CSE 4001

PARALLEL AND DISTRIBUTED COMPUTING

Lab Assessment – 3

L27+L28 | PLBG04
Dr. Narayanan Prasanth

FALL SEMESTER 2021-22

by

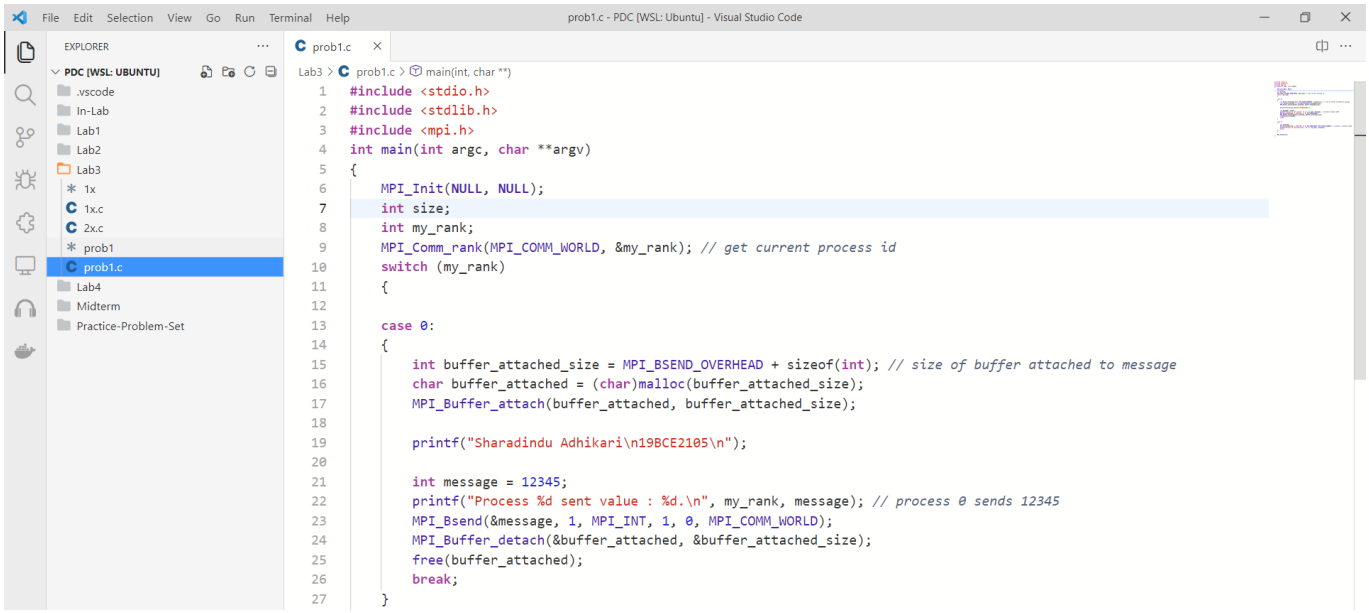
SHARADINDU ADHIKARI

19BCE2105

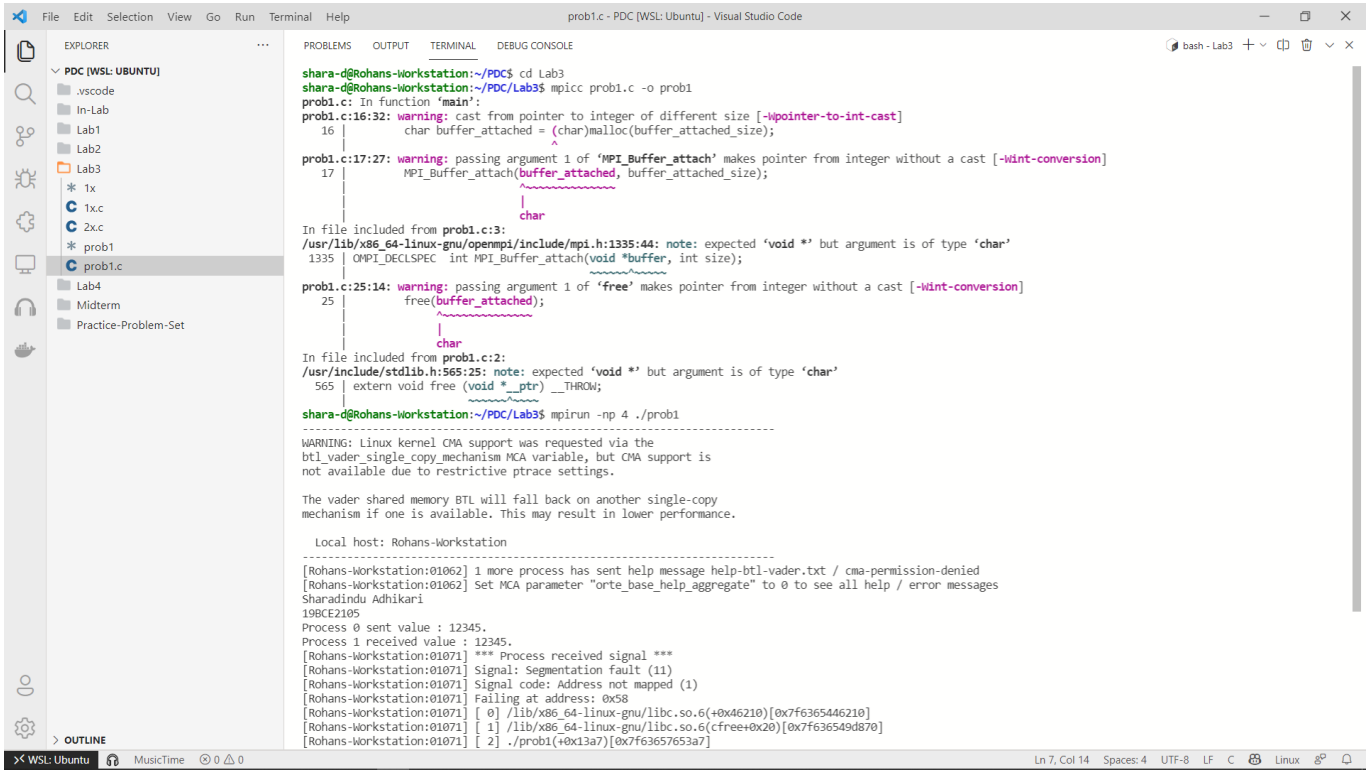
1. MPI_Bsend is the asynchronous blocking send (with user provided Buffering), it will block until a copy of the buffer is passed. Write a MPI program to establish a point to point communication between two process using the above function. (3)

```c
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
int main(int argc, char **argv)
{
    MPI_Init(NULL, NULL);
    int size;
    int my_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank); // get current process id
    switch (my_rank)
    {

    case 0:
    {
        int buffer_attached_size = MPI_BSEND_OVERHEAD + sizeof(int); //
size of buffer attached to message
        char buffer_attached = (char)malloc(buffer_attached_size);
        MPI_Buffer_attach(buffer_attached, buffer_attached_size);

        printf("Sharadindu Adhikari\n19BCE2105\n");

        int message = 12345;
        printf("Process %d sent value : %d.\n", my_rank, message); //
process 0 sends 12345
        MPI_Bsend(&message, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
        MPI_Buffer_detach(&buffer_attached, &buffer_attached_size);
        free(buffer_attached);
        break;
    }

    case 1:
    {
        int received;
        MPI_Recv(&received, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE); // process 1 receives 12345
```

```
            printf("Process %d received value : %d.\n", my_rank, received);
            break;
        }
        }

        MPI_Finalize();
    }
```



OUTPUT:

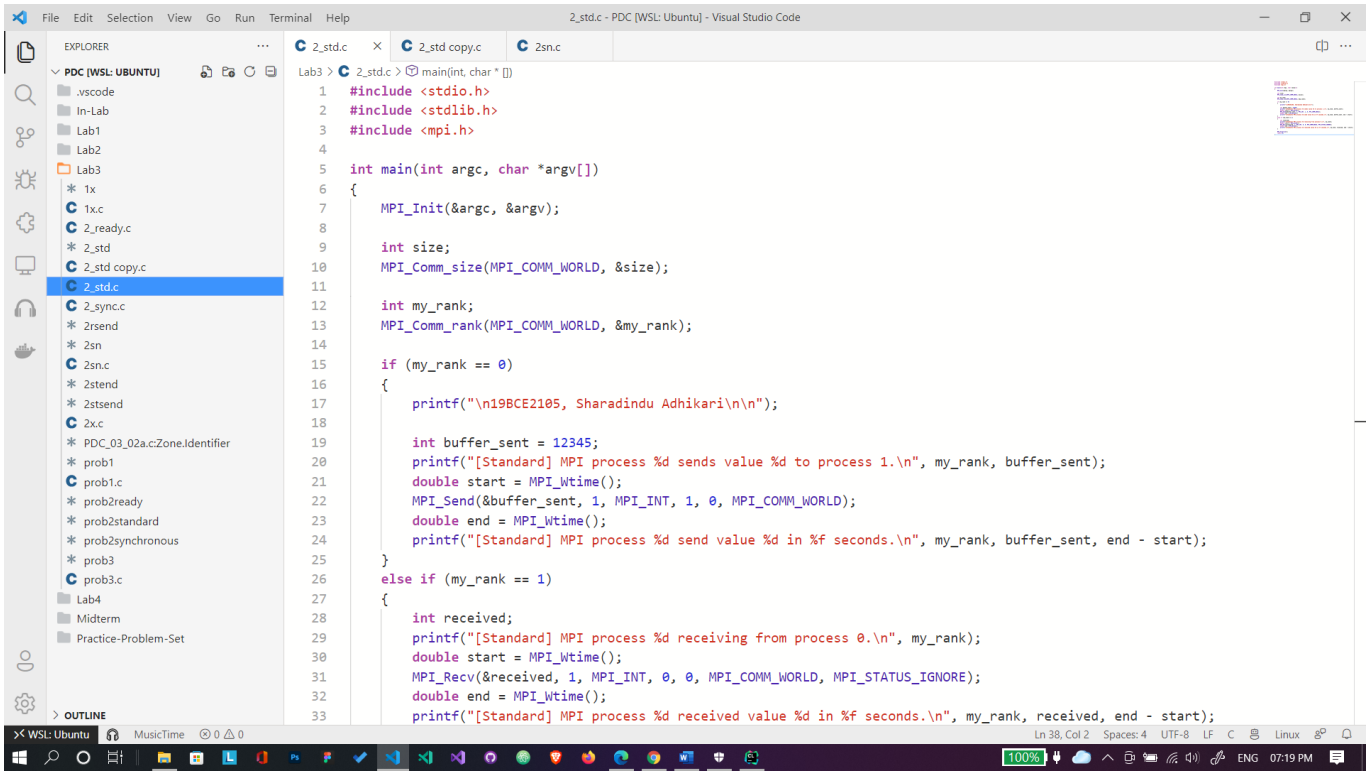


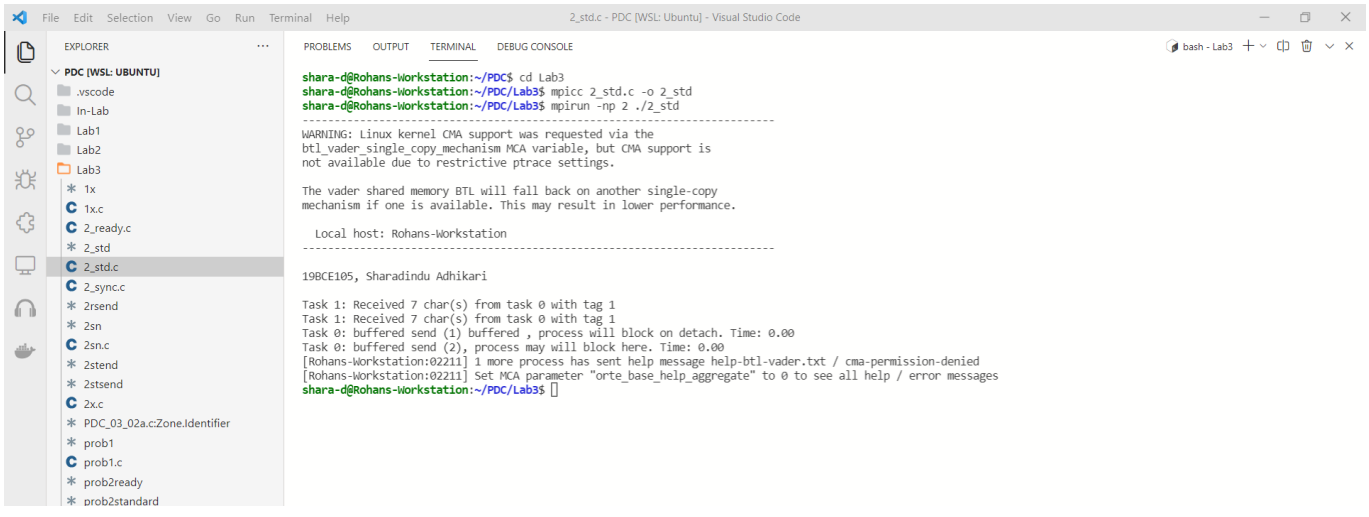
2. Develop a MPI program to perform point to point communication between two processes using the following send functions (Standard, Ready, Synchronous) in blocking mode. Compute the time taken by each send functions and find the optimal one. (3)

