**LINUX PROGRAMMING LAB EXPERIMENTS**

1. **Basic Linux Commands File handling utilities, Security by file permissions, Process utilities, Disk utilities, sed, awk, grep**.

**File handling Commands:**  cat ,touch ,rm, mv, vi, chmod, cp, mkdir, cd, ls , find ,chown ,chgrp

**Process utilities:** ps , who, who am i , top

**Disk Utilities:** df , du

**Filters:** head, tail, cut, paste, sort, tr, wc, cmp

Grep command

Sed command

Awk

**Basic Linux Commands Guide**

**1. File Handling Commands**

- cat: Concatenate and display file contents.

Usage: `cat filename`

- touch: Create an empty file or update the modification timestamp.

Usage: `touch filename`

- rm: Remove files or directories.

Usage: `rm filename` or `rm -r directory`

- mv: Move or rename files.

Usage: `mv oldname newname` or `mv filename directory`

- vi: Open the Vi editor to edit text files.

Usage: `vi filename`

- chmod: Change file permissions.

Usage: `chmod 755 filename`

- cp: Copy files or directories.

Usage: `cp source destination`

- mkdir: Create a new directory.

Usage: `mkdir directory\_name`

- cd: Change directory.

Usage: `cd directory\_name`

- ls: List files and directories.

Usage: `ls -l` (long format)

- find: Search for files and directories.

Usage: `find /path -name filename`

- chown: Change file owner.

Usage: `chown user:group filename`

- chgrp: Change file group ownership.

Usage: `chgrp group filename`

**2. Process Utilities**

- ps: Display current active processes.

Usage: `ps aux`

- who: Show who is logged in.

Usage: `who`

- who am i: Show details of the current session.

Usage: `who am i`

- top: Real-time display of system processes.

Usage: `top`

**3. Disk Utilities**

- df: Show disk space usage.

Usage: `df -h`

- du: Estimate file or directory space usage.

Usage: `du -sh directory\_name`

**4. Filters**

- head: Display the first lines of a file.

Usage: `head filename`

- tail: Display the last lines of a file.

Usage: `tail filename`

- cut: Extract sections of a file.

Usage: `cut -d':' -f1 filename`

- paste: Merge lines of files.

Usage: `paste file1 file2`

- sort: Sort lines in text files.

Usage: `sort filename`

- tr: Translate characters.

Usage: `tr 'a-z' 'A-Z' < filename`

- wc: Count words, lines, or bytes.

Usage: `wc filename`

- cmp: Compare two files.

Usage: `cmp file1 file2`

**5. Grep Command**

- grep: Search for a specific pattern in files.

Usage: `grep 'pattern' filename`

**6. Sed Command**

- sed: Stream editor for filtering and transforming text.

Usage: `sed 's/old/new/g' filename`

**7. Awk Command**

- awk: Pattern scanning and processing language.

Usage: `awk '{print $1}' filename`

1. **Write a shell script that accepts a file name, starting and ending line numbers as arguments and displays all the lines between the given line numbers.**

**PROGRAM:**

echo "enter file name"

read f

echo 'enter starting position'

read st

echo 'enter ending position'

read end

echo 'The lines between' $st 'and' $end 'from' $f

if [ $st -lt $end ]

then

n1=`expr $st + 1`

n2=`expr $end - 1`

sed -n "$n1,$n2 p" $f

elif [ $st -gt $end ]

then

n3=`expr $st - 1`

n4=`expr $end + 1`

sed -n "$n4,$n3 p" $f

fi

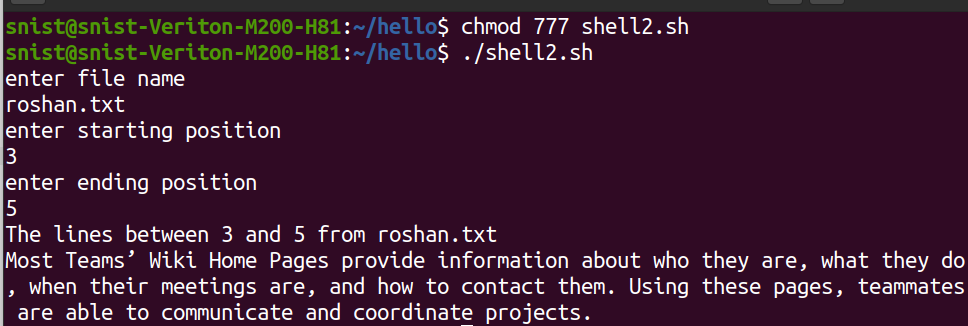
**OUTPUT:**

1.Save the file with .sh extension.

