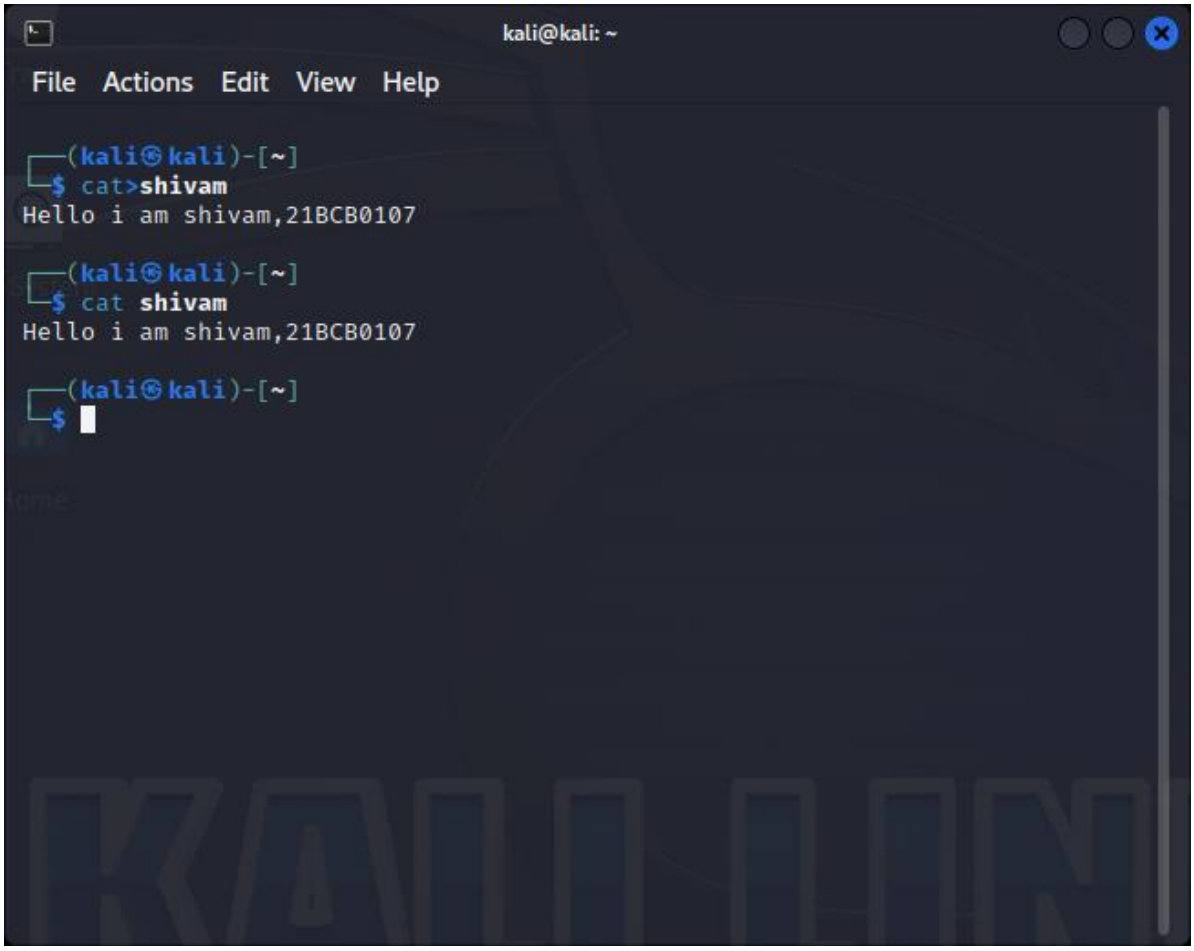
Reg No : 21BCB0107
Name : Shivam Dave
Slot : B2

Exercise 1 – Operating System Basics

Question – I

Demonstrate various Linux commands

1. Create a file:
   cat>shivam
   Hello I am shivam,21BCB0107
   cat shivam
   Output:



2. Copying file:
   cat shivam copy1
   cat copy1

Output:

```
kali@kali: ~
File  Actions  Edit  View  Help

┌──(kali㉿kali)-[~]
└─$ cat>shivam
Hello i am shivam,21BCB0107

┌──(kali㉿kali)-[~]
└─$ cat shivam
Hello i am shivam,21BCB0107

┌──(kali㉿kali)-[~]
└─$

┌──(kali㉿kali)-[~]
└─$ cp shivam copy1

┌──(kali㉿kali)-[~]
└─$ cat copy1
Hello i am shivam,21BCB0107

┌──(kali㉿kali)-[~]
└─$
```

3. Renaming file:
   Code:
   cp shivam copy1
   cat copy1
   Output:

```
┌──(kali㉿kali)-[~]
└─$ cp shivam copy1

┌──(kali㉿kali)-[~]
└─$ cat copy1
Hello i am shivam,21BCB0107

┌──(kali㉿kali)-[~]
└─$
```

4. Removing file:
   Code:
   rm cp1

cat cp1

Output:

```
┌──(kali㉿kali)-[~]
└─$ rm cp1

┌──(kali㉿kali)-[~]
└─$ cat cp1
cat: cp1: No such file or directory

┌──(kali㉿kali)-[~]
└─$
```

5. Creating a directory:

   Code:

   Mkdir dir1

   Output:

```
┌──(kali㉿kali)-[~]
└─$ mkdir dir1

┌──(kali㉿kali)-[~]
└─$
```

6. Moving and copying files into one directory

   Code:

   Mv shivam dir1

   Output:

```
┌──(kali㉿kali)-[~]
└─$ mv shivam dir1

┌──(kali㉿kali)-[~]
└─$
```

7. Copying a directory

   Code:

   cp -r dir1 dir1copy

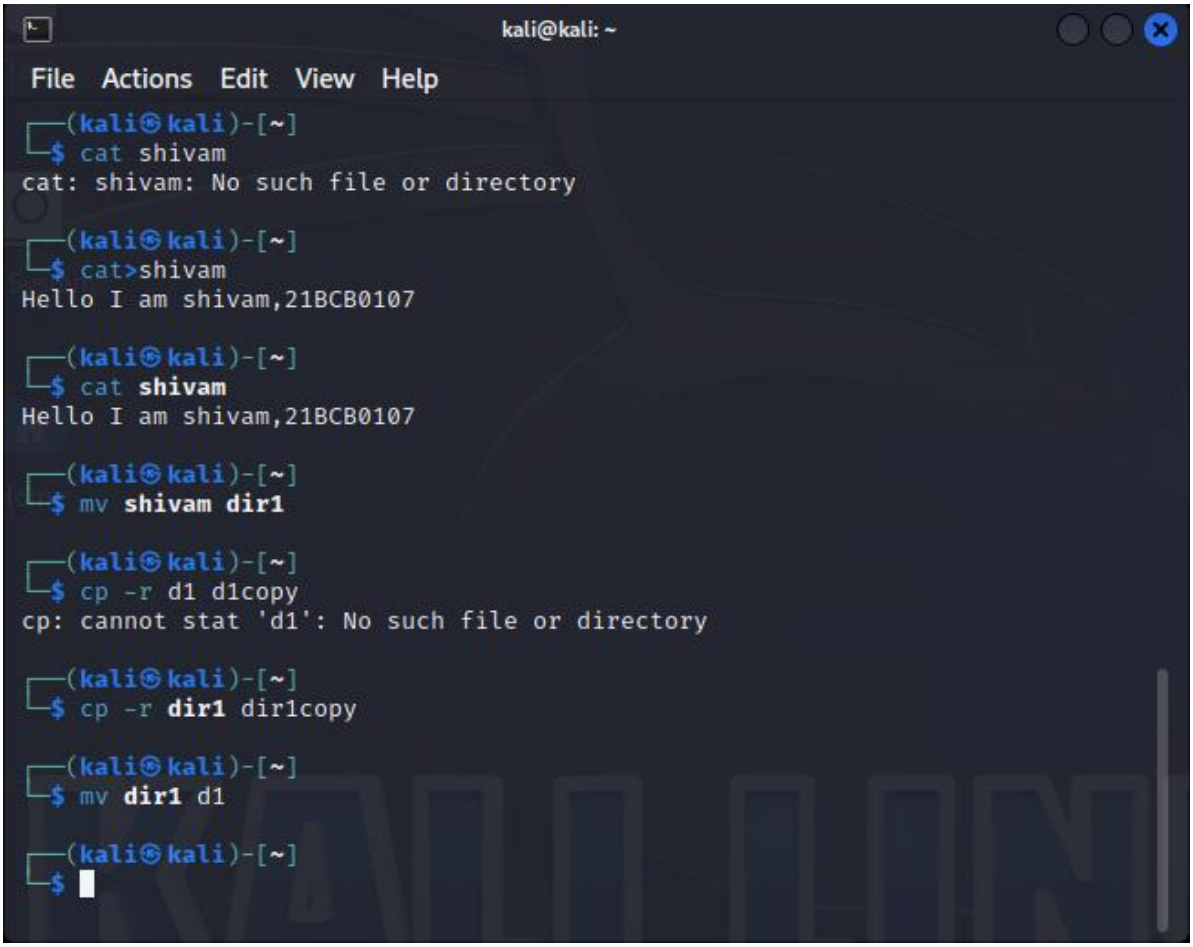
   Output:

```
┌──(kali㉿kali)-[~]
└─$ cp -r dir1 dir1copy

┌──(kali㉿kali)-[~]
└─$
```

