# **Ex. 1 Basic UNIX Commands**

## **AIM:**

To study of Basic UNIX Commands and various UNIX editors such as vi, ed, ex

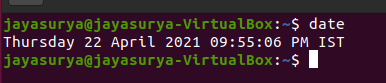
## **CONTENT:**

## **a) date**

– used to check the date and time

Syn: $date

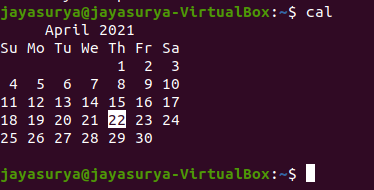
|  |  |  |  |
| --- | --- | --- | --- |
| Format | Purpose | Example | Result |
| +%m | To display only month | $ date +%m | 06 |
| +%h | To display month name | $ date +%h | June |
| +%d | To display day of month | $ date +%d | O1 |
| +%y | To display last two digits of years | $ date +%Y | 09 |
| +%H | To display hours | $ date +%H | 10 |
| +%M | To display minutes | $ date +%M | 45 |
| +%S | To display seconds | $ date +%S | 55 |



## **b) cal**

–used to display the calendar

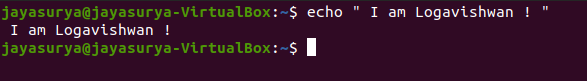
Syn: $cal 2 2009



## **c) echo**

–used to print the message on the screen.

Syn: $echo “text”



## **d) ls**

–used to list the files. Your files are kept in a directory.

Syn: $ls –s

All files (include files with prefix)

ls –l Lodetai (provide file statistics)

ls –t Order by creation time

ls –u Sort by access time (or show when last accessed together with –l)

ls –s Show file size

ls –r Reverse order

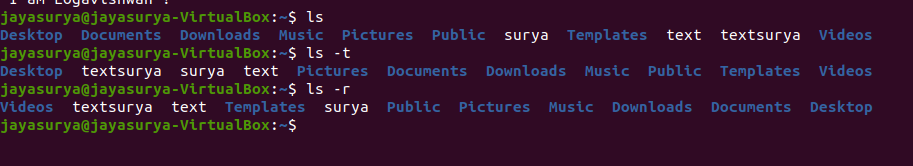
ls –f Mark directories with /,executable with\* , symbolic links with @, local sockets with named pipes(FIFOs)with

ls –S Order by size

ls – h“ Human Readable”, show file size in KiloBytes & MegaBytes (h can be used together with –l or)

ls [a-m]\* List all the files whose name begin with alphabets From „a‟ to „m‟

ls [a]\* List all the files whose name begins with „a‟ or „A‟

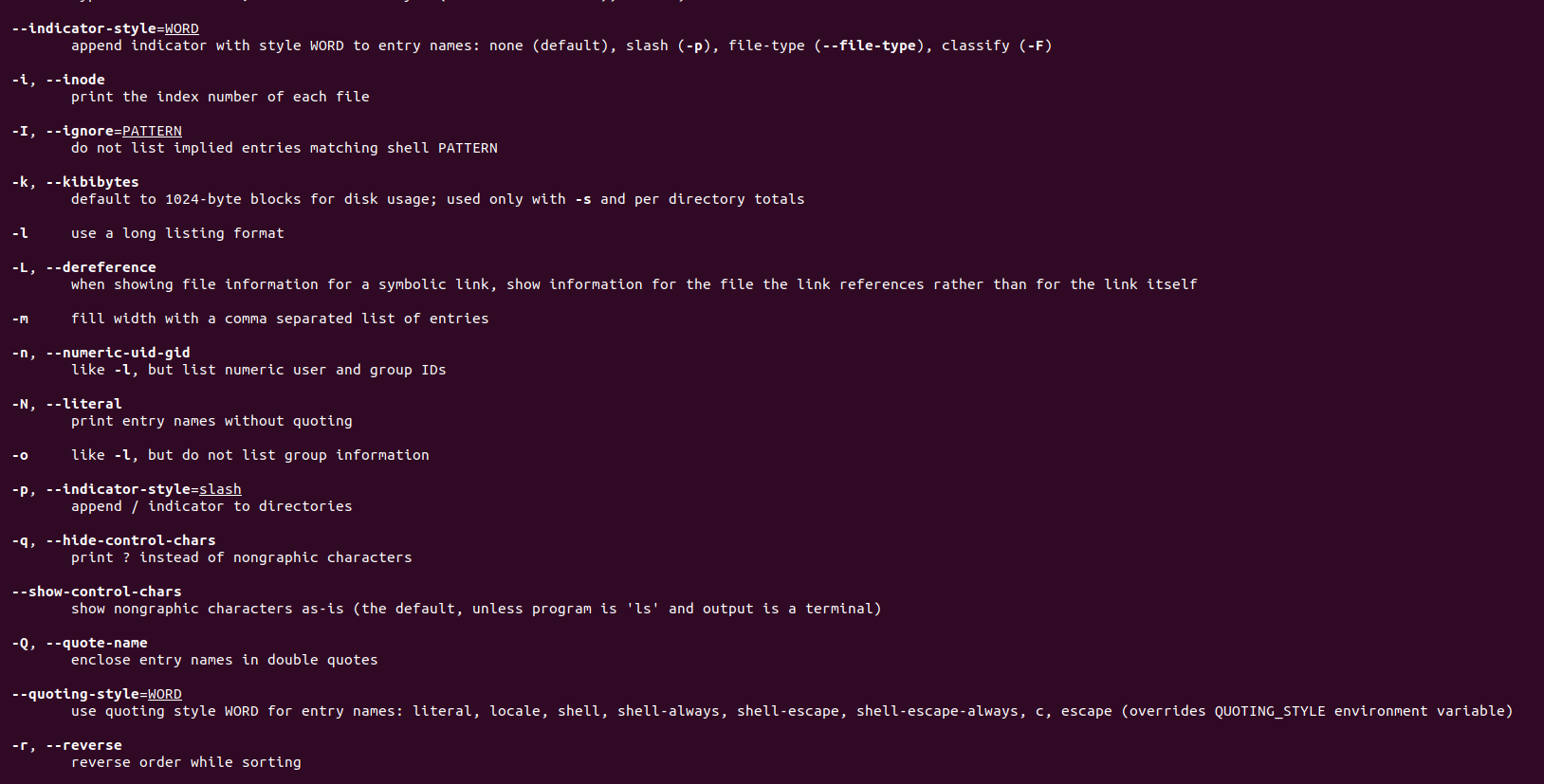
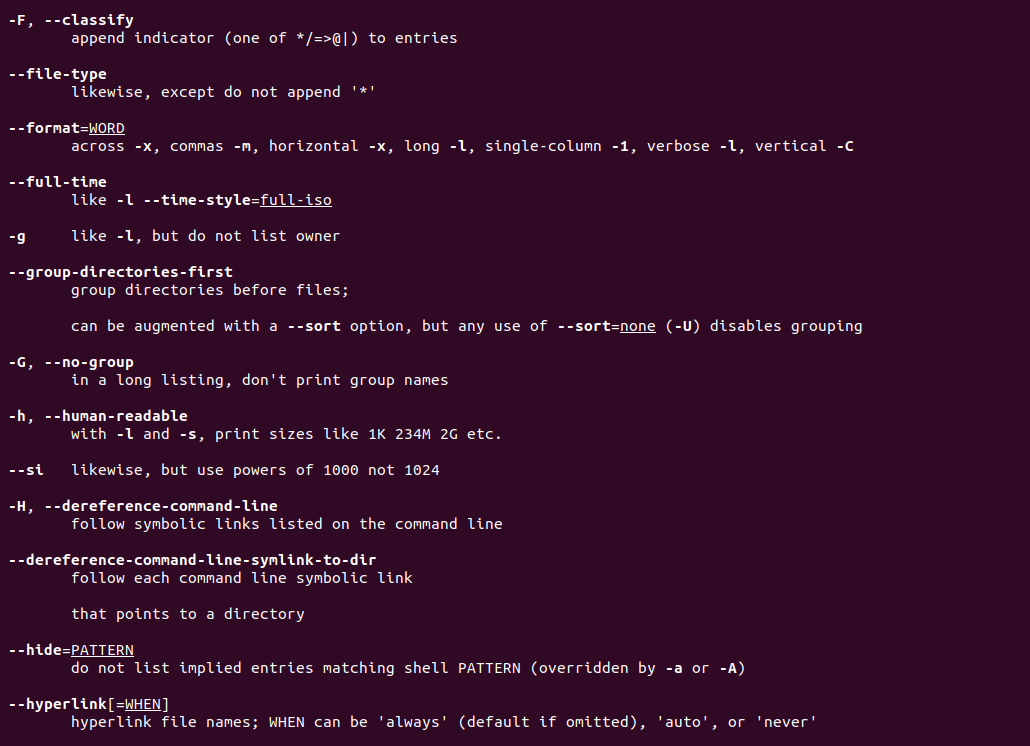
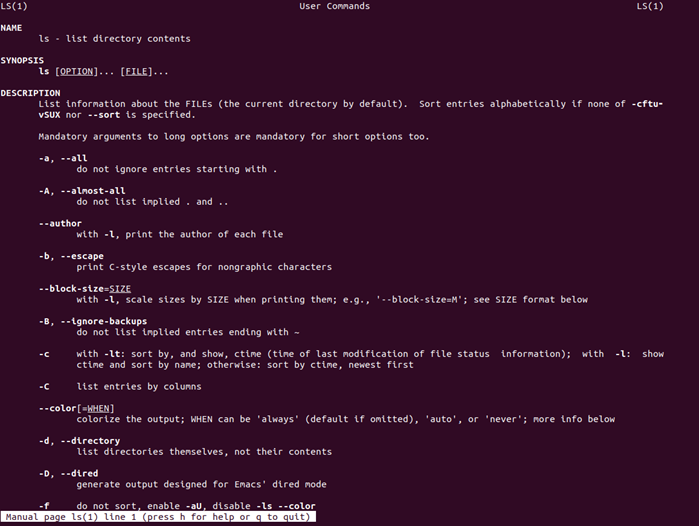


## **e) man**

–used to provide manual help on every UNIX command.

Syn: $man unix command

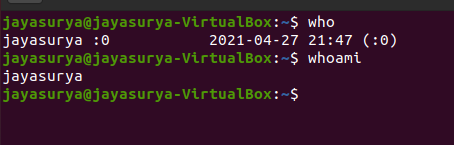
$ man cat



## **f) who & whoami**

–it displays data about all users who have logged into the system currently. The command displays the current user only.

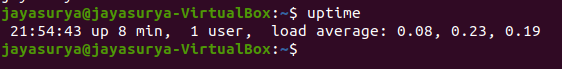
Syn:$ who, $ whoami



## **g) uptime**

–tells you how long the computer has been running since its last reboot or power-off.