2.Using chmod command give the permission to execute the file .

3. Type to see the output

sh filename.sh or ./filename.sh



1. **Write a shell script that deletes all lines containing a specified word in one or more files supplied as arguments to it.**

**PROGRAM:**

echo 'enter a word to be deleted'

read word

echo 'enter file name'

read fname

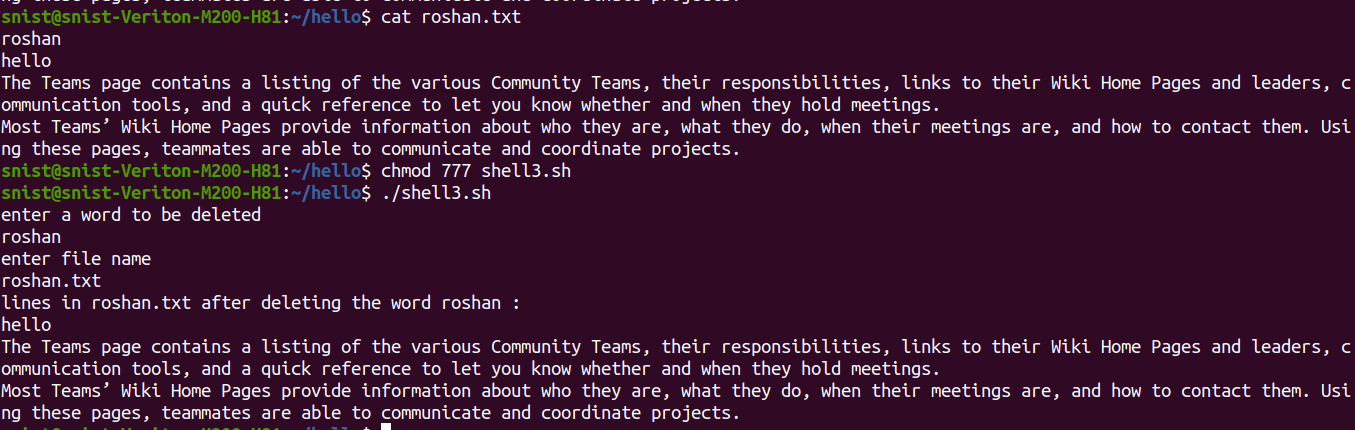
echo 'lines in' $fname 'after deleting the word' $word ':'

sed "/$word/d" $fname

**(or)**

if [ $# -eq 0 ]  
then  
echo "Please enter one or more filenames as argument"  
exit  
fi  
echo "Enter the word to be searched in files"  
read word  
for file in $\*  
do  
sed "/$word/d" $file | tee tmp  
mv tmp $file  
done

**OUTPUT:**



**4. Write a shell script that displays a list of all the files in the current directory to which the user has read, write and execute permissions.**

**PROGRAM:**

for i in \*

do

if [ -r $i -a -w $i -a -x $i ]

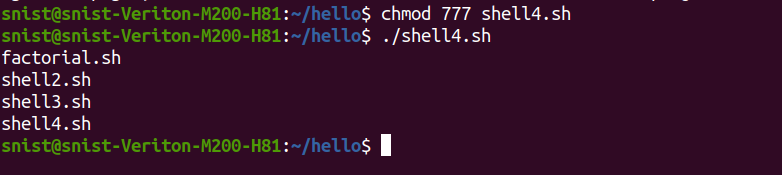
then

echo $i

fi

done

**OUTPUT:**



**5. C Programming examples using Linux operating systems.**

**a. wc**

**PROGRAM:**

#include <stdio.h>

#include <ctype.h>

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

FILE \*file;

int c, lines = 0, words = 0, chars = 0;

if (argc != 2) {

fprintf(stderr, "Usage: %s <filename>\n", argv[0]);

return 1;

}

file = fopen(argv[1], "r");

if (!file) {

perror("Error opening file");

return 1;

}

while ((c = fgetc(file)) != EOF) {

chars++;

if (c == '\n') {

lines++;

}

if (isspace(c)) {

words++;

}

}

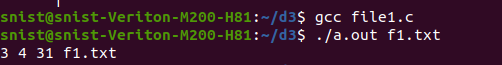
fclose(file);

printf("%d %d %d %s\n", lines, words, chars, argv[1]);

return 0;

}

**OUTPUT:**



**b. Cat**

**PROGRAM:**

**File1.c**

#include <stdio.h>

#include <stdlib.h>

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

    FILE \*file;

    char ch;

