**Lab File**

**Operating System**

**(CSE 202)**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**



Submitted to: Submitted by:

Ms Seema Sharma Shaina Mehta

Ast. Professor A2305219268

CSE Department, ASET B.tech. C.S.E.

4CSE-4Y

Amity School Of Engineering and technology

Amity University Uttar Pradesh

Noida -201301

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Exp No | Assignment  Category | Code | Name of Experiment | Date of Allotment | Date of Evaluation | Max  Marks | Marks  Obtained | Faculty  Sign |
| 1 |  |  | To explore the basic Linux commands. | 17-12-2020 | 24-12-2020 |  |  |  |
| 2 |  | To explore file and directory related commands | 24-12-2020 | 7-01-2021 |  |  |  |
| 3 |  | To explore advance Linux commands. | 7-01-2021 | 14-01-2021 |  |  |  |
| 4 |  | To Explore More Advanced Linux Commands. | 14-01-2021 | 21-01-2021 |  |  |  |
| 5 |  | Shell Scripting | 21-01-2021 | 4-02-2021 |  |  |  |
| 6 |  | Shell Scripting | 4-02-2021 | 11-02-2021 |  |  |  |
| 7 |  | Shell Scripting | 11-02-2021 | 18-02-2021 |  |  |  |
| 8 |  | To simulate FCFS scheduling algorithm using C programming language. | 18-02-2021 | 05-03-2021 |  |  |  |
| 9 |  | To simulate SJF scheduling algorithm using C programming language. | 05-03-2021 | 18-03-2021 |  |  |  |
| 10 |  | To simulate Round Robin scheduling algorithm using C programming language. | 24-03-2021 | 31-03-2021 |  |  |  |
| 11 |  | To simulate Banker’s algorithm using C programming language. | 24-03-2021 | 31-03-2021 |  |  |  |
| 12 |  | To simulate FIFO disk scheduling algorithm using C programming language. | 24-03-2021 | 31-03-2021 |  |  |  |
|  | Viva | Viva |  |  |  |  |  |  |

**Experiment 1**

**Date:** 17-12-2020

**Aim:** To explore the basic Linux commands.

**Software Used:** Cgywin64 Terminal.

**Theory:**

1. **man**: man command provides the user with manual of other commands, type man with the name of the command.

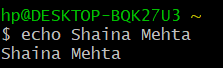
**Syntax:** man <command>





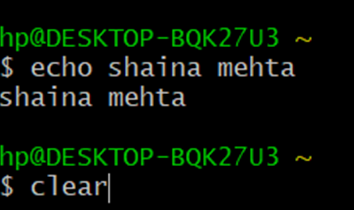
1. **echo:** echo command is used to display the line of text/string that are passed as an argument.

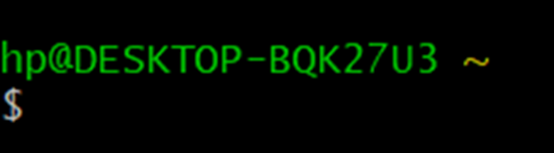
**Syntax:** echo <string to be displayed>



1. **clear:** clear command is used to clear the screen.

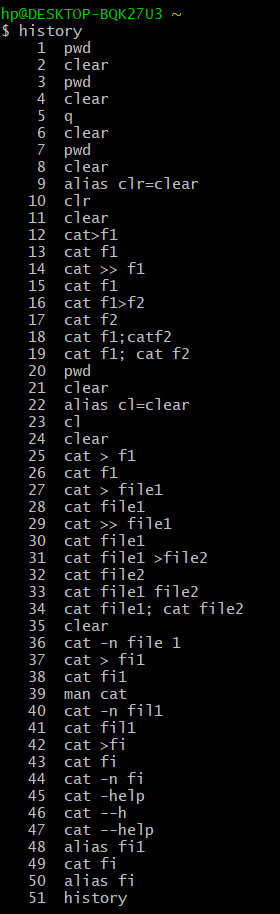
**Syntax:** clear





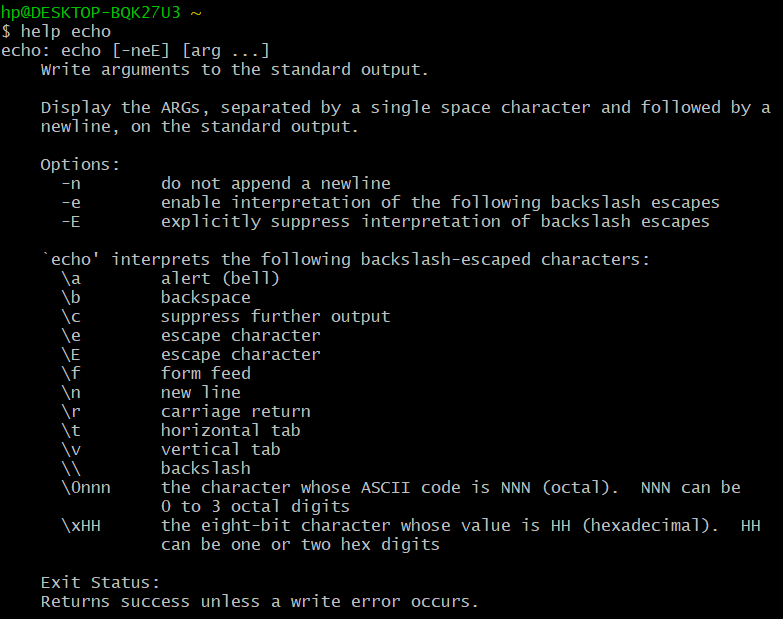
1. **history:** history command is used to view the commands one have entered before.

**Syntax:** history

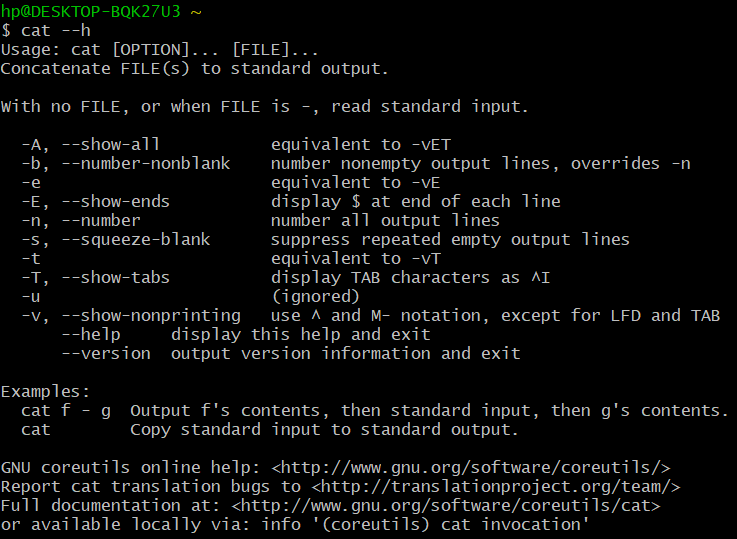


1. **help:** help command is used to display the information about shell build in commands.

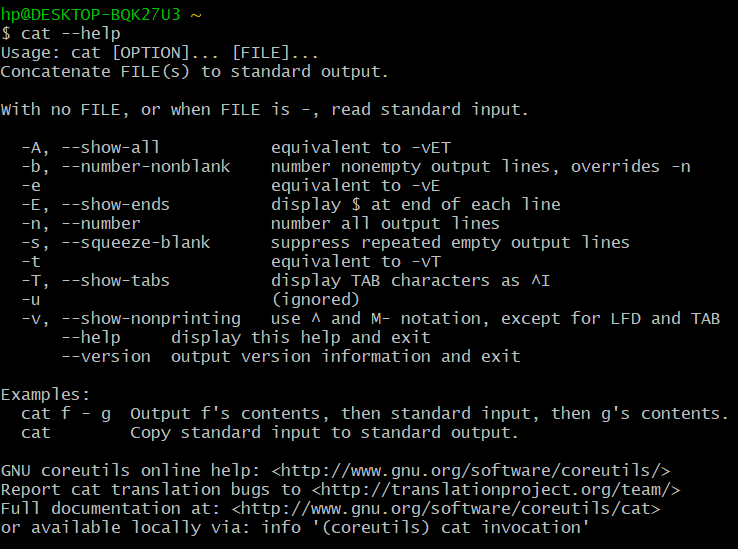
**Syntax:** help <command> (used for echo command only) or <command> -- h (for all the commands except echo command) or <command> -- help (for all the commands except echo command)



Or

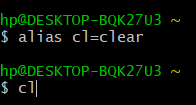


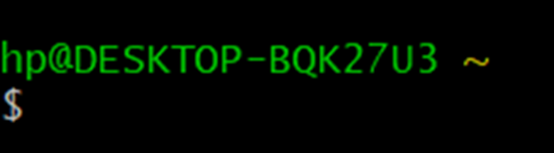
Or



1. **alias:** alias command is used to create custom shortcuts to represent commands.

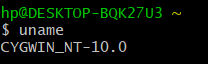
**Syntax:** alias <alias name> = <command>



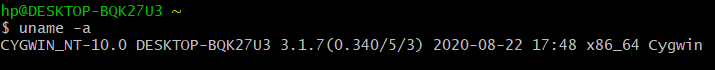


1. **uname:** uname command is used to print the basic information about your operating system (basically of Linux system) like machine name operating system kernel etcetera.

**Syntax:** uname <options> or uname



Or

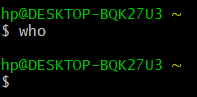


**Note:** Options used in uname command are:

* **-a, --all:** print all the system information, in the manner given above, except omit -p and -i if unknown.
* **-s, --kernel-name:** print the kernel name.
* **-n, --nodename:** print the network node hostname.
* **-r, --kernel-release:** print the kernel release.
* **-v, --kernel-version:** print the kernel version.
* **-m, --machine:** print the machine hardware name.
* **-p, --processor:** print the processor type (non-portable).
* **-i, --hardware-platform:** print the hardware platform (non-portable).
* **-o, --operating system:** print the operating system.
* **--help:** display this help and exit.
* **-version:** output version information and exit.

1. **who:** who command is used to give information of the currently logged in user on to the system. It displays login name of the users, terminal number and login time of the users.

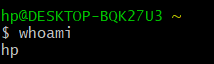
**Syntax:** who



**Note:** The information of the currently logged in user will not be display if we use emulator.

1. **whoami:** whoami command displays the username of the current user when this command is invoked. It is equivalent to id-un command.

**Syntax:** whoami



1. **pwd:** pwd command is used to display the current working directory.

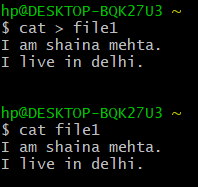
**Syntax:** pwd

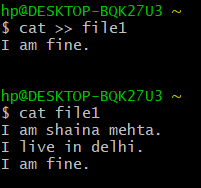


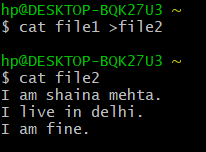
1. **cat:** cat command is used to create single or multiple files, view content of file/s, concatenate files and redirect output in terminal or files.

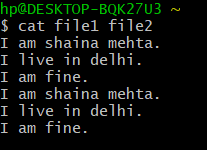
**Syntax:**

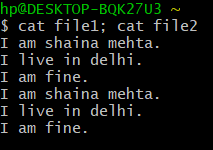
* + - cat > filename - to create a new file
    - cat filename - to open a file
    - cat >> filename - to append the content of a file
    - cat file1>file2 - to copy content of file 1 into file 2
    - cat file1; cat file2 – to open two files simultaneously which can be achieved by using semicolon which is used to perform multiple operations at the same time.
    - Cat file1 file2 – to open two files simultaneously.

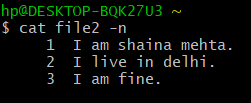


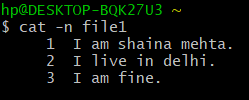


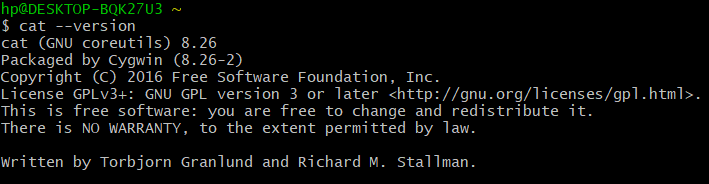












**Result:** Basic Linux commands has been executed successfully.

**Experiment 2**

**Date:** 24-12-2020

**Aim:** To explore file and directory related commands.

**Software Used:** Cgywin64 Terminal.

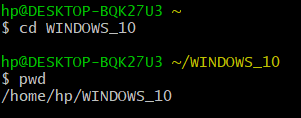
**Theory:**

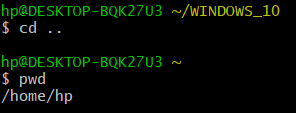
1. **cd:** cd command is known as change directory command. It is used to change the current working directory.

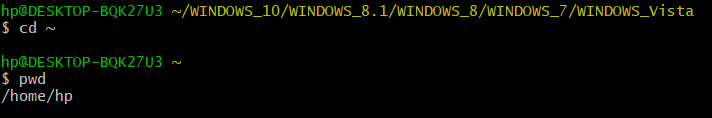
**Syntax:** cd <name of the file/ commands/options>

**Note:** Some symbols used in the form of command/options are:

* **~ :** Specifies the location of your home directory.
* **.. :** Specifies the location of the parent directory.





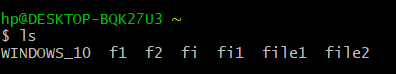


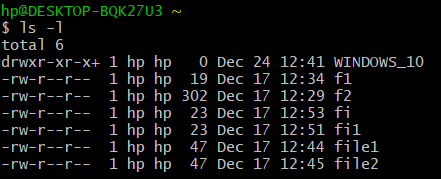
1. **ls:** ls command is used to list the files in the current directory use.