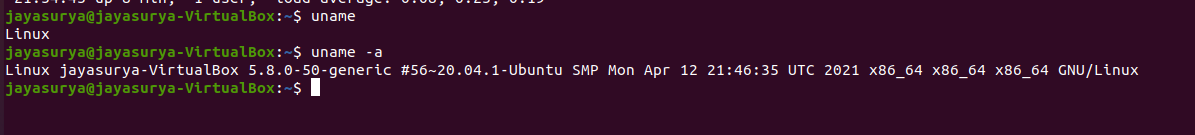
Syn:$ uptime



## **h) uname**

–it displays the system information such as hardware platform, system name and processor, OS type.

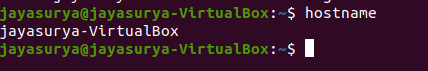
Syn:$ uname –a



**i) hostname**

–displays and set system host name

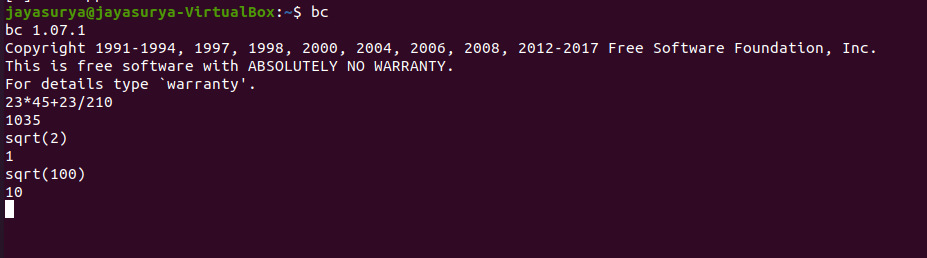
Syn:$ hostname



## **j) bc**

–stands for „best calculator‟

|  |  |  |  |
| --- | --- | --- | --- |
| $ bc | $ bc | $ bc | $ bc |
| 10/2\*3 | scale =1 | ibase=2 | sqrt(196) |
| 15 | 2.25+1 | obase=16 | 14 |
|  | 3.35 | 11010011 |  |
|  |  | 89275 |  |
|  |  | 1010 |  |
|  |  | Ā |  |
|  |  |  |  |
| $ bc | $ bc -l |  |  |
| for(i=1;i<3;i=i+1)I | scale=2 |  |  |
| 1 | s(3.14) |  |  |
| 2 | 0 |  |  |
| 3 |  |  |  |



## **FILE MANIPULATION COMMANDS**

a) **cat**–this create, view and concatenate files.

## **Creation:**

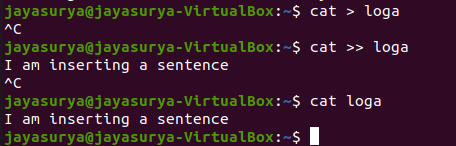
Syn:$ cat > filename

## **Viewing:**

Syn:$ cat filename

## **Add text to an existing file:**

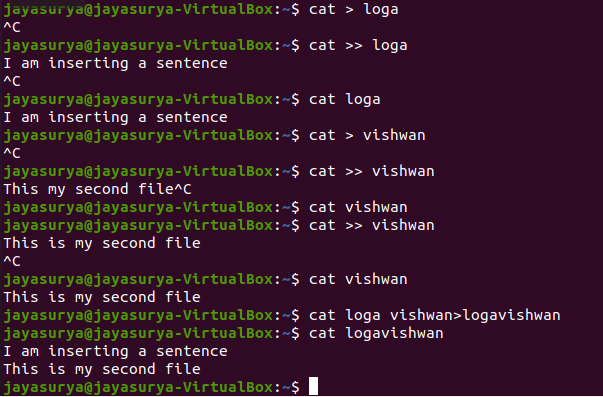
Syn:$ cat >>filename



**Concatenate:**

Syn: $cat file1 file2 > file3

$cat file1 file2 >> file3 (no over writing of file3)

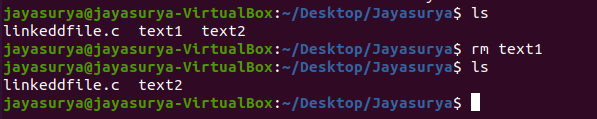


b) **grep**–used to search a particular word or pattern related to that word from the file.

Syn:$ grep search word filename

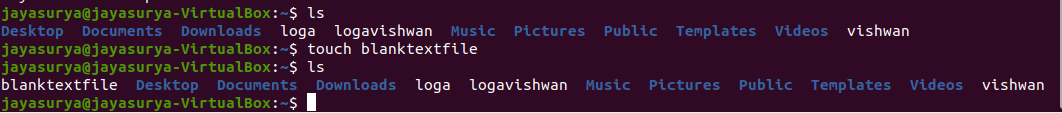
c) **rm**–deletes a file from the file system

Syn: $rm filename



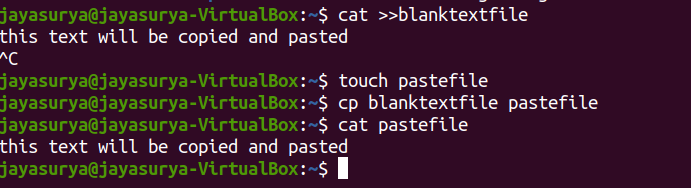
d) **touch**–used to create a blank file.

Syn: $touch file names



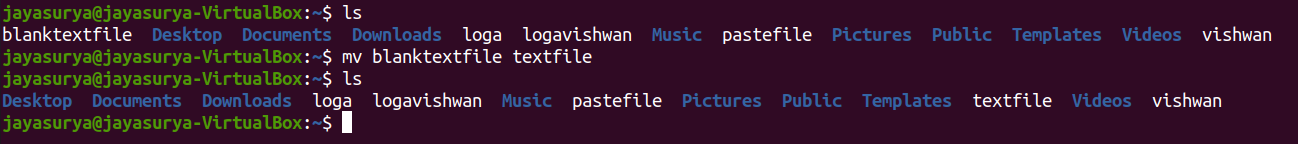
e) **cp**–copies the files or directories

Syn:$cp source file destination file



f) **mv**–to rename the file or directory

Syn:$mv old file new file



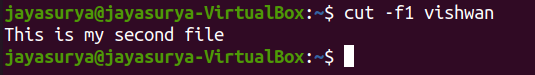
g) **cut**–it cuts or pickup a given number of characters or fields of the file.

Syn:$cut<option><filename>

Eg: $cut –c filename

$cut – c 1-10 emp

$ cut –f 3-6 emp

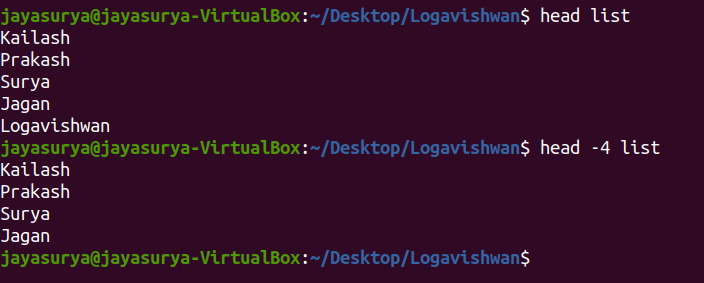


h) **head**–displays 10 lines from the head(top)of a given file

Syn:$head filename

To display the top two lines:

Syn:$head-2student



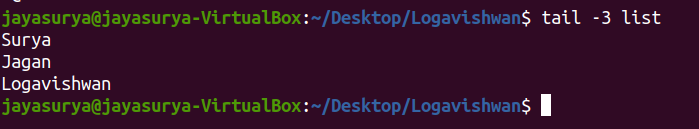
i) **tail**–displays last 10 lines of the file

Syn:$tail filename

Eg:$tail student

To display the bottom two lines;

Syn:$ tail -2 student



j) **chmod**–used to change the permissions of a file or directory.

Syn:$ch mod category operation permission file where,

Category–is the user type

Operation–is used to assign or remove permission

Permission–is the type of permission

File–are used to assign or remove permission all

Examples:

$ chmod -wx student

Removes write and execute permission for users

$chmod +rw, g+rw student

Assigns read and write permission for users and groups

$chmod g=rwx student

Assigns absolute permission for groups of all read, write and execute permissions

|  |  |  |
| --- | --- | --- |
| Category | Operation | Permission |
| u– users  g– group  o– others | +assign  - remove  =assign absolutely | r– read  w– write  x-execute |

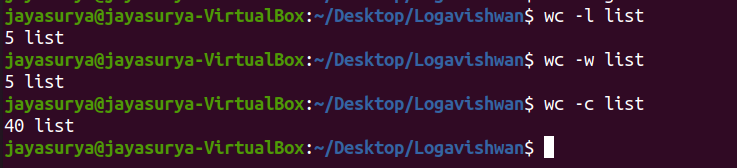


k) **wc**–it counts the number of lines, words, character in a specified file(s) with the options as –l,-w,-c

Syn: $wc –l filename

$wc –w filename

$wc –c filename



## 

## 

## **RESULT:**

Thus the basic UNIX commands are executed and verified in the unix terminal.

# 

# 

# 

# 

**Ex. 2a System Calls of UNIX operating system fork**

## **AIM:**

To write C Programs using system calls of UNIX operating system fork

## **ALGORITHM:**

STEP 1: Start the program.

STEP 2: Declare the variables pid,pid1,pid2.

STEP 3: Call fork() system call to create process.

STEP 4: If pid==-1, exit.

STEP 5: Ifpid!=-1 , get the process id using getpid().

STEP 6: Print the process id.

STEP 7:Stop the program

## **PROGRAM:**

int main( void ) {

printf( "The process identifier (pid) of the parent process is %d\n", (int)getpid() );

int pid = fork();

if ( pid == 0 ) {

printf( "After the fork, the process identifier (pid) "

"of the child is %d\n", (int)getpid() );

} else {

printf( "After the fork, the process identifier (pid) "

"of the parent is still %d\n - fork() returned %d\n",

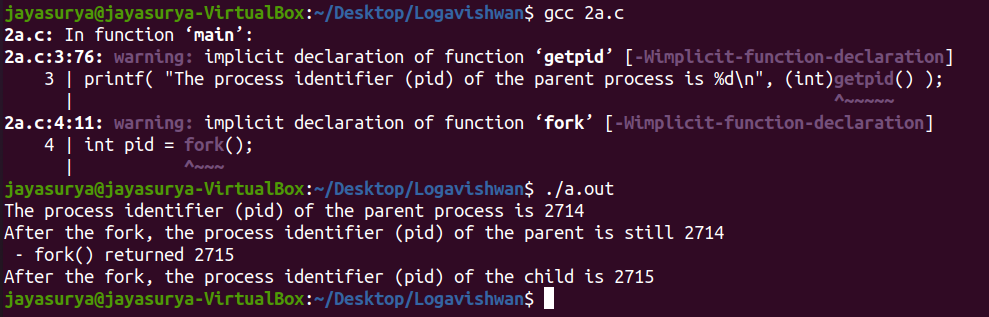
(int)getpid(), pid );

}

return 0;

}

## **OUTPUT:**



**RESULT:**

Thus C programs for using system calls of UNIX operating system fork are written, compiled and executed .

