**INSTALLATION AND CONFIGURATION OF LINUX**

**Ex No: 1**

**Aim:**To install and configure Linux operating system in a Virtual Machine.

**Installation/Configuration Steps:**

1. Install the required packages for virtualization

     dnf install xen virt-manager qemu libvirt

2. Configure xend to start up on boot

     systemctl enable virt-manager.service

3. Reboot the machine

     Reboot

4. Create Virtual machine by first running virt-manager

     virt-manager &

5. Click on File and then click to connect to localhost

6. In the base menu, right click on the localhost(QEMU) to create a new VM

7. Select Linux ISO image

8. Choose puppy-linux.iso then kernel version

9. Select CPU and RAM limits

10.Create default disk image to 8 GB

11.Click finish for creating the new VM with PuppyLinux

**RESULT:**

**BASIC LINUX COMMANDS**

**Ex No: 2**

**1.1 GENERAL PURPOSE COMMANDS**

1. The ‘date’ command:

The date command display the current date with day of week, month, day, time (24 hours clock) and the year.

SYNTAX:  $ date

The date command can also be used with following format.

|  |  |  |
| --- | --- | --- |
| Format | Purpose | Example |
| + %m | To display only month | $ date + %m |
| + %h | To display month name | $ date + %h |
| + %d | To display day of  month | $ date + %d |
| + %y | To display last two digits of the year | $ date + %y |
| + %H | To display Hours | $ date + %H |
| + %M | To display Minutes | $ date + %M |
| + %S | To display Seconds | $ date + %S |

2.  The echo’command:

The echo command is used to print the message on the screen.

SYNTAX:  $ echo

EXAMPLE: $ echo “God is Great”

3. The ‘cal’ command:

The cal command displays the specified month or year calendar.

SYNTAX:  $ cal [month] [year]

EXAMPLE: $ cal Jan 2012

4. The ‘bc’ command:

Unix offers an online calculator and can be invoked by the command bc.

SYNTAX:  $ bc

EXAMPLE: bc  –l

16/4

5/2

5.  The ‘who’ command

The who command is used to display the data about all the users who are currently logged into the system.

SYNTAX:  $ who

6.  The ‘who am i’ command

The who am i command displays data about login details of the user.

SYNTAX:  $ who am i

7.  The ‘id’ command

The id command displays the numerical value corresponding to your login.

SYNTAX:  $ id

8.  The ‘tty’ command

The tty (teletype) command is used to know the terminal name that we are using.

SYNTAX:  $ tty

9.  The ‘clear’ command

The clear command is used to clear the screen of your terminal.

SYNTAX:  $ clear

10.  The ‘man’ command

The man command gives you complete access to the Unix commands.

SYNTAX:  $ man [command]

11.  The ‘ps’ command

The ps command is used to the process currently alive in the machine with the 'ps' (process status) command, which displays information about process that are alive when you run the command. 'ps;' produces a snapshot of machine activity.

SYNTAX:  $ ps

EXAMPLE:    $ ps

        $ ps –e

$ps  -aux

12.  The ‘uname’ command

The uname command is used to display relevant details about the operating system on the standard output.

-m -> Displays the machine id (i.e., name of the system hardware)

-n -> Displays the name of the network node. (host name)

-r -> Displays the release number of the operating system.

-s -> Displays the name of the operating system (i.e.. system name)

-v -> Displays the version of the operating system.

-a -> Displays the details of all the above five options.

SYNTAX:  $ uname [option]

EXAMPLE: $ uname -a

**1.2 DIRECTORY COMMANDS**

1. The ‘pwd’ command:

The pwd (print working directory) command displays the current working directory.

SYNTAX:  $ pwd

2. The ‘mkdir’ command:

The mkdir is used to create an empty directory in a disk.

SYNTAX:  $ mkdir dirname

EXAMPLE: $ mkdir receee

3. The ‘rmdir’ command:

The rmdir is used to remove a directory from the disk. Before removing a directory, the directory must be empty (no files and directories).

SYNTAX:  $ rmdir dirname

EXAMPLE: $ rmdir receee

4. The ‘cd’ command:

The cd command is used to move from one directory to another.

SYNTAX:  $ cd dirname

EXAMPLE: $ cd receee

5. The ‘ls’ command:

The ls command displays the list of files in the current working directory.

SYNTAX:  $ ls

EXAMPLE: $ ls

$ ls –l

$ ls –a

**1.3 FILE HANDLING COMMANDS**

1. The ‘cat’ command:

The cat command is used to create a file.

SYNTAX:  $ cat > filename

EXAMPLE: $ cat > rec

2. The ‘Display contents of a file’ command:

The cat command is also used to view the contents of a specified file.

SYNTAX:  $ cat filename

3. The ‘cp’ command:

The cp command is used to copy the contents of one file to another and copies the file from one place to another.

SYNTAX:  $ cp oldfile newfile

EXAMPLE: $ cp cse ece

4. The ‘rm’ command:

The rm command is used to remove or erase an existing file

SYNTAX:  $ rm filename

EXAMPLE: $ rm rec

    $ rm –f  rec

Use option –fr  to delete recursively the contents of the directory and its subdirectories.

5. The ‘mv’ command:

The mv command is used to move a file from one place to another. It removes a specified file from its original location and places it in specified location.

SYNTAX:  $ mv oldfile newfile

EXAMPLE: $ mv cse eee

6. The ‘file’ command:

The file command is used to determine the type of file.

SYNTAX:  $ file filename

EXAMPLE: $ file receee

7. The ‘wc’ command:

The wc command is used to count the number of words, lines and characters in a file.

SYNTAX:  $ wc filename

EXAMPLE: $ wc receee

8. The ‘Directing output to a file’ command:

The ls command lists the files on the terminal (screen). Using the redirection operator ‘>’ we can send the output to file instead of showing it on the screen.

SYNTAX:  $ ls > filename

EXAMPLE: $ ls > cseeee

9. The ‘pipes’ command:

The Unix allows us to connect two commands together using these pipes. A pipe ( | )  is an mechanism by which the output of one command can be channeled into the input of another command.

SYNTAX:  $ command1 | command2

EXAMPLE: $ who | wc -l

10. The ‘tee’ command:

While using pipes, we have not seen any output from a command that gets piped into another command. To save the output, which is produced in the middle of a pipe, the tee command is very useful.

SYNTAX:  $ command | tee filename

EXAMPLE: $ who | tee sample | wc -l

11. The ‘Metacharacters of unix’ command:

Metacharacters are special characters that are at higher and abstract level compared to most of other characters in Unix. The shell understands and interprets these metacharacters in a special way.

\* -  Specifies number of characters

?- Specifies a single character

[ ]- used to match a whole set of file names at a command line.

