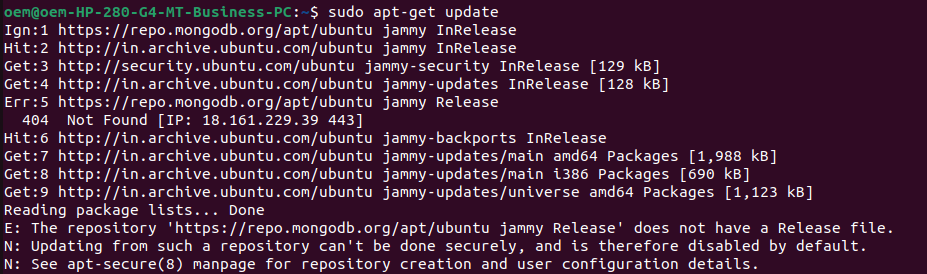
Date: 9/9/24 Experiment- 7

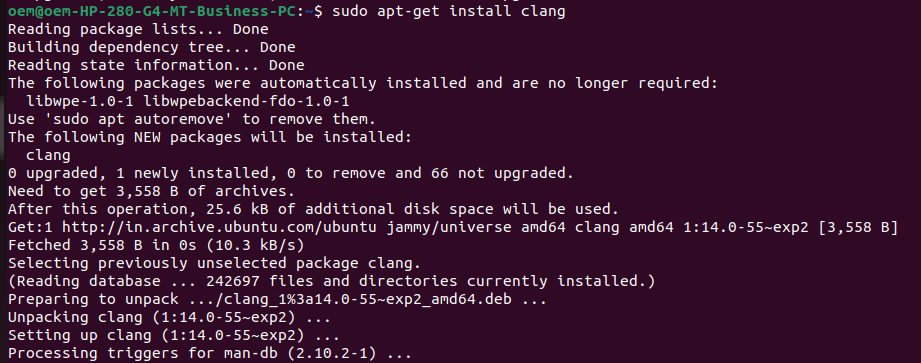
Implementation of Parallel Programming with OpenMp

Aim:

To implement parallel programming with OpenMp to perform Task management

Procedure:





1)

#include <stdio.h>

#include <omp.h>

int main(){

int x;

x = 2;

#pragma omp parallel num\_threads(2) shared(x)

{

if (omp\_get\_thread\_num() == 0) {

x = 5;

} else {

/\* Print 1: the following read of x has a race \*/

printf("1: Thread# %d: x = %d\n", omp\_get\_thread\_num(),x );

}

#pragma omp barrier

if (omp\_get\_thread\_num() == 0) {

/\* Print 2 \*/

printf("2: Thread# %d: x = %d\n", omp\_get\_thread\_num(),x );

} else {

/\* Print 3 \*/

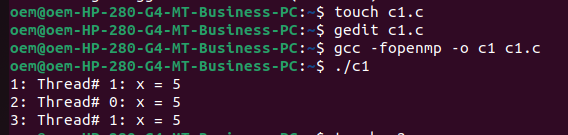
printf("3: Thread# %d: x = %d\n", omp\_get\_thread\_num(),x );

}

}

return 0;

}



#include <omp.h> //<-- necessary header file for OpenMP API

#include <stdio.h>

int main(int argc, char \*argv[]){

printf("OpenMP running with %d threads\n", omp\_get\_max\_threads());

#pragma omp parallel

{

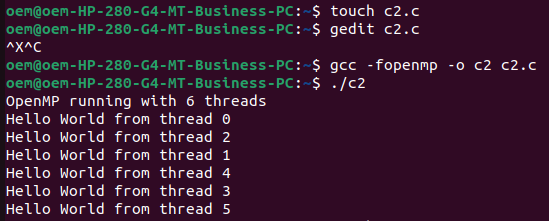
//Code here will be executed by all threads

printf("Hello World from thread %d\n", omp\_get\_thread\_num());

}

return 0;

}



#include <omp.h>

#include <stdio.h>

int main() {

int i;

int N = 1000;

int a = 50;

int b = 0;

#pragma omp parallel for default(none) private(i) shared(N) private(a) private(b)

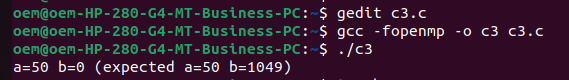
for (i=0; i<N; i++) {

b = a + i;