Standard send:

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

int main(int argc, char *argv[])
{
    MPI_Init(&argc, &argv);

    int size;
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    int my_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

    if (my_rank == 0)
    {
        printf("\n19BCE2105, Sharadindu Adhikari\n\n");

        int buffer_sent = 12345;
        printf("[Standard] MPI process %d sends value %d to process 1.\n",
my_rank, buffer_sent);
        double start = MPI_Wtime();
        MPI_Send(&buffer_sent, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
        double end = MPI_Wtime();
        printf("[Standard] MPI process %d send value %d in %f seconds.\n",
my_rank, buffer_sent, end - start);
    }
    else if (my_rank == 1)
    {
        int received;
        printf("[Standard] MPI process %d receiving from process 0.\n",
my_rank);
        double start = MPI_Wtime();
```

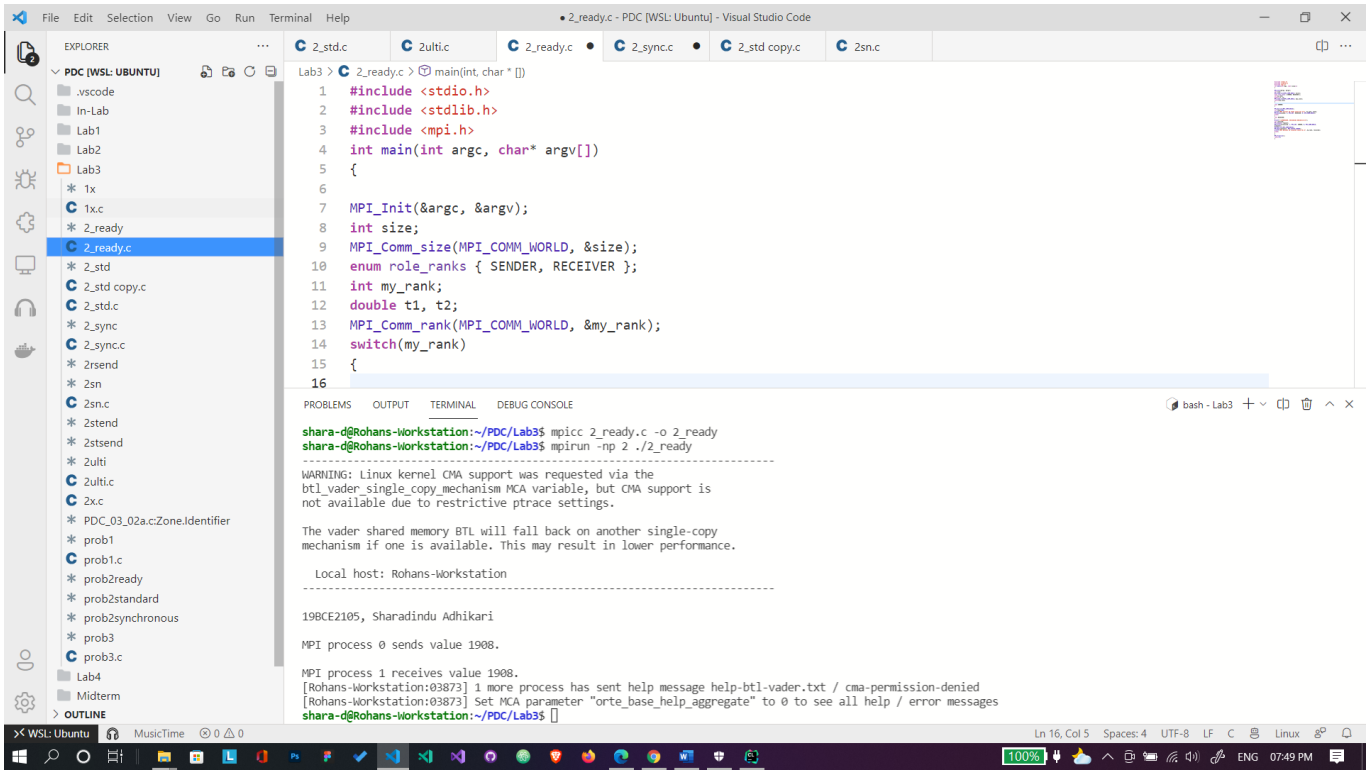
```
        MPI_Recv(&received, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
        double end = MPI_Wtime();
        printf("[Standard] MPI process %d received value %d in %f
seconds.\n", my_rank, received, end - start);
    }

