Distributed Systems Lab-Evaluation-1

Suryanarayan.B

CB.EN.U4CSE19056

1) #include <stdio.h>

#include<iostream>

#include <mpi.h>

#define send\_data\_tag 2001

#define return\_data\_tag 2002

int array[50];

int array2[50];

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

{

int sum, partial\_sum;

MPI\_Status status;

int id, an\_id;

int i;

int procs;

int rows, rows\_receive, avg\_row\_process;

int rows\_received, start\_row, end\_row;

int rows\_send, sender;

MPI\_Init(&argc, &argv);

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

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

if (id == 0) {

scanf\_s("%i", &rows);

avg\_row\_process = rows / procs;

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

array[i] = i + 1;

}

for (an\_id = 1; an\_id < procs; an\_id++) {

start\_row = an\_id \* avg\_row\_process + 1;

end\_row = (an\_id + 1) \* avg\_row\_process;

if ((rows - end\_row) < avg\_row\_process)

end\_row = rows - 1;

rows\_send = end\_row - start\_row + 1;

MPI\_Send(&rows\_send, 1, MPI\_INT,

an\_id, send\_data\_tag, MPI\_COMM\_WORLD);

MPI\_Send(&array[start\_row], rows\_send, MPI\_INT,

an\_id, send\_data\_tag, MPI\_COMM\_WORLD);

}

sum = 0;

for (i = 0; i < avg\_row\_process + 1; i++) {

sum += array[i];

}

printf("The sum %i is calculated by root process \n", sum);

for (an\_id = 1; an\_id < procs; an\_id++) {

MPI\_Recv(&partial\_sum, 1, MPI\_LONG, MPI\_ANY\_SOURCE,

return\_data\_tag, MPI\_COMM\_WORLD, &status);

sender = status.MPI\_SOURCE;

printf("The Partial sum %i is calculated by process %i\n", partial\_sum, sender);

sum += partial\_sum;

}

printf("The total sum obtained after adding all Partial Sum is: %i\n", sum);

}

else {

MPI\_Recv(&rows\_receive, 1, MPI\_INT,

0, send\_data\_tag, MPI\_COMM\_WORLD, &status);

MPI\_Recv(&array2, rows\_receive, MPI\_INT,

0, send\_data\_tag, MPI\_COMM\_WORLD, &status);

rows\_received = rows\_receive;

partial\_sum = 0;

for (i = 0; i < rows\_received; i++) {

partial\_sum += array2[i];

}

MPI\_Send(&partial\_sum, 1, MPI\_LONG, 0,

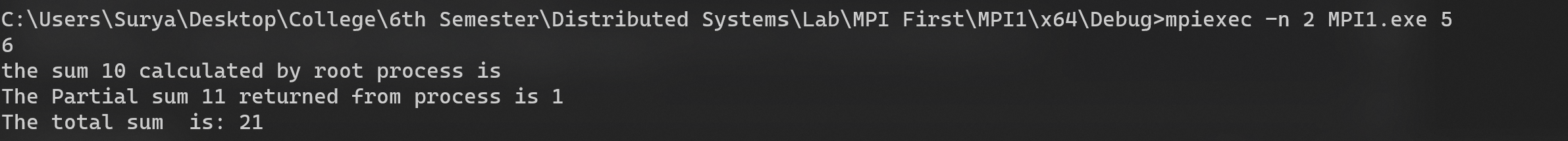
return\_data\_tag, MPI\_COMM\_WORLD);

}

MPI\_Finalize();

}

Output:



Working

* We create (n-1) number of slave process, where n is the number we give while executing in the terminal
* We share the sum among the slave processes and get their results and finally add the results in the root process and display the output
* After getting the number of elements of the array as input from the user, we create an array with elements from 1 to N+1.
* We divide and send the elements to the slave processes and get their results to the root process
* MPISend is used the send the data to master/slave process
* MPIRecieve is used the receive the data that is sent by the master/slave process