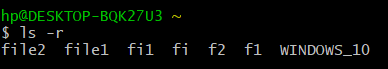
**Syntax**: ls <options> or ls

**Note:**

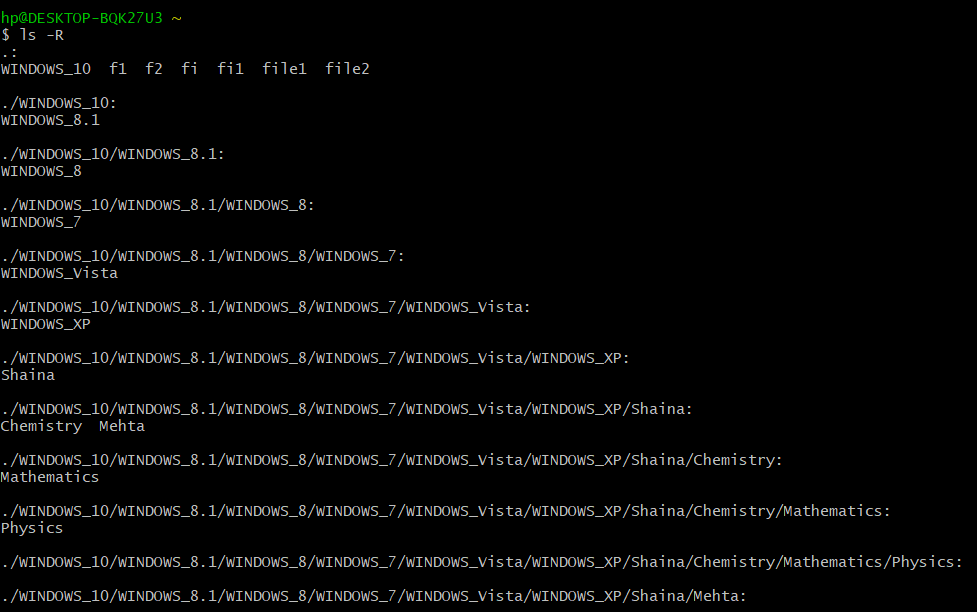
* + - Options used in ls command are:
      * **-l:** uses a long list format.
      * **-t:** sort by modification time, newest first.
      * **-r, --reverse:** reverse the order while sorting.
      * **-R, --recursive:** list subdirectories recursively.
      * **-i, --inode:** print the index number of each file.
      * **\* :** can be used as a wildcard in UNIX/LINUX.
    - Options can be combined: ls -ltr.
    - **For Example:**
      * **ls- lt:** list the files in time in reverse order with long.

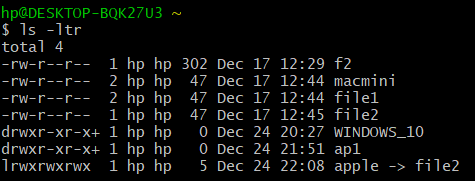


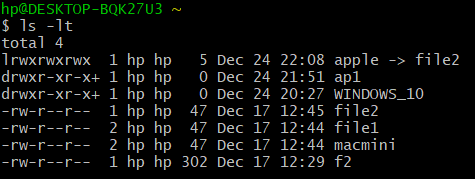


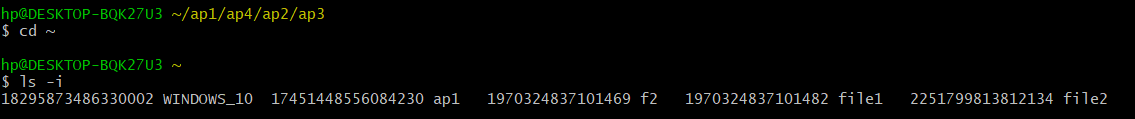












1. **mkdir:** mkdir command is used to create a new directory.

**Syntax:** mkdir <option> <directory> or mkdir <directory>

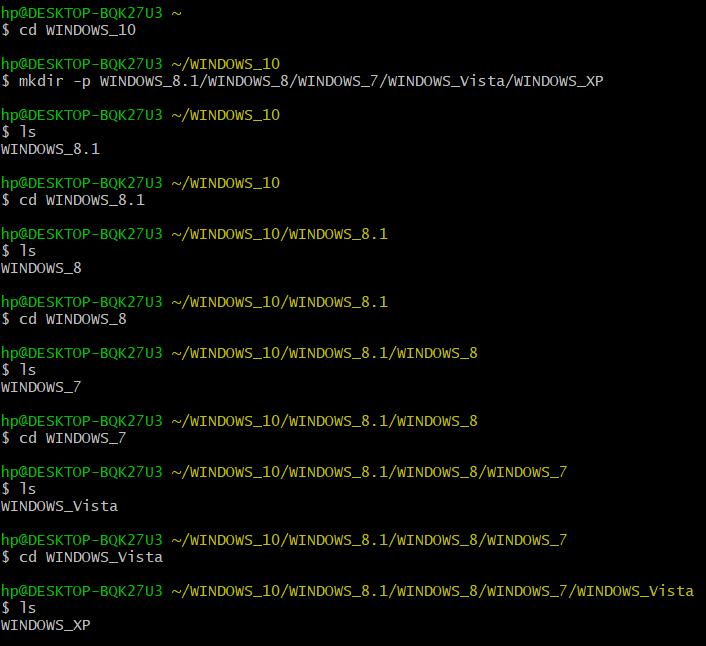
**Note:**

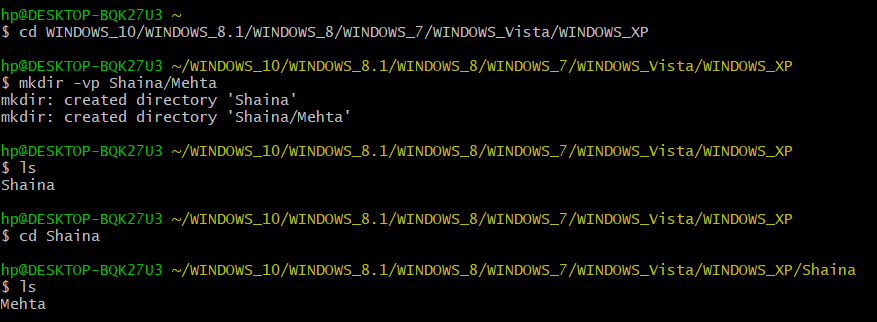
* The command takes more than one directory name as its arguments.
* Options used in mkdir command are:
  + - * + **-m, --mode:** to set a file mode.
        + **-p, -parents:** no error if existing, otherwise make parent directory as needed.
        + **-v, --verbose:** print the message for each created directory.
        + **-z:** set SELinux security context for each created directory to the default type.
        + **context [=CTX]:** like -z, or if CTX is specified then set the SELinux or

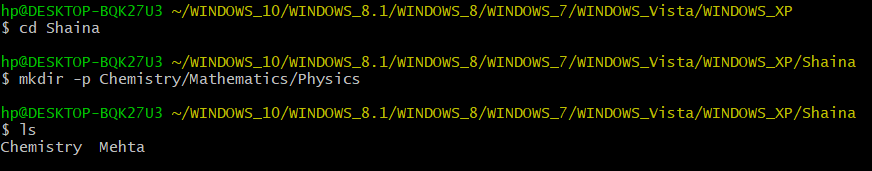
SMACK security to CTX.

* + - * + **--help:** display the help and exit.
        + **--version:** output version information and exit.







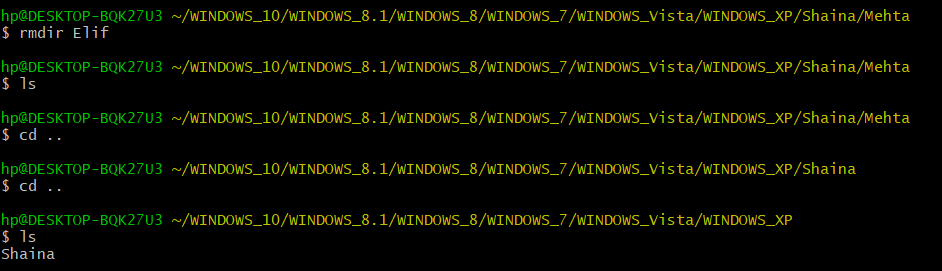


1. **rmdir:** rmdir command is used to remove empty directories.

**Syntax:** rmdir <option> <directory> or rmdir <option>

**Note:** Options used in rmdir are:

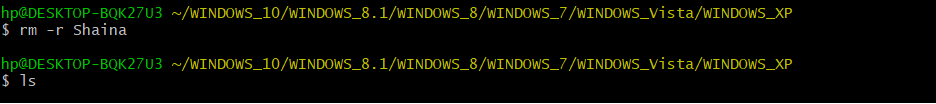
* **--ignore-fail-on-non-empty:** ignore each failure that is solely because a directory is non - empty.
* **-p, --parents:** remove directory and its ancestors. For example:  
  ‘rmdir -p a/b/c’ is similar to ‘a/b/c a/b a’.
* **-v, -verbose:** outputs a diagnostic for every directory processed
* **--help:** displays the help and exit.
* **--version:** outputs the version information and exit.

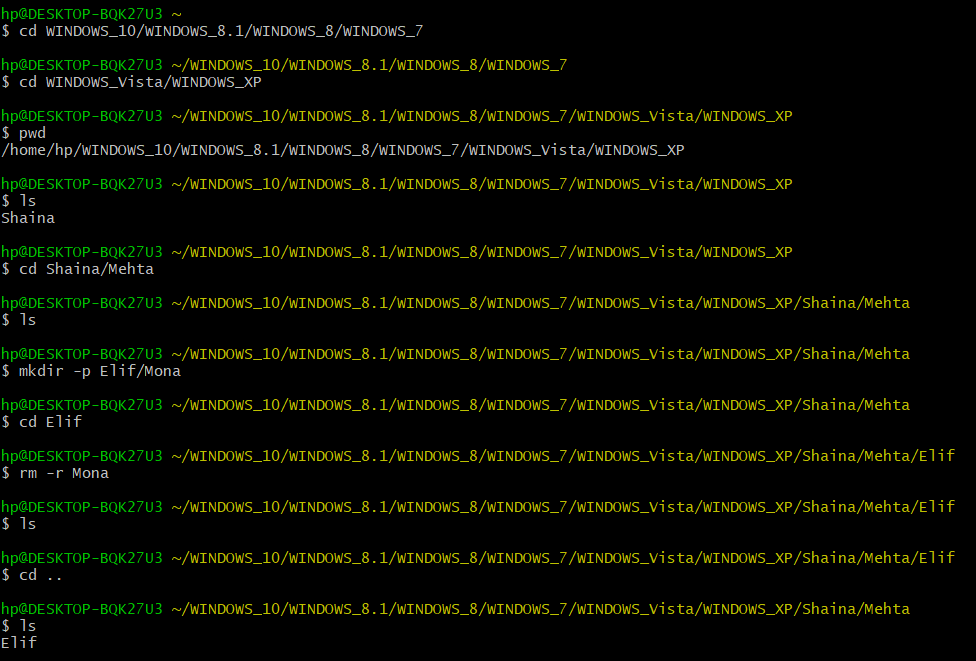


1. **rm:** rm command is used to remove a file.

**Syntax:** rm <directory> or rm <options> <directory>

**Note:** Options used in rm command are:

* **-f, --force:** ignore non existing files and arguments and never prompt.
* **-i:** prompt before every removal.
* **-I:** prompt once before removing more than one files, or when removing recursively; less intuitive than -i, while still giving protection against more mistakes.
* **--interactive [=WHEN]:** prompt according to WHEN: never, once (-I), or always (-i); without WHEN, prompt always.
* **--one – file - system:** when removing the hierarchy recursively, skip any directory that is on the file system different from that corresponding command line argument.
* **--no - preserve – root:** do not treat ‘/’ specially.
* **--preserve – root:** do not remove ‘/’ (default).
* **-r, -R, --recursive:** remove directories and their contents recursively.
* **-d, --dir:** remove empty directories.
* **-v, -verbose:** explain what is being done.
* **--help:** display this help and exit.
* **--version:** output version information and exit.



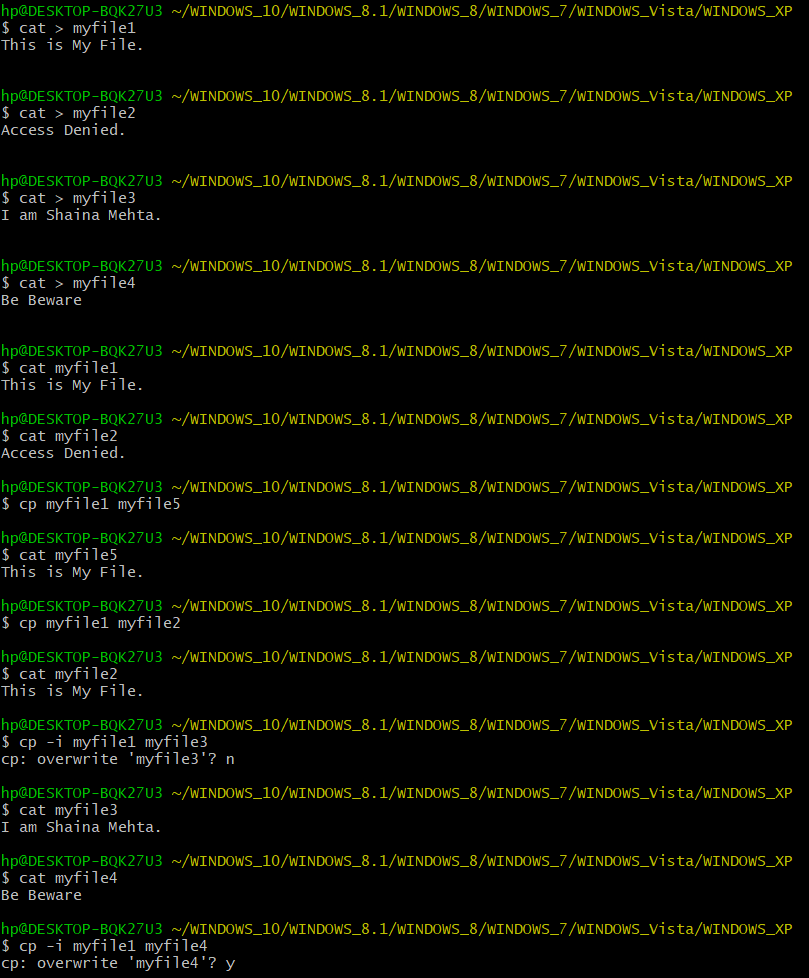
1. **cp:** cp command is used to copy the files or group of files or directories.

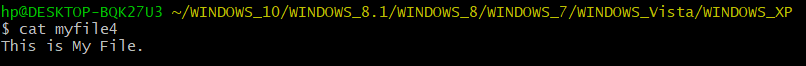
**Syntax:**

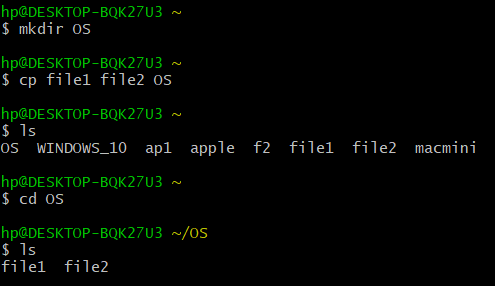
* cp <source file> <destination file>:copy the contents of one file to another.
* cp <file1> <file2> <directory name>: copy multiple files in a directory.
* cp -i <source file> <destination file>: asks the user whether to copy the source file to the destination file or not.

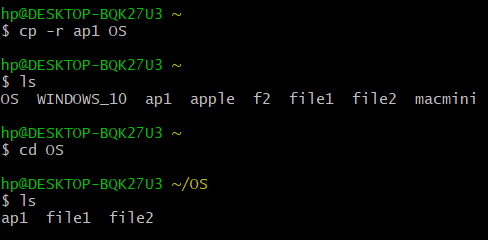
**Note:**

* By default, the cp command will not copy directories. Attempting to copy a directory results in an error.
* To copy a directory, pass the -R or -r or –recursive flag. This will recursively copy a folder and create a copy.
* **Syntax:** Cp –r <source directory> <destination directory>





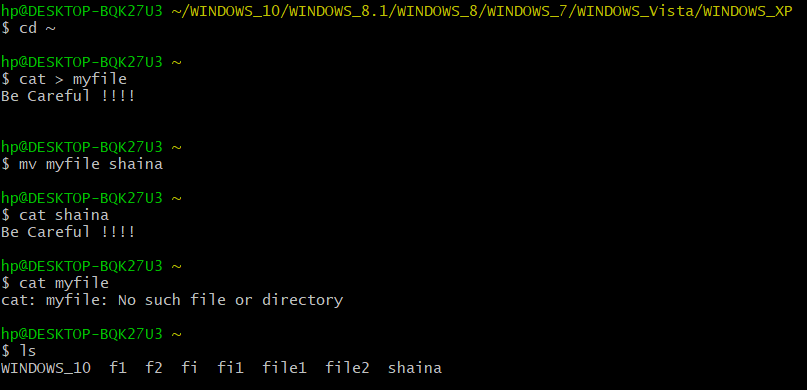


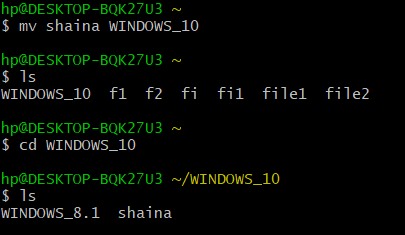


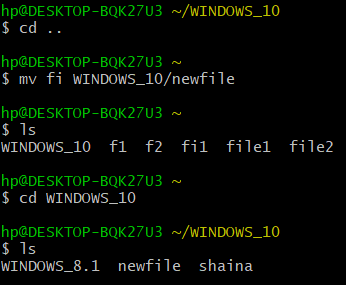
1. **mv:** mv command is used to move files or directories from one place to another.

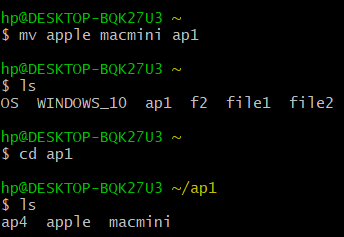
**Syntax:**

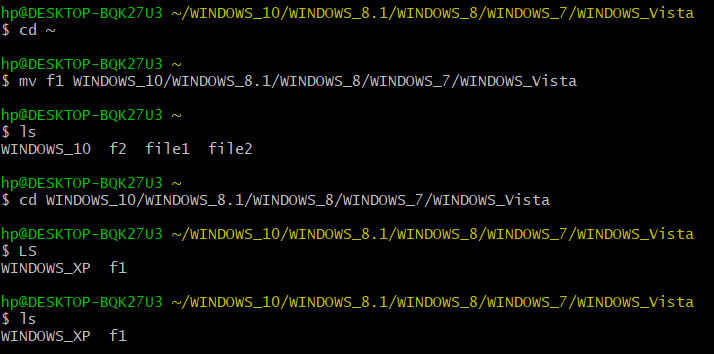
* mv <old file> <new file>:
  + - * + To move a file using the mv command pass the name of the file and then the new name for the file
        + **For Example:** mv file1 file2
        + In above example file1 is renamed to file2.
* mv <old directory> <new directory>: to move a directory.
* mv <file name> <directory name> or mv <file name> <directory name/ filename>: to move a file in a directory.
* mv <file1> <file2> <file3> <directory name>: to move multiple files in a given directory.
* mv -I file1 file2: prompt before overwriting a file.

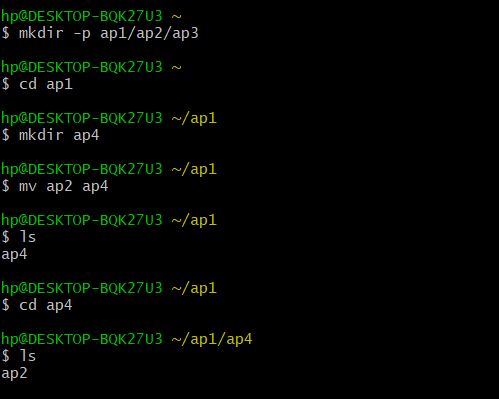


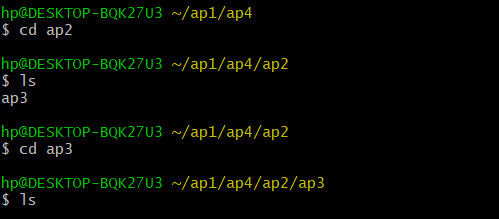












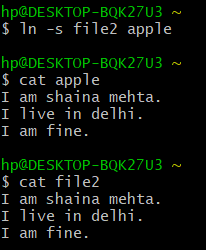
1. **ln:** ln command is used to create links between files.

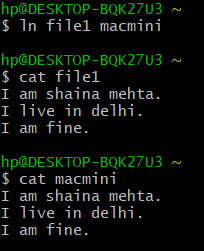
* **Soft Link:** ln –s filename soft\_link name. -s makes symbolic links instead of hard link.

**Syntax:** ln -s <file1> <file2>

* **Hard link:** ln filename hard\_link name.

**Syntax:** ln <file1> <file2>





**Result:** Various file and directory related commands has been explored and executed successfully.

**Experiment 3**

**Date:** 7 - 01 - 2021

**Aim:** To explore advance Linux commands.

**Software Used:** Cgywin64 Terminal.

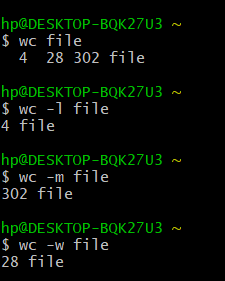
**Theory:**

1. **wc:** wc command is used for printing newline, word and byte counts for files. It can return the number of lines in a file, the number of characters in a file and the number of words in a file. The output is number of lines, number of words, number of bytes, filename.

**Syntax:** wc <filename> or wc <options> <filename>.

**Note:** Options for wc command are:

* -l - To print the number of lines in a file.
* -m - To print the number of characters in a file.
* -w - To print the number of words in a file.



1. **cmp:** cmp command is used to compare the two files byte by byte. It helps you to find out whether the two files are identical or not. It reports the location of the first mismatch to the screen if difference is found and if no difference is found i.e the files compared are identical. It displays no message and simply returns the prompt if the files compared are identical.

**Syntax:** cmp <option> <filename> or cmp <filename>.

**Note:**

* -b - display the differing bytes in its output.

cmp -b file1 file2

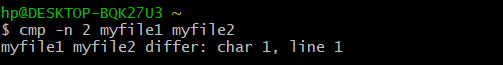
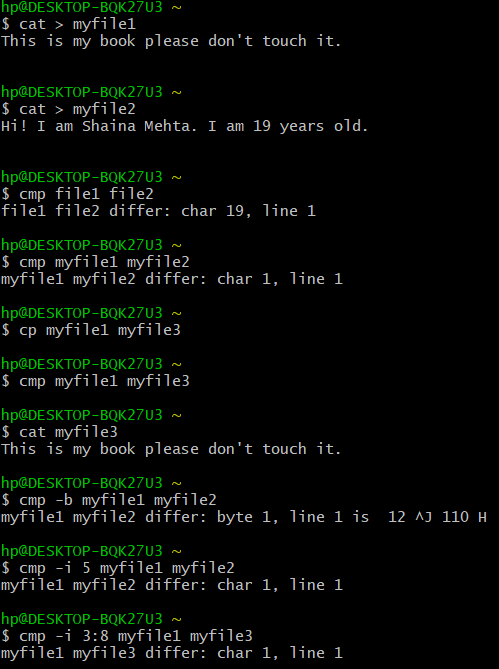
* -i [bytes-to-be-skipped] - Now, this option when used with cmp command helps to skip a particular number of initial bytes from both the files and then after skipping it compares the files.

cmp –i 5 file1 file2

* -i [bytes to be skipped from first file] : [bytes to be skipped from second file] - This option is very much similar to the above -i [bytes to be skipped] option but with the difference that now it allows us to input the number of bytes we want to skip from both the files separately.

cmp –i 4:4 file1 file2

* -n [number of bytes to be compared] option - This option allows you to limit the number of bytes you want to compare, like if there is only need to compare at most 25 or 50 bytes.



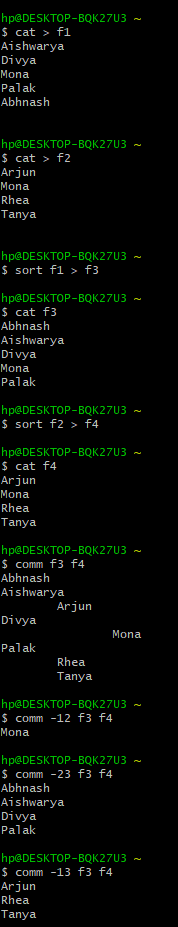
1. **comm:**

* It requires two sorted files and lists the differing entries in different columns.
* When you run comm, it displays a three – columnar output.
* The first column contains the lines unique to the first file, and the second column shows the lines unique to the second file. The third column displays lines to both files.

**Syntax:** comm <options> <sorted\_file\_1> <sorted\_file\_2> or comm <sorted\_file\_1> <sorted\_file\_2>.

**Note:**

* These commands require single column output from comm, and comm can produce using the options -1, -2 or -3.
* To drop a particular column simply use its column number as an option prefix.



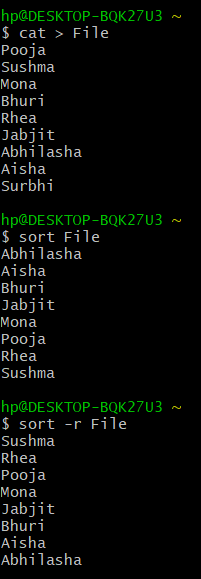
1. **sort:**

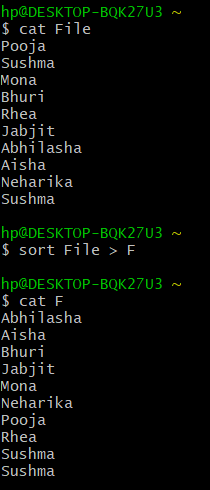
* sort lines alphabetically by default.
* Running sort filename writes the contents of the filename in alphabetical order to standard output.

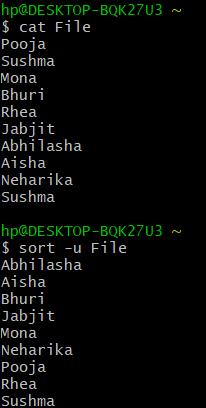
**Syntax:** sort <options> <filename> or sort <filename> or sort <file1> <file2>

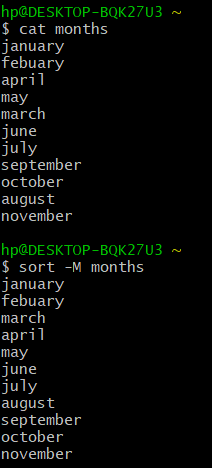
**Note:**

* + -r - sort in reverse order and write the result to standard output.
  + -n - This will sort from lowest number to highest number and write the result to standard output.
  + To sort and remove duplicates pass the -u option to sort. This will write a sorted list to standard output and remove duplicates.
  + To sort by month pass the -M option to sort.









1. **Creating Files in Linux:** It requires the use of an Editor.Various editors are used for this purpose which are:

* nano / pico
* vi
* emacs

**Vi Editor:**

* The VI editor is the most popular and classic text editor in the Linux family.
* Below, are some reasons which make it a widely used editor:
  + - * + available in almost all Linux Distributions
        + works the same across different platforms and Distributions
        + user-friendly

**Modes of Vi Editor:**

* + - * Command Mode
      * Insert Mode
      * Escape Mode

**Command Mode:**

* vi starts in Command Mode.
* vi interprets any characters we type as commands and does not display them in the window.
* This mode allows us to move through a file, and to delete, copy, or paste a piece of text.
* To enter into Command Mode from any other mode, it requires pressing the [Esc] key. If we press [Esc] when we are already in Command Mode, then vi will beep or flash the screen.

**Insert Mode:**

* Enables you to insert text into the file.
* Everything that’s typed in this mode is interpreted as input and finally, it is put in the file.
* The vi always starts in command mode. To enter text, you must be in insert mode. To come in insert mode, you simply type i. To get out of insert mode, press the Esc key, which will put you back into command mode.

**Escape Mode:**

* enables you to perform tasks such as saving files, executing commands.
* invoked by typing a colon [:], while vi is in Command Mode.
* The cursor will jump to the last line of the screen and vi will wait for a command.

**Vi Editor Commands:**

* i - Insert at cursor (goes into insert mode)
* a - Write after cursor (goes into insert mode)
* A - Write at the end of line (goes into insert mode)
* ESC - Terminate insert mode
* U - Undo all changes to the entire line
* - Open a new line (goes into insert mode)
* dd - Delete line
* 3dd - Delete 3 lines.
* D - Delete contents of line after the cursor
* dw - Delete word
* 4dw - Delete 4 words
* cw - Change word
* x - Delete character at the cursor
* r - Replace character
* R - Overwrite characters from cursor onward
* s - Substitute one character under cursor continue to insert
* S - Substitute entire line and begin to insert at the beginning of the line
* k - Move cursor up
* j - Move cursor down
* h - Move cursor left
* l - Move cursor right
* :w - Save the file but keep it open
* :q - Quit without saving
* :wq - Save the file and quit

**Note:**

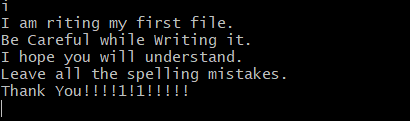
* Make sure you press the right command otherwise you will end up making undesirable changes to the file.
* You can also enter the insert mode by pressing a, A, o,

Opening of Vi Editor

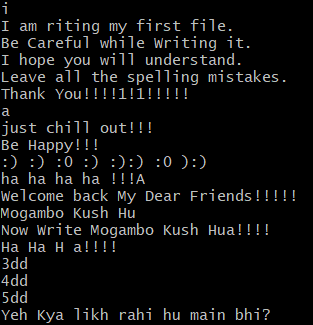




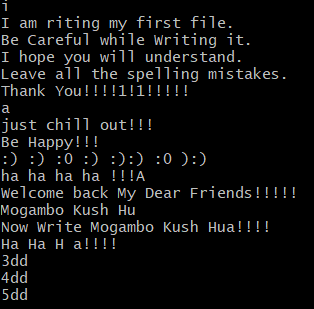
Insert at cursor (goes into insert mode)



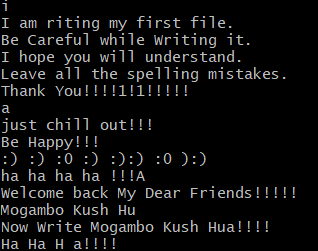
Write after cursor (goes into insert mode) and write at the end of line (goes into insert mode)



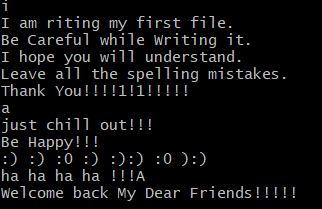
Delete line

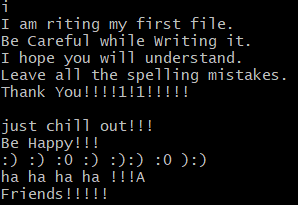


Delete 3 lines.

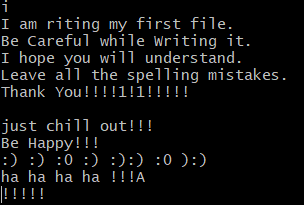


Delete 4 words

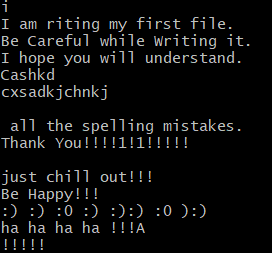


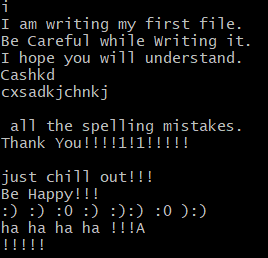


Delete word

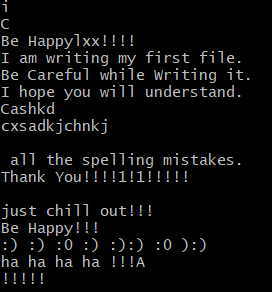


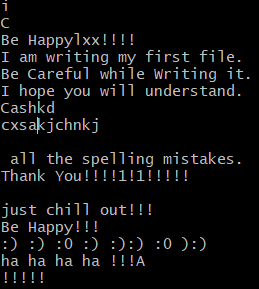
Change word



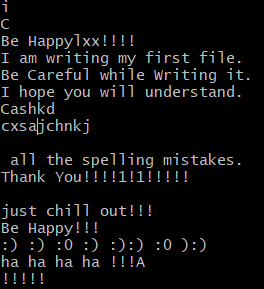


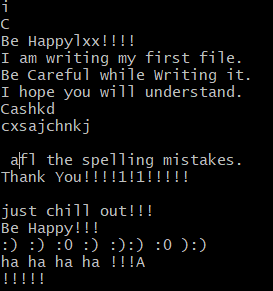
Delete character at the cursor



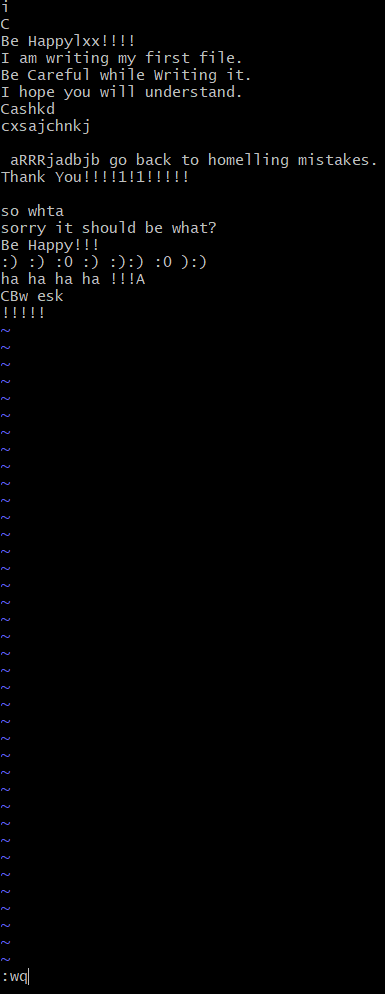


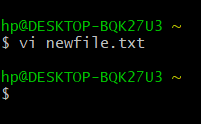
Replace character





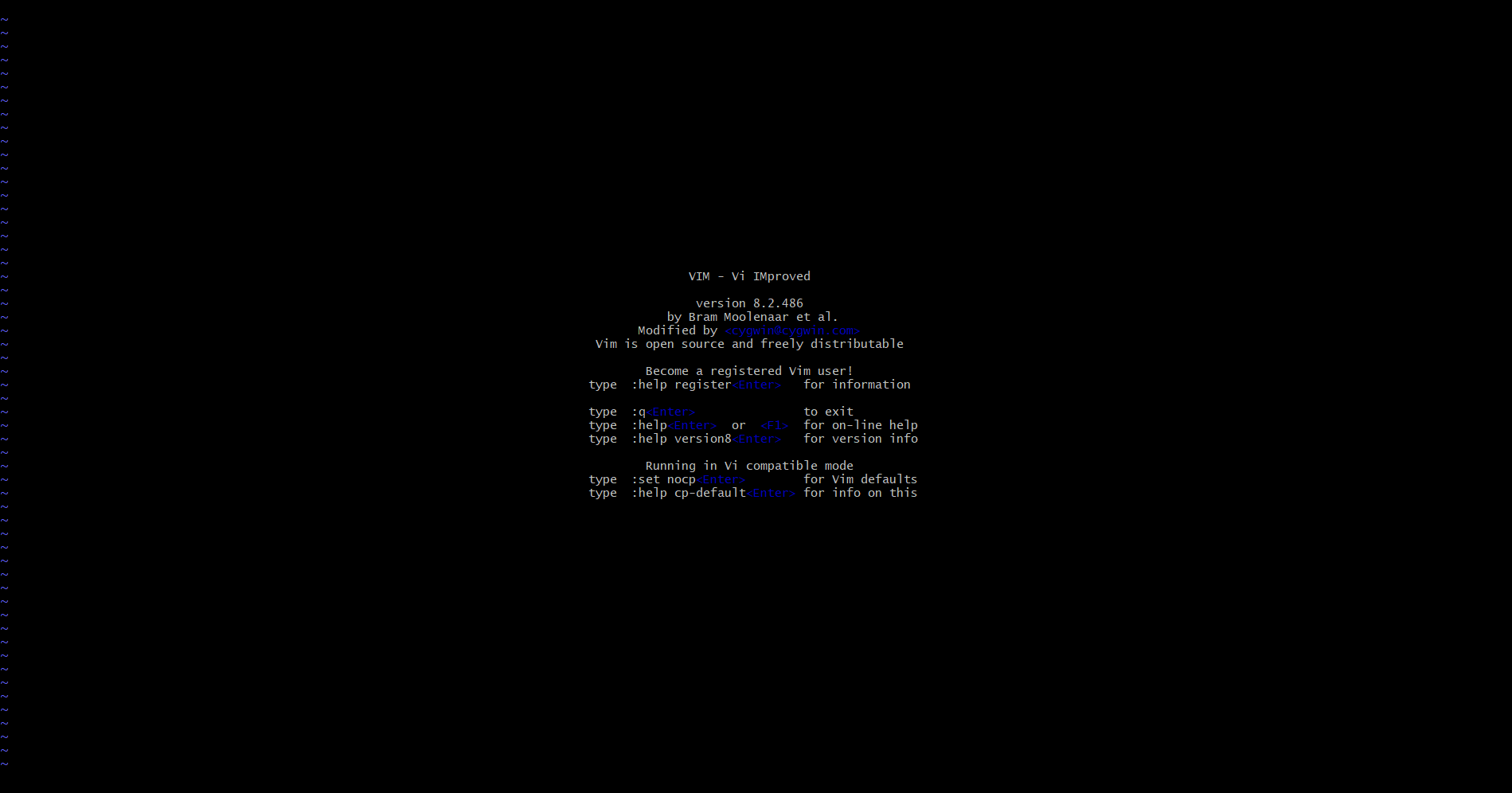
Save the file and quit





Type “vi” at the prompt





**Results:** Linux commands has been executed successfully.

**Experiment 4**

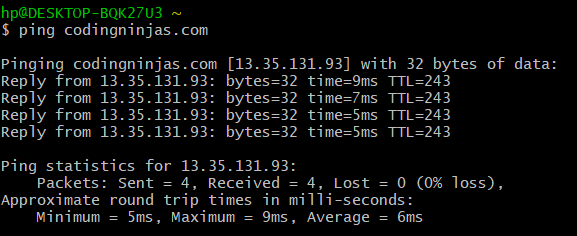
**Date:** 14 – 01 - 2021

**Aim:** To Explore More Advanced Linux Commands.

**Theory:**

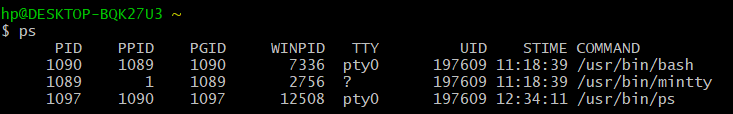
1. **ping:** The ping command lets you verify that you have network connectivity with another network device. It is commonly used to help troubleshoot networking issues. To use ping, provide the IP address or machine name of the other device.

**Syntax:** ping



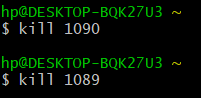
1. **ps:** The ps command lists running processes. Using ps without any options causes it to list the processes running in the current shell.

**Syntax:** ps



1. **kill:**

* To terminate a process use “kill”
* Rules are simple:
  + - * + You can kill all your own process.
        + Only root user can kill system level process.
        + Only root user can kill process started by other users.
* used to terminate processes manually. *kill* command sends a signal to a process which terminates the process. If the user doesn’t specify any signal which is to be sent along with kill command then default *TERM*signal is sent that terminates the process.
* **Syntax:** kill <pid>

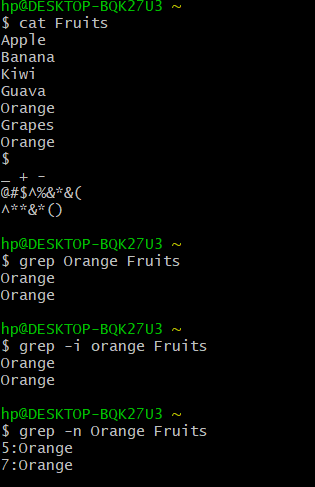


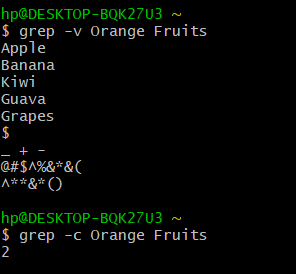
1. **grep:** It is used to grep searches the named input files for lines containing a match to the given pattern.

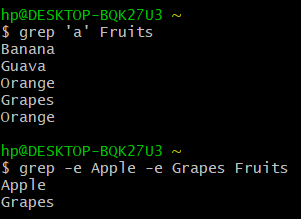
**Syntax:** grep <options> <filename>

**Note:** Options used in grep command are:

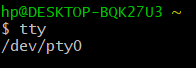
* + - -e: pattern
    - -i: Ignore uppercase vs. lowercase.
    - -v: Invert match.
    - -c: Output count of matching lines only.
    - -l: Output matching files only.
    - -n: Precede each matching line with a line number.
    - -b: A historical curiosity: precede each matching line with a block number.
    - -h: Output matching lines without preceding them by file names.
    - -s: Suppress error messages about nonexistent or unreadable files.
    - -x
    - -f file: Take regexes from a file.
    - -o: Output the matched parts of a matching line.





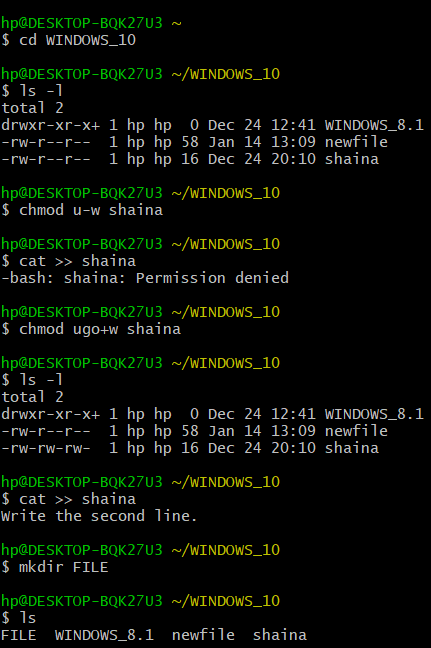


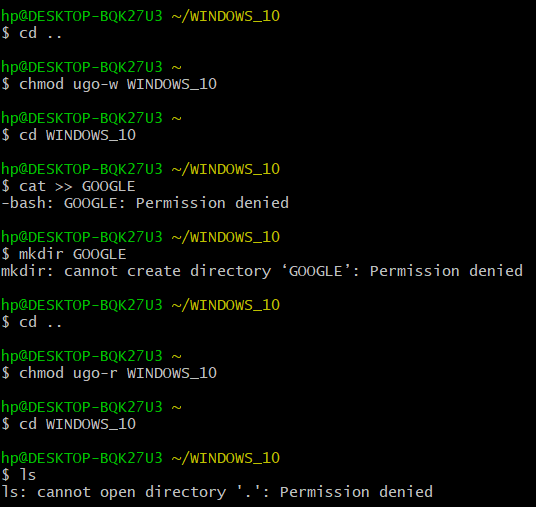
1. **tty:**
   * + tty is a command in Unix and Unix-like operating systems to print the file name of the terminal connected to standard input. tty stands for TeleTYpewriter.
     + The tty command basically prints the file name of the terminal connected to standard input.
     + **Syntax:** tty