    MPI_Finalize();
    return 0;
}
```





Ready send:

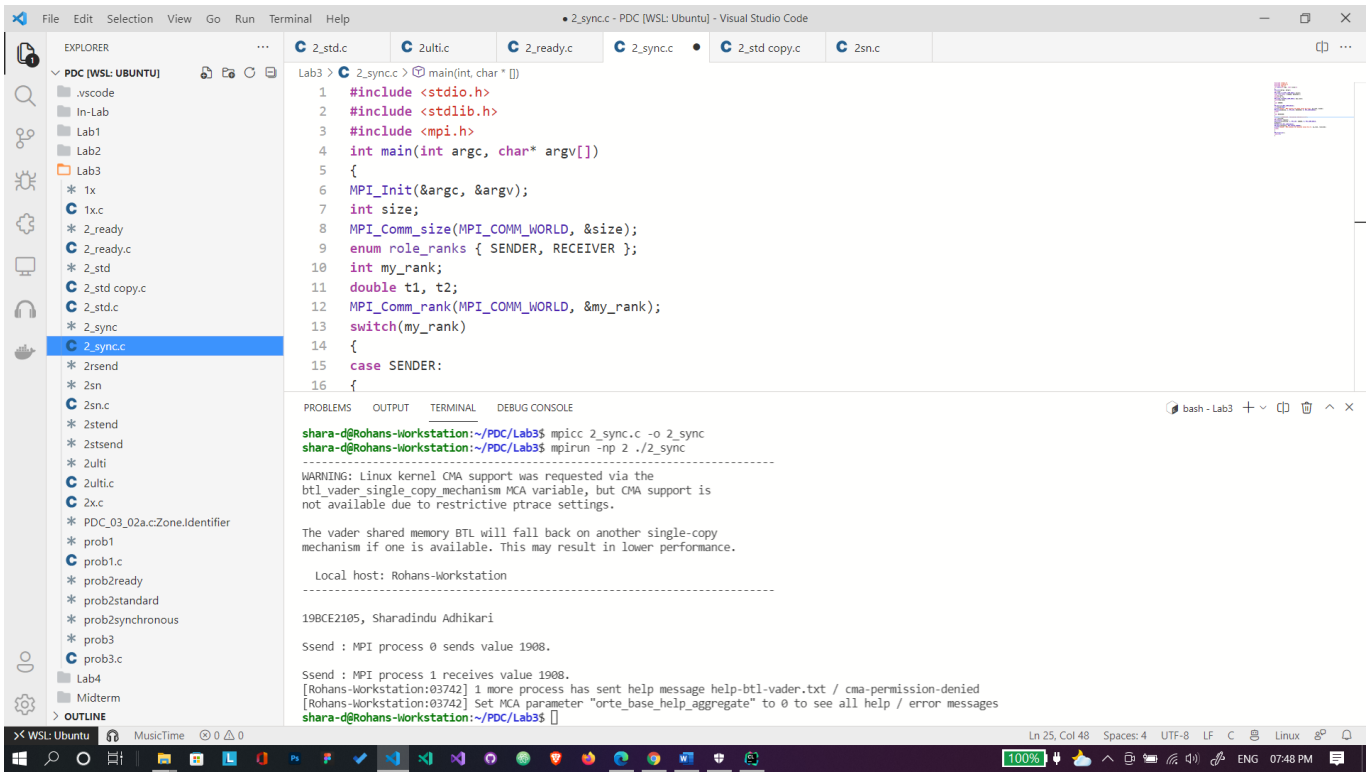
```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
int main(int argc, char* argv[])
{

MPI_Init(&argc, &argv);
int size;
MPI_Comm_size(MPI_COMM_WORLD, &size);
enum role_ranks { SENDER, RECEIVER };
int my_rank;
double t1, t2;
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
switch(my_rank)
{
case SENDER:
{

printf("\n19BCE2105, Sharadindu Adhikari\n\n");

