**Concurrent and Parallel Programming**

Assignment -4

Name- Ashutosh Soni

Id- 2018ucp1505

Q1: Write a program which creates 5 threads and prints “Hello World” message using pthreads.

Ans:

**Program:**

// Including header files

#include<bits/stdc++.h>

#include<pthread.h>

#include<semaphore.h>

#include<unistd.h>

using namespace std;

#define N 5

// Hello World Function

void\* hello\_world(void \*arg){

int i = \*(int\*)arg;

cout<<"Hello World From thread "<<i+1<<endl;

}

int main(){

// Declaration

int item[N];

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

item[i]=i;

}

pthread\_t process[N];

pthread\_attr\_t attr;

// Attribute initialization....

pthread\_attr\_init(&attr);

pthread\_attr\_setdetachstate(&attr,PTHREAD\_CREATE\_JOINABLE);

// Creating pthread

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

int check=pthread\_create(&process[i],&attr,hello\_world,&item[i]);

if(check){

cout<<"Error in creating pthread"<<endl;

exit(-1);

}

}

// destroy attribute

pthread\_attr\_destroy(&attr);

// Joining

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

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

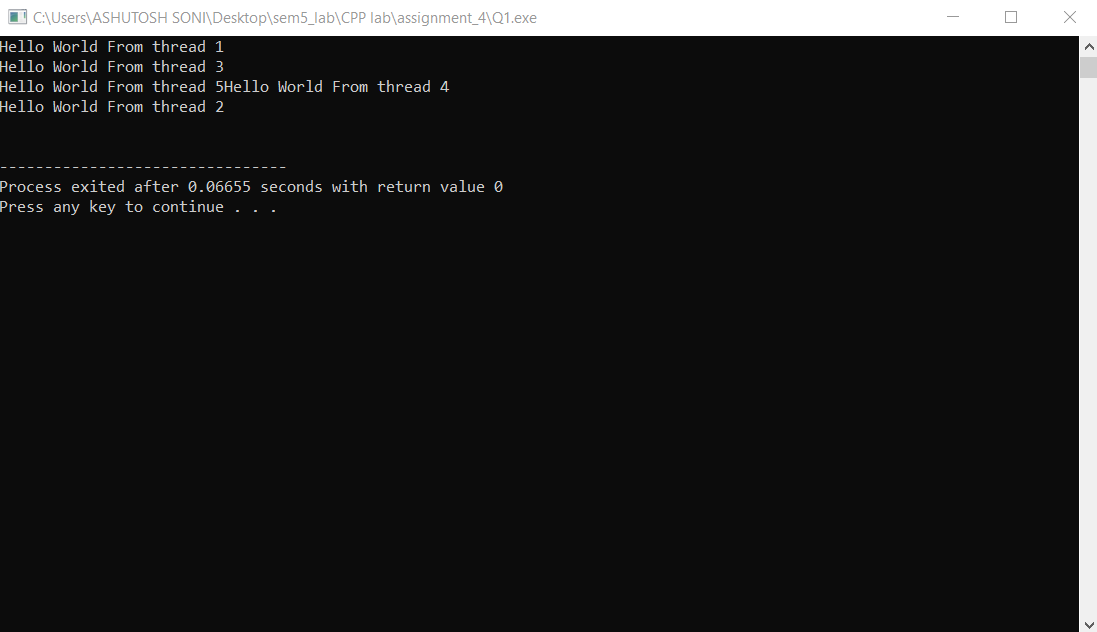
}

// Exiting pthread

pthread\_exit(NULL);

return 0;

}

**Output**:

Q2: Write a program using pthread to find out the sum of following series:

1+4+9+16+.......+ n.

Here main function should write the final output on screen. Main thread will create the child threads and child threads will find out the sum of series.

Ans:

**Program:**

// Write a program using pthread to find out the sum of the series

// 1+4+9+16+.....n

// Including header files

#include<bits/stdc++.h>

#include<pthread.h>

#include<unistd.h>

using namespace std;

// Decalaraion

long long result=0;

pthread\_mutex\_t Lock;

void\* helper(void\* arg){

int val=\*(int\*)arg;

// Evaluating the term

int term = val\*val;

// Locking the mutex varaible

pthread\_mutex\_lock(&Lock);

// Updating the result

result+=term;

// Unlocking the mutex variable

pthread\_mutex\_unlock(&Lock);

}

int main(){

// Decalaration

int N,r;

cout<<"Enter the value of N upto which you want summation "<<endl;

cin>> N;

vector<int> items(N,0);

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

items[i]=i+1;

}

// Declaration

pthread\_t process[N];

pthread\_attr\_t attr;

// attribute initialization

pthread\_attr\_init(&attr);

pthread\_attr\_setdetachstate(&attr,PTHREAD\_CREATE\_JOINABLE);

// creation of thread

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

r=pthread\_create(&process[i],&attr,helper,&items[i]);

if(r){

cout<<"Error in creating Thread"<<endl;

exit(-1);

}

}

// Destroying the attribute variable

pthread\_attr\_destroy(&attr);

// Joining of pthread

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

r=pthread\_join(process[i],NULL);

if(r){

cout<<"Error in joining Thread"<<endl;

exit(-1);

}

}

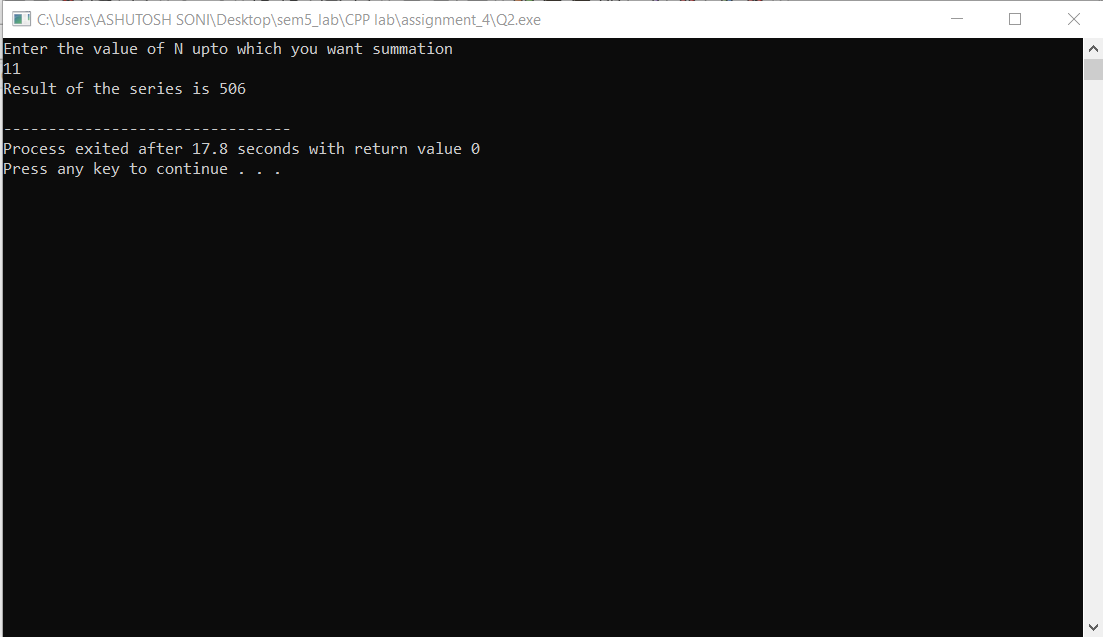
// Printing the value of result of the series

cout<<"Result of the series is "<<result<<endl;

pthread\_exit(NULL);

return 0;

}

**Output:**

Q3: Write a pthread program to find minimum value in an Integer array using Mutex.

Ans:

**Program:**

// Write a program to find minimum val

// Including header files

#include<bits/stdc++.h>

#include<pthread.h>

#include<unistd.h>

using namespace std;

// Decalaraion

int minimum=INT\_MAX;

pthread\_mutex\_t Lock;

void\* helper(void\* arg){

int val=\*(int\*)arg;

// Locking the mutex varaible

pthread\_mutex\_lock(&Lock);

// Updating the minimum if needed

if(minimum > val){

minimum = val;

}

// Unlocking the mutex variable

pthread\_mutex\_unlock(&Lock);

}

int main(){

// Decalaration

int N,r;

cout<<"Enter the value of N :"<<endl;

cin>> N;

vector<int> items(N,0);

cout<<"Now Enter the array: "<<endl;

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

cin>>items[i];

}

// Declaration

pthread\_t process[N];

pthread\_attr\_t attr;

// attribute initialization

pthread\_attr\_init(&attr);

pthread\_attr\_setdetachstate(&attr,PTHREAD\_CREATE\_JOINABLE);

// creation of thread

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

r=pthread\_create(&process[i],&attr,helper,&items[i]);

if(r){

cout<<"Error in creating Thread"<<endl;

exit(-1);

}

}

// Destroying the attribute variable

pthread\_attr\_destroy(&attr);

// Joining of pthread

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

r=pthread\_join(process[i],NULL);

if(r){

cout<<"Error in joining Thread"<<endl;

exit(-1);

}

}

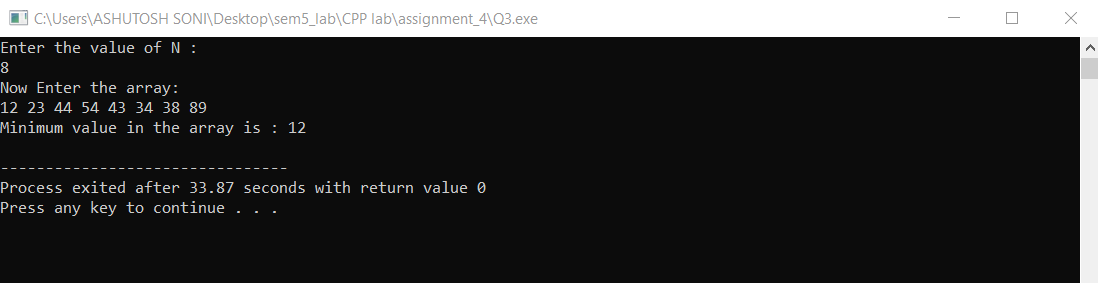
// Printing the minimum value of the array

cout<<"Minimum value in the array is : "<<minimum<<endl;

pthread\_exit(NULL);

return 0;

}

**Output:**

Q4: Write a program to compute the total of the values of the matrix using pthread and Mutex lock on global variable ‘total’.

Ans:

**Program:**

// Write a Program to compute the total of the matrix using pthread and Mutex lock on global varaible 'total'

// Including header files

#include<bits/stdc++.h>

#include<pthread.h>

#include<unistd.h>

using namespace std;

// Decalaraion

long long total=0;

pthread\_mutex\_t Lock;

void\* helper(void\* arg){

int val=\*(int\*)arg;

// Locking the mutex varaible

pthread\_mutex\_lock(&Lock);

// Updating the result

total+=val;

// Unlocking the mutex variable

pthread\_mutex\_unlock(&Lock);

}

int main(){

// Decalaration

int N,M,r;

cout<<"Enter the row and column of the matrix respectively"<<endl;

cin>>N>>M;

vector<vector<int> > items(N,vector<int>(M,0));

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

for(int j=0;j<M;j++){

cin>>items[i][j];

}

}

// Declaration

pthread\_t process[N][M];

pthread\_attr\_t attr;

// attribute initialization

pthread\_attr\_init(&attr);

pthread\_attr\_setdetachstate(&attr,PTHREAD\_CREATE\_JOINABLE);

// creation of thread

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

for(int j=0;j<M;j++){

r=pthread\_create(&process[i][j],&attr,helper,&items[i][j]);

if(r){

cout<<"Error in creating Thread"<<endl;

exit(-1);

}

}

}

// Destroying the attribute variable

pthread\_attr\_destroy(&attr);

// Joining of pthread

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

for(int j=0;j<M;j++){

r=pthread\_join(process[i][j],NULL);

if(r){

cout<<"Error in joining Thread"<<endl;

exit(-1);

}

}

}

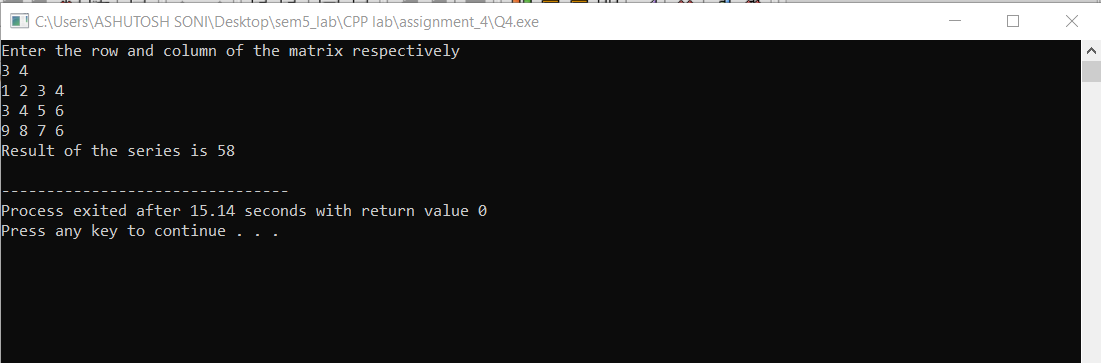
// Printing the total value of the matrix

cout<<"Result of the series is "<<total<<endl;

pthread\_exit(NULL);

return 0;

}

**Output:**

Q5: Take Employee Information (Name, ID, Designation, Salary of last 5 years) from user and find the % hike in salary of the employee using pthread and Mutex lock (use structure).

Ans:

**Program:**

// Including header file

#include<bits/stdc++.h>

#include<pthread.h>

#include<unistd.h>

using namespace std;

// Employee structure

struct Employee{

char name[100];

int ID;

char designation[1000];

vector<int> salary;

};

// Declaration of Elmpoyee

struct Employee info\_employee;

// Declaration of mutex variable

pthread\_mutex\_t Lock;

float hike=0;

void\* calculate\_hike(void \*arg){

int val=\*(int\*)arg;

// Finding percentage hike of the Employee

// Locking the mutex variable

pthread\_mutex\_lock(&Lock);

// Calculation of percenatage hike

hike = info\_employee.salary[val]-info\_employee.salary[val-1];

hike = (hike\*100)/info\_employee.salary[val-1];

// Giving user the output

cout<<"Percentage hike of "<<val<<" year is : "<<hike<<endl;

// Unlocking the mutex variable

pthread\_mutex\_unlock(&Lock);

return arg;

}

int main(){

// Taking Data of Employee

cout<<"Enter the name of the Employee: "<<endl;

cin>>info\_employee.name;

cout<<"Enter the Id of the Employee: "<<endl;

cin>>info\_employee.ID;

cout<<"Enter the Designation of the Employee: "<<endl;

cin>>info\_employee.designation;

cout<<"Enter the salary of the Employee of 5 years: "<<endl;

for(int i=0;i<5;i++){

int temp;

cin>>temp;

info\_employee.salary.push\_back(temp);

}

int id[4];

for(int i=0;i<4;i++){

id[i]=i+1;

}

// Declaration of Employee

pthread\_t find\_hike[4];

pthread\_attr\_t attr;

// Initialization of Attribute

pthread\_attr\_init(&attr);

pthread\_attr\_setdetachstate(&attr,PTHREAD\_CREATE\_JOINABLE);

// Creation of pthread

for(int i=0;i<4;i++){

int r1=pthread\_create(&find\_hike[i],&attr,calculate\_hike,&id[i]);

if(r1){

cout<<"Error in Creating Process"<<endl;

}

}

// Joining the pthread

for(int i=0;i<4;i++){

int r1=pthread\_join(find\_hike[i],NULL);

if(r1){

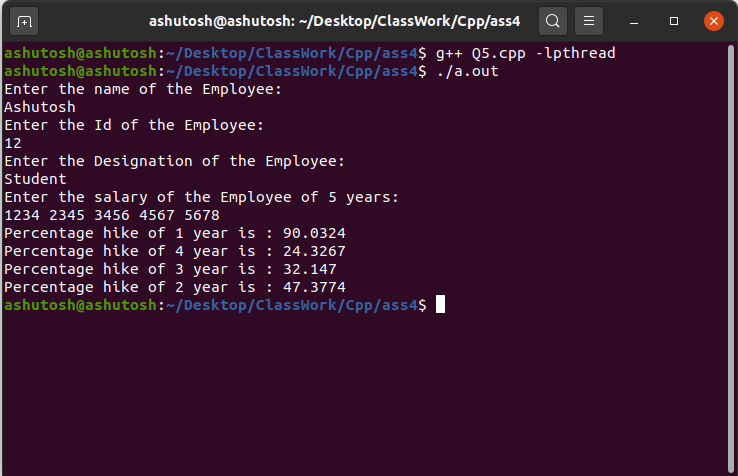
cout<<"Error in Joining Process"<<endl;

}

}

}

**Output:**



**Concurrent and Parallel Programming**

Assignment -4

Name- Ashutosh Soni

Id- 2018ucp1505

Q1: Write an OpenMP program to print each thread’s unique identifier.

Ans:

**Program:**

// Including Header File

#include<bits/stdc++.h>

#include<omp.h>

#include<unistd.h>

using namespace std;

int main(){

// Taking number of threads

int number\_of\_threads;

cout<<"Enter number of threads: "<<endl;

cin>>number\_of\_threads;

if(number\_of\_threads>10){

cout<<"Number of threads are large ...\n Exiting process... Start again"<<endl;

exit(-1);

}

// Setting number of threads for execution

omp\_set\_num\_threads(number\_of\_threads);

#pragma omp parallel

{

int threadid = omp\_get\_thread\_num();

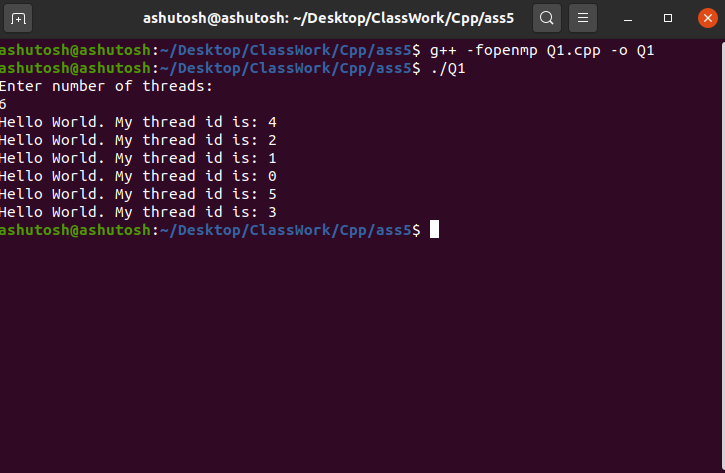
cout<<"Hello World. My thread id is: "<<threadid<<endl;

}

return 0;

}

**Output:**



Q2: Write MPI program to broadcast message "Hello world" to all the processes.

Ans:

**Program:**

// Including Header File

#include <mpi.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

using namespace std;

// Main Function

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

int id=2,myrank;

MPI\_Init(&argc,&argv);

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

printf("Identifier value in starting = %d for process = %d\n",id,myrank);

if(myrank==1){

id=12;

}

MPI\_Bcast(&id,1,MPI\_INT,1,MPI\_COMM\_WORLD);

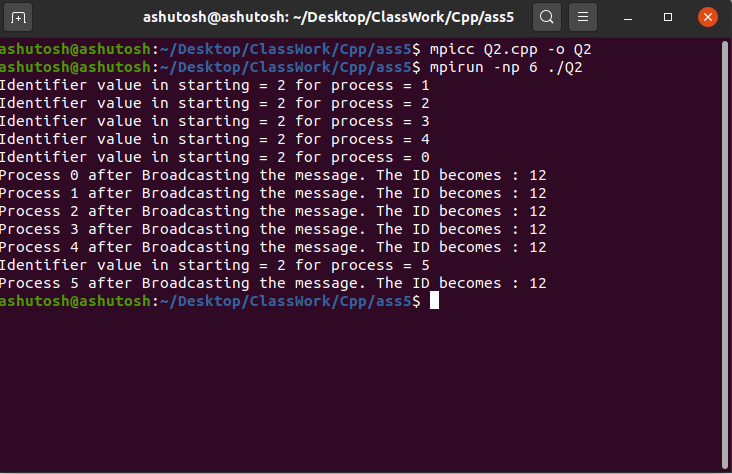
printf("Process %d after Broadcasting the message. The ID becomes : %d \n",myrank,id);

MPI\_Finalize();

return 0;

}

**Output:**



Q3: Write an OpenMP program to find Sum of Natural Numbers using OpenMP Parallel FOR directive and REDUCTION clause.

Ans:

**Program:**

//// Including Header File

#include<bits/stdc++.h>

#include<omp.h>

#include<unistd.h>

using namespace std;

int main()

{

// Taking input Upto which we want the sum

// Declaration

int N,i,chunk;

cout<<"Enter the number upto which you want to get sum"<<endl;

cin>>N;

// Value of all the items

int items[N];

chunk = 2;

// Just setting the items value of we can do directly but for sake we are

// filling items in parallel

#pragma omp parallel for

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

items[i]=i+1;

int result=0;

// Just getting sum in reduction clause

#pragma omp parallel for reduction(+ : result)

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

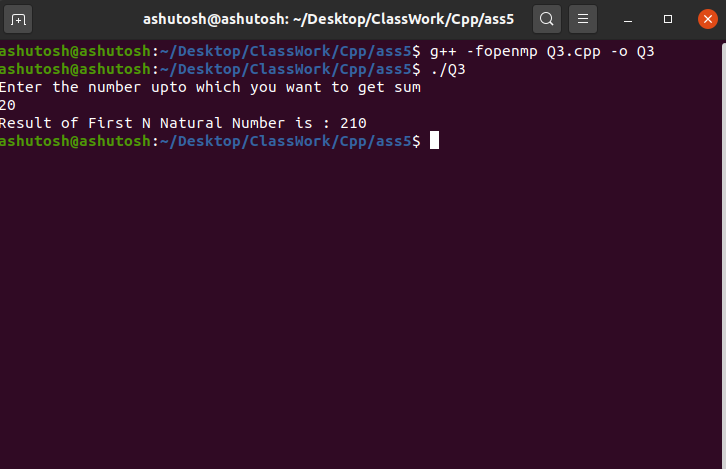
result+=items[i];

cout<<"Result of First N Natural Number is : "<<result<<endl;

return 0;

}

**Output:**



Q4: Write MPI program to find sum of n integers on a Parallel Computing System and use MPI collective blocking communication library calls.

Ans:

**Program:**

// Including header file

#include <mpi.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <time.h>

using namespace std;

// Main Funciton starting....

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

{

int number\_of\_elements;

int \*data=NULL;

int chunk\_size,own\_chunk\_size;

int \*chunk;

double time\_taken;

MPI\_Status status;

int result=0;

if(argc!=2){

printf("Desired number of arguments are not thier in argv....\n");

printf("1 number required...\n");

exit(-1);

}

int number\_of\_process,rank\_of\_process;

int rc=MPI\_Init(&argc,&argv);

if(rc!=MPI\_SUCCESS){

printf("Error in creating MPI program.\n Terminating......\n");

MPI\_Abort(MPI\_COMM\_WORLD,rc);

}

MPI\_Comm\_size(MPI\_COMM\_WORLD,&number\_of\_process);

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

if(rank\_of\_process==0){

// number of elements

number\_of\_elements=atoi(argv[1]);

// Computing chunk size

chunk\_size= (number\_of\_elements%number\_of\_process==0) ? number\_of\_elements/number\_of\_process : number\_of\_elements/(number\_of\_process-1);

data = (int \*)malloc(number\_of\_process\*chunk\_size \* sizeof(int));

for(int i=0;i<number\_of\_elements;i++){

data[i]=i+1;

}

// Padding data with zero

for(int i=number\_of\_elements;i<number\_of\_process\*chunk\_size;i++){

data[i]=0;

}

}

// Blocks all process until reach this point

MPI\_Barrier(MPI\_COMM\_WORLD);

// Starts Timer

time\_taken -= MPI\_Wtime();

// BroadCast the Size to all the process from root process

MPI\_Bcast(&number\_of\_elements,1,MPI\_INT,0,MPI\_COMM\_WORLD);

// Computing chunk size

chunk\_size= (number\_of\_elements%number\_of\_process==0) ? number\_of\_elements/number\_of\_process : number\_of\_elements/(number\_of\_process-1);

// Calculating total size of chunk according to bits

chunk=(int \*)malloc(chunk\_size \* sizeof(int));

// Scatter the chuck size data to all process

MPI\_Scatter(data,chunk\_size,MPI\_INT,chunk,chunk\_size,MPI\_INT,0,MPI\_COMM\_WORLD);

free(data);

data=NULL;

// Compute size of own chunk and then sort them using quick sort

own\_chunk\_size = (number\_of\_elements >= chunk\_size\*(rank\_of\_process+1)) ? chunk\_size : (number\_of\_elements - chunk\_size\*rank\_of\_process);

// Calculating the result of each process

for(int i=0;i<own\_chunk\_size;i++){

result+=chunk[i];

}

if(rank\_of\_process==0){

// Receiving their sum and the add it to main result

for(int i=1;i<number\_of\_process;i++){

int temp;

MPI\_Recv(&temp,1,MPI\_INT,i,0,MPI\_COMM\_WORLD,&status);

result+=temp;

}

}

else{

// Sending of sum of this part to desired process

MPI\_Send(&result,1,MPI\_INT,0,0,MPI\_COMM\_WORLD);

}

// Stop the timer

time\_taken+=MPI\_Wtime();

// Opening the other file as taken form input and writing it to the file and giving it as the output

if(rank\_of\_process==0){

printf("Sum of first n natural numbers is: %d\n",result);

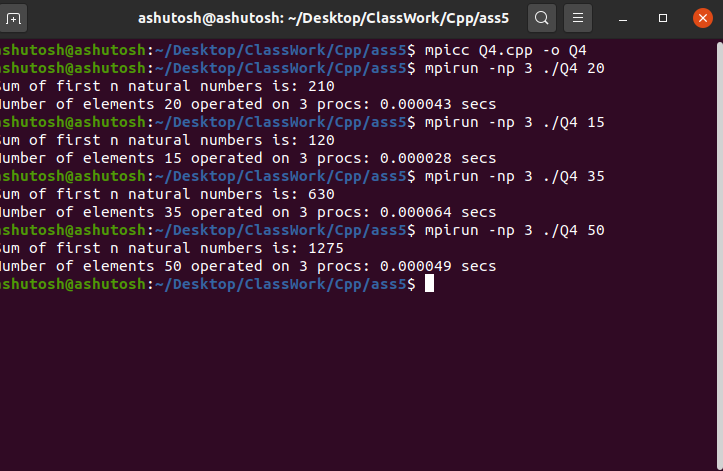
printf("Number of elements %d operated on %d procs: %f secs\n", number\_of\_elements, number\_of\_process, time\_taken);

}

MPI\_Finalize();

return 0;

}

**Output:**

Q5: Write MPI program for prefix sum (scan operation) calculation using MPI point-to-point blocking communication library calls.

Ans:

**Program:**

#include<stdio.h>

#include<math.h>

#include<mpi.h>

#include <stdlib.h>

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

int number\_of\_elements;

int \*data=NULL;

int chunk\_size,own\_chunk\_size;

int \*chunk;

FILE \*file=NULL;

double time\_taken;

MPI\_Status status;

if(argc!=2){

printf("Desired number of arguments are not thier in argv....\n");

printf("1 files containing inputs...\n");

exit(-1);

}

int number\_of\_process,rank\_of\_process;

int rc=MPI\_Init(&argc,&argv);

if(rc!=MPI\_SUCCESS){

printf("Error in creating MPI program.\n Terminating......\n");

MPI\_Abort(MPI\_COMM\_WORLD,rc);

}

MPI\_Comm\_size(MPI\_COMM\_WORLD,&number\_of\_process);

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

if(rank\_of\_process==0){

// Opening the file

file=fopen(argv[1],"r");

// Printing Error message if any

if(file==NULL){

printf("Error in opening file\n");

exit(-1);

}

// Reading number of Elements in file ... First Value in file is number of Elements

printf("Reading number of Elements From file ....\n");

fscanf(file,"%d",&number\_of\_elements);

printf("Number of Elements in the file is %d \n",number\_of\_elements );

// Computing chunk size

chunk\_size= (number\_of\_elements%number\_of\_process==0) ? number\_of\_elements/number\_of\_process : number\_of\_elements/(number\_of\_process-1);

data = (int \*)malloc(number\_of\_process\*chunk\_size \* sizeof(int));

// Reading the rest elements in which operation is being performed

printf("Reading the array from the file.......\n");

for(int i=0;i<number\_of\_elements;i++){

fscanf(file,"%d",&data[i]);

}

// Printing the array read from file

printf("Elements in the array is : \n");

for(int i=0;i<number\_of\_elements;i++){

printf("%d ",data[i]);

}

printf("\n");

fclose(file);

file=NULL;

}

// Blocks all process until reach this point

MPI\_Barrier(MPI\_COMM\_WORLD);

int prefix[number\_of\_elements];

for(int i=0;i<number\_of\_process;i++){

if(i==0){

prefix[i]=data[i];

MPI\_Send(&prefix[i],1,MPI\_INT,i+1,0,MPI\_COMM\_WORLD);

}

else if(i==number\_of\_process-1){

int temp;

MPI\_Recv(&temp,1,MPI\_INT,i-1,0,MPI\_COMM\_WORLD,&status);

prefix[i]=temp+data[i];

}

else{

int temp;

MPI\_Recv(&temp,1,MPI\_INT,i-1,0,MPI\_COMM\_WORLD,&status);

prefix[i]=temp+data[i];

MPI\_Send(&prefix[i],1,MPI\_INT,i+1,0,MPI\_COMM\_WORLD);

}

printf("%d ",prefix[i] );

}

for(int i=0;i<number\_of\_process;i++){

printf("%d ",prefix[i] );

}

MPI\_Finalize();

return 0;

}

-------------------------------------------------------------------------------------------------