8. Renaming a directory

   Code:

   Mv dir1 d1



9. Removing a directory

Code:

Rmdir d1copy

Rmdir -r d1copy

Output:

```
kali@kali: ~
File  Actions  Edit  View  Help

┌──(kali㉿kali)-[~]
└─$ mv dir1 d1

┌──(kali㉿kali)-[~]
└─$ rmdir d1copy
rmdir: failed to remove 'd1copy': No such file or directory

┌──(kali㉿kali)-[~]
└─$ rm -d1 d1copy
rm: invalid option -- '1'
Try 'rm --help' for more information.

┌──(kali㉿kali)-[~]
└─$ cp -r d1 d1copy

┌──(kali㉿kali)-[~]
└─$ rmdir d1copy
rmdir: failed to remove 'd1copy': Directory not empty

┌──(kali㉿kali)-[~]
└─$ rmdir -r d1copy
rmdir: invalid option -- 'r'
Try 'rmdir --help' for more information.

┌──(kali㉿kali)-[~]
└─$
```

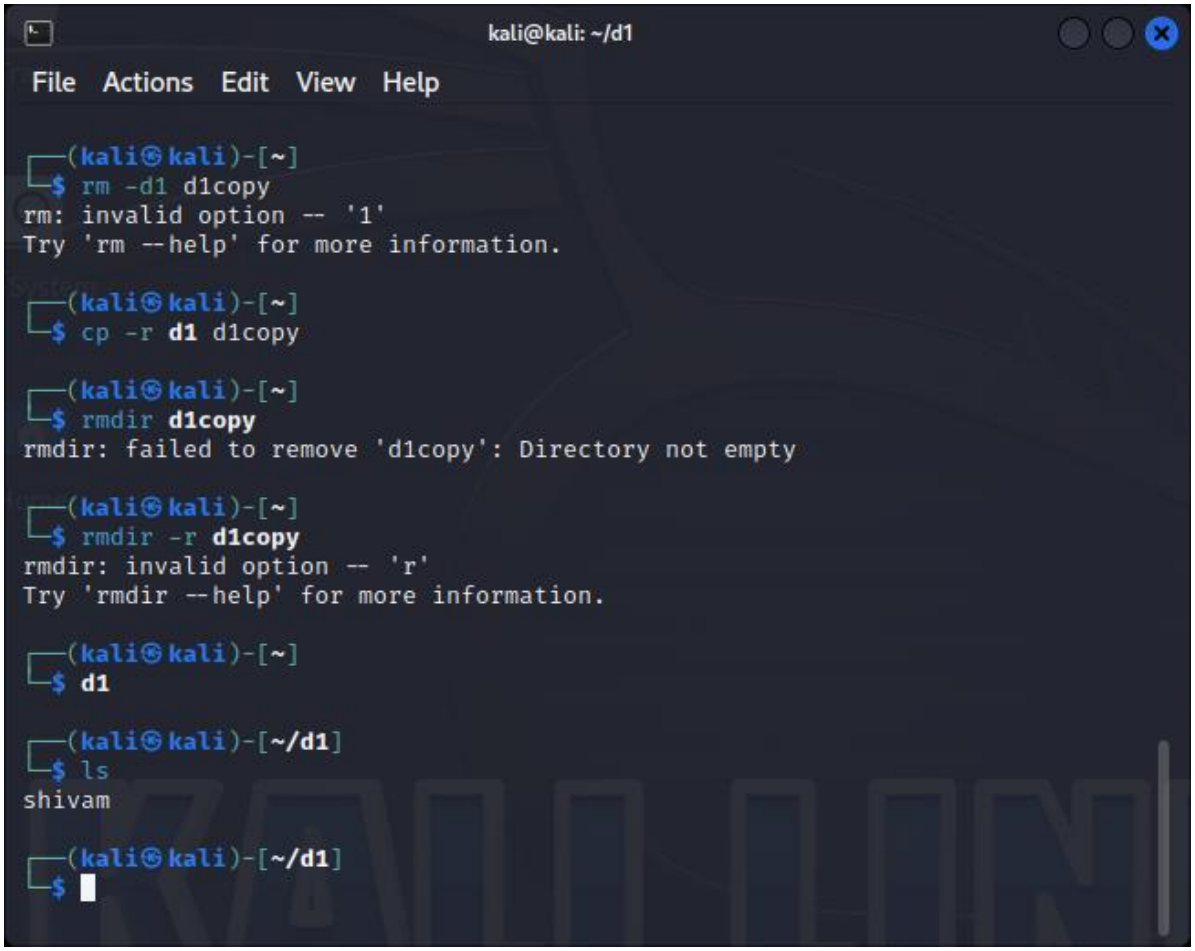
10.Listing a directory's contents

Code:

d1

ls

Output:



11) Using options on the ls command

Code:

Ls -F

Output:

```
kali@kali: ~/d1
File  Actions  Edit  View  Help
Try 'rm --help' for more information.

┌──(kali㉿kali)-[~]
└─$ cp -r d1 d1copy

┌──(kali㉿kali)-[~]
└─$ rmdir d1copy
rmdir: failed to remove 'd1copy': Directory not empty

┌──(kali㉿kali)-[~]
└─$ rmdir -r d1copy
rmdir: invalid option -- 'r'
Try 'rmdir --help' for more information.

┌──(kali㉿kali)-[~]
└─$ d1

┌──(kali㉿kali)-[~/d1]
└─$ ls
shivam

┌──(kali㉿kali)-[~/d1]
└─$ ls -F
shivam

┌──(kali㉿kali)-[~/d1]
└─$
```

12) Determining which directory you are currently in.

Code:

pwd

Output:

```
kali@kali: ~
File  Actions  Edit  View  Help
└─$ cp -r d1 d1copy

┌──(kali㉿kali)-[~]
└─$ rmdir d1copy
rmdir: failed to remove 'd1copy': Directory not empty

┌──(kali㉿kali)-[~]
└─$ rmdir -r d1copy
rmdir: invalid option -- 'r'
Try 'rmdir --help' for more information.

┌──(kali㉿kali)-[~]
└─$ d1

┌──(kali㉿kali)-[~/d1]
└─$ ls
shivam

┌──(kali㉿kali)-[~/d1]
└─$ ls -F
shivam

┌──(kali㉿kali)-[~/d1]
└─$ pwd
/home/kali/d1

┌──(kali㉿kali)-[~/d1]
```

13) Moving from one directory to another

Code:

```
cd
pwd
```

Output:

```
kali@kali: ~
File  Actions  Edit  View  Help
┌──(kali㉿kali)-[~]
└─$ rmdir -r d1copy
rmdir: invalid option -- 'r'
Try 'rmdir --help' for more information.

┌──(kali㉿kali)-[~]
└─$ d1

┌──(kali㉿kali)-[~/d1]
└─$ ls
shivam

┌──(kali㉿kali)-[~/d1]
└─$ ls -F
shivam

┌──(kali㉿kali)-[~/d1]
└─$ pwd
/home/kali/d1

┌──(kali㉿kali)-[~/d1]
└─$ cd

┌──(kali㉿kali)-[~]
└─$ pwd
/home/kali
```

14)Directory abbreviations for faster navigation.

Code:

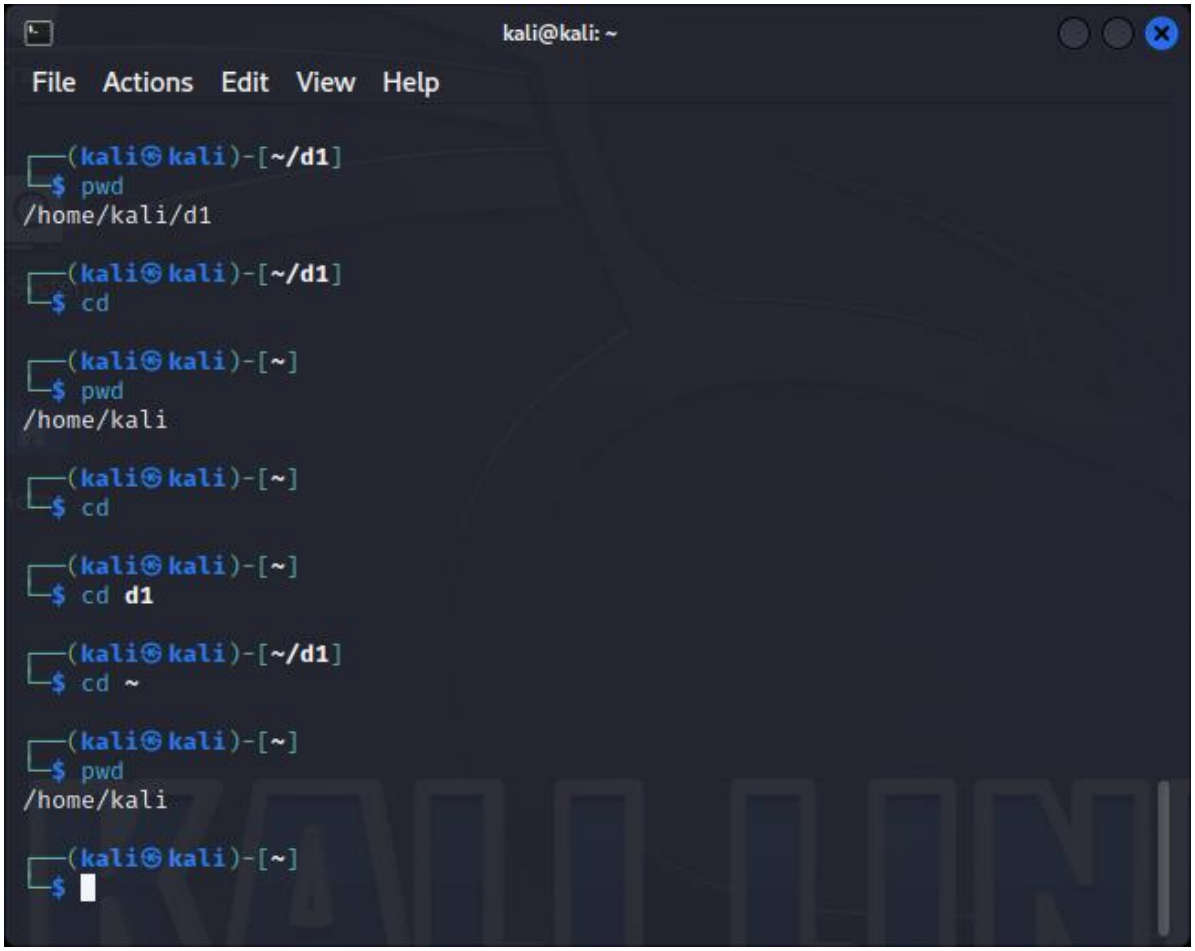
cd

cd d1

cd ~

pwd

Output:



15)Naming and deleting a variable

Code: read Name

“SHIVAM DAVE”

echo “Hello, $Name”

unset Name

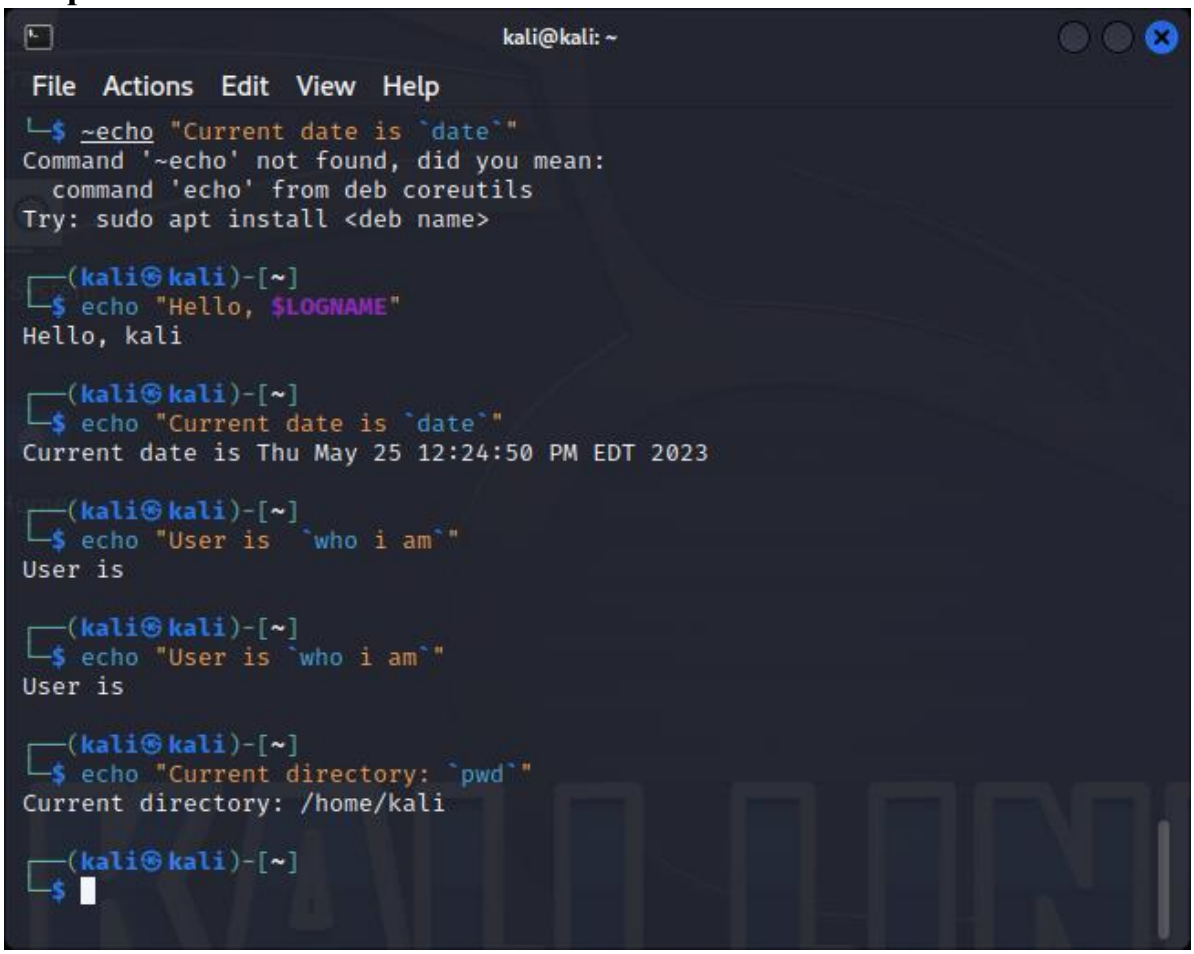
echo $Name

Output:


kali@kali: ~
File Actions Edit View Help
┌──(kali㉿kali)-[~]
└─$ read Name
SHIVAM DAVE
┌──(kali㉿kali)-[~]
└─$ echo "Hello, $Name"
Hello, SHIVAM DAVE
┌──(kali㉿kali)-[~]
└─$ unset Name
┌──(kali㉿kali)-[~]
└─$ echo "$Name"
┌──(kali㉿kali)-[~]
└─$

