**OS\_LAB\_05\_Assignment**

**CE\_054**

**Aim :-** Thread creation and Termination. Synchronization using mutex lock and unlock. (Use

of pthread\_create, ptread\_join, pthread\_mutex\_lock and pthread\_mutex\_unlock

library functions of pthread library).

* **Explaination of Functions** :-

1. **pthread\_create**:-
   * Thread creation.
   * Syntax :-

#include <pthread.h>

int pthread\_create(pthread\_t \*thread, const pthread\_attr\_t \*attr,

void \*(\*start\_routine) (void \*), void \*arg);

* + Description :-

The pthread\_create() function shall create a new thread, with attributes specified by attr, within a process. If attr is NULL, the default attributes shall be used.If the attributes specified by attr are modified later, the thread's attributes shall not be affected.

Upon successful completion, pthread\_create() shall store the ID of the created thread in the location referenced by thread. If pthread\_create() fails, no new thread is created and the contents of the location referenced by thread are undefined.

On success, pthread\_create() returns 0; on error, it returns an error number, and the contents of thread are undefined.

1. **pthread\_join:-**
   * wait for thread termination
   * Syntax :-

#include <pthread.h>

int pthread\_join(pthread\_t thread, void \*\*value\_ptr);

* + Description :-

The pthread\_join() function suspends execution of the calling thread until the target thread terminates, unless the target thread has already terminated.

If status is non-NULL, the value passed to [pthread\_exit()](https://www.mkssoftware.com/docs/man3/pthread_exit.3.asp) by the terminated thread is stored in the location pointed to by *status*.

When a pthread\_join() function returns successfully, the target thread has been terminated. The result of multiple simultaneous calls to pthread\_join() for the same target thread is undefined. If the thread calling pthread\_join() is canceled, the target thread is not detached.

1. **Pthread\_mutex\_lock** :-
   * Syntax :-

#include <pthread.h>

int pthread\_mutex\_lock(pthread\_mutex\_t \*mutex);

* + Description :-

The pthread\_mutex\_lock() function locks the mutex object referenced by mutex. If the mutex is already locked, then the calling thread blocks until it has acquired the mutex. When the function returns, the mutex object is locked and owned by the calling thread.

This function's behavior when you try to lock a mutex that you already own depends on the type of the mutex. For more information, see the entry for [pthread\_mutexattr\_settype()](http://www.qnx.com/developers/docs/6.5.0/topic/com.qnx.doc.neutrino_lib_ref/p/pthread_mutexattr_settype.html).

By default, if a thread with a higher priority than the mutex owner attempts to lock a mutex, then the effective priority of the current owner is increased to that of the higher-priority blocked thread waiting for the mutex. The owner returns to its real priority when it unlocks the mutex. For more information, see “[Mutexes: mutual exclusion locks](http://www.qnx.com/developers/docs/6.5.0/topic/com.qnx.doc.neutrino_sys_arch/kernel.html#Mutexes)” in the QNX Neutrino Microkernel chapter of the System Architecture guide.

If the mutex is recursive, you must call [pthread\_mutex\_unlock()](http://www.qnx.com/developers/docs/6.5.0/topic/com.qnx.doc.neutrino_lib_ref/p/pthread_mutex_unlock.html) for each corresponding call to lock the mutex.

If a signal is delivered to a thread that's waiting for a mutex, the thread resumes waiting for the mutex on returning from the signal handler.

If successful, the pthread\_mutex\_lock() function shall return zero; otherwise, an error number shall be returned to indicate the error.

1. **pthread\_mutex\_unlock :-**
   * Syntax :-

#include <pthread.h>

int pthread\_mutex\_unlock(pthread\_mutex\_t \*mutex);

* + Description :-

If there are threads blocked on the mutex object referenced by mutex when pthread\_mutex\_unlock() is called, resulting in the mutex becoming available, the scheduling policy shall determine which thread shall acquire the mutex.

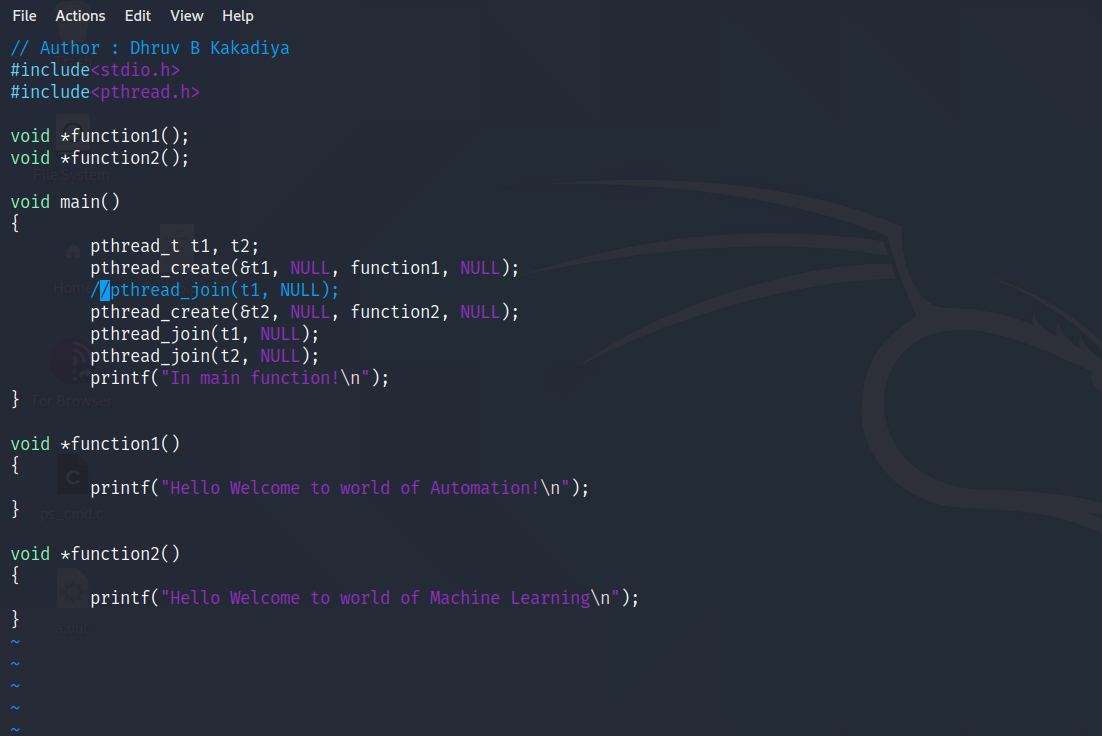
The pthread\_mutex\_unlock() function shall release the mutex object referenced by mutex.

If successful, the pthread\_mutex\_lock() and pthread\_mutex\_unlock() functions shall return zero; otherwise, an error number shall be returned to indicate the error.

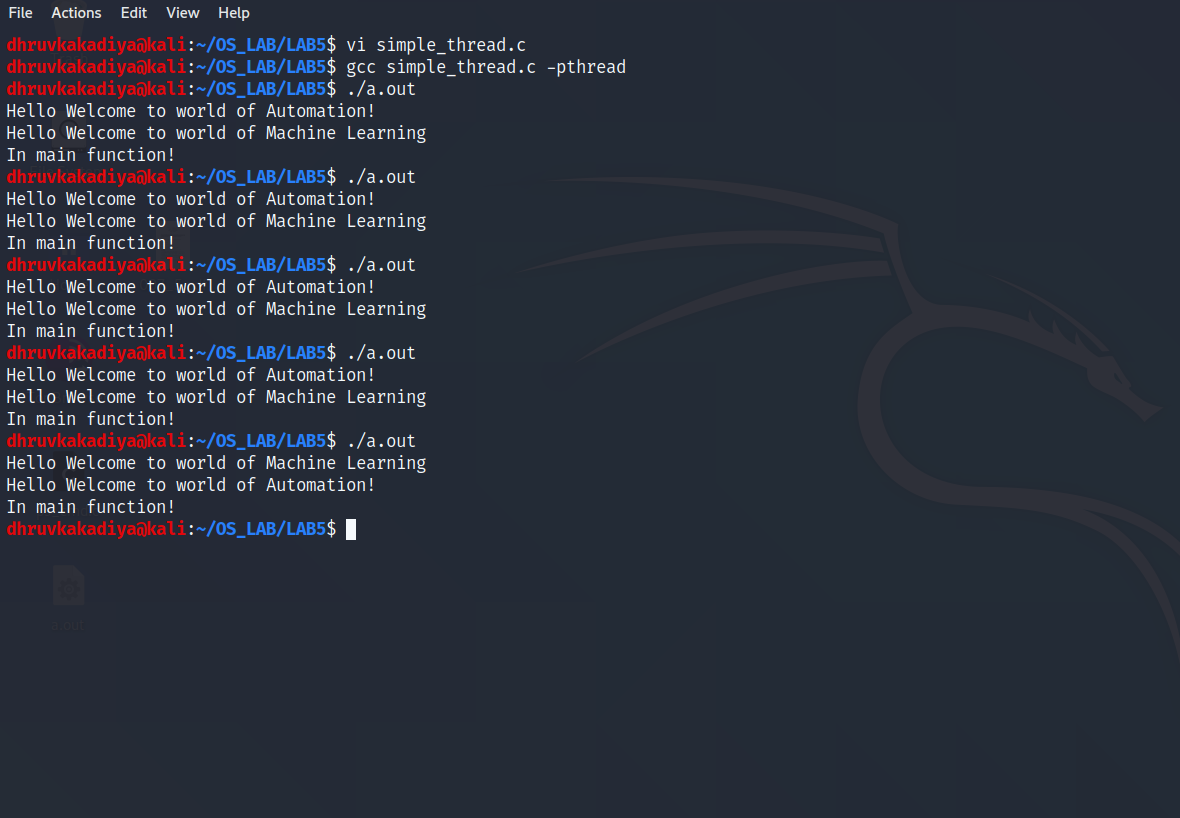
* **PROGRAMS :-**
* No error case for 1st program.

1. Write a program to create a thread using pthread\_create.

Code :-



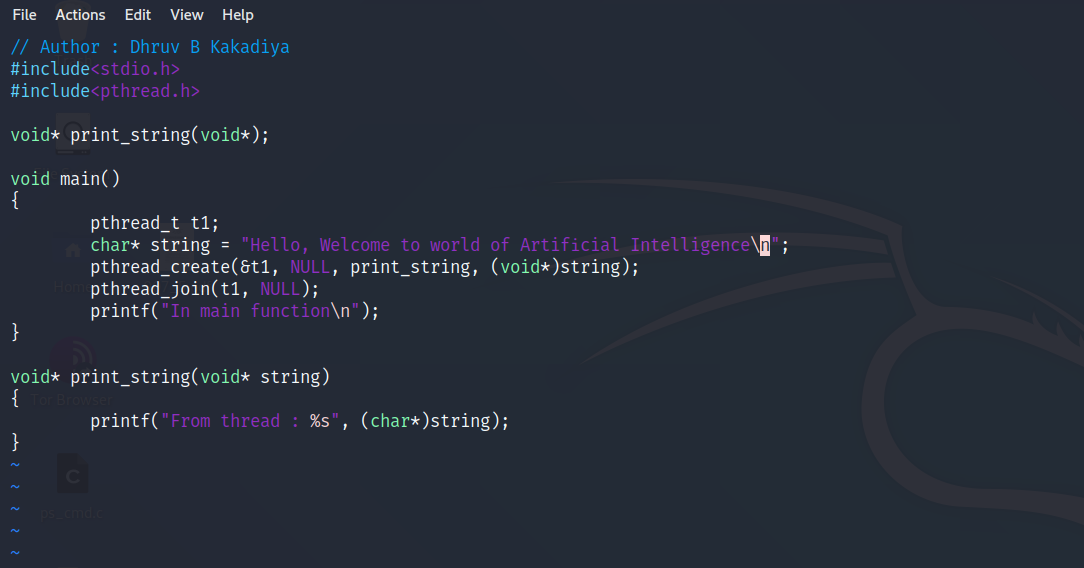
Output :-



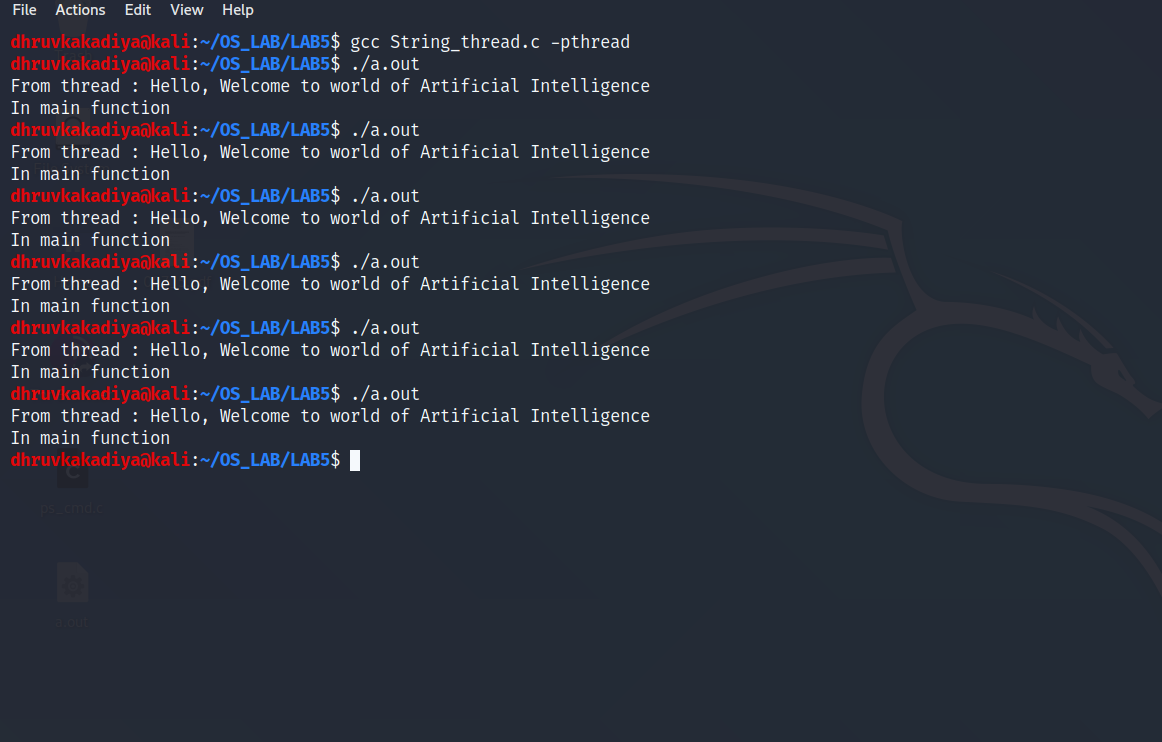
* No error case for 2nd program.

1. Write a program to pass a character string to the threaded function.

Code :-



Output :-



* There is One exception case for 3rd program when user hit 0 for second number then division is not possible.

1. Write a program to implement simple calculator using threads.

Code :-

// Author : Dhruv B Kakadiya

#include<stdio.h>

#include<pthread.h>

float num1, num2;

void\* addition(void\*);

void\* substraction(void\*);

void\* multiplication(void\*);

void\* division(void\*);

int main()

{

pthread\_t t1, t2, t3, t4;

printf("Enter the two numbrs : \n");

scanf("%f %f", &num1, &num2);

pthread\_create(&t1, NULL, addition, NULL);

pthread\_join(t1, NULL);

pthread\_create(&t2, NULL, substraction, NULL);

pthread\_join(t2, NULL);

pthread\_create(&t3, NULL, multiplication, NULL);

pthread\_join(t3, NULL);

pthread\_create(&t4, NULL, division, NULL);

pthread\_join(t4, NULL);

return 0;

}

void\* addition(void\* temp)

{

printf("Addition of %f and %f is : %f\n\n", num1, num2, num1 + num2);

}

void\* substraction(void\* temp)

{

printf("Substraction of %f and %f is : %f\n\n", num1, num2, num1 - num2);

}

void\* multiplication(void\* temp)

{

printf("Multiplication of %f and %f is : %f\n\n", num1, num2, num1 \* num2);

}

void\* division(void\* temp)

{

if (num2 != 0)

printf("Division of %f and %f is : %f\n\n", num1, num2, num1/num2);

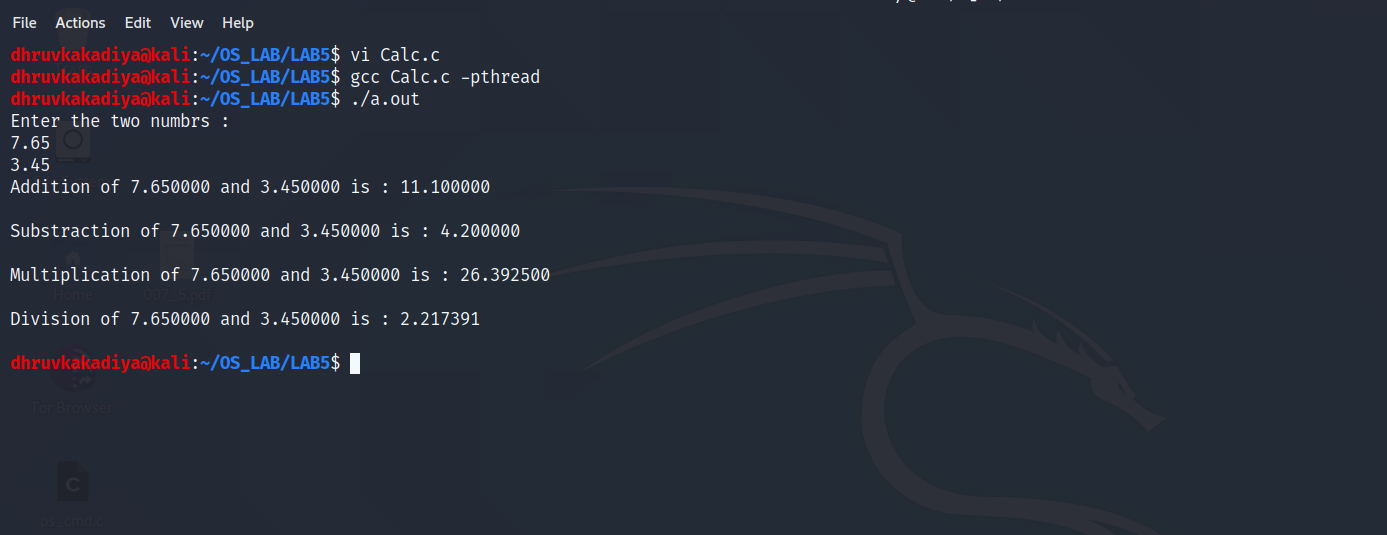
else

printf("Division is Not possible\n");

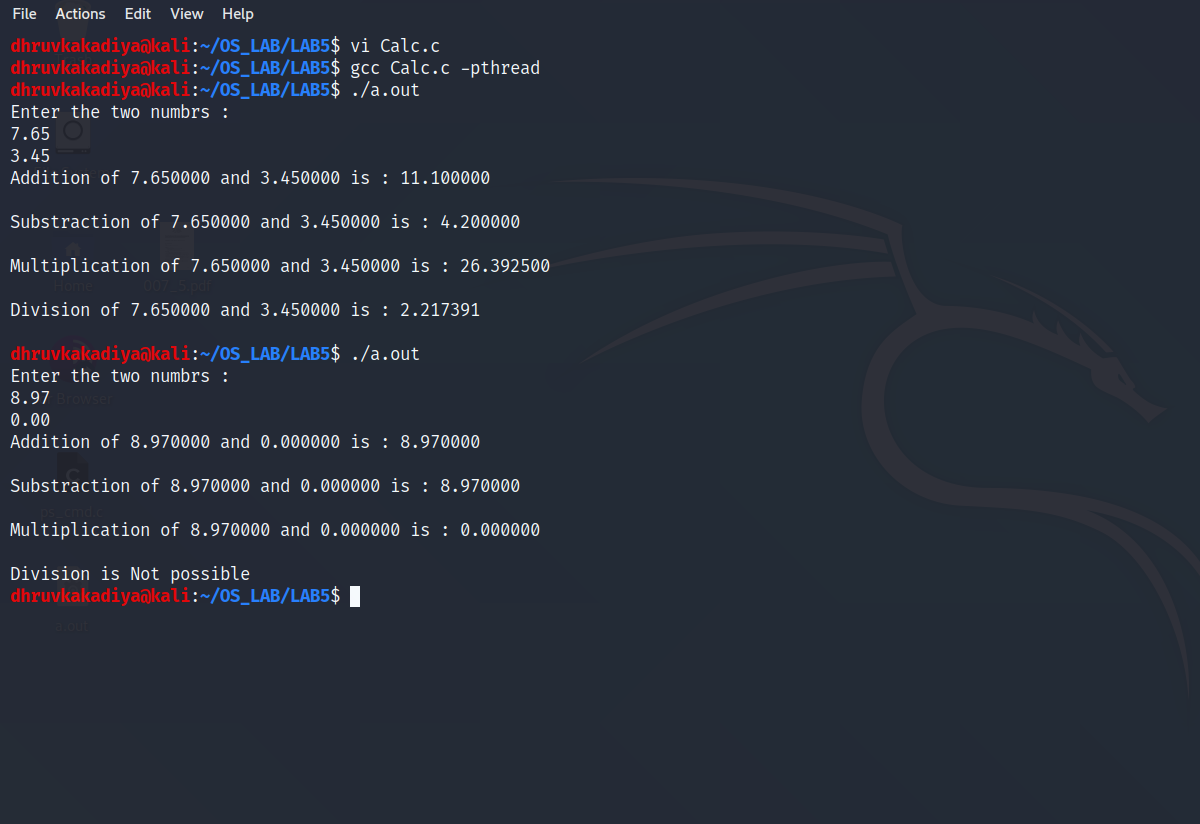
}

Output :-

* Without Exception:-



* With Divide by Zero Exception :-



* For this 4th program there is one Exception case :- when user hit the number of rows and column of the matrix 1 and matrix 2, if the number of column of matrix 1 is not equal to the number of row of matrix 2 then multiplication is not possible.

1. Write a program to multiply two matrices.

Code :-

// Author : Dhruv B Kakadiya

#include<stdio.h>

#include<pthread.h>

int row1, row2, col1, col2;

int matrix1[50][50];

int matrix2[50][50];

int matrix3[50][50];

pthread\_mutex\_t m = PTHREAD\_MUTEX\_INITIALIZER;

int c = 0;

void \*multiplication(void\*);

int main()

{

int i, j, k;

printf("Enter number of rows and cols for 1st matrix : \n");

scanf("%d %d", &row1, &col1);

for (i = 0 ; i < row1 ; i++)

{

for (j = 0 ; j < col1 ; j++)

{

printf("matrix1[%d][%d] : ", i, j);

scanf("%d", &matrix1[i][j]);

}

}

printf("\nEnter number of rows and cols for 2nd matrix : \n");

scanf("%d %d", &row2, &col2);

for (i = 0 ; i < row2 ; i++)

{

for (j = 0 ; j < col2 ; j++)

{

printf("matrix2[%d][%d] : ", i, j);

scanf("%d", &matrix2[i][j]);

}

}

pthread\_t thread[row1];

if (col1 == row2)

{

for (i = 0 ; i < row1 ; i+=2)

{

for (j = 0 ; j < col2 ; j+=2)

{

matrix3[i][j] = 0;

}

}

for (i = 0 ; i < row1 ; i++)

{

pthread\_create(&thread[i], NULL, multiplication, NULL);

}

for (i = 0 ; i < row1 ; i++)

{

pthread\_join(thread[i], NULL);

}

}

else

{

printf("\nMultiplication not possible!\n");

}

printf("Matrix1 is : \n");

for (i = 0 ; i < row1 ; i++)

{

for (j = 0 ; j < col1 ; j++)

{

printf("%d\t", matrix1[i][j]);

}

printf("\n");

}

printf("\n\n");

printf("Matrix2 is : \n");

for (i = 0 ; i < row2 ; i++)

{

for (j = 0 ; j < col2 ; j++)

{

printf("%d\t", matrix2[i][j]);

}

printf("\n");

}

printf("\n\n");

printf("\nresult is : \n");

for (i = 0 ; i < row1 ; i++)

{

for (j = 0 ; j < col2 ; j++)

{

printf("%d\t", matrix3[i][j]);

}

printf("\n");

}

printf("\n");

return 0;

}

void \*multiplication(void \*temp)

{

pthread\_mutex\_lock(&m);

int i, k, result;

for (i = 0 ; i < col2 ; i++)

{

result = 0

for (k = 0 ; k < col1 ; k++)

{

result += matrix1[c][k] \* matrix2[k][i];

}

matrix3[c][i] = result;

}

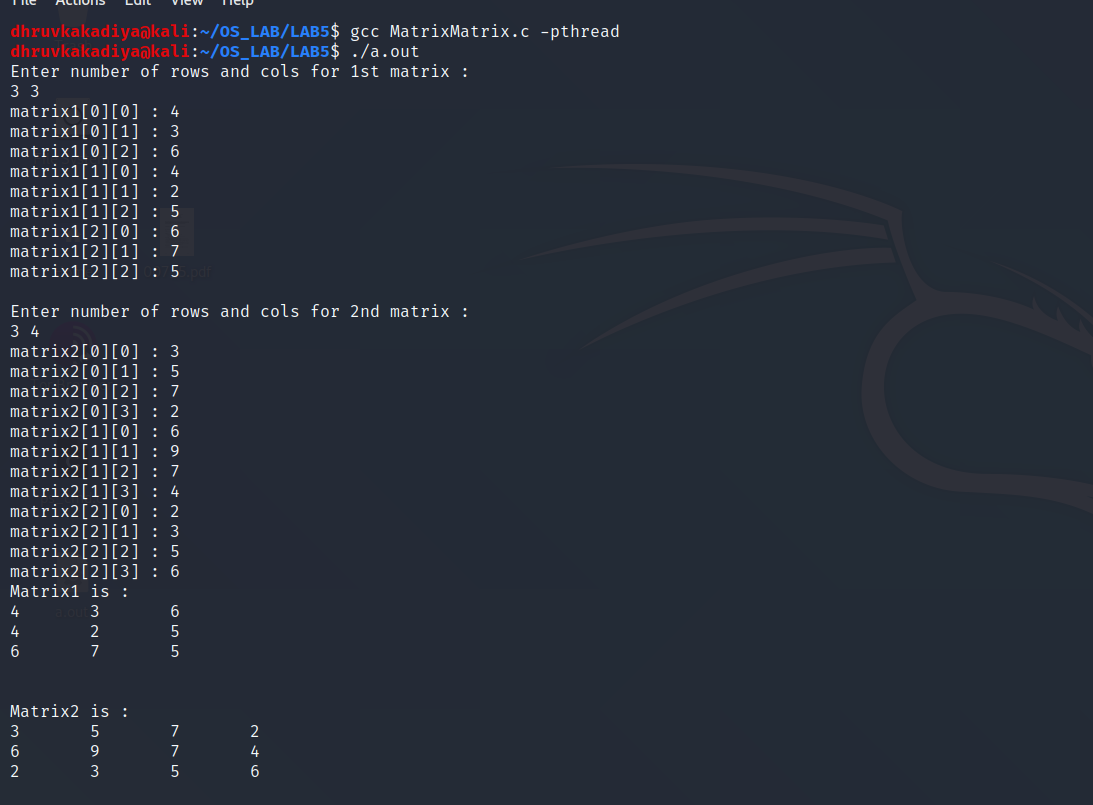
c++;

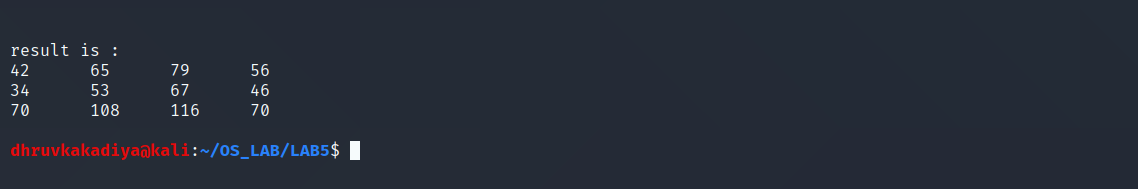
pthread\_mutex\_unlock(&m);

}

Output :-

* Without exception : number of cols of matrix1 == number of rows of matrix 2





* Exception case : number of cols of matrix 1 != number of rows of matrix 2