    // Check if at least one file name is provided

    if (argc < 2) {

        printf("Usage: %s <file1> <file2> ...\n", argv[0]);

        return 1;

    }

    // Loop through each file provided in the command line

    for (int i = 1; i < argc; i++) {

        file = fopen(argv[i], "r");  // Open the file in read mode

        if (file == NULL) {

            perror("Error opening file");

            continue;  // Move to the next file

        }

        // Read the file character by character and print it to the console

        while ((ch = fgetc(file)) != EOF) {

            putchar(ch);  // Print the character to stdout

        }

        fclose(file);  // Close the file after reading

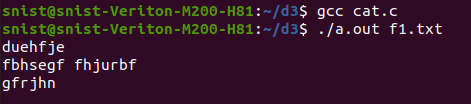
        printf("\n");   // Print a newline after each file's content

    }

    return 0;

}

**OUTPUT:**



**c. cp**

#include <stdio.h>

#include <fcntl.h>

#include <unistd.h>

#define BUF\_SIZE 1024

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

    int src, dest, n;

    char buf[BUF\_SIZE];

    if (argc != 3) {

        write(2, "Usage: cp <source> <destination>\n", 33);

        return 1;

    }

    src = open(argv[1], O\_RDONLY);

    if (src == -1) return 1;

    dest = open(argv[2], O\_WRONLY | O\_CREAT | O\_TRUNC, 0644);

    if (dest == -1) return 1;

    while ((n = read(src, buf, BUF\_SIZE)) > 0)

        write(dest, buf, n);

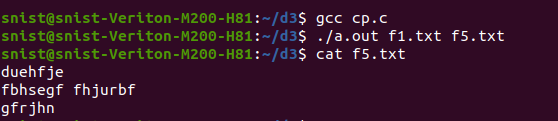
    close(src);

    close(dest);

    return 0;

}

**OUTPUT:**



**6. Write a shell script that receives any number of file names as arguments checks if every argument supplied is a file or directory and reports accordingly. Whenever the argument is a file, the number of lines on it is also reported.**

**PROGRAM:**

for fname in $\*

do

if [ -f $fname ]

then

echo $fname 'is a file'

echo 'no.of lines in' $fname ':'

wc -l $fname

elif [ -d $fname ]

then

echo $fname 'is a directory'

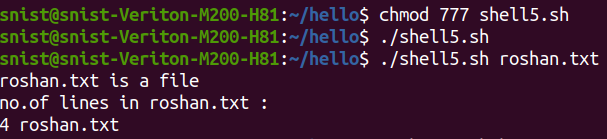
else

echo 'Does not exist'

fi

done

**OUTPUT:**



**7.** **Write the following Shell scripts**:

1. **Write a shell script that accepts a file names as its arguments, counts and reports the occurrence of each word that is present in the file argument file in other argument files.**

**PROGRAM:**

**shell.sh**

if [ $# -lt 2 ]; then

echo "Usage: $0 source\_file target\_file1 [target\_file2 ...]"

exit 1

fi

source\_file=$1

shift

if [ ! -f "$source\_file" ]; then

echo "Source file '$source\_file' does not exist."

exit 1

fi

for word in $(tr -s '[:space:]' '\n' < "$source\_file"); do

total\_count=0

for target\_file in "$@"; do

if [ ! -f "$target\_file" ]; then

echo "Target file '$target\_file' does not exist."

continue

fi

count=$(grep -oiw "$word" "$target\_file" | wc -l)

total\_count=$((total\_count + count))

done

if [ $total\_count -gt 0 ]; then

echo "Word '$word' found $total\_count times in the target files."

else

echo "Word '$word' not found in any target files."

fi

done

**file1.txt**

apple

banana

orange

**file2.txt**

apple orange apple

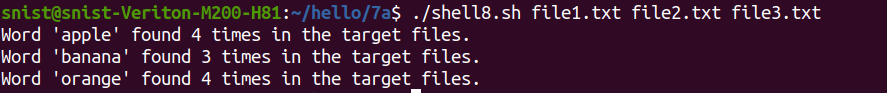
banana orange banana

**file3.txt**

apple apple banana

orange orange

**OUTPUT:**



1. **To list all of the directory files in a directory.**

**PROGRAM:**

echo 'enter a directory name'

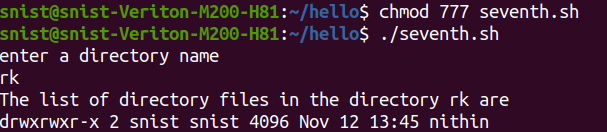
read dname

echo 'The list of directory files in the directory' $dname 'are'

cd $dname

ls -l | grep '^d'

