E2

Program to send data and receive date to/from processors using MPI.

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

int main(int argc, char **argv)
{
    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);
    int number;

    if (world_rank == 0)
    {
        number = -1;
        MPI_Send(&number, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
        MPI_Send(&number, 1, MPI_INT, 2, 0, MPI_COMM_WORLD);
    }
    else if (world_rank == 1)
    {
        MPI_Recv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
        printf("Process 1 received number %d from process 0\n",
number);
    }

    if (world_rank == 2)
    {
```

```
        MPI_Recv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
        printf("Process 2 received number %d from process 0\n",
number);
    }

    MPI_Finalize();
    return 0;
}
```

E3

Program to illustrate broadcasting of data using MPI.

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

void my_bcast(void *data, int count, MPI_Datatype datatype, int
root, MPI_Comm communicator)
{
    int world_rank;
    MPI_Comm_rank(communicator, &world_rank);
    int world_size;
    MPI_Comm_size(communicator, &world_size);

    if (world_rank == root)
    {
        // If we are the root process, send our data to everyone
        int i;
        for (i = 0; i < world_size; i++)
        {
            if (i != world_rank)
            {
                MPI_Send(data, count, datatype, i, 0, communicator);
            }
        }
    }
    else
    {
        // If we are a receiver process, receive the data from the
root
        MPI_Recv(data, count, datatype, root, 0, communicator,
MPI_STATUS_IGNORE);
```

```
    }
}

int main(int argc, char **argv)
{
    MPI_Init(NULL, NULL);

    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    int data;
    if (world_rank == 0)
    {
        data = 100;
        printf("Process 0 broadcasting data %d\n", data);
        my_bcast(&data, 1, MPI_INT, 0, MPI_COMM_WORLD);
    }
    else
    {
        my_bcast(&data, 1, MPI_INT, 0, MPI_COMM_WORLD);
        printf("Process %d received data %d from root process\n",
world_rank, data);
    }

    MPI_Finalize();
    return 0;
}
```

E4

To write a parallel program for sum of n natural number.

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

int main(int argc, char *argv[])
{
      int myRank;
      int size;
      int num;
      int lower, upper;
      int i;
      double local_result = 0.0;
      double total;

      MPI_Init(&argc, &argv);
      MPI_Comm_rank(MPI_COMM_WORLD, &myRank);
      MPI_Comm_size(MPI_COMM_WORLD, &size);

      if (myRank == 0)
      {
            printf("Enter a number: ");
            scanf("%d", &num);
      }

      MPI_Bcast(&num, 1, MPI_INT, 0, MPI_COMM_WORLD);

      if (myRank == 0)
      {
            lower = 1;
      }
      else
```

```
	{
		lower = myRank * (num / size) + 1;
	}

	if (myRank == (size - 1))
	{
		upper = num;
	}
	else
	{
		upper = (myRank + 1) * (num / size);
	}

	for (i = lower; i <= upper; i++)
	{
		local_result = local_result + (double)i;
		printf("\nMy upper=%d lower=%d rank=%d val=%lf", upper,
lower, myRank, local_result);
	}

	MPI_Reduce(&local_result, &total, 1, MPI_DOUBLE, MPI_SUM, 0,
MPI_COMM_WORLD);

	if (myRank == 0)
	{
		printf("\nThe sum of %d is %lf, and was calculated using
%d processes\n", num, total, size);
	}

	MPI_Finalize();
	return 0;
}
```

E5

To write a parallel program for factorial of a number.

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

int main(int argc, char *argv[])
{
    int myRank;
    int size;
    int fact;
    int lower, upper;
    int i;
    double local_result = 1.0;
    double total;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &myRank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    if (myRank == 0)
    {
        printf("Enter a number: ");
        scanf("%d", &fact);
    }

    MPI_Bcast(&fact, 1, MPI_INT, 0, MPI_COMM_WORLD);

    if (myRank == 0)
    {
        lower = 1;
    }
    else
    {
```

```
        lower = myRank * (fact / size) + 1;
    }

    if (myRank == (size - 1))
    {
        upper = fact;
    }
    else
    {
        upper = (myRank + 1) * (fact / size);
    }

    for (i = lower; i <= upper; i++)
    {
        local_result = local_result * (double)i;
        printf("\nMy upper=%d lower=%d rank=%d val=%lf", upper,
lower, myRank, local_result);
    }

    MPI_Reduce(&local_result, &total, 1, MPI_DOUBLE, MPI_PROD, 0,
MPI_COMM_WORLD);

    if (myRank == 0)
    {
        printf("The factorial of %d is %lf, and was calculated using
%d processes\n", fact, total, size);
    }

    MPI_Finalize();

    return 0;
}
```