EXPERIMENT 10

NAME: Rahil Sharma
PRN: 18070123062
BATCH: EA-3

SUBJECT: ESRTOS

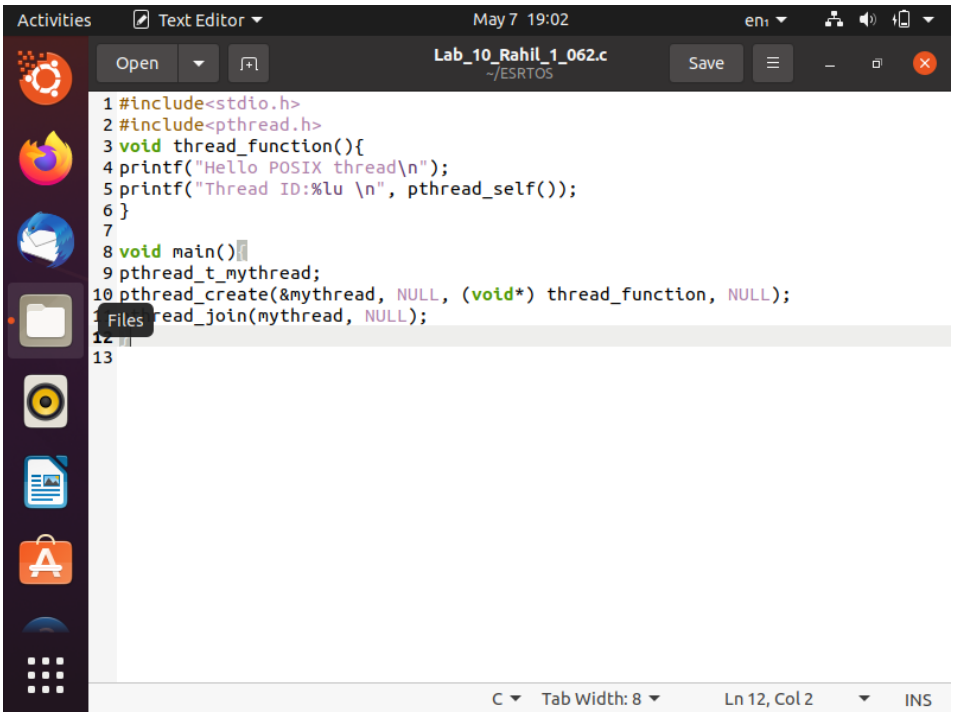
AIM: Write a C program to explain POSIX thread

i. Code 1:

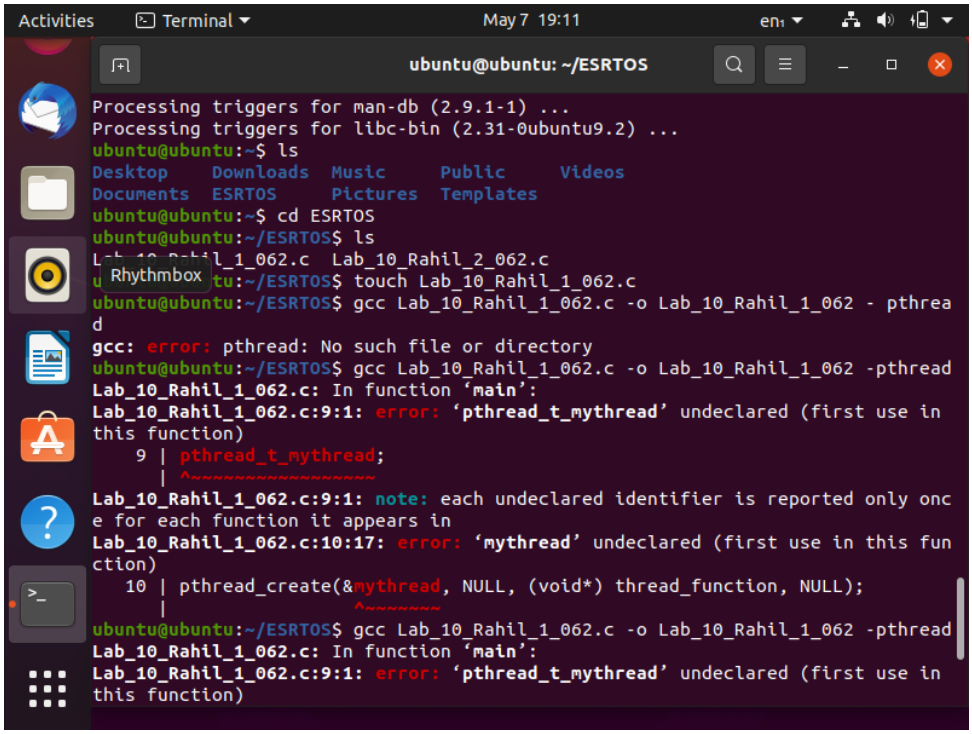
Code of the Program:

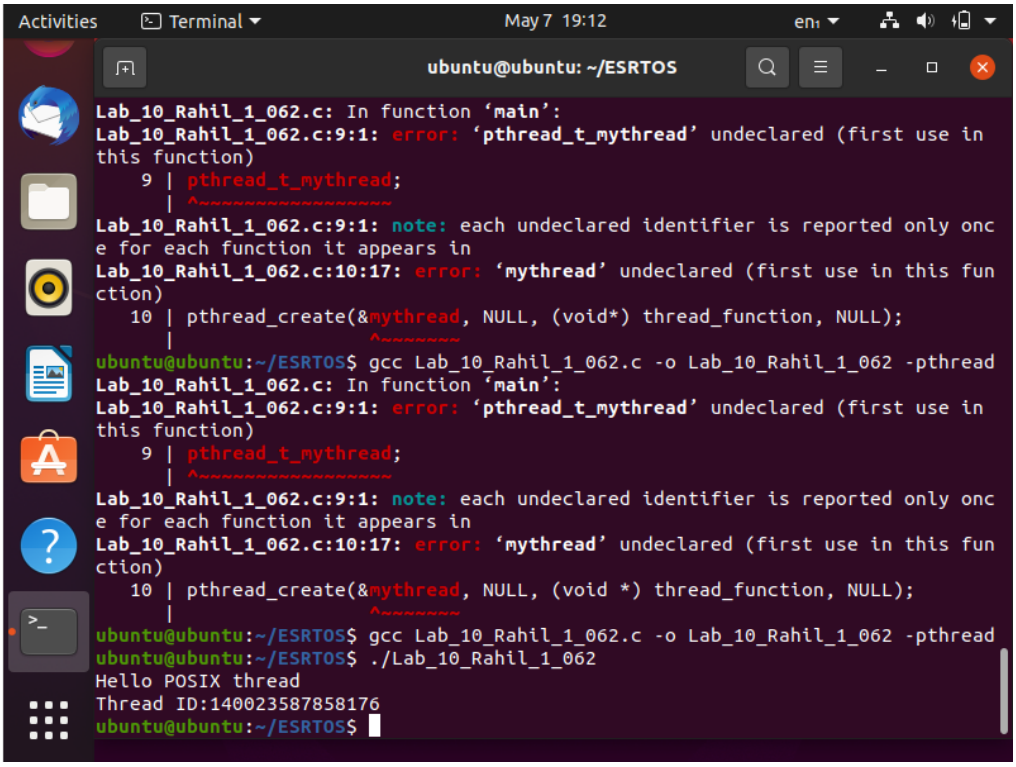
```
#include<stdio.h>
#include<pthread.h>
void thread_function(){
printf("Hello POSIX thread\n");
printf("Thread ID: %lu \n", pthread_self());
}
void main(){
pthread_t mythread;
pthread_create(&mythread, NULL, (void *) thread_function, NULL);
pthread_join(mythread, NULL);
}
```

Screenshot of the Program:



Output of the Program:



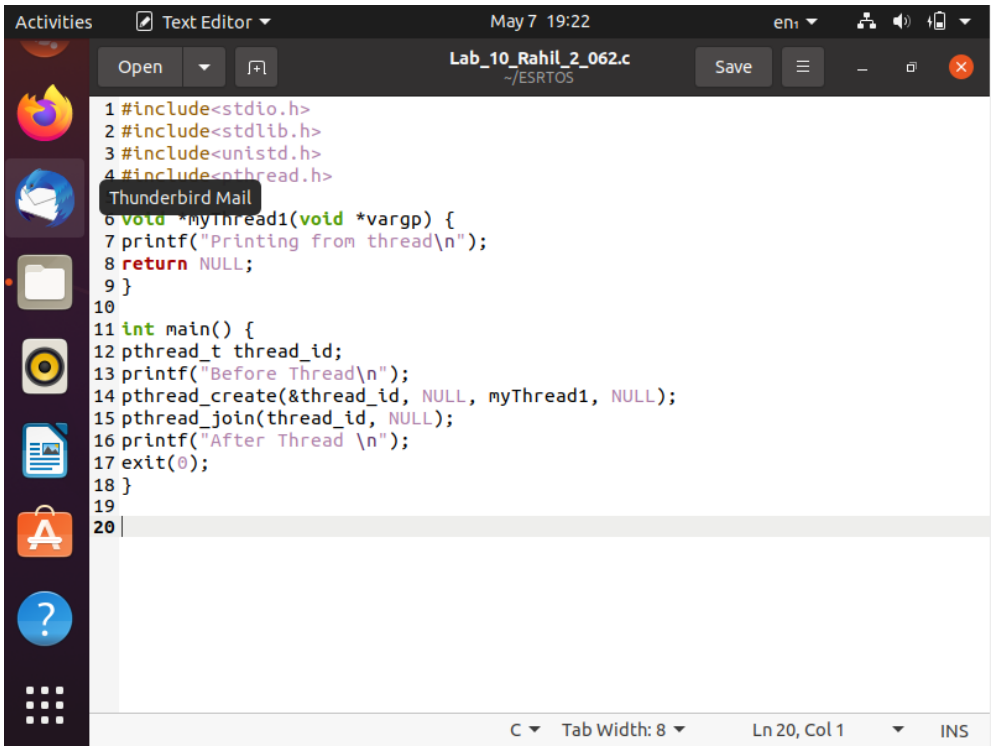


ii. Code 2:

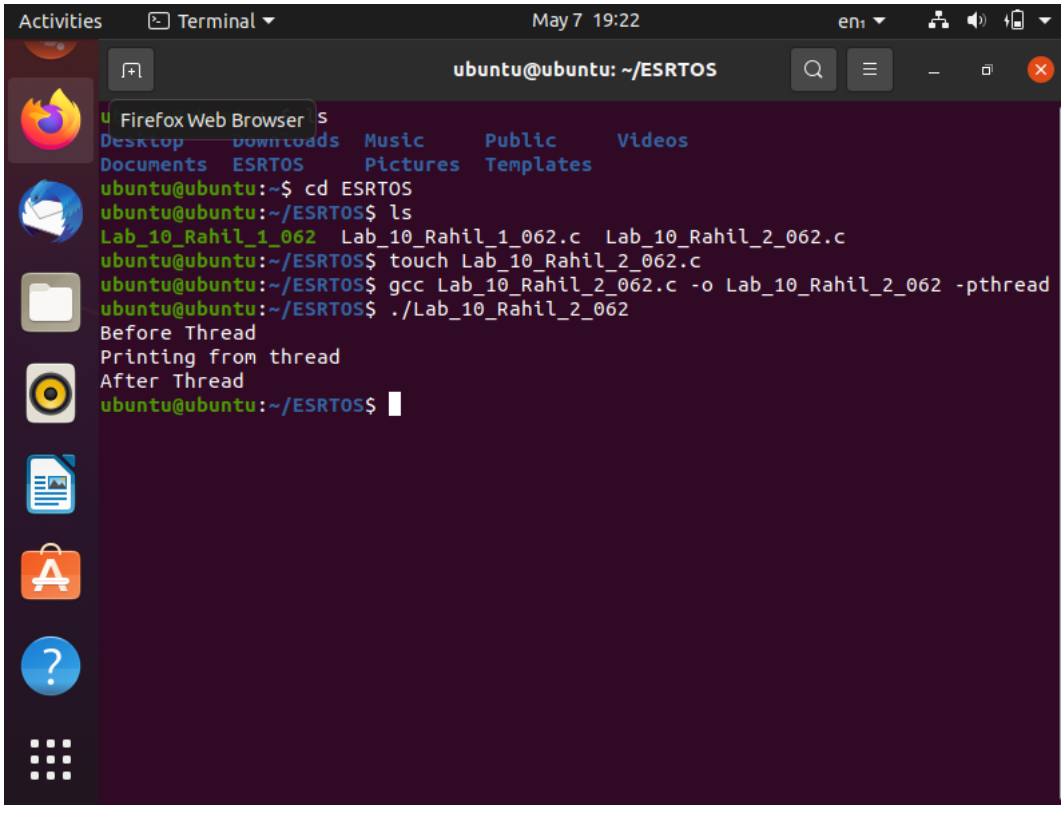
Code of the Program:

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

void *myThread1(void *vargp){

printf("Printing from thread\n");
return NULL;
}
int main(){
pthread_t thread_id;
printf("Before Thread\n");
pthread_create(&thread_id, NULL, myThread1,
NULL); pthread_join(thread_id, NULL);
printf("After Thread \n");
exit(0);
}
```

Screenshot of the Program:



Output of the Program:

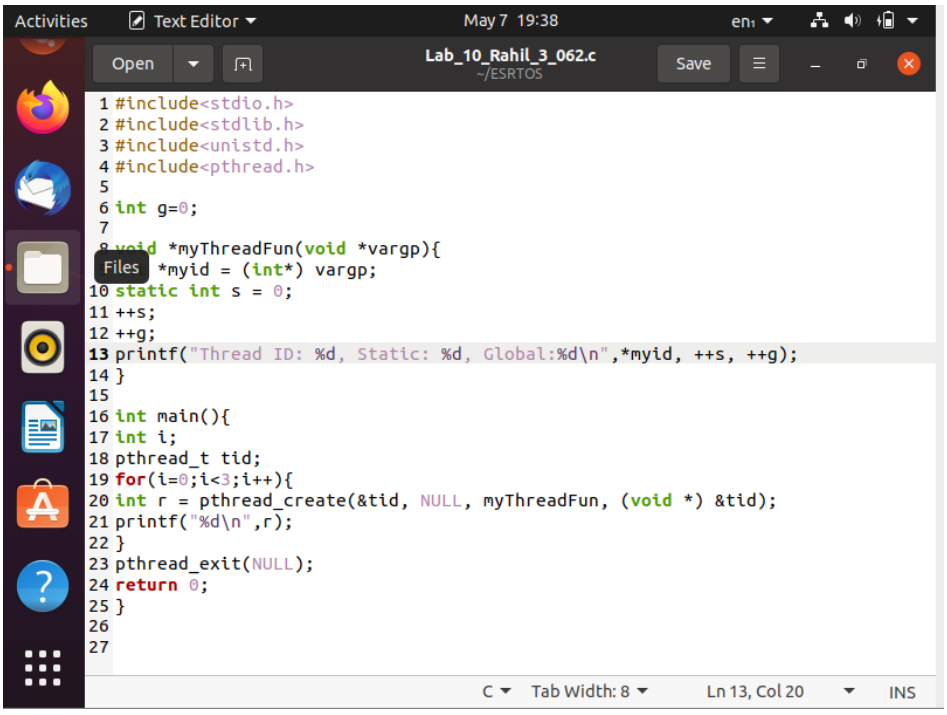


iii. Code 3

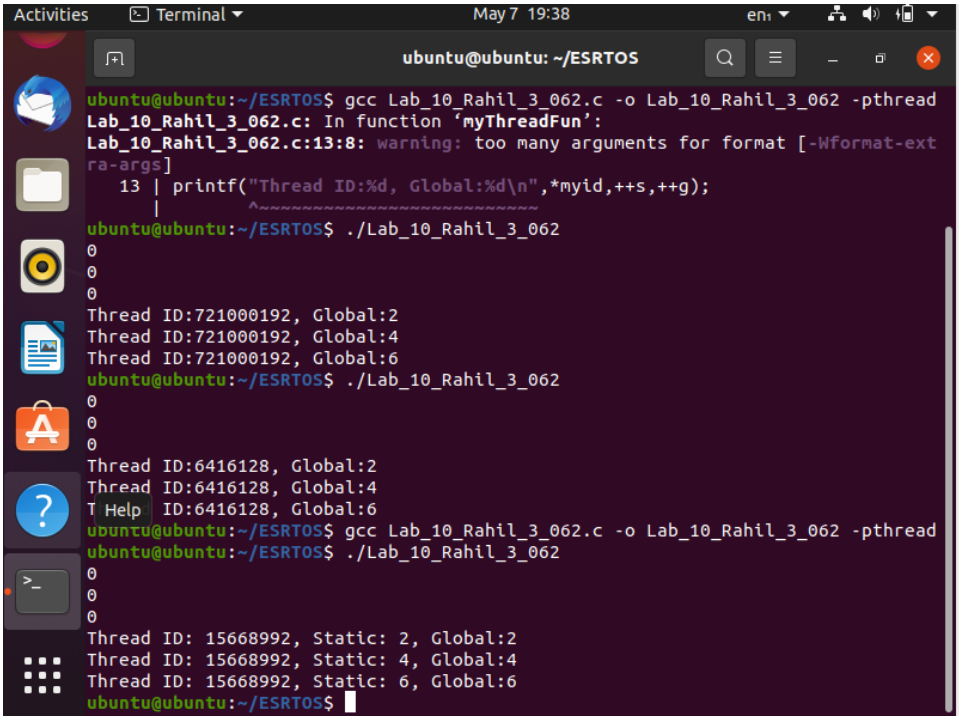
Code of the Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<pthread.h>
int g=0;
void *myThreadFun(void *vargp){
int *myid = (int*) vargp;
static int s = 0;
++s;
++g;
printf("Thread ID: %d, Static: %d, Global: %d\n", *myid, ++s, ++g); }

int main(){

int i;
pthread_t tid;
for (i=0;i<3;i++){
int r = pthread_create(&tid, NULL, myThreadFun, (void *) &tid);
printf("%d\n",r);
}
pthread_exit(NULL);
return 0;
}
```

Screenshot of the Program:



Output of the Program:



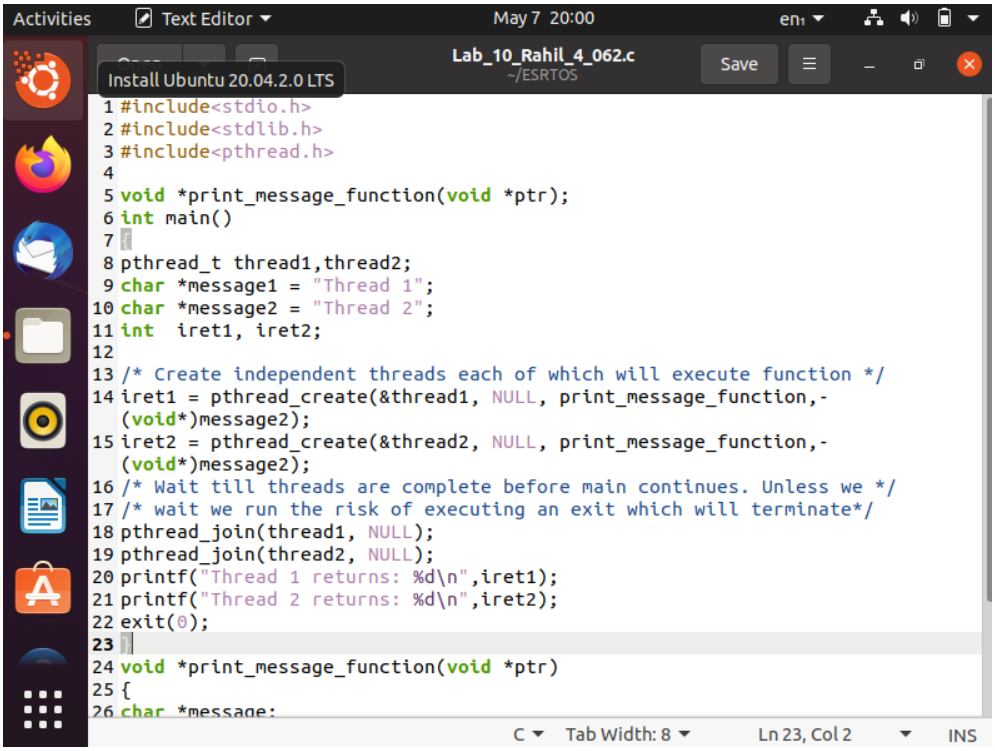
iv. CODE 4:

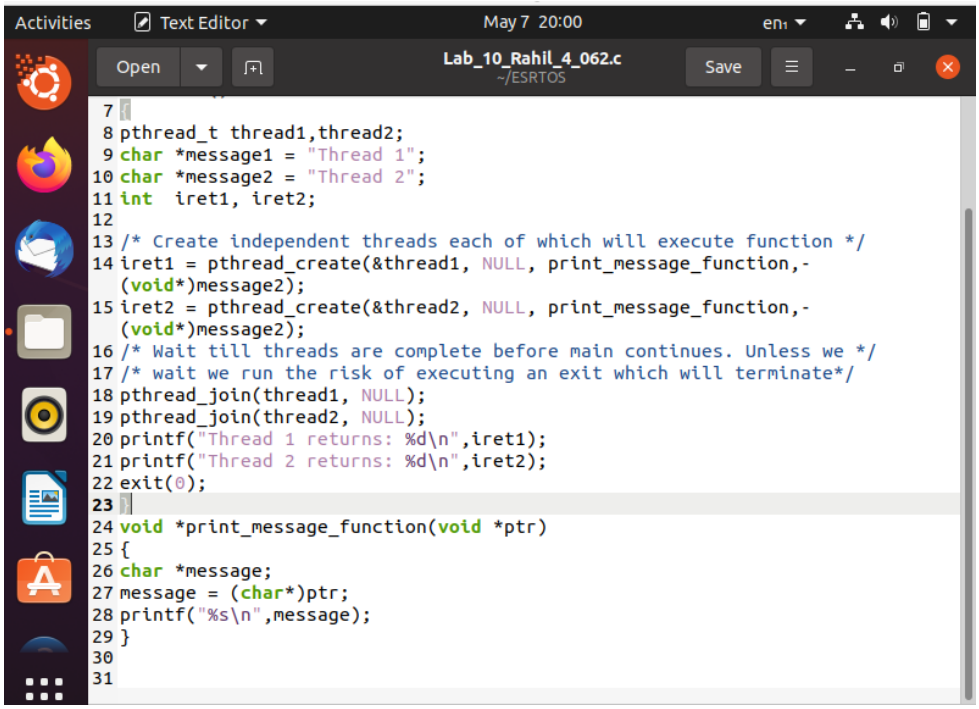
Code of the Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
void *print_message_function( void *ptr );
int main()
{
pthread_t thread1, thread2;
char *message1 = "Thread 1";
char *message2 = "Thread 2";
int iret1, iret2;
/* Create independent threads each of which will execute function */ iret1 =
pthread_create( &thread1, NULL, print_message_function, (void*)  message1);
iret2 = pthread_create( &thread2, NULL, print_message_function,
(void*)  message2);
/* Wait till threads are complete before main continues. Unless we */ /* wait we run the
risk of executing an exit which will terminate */ /* the process and all threads before the