**OUTPUT:**

****

1. **Write a shell script to find factorial of a given number.**

**PROGRAM:**

echo "enter n"

read n

i=1

fact=1

while [ $i -le $n ]

do

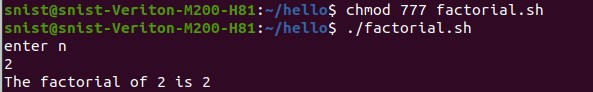
fact=`expr $fact \\* $i`

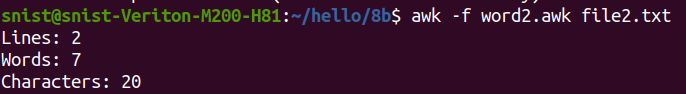
i=`expr $i + 1`

done

echo 'The factorial of' $n 'is' $fact

**OUTPUT:**





**8. Implement in C the following Unix commands using System calls**

**a. rename**

**PROGRAM:**

#include <stdio.h>

#include <stdlib.h>

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

    // Check if correct number of arguments is provided

    if (argc != 3) {

        fprintf(stderr, "Usage: %s <old\_name> <new\_name>\n", argv[0]);

        return 1;

    }

    // Rename the file using the rename() system call

    if (rename(argv[1], argv[2]) == 0) {

        printf("File renamed successfully: %s -> %s\n", argv[1], argv[2]);

    } else {

        perror("Error renaming file");

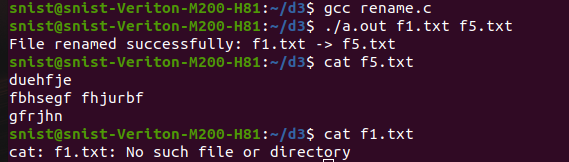
        return 1;

    }

    return 0;

}

**OUTPUT:**



**b.link**

**PROGRAM:**

#include <stdio.h>

#include <unistd.h>

#include <errno.h>

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

    if (argc != 3) {

        // Check if the correct number of arguments is provided

        fprintf(stderr, "Usage: %s <source\_file> <link\_name>\n", argv[0]);

        return 1;

    }

    char \*source\_file = argv[1];

    char \*link\_name = argv[2];

    // Use the link system call to create a hard link

    if (link(source\_file, link\_name) == -1) {

        // If the link creation fails, print an error

        perror("Error creating link");

        return 1;

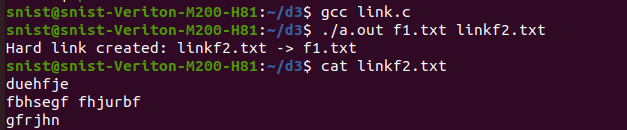
    }

    printf("Hard link created: %s -> %s\n", link\_name, source\_file);

    return 0;

}

**OUTPUT:**



**9**. **Write a C program to emulate the Unix ls – l command.**

#include <stdio.h>

#include <stdlib.h>

#include <dirent.h>

int main(int argc, char \*\*argv) {

    DIR \*dirop;

    struct dirent \*dired;

    if (argc != 2) { // Check if the correct number of arguments is provided

        printf("Invalid number of arguments. Usage: %s <directory\_name>\n", argv[0]);

        exit(1); // Exit if the number of arguments is incorrect

    }

    // Try to open the directory specified in the argument

    if ((dirop = opendir(argv[1])) == NULL) {

        printf("Cannot open directory %s\n", argv[1]);

        exit(1); // Exit if the directory cannot be opened

    }

    while ((dired = readdir(dirop)) != NULL) { // Read and print directory entries

        printf("%10lu %s\n", dired->d\_ino, dired->d\_name); // Print inode number and filename

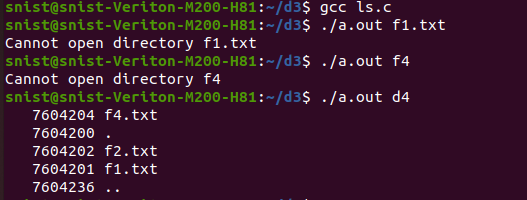
    }

    closedir(dirop);     // Close the directory stream

    return 0;

}

**OUTPUT:**



**10.** **Write a C program on zombie process**

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

int main() {

    pid\_t pid

    pid = fork(); // Create a child process

    if (pid < 0) {

        perror("Fork failed");

        return 1;

    }

    else if (pid == 0) { // Child process

        printf("Child process (PID: %d) is exiting...\n", getpid());

