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;
}
```

```
divansh@ROG-STRIX:~/Desktop/PP-Lab/Week-7$ mpiexec -n 2 ./SyncSend
Sender: Sent word: hello
Sender: Received toggled word: HELLO
Receiver: Received word: hello
Receiver: Sent toggled word: HELLO
divansh@ROG-STRIX:~/Desktop/PP-Lab/Week-7$ mpic++ -o SyncSend SyncSend.cpp
divansh@ROG-STRIX:~/Desktop/PP-Lab/Week-7$ mpiexec -n 2 ./SyncSend
Sender: Sent word: hello
Sender: Received toggled word: HELLO
Receiver: Received word: hello
Receiver: Sent toggled word: HELLO
divansh@ROG-STRIX:~/Desktop/PP-Lab/Week-7$ mpiexec -n 4 ./SyncSend
This program requires exactly 2 processes.
--------------------------------------------------------------------------
Primary job  terminated normally, but 1 process returned
a non-zero exit code. Per user-direction, the job has been aborted.
--------------------------------------------------------------------------
--------------------------------------------------------------------------
mpiexec detected that one or more processes exited with non-zero status, thus causing
the job to be terminated. The first process to do so was:

  Process name: [[48900,1],2]
  Exit code:    1
--------------------------------------------------------------------------
```

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;}
```

```
divansh@ROG-STRIX:~/Desktop/PP-Lab/Week-7$ mpic++ -o MasterSlave MasterSlave.cpp
divansh@ROG-STRIX:~/Desktop/PP-Lab/Week-7$ mpiexec -n 6 MasterSlave
Master: Sent number 1 to process 1
Master: Sent number 2 to process 2
Master: Sent number 3 to process 3
Master: Sent number 4 to process 4
Master: Sent number 5 to process 5
Process 2: Received number 2
Process 3: Received number 3
Process 4: Received number 4
Process 5: Received number 5
Process 1: Received number 1
divansh@ROG-STRIX:~/Desktop/PP-Lab/Week-7$ mpiexec -n 1 MasterSlave
This program requires at least 2 processes.
--------------------------------------------------------------------------
Primary job  terminated normally, but 1 process returned
a non-zero exit code. Per user-direction, the job has been aborted.
--------------------------------------------------------------------------
--------------------------------------------------------------------------
mpiexec detected that one or more processes exited with non-zero status, thus causing
the job to be terminated. The first process to do so was:

  Process name: [[42079,1],0]
  Exit code:    1
--------------------------------------------------------------------------
```

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.

```c
#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;
}
```

```
divansh@ROG-STRIX:~/Desktop/PP-Lab/Week-7$ mpic++ -o OddEven OddEven.cpp
divansh@ROG-STRIX:~/Desktop/PP-Lab/Week-7$ mpiexec -n 5 OddEven
Enter 5 elements of the array:
1
2
3
4
5
Root: Sent element 1 to process 1
Root: Sent element 2 to process 2
Root: Sent element 3 to process 3
Root: Sent element 4 to process 4
Process 1: Received element 1
Process 1: Cube of 1 is 1
Process 2: Received element 2
Process 2: Square of 2 is 4
Process 3: Received element 3
Process 3: Cube of 3 is 27
Process 4: Received element 4
Process 4: Square of 4 is 16
divansh@ROG-STRIX:~/Desktop/PP-Lab/Week-7$ mpiexec -n 1 OddEven
This program requires at least 2 processes.
--------------------------------------------------------------------------
Primary job  terminated normally, but 1 process returned
a non-zero exit code. Per user-direction, the job has been aborted.
--------------------------------------------------------------------------
--------------------------------------------------------------------------
mpiexec detected that one or more processes exited with non-zero status, thus causing
the job to be terminated. The first process to do so was:

  Process name: [[9022,1],0]
  Exit code:    1
--------------------------------------------------------------------------
```

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.

```c
#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;
}
```

```
divansh@ROG-STRIX:~/Desktop/PP-Lab/Week-7$ mpic++ -o  Point-to-Point Point-to-Point.cp
divansh@ROG-STRIX:~/Desktop/PP-Lab/Week-7$ mpiexec -n 8  Point-to-Point
Enter an integer value:
2

Root: Sent value 3 to Process 1
Process 1: Received value 3 from Process 0
Process 1: Sent value 4 to Process 2
Process 2: Received value 4 from Process 1
Process 2: Sent value 5 to Process 3
Process 3: Received value 5 from Process 2
Process 3: Sent value 6 to Process 4
Process 4: Received value 6 from Process 3
Process 4: Sent value 7 to Process 5
Process 5: Received value 7 from Process 4
Process 5: Sent value 8 to Process 6
Process 6: Received value 8 from Process 5
Process 6: Sent value 9 to Process 7
Process 7: Received value 9 from Process 6
Process 7: Sent value 10 back to the Root
Root: Received value 10 from Process 7
divansh@ROG-STRIX:~/Desktop/PP-Lab/Week-7$ mpiexec -n 3 Point-to-Point
Enter an integer value:
97

Root: Sent value 98 to Process 1
Process 1: Received value 98 from Process 0
Process 1: Sent value 99 to Process 2
Process 2: Received value 99 from Process 1
Process 2: Sent value 100 back to the Root
Root: Received value 100 from Process 2
```

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>
```

```c
// 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;
}
```

```
divansh@ROG-STRIX:~/Desktop/PP-Lab/Week-7$ mpic++ -o ArrayPrime ArrayPrime.cpp
divansh@ROG-STRIX:~/Desktop/PP-Lab/Week-7$ mpiexec -n 8 ArrayPrime
Enter 8 elements of the array:
1
2
3
4
5
6
7
8
Process 0: 1 is not prime.
Process 1: 2 is prime.
Process 2: 3 is prime.
Process 4: 5 is prime.
Process 5: 6 is not prime.
Process 6: 7 is prime.
Process 3: 4 is not prime.
Process 7: 8 is not prime.
```