Sriram.S and Suryanarayan.B

CB.EN.U4CSE19055 and CB.EN.U4CSE19056

Distributed Systems

Lab-Assignment-1

Question 1: Write an MPI Point –to -point C Program involving only 2 processes, where Node 1 sends the array of numbers to Node 0 in the communicating world.

Ans: #include <mpi.h>

#include <stdio.h>

#include <iostream>

using namespace std;

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

MPI\_Init(&argc, &argv);

int world\_size;

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

int world\_rank;

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

if (world\_rank == 0)

{

int b[5];

MPI\_Recv(b, 5, MPI\_INT, 1, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

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

cout << b[i] << ",";

}

else

{

int a[10] = { 1,3,4,2,8,6,10,100,500 };

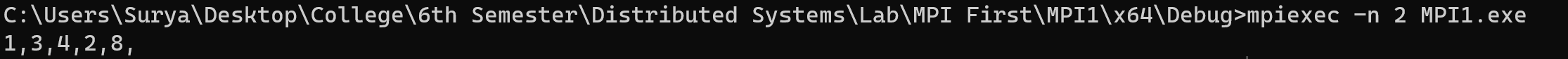
MPI\_Send(a, 5, MPI\_INT, 0, 0, MPI\_COMM\_WORLD);

}

MPI\_Finalize();

}

Output:



Question 2: Write an MPI Point-to –point C program involving ‘n’ number of nodes in communicating world. The objective of this program is to find the sum of numbers from 1 to 1000. Each node in the communicating world should be assigned disjoint set of numbers between 1 and 1000. Each node should compute the sum of numbers assigned to it and communicate it to root node(I.e., node 0) to compute the overall summation value.Eg: Suppose Number of nodes (n) = 5Node 0Assigned with 1 to 200 Node 1 Assigned with 201 to 400 and so on.Final sum accumulated at Node 0 = 500500.

Ans: #include<stdio.h>

#include "mpi.h"

#include<iostream>

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

{

MPI\_Init(&argc, &argv);

//MPI\_Status status;

int size;

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

int rank;

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

int i;

int sum1 = 0, sum2 = 0, sum3 = 0, sum4 = 0, sum5 = 0, sum;

if (rank == 0)

{

for (i = 1; i <= 200; i++)

{

sum1 = sum1 + i;

}

MPI\_Recv(&sum2, 1, MPI\_INT, 1, 6, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

MPI\_Recv(&sum3, 1, MPI\_INT, 2, 7, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

MPI\_Recv(&sum4, 1, MPI\_INT, 3, 8, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

MPI\_Recv(&sum5, 1, MPI\_INT, 4, 9, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

sum = sum1 + sum2 + sum3 + sum4 + sum5;

printf("This is Parent number %d from processors %d and got the final sum equal to %d", rank, size, sum);

}

else if (rank == 1)

{

for (i = 201; i <= 400; i++)

{

sum2 = sum2 + i;

}

MPI\_Send(&sum2, 1, MPI\_INT, 0, 6, MPI\_COMM\_WORLD);

printf("This is rank number %d sending the sum of 201 to 400 =>%d to rank 0", rank, sum2);

}

else if (rank == 2)

{

for (i = 401; i <= 600; i++)

{

sum3 = sum3 + i;

}

MPI\_Send(&sum3, 1, MPI\_INT, 0, 7, MPI\_COMM\_WORLD);

printf("This is rank number %d sending the sum of =>%d to rank 0", rank, sum3);

}

else if (rank == 3)

{

for (i = 601; i <= 800; i++)

{

sum4 = sum4 + i;

}

MPI\_Send(&sum4, 1, MPI\_INT, 0, 8, MPI\_COMM\_WORLD);

printf("This is rank number %d sending the sum of=>%d to rank 0", rank, sum4);

}

else if (rank == 4)

{

for (i = 801; i <= 1000; i++)

{

sum5 = sum5 + i;

}

MPI\_Send(&sum5, 1, MPI\_INT, 0, 9, MPI\_COMM\_WORLD);

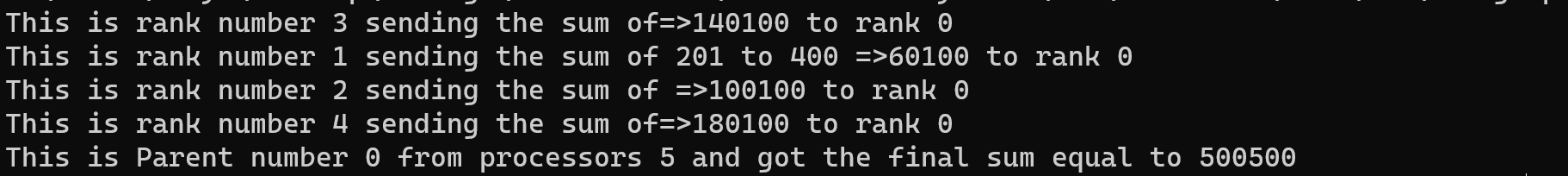
printf("This is rank number %d sending the sum of=>%d to rank 0", rank, sum5);

}

MPI\_Finalize();

}

Output:



Question 3:Analyse the below code and write your observations

Ans: #include <mpi.h>

#include <stdio.h>

#include <stdlib.h>

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

int world\_rank;

int world\_size;

int token;

MPI\_Init(NULL,NULL);

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

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

if (world\_rank != 0) {

MPI\_Recv(&token, 1, MPI\_INT, world\_rank - 1, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

printf("Process %d received token %d from process%d\n", world\_rank, token, world\_rank - 1);

}

else

{

token = - 1;

}

MPI\_Send(&token, 1, MPI\_INT, (world\_rank + 1) % world\_size, 0, MPI\_COMM\_WORLD);

if (world\_rank == 0) {

MPI\_Recv(&token, 1, MPI\_INT, world\_size - 1, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

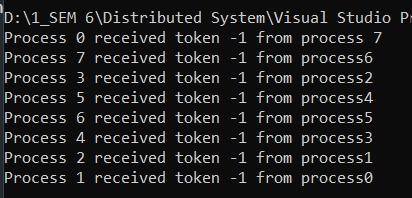
printf("Process %d received token %d from process %d\n", world\_rank, token, world\_size - 1);

}

MPI\_Finalize();

}

Output:



Inference:

The code passes the tokens to processes that are one rank higher than the current process and finally comes back to initial process.

We can say that process 0 will pass to 1, then 1 to 2, and so on until we reach n-1 and then return to 0.

Question 4: Write a MPI collective C program using the functions Bcast and Barrierinvolving n nodes in communication. Each node has part of the whole data, which the program mustbroadcast their part of the data to all the other nodes.

Ans: #include <stdio.h>

#include <stdlib.h>

#include <mpi.h>

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

{

MPI\_Init(&argc, &argv);

// Get my rank

int my\_rank;

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

printf("[MPI process %d] I wait on the barrier.\n", my\_rank);

MPI\_Barrier(MPI\_COMM\_WORLD);

printf("[MPI process %d] I know all the processes have waited on this barrier.\n", my\_rank);

int broadcast\_root = 0;

int buffer;

if (my\_rank == broadcast\_root)

{

buffer = 45678;

printf("[MPI process %d] I am the broadcast root, and I send the value of %d.\n", my\_rank, buffer);

}

MPI\_Bcast(&buffer, 1, MPI\_INT, broadcast\_root, MPI\_COMM\_WORLD);

if (my\_rank != broadcast\_root)

{

printf("[MPI process %d] I am the broadcast receiver, and obtained the value of %d.\n", my\_rank, buffer);

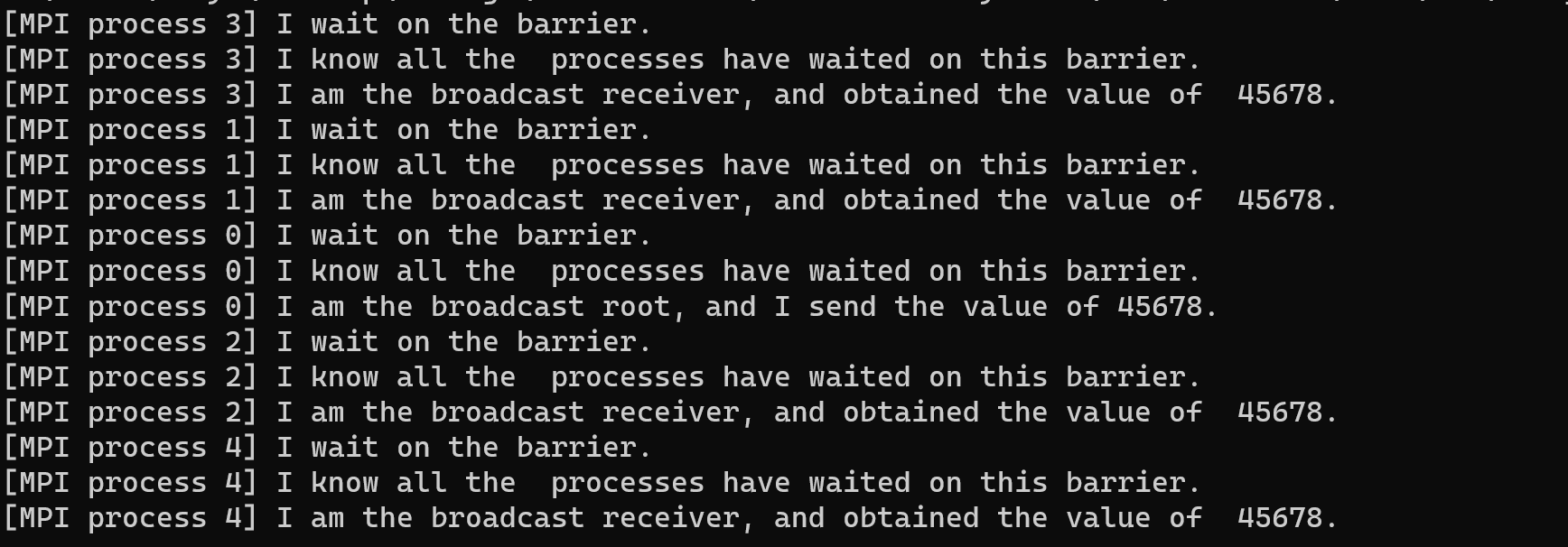
}

MPI\_Finalize();

return EXIT\_SUCCESS;

}

Output:



Question 5: Write a MPI collective C program that computes the average of an array of elements in parallel using (MPI\_Scatter and MPI\_Gather):

Ans: #include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <mpi.h>

#include <assert.h>

float\* create\_rand\_nums(int num\_elements) {

float\* rand\_nums = (float\*)malloc(sizeof(float) \* num\_elements);

assert(rand\_nums != NULL);

int i;

for (i = 0; i < num\_elements; i++) {

rand\_nums[i] = (rand() / (float)RAND\_MAX);

}

return rand\_nums;

}

float compute\_avg(float\* array, int num\_elements) {

float sum = 0.f;

int i;

for (i = 0; i < num\_elements; i++) {

sum += array[i];

}

return sum / num\_elements;

}

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

if (argc != 2) {

fprintf(stderr, "Usage: avg num\_elements\_per\_proc\n");

exit(1);

}

int num\_elements\_per\_proc = atoi(argv[1]);

srand(time(NULL));

MPI\_Init(NULL, NULL);

int world\_rank;

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

int world\_size;

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

float\* rand\_nums = NULL;

if (world\_rank == 0) {

rand\_nums = create\_rand\_nums(num\_elements\_per\_proc \* world\_size);

}

float\* sub\_rand\_nums = (float\*)malloc(sizeof(float) \* num\_elements\_per\_proc);

assert(sub\_rand\_nums != NULL);

MPI\_Scatter(rand\_nums, num\_elements\_per\_proc, MPI\_FLOAT, sub\_rand\_nums,

num\_elements\_per\_proc, MPI\_FLOAT, 0, MPI\_COMM\_WORLD);

float sub\_avg = compute\_avg(sub\_rand\_nums, num\_elements\_per\_proc);

float\* sub\_avgs = NULL;

if (world\_rank == 0) {

sub\_avgs = (float\*)malloc(sizeof(float) \* world\_size);

assert(sub\_avgs != NULL);

}

MPI\_Gather(&sub\_avg, 1, MPI\_FLOAT, sub\_avgs, 1, MPI\_FLOAT, 0, MPI\_COMM\_WORLD);

if (world\_rank == 0) {

float avg = compute\_avg(sub\_avgs, world\_size);

printf("Avg of all elements is %f\n", avg);

float original\_data\_avg =

compute\_avg(rand\_nums, num\_elements\_per\_proc \* world\_size);

printf("Avg computed across original data is %f\n", original\_data\_avg);

}

if (world\_rank == 0) {

free(rand\_nums);

free(sub\_avgs);

}

free(sub\_rand\_nums);

MPI\_Barrier(MPI\_COMM\_WORLD);

MPI\_Finalize();

}

Output:

