**PDC Assignment Lab Manual Solutions**

***LAB # 07:***

**Task# 01:**

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

void \*func1(){

int num = 5, i;

for(i=1; i<=1000; i++)

printf("\n%d x %d : %d",num,i, num\*i);

}

void \*func2(){

int num = 6, i;

for(i=1; i<=1000; i++)

printf("\n%d x %d : %d",num,i, num\*i);

}

void \*func3(){

int num = 7, i;

for(i=1; i<=1000; i++)

printf("\n%d x %d : %d",num,i, num\*i);

}

void \*func4(){

int num = 8, i;

for(i=1; i<=1000; i++)

printf("\n%d x %d : %d",num,i, num\*i);

}

int main(){

pthread\_t threads[4];

pthread\_create(&threads[0], NULL, func1, NULL);

pthread\_join(threads[0], NULL);

printf("\n----------------------------");

pthread\_create(&threads[1], NULL, func2, NULL);

pthread\_join(threads[1], NULL);

printf("\n----------------------------");

pthread\_create(&threads[2], NULL, func3, NULL);

pthread\_join(threads[2], NULL);

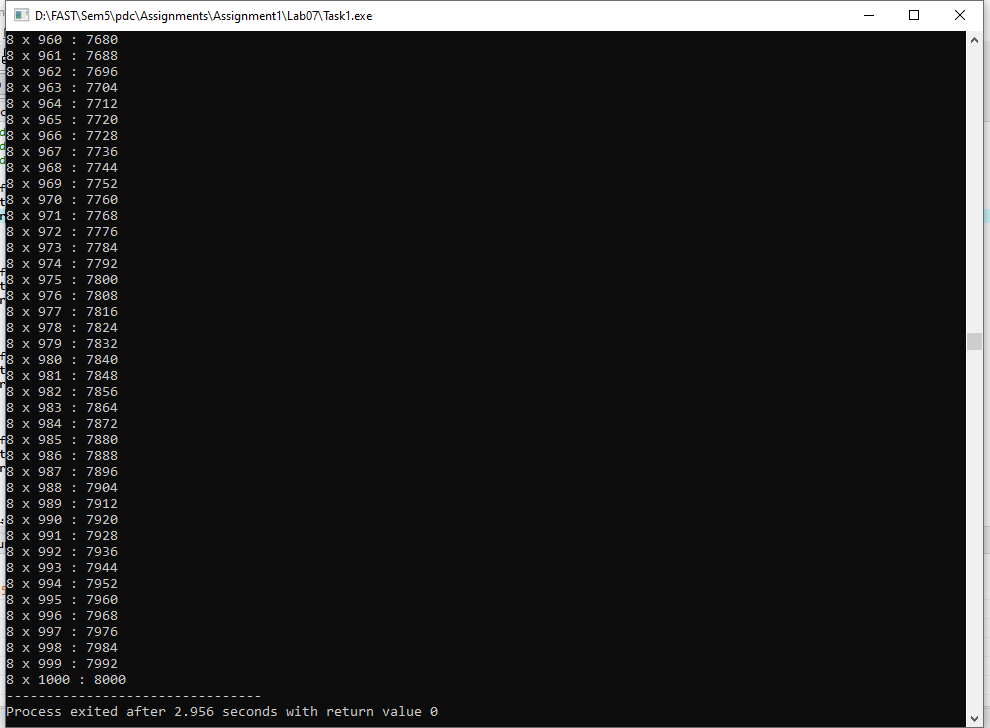
printf("\n----------------------------");

pthread\_create(&threads[3], NULL, func4, NULL);

pthread\_join(threads[3], NULL);

return 0;

}

**Output:**

**Explanation:**

This code is very simple and self-explanatory. I just made 4 different threads and created them and initialized them with 4 different functions and each runs a different table till 1000 times. Since the output is very large, I’ve just shown the table 8’s last few iterations but you can rerun to check itss output.

**Task# 02:**

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

void \*Volunteer(void \*ThreadId){

int id = (int)ThreadId, i2=0, i;

if(id == 0)

printf("Volunteer 1 Manages on Day Registeration.\n");

else if (id==1)

printf("Volunteer 2 handles announcements.\n");

else if (id==2)

printf("Volunteer 3 handles sponsors.\n");

else if (id==3)

printf("Volunteer 4 resolve queries of particpants.\n");

pthread\_t pid[100];

for(i=0; i<100; i++, i2++){

pthread\_create(&pid[i], NULL, Volunteer, (void\*)i2);

}

}

int main(){

int status, i=0;

pthread\_t tid1, tid2, tid3, tid4;

pthread\_create(&tid1, NULL, Volunteer, (void\*)i++);

pthread\_join(tid1, NULL);

pthread\_create(&tid2, NULL, Volunteer, (void\*)i++);

pthread\_join(tid2, NULL);

pthread\_create(&tid3, NULL, Volunteer, (void\*)i++);

pthread\_join(tid3, NULL);

pthread\_create(&tid4, NULL, Volunteer, (void\*)i++);

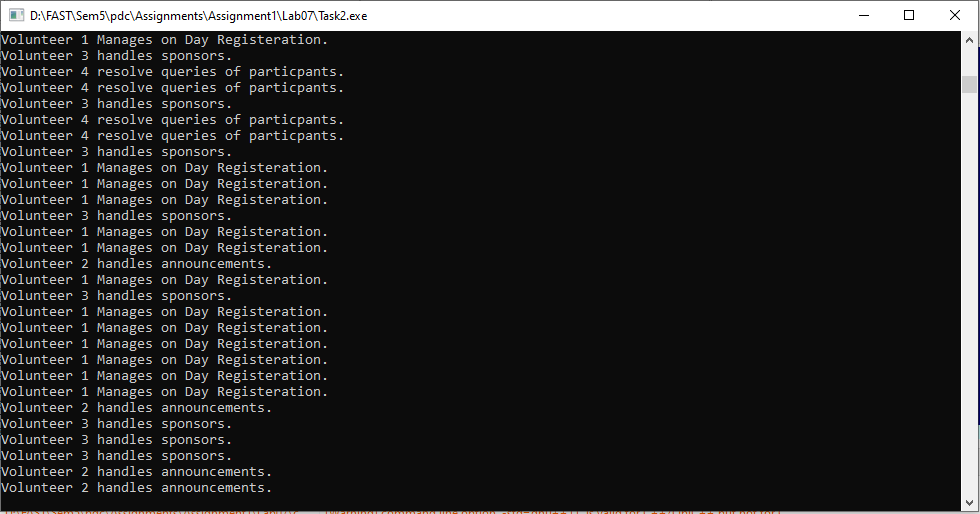
pthread\_join(tid4, NULL);

pthread\_exit(NULL);

return 0;

}

**Output:**



**Explanation:**

For the solution of this question, I’ve just made 4 different threads and 1 running function which will check the thread number and run accordingly. Inside that function I’ve run 100 times each thread for 100 participants. This satisfies the question. The output is a subset of the total output as the complete output can’t be shown in this.

***LAB # 08:***

**Task# 01 (a):**

**Code:**

#include <omp.h>

#include <stdio.h>

#include <stdlib.h>

int main(){

int r,c, i,j, sum=0,k;

printf("Enter row size : ");

scanf("%d",&r);

printf("Enter column size : ");

scanf("%d", &c);

int MatA[r][c], MatB[r][c], Sum[r][c], Mul[r][c];

printf("Matrix A\n");

for(i =0 ; i<r; i++){

for (j=0; j<c; j++){

printf("Enter entry for Row # %d Col # %d : ",i,j);

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

}

}

printf("Matrix B\n");

for(i =0; i<r; i++){

for (j=0; j<c; j++){

printf("Enter entry for Row # %d Col # %d : ",i,j);

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

}

}

printf("\nThe Sum Matrix\n");

#pragma omp parallel shared(MatA, MatB, Sum) private(i,j)

{

#pragma omp for

for(i=0; i<r; i++){

for(j=0; j<c; j++){

Sum[i][j] = MatA[i][j] + MatB[i][j];

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

}

printf("\n");

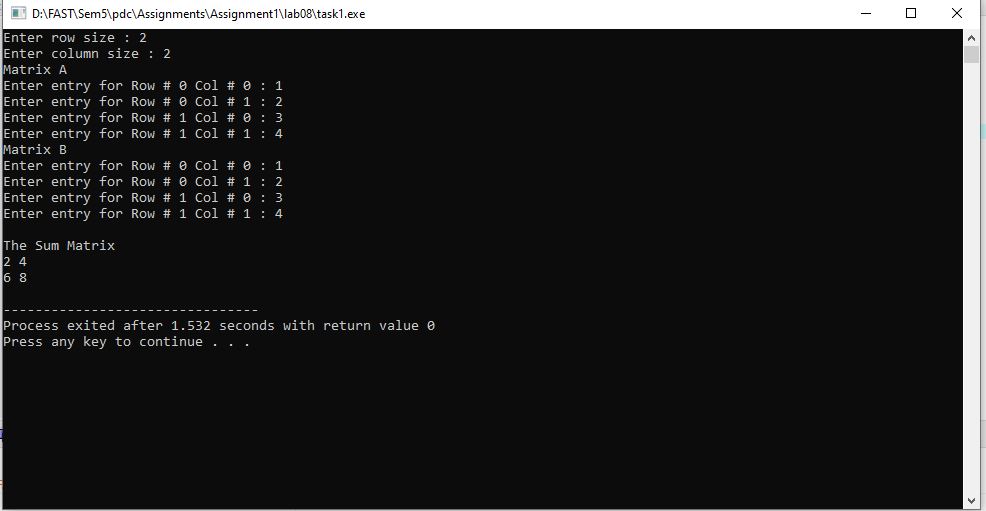
}

}

return 0;

}

**Output:**



**Explanation:**

For the solution of this question, I’ve used open MP library and its function solve the output. I’ve taken matrix input at run time which will run sequentially as it can’t be run in parallel. Then when the inputs are taken, we can find the sum of the 2 matrices in parallel. MatrixA, B, and Sum are shared while the variables I and j are kept private.

**Task# 01(b):**

**Code:**

#include <omp.h>

#include <stdio.h>

#include <stdlib.h>

int main(){

int r1,c1, r2, c2, i,j, sum=0,k;

printf("Enter row size for Matrix A : ");

scanf("%d",&r1);

printf("Enter column size for Matrix A : ");

scanf("%d", &c1);

printf("Enter row size for Matrix B : ");

scanf("%d",&r2);

printf("Enter column size for Matrix B : ");

scanf("%d", &c2);

if(c1 != r2){

printf("Matrix can't be multiplied\n");

exit(0);

}

int MatA[r1][c1], MatB[r2][c2], Mul[r1][c2];

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

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

Mul[i][j]=0;

}

}

printf("Matrix A\n");

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

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

printf("Enter entry for Row # %d Col # %d : ",i,j);

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

}

}

printf("Matrix B\n");

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

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

printf("Enter entry for Row # %d Col # %d : ",i,j);

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

}

}

printf("\nThe Multplied Matrix\n");

#pragma omp parallel shared(MatA, MatB, Mul) private(i,j,k,sum)

{

#pragma omp for

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

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

for(k=0; k<c1; k++){

Mul[i][j] += MatA[i][k]\*MatB[k][j];

}

}

}

}

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

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

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

}

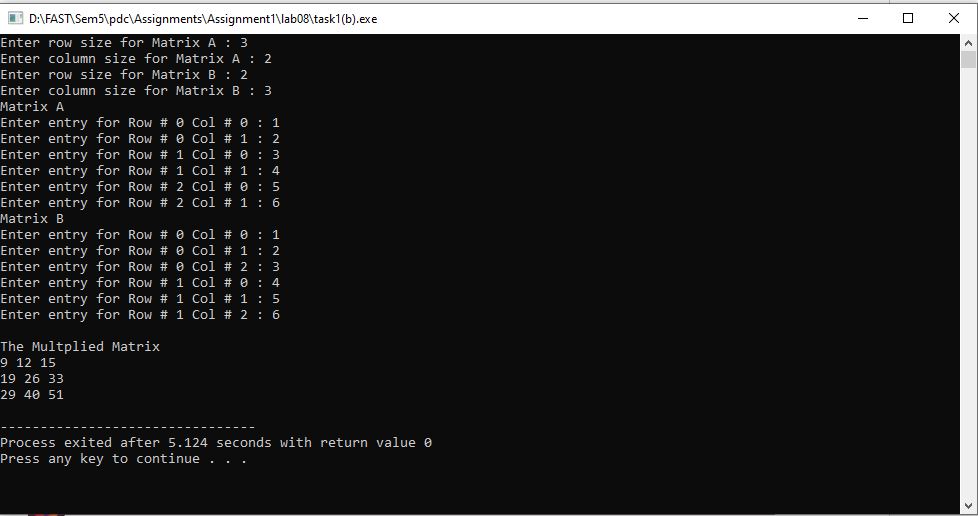
printf("\n");

}

return 0;

}

**Output:**

****

**Explanation:**

For the solution of this question, I’ve used open MP library and its function solve the output. I’ve taken matrix input at run time which will run sequentially as it can’t be run in parallel. For the multiplication, I’ve also checked if the number of rows and number of columns are matching and can be multiplied or not. Then when the inputs are taken, we can find the multiplication of the 2 matrices in parallel. MatrixA, B, and Mult are shared while the variables i, j, k, sum are kept private.

**Task# 02:**

**Code:**

#include <omp.h>

#include <stdlib.h>

#include <stdio.h>

#include<math.h>

int main(void) {

double x, fact[100], power[100], sum, f;

int i, term, k, j;

printf("Exponential [PROMPT] Enter the value of x (between 0 to 100) (for calculating exp(x)) :");

scanf("%lf",&x);

#pragma omp parallel num\_threads(10)

#pragma omp for

for(k=0; k<100; k++)

power[k] = pow(x,k);

fact[0]=1.0;

#pragma omp for

for(j=1; j<100; j++){

f=1.0;

int term;

for(term = j; term>=1; term--)

f \*= term;

fact[j]=f;

}

sum = 0;

#pragma omp for

for(i=0;i<100;i++)

sum = sum + (power[i] / fact[i]);

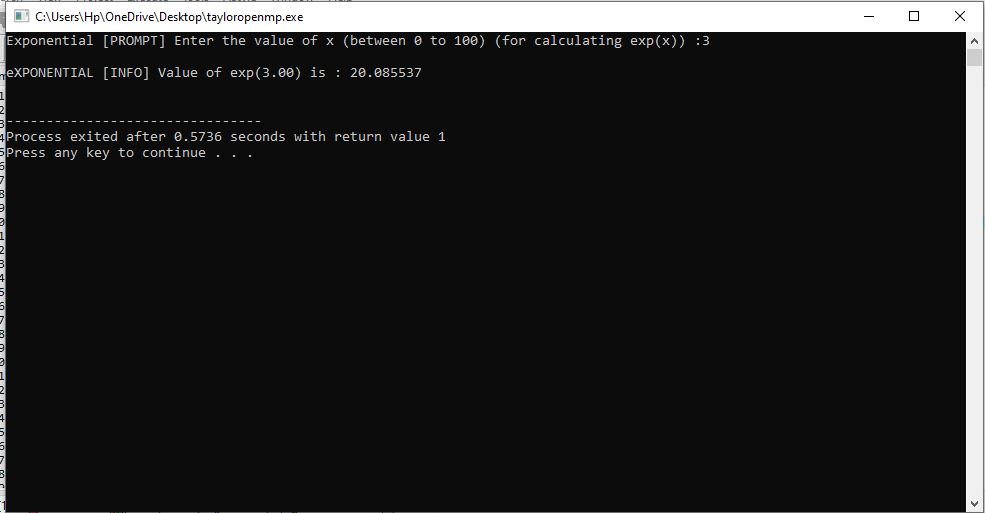
printf("\neXPONENTIAL [INFO] Value of exp(%.2lf) is : %lf\n\n",x,sum);

exit(1);

return 0;

}

**Output:**

****

**Explanation:**

Through open MP, the solution for this problem has become very easy. I’ve made 10 threads that are running in parallel and 3 tasks are running in parallel. The first parallel for loop will find the power of variables just like in Taylor series, the 2nd for loop will find the factorial of the variables in Taylor series and third will divide the power and factorial and then add it into single variable. The first two tasks are independent and can be run in parallel, whereas the sum has to wait until power and factorial are completed.