing hardware. Consider the pros and cons in light of your needs to determine if it's the most suitable approach for you.