**Ex. 2b**  **System Calls of UNIX operating system exec**

## **AIM:**

To write C Programs using system calls of UNIX operating system exec

## **ALGORITHM:**

STEP 1: Start the program.

STEP 2: Declare the variables i.

STEP 3: Call exec() system call to create process.

STEP 4: Print the process id

STEP 5: Stop

## **PROGRAM:**

#include<stdio.h>

#include<unistd.h>

int main()

{

int i;

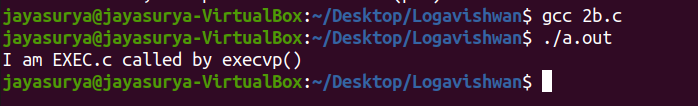
printf("I am EXEC.c called by execvp() ");

printf("\n");

return 0;

}

## **OUTPUT:**



## **RESULT:**

Thus C Programs using system calls of UNIX operating system exec are written compiled and executed

# 

**Ex. 2c**  **System Calls of UNIX operating system exec**

## **AIM:**

To write C Programs using system calls of UNIX operating system exec

## **ALGORITHM:**

STEP 1: Start the program.

STEP 2: Declare the variables i.

STEP 3: Call exec() system call to create process.

STEP 4: Print the process id

STEP 5: Stop

## **PROGRAM:**

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

int main()

{

//A null terminated array of character

//pointers

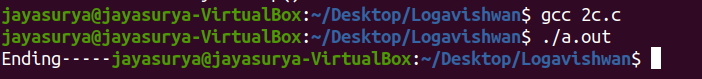
char \*args[]={"./EXEC",NULL};

execvp(args[0],args);

printf("Ending-----");

return 0; }

## **OUTPUT:**



## **RESULT:**

Thus C Programs using system calls of UNIX operating system exec are executed

# **Ex. 2d System Calls of UNIX operating system wait**

## **AIM:**

To write C Programs using system calls of UNIX operating system wait

## **ALGORITHM:**

STEP 1: Start the program.

STEP 2: Call exec() system call to create process.

STEP 3: Take one argument status and returns a process ID of dead children.

STEP 4: Print that the child is terminated

STEP 5: Stop

## **PROGRAM:**

#include<stdio.h>

#include<sys/wait.h>

#include<unistd.h>

int main()

{

if (fork()== 0)

printf("HC: hello from child\n");

else

{

printf("HP: hello from parent\n");

wait(NULL);

printf("CT: child has terminated\n");

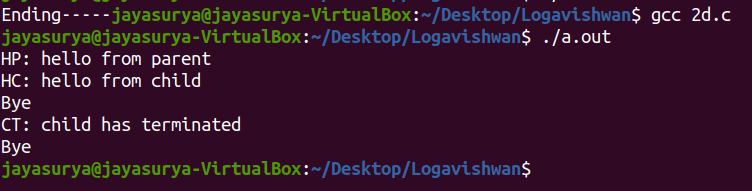
}

printf("Bye\n");

return 0;

}

## **OUTPUT:**



## **RESULT:**

Thus C Programs using system calls of UNIX operating system wait are written compiled and executed.

**Ex. 2e System Calls of UNIX operating system stat**

## **AIM:**

To write C Programs using system calls of UNIX operating system stat .

## **ALGORITHM:**

STEP 1: Start the program.

STEP 2: Declare the struct variables.

STEP 3: Call stat() system call to create process.

STEP 4: Print the stat mode

STEP 5: Stop

## **PROGRAM:**

#include<stdio.h>

#include<sys/stat.h>

int main()

{

//pointer to stat struct

struct stat sfile;

//stat system call

stat("stat.c", &sfile);

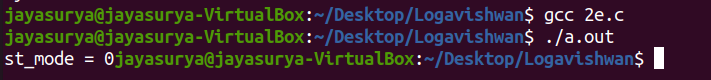
//accessing st\_mode (data member of stat struct)

printf("st\_mode = %o", sfile.st\_mode);

return 0;

}

## **OUTPUT:**



## **RESULT:**

To write C Programs using system calls of UNIX operating system stat .

# 

# **Ex. 2f System Calls of UNIX operating system opendir**

## 

## **AIM:**

To write C Programs using system calls of UNIX operating system opendir

## **ALGORITHM:**

STEP 1: Start the program.

STEP 2: Create struct dirent.

STEP 3: declare the variable buff and pointer dptr.

STEP 4: Get the directory name.

STEP 5: Open the directory.

STEP 6: Read the contents in the directory and print it.

STEP 7: Close the directory.

## **PROGRAM:**

#include <stdio.h>

#include <dirent.h>

int main()

{

DIR \*folder;

folder = opendir(".");

if(folder == NULL)

{

puts("Unable to read directory");

return(1);

}

else

{

puts("Directory is opened!");

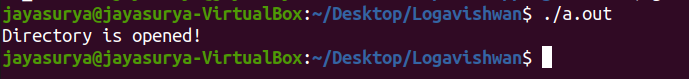
}

closedir(folder);

return(0);

}

## **OUTPUT:**



## **RESULT:**

Thus C Programs using system calls of UNIX operating system opendir are written compiled and executed .

**Ex. 2g System Calls of UNIX operating system readdir**

## **AIM:**

To write C Programs using system calls of UNIX operating system readdir

## **ALGORITHM:**

STEP 1: Start the program.

STEP 2: Create struct dirent.

STEP 3: declare the variable buff and pointer dptr.

STEP 4: Get the directory name.

STEP 5: Open the directory.

STEP 6: Read the contents in the directory and print it.

STEP 7: Close the directory.

## **PROGRAM:**

#include <stdio.h>

#include <dirent.h>

int main()

{

DIR \*folder;

struct dirent \*entry;

int files = 0;

folder = opendir(".");

if(folder == NULL)

{

perror("Unable to read directory");

return(1);

}

while( (entry=readdir(folder)) )

{

files++;

printf("File %3d: %s\n",

files,

entry->d\_name

);

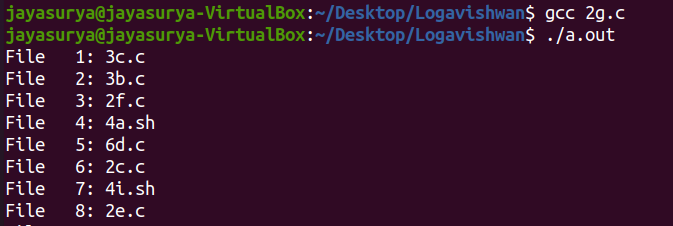
}

closedir(folder);

return(0);

}

## **OUTPUT:**



## **RESULT:**

Thus C Programs using system calls of UNIX operating system readdir are written compiled and executed .

**Ex. 3a Simulation of ls UNIX Commands**

## **AIM:**

To write C programs to simulate the UNIX command ls

## **ALGORITHM:**

STEP1 : Start the program

STEP2 : Open the directory with directory object dp

STEP3 : Read the directory content and print it.

STEP4: Close the directory.

## **PROGRAM:**

//Used for basic input/output stream

#include <stdio.h>

//Used for handling directory files

#include <dirent.h>

//For EXIT codes and error handling

#include <errno.h>

#include <stdlib.h>

void \_ls(const char \*dir,int op\_a,int op\_l)

{

//Here we will list the directory

struct dirent \*d;

DIR \*dh = opendir(dir);

if (!dh)

{

if (errno = ENOENT)

{

//If the directory is not found

perror("Directory doesn't exist");

}

else

{

//If the directory is not readable then throw error and exit

perror("Unable to read directory");

}

exit(EXIT\_FAILURE);

}

//While the next entry is not readable we will print directory files

while ((d = readdir(dh)) != NULL)

{

//If hidden files are found we continue

if (!op\_a && d->d\_name[0] == '.')

continue;

printf("%s ", d->d\_name);

if(op\_l) printf("\n");

}

if(!op\_l)

printf("\n");

}

int main(int argc, const char \*argv[])

{

if (argc == 1)

{

\_ls(".",0,0);

}

else if (argc == 2)

{

if (argv[1][0] == '-')

{

//Checking if option is passed

//Options supporting: a, l

int op\_a = 0, op\_l = 0;

char \*p = (char\*)(argv[1] + 1);

while(\*p){

if(\*p == 'a') op\_a = 1;

else if(\*p == 'l') op\_l = 1;

else

{

perror("Option not available");

exit(EXIT\_FAILURE);

}

p++;

}

\_ls(".",op\_a,op\_l);

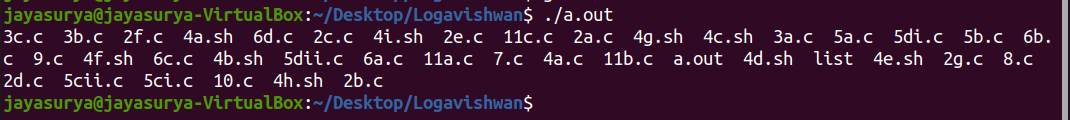
}

}

return 0;

}

## **OUTPUT:**



## **RESULT:**

Thus C programs to simulate the UNIX command ls are written compiled and executed .

# 

# 

# 

**Ex. 3b Simulation of grep UNIX Commands**

## **AIM:**

To write C programs to simulate the UNIX command grep

## **ALGORITHM:**

STEP1: Start the program

STEP2: Declare the variables fline[max], count=0, occurrences=0 and pointers \*fp,

\*newline.

STEP 3: Open the file in read mode.

STEP4: In while loop check fgets(fline,max,fp)!=NULL

STEP 5: Increment count value.

STEP 6: Check newline=strchr(fline, „\n‟)

STEP 7: print the count,fline value and increment the occurrence value.