MPI_Barrier(MPI_COMM_WORLD);
int send=1908;
printf("MPI process %d sends value %d.\n\n", my_rank, send);
MPI_Rsend(&send, 1, MPI_INT, RECEIVER, 0, MPI_COMM_WORLD);
break;
}
case RECEIVER:
{
int received;
MPI_Request request;
MPI_Irecv(&received, 1, MPI_INT, SENDER, 0, MPI_COMM_WORLD,
&request);
MPI_Barrier(MPI_COMM_WORLD);
MPI_Wait(&request, MPI_STATUS_IGNORE);
printf("MPI process %d receives value %d.\n", my_rank, received);
break;
}
}
MPI_Finalize();
return 0;
}
```



Synchronous send:

```c
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
int main(int argc, char* argv[])
{
MPI_Init(&argc, &argv);
int size;
MPI_Comm_size(MPI_COMM_WORLD, &size);
enum role_ranks { SENDER, RECEIVER };
int my_rank;
double t1, t2;
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
switch(my_rank)
{
case SENDER:
{
MPI_Barrier(MPI_COMM_WORLD);
int Ssend=1908;
printf("Ssend : MPI process %d sends value %d.\n\n", my_rank, Ssend);
MPI_Ssend(&Ssend, 1, MPI_INT, RECEIVER, 0, MPI_COMM_WORLD);
```

```
break;
}
case RECEIVER:
{
printf("\n19BCE2105, Sharadindu Adhikari\n\n");
int received;
MPI_Request request;
MPI_Irecv(&received, 1, MPI_INT, SENDER, 0, MPI_COMM_WORLD,
&request);

MPI_Barrier(MPI_COMM_WORLD);
MPI_Wait(&request, MPI_STATUS_IGNORE);
printf("Ssend : MPI process %d receives value %d.\n", my_rank, received);
break;
}
}
MPI_Finalize();
return 0;
}
```



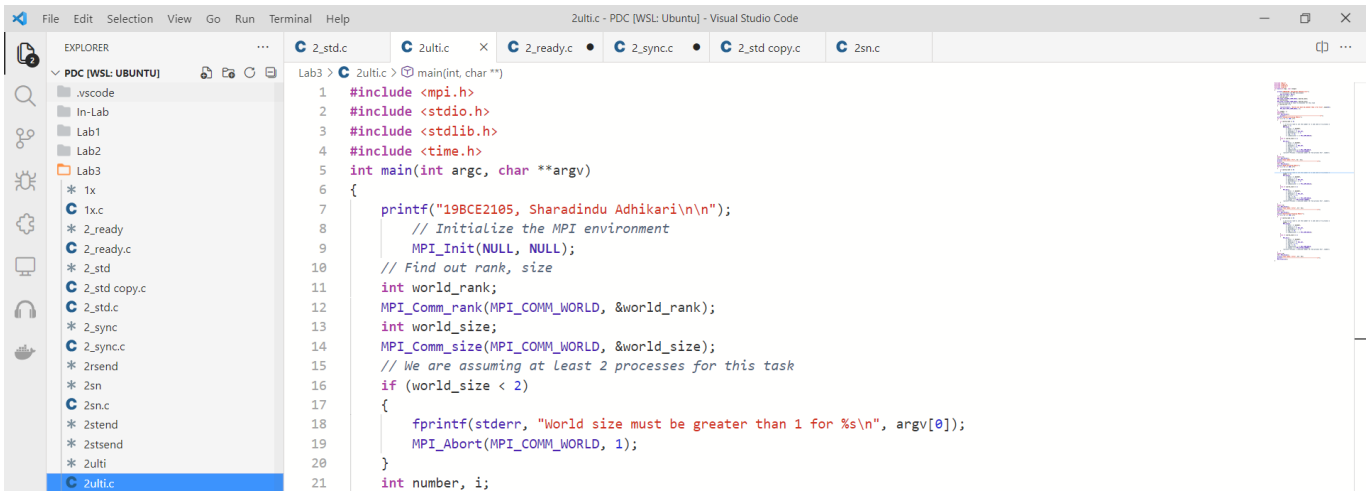
ALTERNATIVE SOLUTION TO PROBLEM 2:

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main(int argc, char **argv)
{
    printf("19BCE2105, Sharadindu Adhikari\n\n");
        // Initialize the MPI environment
        MPI_Init(NULL, NULL);
    // Find out rank, size
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);
    // We are assuming at least 2 processes for this task
    if (world_size < 2)
    {
        fprintf(stderr, "World size must be greater than 1 for %s\n",
argv[0]);
        MPI_Abort(MPI_COMM_WORLD, 1);
    }
    int number, i;
    double t1;
    t1 = MPI_Wtime();
        printf("-------------------------------------------------\n");
    printf("Standard Blocking Mode\n");
    for (i = 0; i < 100; i++)
    {
        if (world_rank == 0)
        {
            // If we are rank 0, set the number to -1 and send it to process
1
            number = -1;
            MPI_Send(
                /* data = */ &number,
                /* count = */ 1,
                /* datatype = */ MPI_INT,
                /* destination = */ 1,
                /* tag = */ 0,
                /* communicator = */ MPI_COMM_WORLD);
        }
        else if (world_rank == 1)
        {
```