QUESTION– II – Shell Scripts

1. **Write Script to see current date, time, username, and current directory**
   Source code:
   echo "Hello, $LOGNAME"
   echo "Current date is `date`"
   echo "User is `who i am`"
   echo "Current direcotry `pwd`"
   Output:

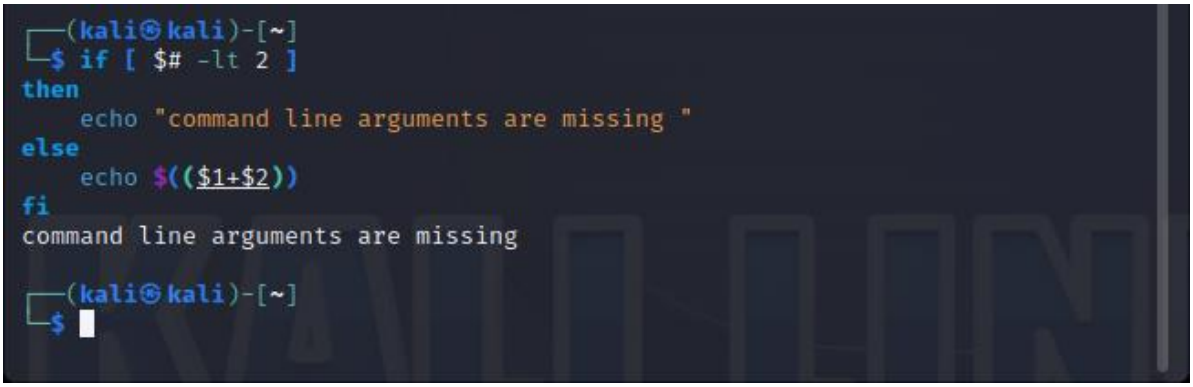


2. **How to write shell script that will add two numbers, which are supplied as command line argument, and if this two numbers are not given show error and its usage.**

Source Code:
if [$# -lt 2]
then

```
    echo "command line arguments are missing "
else
    echo $(($1+$2))
fi
```

Output:



3. **Write Script to find out biggest number from given three nos. Numbers are supplied as command line argument. Print error if sufficient arguments are not supplied.**

Source Code:

```
echo "Enter Num1"
read num1
echo "Enter Num2"
read num2
echo "Enter Num3"
read num3

if [ $num1 -gt $num2 ] && [ $num1 -gt $num3 ]
then
    echo $num1
elif [ $num2 -gt $num1 ] && [ $num2 -gt $num3 ]
then
    echo $num2
else
    echo $num3
fi
```

Output:

```
kali@kali: ~
File  Actions  Edit  View  Help
1

┌──(kali㉿kali)-[~]
└─$ echo "Enter Num1"
read num1
echo "Enter Num2"
read num2
echo "Enter Num3"
read num3

if [ $num1 -gt $num2 ] && [ $num1 -gt $num3 ]
then
    echo $num1
elif [ $num2 -gt $num1 ] && [ $num2 -gt $num3 ]
then
    echo $num2
else
    echo $num3
fi
Enter Num1
19
Enter Num2
30
Enter Num3
29
30
```

4. **Write script to print the following numbers as 5,4,3,2,1 using while loop.**

Source Code:

```
i=5
while test $i != 0
do
echo "$i
"
i=`expr $i - 1`
done
```

Output:

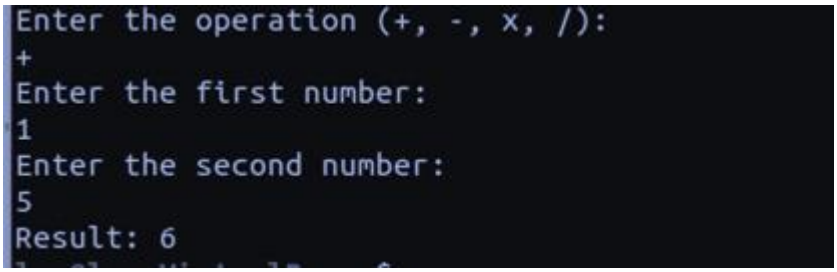
5. **Write Script, using case statement to perform basic math operation as follows: + addition, - subtraction, x multiplication, / division**

Source Code:

```
# Perform the math operation based on the operator
case $operator in
   "+")
      result=$((number1 + number2))
      echo "Result: $result"
      ;;
   "-")
      result=$((number1 - number2))
      echo "Result: $result"
      ;;
   "x")
      result=$((number1 * number2))
      echo "Result: $result"
      ;;
   "/")
      if [ $number2 -eq 0 ]; then
         echo "Error: Division by zero is not allowed."
```

```
        else
            result=$((number1 / number2))
            echo "Result: $result"
        fi
        ;;
    *)
        echo "Error: Invalid operator."
        ;;
Esac
```

OUTPUT:



Reg No : 21BCB0107
Name : Shivam Dave
Slot : B2

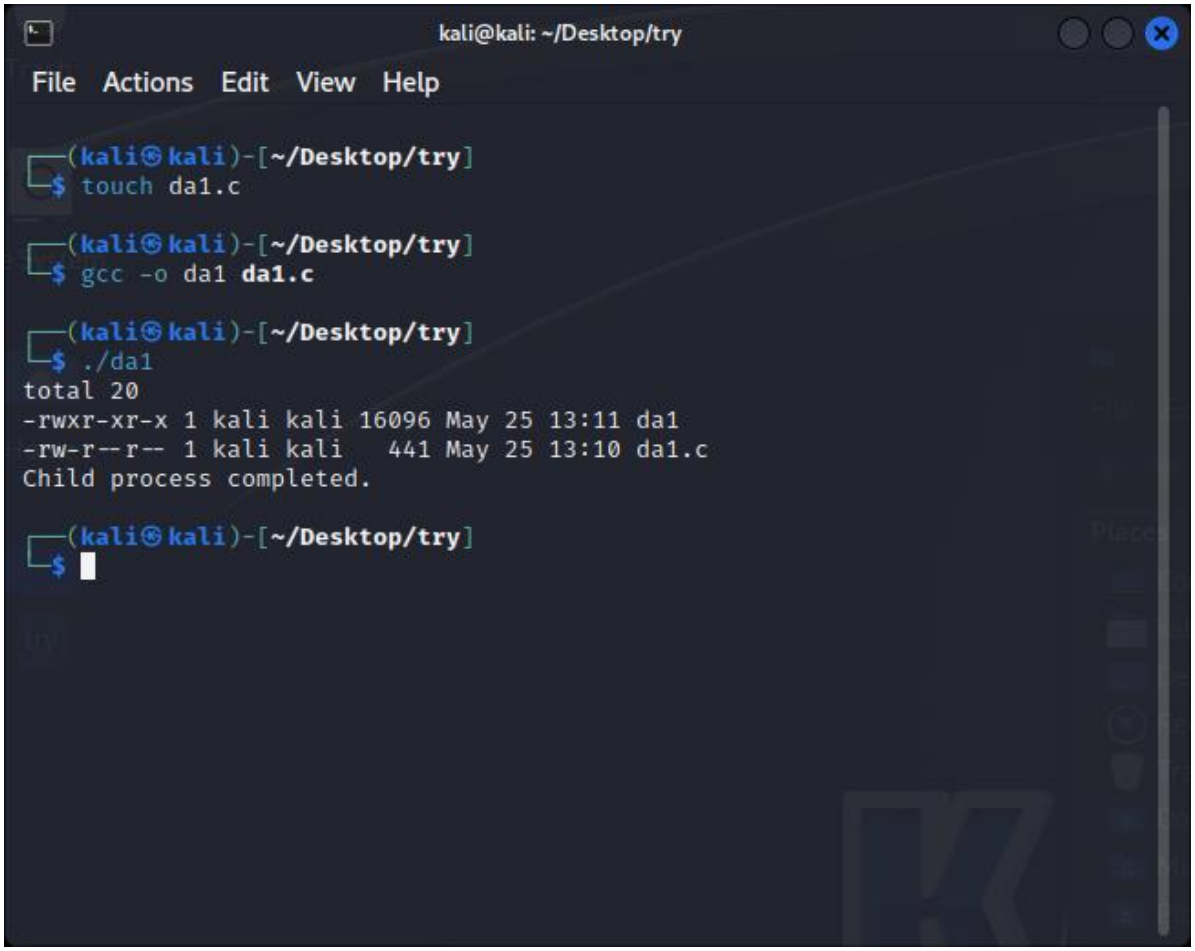
QUESTION 3 – System Calls

1. Write Programs using the following system calls of LINUX operating system:
    a. fork, exec, getpid, exit, wait, close, opendir, readdir,

