horizontal line

**Assignment 05 | Operating System**

**CE-092**

Assignment submission for Operating System subject week 5.

nevilparmar24@gmail.com

**─**

**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).**

**Pthread\_create:**

**#include <pthread.h>**

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

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

The pthread\_create() function starts a new thread in the calling process. The new thread starts execution by invoking *start\_routine*(); *arg* is passed as the sole argument of *start\_routine*().

The *attr* argument points to a *pthread\_attr\_t* structure whose contents are used at thread creation time to determine attributes for the new thread; this structure is initialized using pthread\_attr\_init and related functions. If *attr* is NULL, then the thread is created with default attributes.

Before returning, a successful call to pthread\_create() stores the ID of the new thread in the buffer pointed to by *thread*; this identifier is used to refer to the thread in subsequent calls to other pthreads functions.

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

**Pthread\_join:**

**#include <pthread.h>**

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

Compile and link with *-pthread*.

The pthread\_join() function waits for the thread specified by *thread* to terminate. If that thread has already terminated, then pthread\_join() returns immediately. The thread specified by *thread* must be joinable.

On success, pthread\_join() returns 0; on error, it returns an error number.

**Pthread mutext\_lock:**

**#include <pthread.h>**

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

The mutex object referenced by *mutex* shall be locked by calling *pthread\_mutex\_lock*(). If the mutex is already locked, the calling thread shall block until the mutex becomes available. This operation shall return with the mutex object referenced by *mutex* in the locked state with the calling thread as its owner.

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.

**Pthread mutext\_unlock:**

**#include <pthread.h>**

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

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

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

**Task 1:**

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

Code:

#*include*<stdio.h>

#*include*<pthread.h>

void \* *f1*()

{

*printf*("Hello from thread\n");

}

int *main*()

{

pthread\_t t1;

*pthread\_create*(&t1, NULL , f1, NULL);

// *pthread\_join(t1, NULL);*

*return* 0 ;

}

/\*

*It prints nothing because we are not making the main thread to wait for this t1 thread.*

*Hence the output got stuck in the buffer and nothing gets printed on the terminal*

*So , if you run it 5 6 times continuosly, you might see the output of f1 function.*

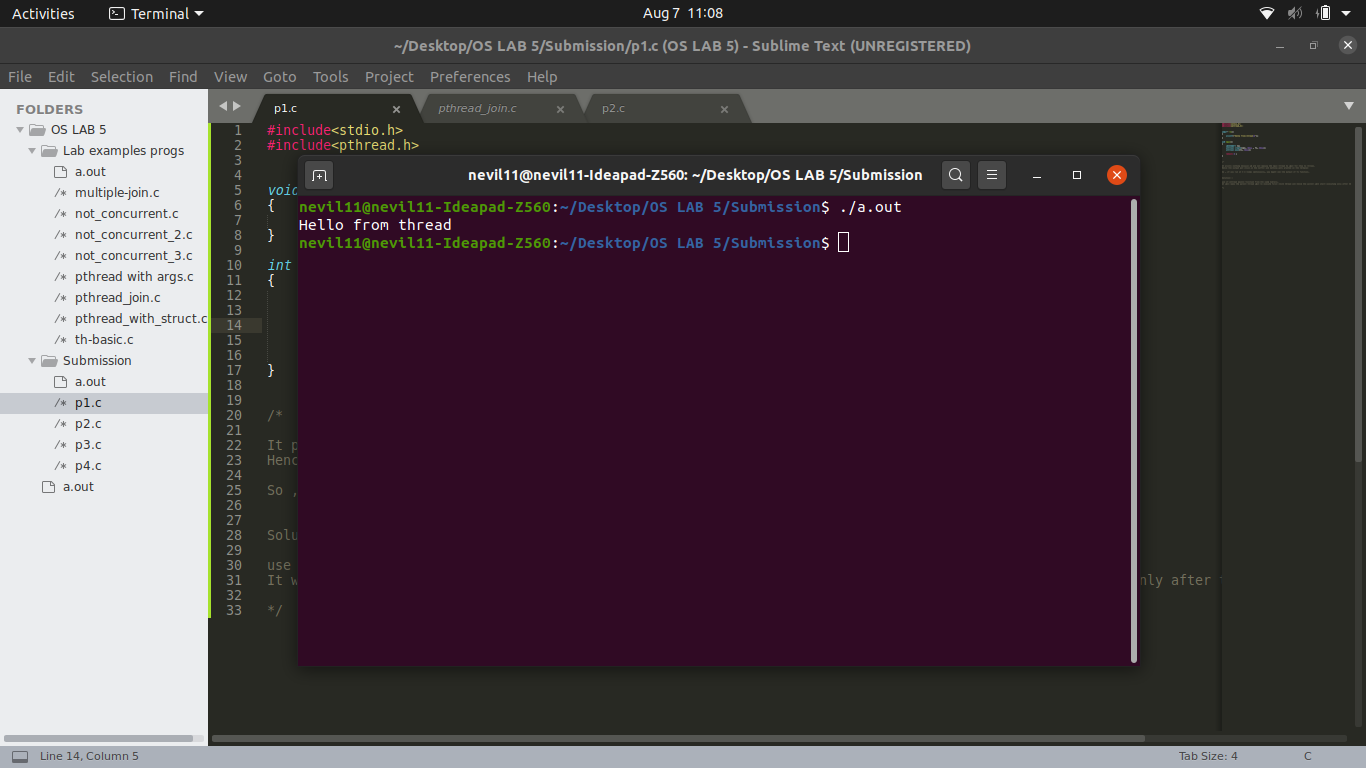
*Solution :*

*use of pthread\_join() function from the same library.*

*It will make the parent thread wait to execute first child thread and hence the parent will start executing only after the child gets destroyed.*

\*/

Output:



**Task 2:**

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

Code:

#*include*<stdio.h>

#*include*<pthread.h>

void \* *fun*(void \*str)

{

*printf*("Passed String: %s\n",str);

*return* *NULL*;

}

int *main*()

{

pthread\_t newth1;

char \*s = "I am a string passed as arg.";

*pthread\_create*(&newth1,*NULL*,*fun*,s);

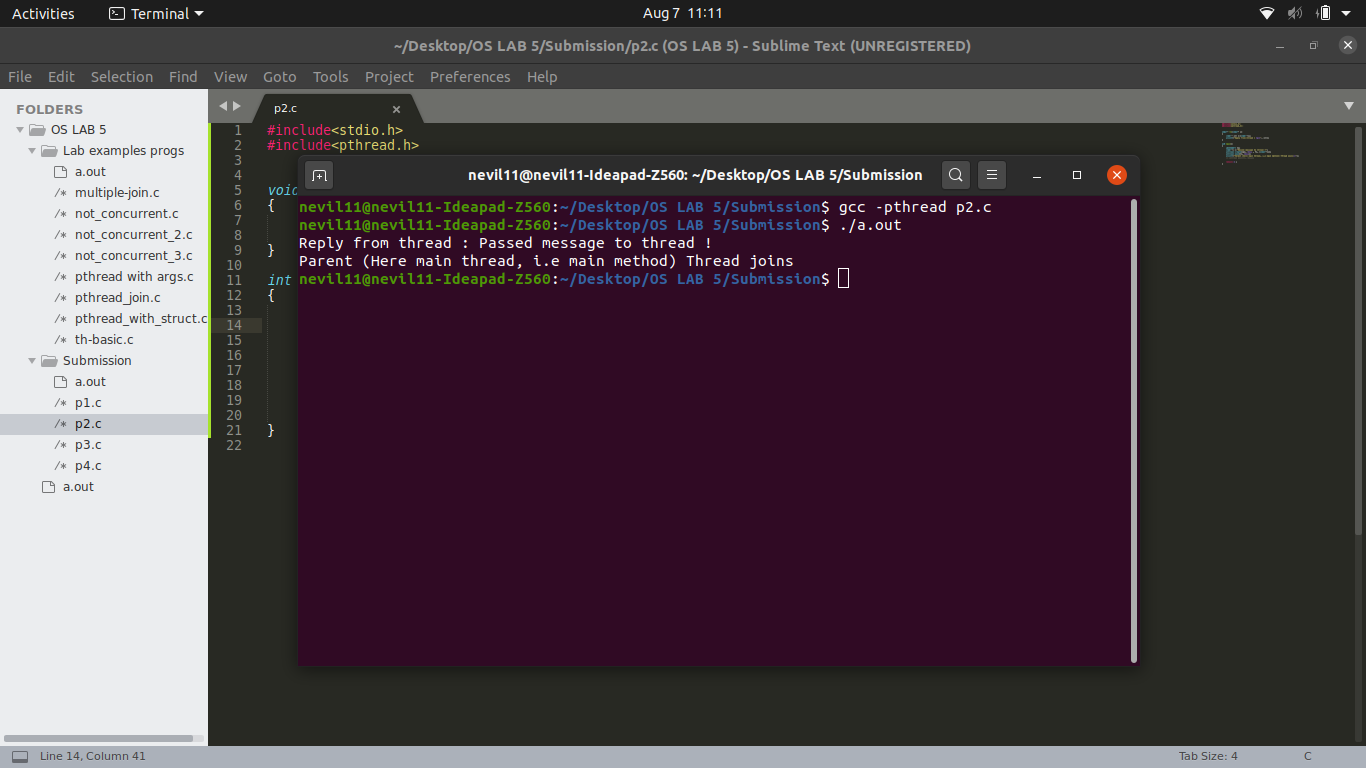
*pthread\_join*(newth1,*NULL*);

*printf*("I am the last line in main.\n");

*return* 0;

}

Output:



**Task 3:**

Write a program to implement a simple calculator using threads.

Code:

// *simple calc*

#*include*<stdio.h>

#*include*<pthread.h>

#*include*<time.h>

struct nums

{

int x;

int y;

};

void \**add*(void \*numsref)

{

*printf*("Add: %d\n", ((struct nums\*)numsref)->x + ((struct nums\*)numsref)->y);

*return* *NULL*;

}

void \**sub*(void \*numsref)

{

*printf*("Sub: %d\n", ((struct nums\*)numsref)->x - ((struct nums\*)numsref)->y);

*return* *NULL*;

}

void \**mul*(void \*numsref)

{

*printf*("Mul: %d\n", ((struct nums\*)numsref)->x \* ((struct nums\*)numsref)->y);

*return* *NULL*;

}

void \**div*(void \*numsref)

{

*printf*("Div: %d\n", ((struct nums\*)numsref)->x / ((struct nums\*)numsref)->y);

*return* *NULL*;

}

int *main*()

{

// *Calculate the time taken by fun()*

clock\_t t;

t = *clock*();

int i;

struct nums n;

n.x = 10;

n.y = 20;

pthread\_t threads[4];

// *array of function pointers*

void \* f[4];

f[0] = *add*;

f[1] = *sub*;

f[2] = *mul*;

f[3] = *div*;

*for*(i=0;i<4;i++)

{

*pthread\_create*(&threads[i],*NULL*,f[i],&n);

}

*for*(i=0;i<4;i++)

{

*pthread\_join*(threads[i],*NULL*);

}

*printf*("I am the last line in main.\n");

t = *clock*() - t;

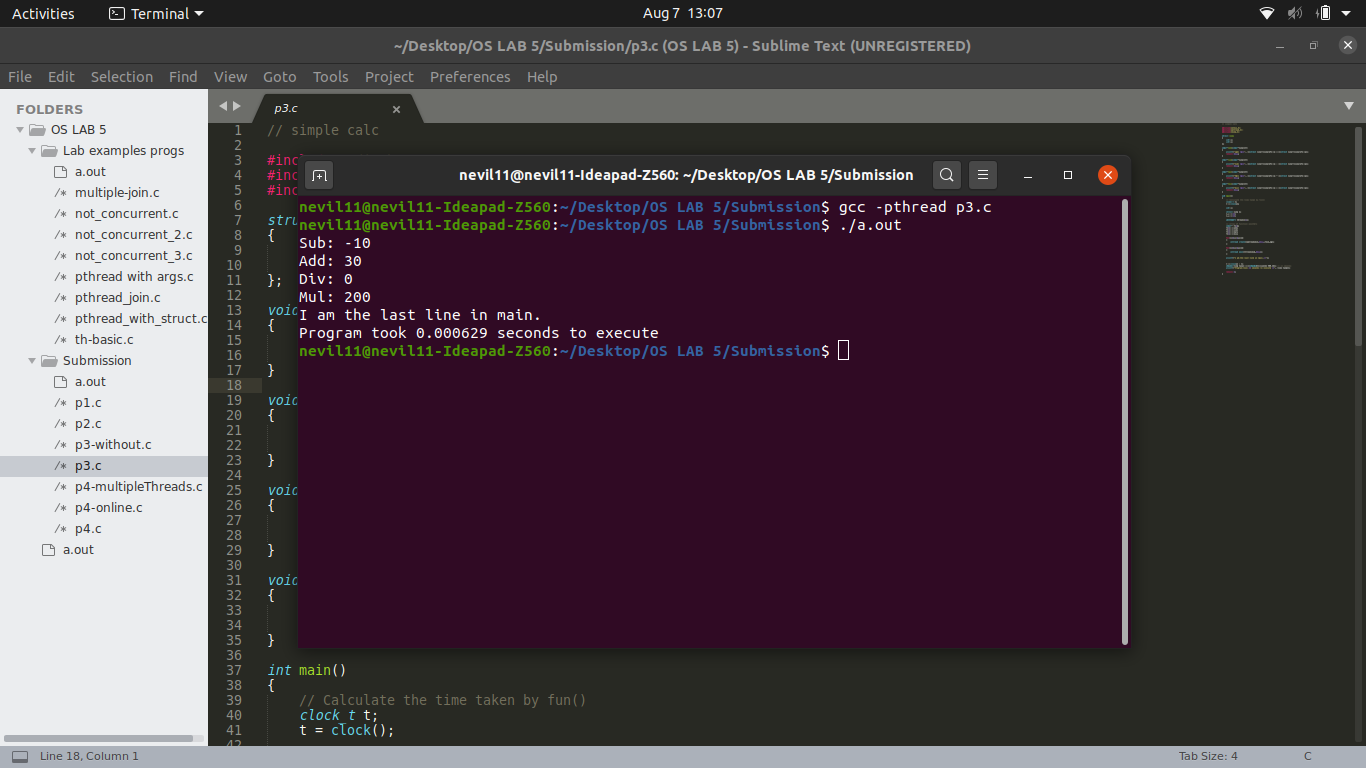
double time\_taken = ((double)t)/*CLOCKS\_PER\_SEC*; // *in seconds*

*printf*("Program took %f seconds to execute \n", time\_taken);

*return* 0;

}

Output:



**Task 4:**

Write a program to multiply two matrices.

Code:

# *include* <stdio.h>

# *include* <pthread.h>

#*include*<stdlib.h>

int MAT1[10][10];

int MAT2[10][10];

int MAT3[10][10];

int r1,c1,r2,c2;

void \**multiply*(void \*);

int *main*()

{

pthread\_t tid;

int i,j,kCount;

*printf*("Enter Number of Rows For Matrix 1 :");

*scanf*("%d",&r1);

*printf*("Enter Number of Columns For Matrix 1 :");

*scanf*("%d",&c1);

*for*(i=0;i<r1;i++)

*for*(j=0;j<c1;j++)

MAT1[i][j] = *rand*() %10;

*printf*("Enter Numer of Rows For Matrix 2 :");

*scanf*("%d",&r2);

*printf*("Enter Number of Columns For Matrix 2 :");

*scanf*("%d",&c2);

*for*(i=0;i<r2;i++)

*for*(j=0;j<c2;j++)

MAT2[i][j] = *rand*() %10;

*if*(c1!=r2)

{

*printf*("Multipication of Matrix not Possible !!!");

}

*else*

{

*for*(i=0;i<r1;i=i+2)

{

*for*(j=0;j<c2;j=j+2)

{

MAT3[i][j]=0;

}

}

*pthread\_create*(&tid,*NULL*,*multiply*,*NULL*);

*for*(i=0;i<r1;i=i+2)

{

*for*(j=0;j<c2;j++)

{

*for*(kCount=0;kCount<c1;kCount++)

{

MAT3[i][j]+=MAT1[i][kCount] \* MAT2[kCount][j];

}

}

}

*pthread\_join*(tid,*NULL*);

}

*printf*("\nMatrix 1 \n");

*for*(i=0;i<r1;i++)

{

*for*(j=0;j<c1;j++)

{

*printf*("%d \t",MAT1[i][j]);

}

*printf*("\n");

}

*printf*("\nMatrix 2 \n");

*for*(i=0;i<r2;i++)

{

*for*(j=0;j<c2;j++)

{

*printf*("%d \t",MAT2[i][j]);

}

*printf*("\n");

}

*printf*("\nMultipication of Matrix ...\n");

*for*(i=0;i<r1;i++)

{

*for*(j=0;j<c2;j++)

{

*printf*("%d \t",MAT3[i][j]);

}

*printf*("\n");

}

*return* 0;

}

void \**multiply*(void \*para)

{

int i,j,kCount;

*for*(i=1;i<r1;i=i+2)

{

*for*(j=0;j<c2;j++)

{

*for*(kCount=0;kCount<c1;kCount++)

{

MAT3[i][j]+=MAT1[i][kCount] \* MAT2[kCount][j];

}

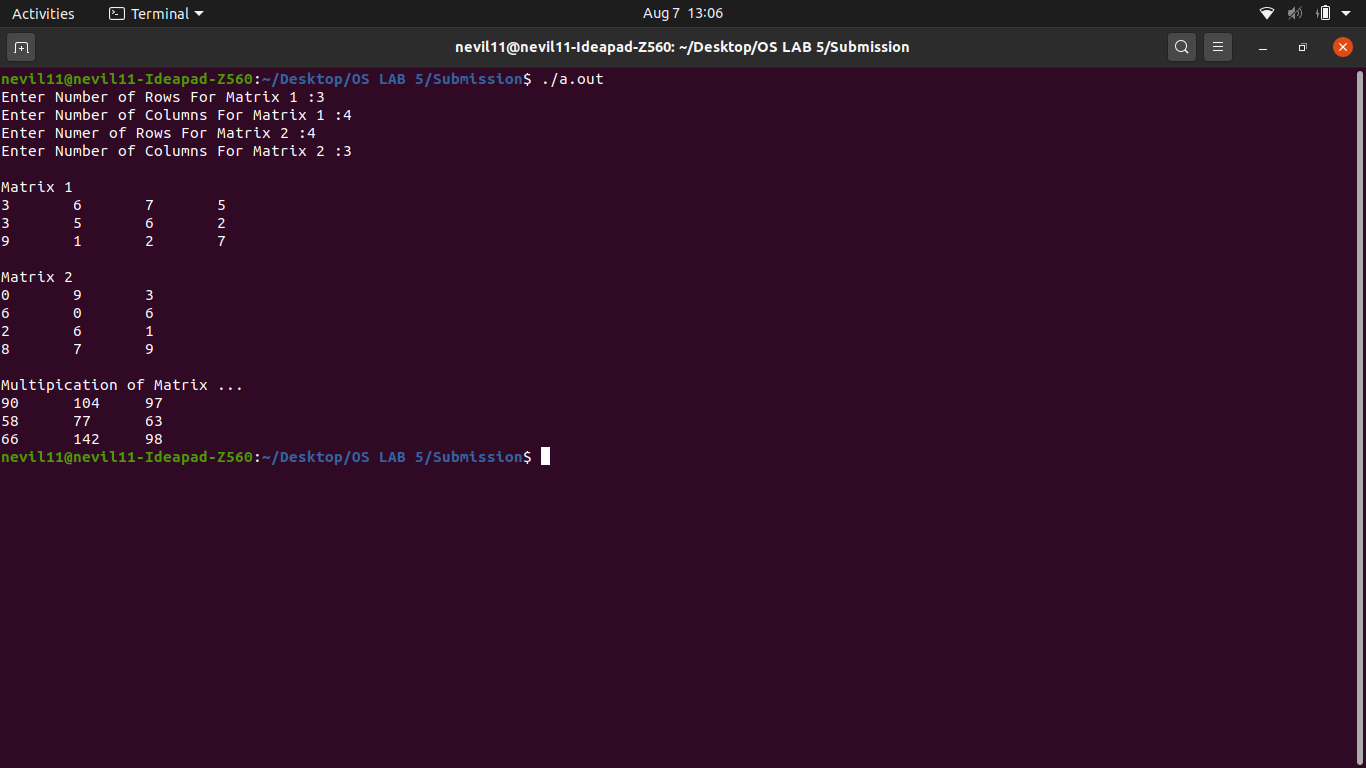
}

}

*pthread\_exit*(*NULL*);

}

Output:



Nevil Parmar

CE-092

https://nevilparmar.me