}

printf("a=%d b=%d (expected a=50 b=1049)\n", a, b);

}



#include <omp.h>

#include <stdio.h>

int main() {

int i;

const int N = 1000;

int a = 50;

int b = 0;

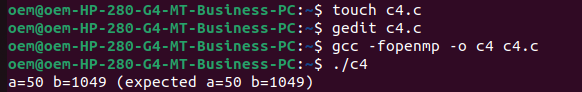
#pragma omp parallel for default(shared)

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

b = a + i;

}

printf("a=%d b=%d (expected a=50 b=1049)\n", a, b);

}  
  
#include <omp.h>

#include <stdio.h>

int main() {

int i;

const int N = 1000; // This variable should be shared among threads

int a = 50;

int b = 0;

#pragma omp parallel for default(none) lastprivate(i) private(a) shared(N) private(b)

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

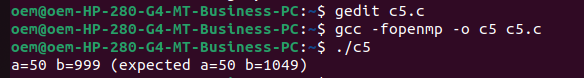
b = a + i;

}

printf("a=%d b=%d (expected a=50 b=1049)\n", a, b);

return 0;

}



#include <omp.h>

#include <stdio.h>

int main() {

int i;

const int N = 1000;

int sum = 0;

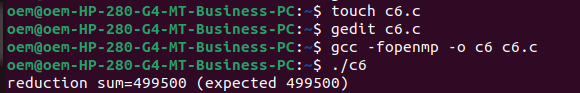
#pragma omp parallel for private(i) reduction(+: sum)

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

sum += i;

}

printf("reduction sum=%d (expected %d)\n", sum, ((N-1)\*N)/2);

}  


#include <omp.h>

#include <stdio.h>

int main() {

double t1, t2;

t1 = omp\_get\_wtime(); // Get the current wall clock time before the expensive operation

// Perform the expensive operation here

// For example:

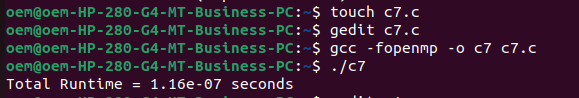
// for (int i = 0; i < 100000000; i++);

t2 = omp\_get\_wtime(); // Get the current wall clock time after the operation

printf("Total Runtime = %g seconds\n", t2 - t1);

return 0;

}



#include <omp.h>

#include <stdio.h>

int main() {

int i;

const int N = 1000;

int a = 50;

int b = 0;

#pragma omp parallel for default(none) private(i) shared(N) firstprivate(a) lastprivate(b)

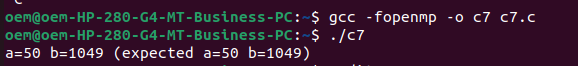
for (i=0; i<N; i++) {

b = a + i;

}

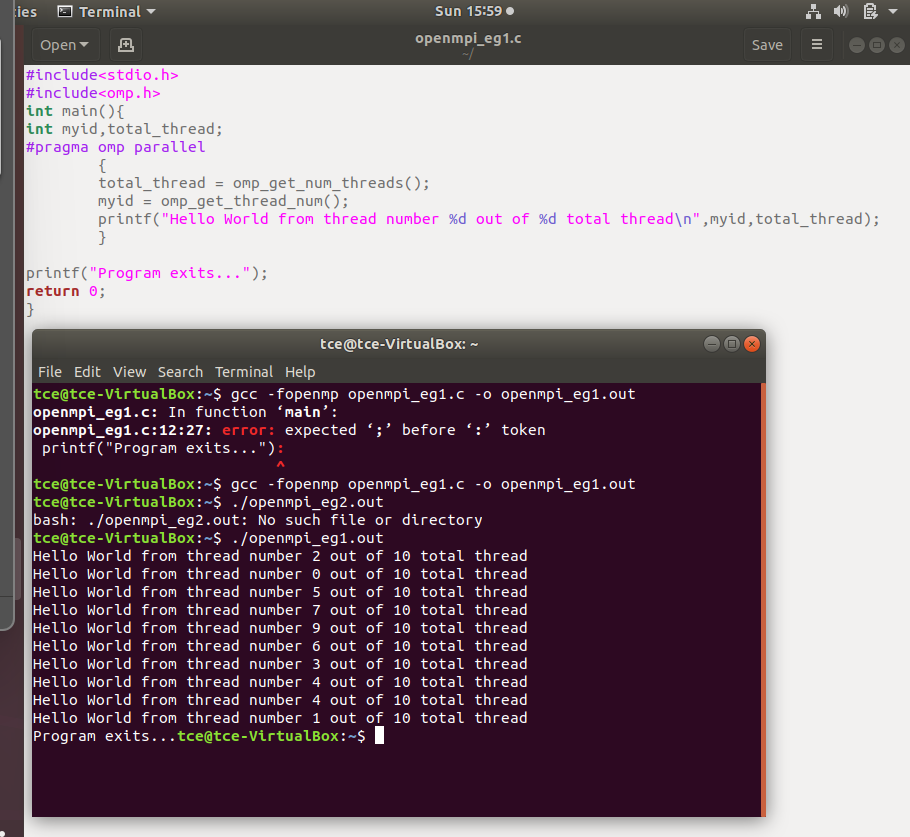
printf("a=%d b=%d (expected a=50 b=1049)\n", a, b);

}



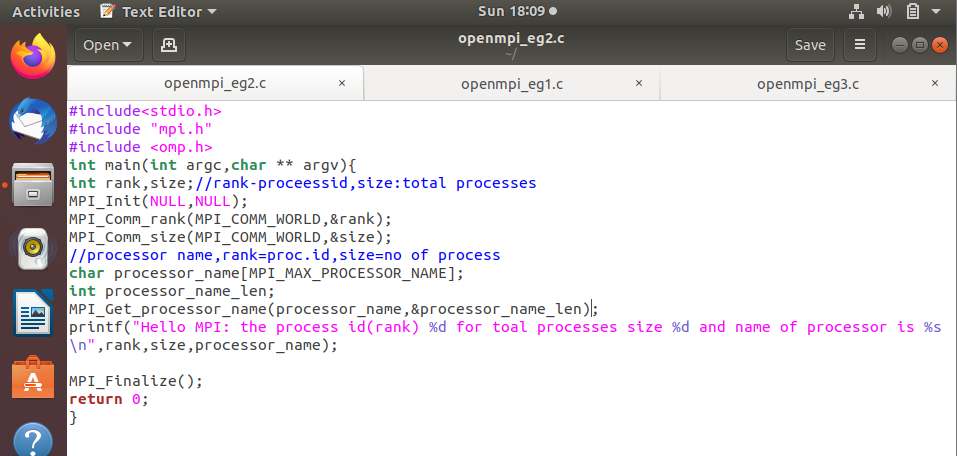
***Jiteshraaju.r***

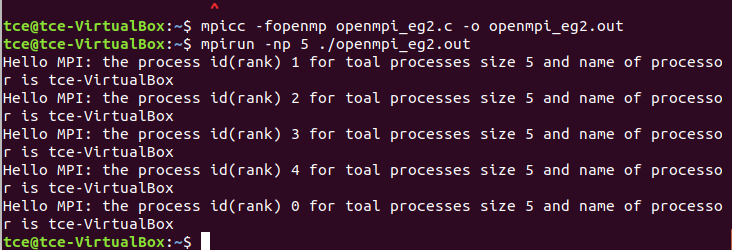
Installation command no need gcc defaultly has openmp(shared memory access),single system with more than 1 cores.



Installation of mpi  
sudo apt-get install libopenmpi-dev  
sudo apt-get install openmpi-bin  
sudo apt-get update  
sudo apt-get upgrade

Mpi-distributed memory space-> clusters of system,each processor with individual memory spaces





**Matrix multiplication**:

#include <stdio.h>

#include<stdio.h>

#include "mpi.h"

#include<omp.h>

#define M 4

#define N 4

#define P 4

int main(int argc,char \*\* argv){

int rank,size,i,j,k;

int A[M][N],B[N][P],C[M][P],A\_local[i][j],C\_local[i][j];

MPI\_Init(&argc,&argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD,&rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD,&size);

//master process i.e rank=0 intialise matrices

if(rank==0){

printf("MatrixA\n");

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

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

A[i][j] = i+j;

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

}

printf("\n");

}//i for

printf("matrix B\n");

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

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

B[i][j] = i\*j;

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

}

printf("\n");

}

}//if rank=0 assign matrix values A,B

// broadcast B matrix to all process

MPI\_Bcast(B,N\*P,MPI\_INT,0,MPI\_COMM\_WORLD);//whichmatrix to broadcast,size of matrix,dataype of each element,which assigh matrices,MPI\_COMM\_WORLd

//scateer parts of A

int A\_rows\_per\_process = M/size;//no of A rows/total process

MPI\_Scatter(A,A\_rows\_per\_process\*N,MPI\_INT,A\_local,A\_rows\_per\_process\*N,MPI\_INT,0,MPI\_COMM\_WORLD);//a mat,matrix size,datatype,in which name matrix stored in each process,each process strored matrix size,0,...

//each process calculate c cvalue

for(i=0;i<A\_rows\_per\_process;i++){{

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

C\_local[i][j] = 0; //init C\_local

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

C\_local[i][j] = A\_local[i][k] \* B[k][j]; }

}

}

}

//gathering all clocal values and storing in and displaying in maon process

MPI\_Gather(C\_local,A\_rows\_per\_process\*P,MPI\_INT,C,A\_rows\_per\_process\*P,MPI\_INT,0,MPI\_COMM\_WORLD);

//printing in main process

if(rank==0){

printf("Resultant C matrix is: \n");

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

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

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

}

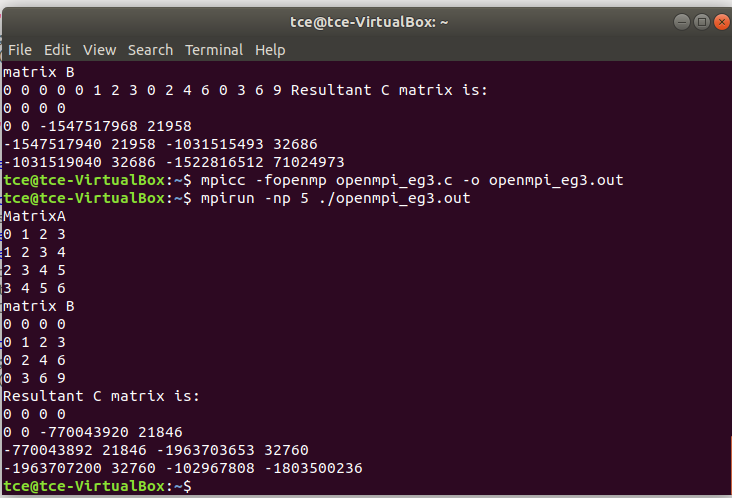
printf("\n");

}

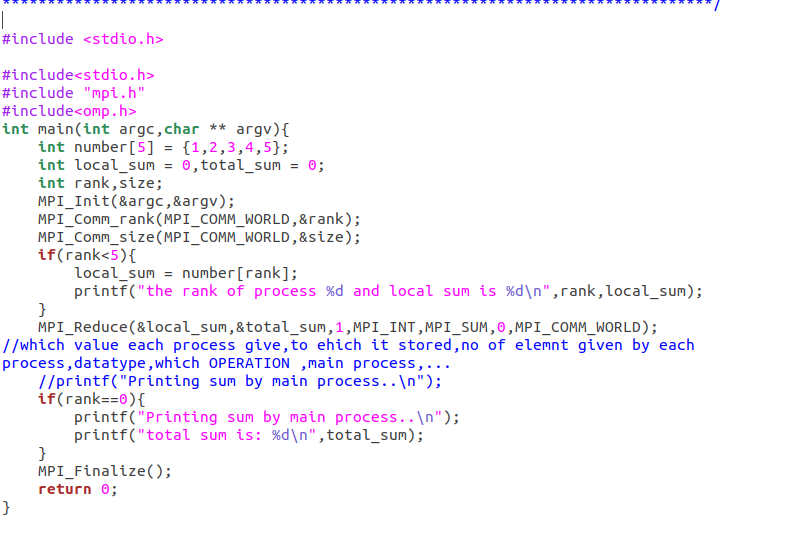
}

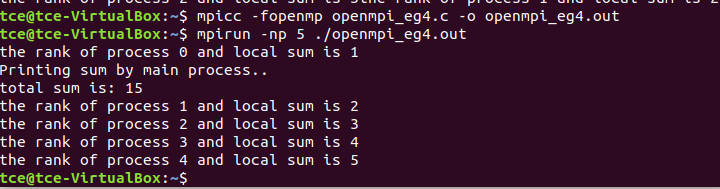
MPI\_Finalize();

return 0;

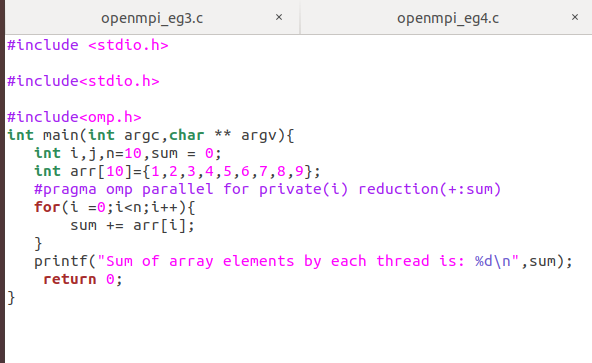
}//main

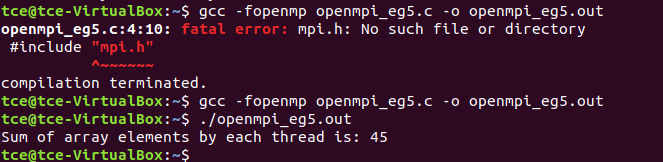
**Sum of first 5 digits:**



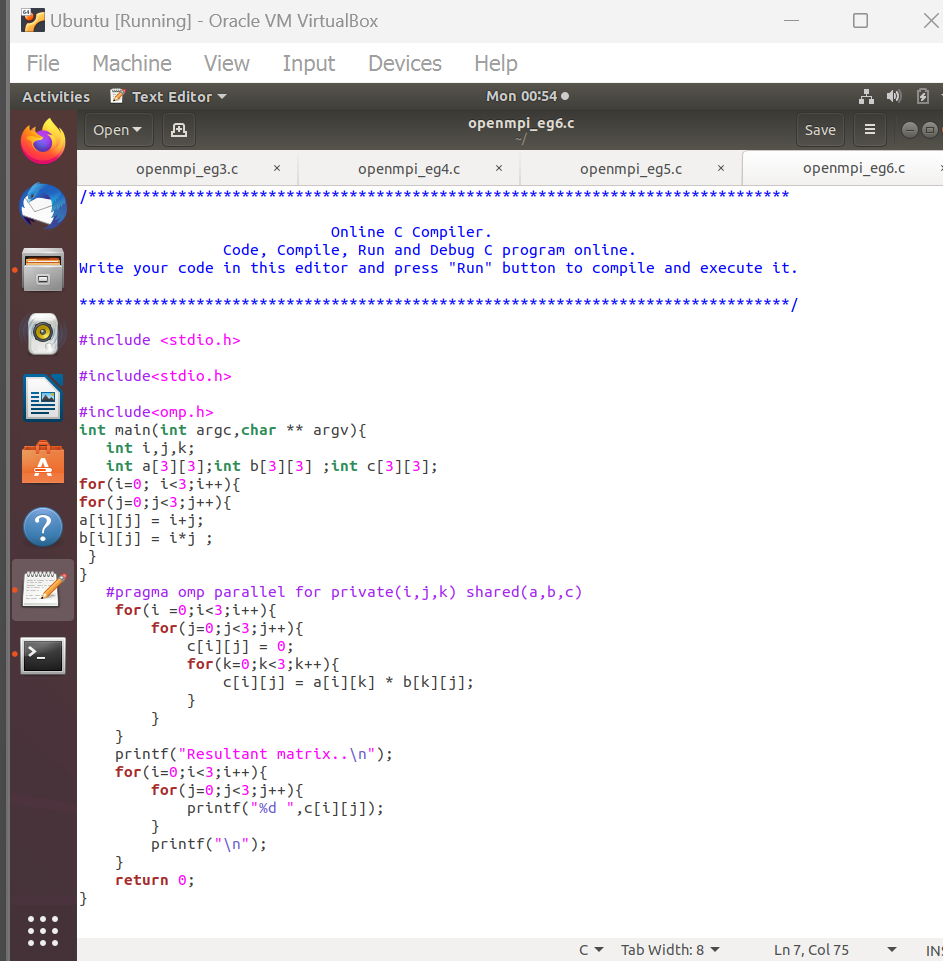


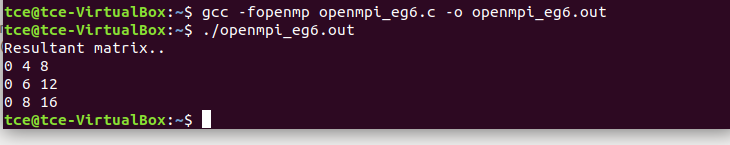
**Sum of n numberds in OMP**

****

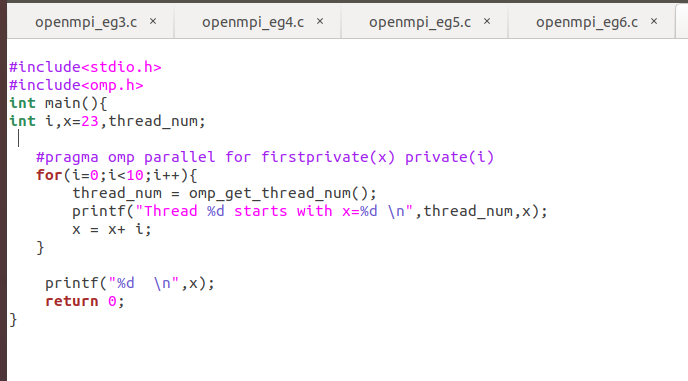