! – Used to Specify Not

EXAMPLE:

$ ls r\*\* - Displays all the files whose name begins with ‘r’

$ ls ?kkk - Displays the files which are having ‘kkk’, from the second characters irrespective of the first character.

$ ls [a-m] – Lists the files whose names begins alphabets from ‘a’ to ‘m’

$ ls [!a-m] – Lists all files other than files whose names begins alphabets from ‘a’ to ‘m’

12. The ‘File permissions’ command:

File permission is the way of controlling the accessibility of file for each of three users namely Users, Groups  and  Others.

  There are three types of file permissions are available, they are

The permissions for each file can be divided into three parts of three bits each.

|  |  |
| --- | --- |
| First three bits | Owner of the file |
| Next three bits | Group to which owner of the file belongs |
| Last three bits | Others |

EXAMPLE: $ ls college

-rwxr-xr-- 1 Lak std 1525 jan10 12:10 college

Where,

-rwx    The file is readable, writable and executable by the owner of the file.

Lak     Specifies Owner of the file.

r-x      Indicates the absence of the write permission by the Group owner of the file.

Std     Is the Group Owner of the file.

r--      Indicates read permissions for others.

13. The ‘chmod’ command:

The chmod command is used to set the read, write and execute permissions for all categories of users for file.

SYNTAX:  $ chmod category operation permission file

|  |  |  |
| --- | --- | --- |
| Category | Operation | permission |
| u-users | + assign | r-read |
| g-group | -Remove | w-write |
| o-others | = assign absolutely | x-execute |
| a-all |  |  |

EXAMPLE:

$ chmod u –wx college

Removes write & execute permission for users for ‘college’ file.

$ chmod u +rw, g+rw college

Assigns read & write permission for users and groups for ‘college’ file.

$ chmod g=wx college

Assigns absolute permission for groups of all read, write and execute permissions for  ‘college’ file.

14. The ‘Octal Notations’ command:

The file permissions can be changed using octal notations also. The octal notations for file permission are

|  |  |
| --- | --- |
| Read permission | 4 |
| Write permission | 2 |
| Execute permission | 1 |

EXAMPLE:

$ chmod 761 college

Assigns all permission to the owner, read and write permissions to the group and only executable permission to the others for ‘college’ file.

**1.4 GROUPING COMMANDS**

1. The ‘semicolon’ command:

The semicolon(;) command is used to separate multiple commands at the command line.

SYNTAX:  $ command1;command2;command3…………….;commandn

EXAMPLE: $ who;date

2. The ‘&&’ operator:

The ‘&&’ operator signifies the logical AND operation in between two or more valid Unix commands.It means that only if the first command is successfully executed, then the next command will executed.

SYNTAX:  $ command1 && command && command3…………….&&commandn

EXAMPLE: $ who && date

3. The ‘||’ operator:

The ‘||’ operator signifies the logical OR operation in between two or more valid Unix commands.It means, that only if the first command will happen to be un successfully,it will continue to execute next commands.

SYNTAX:  $ command1 || command || command3…………….||commandn

EXAMPLE: $ who || date

1.5 FILTERS

1. The head filter

It displays the first ten lines of a file.

SYNTAX:  $ head filename

EXAMPLE: $ head college Display the top ten lines.

$ head -5 college Display the top five lines.

2. The tail filter

It displays ten lines of a file from the end of the file.

SYNTAX:  $ tail filename

EXAMPLE: $ tail college Display the last ten lines.

$tail -5 college Display the last five lines.

3. The more filter:

The pg command shows the file page by page.

SYNTAX:  $ ls –l | more

4. The ‘grep’ command:

This command is used to search for a particular pattern from a file or from the standard input and display those lines on the standard output. “Grep” stands for “global search for regular expression.”

SYNTAX:  $ grep [pattern] [file\_name]

EXAMPLE: $  cat > student

Arun cse

Ram ece

Kani cse

$  grep “cse”  student

 Arun cse

Kani cse

5. The ‘sort’ command:

The sort command is used to sort the contents of a file. The sort command reports only to the screen, the actual file remains unchanged.

SYNTAX:  $ sort filename

EXAMPLE: $  sort college

OPTIONS:

|  |  |
| --- | --- |
| Command | Purpose |
| Sort –r college | Sorts and displays the file contents in reverse order |
| Sort –c college | Check if the file is sorted |
| Sort –n college | Sorts numerically |
| Sort –m college | Sorts numerically in reverse order |
| Sort –u college | Remove duplicate records |
| Sort –l college | Skip the column with +1 (one) option.Sorts according to second column |

6. The ‘nl’ command:

The nl filter adds lines numbers to a file and it displays the file and not provides access to edit but simply displays the contents on the screen.

SYNTAX:  $ nl filename

EXAMPLE: $  nl college

7. The ‘cut’ command:

We can select specified fields from a line of text using cut command.

SYNTAX:  $ cut -c filename

EXAMPLE: $ cut -c college

OPTION:

-c – Option cut on the specified character position from each line.

**1.5 OTHER ESSENTIAL COMMANDS**

1. **free**

Display amount of free and used physical and swapped memory system.

    synopsis-      free [options]

    example

    [root@localhost ~]# free -t

                          total          used             free         shared    buff/cache    available

    Mem:        4044380      605464     2045080      148820     1393836     3226708

    Swap:       2621436           0           2621436

    Total:        6665816      605464      4666516

2. **top**

         It provides a dynamic real-time view of processes in the system.

    synopsis-   top [options]

    example

    [root@localhost ~]# top

    top - 08:07:28 up 24 min,  2 users,  load average: 0.01, 0.06, 0.23

    Tasks: 211 total,   1 running, 210 sleeping,   0 stopped,   0 zombie

    %Cpu(s):  0.8 us,  0.3 sy,  0.0 ni, 98.9 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st

    KiB Mem :  4044380 total,  2052960 free,   600452 used,  1390968 buff/cache

    KiB Swap:  2621436 total,  2621436 free,        0 used.  3234820 avail Mem

   PID USER  PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND

  1105 root      20   0  175008  75700  51264 S   1.7  1.9   0:20.46   Xorg

  2529 root      20   0   80444   32640  24796 S   1.0  0.8   0:02.47   gnome-term

3. **ps**

It reports the snapshot of current processes

    synopsis-     ps [options]

    example

    [root@localhost ~]# ps -e

   PID TTY          TIME CMD

     1 ?        00:00:03             systemd

     2 ?        00:00:00             kthreadd

     3 ?        00:00:00             ksoftirqd/0

4. **vmstat**

     It reports virtual memory statistics

    synopsis-      vmstat [options]

    example

    [root@localhost ~]# vmstat

    procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----

    r  b   swpd    free        buff  cache        si   so    bi    bo   in   cs  us sy id wa st

    0  0      0   1879368   1604 1487116    0    0    64     7   72  140  1  0 97  1  0

5. **df**

   It displays the amount of disk space available in file-system.