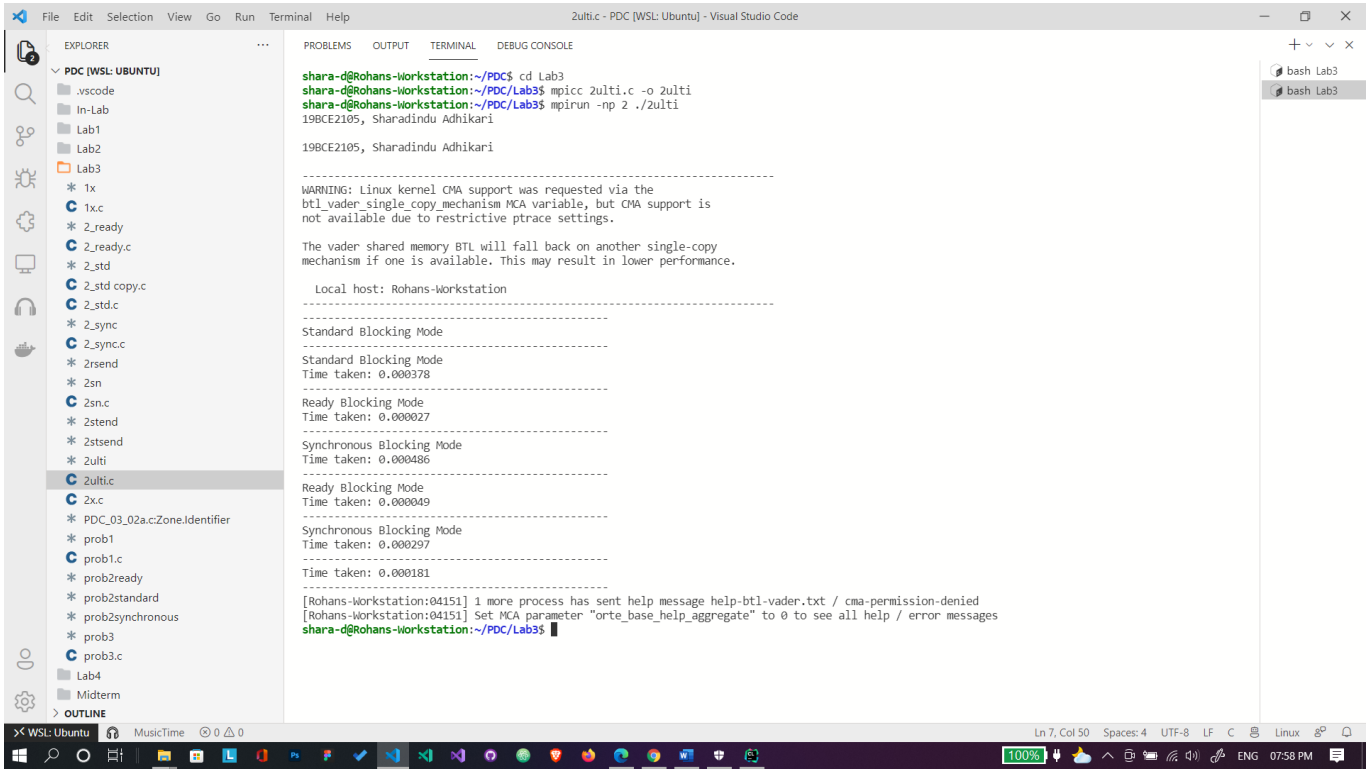
```
            MPI_Recv(
                /* data = */ &number,
                /* count = */ 1,
                /* datatype = */ MPI_INT,
                /* source = */ 0,
                /* tag = */ 0,
                /* communicator = */ MPI_COMM_WORLD,
                /* status = */ MPI_STATUS_IGNORE);
            //printf("Process 1 received number %d from process 0\n",
number);
        }
    }
    double t2;
    t2 = MPI_Wtime();
    printf("Time taken: %f\n", t2 - t1);
    printf("------------------------------------------------\n");
    double t3;
    t3 = MPI_Wtime();
    printf("Ready Blocking Mode\n");
    for (i = 0; i < 100; i++)
    {
        if (world_rank == 0)
        {
            // If we are rank 0, set the number to -1 and send it to process
1
            number = -1;
            MPI_Rsend(
                /* data = */ &number,
                /* count = */ 1,
                /* datatype = */ MPI_INT,
                /* destination = */ 1,
                /* tag = */ 0,
                /* communicator = */ MPI_COMM_WORLD);
        }
        else if (world_rank == 1)
        {
            MPI_Recv(
                /* data = */ &number,
                /* count = */ 1,
                /* datatype = */ MPI_INT,
                /* source = */ 0,
                /* tag = */ 0,
                /* communicator = */ MPI_COMM_WORLD,
                /* status = */ MPI_STATUS_IGNORE);
```

```
            //printf("Process 1 received number %d from process 0\n",
number);
        }
    }
    double t4;
    t4 = MPI_Wtime();
    printf("Time taken: %lf\n", t4 - t3);
    printf("------------------------------------------------\n");
    double t5;
    t5 = MPI_Wtime();
    printf("Synchronous Blocking Mode\n");
    for (i = 0; i < 100; i++)
    {
        if (world_rank == 0)
        {
            // If we are rank 0, set the number to -1 and send it to process
1
            number = -1;
            MPI_Ssend(
                /* data = */ &number,
                /* count = */ 1,
                /* datatype = */ MPI_INT,
                /* destination = */ 1,
                /* tag = */ 0,
                /* communicator = */ MPI_COMM_WORLD);
        }
        else if (world_rank == 1)
        {
            MPI_Recv(
                /* data = */ &number,
                /* count = */ 1,
                /* datatype = */ MPI_INT,
                /* source = */ 0,
                /* tag = */ 0,
                /* communicator = */ MPI_COMM_WORLD,
                /* status = */ MPI_STATUS_IGNORE);
            //printf("Process 1 received number %d from process 0\n",
number);
        }
    }
    double t6;
    t6 = MPI_Wtime();
    printf("Time taken: %lf\n", t6 - t5);
    printf("------------------------------------------------\n");
```

```
    MPI_Finalize();
}
```