STEP 8: Stop the program

## **PROGRAM:**

#include<stdio.h>

#include<string.h>

void main()

{

char fn[30],pat[30],temp[200];

FILE \*fp;

printf("Enter file name\n");

scanf("%s",fn);

printf("Enter pattern to be searched\n");

scanf("%s",pat);

fp=fopen(fn,"r");

while(!feof(fp))

{

fgets(temp,1000,fp);

if(strstr(temp,pat)!=NULL)

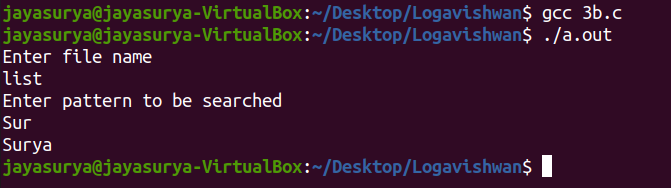
printf("%s",temp);

}

fclose(fp);

}

## **OUTPUT:**



## **RESULT:**

# Thus C programs to simulate UNIX command grep are written , compiled and executed .

# 

**Ex. 3c Simulation of cp UNIX Commands**

## **AIM:**

To write C programs to simulate the UNIX command cp

## **ALGORITHM:**

STEP1: Start the program

STEP 2:Declare the variables ch, \*fp, sc=0

STEP3: Open the file in read mode

STEP 4: Get the character

STEP 5: If ch== “ “ then increment sc value by one

STEP 6: Print no of spaces

STEP 7:Close the file

## **PROGRAM:**

#include <stdio.h>

#include <stdlib.h>

int main()

{

FILE \*fptr1, \*fptr2;

char filename[100], c;

printf("Enter the filename to open for reading \n");

scanf("%s", filename);

fptr1 = fopen(filename, "r");

if (fptr1 == NULL)

{

printf("Cannot open file %s \n", filename);

exit(0);

}

printf("Enter the filename to open for writing \n");

scanf("%s", filename);

fptr2 = fopen(filename, "w");

if (fptr2 == NULL)

{

printf("Cannot open file %s \n", filename);

exit(0);

}

c = fgetc(fptr1);

while (c != EOF)

{

fputc(c, fptr2);

c = fgetc(fptr1);

}

printf("\nContents copied to %s", filename);

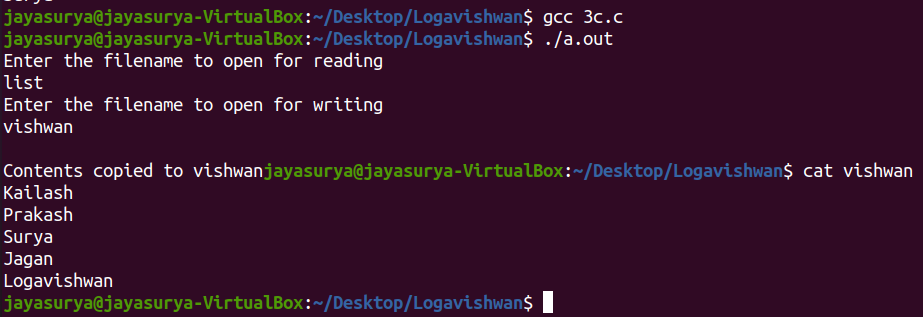
fclose(fptr1);

fclose(fptr2);

return 0;

}

## **OUTPUT:**



## **RESULT:**

Thus C programs to simulate the UNIX command cp are written ,compiled and executed.

**Ex. 4a Shell program to swap two numbers**

## 

## **AIM:**

To write a simple shell program to swap two numbers

## **ALGORITHM:**

STEP 1: Start the program.

STEP 2: Read the value of a,b.

STEP 3: Calculate the swapping of two values by using a temporary variable temp.

STEP 4: Print the value of a and b.

## **PROGRAM:**

echo "Enter value for x : "

read x

echo "Enter value for y : "

read y

echo "Before swap, x = $x and y = $y"

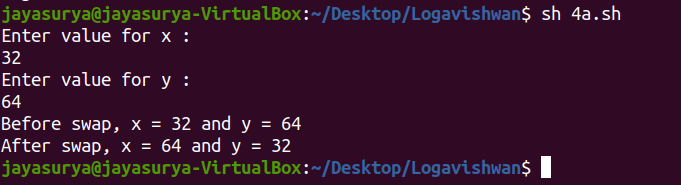
z=$x

x=$y

y=$z

echo "After swap, x = $x and y = $y"

## **OUTPUT:**



## **RESULT:**

Thus a simple shell program to swap two numbers is written , compiled and executed.

# 

# **Ex. 4b Shell program to convert from Celsius to Fahrenheit and vice versa**

## **AIM:**

To write a simple shell program to convert from Celsius to Fahrenheit and vice versa

## **ALGORITHM:**

STEP 1: Read the temperature in degree Centigrade.

STEP 2: Convert the Centigrade to Fahrenheit using the formula

F=9/5\*c+32

STEP 3: Print the Celsius and Fahrenheit value.

STEP 4: Read the temperature in degree Fahrenheit.

STEP 5: Convert the Fahrenheit to Centigrade using the formula

C=5/9\*(F-32)

STEP 6: Print the Fahrenheit and Celsius value.

STEP 7: Stop

## **PROGRAM:**

echo "1. Convert Celsius temperature into Fahrenheit"

echo "2. Convert Fahrenheit temperatures into Celsius"

echo -n "Select your choice (1-2) : "

read choice

if [ $choice -eq 1 ]

then

echo -n "Enter temperature (C) : "

read tc

# formula Tf=(9/5)\*Tc+32

tf=$(echo "scale=2;((9/5) \* $tc) + 32" |bc)

echo "$tc C = $tf F"

elif [ $choice -eq 2 ]

then

echo -n "Enter temperature (F) : "

read tf

# formula Tc=(5/9)\*(Tf-32)

tc=$(echo "scale=2;(5/9)\*($tf-32)"|bc)

echo "$tf = $tc"

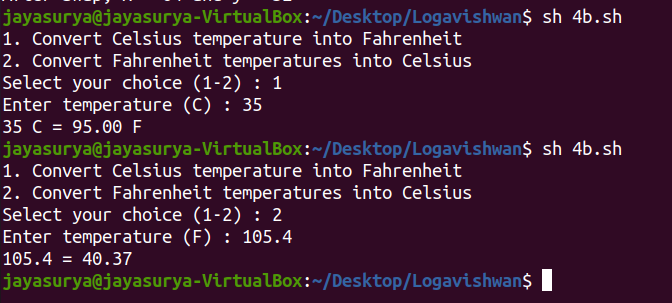
else

echo "Please select 1 or 2 only"

exit 1

fi

## **OUTPUT:**



## **RESULT:**

## 

Thus a simple shell program to convert from Celsius to Fahrenheit and vice versa is written, compiled and executed .

# 

# 

**Ex. 4c Shell program to find the Biggest of three numbers**

## **AIM:**

To write a simple shell program to find the Biggest of three numbers

## **ALGORITHM:**

STEP 1: start the problem

STEP 2: take three inputs from the user

STEP 3: in if-else condition, check which is the greatest

STEP 4: also check with the third number

STEP 5: find the result

STEP 6: print the result

STEP 7: stop the program

## **PROGRAM:**

echo -n "Give value for A B and C: "

read a b c

if [ $a -gt $b -a $a -gt $c ]

then

echo "A is the Biggest number"

elif [ $b -gt $c ]

then

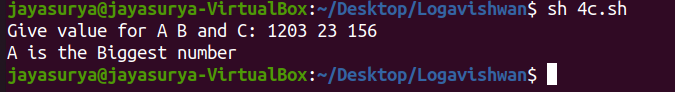
echo "B is the Biggest number"

else

echo "C is the Biggest number"

fi

**OUTPUT:**



# 

## **RESULT:**

Thus a simple shell program to find the Biggest of three numbers is written , compiled and executed .

**Ex. 4d Shell program to find the grade**

## **AIM:**

To write a simple shell program to find the grade

## **ALGORITHM:**

Step 1 : Start

Step 2 : read marks or Percentage

Step 3 : if marks >= 90 then grade =S, go to step 7

Step 4 : if marks >= 80 and marks <=70 then grade = A, go to step 7

Step 5 : if marks >=70 and marks <=60 then grade = B go to step 7

Step 5 : if marks >=60 and marks <=55 then grade = C go to step 7

Step 5 : if marks >=55 and marks <=50 then grade = D go to step 7

Step 3 : if marks <= 50 then grade =E, go to step 7

Step 6 : else display U

Step 7 : display grade.

Step 8 : Stop.

## **PROGRAM:**

echo -n "Enter the mark : "

read mark

if [ $mark -gt 90 ]

then

echo "S Grade"

elif [ $mark -gt 80 ]

then

echo "A Grade"

elif [ $mark -gt 70 ]

then

echo "B Grade"

elif [ $mark -gt 60 ]

then

echo "C Grade"

elif [ $mark -gt 55 ]

then

echo "D Grade"

elif [ $mark -ge 50 ]

then

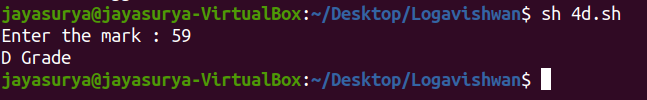
echo "E Grade"

else

echo "U Grade"

fi

## **OUTPUT:**



## **RESULT:**

Thus a simple shell program to find the grade is written, executed and output is obtained.

# 

**Ex. 4e Shell program to check the given alphabet is vowel or consonant**

## **AIM:**

To write a simple shell program to check the given alphabet is vowel or consonant

## **ALGORITHM:**

Step 1: Start

Step 2: Declare character type variable ch

Step 3: Read ch from User

Step 4: IF (ch == 'a' || ch == 'A' ||

ch == 'e' || ch == 'E' ||

ch == 'i' || ch == 'I' ||

ch == 'o' || ch == 'O' ||

ch == 'u' || ch == 'U' )

Print "Vowel"

ELSE

Print "Consonant"

Step 5: Stop

## **PROGRAM:**

echo -n "Key in a lower case character : "

read choice

case $choice in

a|e|i|o|u) echo "It's a Vowel";;

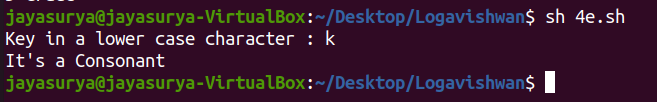
\*) echo "It's a Consonant"

esac

## 

## 

## **OUTPUT:**



# 

## **RESULT:**

Thus a simple shell program to find whether the character is vowel or consonant is written, executed and output is obtained.

**Ex. 4f Shell program to perform basic calculation**

## **AIM:**

To write a simple shell program to perform basic calculation

## **ALGORITHM:**

STEP 1: Start the program.

STEP 2:Take two numbers as input and store it in the variables a and b..

STEP 3: Take an input to let the user choose the operation that needs to be performed.

STEP 4: If the user chooses addition, then perform a+b.

STEP 5: If the user chooses subtraction, then perform a-b

STEP 6: If the user chooses multiplication, then perform a\*b.

STEP 7: If the user chooses division, then perform a/b.

STEP 8: Print the value after carrying out the operation according to the user's choice.

STEP 9: End of algorithm.

## **PROGRAM:**

echo -n "Enter the two numbers : "

read a b

echo " 1. Addition"

echo " 2. Subtraction"

echo " 3. Multiplication"

echo " 4. Division"

echo -n "Enter the option : "

read option

case $option in

1) c=`expr $a + $b`

echo "$a + $b = $c";;

2) c=`expr $a - $b`

echo "$a - $b = $c";;

3) c=`expr $a \\* $b`

echo "$a \* $b = $c";;

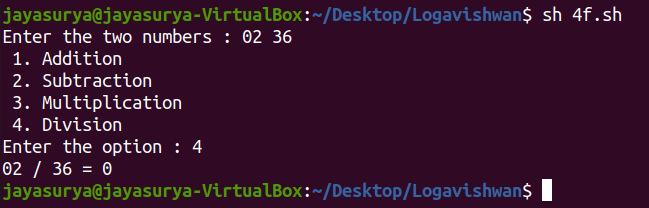
4) c=`expr $a / $b`

echo "$a / $b = $c";;

\*) echo "Invalid Option"

esac

## **OUTPUT:**



## **RESULT:**

Thus a simple shell program to perform simple calculation is written, executed and output is obtained.