2) #include <stdio.h>

#include<iostream>

#include <mpi.h>

#define send\_data\_tag 2001

#define return\_data\_tag 2002

int array[50];

int array2[50];

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

{

int sum, partial\_sum;

MPI\_Status status;

int id, an\_id;

int i;

int procs;

int rows, rows\_receive, avg\_row\_process;

int rows\_received, start\_row, end\_row;

int rows\_send, sender;

MPI\_Init(&argc, &argv);

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

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

if (id == 0) {

scanf\_s("%i", &rows);

avg\_row\_process = rows / procs;

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

array[i] = (i + 1)\*(i+1);

}

for (an\_id = 1; an\_id < procs; an\_id++) {

start\_row = an\_id \* avg\_row\_process + 1;

end\_row = (an\_id + 1) \* avg\_row\_process;

if ((rows - end\_row) < avg\_row\_process)

end\_row = rows - 1;

rows\_send = end\_row - start\_row + 1;

MPI\_Send(&rows\_send, 1, MPI\_INT,

an\_id, send\_data\_tag, MPI\_COMM\_WORLD);

MPI\_Send(&array[start\_row], rows\_send, MPI\_INT,

an\_id, send\_data\_tag, MPI\_COMM\_WORLD);

}

sum = 0;

for (i = 0; i < avg\_row\_process + 1; i++) {

sum += array[i];

}

printf("The sum %i is calculated by root process \n", sum);

for (an\_id = 1; an\_id < procs; an\_id++) {

MPI\_Recv(&partial\_sum, 1, MPI\_LONG, MPI\_ANY\_SOURCE,

return\_data\_tag, MPI\_COMM\_WORLD, &status);

sender = status.MPI\_SOURCE;

printf("The Partial sum %i is calculated by process %i\n", partial\_sum, sender);

sum += partial\_sum;

}

printf("The total sum obtained after adding all Partial Sum is: %i\n", sum);

}

else {

MPI\_Recv(&rows\_receive, 1, MPI\_INT,

0, send\_data\_tag, MPI\_COMM\_WORLD, &status);

MPI\_Recv(&array2, rows\_receive, MPI\_INT,

0, send\_data\_tag, MPI\_COMM\_WORLD, &status);

rows\_received = rows\_receive;

partial\_sum = 0;

for (i = 0; i < rows\_received; i++) {

partial\_sum += array2[i];

}

MPI\_Send(&partial\_sum, 1, MPI\_LONG, 0,

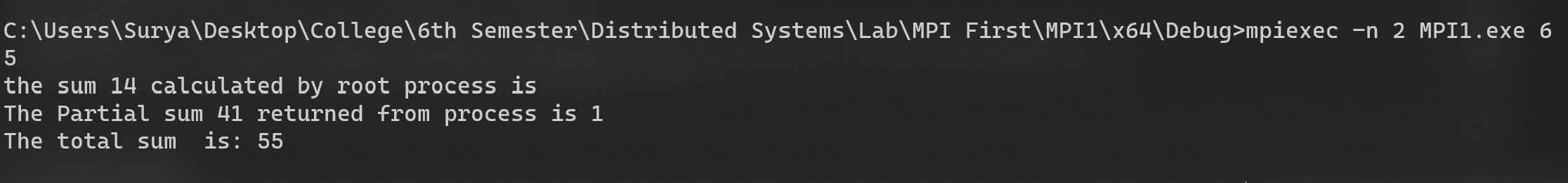
return\_data\_tag, MPI\_COMM\_WORLD);

}

MPI\_Finalize();

}

Output:



Working

* We create (n-1) number of slave process, where n is the number we give while executing in the terminal
* We share the sum of squares among the slave processes and get their results and finally add the results in the root process and display the output
* After getting the number of elements of the array as input from the user, we create an array with elements from 1 to N+1.
* We divide and send the elements to the slave processes and get their results to the root process
* MPISend is used the send the data to master/slave process
* MPIRecieve is used the receive the data that is sent by the master/slave process