**S**ynopsis-   df [options]

    example

    [root@localhost ~]# df

    Filesystem              1K-blocks     Used Available Use% Mounted on

    devtmpfs                  2010800        0   2010800   0% /dev

    tmpfs                        2022188      148   2022040   1% /dev/shm

    tmpfs                        2022188     1404   2020784   1% /run

    /dev/sda6                  487652      168276    289680           37% /boot

**6.** **ping**

It is used verify that a device can communicate with another on network. PING stands for   Packet Internet Groper.

synopsis- ping [options]

[root@localhost ~]# ping 172.16.4.1

PING 172.16.4.1 (172.16.4.1) 56(84) bytes of data.

64 bytes from 172.16.4.1: icmp\_seq=1 ttl=64 time=0.328 ms

64 bytes from 172.16.4.1: icmp\_seq=2 ttl=64 time=0.228 ms

64 bytes from 172.16.4.1: icmp\_seq=3 ttl=64 time=0.264 ms

64 bytes from 172.16.4.1: icmp\_seq=4 ttl=64 time=0.312 ms

^C

--- 172.16.4.1 ping statistics ---

4 packets transmitted, 4 received, 0% packet loss, time 3000ms

rtt min/avg/max/mdev = 0.228/0.283/0.328/0.039 ms

**7.** **ifconfig**

It is used configure network interface.

    synopsis- ifconfig [options]

    example

    [root@localhost ~]# ifconfig

    enp2s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500

         inet 172.16.6.102  netmask 255.255.252.0  broadcast 172.16.7.255

         inet6 fe80::4a0f:cfff:fe6d:6057  prefixlen 64  scopeid 0x20<link>

         ether 48:0f:cf:6d:60:57  txqueuelen 1000  (Ethernet)

         RX packets 23216  bytes 2483338 (2.3 MiB)

         RX errors 0  dropped 5  overruns 0  frame 0

         TX packets 1077  bytes 107740 (105.2 KiB)

         TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

**8.** **traceroute**

It tracks the route the packet takes to reach the destination.

      synopsis- traceroute [options]

      example

      [root@localhost ~]# traceroute www.rajalakshmi.org

      traceroute to www.rajalakshmi.org (220.227.30.51), 30 hops max, 60 byte packets

      1  gateway (172.16.4.1)  0.299 ms  0.297 ms  0.327 ms

      2  220.225.219.38 (220.225.219.38)  6.185 ms  6.203 ms  6.189 ms

**RESULT:**

**PROGRAM:**

#!/bin/bash

# Sample script written for Part 4 of the RHCE series

# This script will return the following set of system information:

# -Hostname information:

echo -e "\e[31;43m\*\*\*\*\* HOSTNAME INFORMATION \*\*\*\*\*\e[0m"

hostnamectl

echo ""

# -File system disk space usage:

echo -e "\e[31;43m\*\*\*\*\* FILE SYSTEM DISK SPACE USAGE \*\*\*\*\*\e[0m"

df -h

echo ""

# -Free and used memory in the system:

echo -e "\e[31;43m \*\*\*\*\* FREE AND USED MEMORY \*\*\*\*\*\e[0m"

free

echo ""

# -System uptime and load:

echo -e "\e[31;43m\*\*\*\*\* SYSTEM UPTIME AND LOAD \*\*\*\*\*\e[0m"

uptime

echo ""

# -Logged-in users:

echo -e "\e[31;43m\*\*\*\*\* CURRENTLY LOGGED-IN USERS \*\*\*\*\*\e[0m"

who

echo ""

# -Top 5 processes as far as memory usage is concerned

echo -e "\e[31;43m\*\*\*\*\* TOP 5 MEMORY-CONSUMING PROCESSES \*\*\*\*\*\e[0m"

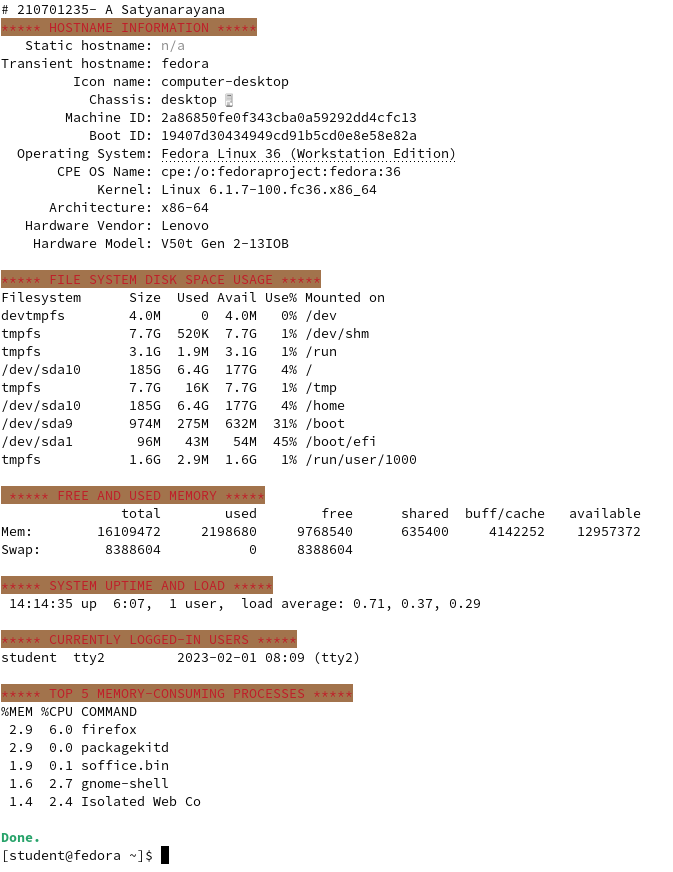
ps -eo %mem,%cpu,comm --sort=-%mem | head -n 6

echo ""

echo -e "\e[1;32mDone.\e[0m"

# 210701235- A Satyanarayana

**OUTPUT:**



**RESULT:**

**PROGRAM:**

**emp.awk**

BEGIN{print "EMPLOYEES DETAILS"}

{#salary should be greater than 6000 and days more than 4

if($2>6000 && $3>4)

{print $1,"\t\t", $2\*$3

pay=pay+ $2\*$3

count=count+1}

}

END{

{#action part

print "no of employees are=", count

print "total pay=",pay

print "average pay=",pay/count

}

}

**emp.dat**

JOE 8000 5

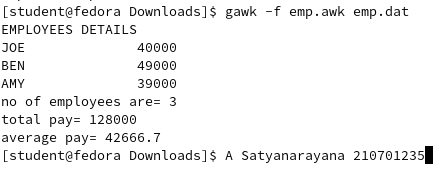
RAM 6000 5

TIM 5000 6

BEN 7000 7

AMY 6500 6

**OUTPUT:**



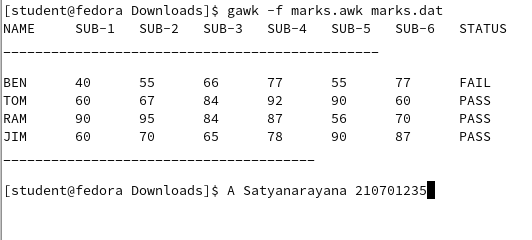
**RESULT:**

**PROGRAM:**

**//marks.awk**  
BEGIN{  
print "NAME","\t","SUB-1","\t","SUB-2","\t","SUB-3","\t","SUB-4","\t","SUB-5","\t","SUB-6","\t","STATUS"  
print"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n"  
 }  
 {  
#BODY  
if ( $2 < 45 || $3 < 45 || $4 < 45 || $5 < 45 || $6 < 45   || $7 < 45){  
 print $1,"\t",$2,"\t",$3,"\t",$4,"\t",$5,"\t", $6,"\t", $7,"\t","FAIL"  
}  
else{  
print $1,"\t",$2,"\t",$3,"\t",$4,"\t",$5,"\t", $6,"\t",$7,"\t","PASS"   
 } }  
END {    
print "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n"  
}  
**//marks.dat**  
**//**Col1- name, Col 2 to Col7 – marks in various  
subjects  
BEN    
40         55        66        77        55       77  
TOM    
60        67        84        92        90       60  
RAM    
90        95        84        87        56       70

JIM    
60           70        65        78        90       87

**OUTPUT:**

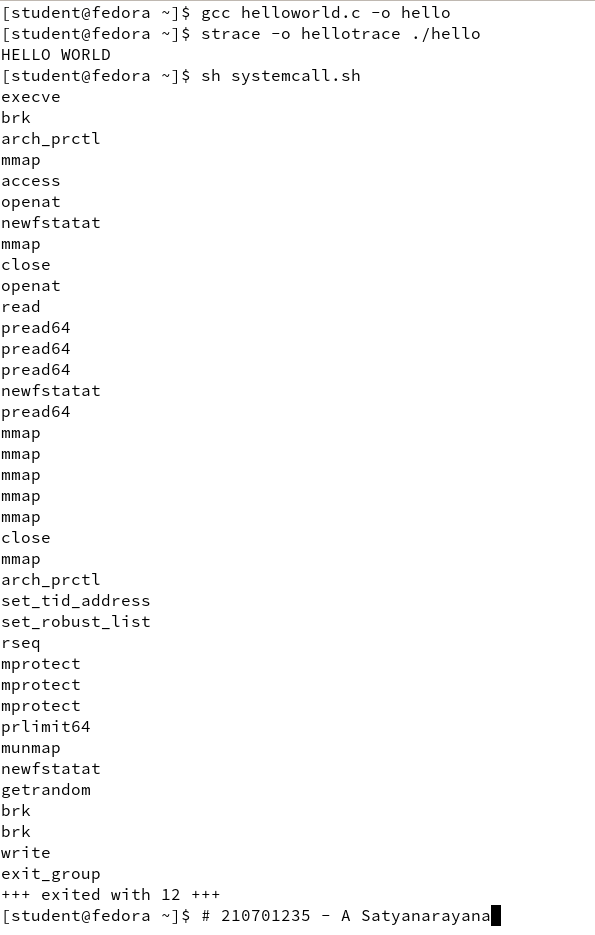


**RESULT:**

**PROGRAM:**

// helloworld.c  
#include<stdio.h>   
void main()   
{  
   printf("Hello World \n");   
}  
// systemcall.sh  
cat hellotrace|cut -f1 -d"("

**OUTPUT:**

****

**RESULT:**

**PROGRAM:**

#include <signal.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

void my\_handler (int sig); /\* function prototype \*/

int main()

{ struct sigaction my\_action;

/\* Part I: Catch SIGINT \*/

my\_action.sa\_handler = my\_handler;

my\_action.sa\_flags = SA\_RESTART;

sigaction(SIGINT, &my\_action, NULL);

printf("Catching SIGINT\n");

sleep(3);

printf(" No SIGINT within 3 seconds\n");

/\* Part II: Ignore SIGINT \*/

my\_action.sa\_handler = SIG\_IGN;

my\_action.sa\_flags = SA\_RESTART;

sigaction(SIGINT, &my\_action, NULL);

printf("Ignoring SIGINT\n");

sleep(3);

printf(" Sleep is over\n");

/\* Part III: Default action for SIGINT \*/

my\_action.sa\_handler = SIG\_DFL;

my\_action.sa\_flags = SA\_RESTART;

sigaction(SIGINT, &my\_action, NULL);

sleep(3);

printf("No SIGINT within 3 seconds\n");

}

void my\_handler (int sig)

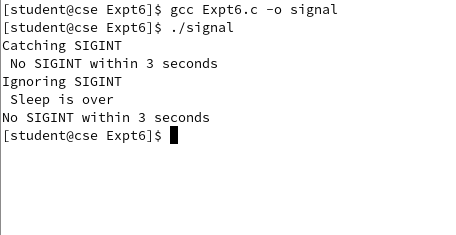
{

printf (" \t I got SIGINT, number %d\n", sig);

exit(0);

}

**OUTPUT:**



**RESULT:**

**PROGRAM:**

**sender.c**

#include<sys/types.h>

#include<sys/ipc.h>

#include<sys/shm.h>

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#define SharedMemSize 50

void main()

{

char c;

int shmid;

key\_t key;

char \* shared\_memory;

key = 5677;

//Create segment with the key specified

if ((shmid = shmget(key, SharedMemSize,

IPC\_CREAT |0666)) < 0)

{

//perror explains error code

perror("shmget");

exit(1);

}

//Attach the segment

if ((shared\_memory = shmat(shmid, NULL, 0)) ==

(char \* ) - 1)

{

perror("shmat");

exit(1);

}

sprintf(shared\_memory, " Welcome to Shared Memory");

sleep(2);

exit(0);

}

**receiver.c**

#include<sys/types.h>

#include<sys/ipc.h>

#include<sys/shm.h>

#include<stdio.h>

#include<stdlib.h>

#define SharedMemSize 50

void main()

{ int shmid;

key\_t key;

char \* shared\_memory;

key = 5677;

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

{

perror("shmget");

exit(1);

}

//Attach the segment to our data space

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

{

perror("shmat");

exit(1);

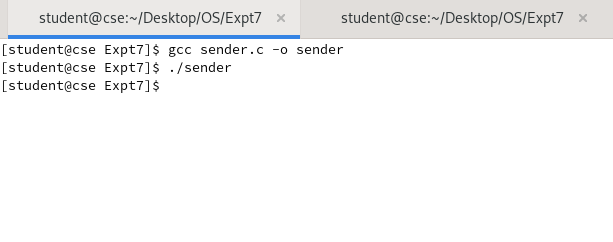
}

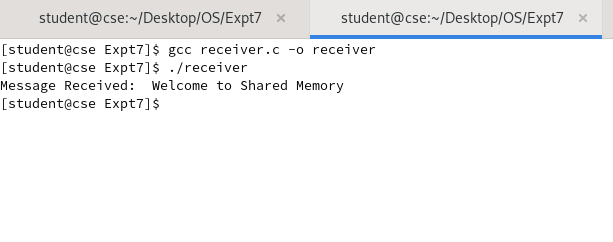
//Read the message sender sent to the shared memory

printf("Message Received: %s \n", shared\_memory);

exit(0);

}

**OUTPUT:**



**RESULT:**

**PROGRAM:**

**writer.py**

import os

import sys

pipe\_name = "/home/student/Desktop/OS/Expt8/rec.fifo"

if not os.path.exists(pipe\_name):

os.mkfifo(pipe\_name)

fifo = open(pipe\_name, "w")

mystring = input("Enter string to write into pipe:")

fifo.write(mystring)

fifo.close()

**reader.py**

import os

import sys

pipe\_name="/home/student/Desktop/OS/Expt8/rec.fifo"

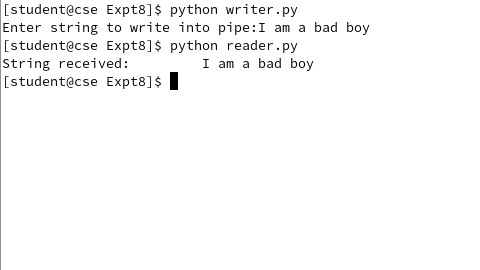
fifo=open(pipe\_name,'r')

stringReceived=fifo.read()

print("String received:\t",stringReceived)

fifo.close()

**OUTPUT:**



**RESULT:**

**PROGRAM:**

#include <stdio.h>

#include <stdlib.h>

void main()

{

    int n, pid[10], at[10], bt[10], ft[10], wt[10], ta[10], i, j, t, stt = 0, totta = 0, totwt = 0;

    float avgta, avgwt;

    printf("ENTER THE NO.OF PROCESSES: ");

    scanf("%d", &n);

    for (i = 1; i <= n; i++)

    {

        pid[i] = i;

        printf("\nENTER THE ARRIVAL TIME: ");

        scanf("%d", &at[i]);

        printf("\nENTER THE BURST TIME: ");

        scanf("%d", &bt[i]);

    }

    for (i = 1; i <= n; i++)

    {

        for (j = i + 1; j <= n; j++)

        {

            if (at[i] > at[j])

            {

                t = pid[i];

                pid[i] = pid[j];

                pid[j] = t;

                t = at[i];

                at[i] = at[j];

                at[j] = t;

                t = bt[i];

                bt[i] = bt[j];

                bt[j] = t;

            }

            stt = at[1];

        }

    }

    printf("\nTHE VALUES OF THE ARRIVAL TIME IS %d", stt);

    for (i = 1; i <= n; i++)

    {

        ft[i] = stt + bt[i];

        wt[i] = stt - at[i];

        ta[i] = ft[i] - at[i];

        totta = totta + ta[i];

        totwt = totwt + wt[i];

        stt = ft[i];

    }

    avgta = (float)totta / n;

    avgwt = (float)totwt / n;

    printf("\nPNO\tARRIVAL TIME\tBURST TIME\tCOMPLETION TIME\t\tWAITING TIME\tTAT\n");

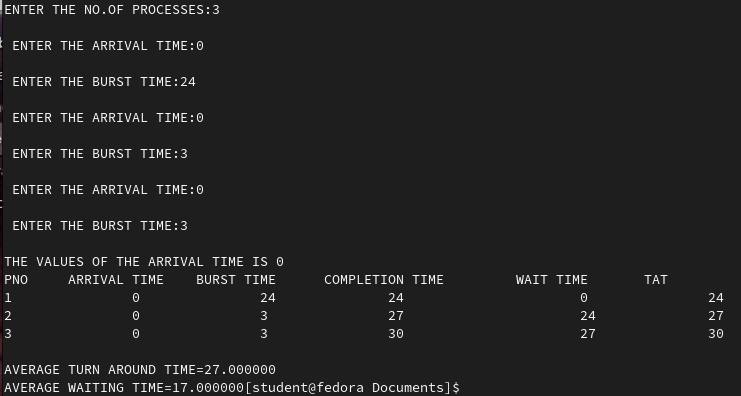
    for (i = 1; i <= n; i++)

        printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", pid[i], at[i], bt[i], ft[i], wt[i], ta[i]);

    printf("\nAVERAGE TURN AROUND TIME=%f", avgta);

    printf("\nAVERAGE WAITING TIME=%f", avgwt);

}

**OUTPUT:**

**RESULT:**

Thus the given program was executed and output is verified.

**PROGRAM:**

#include <stdio.h>

void main()

{

int i, j, t, n, stt = 0, pid[10], at[10], bt[10], ft[10], att, wt[10], ta[10], totwt = 0, totta = 0;

float avgwt, avgta;

printf("ENTER THE NUMBER OF PROCESSES: ");

scanf("%d", &n);

printf("\nENTER THE ARRIVAL TIME: ");

scanf("%d", &att);

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

{

pid[i] = i;

at[i] = att;

printf("\nENTER THE BURST TIME: ");

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

}

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

{

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

{

if (bt[i] > bt[j])

{

t = pid[i];

pid[i] = pid[j];

pid[j] = t;

t = bt[i];

bt[i] = bt[j];

bt[j] = t;

}

stt = att;

}

}

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

{

ft[i] = stt + bt[i];

wt[i] = stt - at[i];

ta[i] = ft[i] - at[i];

totta = totta + ta[i];

totwt = totwt + wt[i];

stt = ft[i];

}

avgwt = (float)totwt / n;

avgta = (float)totta / n;

printf("\nPNO\tARRIVAL TIME\tBURST TIME\tCOMPLETION TIME\t\tWAIT TIME\tTAT");

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

{

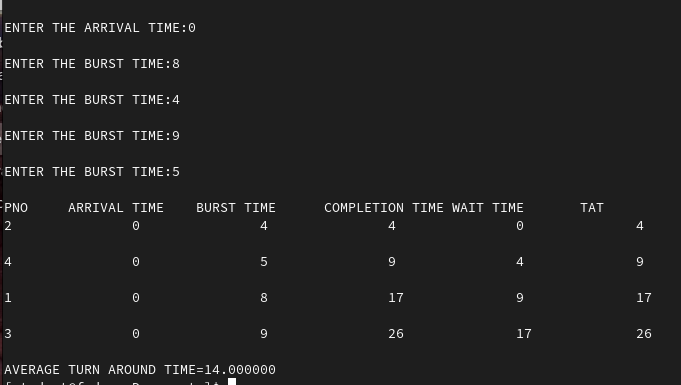
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", pid[i], at[i], bt[i], ft[i], wt[i], ta[i]);

}

printf("\nAVERAGE TURN AROUND TIME=%f", avgta);

printf("\nAVERAGE WAITING TIME=%f", avgwt);

}

**OUTPUT:**

**RESULT:**

Thus the given program was executed and output is verified.

**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("ENTER 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;

}

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;

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;

total = 0;

printf("\nPROCESS\t\tBURST TIME\tWAITING TIME\tTURNAROUND TIME");

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

{

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

total += tat[i];

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

}

avg\_tat = total / n;

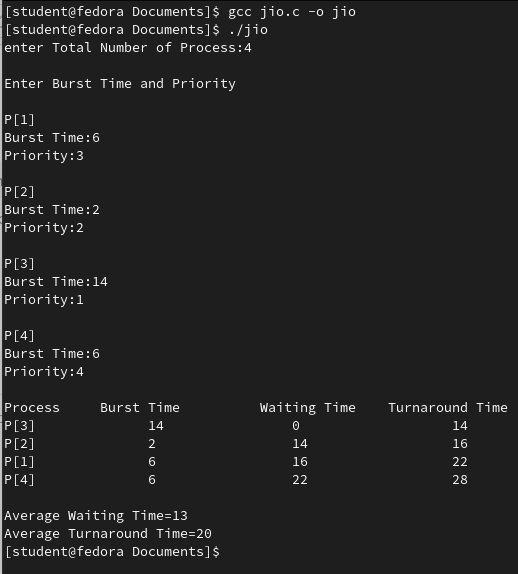
printf("\n\nAVERAGE WAITING TIME= %d", avg\_wt);

printf("\nAVERAGE TURNAROUND TIME= %d\n", avg\_tat);

return 0;

}

**OUTPUT:**



**RESULT:**

Thus the given program was executed and output is verified.

**PROGRAM:**

#include <stdio.h>

int main()

{

int i, limit, total = 0, x, counter = 0, time\_quantum;

int wait\_time = 0, turnaround\_time = 0, arrival\_time[10], burst\_time[10], temp[10];

float average\_wait\_time, average\_turnaround\_time;

printf("\nENTER TOTAL NUMBER OF PROCESSES: ");

scanf("%d", &limit);

x = limit;

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

{

printf("\nENTER DETAILS OF PROCESS[%d]\n", i + 1);

printf("Arrival Time: ");

scanf("%d", &arrival\_time[i]);

printf("Burst Time: ");

scanf("%d", &burst\_time[i]);

temp[i] = burst\_time[i];

}

printf("\nENTER TIME QUANTUM: ");

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

printf("\nPROCESS ID\t\tBURST TIME\tTURNAROUND TIME\tWAITING TIME\n");

for (total = 0, i = 0; x != 0;)

{

if (temp[i] <= time\_quantum && temp[i] > 0)

{

total = total + temp[i];

temp[i] = 0;

counter = 1;

}

else if (temp[i] > 0)

{

temp[i] = temp[i] - time\_quantum;

total = total + time\_quantum;

}

if (temp[i] == 0 && counter == 1)

{

x--;

printf("\nProcess[%d]tt%d\t\t %d\t\t\t %d", i + 1, burst\_time[i], total - arrival\_time[i], total - arrival\_time[i] - burst\_time[i]);

wait\_time = wait\_time + total - arrival\_time[i] - burst\_time[i];

turnaround\_time = turnaround\_time + total - arrival\_time[i];

counter = 0;

}

if (i == limit - 1)

{

i = 0;

}

else if (arrival\_time[i + 1] <= total)

{

i++;

}

else

{

i = 0;

}

}

average\_wait\_time = wait\_time \* 1.0 / limit;

average\_turnaround\_time = turnaround\_time \* 1.0 / limit;

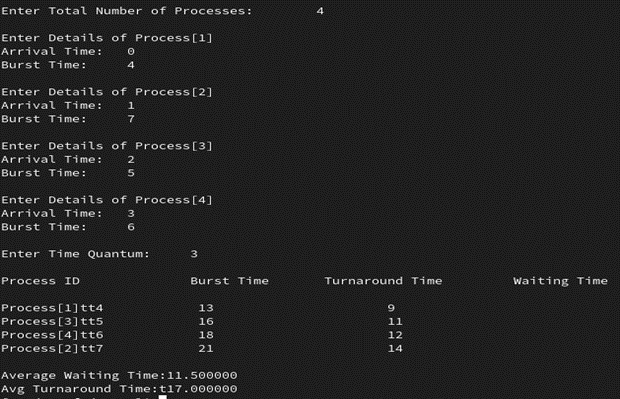
printf("\n\nAverage Waiting Time:%f", average\_wait\_time);

printf("\nAvg Turnaround Time:t%f\n", average\_turnaround\_time);

return 0;

}

**OUTPUT:**



**RESULT:**

Thus the given program was executed and output is verified.

**PROGRAM:**

prodcons.h

#include <stdio.h>

#include <semaphore.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <fcntl.h>

#define BUFFER\_SIZE 10

#define CONSUMER\_SLEEP\_SEC 3

#define PRODUCER\_SLEEP\_SEC 1

#define KEY 1010;

typedef struct

{

int buff[BUFFER\_SIZE];

sem\_t mutex, empty, full;

} MEM;

MEM \*memory()

{

key\_t key = KEY;

int shmid = shmget(key, sizeof(MEM), IPC\_CREAT | 0666);

return (MEM \*)shmat(shmid, NULL, 0);

}

void init()

{

MEM \*m = memory(0);

sem\_init(&M->mutex, 1, 1);

sem\_init(&M->empty, 1, BUFFER\_SIZE);

sem\_init(&M->full, 1, 0);

}

producer.c

#include "prodcons.h"

void producer()

{

int i = 0, n;

MEM \*s = memory();

while (1)

{

i++;

sem\_wait(&s->empty);

sem\_wait(&s->mutex);

sem\_getvalue(&s->full, &n);

(s->buff)[n] = i;

printf("[producer] placed item [%d] \n", i);

sem\_post(&s->mutex);

sem\_post(&s->full);

sleep(PRODUCER\_SLEEP\_SEC);

}

}

void main()

{

init();

producer();

}

consumer.c

#include "prodcons.h"

void consumer()

{

int n;

MEM \*s = memory();

while (1)

{

sem\_wait(&s->full);

sem\_wait(&s->mutex);

sem\_getvalue(&s->full, &n);

printf("[consumer] removed item [%d]\n", (s->buff)[n]);

sem\_post(&s->mutex);

sem\_post(&s->empty);

sleep(CONSUMER\_SLEEP\_SEC);

}

}

void main()

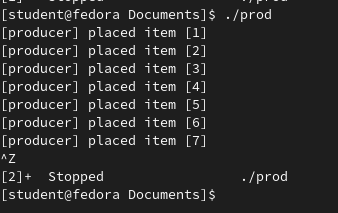
{

consumer();

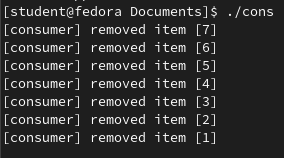
}

**OUTPUT:**

**TERMINAL FOR PRODUCER:**



**TERMINAL FOR CONSUMER:**



**RESULT:**

Thus the given program was executed and output is verified.

**PROGRAM:**

#include <stdio.h>

int main()

{

printf("Enter the number of process: ");

int n, r, k, p = 0, mark, deadlock = 1;

scanf("%d", &n);

k = n, mark = n;

printf("Enter the number of resources: ");

scanf("%d", &r);

int alloc[n][r], max[n][r], avail[r], need[n][r], marked[n];

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

marked[i] = 0;

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

{

printf("Enter the allocated resources for P%d: ", i + 1);

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

{

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

}

}

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

{

printf("Enter the max resources for P%d: ", i + 1);

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

{

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

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

}

}

printf("Enter the available resources: ");

for (int i = 0; i < r; i++)

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

printf("\nALLOC\n");

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

{

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

{

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

}

printf("\n");

}

printf("\nMAX\n");

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

{

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

{

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

}

printf("\n");

}

printf("\nNEED\n");

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

{

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

{

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

}

printf("\n");

}

printf("\nChecking....\n");

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

{

if (marked[i] != 0)

continue;

int flag = 0;

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

{

if (need[i][j] > avail[j])

{

flag = 1;

break;

}

}

if (flag == 0)

{

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

{

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

}

p++;

marked[i] = p;

mark--;

}

if (mark == 0)

break;

if (i == n - 1)

{

i = -1;

k--;

}

if (k == 0)

{

deadlock = 1;

}

}

if (deadlock == 1)

{

printf("!!!DEADLOCK!!!");

return 0;

}

int ans[n];

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

{

ans[marked[i] - 1] = i + 1;

}

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

{

if (i < n - 1)

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

else

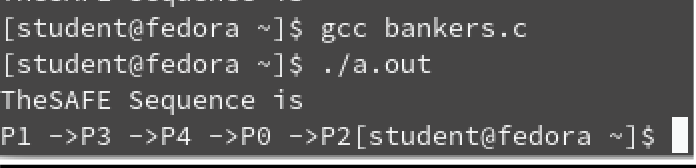
printf("P%d", ans[i]);

}

return 0;

}

**OUTPUT:**



**RESULT:**

Thus the given program was executed and output is verified.

**PROGRAM:**

#include <stdio.h>

#include<string.h>

void bestFit(int blockSize[], int m, int processSize[], int n){

int allocation[n];

memset(allocation, -1, sizeof(allocation));

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

int bestIdx = -1;

for (int j = 0; j < m; j++)

{

if (blockSize[j] >= processSize[i])

{

if (bestIdx == -1)

bestIdx = j;

else if (blockSize[bestIdx] > blockSize[j])

bestIdx = j;

}

}

if (bestIdx != -1)

{

allocation[i] = bestIdx;

blockSize[bestIdx] -= processSize[i];

}

}

printf("\nProcess No. \tProcessSize\tBlock no. \n");

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

{

printf("%d \t\t %d ", i + 1, processSize[i]);

if (allocation[i] != -1)

printf("\t\t%d", allocation[i] + 1);

else

printf("\n Not Allocated");

printf("\n");

}}

int main(){

int blockSize[] = {100, 500, 200, 300, 600};

int processSize[] = {212, 417, 112, 426};

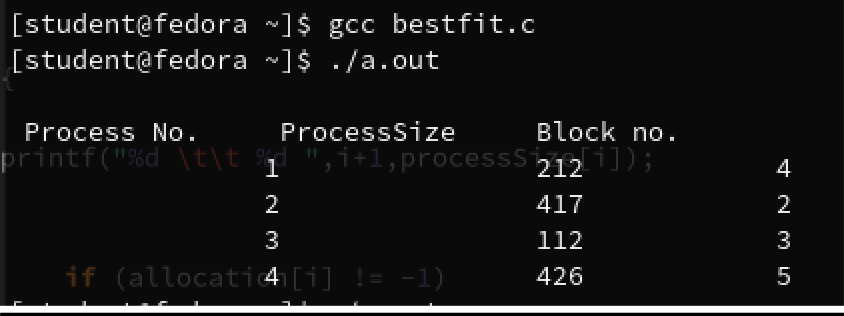
int m = sizeof(blockSize) / sizeof(blockSize[0]);

int n = sizeof(processSize) / sizeof(processSize[0]);

bestFit(blockSize, m, processSize, n);

return 0;

}

**OUTPUT:**

**RESULT:**

Thus the given program was executed and output is verified.

**PROGRAM:**

#include <stdio.h>

#define max 25

void main(){

int frag[max], b[max], f[max], i, j, nb, nf, temp;

static int bf[max], ff[max];

printf("\nEnter the number of blocks:");

scanf("%d", &nb);

printf("Enter the number of files:");

scanf("%d", &nf);

printf("\nEnter the size of the blocks:-\n");

for (i = 1; i <= nb; i++){

printf("Block %d:", i);

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

printf("Enter the size of the files:-\n");

for (i = 1; i <= nf; i++){

printf("File %d:", i);

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

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

{

for (j = 1; j <= nb; j++)

{

if (bf[j] != 1)

{

temp = b[j] - f[i];

if (temp >= 0)

{

ff[i] = j;

break;}}}

frag[i] = temp;

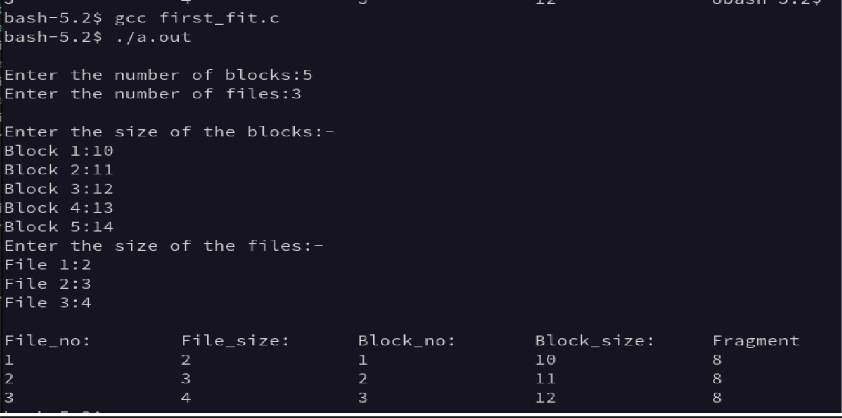
bf[ff[i]] = 1;

}

printf("\nFile\_no:\tFile\_size:\tBlock\_no:\tBlock\_size:\tFragment");

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

printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i, f[i], ff[i], b[ff[i]], frag[i]);}

**OUTPUT:**

**RESULT:**

Thus the given program was executed and output is verified.

**PROGRAM:**

#include <stdio.h>

int main( )

{

int i, j, n, a[50], frame[10], no, k, avail, count = 0;

printf("\n ENTER THE NUMBER OF PAGES:\n");

scanf("%d", &n);

printf("\n ENTER THE PAGE NUMBER :\n");

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

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

printf("\n ENTER THE NUMBER OF FRAMES :");

scanf("%d", &no);

for (i = 0; i < no; i++)

frame[i] = -1;

j = 0;

printf("\tREF STRING\t PAGE FRAMES\n");

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

{

printf("%d\t\t", a[i]);

avail = 0;

for (k = 0; k < no; k++)

if (frame[k] == a[i])

avail = 1;

if (avail == 0)

{

frame[j] = a[i];

j = (j + 1) % no;

count++;

for (k = 0; k < no; k++)

printf("%d\t", frame[k]);

}

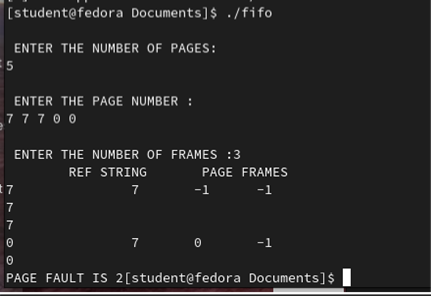
printf("\n");

}

printf("PAGE FAULT IS %d", count);

return 0;

}

**OUTPUT:**

**RESULT:**

Thus the given program was executed and output is verified.

**PROGRAM:**

#include <stdio.h>

int findLRU(int time[], int n)

{

int i, minimum = time[0], pos = 0;

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

{

if (time[i] < minimum)

{

minimum = time[i];

pos = i;

}

}

return pos;

}

int main()

{

int no\_of\_frames, no\_of\_pages, frames[10], pages[30], counter = 0, time[10], flag1, flag2, i, j,

pos, faults = 0;

printf("Enter number of frames: ");

scanf("%d", &no\_of\_frames);

printf("Enter number of pages: ");

scanf("%d", &no\_of\_pages);

printf("Enter reference string: ");

for (i = 0; i < no\_of\_pages; ++i)

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

for (i = 0; i < no\_of\_frames; ++i)

frames[i] = -1;

for (i = 0; i < no\_of\_pages; ++i)

{

flag1 = flag2 = 0;

for (j = 0; j < no\_of\_frames; ++j)

{

if (frames[j] == pages[i])

{

counter++;

time[j] = counter;

flag1 = flag2 = 1;

break;

}

}

if (flag1 == 0)

{

for (j = 0; j < no\_of\_frames; ++j)

{

if (frames[j] == -1)

{

counter++;

faults++;

frames[j] = pages[i];

time[j] = counter;

flag2 = 1;

break;

}

}

}

if (flag2 == 0)

{

pos = findLRU(time, no\_of\_frames);

counter++;

faults++;

frames[pos] = pages[i];

time[pos] = counter;

}

printf("\n");

for (j = 0; j < no\_of\_frames; ++j)

printf("%d\t", frames[j]);

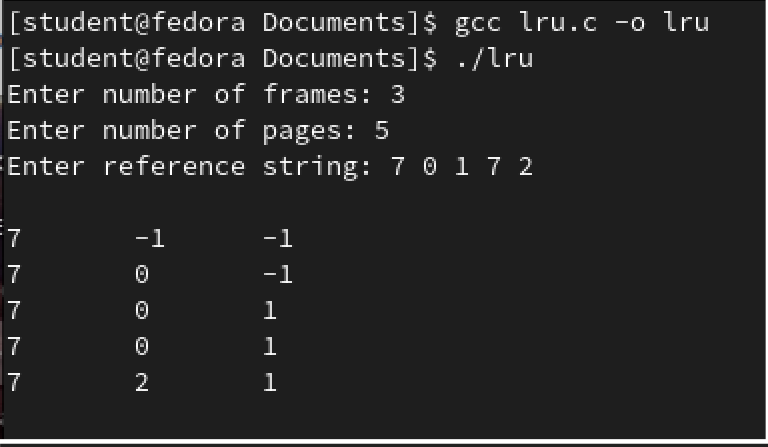
}

printf("\n\nTotal Page Faults = %d", faults);

return 0;

}

**OUTPUT:**



**RESULT:**

Thus the given program was executed and output is verified.

**Customization Steps:**

1. Download the vanilla kernel from www.kernel.org

2. Switch to root user using the command

[root@localhost os]#su

3. Use dnf to install kernel-devel package

[root@localhost os]#dnf install kernel-devel

4. Install gcc development tools

[root@localhost os]#dnf group install "Development Tools"

5. Install additional software packages

[root@localhost os]#dnf install ncurses-devel bison flex elfutils-libelf-devel openssl-devel

6. Copy the downloaded kernel source to /usr/src/kernels

[root@localhost os]#cp linux-5.0.0.tar.xz /usr/src/kernels

7. Go to kernel source directory

[root@localhost os]#cd /usr/src/kernels

8. Extract the downloaded vanilla kernel

[root@localhost os]#unxz linux-5.0.2.tar.xz [root@localhost os]#tar xvf linux-5.0.2.tar

9. Remove all old configuration files

[root@localhost os]#make mrproper

10. Configure the Kernel

[root@localhost os]#make menuconfig

11. Build the Kernel(For faster build use –j 2 option)

[root@localhost os]#make all

12. Remove all temporary files

[root@localhost os]#make clean

13. Install Kernel and its modules

[root@localhost os]#make modules\_install [root@localhost os]#make install

14.Reboot the system