threads have completed. The */  //pthread_join() function provides a simple mechanism
allowing an  application //to wait for a thread to terminate. After the thread
terminates,  the //application may then choose to clean up resources that were used
by  the thread.
pthread_join( thread1, NULL);
pthread_join( thread2, NULL);
printf("Thread 1 returns: %d\n",iret1);
printf("Thread 2 returns: %d\n",iret2);
exit(0);
}
```

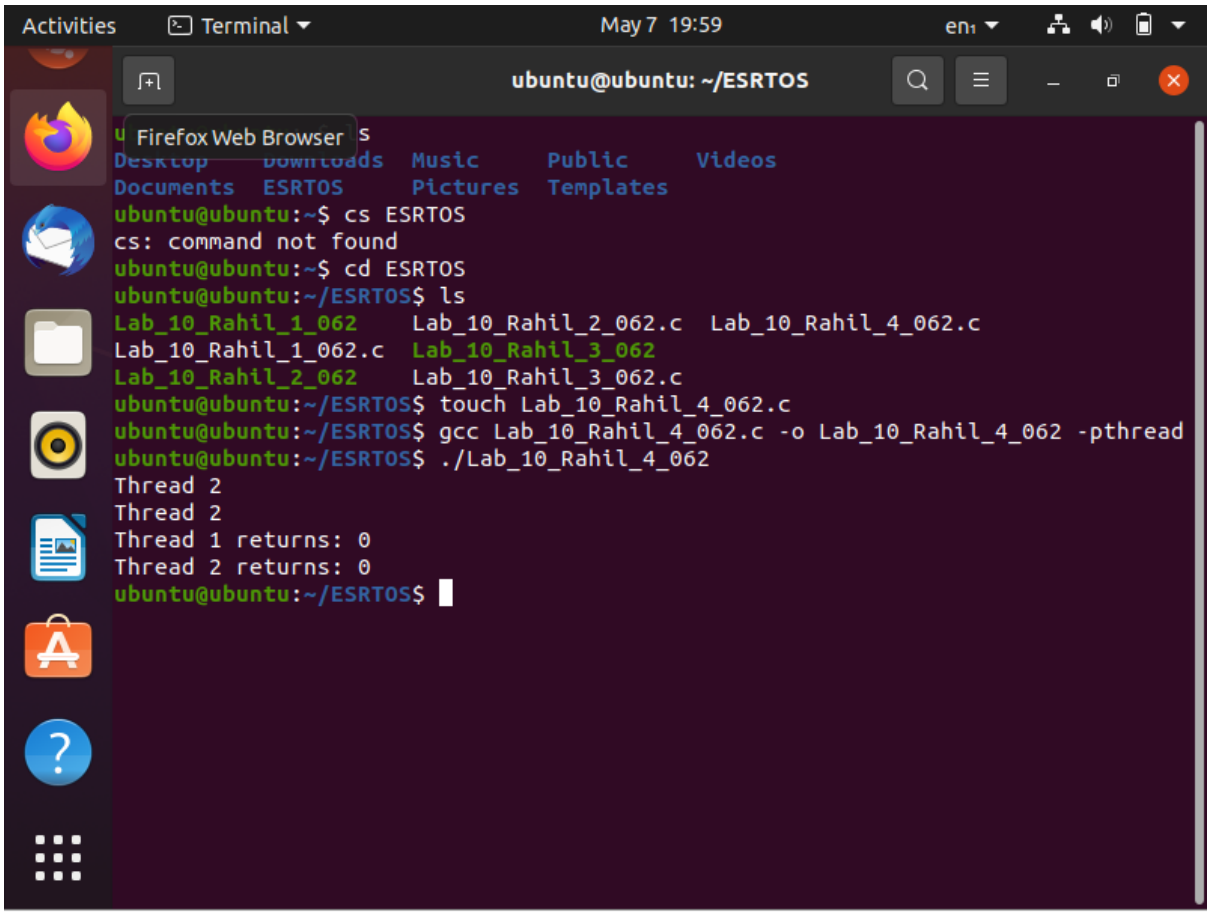
```
void *print_message_function( void *ptr )
{
char *message;
message = (char *) ptr;
printf("%s \n", message);
}
```

Screenshot of the Program:





Output of the Program:



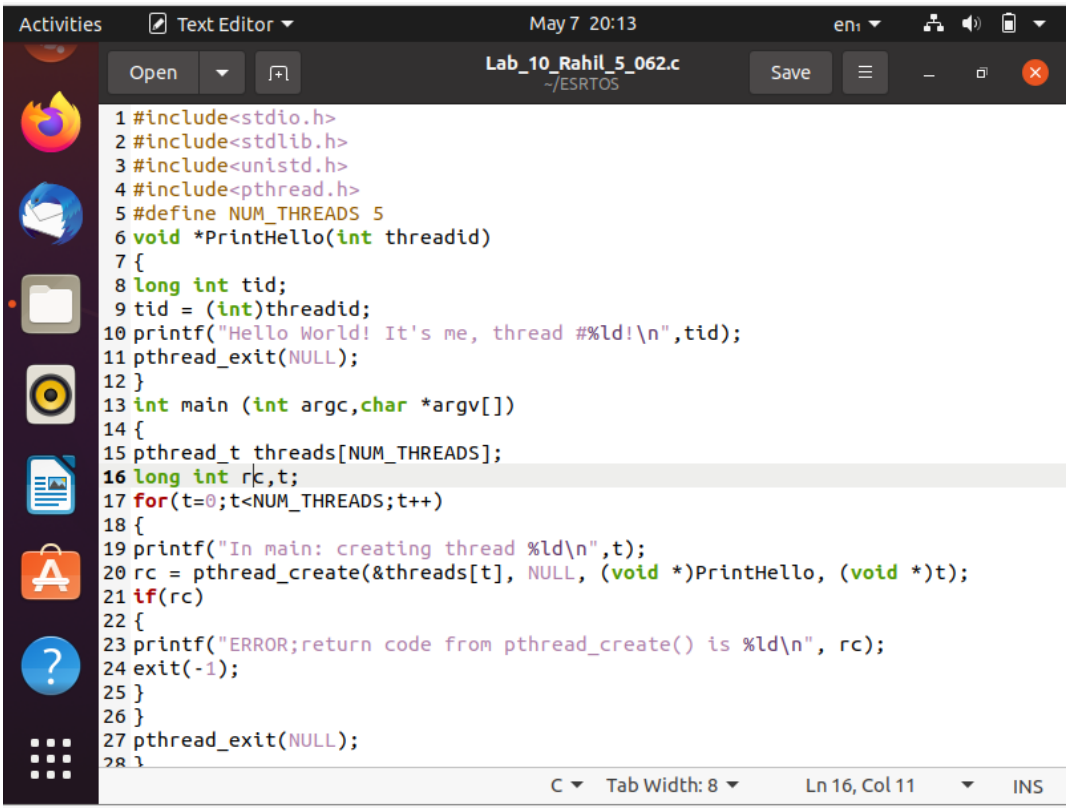
v. CODE 5:

Code of the Program:

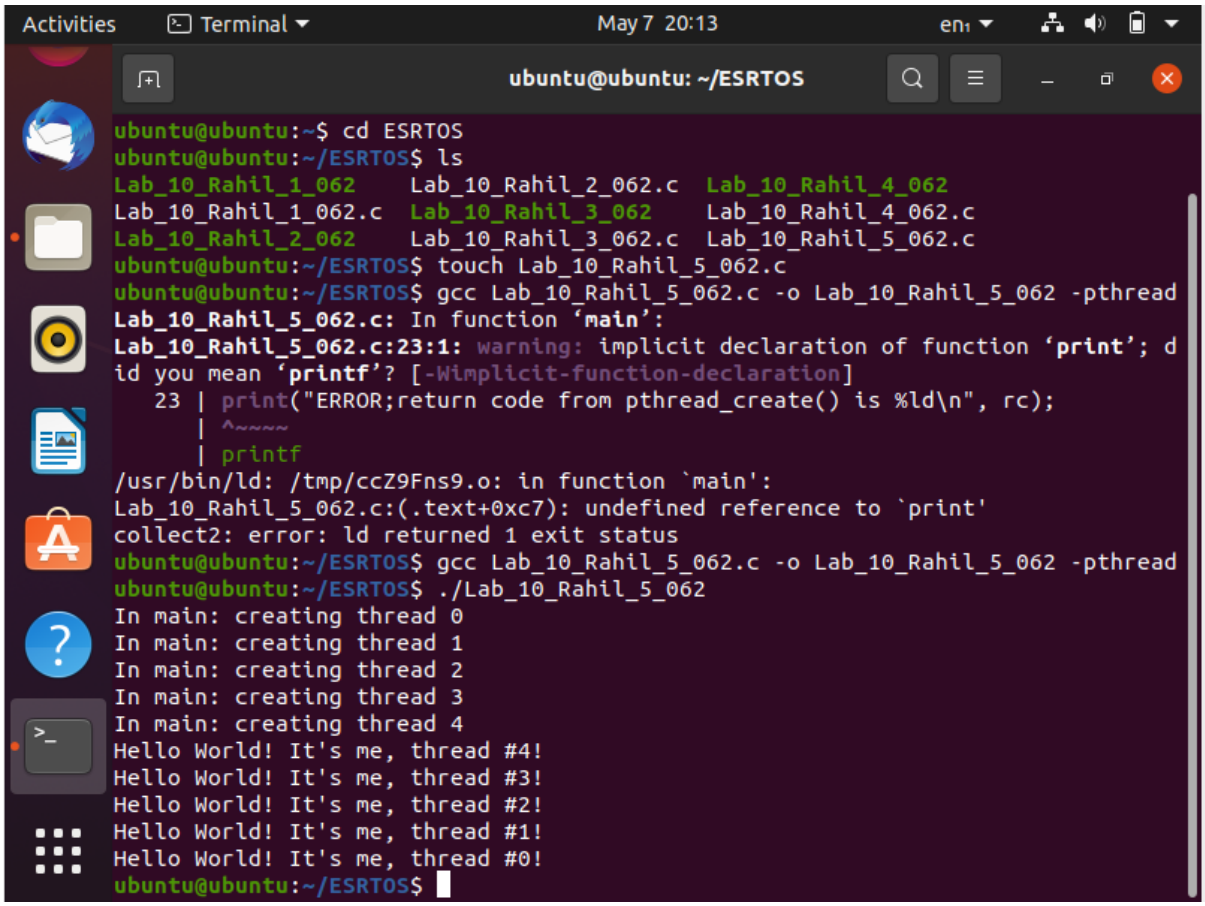
```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<pthread.h>
#define NUM_THREADS 5
void *PrintHello(int threadid)
{
long int tid;
tid= (int)threadid;
printf("Hello World! It's me, thread #%ld!\n", tid);
pthread_exit(NULL);
}
int main (int argc, char *argv[])
{
pthread_t threads[NUM_THREADS];
long int rc, t;
for(t=0; t<NUM_THREADS; t++)
{
printf("In main: creating thread %ld\n", t);
```

```
rc= pthread_create(&threads[t],NULL, (void *)PrintHello, (void *)t);
if (rc)
{
printf("ERROR; return code from pthread_create() is %ld\n", rc);
exit(-1);
}
}
pthread_exit(NULL);
}
```

Screenshot of the Program:



Output of the Program:



CONCLUSION: In this experiment we have learnt about POSIX thread in C and Linux.