**Ex. 4g Shell program to display multiplication table**

## **AIM:**

To write a simple shell program to display multiplication table

## **ALGORITHM:**

STEP 1: Start the program.

STEP 2: Input the number for which the multiplication table is to be generated.

STEP 3: Repeat from i = 1 to end

STEP 4: Display the table values in the given output format.(num \* i = num\*i)

STEP 5: Stop

## **PROGRAM:**

echo -n "Which multiplication table? : "

read n

for x in 1 2 3 4 5 6 7 8 9 10

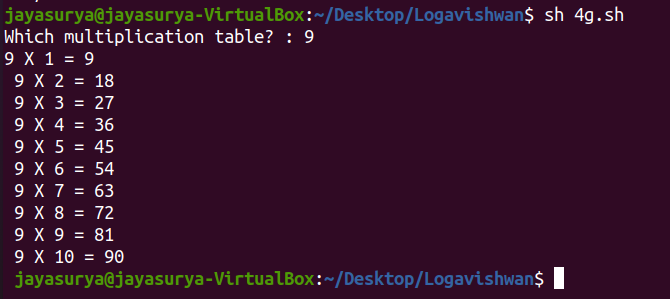
do

p=`expr $x \\* $n`

echo -n "$n X $x = $p"

sleep 1

Done

**OUTPUT:** 

**RESULT:**

Thus a simple shell program to display multiplication table is written, executed and output is obtained.

# 

# **Ex. 4h Shell program to reverse a number**

## **AIM:**

To write a simple shell program to reverse a numbers

## **ALGORITHM:**

STEP 1: Start the program.

STEP 2: Initialize rev\_num = 0

STEP 3: Loop while num > 0.

STEP 4: (a) Multiply rev\_num by 10 and add remainder of num

divide by 10 to rev\_num

rev\_num = rev\_num\*10 + num%10;

(b) Divide num by 10

STEP 5: Return rev\_num

STEP 6: Stop

## **PROGRAM:**

echo -n "Enter a number : "

read n

rd=0

while [ $n -gt 0 ]

do

rem=`expr $n % 10`

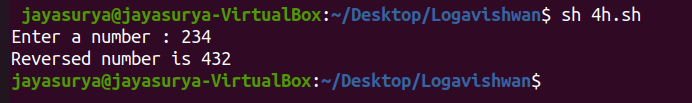
rd=`expr $rd \\* 10 + $rem`

n=`expr $n / 10`

done

echo "Reversed number is $rd"

## **OUTPUT:**



## **RESULT:**

Thus a simple shell program to reverse a number is written, executed and output is obtained.

# 

# 

**Ex. 4i Shell program to find whether the number is prime or not**

## **AIM:**

To write a simple shell program to find whether the number is prime or not

## **ALGORITHM:**

STEP 1: Start the program.

STEP 2: Take integer variable A

STEP 3: Divide the variable A with (A-1 to 2)

STEP 4: If A is divisible by any value (A-1 to 2) it is not prime

STEP 5: Else it is prime

STEP 6: Stop

## **PROGRAM:**

echo -n "Enter the number : "

read n

i=2

m=`expr $n / 2`

until [ $i -gt $m ]

do

q=`expr $n % $i`

if [ $q -eq 0 ]

then

echo "Not a Prime number"

exit

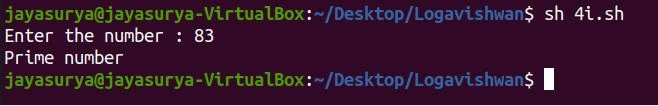
fi

i=`expr $i + 1`

done

echo "Prime number"

## **OUTPUT:**



## **RESULT:**

Thus a simple shell program to find prime or not is written, executed and output is obtained.

**Ex. 5a Generate 25 fibonacci numbers and determine prime among them using pipe**

# 

## **AIM:**

To write a c program to generate 25 fibonacci numbers and determine prime among them using pipe

## **ALGORITHM:**

1. Declare a array to store fibonacci numbers

2. Decalre a array pfd with two elements for pipe descriptors.

3. Create pipe on pfd using pipe function call. a.If return value is -1 then stop

4. Using fork system call, create a child process.

5. Let the child process generate 25 fibonacci numbers and store them in a array.

6. Write the array onto pipe using write system call.

7. Block the parent till child completes using wait system call.

8. Store fibonacci nos. written by child from the pipe in an array using read system call

9. Inspect each element of the fibonacci array and check whether they are prime a.If prime then

Print the fibonacci term

10. Stop

## **PROGRAM:**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

main()

{

pid\_t pid; int

pfd[2];

int i,j,flg,f1,f2,f3;

static unsigned int ar[25],br[25];

if(pipe(pfd) == -1)

{

printf("Error in pipe"); exit(-

1);

}

pid=fork(); if

(pid == 0)

{

printf("Child process generates Fibonacci series\n" ); f1 = -1;

f2 = 1;

for(i = 0;i < 25; i++)

{

f3 = f1 + f2;

printf("%d\t",f3); f1 =

f2;

f2 = f3; ar[i] =

f3;

}

write(pfd[1],ar,25\*sizeof(int));

}

else if (pid > 0)

{

wait(NULL);

read(pfd[0], br, 25\*sizeof(int));

printf("\nParent prints Fibonacci that are Prime\n");

for(i = 0;i < 25; i++)

{

flg = 0;

if (br[i] <= 1) flg =

1;

for(j=2; j<=br[i]/2; j++)

{

if (br[i]%j == 0)

{

flg=1;

break;

}

}

if (flg == 0) printf("%d\t",

br[i]);

}

printf("\n");

}

else

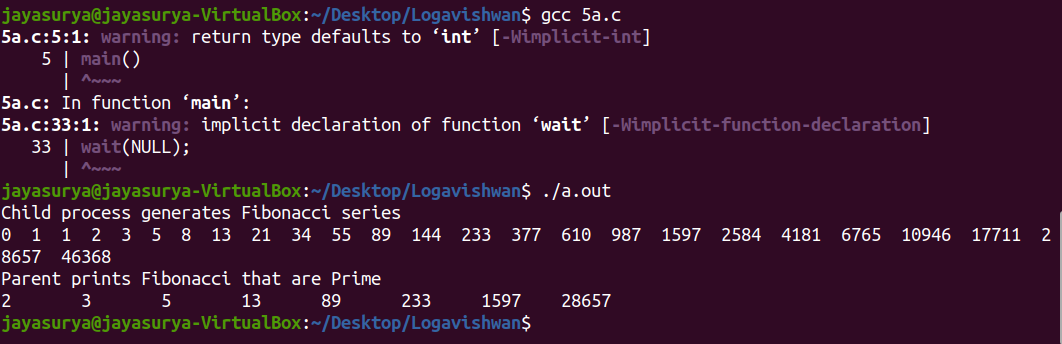
{

printf("Process creation failed"); exit(-1);

}

}

## **OUTPUT:**



## **RESULT:**

Thus c program to generate 25 fibonacci numbers and determine prime among them using pipe is written, executed and output is obtained.

**Ex. 5b Determine the number of users logged in using pipe**

# 

## **AIM:**

To write a c program to determine the number of users logged in using pipe

## **ALGORITHM:**

1. Decalre a array pfd with two elements for pipe descriptors.

2. Create pipe on pfd using pipe function call. a. If return value is -1 then stop

3. Using fork system call, create a child process.

4. Free the standard output (1) using close system call to redirect the output to pipe.

5. Make a copy of write end of the pipe using dup system call.

6. Execute who command using execlp system call.

7. Free the standard input (0) using close system call in the other process.

8. Make a close of read end of the pipe using dup system call.

9. Execute wc –l command using execlp system call.

10. Stop

## **PROGRAM:**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

int main()

{

int pfds[2];

pipe(pfds);

if (!fork())

{

close(1);

dup(pfds[1]);

close(pfds[0]); execlp("who",

"who", NULL);

}

else

{

close(0);

dup(pfds[0]);

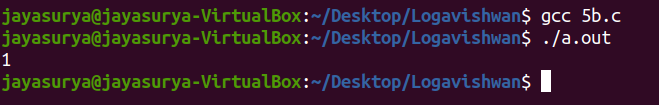
close(pfds[1]);

execlp("wc", "wc", "-l", NULL);

}

}

## **OUTPUT:**



## **RESULT:**

Thus c program to determine the number of users logged in using pipe is written, executed and output is obtained.

**Ex. 5c Exchange messages between server time using message queue**

# 

## **AIM:**

To write a c program to exchange messages between server time using message queue

## **ALGORITHM:**

**Server**

1. Decalre a structure mesgq with type and text fields.

2. Initialize key to 2013 (some random value).

3. Create a message queue using msgget with key & IPC\_CREAT as parameter. a. If message queue cannot be created then stop.

4. Initialize the message type member of mesgq to 1.

5. Do the following until user types Ctrl+D

a. Get message from the user and store it in text member.

b. Delete the newline character in text member.

c. Place message on the queue using msgsend for the client to read.

d. Retrieve the response message from the client using msgrcv function

e. Display the text contents.

6. Remove message queue from the system using msgctl with IPC\_RMID as parameter.

7. Stop

**Client**

1. Decalre a structure mesgq with type and text fields.

2. Initialize key to 2013 (same value as in server).

3. Open the message queue using msgget with key as parameter. a. If message queue cannot be

opened then stop.

4. Do while the message queue exists

a. Retrieve the response message from the server using msgrcv function

b. Display the text contents.

c. Get message from the user and store it in text member.

d. Delete the newline character in text member.

e. Place message on the queue using msgsend for the server to read. 5. Print "Server

Disconnected".

6. Stop

## **PROGRAM:**

**Server**

/\* Server chat process - srvmsg.c \*/

#include <stdio.h>

#include<stdlib.h>

#include<string.h>

#include<sys/types.h>

#include<sys/ipc.h>

#include<sys/msg.h>

struct mesgq

{

long type;

char text[200]; }

mq;

main()

{

int msqid, len; key\_t

key = 2013;

if((msqid = msgget(key, 0644|IPC\_CREAT)) == -1)

{

perror("msgget");

exit(1);

}

printf("Enter text, ^D to quit:\n"); mq.type =

1;

while(fgets(mq.text, sizeof(mq.text), stdin) != NULL)

{

len = strlen(mq.text);

if (mq.text[len-1] == '\n')

mq.text[len-1] = '\0';

msgsnd(msqid, &mq, len+1, 0);

msgrcv(msqid, &mq, sizeof(mq.text), 0, 0);

printf("From Client: \"%s\"\n", mq.text);

}

msgctl(msqid, IPC\_RMID, NULL);

}

**Client**

#include <stdio.h> #include

<stdlib.h> #include

<string.h> #include

<sys/types.h> #include

<sys/ipc.h> #include

<sys/msg.h>

struct mesgq {

long type;

char text[200]; }

mq;

main() {

int msqid, len; key\_t

key = 2013;

if ((msqid = msgget(key, 0644)) == -1)

{

printf("Server not active\n"); exit(1);

}

printf("Client ready :\n");

while (msgrcv(msqid, &mq, sizeof(mq.text), 0, 0) != -1)

{

printf("From Server: \"%s\"\n", mq.text);

fgets(mq.text, sizeof(mq.text), stdin); len =

strlen(mq.text);

if (mq.text[len-1] == '\n')

mq.text[len-1] = '\0';

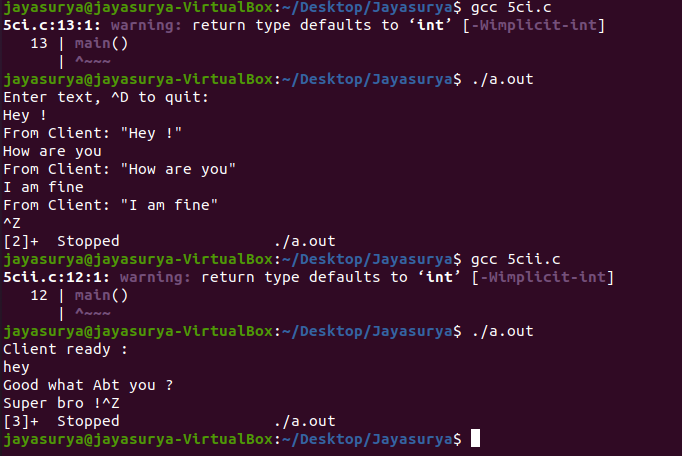
msgsnd(msqid, &mq, len+1, 0);

}

printf("Server Disconnected\n");

}

## **OUTPUT:**



## **RESULT:**

Thus C program to exchange messages between server time using message queue is written, executed and output is obtained.

**Ex. 5d Demonstrate communication between processes using shared memory**

# 

## **AIM:**

To write a c program to demonstrate communication between processes using shared memory

## **ALGORITHM:**

**Server**

1. Initialize size of shared memory shmsize to 27.

2. Initialize key to 2013 (some random value).

3. Create a shared memory segment using shmget with key & IPC\_CREAT as parameter.

a. If shared memory identifier shmid is -1, then stop.

4. Display shmid.

5. Attach server process to the shared memory using shmmat with shmid as parameter.

a. If pointer to the shared memory is not obtained, then stop.

6. Clear contents of the shared region using memset function.

7. Write a–z onto the shared memory.

8. Wait till client reads the shared memory contents

9. Detatch process from the shared memory using shmdt system call.

10. Remove shared memory from the system using shmctl with IPC\_RMID argument

11. Stop

**Client**

1. Initialize size of shared memory shmsize to 27.

2. Initialize key to 2013 (same value as in server).

3. Obtain access to the same shared memory segment using same key.

a. If obtained then display the shmid else print "Server not started"

4. Attach client process to the shared memory using shmmat with shmid as parameter.

a. If pointer to the shared memory is not obtained, then stop.

5. Read contents of shared memory and print it.

6. After reading, modify the first character of shared memory to '\*'

7. Stop

**PROGRAM:**

## **Server**

#include <stdio.h>

#include<stdlib.h>

#include<sys/un.h>

#include<sys/types.h>

#include<sys/ipc.h>

#include<sys/shm.h>

#define shmsize 27

main()

{

char c; int

shmid;

key\_t key = 2013; char

\*shm, \*s;

if ((shmid = shmget(key, shmsize, IPC\_CREAT|0666)) < 0)

{

perror("shmget");

exit(1);

}

printf("Shared memory id : %d\n", shmid);

if ((shm = shmat(shmid, NULL, 0)) == (char \*) -1)

{

perror("shmat");

exit(1);

}

memset(shm, 0, shmsize); s =

shm;

printf("Writing (a-z) onto shared memory\n"); for (c = 'a';

c <= 'z'; c++)

\*s++ = c; \*s

= '\0';

while (\*shm != '\*');

printf("Client finished reading\n");

if(shmdt(shm) != 0)

fprintf(stderr, "Could not close memory segment.\n");

shmctl(shmid, IPC\_RMID, 0);

}

**Client**

#include <stdio.h>

#include<stdlib.h>

#include<sys/types.h>

#include<sys/ipc.h>

#include<sys/shm.h>

#define shmsize 27

main()

{

int shmid;

key\_t key = 2013;

char \*shm, \*s;

if ((shmid = shmget(key, shmsize, 0666)) < 0)

{

printf("Server not started\n"); exit(1);

}

else

printf("Accessing shared memory id : %d\n",shmid);

if ((shm = shmat(shmid, NULL, 0)) == (char \*) -1)

{

perror("shmat");

exit(1);

}

printf("Shared memory contents:\n"); for (s =

shm; \*s != '\0'; s++)

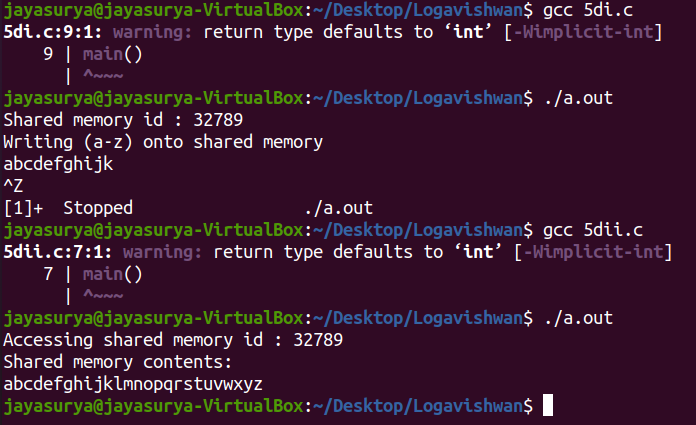
putchar(\*s);

putchar('\n');

\*shm = '\*';

}

**OUTPUT:**



## **RESULT:**

Thus a C program to demonstrate communication between processes using shared memory is executed and the output is obtained.

# 

**Ex. 6a Implementation of Priority scheduling algorithms.**

# 

## **AIM:**

To write a C program for implementation of Priority scheduling algorithms

## **ALGORITHM:**

Step 1: Inside the structure declare the variables.

Step 2: Declare the variable i,j as integer, totwtime and totttime is equal to zero.

Step 3: Get the value of „n‟ assign p and allocate the memory.

Step 4: Inside the for loop get the value of burst time and priority.

Step 5: Assign wtime as zero .

Step 6: Check p[i].pri is greater than p[j].pri .

Step 7: Calculate the total of burst time and waiting time and assign as turnaround time.

Step 8: Stop the program

## **PROGRAM:**

#include<stdio.h>

int main()

{

int bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp,avg\_wt,avg\_tat;

printf("Enter Total Number of Process:");

scanf("%d",&n);

printf("\nEnter Burst Time and Priority\n");

for(i=0;i<n;i++)

{

printf("\nP[%d]\n",i+1);

printf("Burst Time:");

scanf("%d",&bt[i]);

printf("Priority:");

scanf("%d",&pr[i]);

p[i]=i+1; //contains process number

}

//sorting burst time, priority and process number in ascending order using selection sort

for(i=0;i<n;i++)

{

pos=i;

for(j=i+1;j<n;j++)

{

if(pr[j]<pr[pos])

pos=j;

}

temp=pr[i];

pr[i]=pr[pos];

pr[pos]=temp;

temp=bt[i];

bt[i]=bt[pos];

bt[pos]=temp;

temp=p[i];

p[i]=p[pos];

p[pos]=temp;

}

wt[0]=0; //waiting time for first process is zero

//calculate waiting time

for(i=1;i<n;i++)

{

wt[i]=0;

for(j=0;j<i;j++)

wt[i]+=bt[j];

total+=wt[i];

}

avg\_wt=total/n; //average waiting time

total=0;

printf("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time");

for(i=0;i<n;i++)

{

tat[i]=bt[i]+wt[i]; //calculate turnaround time

total += tat[i];

printf("\nP[%d]\t\t %d\t\t %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);

}

avg\_tat=total/n; //average turnaround time

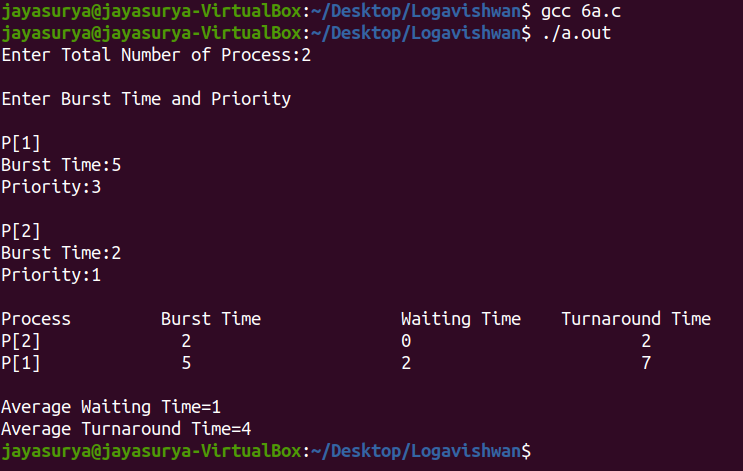
printf("\n\nAverage Waiting Time=%d",avg\_wt);

printf("\nAverage Turnaround Time=%d\n",avg\_tat);

return 0;

}

## **OUTPUT:**



## **RESULT:**

Thus c program program for implementation of Priority scheduling algorithm is written, executed and output is obtained.

# **Ex. 6b Implementation of Round Robin scheduling algorithms**

## **AIM:**

To write a C program for implementation of Round Robin scheduling algorithms

## **ALGORITHM:**

Step 1: Inside the structure declare the variables.

Step 2: Declare the variable i,j as integer, totwtime and totttime is equal to zero.

Step 3: Get the value of „n‟ assign p and allocate the memory.

Step 4: Inside the for loop get the value of burst time and priority and read the time quantum.

Step 5: Assign wtime as zero.

Step 6: Check p[i].pri is greater than p[j].pri .

Step 7: Calculate the total of burst time and waiting time and assign as turnaround time.

Step 8: Stop the program.

## **PROGRAM:**

#include<stdio.h>

int main()

{

int count,j,n,time,remain,flag=0,time\_quantum;

int wait\_time=0,turnaround\_time=0,at[10],bt[10],rt[10];

printf("Enter Total Process:\t ");

scanf("%d",&n);

remain=n;

for(count=0;count<n;count++)

{

printf("Enter Arrival Time and Burst Time for Process Process Number %d :",count+1);

scanf("%d",&at[count]);

scanf("%d",&bt[count]);

rt[count]=bt[count];

}

printf("Enter Time Quantum:\t");

scanf("%d",&time\_quantum);

printf("\n\nProcess\t|Turnaround Time|Waiting Time\n\n");

for(time=0,count=0;remain!=0;)

{

if(rt[count]<=time\_quantum && rt[count]>0)

{

time+=rt[count];

rt[count]=0;

flag=1;

}

else if(rt[count]>0)

{

rt[count]-=time\_quantum;

time+=time\_quantum;

}

if(rt[count]==0 && flag==1)

{

remain--;

printf("P[%d]\t|\t%d\t|\t%d\n",count+1,time-at[count],time-at[count]-bt[count]);

wait\_time+=time-at[count]-bt[count];

turnaround\_time+=time-at[count];

flag=0;

}

if(count==n-1)

count=0;

else if(at[count+1]<=time)

count++;

else

count=0;

}

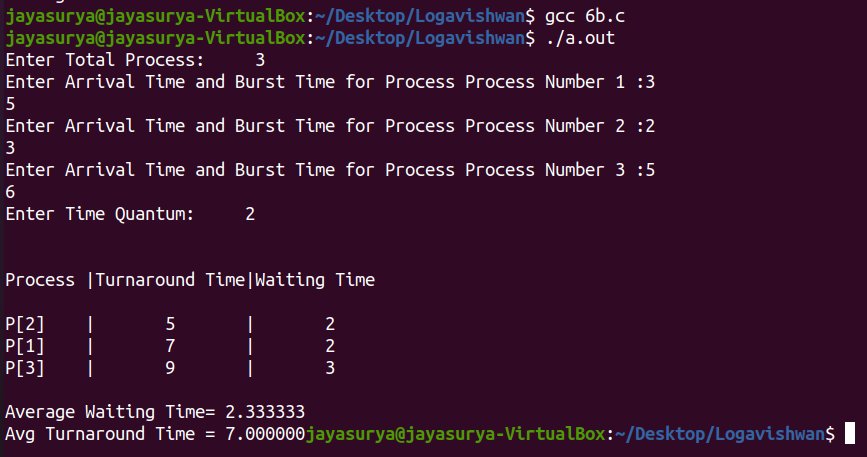
printf("\nAverage Waiting Time= %f\n",wait\_time\*1.0/n);

printf("Avg Turnaround Time = %f",turnaround\_time\*1.0/n);

return 0;

}

## **OUTPUT:**



## **RESULT:**

Thus c program program for implementation of Round robin algorithm is written, executed and output is obtained.

# 

# **Ex. 6c Implementation of FCFS scheduling algorithms.**

## **AIM:**

To write a C program for implementation of FCFS scheduling algorithms

## **ALGORITHM:**

Step 1: Inside the structure declare the variables.

Step 2: Declare the variable i,j as integer,totwtime and totttime is equal to zero.

Step 3: Get the value of „n‟ assign pid as I and get the value of p[i].btime.

Step 4: Assign p[0] wtime as zero and tot time as btime and inside the loop calculate wait time

and turnaround time.

Step 5: Calculate total wait time and total turnaround time by dividing by total number of

process.

Step 6: Print total wait time and total turnaround time.

Step 7: Stop the program.

## **PROGRAM:**

#include<stdio.h>

int main()

{

int n,bt[20],wt[20],tat[20],avwt=0,avtat=0,i,j;

printf("Enter total number of processes(maximum 20):");

scanf("%d",&n);

printf("\nEnter Process Burst Time\n");

for(i=0;i<n;i++)

{

printf("P[%d]:",i+1);

scanf("%d",&bt[i]);

}

wt[0]=0; //waiting time for first process is 0

//calculating waiting time

for(i=1;i<n;i++)

{

wt[i]=0;

for(j=0;j<i;j++)

wt[i]+=bt[j];

}

printf("\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time");

//calculating turnaround time

for(i=0;i<n;i++)

{

tat[i]=bt[i]+wt[i];

avwt += wt[i];

avtat += tat[i];

printf("\nP[%d]\t\t%d\t\t%d\t\t%d",i+1,bt[i],wt[i],tat[i]);

}

avwt/=i;

avtat/=i;

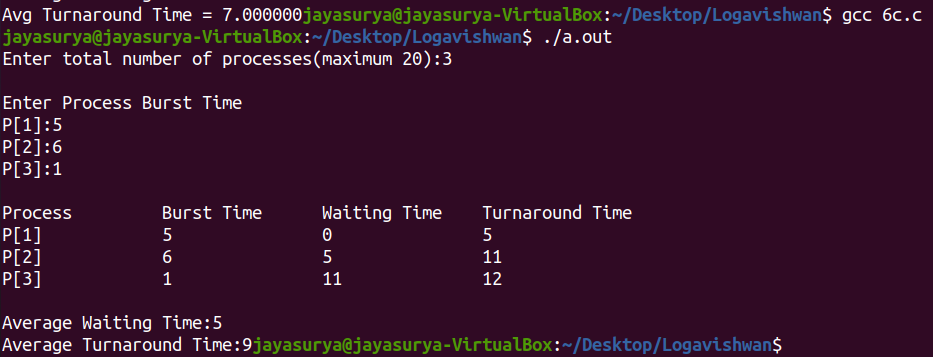
printf("\n\nAverage Waiting Time:%d",avwt);

printf("\nAverage Turnaround Time:%d",avtat);

return 0;

}

## **OUTPUT:**



## **RESULT:**

# Thus c program program for implementation of FCFS scheduling algorithm is written, executed and output is obtained.

# 

# 

# 

# 

**Ex. 6d Implementation of SJF scheduling algorithms.**

## **AIM:**

To write a C program for implementation of SJF scheduling algorithms

## **ALGORITHM:**

Step 1: Inside the structure declare the variables.

Step 2: Declare the variable i,j as integer,totwtime and totttime is equal to zero.

Step 3: Get the value of „n‟ assign pid as I and get the value of p[i].btime.

Step 4: Assign p[0] wtime as zero and tot time as btime and inside the loop calculate wait time

and turnaround time.

Step 5: Calculate total wait time and total turnaround time by dividing by total number of

process.

Step 6: Print total wait time and total turnaround time.

Step 7: Stop the program.

## **PROGRAM:**

#include<stdio.h>

void main()

{

int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;

float avg\_wt,avg\_tat;

printf("Enter number of process:");

scanf("%d",&n);

printf("\nEnter Burst Time:\n");

for(i=0;i<n;i++)

{

printf("p%d:",i+1);

scanf("%d",&bt[i]);

p[i]=i+1; //contains process number

}

//sorting burst time in ascending order using selection sort

for(i=0;i<n;i++)

{

pos=i;

for(j=i+1;j<n;j++)

{

if(bt[j]<bt[pos])

pos=j;

}

temp=bt[i];

bt[i]=bt[pos];

bt[pos]=temp;

temp=p[i];

p[i]=p[pos];

p[pos]=temp;

}

wt[0]=0; //waiting time for first process will be zero

//calculate waiting time

for(i=1;i<n;i++)

{

wt[i]=0;

for(j=0;j<i;j++)

wt[i]+=bt[j];

total+=wt[i];

}

avg\_wt=(float)total/n; //average waiting time

total=0;

printf("\nProcess\tBurst Time \tWaiting Time\tTurnaround Time");

for(i=0;i<n;i++)

{

tat[i]=bt[i]+wt[i]; //calculate turnaround time

total+=tat[i];

printf("\np%d\t\t %d\t\t%d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);

}

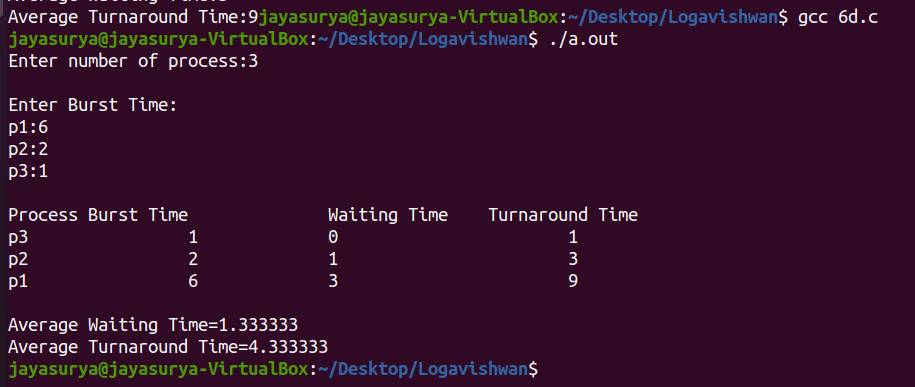
avg\_tat=(float)total/n; //average turnaround time

printf("\n\nAverage Waiting Time=%f",avg\_wt);

printf("\nAverage Turnaround Time=%f\n",avg\_tat);

}

## **OUTPUT:**



## **RESULT:**

Thus c program program for implementation of SJF scheduling algorithm is written, executed and output is obtained.

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

**Ex. 7 Implementation of semaphores**

## **AIM:**

To write a C-program to implement the producer – consumer problem using

semaphores.

## **ALGORITHM:**

Step 1: Start the program.

Step 2: Declare the required variables.

Step 3: Initialize the buffer size and get maximum item you want to produce.

Step 4: Get the option, which you want to do either producer, consumer or exit from the

operation.

Step 5: If you select the producer, check the buffer size if it is full the producer should not

produce the item or otherwise produce the item and increase the value buffer size.

Step 6: If you select the consumer, check the buffer size if it is empty the consumer should not

consume the item or otherwise consume the item and decrease the value of buffer size.

Step 7: If you select exit come out of the program.

Step 8: Stop the program.

## **PROGRAM:**

#include<stdio.h>

int mutex=1,full=0,empty=3,x=0;

main()

{

int n;

void producer();

void consumer();

int wait(int);

int signal(int);

printf("\n1.PRODUCER\n2.CONSUMER\n3.EXIT\n");

while(1) {

printf("\nENTER YOUR CHOICE\n");

scanf("%d",&n);

switch(n)

{ case 1:

if((mutex==1)&&(empty!=0))

producer();

else

printf("BUFFER IS FULL");

break; case 2:

if((mutex==1)&&(full!=0))

consumer();

else

printf("BUFFER IS EMPTY");

break;

case 3:

exit(0);

break;

}

}

}

int wait(int s) {

return(--s); }

int signal(int s) {

return(++s); }

void producer() {

mutex=wait(mutex);

full=signal(full);

empty=wait(empty);

x++;

printf("\nproducer produces the item%d",x);

mutex=signal(mutex); }

void consumer() {

mutex=wait(mutex);

full=wait(full);

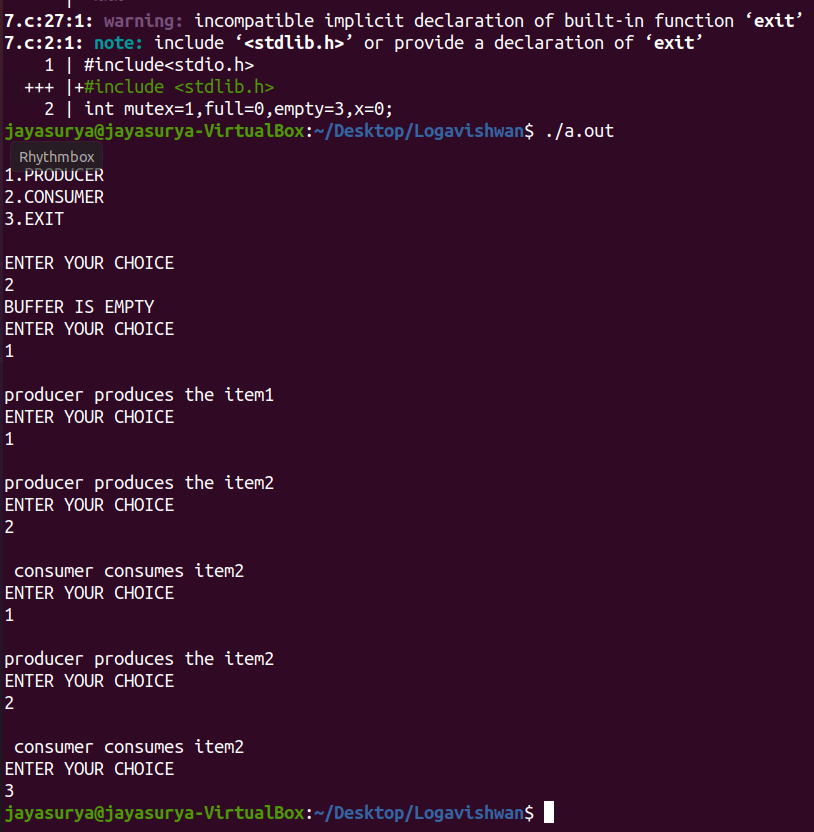
empty=signal(empty);

printf("\n consumer consumes item%d",x);

x--;

mutex=signal(mutex); }

## **OUTPUT:**



## **RESULT:**

Thus C-program to implement the producer – consumer problem using

Semaphore is written, executed and output is obtained.