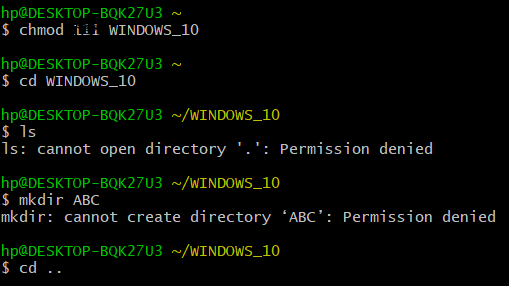


1. **chmod:**

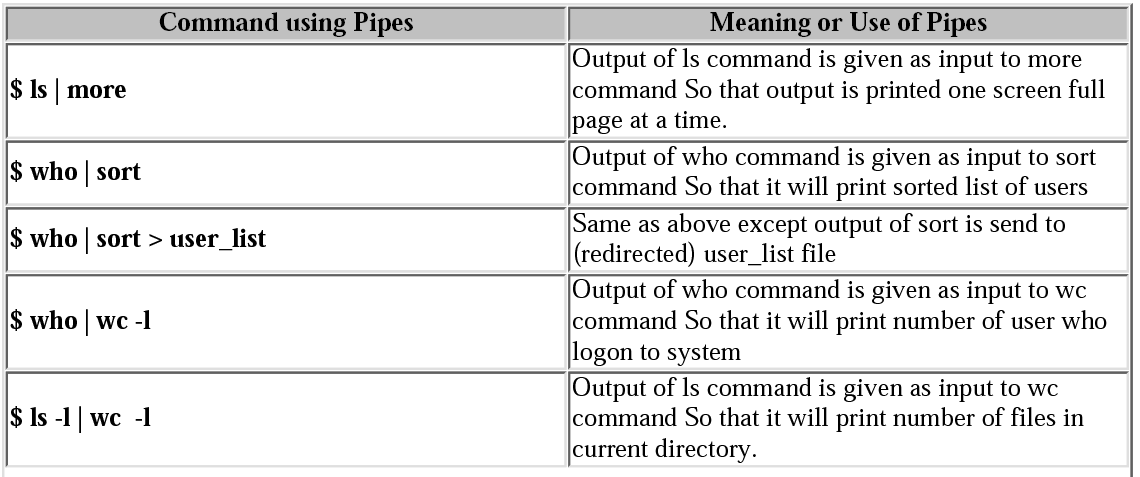
* There are three types of permissions: read (r), write (w), and execute (x).
* To read a file is to view its contents. For example, a text file must have read permission for someone to read the text within.
* If the user wants to add a sentence to that file, it needs write permission.
* The execute permission enables someone to run a file, such as a shell script or a binary program file.
* The ls -l command displays the permissions assigned to a file.
* *user*, *group*, and *other*.
* Each file is associated with an owner and a group and assigned with permission access rights for three different classes of users:
* The file owner.
* The group members.
* Others (everybody else)
* **Operation:**
  + - * - Removes the specified permissions.
      * + Adds specified permissions.
      * = Changes the current permissions to the specified permissions. If no permissions are specified after the = symbol, all permissions from the specified user class are removed.
* Directories are special types of files that contain other files and directories.
* The chmod command allows you to change the permissions on a file using either a symbolic or numeric mode or a reference file.
* **Symbolic (Text) Method:**
  + - * **Syntax:** chmod <options> <ugoa…><-+=perm s> <file>
      * The permissions (perms...) can be explicitly set using either zero or one or more of the following letters: r, w, x. Use a single letter from the set u, g, and o when copying permissions from one to another user class.
* **Numeric Method:**
  + - * The syntax of the chmod command when using numeric method has the following format.
      * **Syntax:** chmod <options> <Number File...>
      * When using the numeric mode, you can set the permissions for all three user classes (owner, group, and all others) at the same time. the first digit represents the permissions of the file’s owner, the second one the file’s group, and the last one all other users.
      * Each write, read, and execute permissions have the following number value:
        + r (read) = 4
        + w (write) = 2
        + x (execute) = 1
        + no permissions = 0
      * The permissions number of a specific user class is represented by the sum of the values of the permissions for that group.
      * calculate the totals for all users classes. For example, to give read, write and execute permission to the file’s owner, read and execute permissions to the file’s group and only read permissions to all other users you would do the following:
        + Owner: rwx=4+2+1=7
        + Group: r-x=4+0+1=5
        + Others: r-x=4+0+0=4
      * Using the method above we come up to the number 754, which represents the desired permissions.

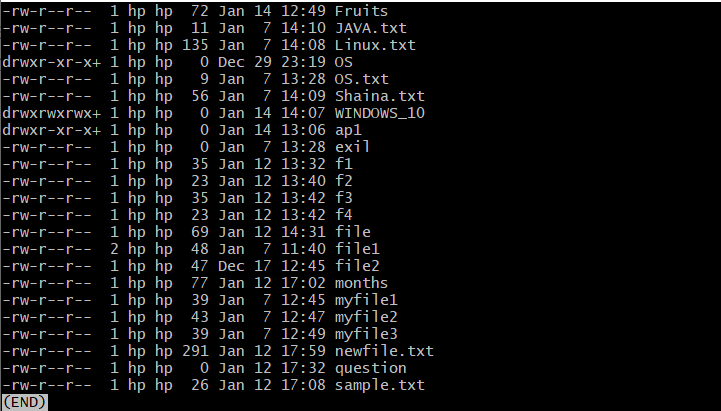
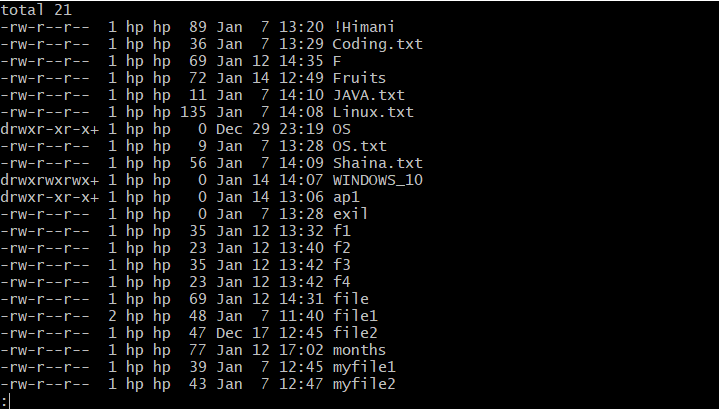
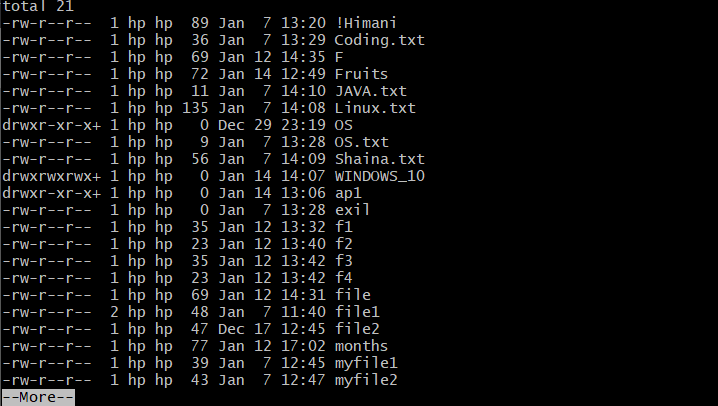






1. **Pipe ( | ):**
   * + A pipe is a form of redirection (transfer of standard output to some other destination) that is used in Linux and other Unix-like operating systems to send the output of one command/program/process to another command/program/process for further processing.
     + You can make it do so by using the pipe character ‘|’.
     + It can also be visualized as a temporary connection between two or more commands/ programs/ processes.
     + Pipes are unidirectional i.e. data flows from left to right through the pipeline.
     + The pipe acts as a container which takes the output of ls -l and gives it to more as input. This command does not use a disk to connect standard output of ls -l to the standard input of more because pipe is implemented in the main memory.
     + **Syntax:** command 1| command 2| command 3|……. | command n





**Note:**

1. **more:** more command is used to view the text files in the command prompt, displaying one screen at a time in case the file is large (For example log files). The more command also allows the user do scroll up and down through the page. The syntax along with options and command is as follows. Another application of more is to use it with some other command after a pipe. When the output is large, we can use more command to see output one by one.

**Syntax:** more <-options> <-num] ><+/pattern> <+linenum> <file\_name>

* + - **[-options]:** any option that you want to use in order to change the way the file is displayed. Choose any one from the followings: (-d, -l, -f, -p, -c, -s, -u)
    - **[-num]:** type the number of lines that you want to display per screen.
    - **[+/pattern]:** replace the pattern with any string that you want to find in the text file.
    - **[+linenum]:** use the line number from where you want to start displaying the text content.
    - **[file\_name]:** name of the file containing the text that you want to display on the screen.

1. **less:** Less command is linux utility which can be used to read contents of text file one page(one screen) per time. It has faster access because if file is large, it don’t access complete file, but access it page by page. For example, if it’s a large file and you are reading it using any text editor, then the complete file will be loaded to main memory, but less command don’t load entire file, but load it part by part, which makes it faster.

**Syntax:** less <filename>

Options used in less command are:

* + - -E : causes less to automatically exit the first time it reaches end of file.
    - -f : forces non-regular file to open.
    - -F : causes less to exit if entire file can be displayed on first screen
    - -g : highlight the string which was found by last search command
    - -G : suppresses all highlighting of strings found by search commands
    - -i : cause searches to ignore case
    - -n : suppresses line numbers
    - -p pattern : it tells less to start at the first occurrence of pattern in the file
    - -s : causes consecutive blank lines to be squeezed into a single blank line

**Result:** More linux commands has been explored successfully.

**Experiment 5**

**Q1) Write a program to swap 2 numbers.**

**Code:**

#!/bin/bash

read -p "enter the first number a=" a

read -p "enter the second number b=" b

temp=`expr $a`

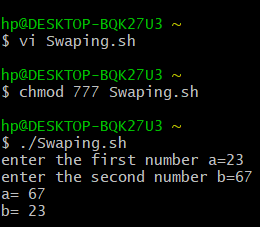
a=`expr $b`

b=`expr $temp`

echo "a=" $a

echo "b=" $b

**Output:**



**Q2) Write a script to add some text in already existing file.**

**Code:**

#!/bin/bash

echo "The contents of the file are:"

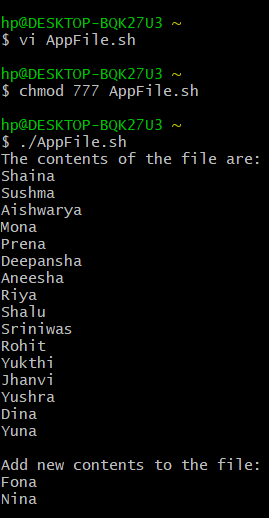
cat MyFile

echo " "

echo "Add new contents to the file:"

cat >> MyFile

**Output:**



**Q3) Write a script to demonstrate use of arithmetic operator.**

**Code:**

#!/bin/bash

read -p "enter the first number: " a;

read -p "enter the second number: " b;

c=`expr $a + $b`

d=`expr $a - $b`

e=`expr $a "\*" $b`

f=`expr $a / $b`

g=`expr $a % $b`

echo "a + b =" $c

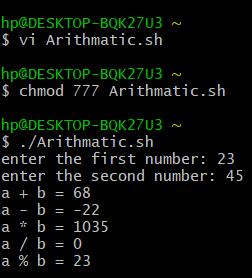
echo "a - b =" $d

echo "a "\*" b =" $e

echo "a / b =" $f

echo "a % b =" $g

**Output:**



**Q4) Write a script to delete file.**

**Code:**

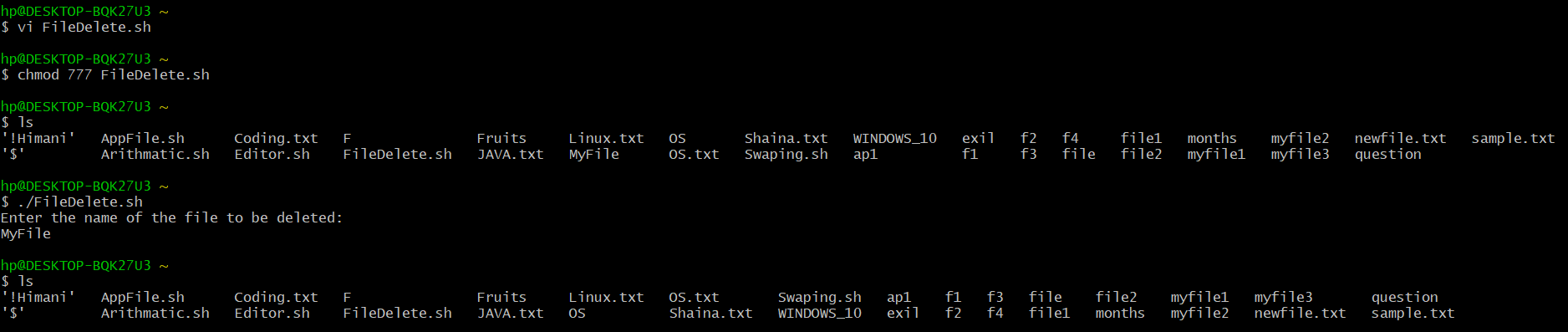
#!/bin/bash

echo "Enter the name of the file to be deleted:"

read fname

rm $fname

**Output:**



**Experiment 6**

**Q1) Write a script to check whether number is positive or negative.**

**Code:**

#!/bin/bash

read -p "enter the number" a

if [ $a -gt 0 ]

then

echo "the number is a positive number"

elif [ $a -le 0 ]

then

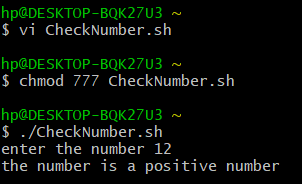
echo "the number is a negative number"

else

echo "Zero"

fi

**Output:**



**Q2) Write a script to find greatest number among three numbers.**

**Code:**

#!/bin/bash

read -p "enter the first number a=" a

read -p "enter the second number b=" b

read -p "enter the third number c=" c

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

then

echo "$a is greater"

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

then

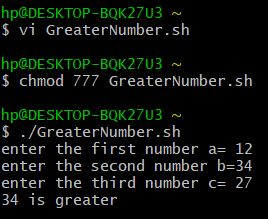
echo "$b is greater"

else

echo "$c is greater"

fi

**Output:**



**Q3) Write a script to enter the marks of a student. If the marks are greater than 70 display grade A, if the grade is greater than 60 and less than 70 display grade B, else display “Fail”.**

**Code:**

#!/bin/bash

read -p "enter the marks of the student" m

if [ $m -gt 70 ]

then

echo "A Grade"

elif [ $m -le 70 -a $m -gt 60 ]

then

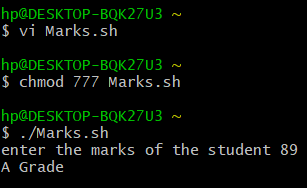
echo "B Grade"

else

echo "Fail"

fi

**Output:**



**Q4) Write a script to calculate factorial of a number.**

**Code:**

#!/bin/bash

echo Enter a number:

read a

fact=1

for (( i=$a; i > 1; i-- ))

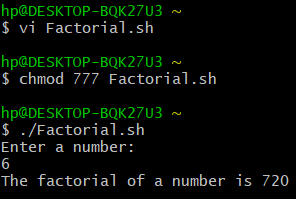
{

fact=$((fact\*i))

}

echo The factorial of a number is $fact

**Output:**



**Q5) Write a script to display whether a user is valid or not.**

**Code:**

#!/bin/bash

echo Enter the user name:

read uname

if [ "$uname" = "$USER" ]

then

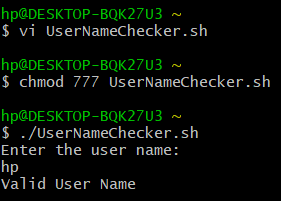
echo Valid User Name

else

echo Invalid User Name

fi

**Output:**



**Experiment 7**

**Q1) Write a script to reverse number passed using positional parameter.**

**Code:**

#!/bin/bash

a=`expr $1`

num=`expr $a`

d=0

until [ $num -eq 0 ]

do

r=`expr $num % 10`

b=`expr $d \\* 10`

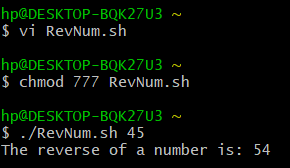
d=`expr $b + $r`

num=`expr $num / 10`

done

echo The reverse of a number is: $d

**Output:**



**Q2) Write a script to list all files in a directory using for loop.**

**Code:**

#!/bin/bash

echo Enter a directory:

read dire

cd $dire

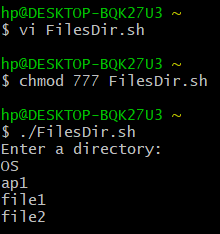
for dire in `ls`

do

echo $dire

done

**Output:**



**Q3) Write a script to find largest value passed using command line.**

**Code:**

#!/bin/bash

if [ $1 -gt $2 -a $1 -gt $3 ]

then

echo "$1 is greater"

elif [ $2 -gt $1 -a $2 -gt $3 ]

then

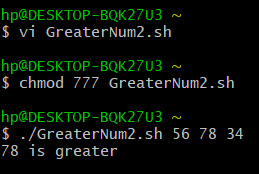
echo "$2 is greater"

else

echo "$3 is greater"

fi

**Output:**



**Q4) Write a script to search a particular file and rename it.**

**Code:**

#!/bin/bash

dire=`expr $1`

a=`expr $2`

if [ "$(ls | grep $dire)" = "$dire" ]

then

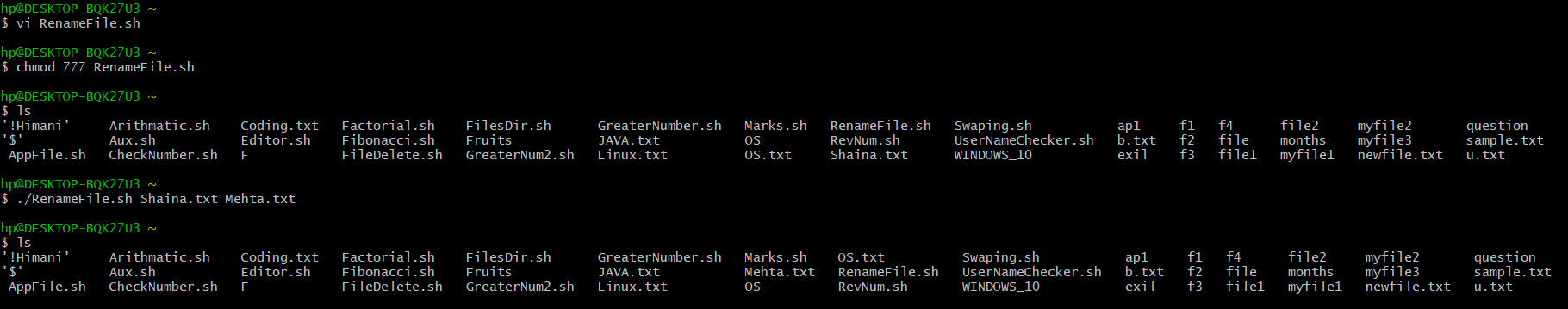
mv $dire $a

else

echo File does not exist.

fi

**Output:**



**Q5) Write a script to print Fibonacci series.**

**Code:**

#!/bin/bash

read -p "Enter a number: " a

b=0

c=1

echo $b

echo $c

d=`expr $a - 2`

for ((i=0; i<$d; i++))

do

e=`expr $b + $c`

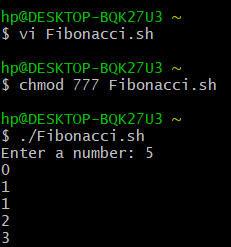
echo $e

b=`expr $c`

c=`expr $e`

done

**Output:**



**Q6) Write a script to create and delete directory using case.**

**Code:**

#!/bin/bash

echo Main Menu

echo 1. Make Directory

echo 2. Remove Diretory

read -p "Enter the choice: " ch

case $ch in

1)

read -p "Enter the name of the directory to be created: " mdire

mkdir $mdire

;;

2)

read -p "Enter the name of the directory to be deleted: " remdire

rmdir $remdire

;;

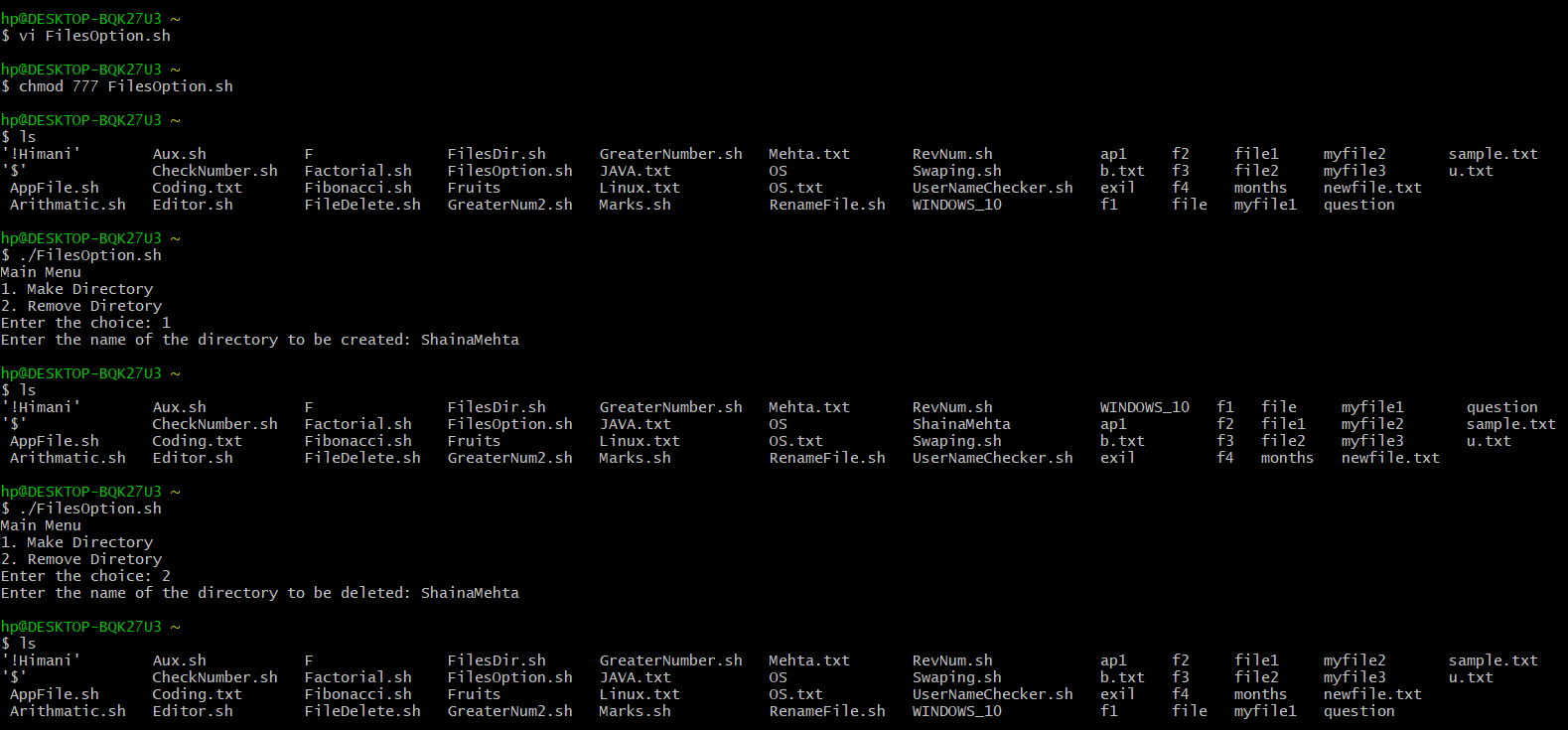
\*)

echo Invalid Choice

;;

esac

**Output:**



**Experiment 8**

**Date:** 18-02-2021

**Aim:** To simulate FCFS scheduling algorithm using C programming language.

**Software Used:** Code Blocks IDE

**Code:**

#include<stdio.h>

int main()

{

int at[10], at2[10], bt[100], ex[100], seq[100], re[100], wt[100];

int tat[100];

int n, i, j, start, position, max\_time=0, min\_time, idle=0, k=0;

float av1=0, av2=0;

printf("Enter number of process\n");

scanf("%d",&n);

printf("Enter arrival time for processes\n");

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

{

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

at2[i]=at[i];

}

printf("Enter burst time for processes\n");

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

{

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

}

start=at[0];

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

{

if(start>at[i])

{

start=at[i];

}

}

printf("Sequence of execution is\n");

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

{

if(max\_time<at[i])

{

max\_time=at[i];

}

}

max\_time=max\_time+1;

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

{ min\_time=max\_time;

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

if(at[j]!=-1)

{

if(at[j]<min\_time)

{

min\_time=at[j];

position=j;

}

} }

printf("[P%d] ",position);

seq[k]=position;

if(start<at[position]){

re[position]=start;

idle+=at[position]-start;

start=at[position];

start+=bt[position];

at[position]=-1;

ex[position]=start;

}

else{

re[position]=start;

start+=bt[position];

at[position]=-1;

ex[position]=start;

}

}

printf("\n");

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

{

tat[i]=ex[i]-at2[i];

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

}

printf("Process Arrival-time(s) Burst-time(s) Waiting-time(s) Turnaround-time(s)\n");

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

{

printf("P%d %d %d %d %d\n", i, at2[i] ,bt[i] , wt[i], tat[i]);

}

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

{

av1+=tat[i];

av2+=wt[i];

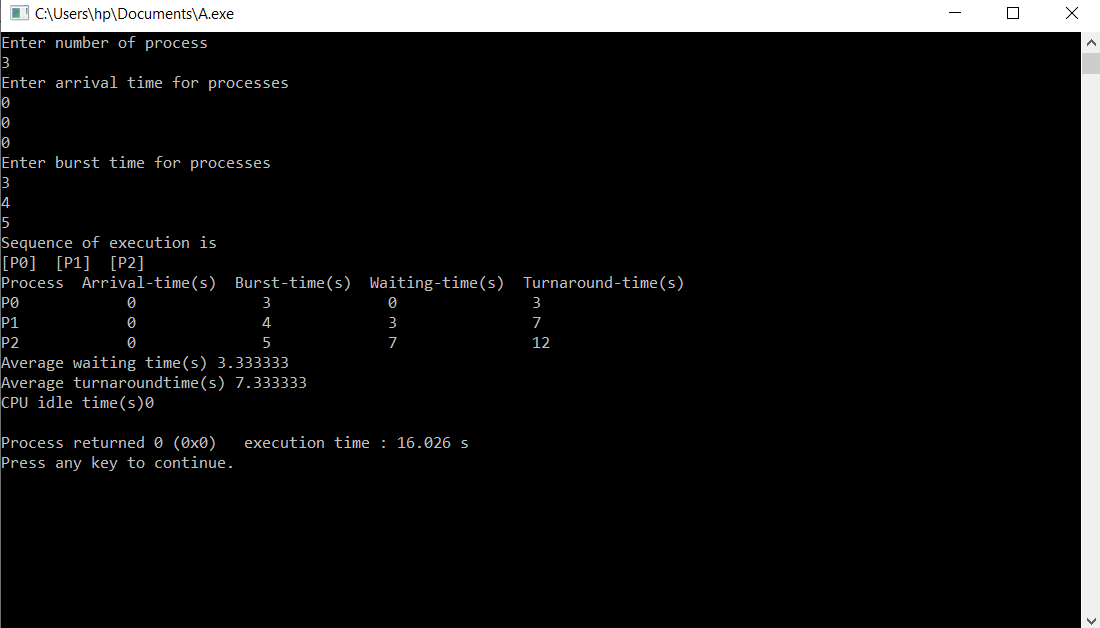
}

printf("Average waiting time(s) %f\nAverage turnaroundtime(s) %f\nCPU idle time(s)%d\n",av2/n,av1/n,idle);

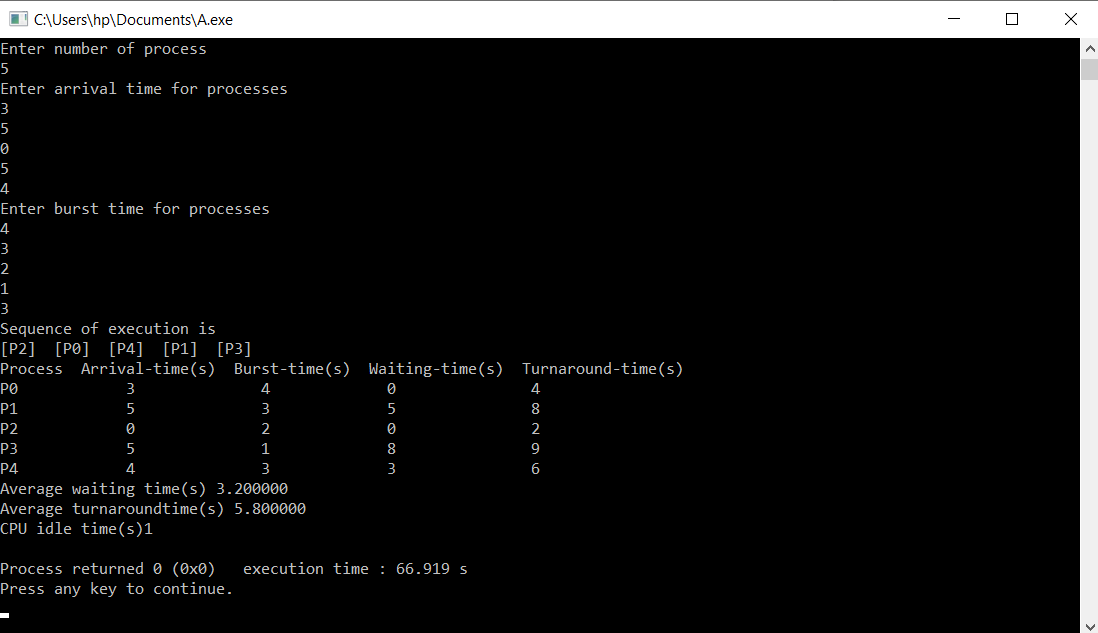
}

**Output:**

**Case 1:** Zero Arrival Time



**Case 2:** Different Arrival Time



**Conclusion:** The simulation of FCFS scheduling algorithm has been done successfully.

**Experiment 9**

**Date:** 05-03-2021

**Aim:** To simulate SJF scheduling algorithm using C programming language.

**Software Used:** Code Blocks IDE

**Preemptive:**

**Code:**

#include <stdio.h>

int main()

{

int at[10], bt[10], temp[10];

int i, smallest, count = 0, time, limit;

double wt = 0, tt = 0, end;

float avgWT, avgTT;

printf("\nEnter the Total Number of Processes:\t");

scanf("%d", &limit);

printf("\nEnter Details of %d Processes ", limit);

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

{

printf("\nEnter Arrival Time:\t");

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

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

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

temp[i] = bt[i];

}

bt[9] = 9999;

for(time = 0; count != limit; time++)

{

smallest = 9;

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

{

if(at[i] <= time &&bt[i] <bt[smallest] &&bt[i] > 0)

{

smallest = i;

}

}

bt[smallest]--;

if(bt[smallest] == 0)

{

count++;

end = time + 1;

wt = wt + end - at[smallest] - temp[smallest];

tt = tt + end - at[smallest];

}

}

avgWT = wt / limit;

avgTT= tt / limit;

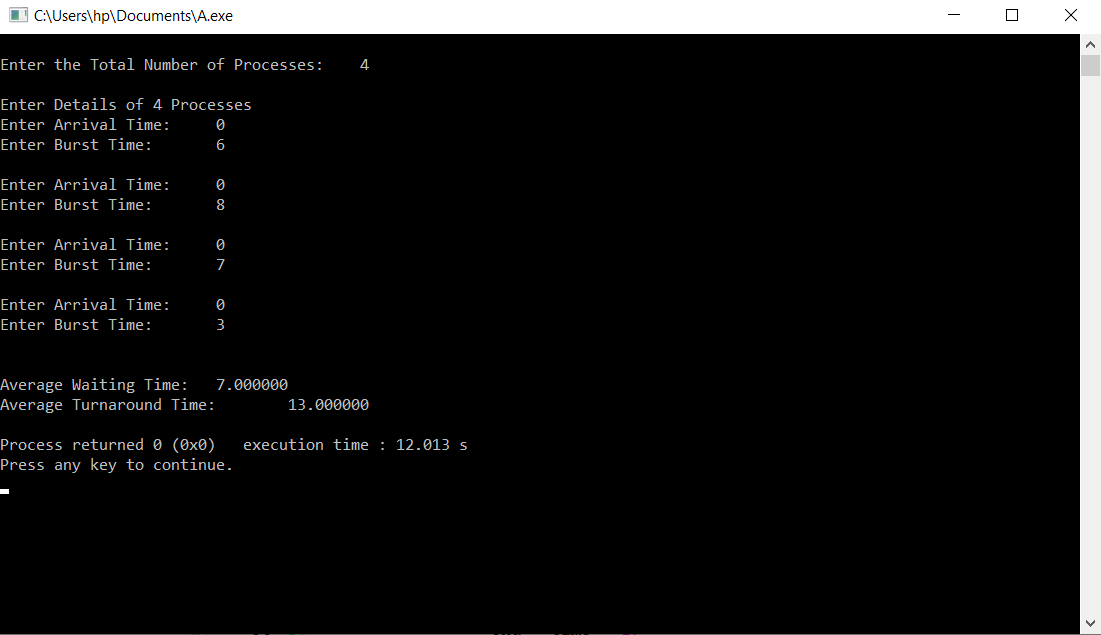
printf("\n\nAverage Waiting Time:\t%lf\n", avgWT);

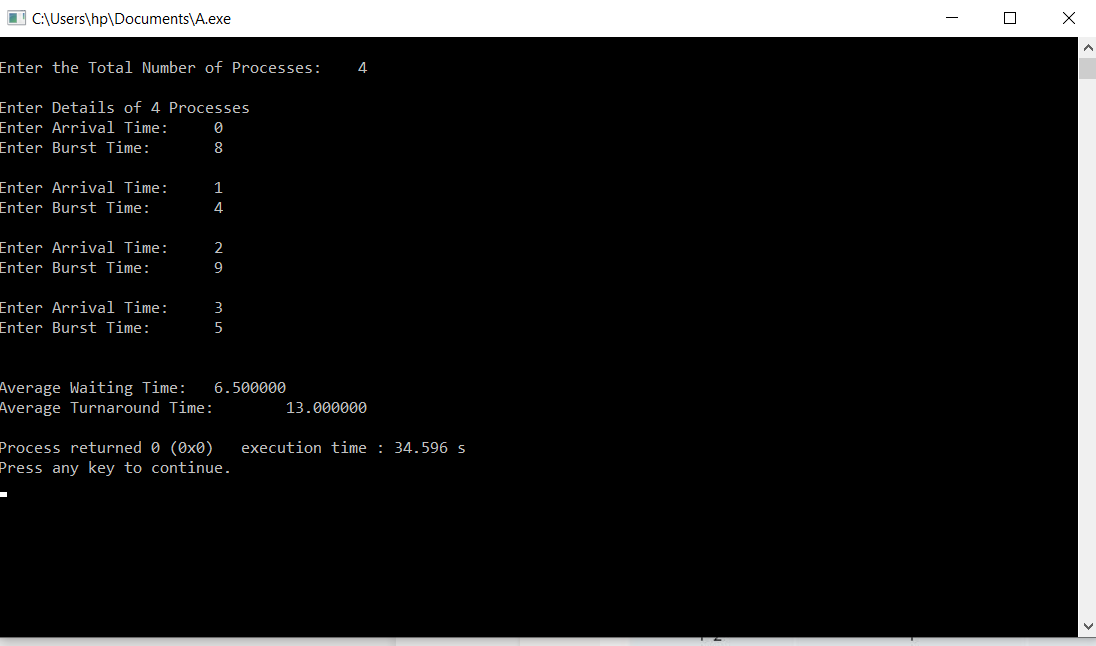
printf("Average Turnaround Time:\t%lf\n", avgTT);

return 0;

}

**Output:**





**Non - Preemptive:**

**Code:**

#include<stdio.h>

#include<conio.h>

void main()

{

int n,temp,tt=0,min,d,i,j;

float atat=0,awt=0,stat=0,swt=0;

printf("Enter no of process: ");

scanf("%d",&n);

int a[10],b[10],e[10],tat[10],wt[10];

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

{

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

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

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

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

}

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

{

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

{

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

{

temp=a[i];

a[i]=a[j];

a[j]=temp;

temp=b[i];

b[i]=b[j];

b[j]=temp;

}

}

}

min=a[0];

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

{

if(min>a[i])

{

min=a[i];

d=i;

}

}

tt=min;

e[d]=tt+b[d];

tt=e[d];

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

{

if(a[i]!=min)

{

e[i]=b[i]+tt;

tt=e[i];

}

}

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

{

tat[i]=e[i]-a[i];

stat=stat+tat[i];

wt[i]=tat[i]-b[i];

swt=swt+wt[i];

}

atat=stat/n;

awt=swt/n;

printf("Process Arrival-time(s) Burst-time(s) Waiting-time(s) Turnaround-time(s)\n");

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

{

printf("P%d\t\t%d\t\t%d\t\t%d\t\t%d\n",i+1,a[i],b[i],wt[i],tat[i]);

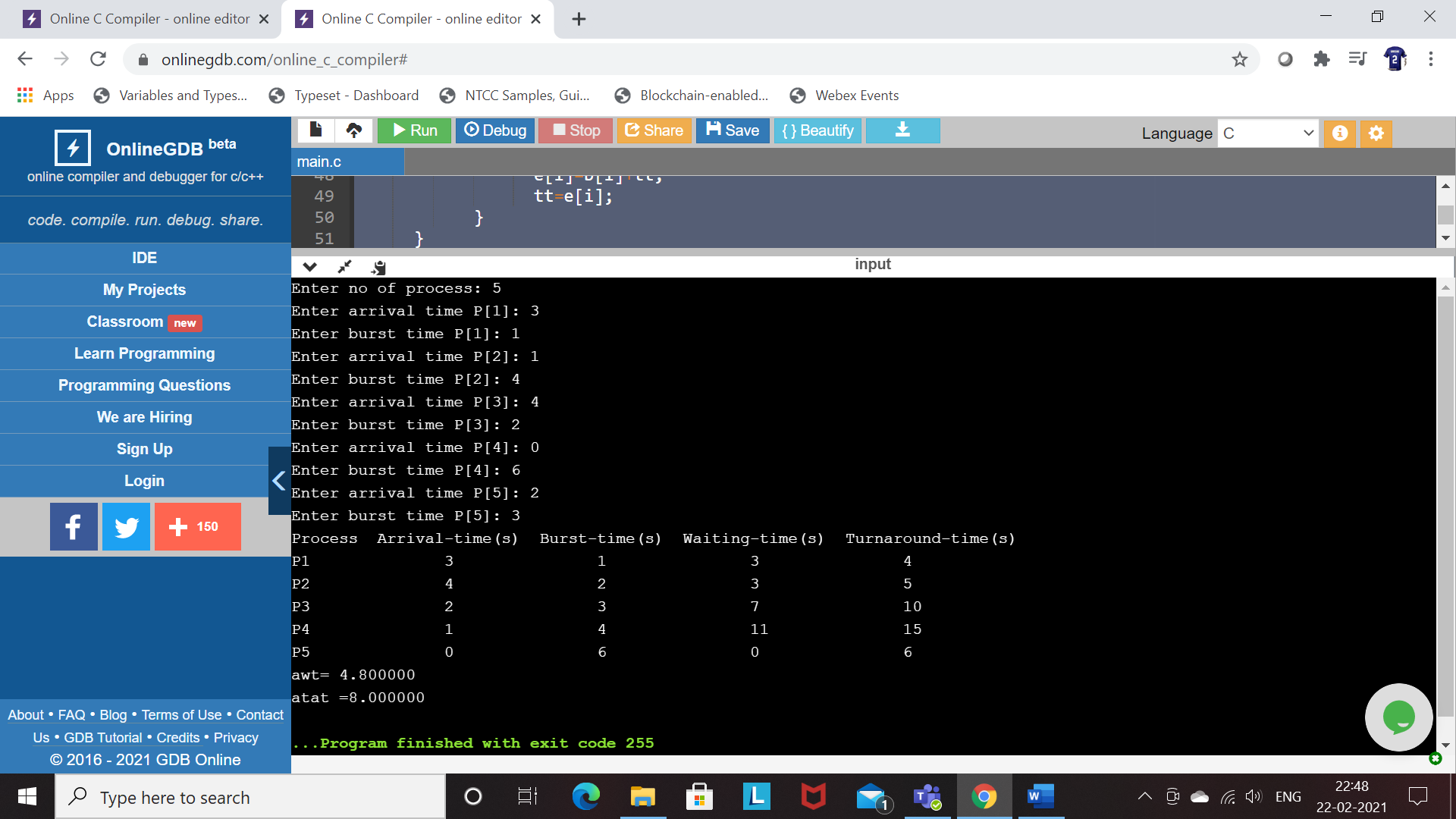
}

printf("awt= %f\natat =%f",awt,atat);

getch();

}

**Output:**



**Conclusion:** The simulation of SJF scheduling algorithm has been done successfully.

**Experiment 10**

**Date:** 24-03-2021

**Aim:** To simulate Round Robin scheduling algorithm using C programming language.

**Software Used:** Code Blocks IDE

**Code:**

#include<stdio.h>

int main()

{

int i, limit, total = 0, x, counter = 0, tq;

int wt = 0, tt = 0, at[10], bt[10], temp[10];

float avgWT, avgTT;

printf("\nEnter Total Number of Processes:\t");

scanf("%d", &limit);

x = limit;

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

{

printf("\nEnter Details of Process[%d]\n", i + 1);

printf("Arrival Time:\t");

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

printf("Burst Time:\t");

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

temp[i] = bt[i];

}

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

scanf("%d", &tq);

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

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

{

if(temp[i] <= tq && temp[i] > 0)

{

total = total + temp[i];

temp[i] = 0;

counter = 1;

}

else if(temp[i] > 0)

{

temp[i] = temp[i] - tq;

total = total + tq;

}

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

{

x--;

printf("\nProcess[%d]\t\t%d\t\t %d\t\t\t %d", i + 1, bt[i], total - at[i], total - at[i] - bt[i]);

wt = wt + total - at[i] - bm[i];

tt = tt + total - at[i];

counter = 0;

}

if(i == limit - 1)

{

i = 0;

}

else if(at[i + 1] <= total)

{

i++;

}

else

{

i = 0;

}

}

avgWT = wt \* 1.0 / limit;

avgTT = tt \* 1.0 / limit;

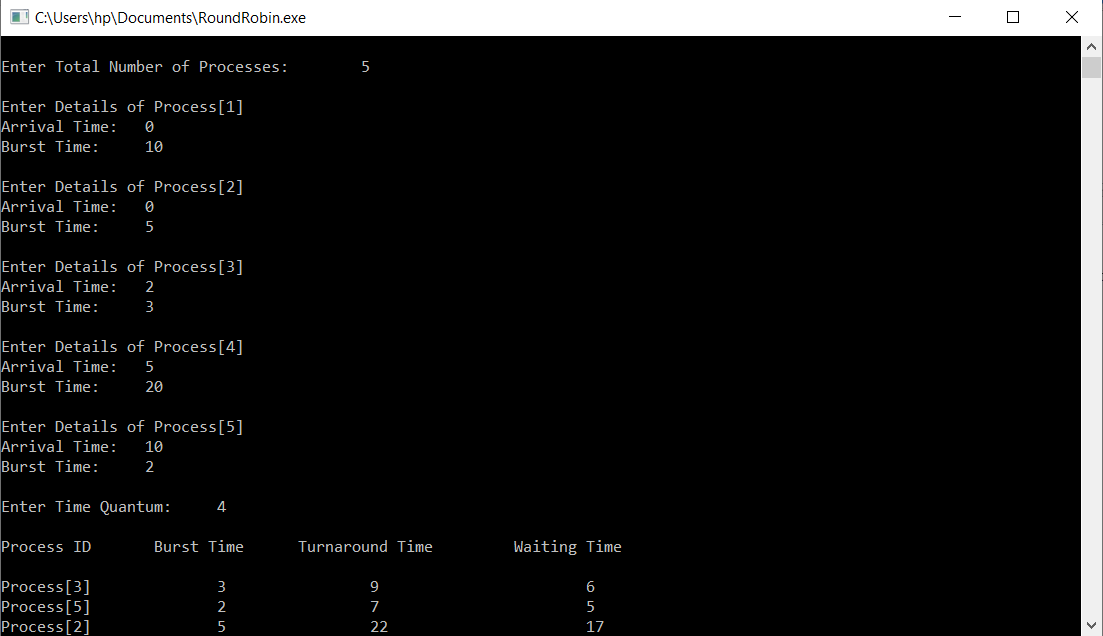
printf("\n\nAverage Waiting Time:\t%f", avgWT);