        \_exit(0);  // Child exits without the parent collecting the exit status

    }

    else { // Parent process

        printf("Parent process (PID: %d) is waiting...\n", getpid());

        // Parent does not call wait() immediately, so the child becomes a zombie

        sleep(10);  // Sleep for 10 seconds to allow the child to become a zombie

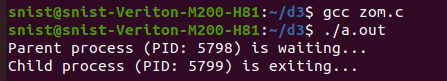
        printf("Parent process (PID: %d) exiting...\n", getpid());

    }

    return 0;

}

**OUTPUT:**



**11. Write a C program that illustrates the following. a) Creating a message queue. b) Writing to a message queue. c) Reading from a message queue.**

**PROGRAM:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/msg.h>

// Define the structure for the message queue

struct msgbuf {

    long mtype;      // Message type (must be a positive integer)

    char mtext[100]; // Message text (content)

};

int main() {

    int msqid;          // Message queue ID

    int len, ret;       // Length of the message and return values

    struct msgbuf msgsend = {0, "\0"}, msgrecv; // Message buffers for sending and receiving

    // Step 1: Create a message queue (with key 1234)

    msqid = msgget((key\_t)1234, IPC\_CREAT | 0666);  // Create the message queue with read-write permissions for all users

    if (msqid == -1) {

        perror("msgget:");

        exit(1);

    }

    printf("Message Queue Created with ID: %d\n", msqid);

    // Step 2: Write a message to the message queue

    printf("Enter message type (positive integer): ");

    if (scanf("%ld", &msgsend.mtype) != 1) {

        fprintf(stderr, "Invalid message type input.\n");

        exit(1);

    }

    printf("Enter message text: ");

    if (scanf("%s", msgsend.mtext) != 1) {

        fprintf(stderr, "Invalid message text input.\n");

        exit(1);

    }

    len = strlen(msgsend.mtext);  // Calculate the length of the message text

    ret = msgsnd(msqid, &msgsend, len, 0);  // Send the message to the queue

    if (ret == -1) {

        perror("msgsnd:");

        exit(1);

    }

    printf("Message Sent: %s\n", msgsend.mtext);

    // Step 3: Read the message from the message queue

    ret = msgrcv(msqid, &msgrecv, sizeof(msgrecv.mtext), msgsend.mtype, 0);  // Receive the message with the same type

    if (ret == -1) {

        perror("msgrcv:");

        exit(1);

    }

    printf("Message Received: %s\n", msgrecv.mtext);

    // Step 4: Clean up by removing the message queue (optional)

    ret = msgctl(msqid, IPC\_RMID, NULL);  // Remove the message queue

    if (ret == -1) {

        perror("msgctl:");

        exit(1);

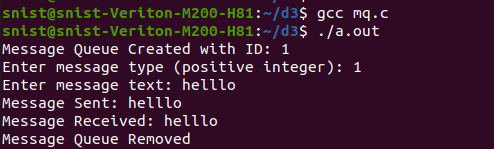
    }

    printf("Message Queue Removed\n");

    return 0;

}

**OUTPUT:**



**12. Write a C program to implement record locking.**

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <unistd.h>

#include <errno.h>

#define FILENAME "record\_file.txt"

int main() {

    int fd;

    struct flock lock;

    // Open the file

    fd = open(FILENAME, O\_RDWR | O\_CREAT, 0666);

    if (fd == -1) {

        perror("Failed to open file");

        exit(1);

    }

    // Initialize the lock structure

    lock.l\_type = F\_WRLCK;  // Set the lock type to write lock

    lock.l\_whence = SEEK\_SET;  // Lock is from the beginning of the file

    lock.l\_start = 0;   // Start of the file (you can set it to a specific record offset)

    lock.l\_len = 100;   // Length of the record to lock (here locking 100 bytes)

    // Try to acquire the lock

    if (fcntl(fd, F\_SETLK, &lock) == -1) {

        if (errno == EACCES || errno == EAGAIN) {

            printf("Record is already locked by another process.\n");

        } else {

            perror("Failed to lock record");

        }

        close(fd);

        exit(1);

    }

    printf("Record locked successfully.\n");

    // Simulate file access (sleep for 5 seconds)

    sleep(5);

    // Release the lock

    lock.l\_type = F\_UNLCK;  // Change the lock type to unlock

    if (fcntl(fd, F\_SETLK, &lock) == -1) {

        perror("Failed to unlock record");

        close(fd);

        exit(1);

    }

    printf("Record unlocked successfully.\n");

    // Close the file

    close(fd);

    return 0;

}

**OUTPUT:**