Source Code:
Using 'fork' and 'exec':
Code:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int main() {
    pid_t pid = fork();

    if (pid == 0) {
        // Child process
        execl("/bin/ls", "ls", "-l", NULL);
    } else if (pid > 0) {
        // Parent process
        wait(NULL);
        printf("Child process completed.\n");
    } else {
        // Fork failed
        printf("Fork failed.\n");
        return 1;
    }

    return 0;
}
```

Output:

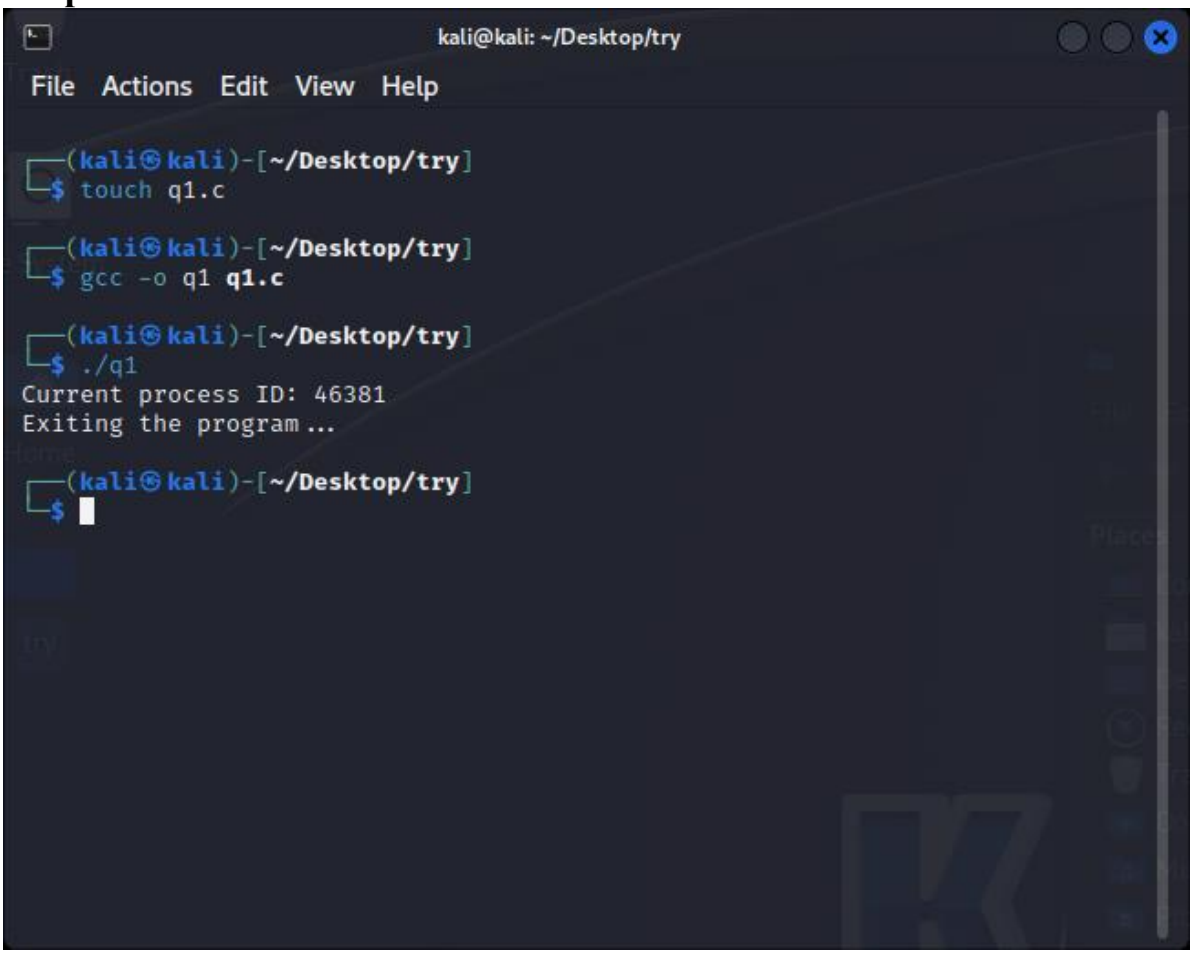


Using getpid and exit:

Code:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <stdlib.h>

int main() {
    pid_t pid = getpid();

    printf("Current process ID: %d\n", pid);
    printf("Exiting the program...\n");

    exit(0);
}
```

Output:



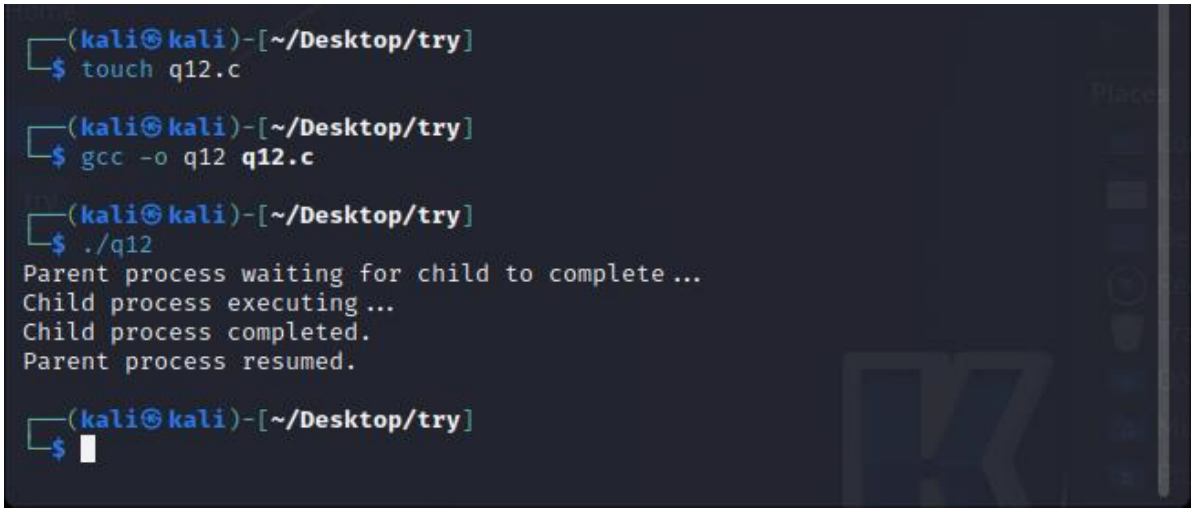
Using 'wait'

Source Code:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int main() {
    pid_t pid = fork();

    if (pid == 0) {
        // Child process
        printf("Child process executing...\n");
        sleep(2);
        printf("Child process completed.\n");
    } else if (pid > 0) {
        // Parent process
        printf("Parent process waiting for child to complete...\n");
        wait(NULL);
```

```
        printf("Parent process resumed.\n");
    } else {
        // Fork failed
        printf("Fork failed.\n");
        return 1;
    }

    return 0;
}
```

Output:



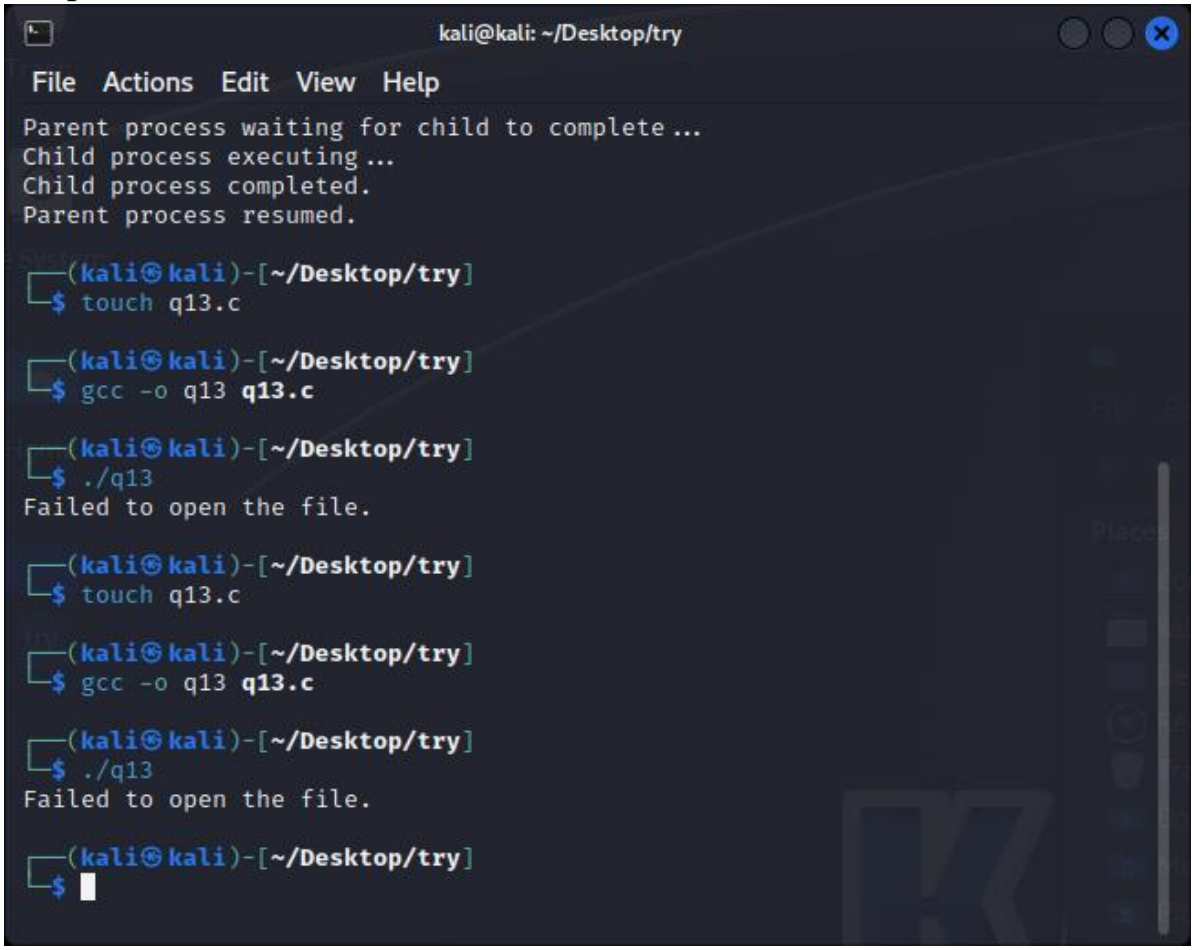
Using 'close'

Source Code:

```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>

int main() {
    int fd = open("file.txt", O_RDONLY);

    if (fd == -1) {
        printf("Failed to open the file.\n");
        return 1;
    }

    printf("File opened successfully.\n");

    if (close(fd) == -1) {
        printf("Failed to close the file.\n");
```

```
        return 1;
    }

    printf("File closed successfully.\n");

    return 0;
}
```

Output:



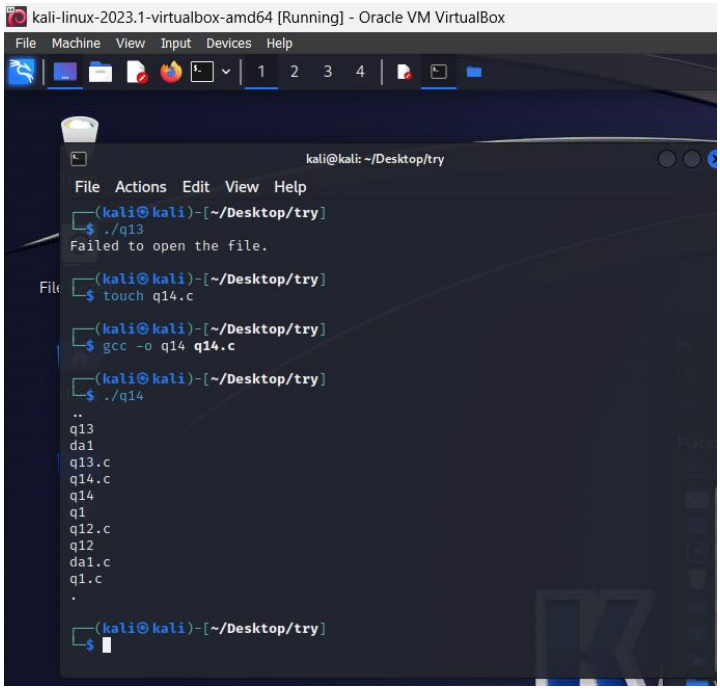
Using 'opendir' and 'readdir':

Source Code:

```
#include <stdio.h>
#include <dirent.h>

int main() {
    DIR *dir = opendir(".");

    if (dir == NULL) {
        printf("Failed to open directory.\n");
        return 1;
```

```
    }

    struct dirent *entry;
    while ((entry = readdir(dir)) != NULL) {
        printf("%s\n", entry->d_name);
    }

    closedir(dir);

    return 0;
}
```

Output:



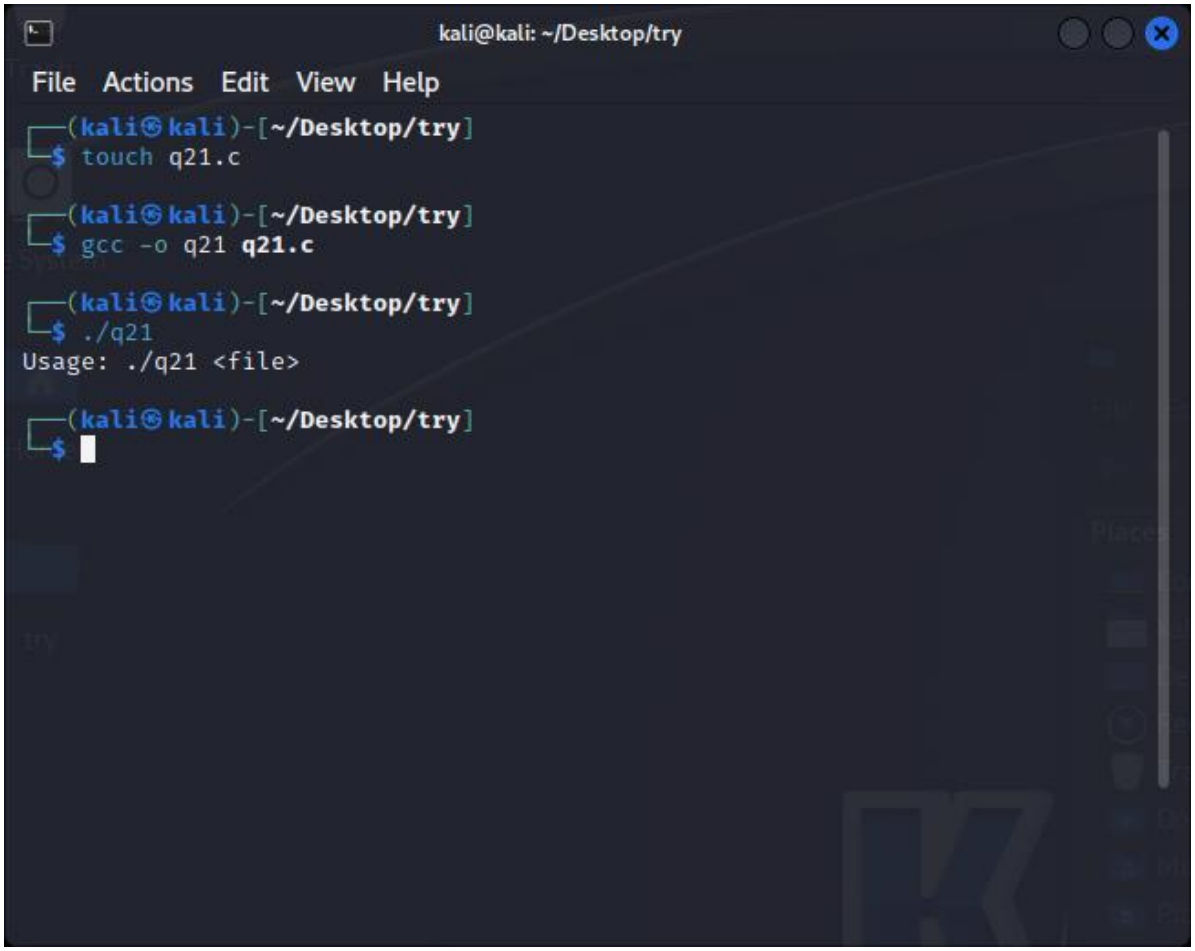
2. **Write Programs using I/O system calls of LINUX operating system: open, read, write etc.**

Source Code:

```
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>

#define BUF_SIZE 4096

int main(int argc, char *argv[]) {
    if (argc != 2) {
        printf("Usage: %s <file>\n", argv[0]);
```

```
        return 1;
    }

    int fd = open(argv[1], O_RDONLY);
    if (fd == -1) {
        perror("Failed to open file");
        return 1;
    }

    char buffer[BUF_SIZE];
    ssize_t bytes_read;

    while ((bytes_read = read(fd, buffer, BUF_SIZE)) > 0) {
        // Process or display the read data
        write(STDOUT_FILENO, buffer, bytes_read);
    }

    if (bytes_read == -1) {
        perror("Read error");
        return 1;
    }

    close(fd);

    return 0;
}
```

Output:



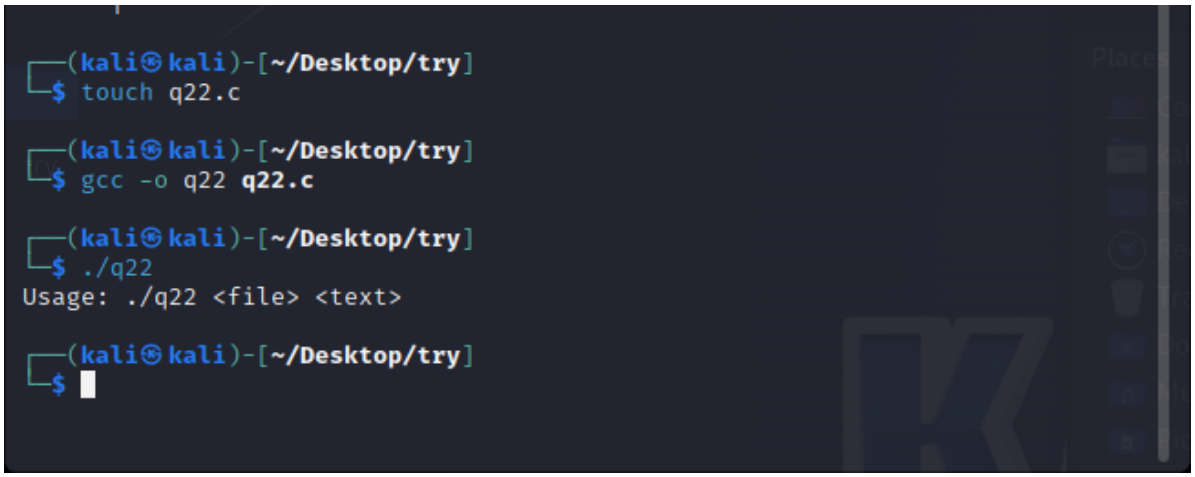
b) write()

Source Code:

```
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[]) {
    if (argc != 3) {
        printf("Usage: %s <file> <text>\n", argv[0]);
        return 1;
    }

    int fd = open(argv[1], O_WRONLY | O_CREAT | O_TRUNC, 0666);
    if (fd == -1) {
        perror("Failed to open file");
        return 1;
    }

    const char *text = argv[2];
```

```
    ssize_t bytes_written = write(fd, text, strlen(text));

    if (bytes_written == -1) {
        perror("Write error");
        return 1;
    }

    close(fd);

    return 0;
}
```

Output:



3. **Write Programs using C to simulate LINUX commands**
    a. **ls command,**

Source code:

```
#include <stdio.h>
#include <dirent.h>

int main() {
    struct dirent *entry;
    DIR *directory = opendir(".");

    if (directory == NULL) {
        printf("Error opening directory.\n");
        return 1;
    }

    while ((entry = readdir(directory)) != NULL) {
        printf("%s\n", entry->d_name);
    }
```

```
    closedir(directory);
    return 0;
}
```

Output:

b. **grep command**

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_LINE_LENGTH 1000

int main(int argc, char *argv[]) {
    if (argc != 3) {
        printf("Usage: grep <pattern> <filename>\n");
        return 1;
    }

    char *pattern = argv[1];
    char *filename = argv[2];

    FILE *file = fopen(filename, "r");
    if (file == NULL) {
        printf("Error opening file: %s\n", filename);
        return 1;
    }

    char line[MAX_LINE_LENGTH];
    while (fgets(line, MAX_LINE_LENGTH, file) != NULL) {
```

```
        if (strstr(line, pattern) != NULL) {
            printf("%s", line);
        }
    }

    fclose(file);
    return 0;
}
```

Output:

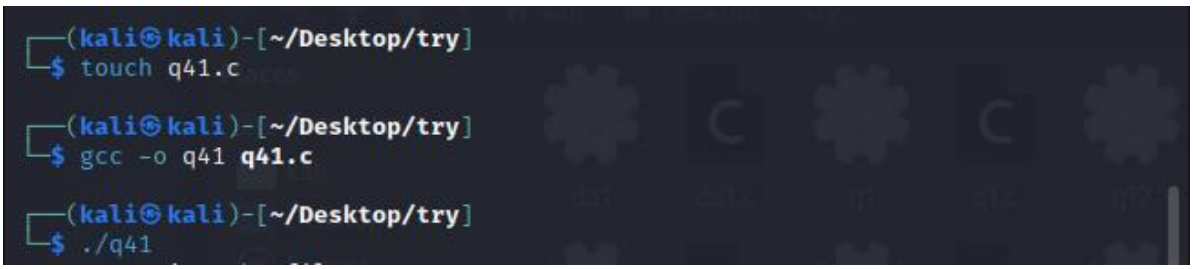
4. **Create a file with few lines, write a C program to read the file and delete the spaces more than one in the file.**

Source Code:

```
#include <stdio.h>
#include <stdlib.h>

void removeExtraSpaces(char* filename) {
    FILE* file = fopen(filename, "r");
    if (file == NULL) {
        printf("Error opening the file.\n");
        return;
    }

    char tempFilename[] = "temp.txt";
    FILE* tempFile = fopen(tempFilename, "w");
    if (tempFile == NULL) {
        printf("Error creating temporary file.\n");
        fclose(file);
        return;
    }

    int prevSpace = 0; // flag to track previous space
    int currentChar;
    while ((currentChar = fgetc(file)) != EOF) {
```

```
        if (currentChar == ' ') {
            if (!prevSpace) {
                fputc(currentChar, tempFile); // write the first space
            }
            prevSpace = 1;
        } else {
            fputc(currentChar, tempFile); // write non-space characters
            prevSpace = 0;
        }
    }

    fclose(file);
    fclose(tempFile);

    if (remove(filename) != 0) {
        printf("Error deleting the original file.\n");
        return;
    }

    if (rename(tempFilename, filename) != 0) {
        printf("Error renaming the temporary file.\n");
        return;
    }

    printf("Spaces removed successfully.\n");
}

int main() {
    char filename[] = "example.txt";
    removeExtraSpaces(filename);
    return 0;
}
```

Output:



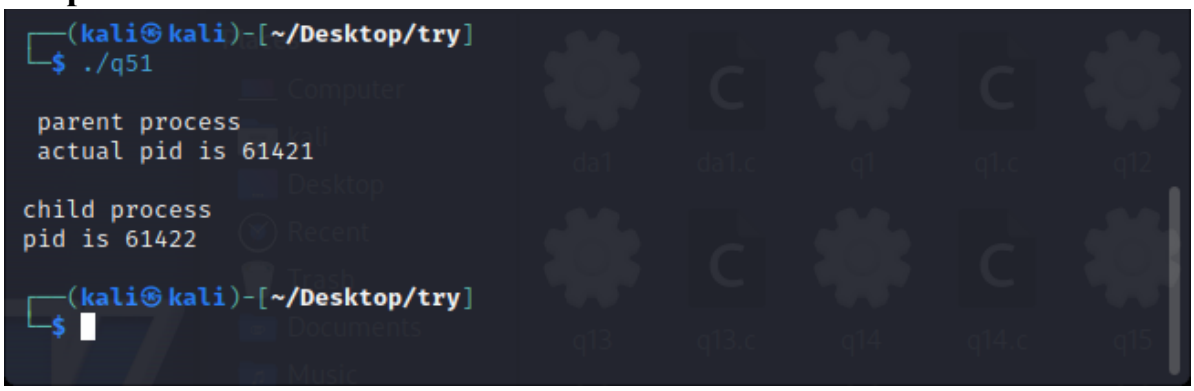
5. **Write a program:**

a. **To create parent & child process and print their id.**

Source Code:

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>

int main()
{
   int pid;
   pid=fork();
   if(pid<0)
   {
      printf("\n Error ");
      exit(1);
   }
   else if(pid==0)
   {
      printf("\nchild process ");
      printf("\npid is %d ",getpid());
      exit(0);
   }
   else
   {
      printf("\n parent process ");
      printf("\n actual pid is %d \n ",getpid());
      exit(1);
   }

}
```

Output:

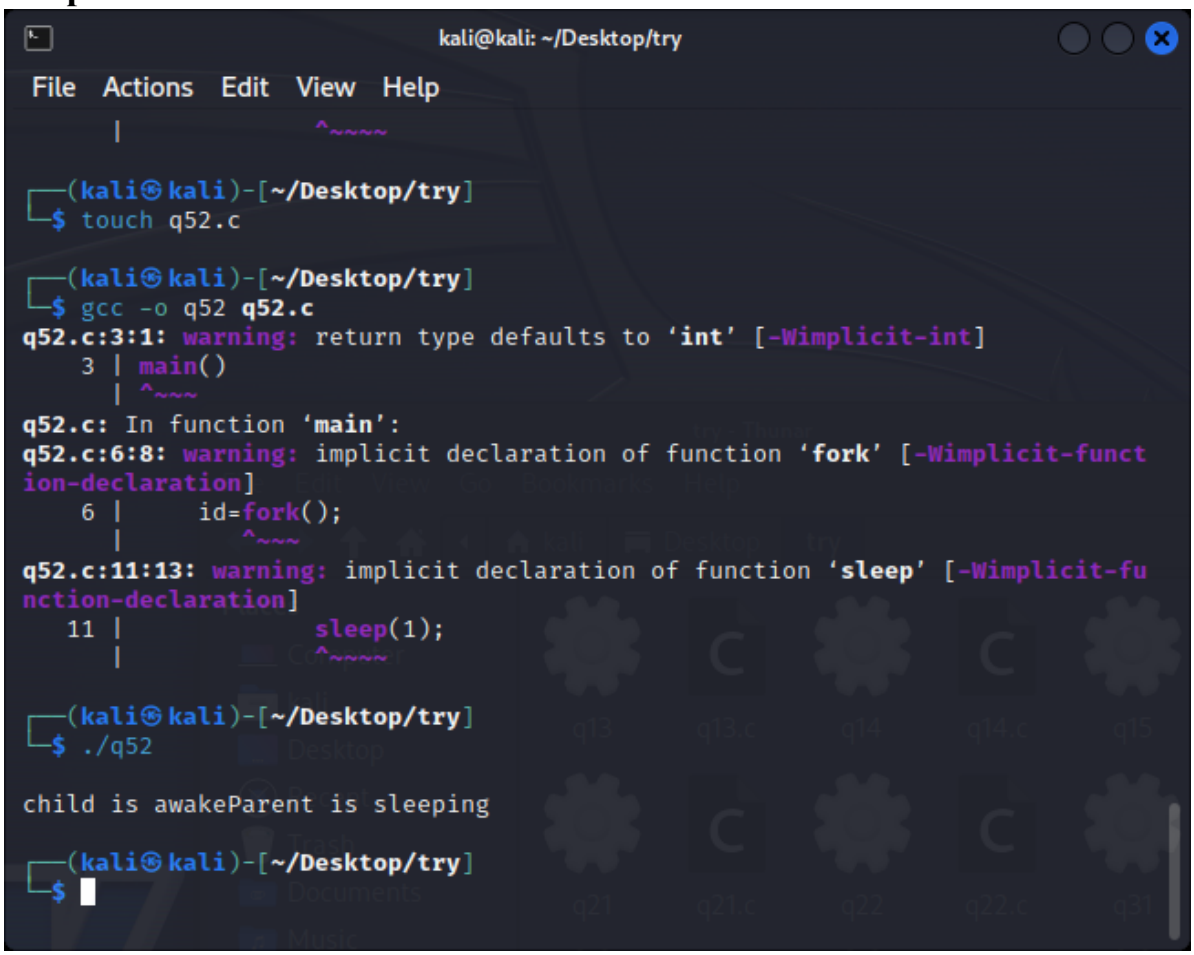


b. **To create a zombie process.**

Source Code:

```
#include<stdio.h>

main()
{
   int id;
   id=fork();

   if(id>0)
   {
      printf("\nParent is sleeping");
         sleep(1);
   }
   if(id==0)
      printf("child is awake");
}
```

Output:



c. **To create orphan process.**

Source Code:

```
#include<stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
   // Create a child process
   int pid = fork();

   if (pid > 0)
            printf("in parent process");

   // Note that pid is 0 in child process
   // and negative if fork() fails
   else if (pid == 0)
   {
            sleep(30);
            printf("in child process");
   }

   return 0;
}
```

Output:

d. **To make the process to sleep for few seconds.**

Source Code:

```
#include<stdio.h>
```

```
#include<unistd.h>
#include<sys/types.h>
#include<sys/wait.h>

int main(void){
pid_t f;

f = fork();

if(f < 0){
printf("Failed to fork\n");
_exit(1);
}

else if(f == 0){
int i;

printf("\nChild: PID is %d\n", getpid());
for(i = 0; i < 10; i++){
printf("c ");
if(i == 5)
sleep(2);
}
printf("\n");

_exit(0);
}

else{
int j;

printf("\nParent: PID is %d\n", getpid());
for(j = 0; j < 10; j++){
printf("p ");
}
printf("\n");
}

return 0;
}
```

Output:

```
┌──(kali㉿kali)-[~/Desktop/try]
└─$ gcc -o q54 q54.c

┌──(kali㉿kali)-[~/Desktop/try]
└─$ ./q54

Parent: PID is 64334
p p p p p p p p p p

Child: PID is 64335
```

e.Implement the program to pass messages using pipes.

Source Code:

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>

int main()
{
   int pid;
   pid=fork();
   if(pid<0)
   {
      printf("\n Error ");
      exit(1);
   }
   else if(pid==0)
   {
      printf("\nchild process ");
      printf("\npid is %d ",getpid());
      exit(0);
   }
   else
   {
      printf("\n parent process ");
      printf("\n actual pid is %d \n ",getpid());
      exit(1);
   }

}
```

Output:

```
kali@kali: ~/Desktop/try
File  Actions  Edit  View  Help

Parent: PID is 64334
p p p p p p p p p p

Child: PID is 64335

┌──(kali㉿kali)-[~/Desktop/try]
└─$ c c c c c c c c c c


┌──(kali㉿kali)-[~/Desktop/try]
└─$ touch q55.c

┌──(kali㉿kali)-[~/Desktop/try]
└─$ gcc -o q55 q55.c

┌──(kali㉿kali)-[~/Desktop/try]
└─$ ./q55


 parent process
child process
 actual pid is 65125
pid is 65126

┌──(kali㉿kali)-[~/Desktop/try]
└─$
```