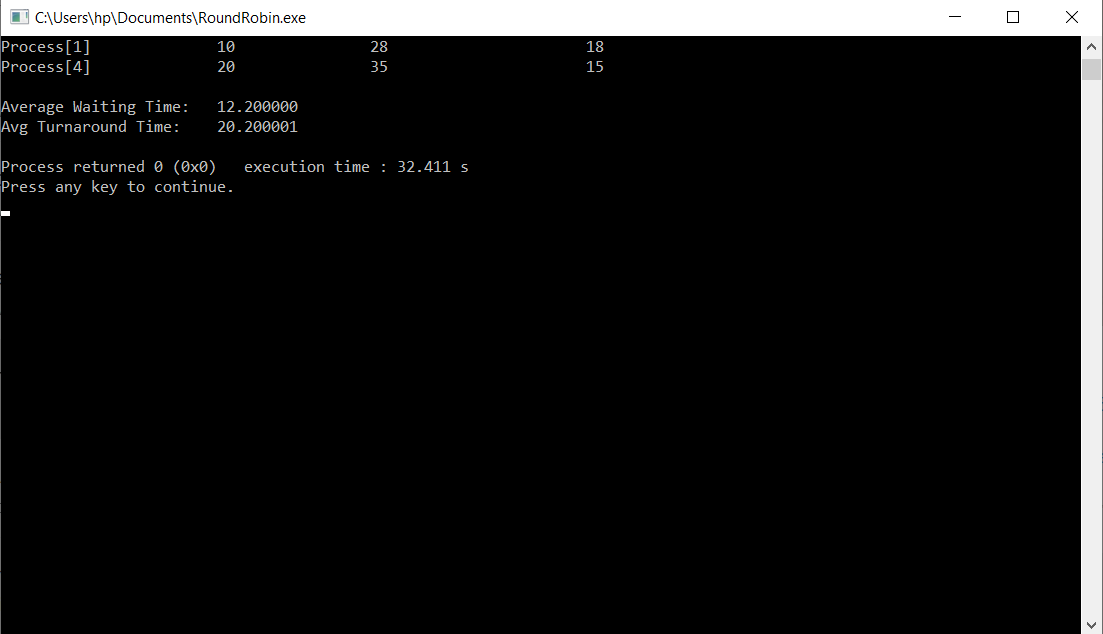
printf("\nAvg Turnaround Time:\t%f\n", avgTT);

return 0;

}

**Output:**





**Conclusion:** The simulation of Round Robin scheduling algorithm has been done successfully.

**Experiment 11**

**Date:** 24-03-2021

**Aim:** To simulate Banker’s algorithm using C programming language.

**Software Used:** Code Blocks IDE

**Code:**

#include<stdio.h>

#include<stdlib.h>

void print(int x[][10],int n,int m){

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

printf("\n");

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

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

}

}

}

void resRequest(int A[10][10],int N[10][10],int AV[10][10],int pid,int m)

{

int reqmat[1][10];

printf("\n Enter additional request :- \n");

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

printf(" Request for resource %d : ",i+1);

scanf("%d",&reqmat[0][i]);

}

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

if(reqmat[0][i] > N[pid][i]){

printf("\n Error encountered.\n");

exit(0);

}

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

if(reqmat[0][i] > AV[0][i]){

printf("\n Resources unavailable.\n");

exit(0);

}

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

AV[0][i]-=reqmat[0][i];

A[pid][i]+=reqmat[0][i];

N[pid][i]-=reqmat[0][i];

}

}

int safetyCheck(int A[][10],int N[][10],int AV[1][10],int n,int m,int a[]){

int x=0;

int F[10],W[1][10];

int pf=0,f=0;

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

F[i]=0;

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

W[0][i]=AV[0][i];

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

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

if(F[i] == 0){

f=0;

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

if(N[i][j] > W[0][j])

f=1;

}

if(f == 0 && F[i] == 0){

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

W[0][j]+=A[i][j];

F[i]=1;

pf++;

a[x++]=i;

}

}

}

if(pf == n)

return 1;

}

return 0;

}

void accept(int A[][10],int N[][10],int M[10][10],int W[1][10],int \*n,int \*m){

printf("\n Enter total no. of processes : ");

scanf("%d",n);

printf("\n Enter total no. of resources : ");

scanf("%d",m);

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

printf("\n Process %d\n",i+1);

for(int j=0;j<\*m;j++){

printf(" Allocation for resource %d : ",j+1);

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

printf(" Maximum for resource %d : ",j+1);

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

}

}

printf("\n Available resources : \n");

for(int i=0;i<\*m;i++){

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

scanf("%d",&W[0][i]);

}

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

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

N[i][j]=M[i][j]-A[i][j];

printf("\n Allocation Matrix");

print(A,\*n,\*m);

printf("\n Maximum Requirement Matrix");

print(M,\*n,\*m);

printf("\n Need Matrix");

print(N,\*n,\*m);

}

int banker(int A[][10],int N[][10],int W[1][10],int n,int m){

int j,a[10];

j=safetyCheck(A,N,W,n,m,a);

if(j != 0 ){

printf("\n\n");

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

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

printf("\n A safety sequence has been detected.\n");

return 1;

}else{

printf("\n Deadlock has occured.\n");

return 0;

}

}

int main(){

int ret;

int A[10][10];

int M[10][10];

int N[10][10];

int W[1][10];

int n,m,pid,ch;

printf("\n DEADLOCK AVOIDANCE USING BANKER'S ALGORITHM\n");

accept(A,N,M,W,&n,&m);

ret=banker(A,N,W,n,m);

if(ret !=0 ){

printf("\n Do you want make an additional request ? (1=Yes|0=No)");

scanf("%d",&ch);

if(ch == 1){

printf("\n Enter process no. : ");

scanf("%d",&pid);

resRequest(A,N,W,pid-1,m);

ret=banker(A,N,W,n,m);

if(ret == 0 )

exit(0);

}

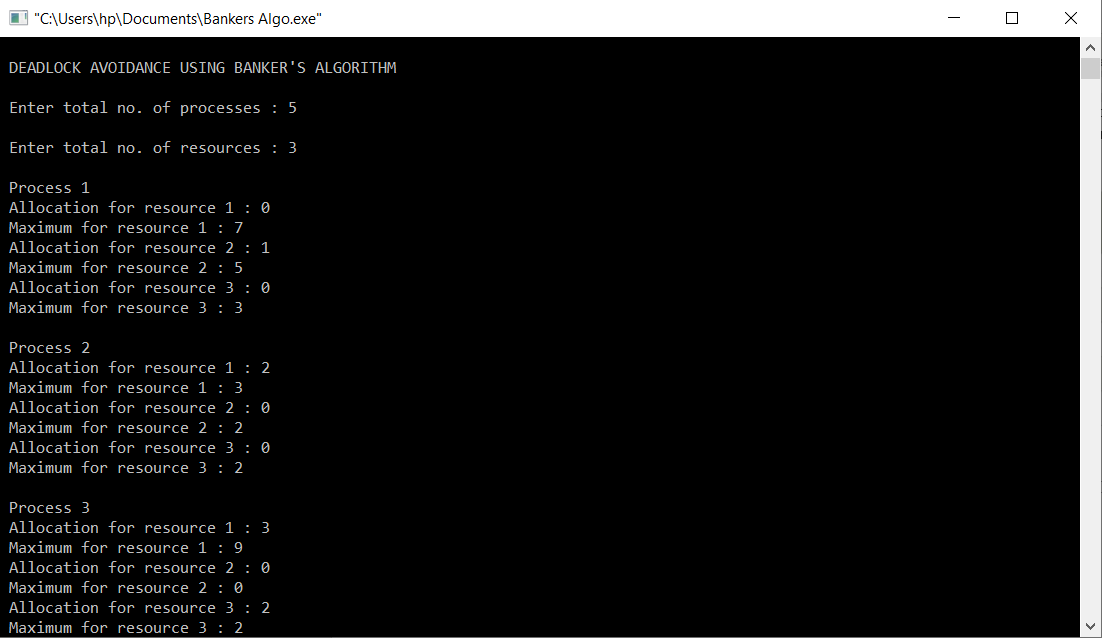
}else

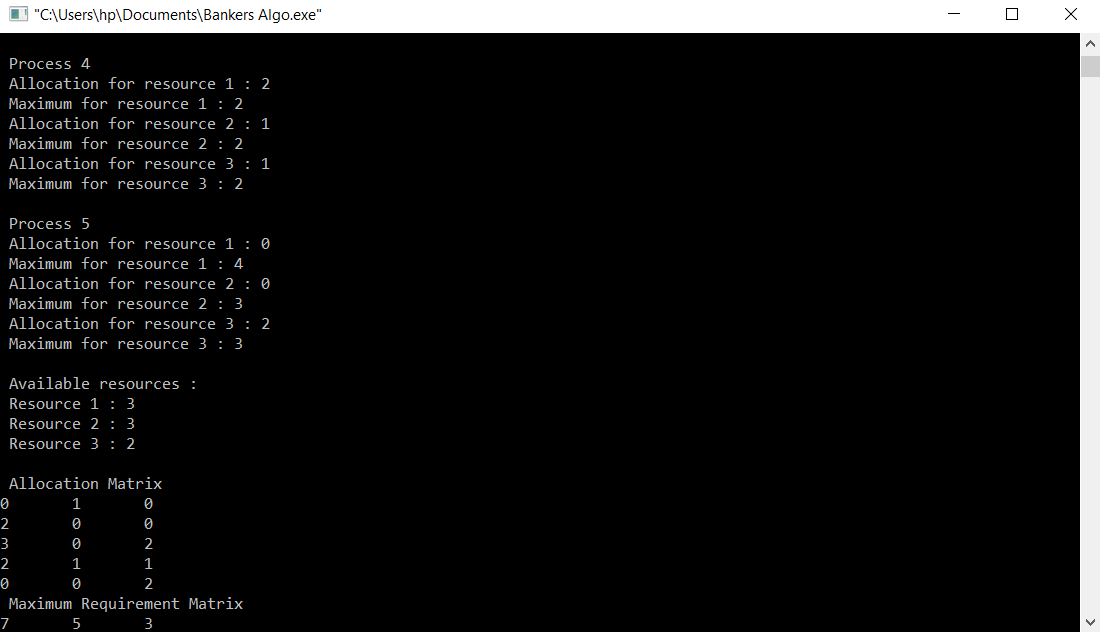
exit(0);

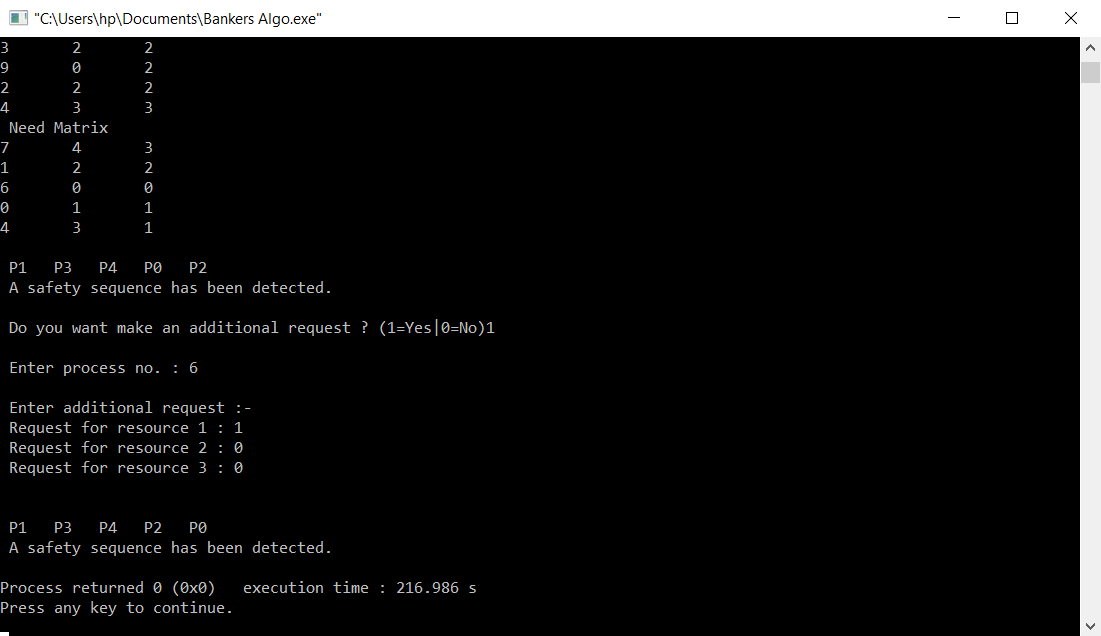
return 0;

}

**Output:**







**Conclusion:** The simulation of Banker’s algorithm has been done successfully.

**Experiment 12**

**Date:** 24-03-2021

**Aim:** To simulate FIFO disk scheduling algorithm using C programming language.

**Software Used:** Code Blocks IDE

**Code:**

#include <stdio.h>

#include <math.h>

int main(){

int requests[51],rComp[50],n,total=0;

printf("Enter the current position of the pointer of the disk: ");

scanf("%d",&requests[0]);

printf("Enter number of requests to be executed(Not more than 50): ");

scanf("%d",&n);

printf("Enter the requests: ");

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

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

}

for(int i=1,j=i-1;i<=n;i++,j++){

rComp[j]=abs(requests[j]-requests[i]);

total+=rComp[j];

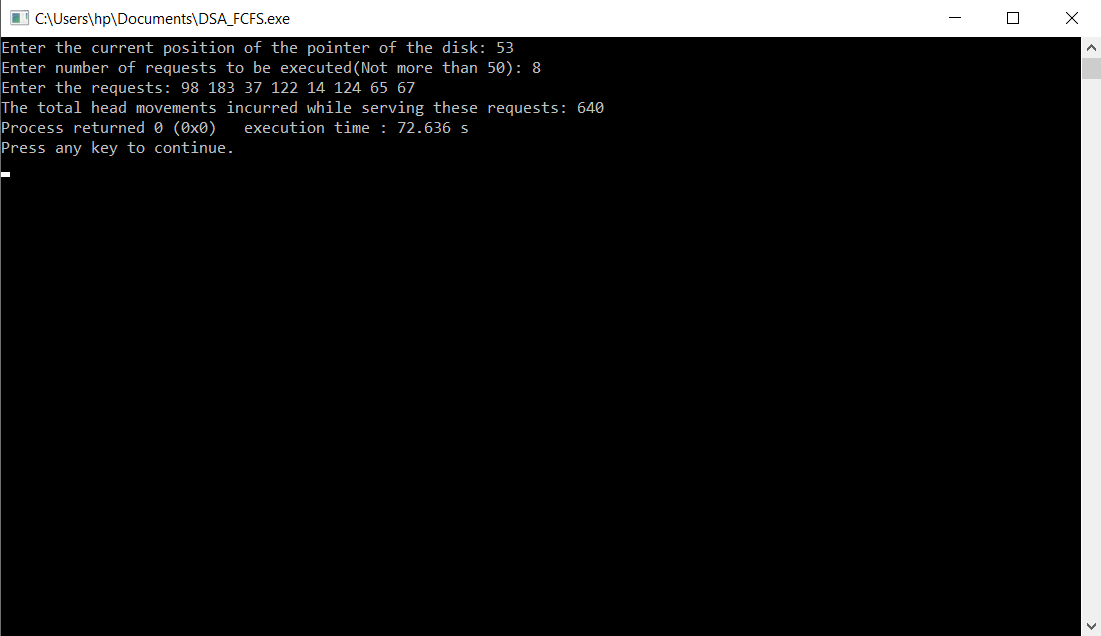
}

printf("The total head movements incurred while serving these requests: %d",total);

return 0;

}

**Output:**



**Conclusion:** The simulation of FIFO disk scheduling algorithm has been done successfully.