Output:



Inference: From the output, we can definitely conclude that Ready send function is the most optimal one!

Q3:

2. A communicator has 3 process (ID: 0, 1 and 2) such that process 0 sends a binary value 001 to process 1, in-turn process 1 increment the value by 1 (i.e.010) and send the same to process 2, in-turn process 2 increment the value by 1 (i.e.011) and send the same to process 0. Continue the process until the binary value becomes 111. Develop a MPI program to implement the above scenario. (4)

```
#include <mpi.h>
#include <stdio.h>
#include <string.h>
int main(int argc, char **argv)

{
    MPI_Init(NULL, NULL);
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
    char binaryno[4];
    if (world_rank == 0)

    {
        printf("Sharadindu Adhikari\n19BCE2105\n");

        strcpy(binaryno, "001");
        MPI_Send(&binaryno, 4, MPI_CHAR, 1, 0, MPI_COMM_WORLD);
        printf("MessageSentfromProcess%d:%s\n", world_rank, binaryno);
        MPI_Recv(&binaryno, 4, MPI_CHAR, 2, 2, MPI_COMM_WORLD,
                 MPI_STATUS_IGNORE);
        printf("FinalMessageReceivedatProcess%d:%s\n", world_rank,
binaryno);
    }

    else if (world_rank == 1)
    {
        MPI_Recv(&binaryno, 4, MPI_CHAR, 0, 0, MPI_COMM_WORLD,
                 MPI_STATUS_IGNORE);
        printf("MessageReceivedatProcess%d:%s\n", world_rank, binaryno);
        strcpy(binaryno, "010");
        MPI_Send(&binaryno, 4, MPI_CHAR, 2, 1, MPI_COMM_WORLD);
        printf("MessageSentfromProcess%d:%s\n", world_rank, binaryno);
    }
```

```
    else if (world_rank == 2)
    {
        MPI_Recv(&binaryno, 4, MPI_CHAR, 1, 1, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
        printf("MessageReceivedatProcess%d:%s\n", world_rank, binaryno);
        strcpy(binaryno, "011");
        MPI_Send(&binaryno, 4, MPI_CHAR, 0, 2, MPI_COMM_WORLD);
        printf("MessageSentfromProcess%d:%s\n", world_rank, binaryno);
    }

    MPI_Finalize();
}
```

OUTPUT: