**Assignment no:3**

**arr\_sum\_mpi.c**

**#include<stdio.h>**

**#include<mpi.h>**

**#define arr\_size 15**

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

**int rank, size;**

**MPI\_Init(&argc, &argv);**

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

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

**//Code that will execute inside process 0 or rank 0**

**if(rank == 0){**

**int arr[]= {12,4,6,3,21,15,3,5,7,8,9,1,5,3,5};**

**int global\_sum = 0, local\_sum = 0, recv\_local\_sum;**

**//If the array size is perfectly divisible by number of process.**

**if(arr\_size%size == 0){**

**int array\_element\_per\_process = arr\_size/size;**

**int sub\_arr[array\_element\_per\_process];**

**for(int i=1; i<size; i++){**

**//Copying the sub array**

**for(int j=0; j<array\_element\_per\_process;j++){**

**sub\_arr[j] = arr[i\*array\_element\_per\_process+j];**

**}**

**//Sending array chunk of equal size to all the process.**

**MPI\_Send(sub\_arr, array\_element\_per\_process, MPI\_INT, i, 1, MPI\_COMM\_WORLD);**

**MPI\_Send(&array\_element\_per\_process, 1, MPI\_INT, i, 1, MPI\_COMM\_WORLD);**

**}**

**//Calculating the local sum of rank 0 itself**

**for(int j=0; j<array\_element\_per\_process; j++){**

**local\_sum += arr[j];**

**}**

**printf("Rank %d: local sum: %d\n", rank, local\_sum);**

**global\_sum += local\_sum;**

**//When the array size is not perfectly divisible by number of process.**

**}else{**

**int array\_element\_per\_process = arr\_size/size + 1;**

**int sub\_arr[array\_element\_per\_process];**

**for(int i=1; i<size; i++){**

**if(i == size - 1){**

**//last sub array will have the size less than other process array size**

**int total\_array\_size\_of\_last\_process = arr\_size - array\_element\_per\_process \* i;**

**for(int j=0; j< total\_array\_size\_of\_last\_process; j++){**

**sub\_arr[j] = arr[i\*array\_element\_per\_process+j];**

**}**

**MPI\_Send(&sub\_arr, total\_array\_size\_of\_last\_process, MPI\_INT, i, 1, MPI\_COMM\_WORLD);**

**MPI\_Send(&total\_array\_size\_of\_last\_process, 1, MPI\_INT, i, 1, MPI\_COMM\_WORLD);**

**}else{**

**//Copying the sub array**

**for(int j=0; j<array\_element\_per\_process;j++){**

**sub\_arr[j] = arr[i\*array\_element\_per\_process+j];**

**}**

**MPI\_Send(&sub\_arr, array\_element\_per\_process, MPI\_INT, i, 1, MPI\_COMM\_WORLD);**

**MPI\_Send(&array\_element\_per\_process, 1, MPI\_INT, i, 1, MPI\_COMM\_WORLD);**

**}**

**}**

**//Calculating the local sum of rank 0 itself**

**for(int j=0; j<array\_element\_per\_process; j++){**

**local\_sum += arr[j];**

**}**

**printf("Rank %d: local sum: %d\n", rank, local\_sum);**

**global\_sum += local\_sum;**

**}**

**//calculating the global sum of the array**

**//Receving the local sum from the other process and updating the global sum**

**for(int i=1; i<size; i++){**

**MPI\_Recv(&recv\_local\_sum, 1, MPI\_INT, i, 1, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);**

**global\_sum += recv\_local\_sum;**

**}**

**//Printing the output**

**printf("The sum of the array is %d\n", global\_sum);**

**//Code that will get executed inside other than process 0 or rank 0.**

**}else{**

**//The other process will receive the chunck of array**

**int array\_element\_per\_process = arr\_size/size + 1;**

**int recv\_sub\_arr[array\_element\_per\_process];**

**int recv\_array\_element\_per\_process, local\_sum = 0;**

**MPI\_Recv(recv\_sub\_arr, recv\_array\_element\_per\_process, MPI\_INT, 0, 1, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);**

**MPI\_Recv(&recv\_array\_element\_per\_process, 1, MPI\_INT, 0, 1, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);**

**//Calculating local sum for the sub array**

**for(int j=0; j<recv\_array\_element\_per\_process; j++){**

**local\_sum += recv\_sub\_arr[j];**

**}**

**//Printing the local sum**

**printf("Rank %d: local sum: %d\n", rank, local\_sum);**

**//Sending back the local sum to the rank 0 or process 0.**

**MPI\_Send(&local\_sum, 1, MPI\_INT, 0, 1, MPI\_COMM\_WORLD);**

**}**

**MPI\_Finalize();**

**return 0;**

**}**

**arr\_sum.c**

#include <mpi.h>

#include <stdio.h>

#include <stdlib.h>

#define ARRAY\_SIZE 16

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

int rank, size;

int sum = 0;

int array[ARRAY\_SIZE];

// Initialize MPI

MPI\_Init(&argc, &argv);

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

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

// Populate the array on the root process

if (rank == 0) {

for (int i = 0; i < ARRAY\_SIZE; i++) {

array[i] = i + 1;

}

}

// Scatter the array to all processes

int subarray\_size = ARRAY\_SIZE / size;

int subarray[subarray\_size];

MPI\_Scatter(array, subarray\_size, MPI\_INT, subarray, subarray\_size, MPI\_INT, 0, MPI\_COMM\_WORLD);

// Sum the local elements

int local\_sum = 0;

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

local\_sum += subarray[i];

}

// Display the local sum of each process

printf("Process %d local sum is %d\n", rank, local\_sum);

// Reduce the local sums to get the final sum on the root process

MPI\_Reduce(&local\_sum, &sum, 1, MPI\_INT, MPI\_SUM, 0, MPI\_COMM\_WORLD);

// Print the result on the root process

if (rank == 0) {

printf("The sum of the elements is %d\n", sum);

}

// Finalize MPI

MPI\_Finalize();

return 0;

}

**ring\_token.c**

#include<stdio.h>

#include<conio.h>

#include<dos.h>

#include<time.h>

void main(){

int cs=0,pro=0;

double run=5;

char key='a';

time\_t t1,t2;

printf("Press a key(except q) to enter a process into critical section.");

printf(" \nPress q at any time to exit.");

t1 = time(NULL) - 5;

while(key!='q')

{

while(!kbhit())

if(cs!=0)

{

t2 = time(NULL);

if(t2-t1 > run)

{

printf("Process%d ",pro-1);

printf(" exits critical section.\n");

cs=0;

}

}

key = getch();

if(key!='q')

{

if(cs!=0)

printf("Error: Another process is currently executing critical section Please wait till its execution is over.\n");

else

{

printf("Process %d ",pro);

printf(" entered critical section\n");

cs=1;

pro++;

t1 = time(NULL);

}

}

}

}

**OUTPUT:**

