**PP LAB WEEK-7**

# DSE VI-A2 Divansh Prasad 210968140

1) Write a MPI program using synchronous send. The sender process sends a word to the receiver. The second process receives the word, toggles each letter of the word and sends it back to the first process. Both processes use synchronous send operations.

#include <stdio.h>

#include <string.h>

#include <mpi.h>

#define MAX\_WORD\_LENGTH 100

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

int myrank, numprocs;

char word[MAX\_WORD\_LENGTH];

MPI\_Init(&argc, &argv);

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

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

if (numprocs != 2) {

if (myrank == 0) {

printf("This program requires exactly 2 processes.\n");

}

MPI\_Finalize();

return 1;

}

if (myrank == 0) {

strcpy(word, "hello");

MPI\_Ssend(word, strlen(word) + 1, MPI\_CHAR, 1, 0, MPI\_COMM\_WORLD);

printf("Sender: Sent word: %s\n", word);

MPI\_Recv(word, MAX\_WORD\_LENGTH, MPI\_CHAR, 1, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

printf("Sender: Received toggled word: %s\n", word);

} else if (myrank == 1) {

MPI\_Recv(word, MAX\_WORD\_LENGTH, MPI\_CHAR, 0, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

printf("Receiver: Received word: %s\n", word);

int i;

for (i = 0; word[i] != '\0'; ++i) {

if (word[i] >= 'a' && word[i] <= 'z') {

word[i] = word[i] - 32;

} else if (word[i] >= 'A' && word[i] <= 'Z') {

word[i] = word[i] + 32;

}

}

MPI\_Ssend(word, strlen(word) + 1, MPI\_CHAR, 0, 0, MPI\_COMM\_WORLD);

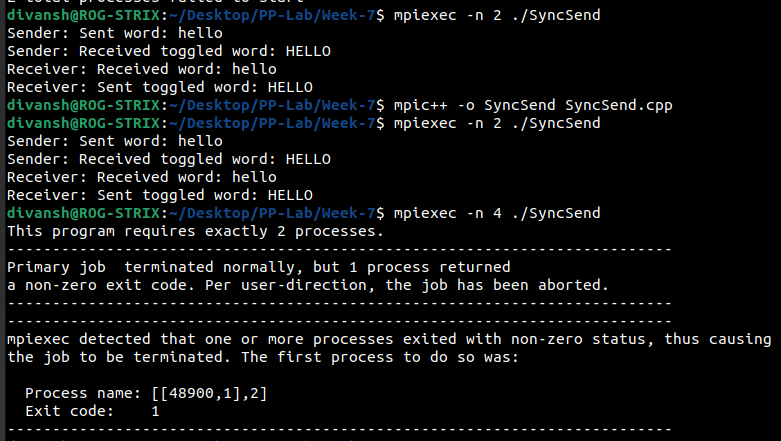
printf("Receiver: Sent toggled word: %s\n", word);

}

MPI\_Finalize();

return 0;

}



2) Write a MPI program where the master process (process 0) sends a number to each of the slaves and the slave processes receive the number and print it. Use standard send.

#include <stdio.h>

#include <mpi.h>

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

int myrank, numprocs;

int number;

MPI\_Init(&argc, &argv);

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

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

if (numprocs < 2) {

printf("This program requires at least 2 processes.\n");

MPI\_Finalize();

return 1;}

if (myrank == 0) {

for (int dest = 1; dest < numprocs; ++dest) {

number = dest;

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

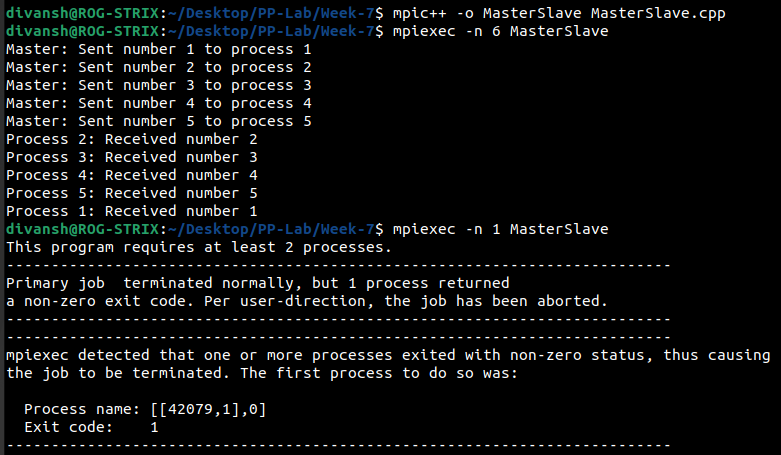
printf("Master: Sent number %d to process %d\n", number, dest);}

} else {

MPI\_Recv(&number, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

printf("Process %d: Received number %d\n", myrank, number);}

MPI\_Finalize(); return 0;}



3) Write a MPI program to read N elements of the array in the root process (process 0) where N is equal to the total number of processes. The root process sends one value to each of the slaves. Let an even ranked process find the square of the received element and odd ranked element find the cube of the received element. Use Buffered send.

#include <stdio.h>

#include <mpi.h>

#define ARRAY\_SIZE 100

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

int myrank, numprocs;

int data;

int result;

MPI\_Init(&argc, &argv);

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

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

if (numprocs < 2) {

printf("This program requires at least 2 processes.\n");

MPI\_Finalize();

return 1;

}

if (myrank == 0) {

int array[ARRAY\_SIZE];

printf("Enter %d elements of the array:\n", numprocs);

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

scanf("%d", &array[i]);

}

for (int dest = 1; dest < numprocs; ++dest) {

MPI\_Buffer\_attach(malloc(1000), 1000); // Attach buffer

MPI\_Bsend(&array[dest - 1], 1, MPI\_INT, dest, 0, MPI\_COMM\_WORLD);

printf("Root: Sent element %d to process %d\n", array[dest - 1], dest);

}

} else {

MPI\_Recv(&data, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

printf("Process %d: Received element %d\n", myrank, data);

if (myrank % 2 == 0) {

result = data \* data;

printf("Process %d: Square of %d is %d\n", myrank, data, result);

} else {

result = data \* data \* data;

printf("Process %d: Cube of %d is %d\n", myrank, data, result);

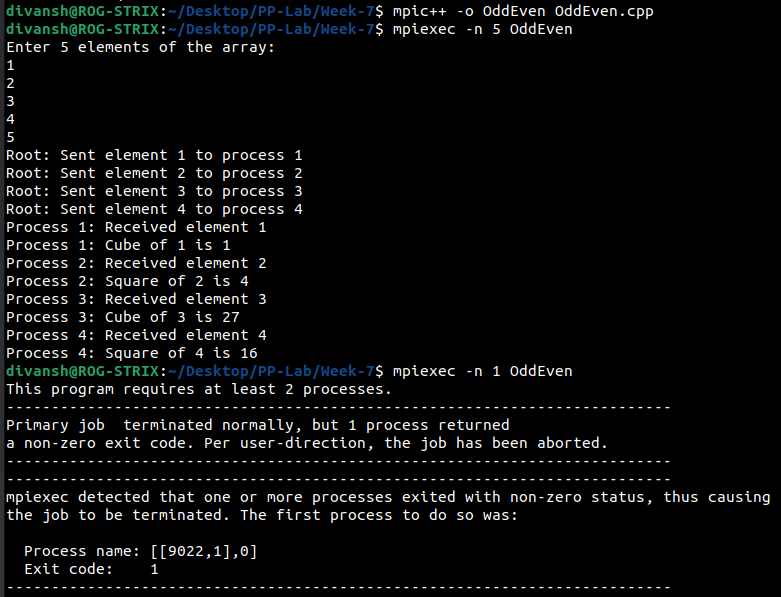
}

}

MPI\_Finalize();

return 0;

}



4) Write a MPI program to read an integer value in the root process. Root process sends this value to Process 1. Process 1 sends this value to Process 2 and so on. Last process sends the value back to the root process. When sending the value, each process will first increment the value by one. Write the program using point to point communication.

#include <stdio.h>

#include <mpi.h>

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

int myrank, numprocs;

int value;

MPI\_Init(&argc, &argv);

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

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

if (numprocs < 2) {

printf("This program requires at least 2 processes.\n");

MPI\_Finalize();

return 1;

}

if (myrank == 0) {

printf("Enter an integer value: \n");

scanf("%d", &value);

value++;

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

printf("\nRoot: Sent value %d to Process 1\n", value);

MPI\_Recv(&value, 1, MPI\_INT, numprocs - 1, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

printf("Root: Received value %d from Process %d\n", value, numprocs - 1);

} else if (myrank < numprocs - 1) {

MPI\_Recv(&value, 1, MPI\_INT, myrank - 1, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

printf("Process %d: Received value %d from Process %d\n", myrank, value, myrank - 1);

value++;

MPI\_Send(&value, 1, MPI\_INT, myrank + 1, 0, MPI\_COMM\_WORLD);

printf("Process %d: Sent value %d to Process %d\n", myrank, value, myrank + 1);

} else {

MPI\_Recv(&value, 1, MPI\_INT, myrank - 1, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

printf("Process %d: Received value %d from Process %d\n", myrank, value, myrank - 1);

value++;

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

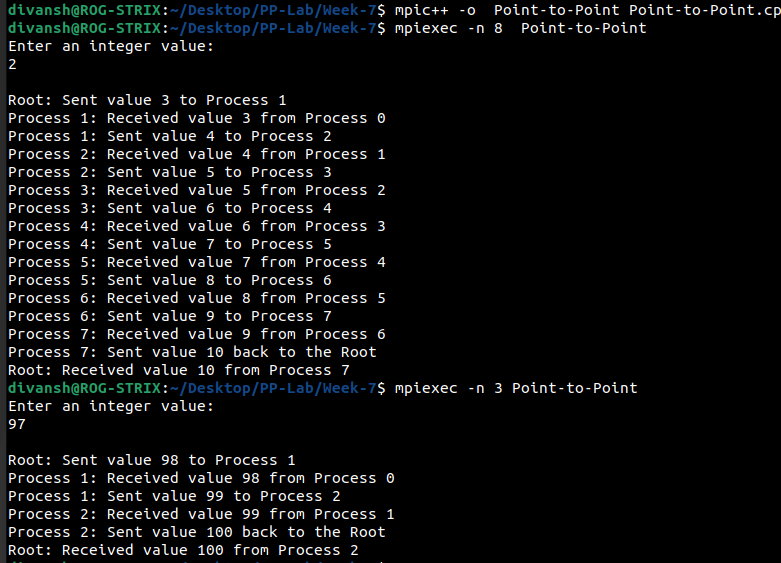
printf("Process %d: Sent value %d back to the Root\n", myrank, value);

}

MPI\_Finalize();

return 0;

}



5) Write a MPi program to read N elements of an array in the master process. Let N processes including master process check if the array values are prime or not.

#include <stdio.h>

#include <mpi.h>

// Function to check if a number is prime

int isPrime(int num) {

if (num <= 1) return 0;

for (int i = 2; i \* i <= num; ++i) {

if (num % i == 0) return 0;

return 1;

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

int myrank, numprocs;

int N = 8;

int array[N];

MPI\_Init(&argc, &argv);

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

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

if (numprocs != N) {

printf("This program requires %d processes.\n", N);

MPI\_Finalize();

return 1;

}

if (myrank == 0) {

printf("Enter %d elements of the array:\n", N);

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

scanf("%d", &array[i]);

}

}

// Broadcast the array to all processes

MPI\_Bcast(array, N, MPI\_INT, 0, MPI\_COMM\_WORLD);

// Each process checks if its element is prime

if (isPrime(array[myrank])) {

printf("Process %d: %d is prime.\n", myrank, array[myrank]);

} else {

printf("Process %d: %d is not prime.\n", myrank, array[myrank]);

}

MPI\_Finalize();

return 0;

}