****

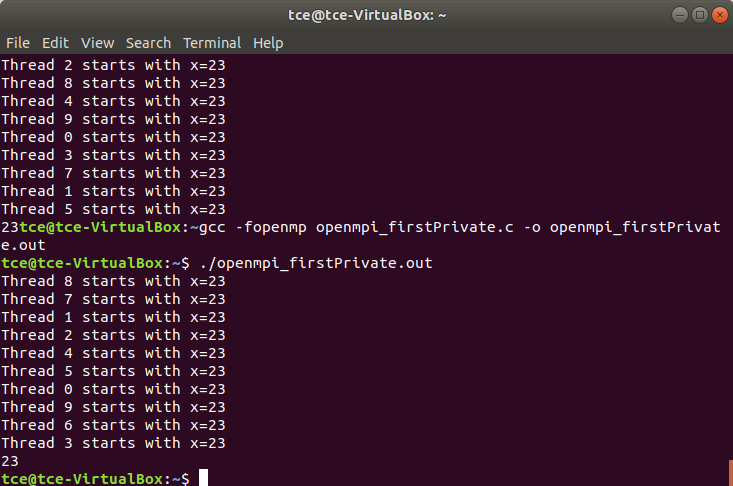
Matrix multilication in omp



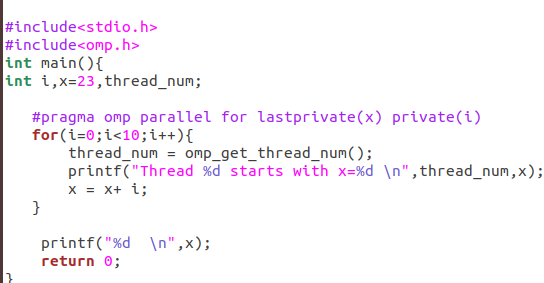


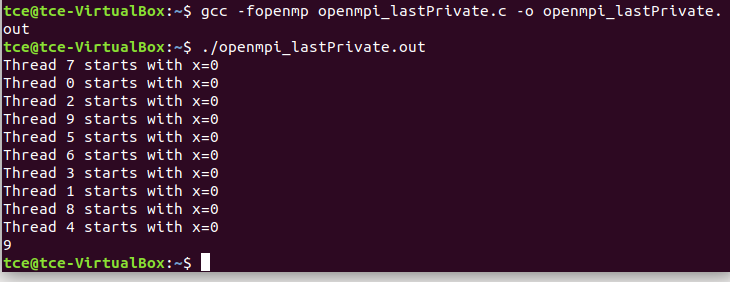
**First private**

****

****

**Lastprivate**

****

****

**Result:**

The mpi and omp programs for parallel execution have been successfully implemented….