**Ex. 8 Implement Banker's algorithm for Deadlock avoidance**

## **AIM:**

To write a C program to implement a banker's algorithm for deadlock avoidance.

## **ALGORITHM:**

Step 1: Start the program.

Step 2: Declare the memory for the process.

Step 3: Read the number of process, resources, allocation matrix and available matrix.

Step 4: Compare each and every process using the banker's algorithm.

Step 5: If the process is in safe state then it is a not a deadlock process otherwise it is a

deadlock process

Step 6: produce the result of state of process

Step 7: Stop the program

## **PROGRAM:**

#include<stdio.h>

#include<conio.h>

int max[100][100];

int alloc[100][100];

int need[100][100];

int avail[100];

int n,r;

void input();

void show();

void cal();

int main()

{

int i,j;

printf("Banker's Algorithm\n");

input();

show();

cal();

getch();

return 0;

}

void input()

{

int i,j;

printf("Enter the no of Processes\t");scanf("%d",&n);

printf("Enter the no of resources instances\t");

scanf("%d",&r);

printf("Enter the Max Matrix\n");

for(i=0;i<n;i++)

{

for(j=0;j<r;j++)

{

scanf("%d",&max[i][j]);

}}

printf("Enter the Allocation Matrix\n");

for(i=0;i<n;i++)

{

for(j=0;j<r;j++)

{

scanf("%d",&alloc[i][j]);

}}

printf("Enter the available Resources\n");

for(j=0;j<r;j++)

{

scanf("%d",&avail[j]);

}}

void show()

{

int i,j;

printf("Process\t Allocation\t Max\t Available\t");

for(i=0;i<n;i++)

{

printf("\nP%d\t ",i+1);

for(j=0;j<r;j++)

{

printf("%d ",alloc[i][j]);

}

printf("\t");

for(j=0;j<r;j++)

{

printf("%d ",max[i][j]);

}

printf("\t");

if(i==0)

{

for(j=0;j<r;j++)

printf("%d ",avail[j]);

}}}

void cal()

{

int finish[100],temp,need[100][100],flag=1,k,c1=0;

int safe[100];

int i,j;

for(i=0;i<n;i++)

{

finish[i]=0;

}

//find need matrix

for(i=0;i<n;i++)

{

for(j=0;j<r;j++)

{

need[i][j]=max[i][j]-alloc[i][j];

}}

printf("\n");

while(flag)

{

flag=0;

for(i=0;i<n;i++)

{

int c=0;

for(j=0;j<r;j++)

{

if((finish[i]==0)&&(need[i][j]<=avail[j]))

{

c++;

if(c==r)

{

for(k=0;k<r;k++)

{

avail[k]+=alloc[i][j];

finish[i]=1;

flag=1;

}

printf("P%d->",i);

if(finish[i]==1){

i=n;

}}}}}}

for(i=0;i<n;i++){

if(finish[i]==1){

c1++;

}

else

{printf("P%d->",i);

}}

if(c1==n){

printf("\n The system is in safe state");

}

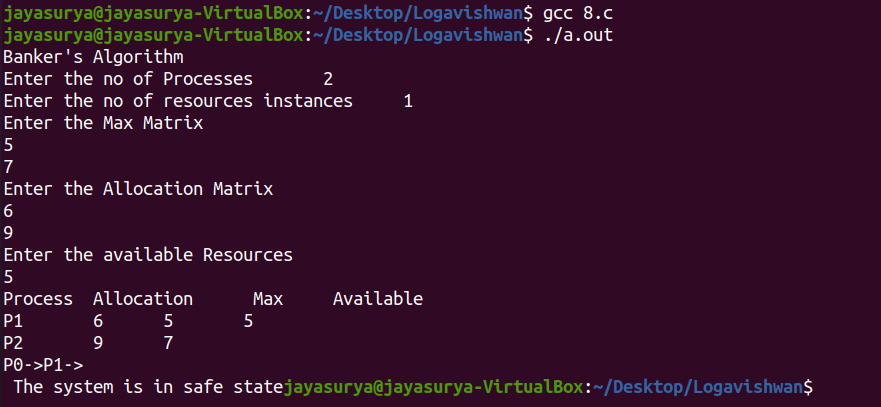
else{

printf("\n Process are in dead lock");

printf("\n System is in unsafe state");

}}

## **OUTPUT:**



## **RESULT:**

Thus C program to implement a banker's algorithm for deadlock avoidance is written, executed and output is obtained.

**Ex. 9 Implement Banker's algorithm for Deadlock detection**

## **AIM:**

To write a C program to implement an algorithm for deadlock detection.

## **ALGORITHM:**

Step 1: Start the program.

Step 2: Declare the memory for the process.

Step 3: Read the number of process, resources, allocation matrix and available matrix.

Step 4: Compare each and every process using the banker‟s algorithm.

Step 5: If the process is in safe state then it is a not a deadlock process otherwise it is a

deadlock process

Step 6: produce the result of state of process

Step 7: Stop the program

## **PROGRAM:**

#include<stdio.h>

int max[100][100];

int alloc[100][100];

int need[100][100];

int avail[100];

int n,r;

void input();

void show();

void cal();

int main()

{

int i,j;

printf("Deadlock Detection Algorithm\n");

input();

show();

cal();

return 0;

}

void input()

{int i,j;

printf("Enter the no of Processes\t");

scanf("%d",&n);printf("Enter the no of resource instances\t");

scanf("%d",&r);

printf("Enter the Max Matrix\n");

for(i=0;i<n;i++)

{for(j=0;j<r;j++)

{

scanf("%d",&max[i][j]);

}}

printf("Enter the Allocation Matrix\n");

for(i=0;i<n;i++)

{for(j=0;j<r;j++)

{

scanf("%d",&alloc[i][j]);

}}

printf("Enter the available Resources\n");

for(j=0;j<r;j++)

{

scanf("%d",&avail[j]);

}}

void show()

{

int i,j;

printf("Process\t Allocation\t Max\t Available\t");

for(i=0;i<n;i++)

{

printf("\nP%d\t ",i+1);

for(j=0;j<r;j++)

{

printf("%d ",alloc[i][j]);

}

printf("\t");

for(j=0;j<r;j++)

{printf("%d ",max[i][j]);

}

printf("\t");

if(i==0)

{

for(j=0;j<r;j++)

printf("%d ",avail[j]);

}}}

void cal()

{ int finish[100],temp,need[100][100],flag=1,k,c1=0;

int dead[100];

int safe[100];

int i,j;

for(i=0;i<n;i++)

{finish[i]=0;

}

//find need matrix

for(i=0;i<n;i++)

{for(j=0;j<r;j++)

{

need[i][j]=max[i][j]-alloc[i][j];

}}

while(flag)

{flag=0;

for(i=0;i<n;i++)

{int c=0;

for(j=0;j<r;j++)

{if((finish[i]==0)&&(need[i][j]<=avail[j]))

{c++;

if(c==r)

{

for(k=0;k<r;k++)

{avail[k]+=alloc[i][j];

finish[i]=1;

flag=1;

}//printf("\nP%d",i);

if(finish[i]==1)

{i=n;

}}}}}}

j=0;

flag=0;

for(i=0;i<n;i++)

{

if(finish[i]==0)

{dead[j]=i;

j++;

flag=1;

}}

if(flag==1)

{

printf("\n\nSystem is in Deadlock and the Deadlock process are\n");

for(i=0;i<n;i++)

{printf("P%d\t",dead[i]);

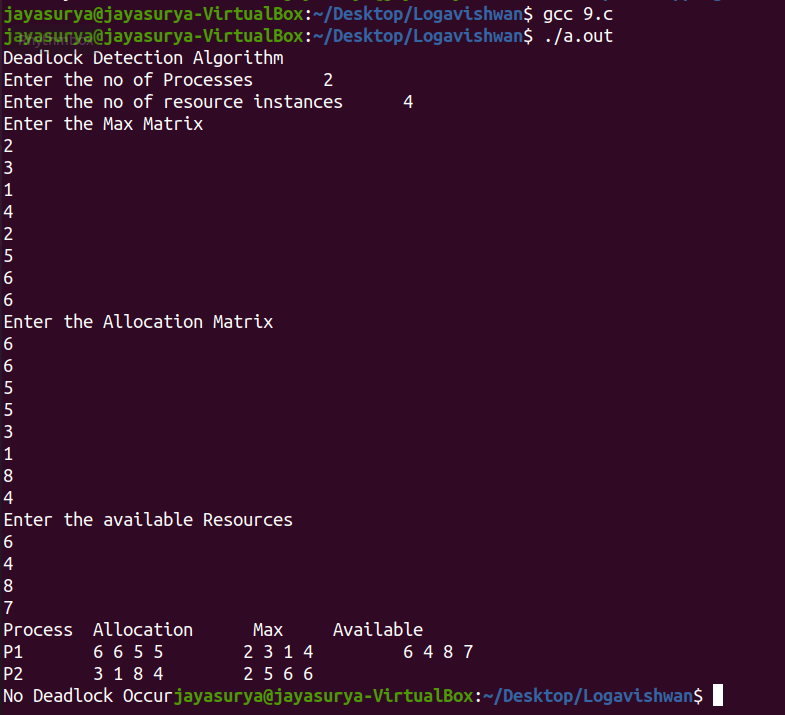
}}

else

{

printf("\nNo Deadlock Occur"); }}

## **OUTPUT:**



## **RESULT:**

Thus C program to implement a banker's algorithm for deadlock detection is written, executed and output is obtained.