Lab 9- CSE 4001

Name - Ayush Sharma
Reg no - 15BCE1335
Faculty: Prof Gayatri R.

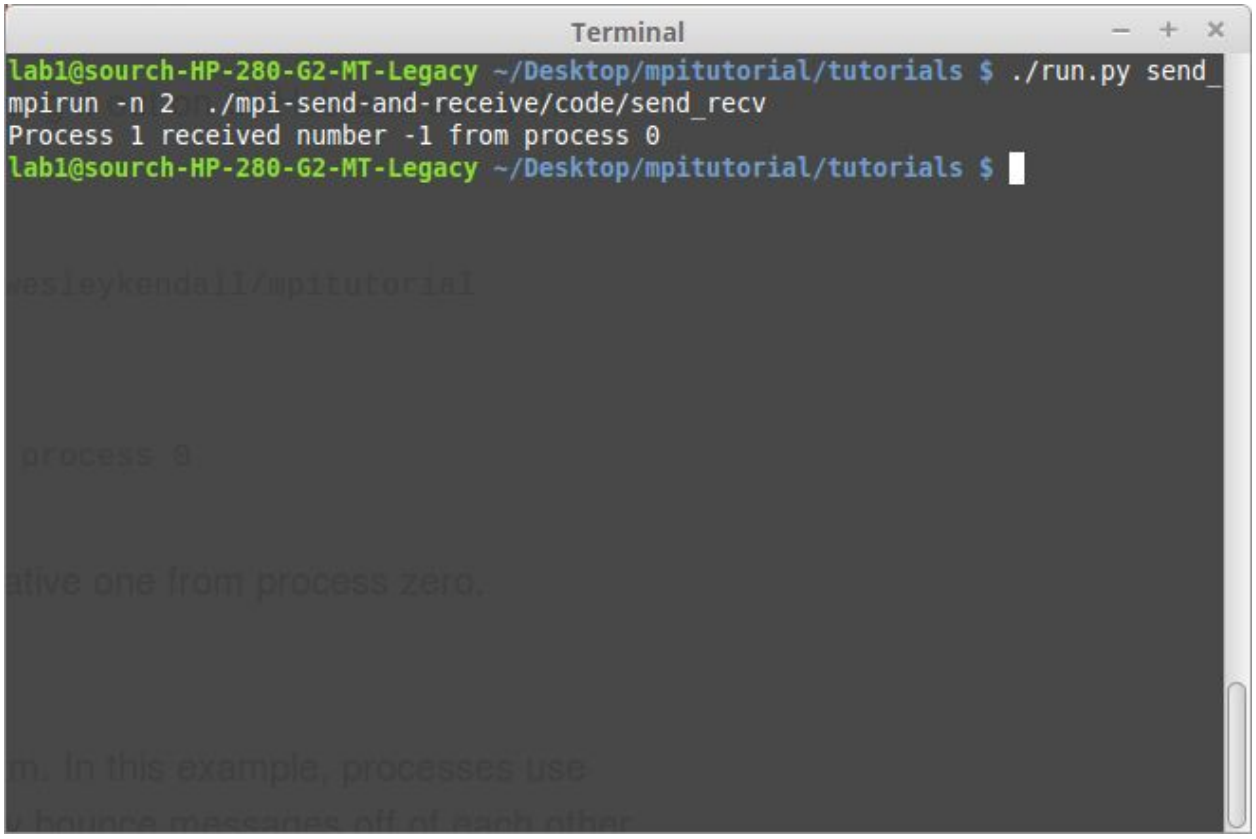
1) Send and Receive program of MPI

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

int main(int argc, char** argv) {
  // 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;
  if (world_rank == 0) {
        // If we are rank 0, set the number to -1 and send it to process 1
        number = -1;
        MPI_Send(&number, 1, MPI_INT, 1, 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);
  }
  MPI_Finalize();
}
```



2) Ping Pong Problem

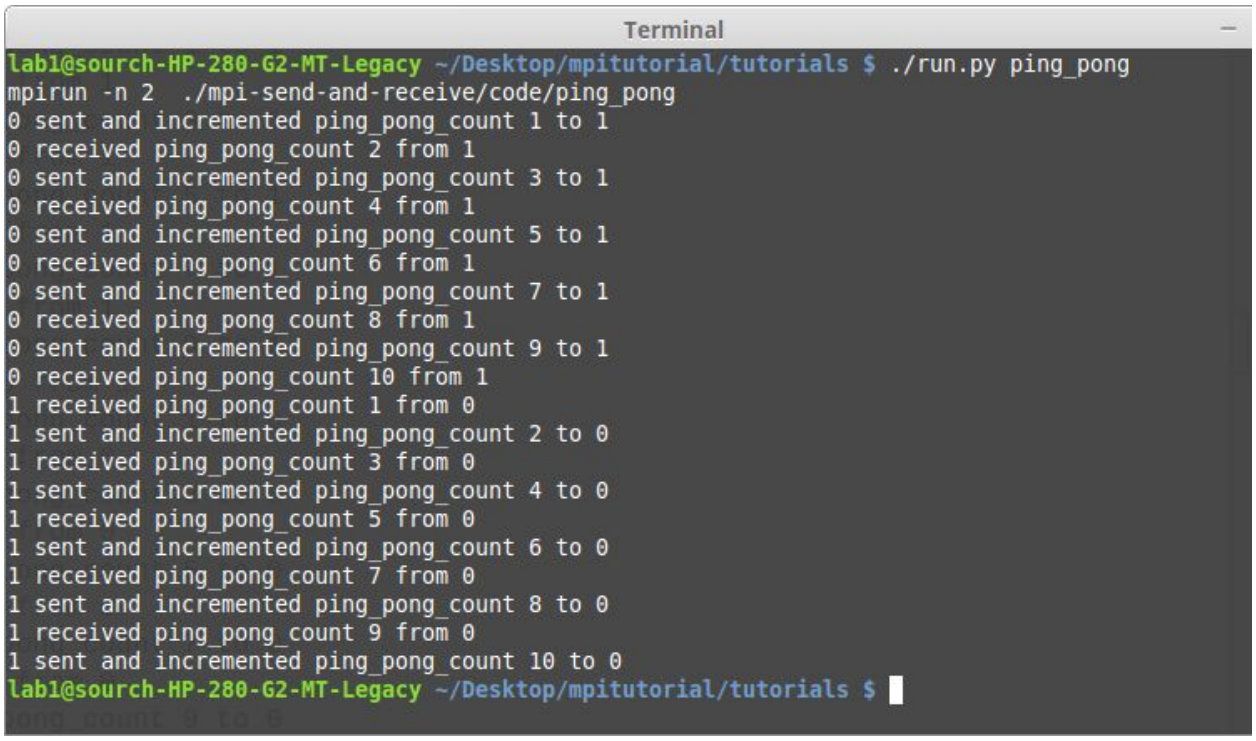
```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char** argv) {
  const int PING_PONG_LIMIT = 10;

  // 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 two for %s\n", argv[0]);
    MPI_Abort(MPI_COMM_WORLD, 1);
  }

  int ping_pong_count = 0;
  int partner_rank = (world_rank + 1) % 2;
  while (ping_pong_count < PING_PONG_LIMIT) {
    if (world_rank == ping_pong_count % 2) {
      // Increment the ping pong count before you send it
      ping_pong_count++;
      MPI_Send(&ping_pong_count, 1, MPI_INT, partner_rank, 0, MPI_COMM_WORLD);
      printf("%d sent and incremented ping_pong_count %d to %d\n",
             world_rank, ping_pong_count, partner_rank);
    } else {
      MPI_Recv(&ping_pong_count, 1, MPI_INT, partner_rank, 0, MPI_COMM_WORLD,
               MPI_STATUS_IGNORE);
      printf("%d received ping_pong_count %d from %d\n",
             world_rank, ping_pong_count, partner_rank);
    }
  }
  MPI_Finalize();
}
```



3) Ring

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

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

  int token;
  // Receive from the lower process and send to the higher process. Take care
  // of the special case when you are the first process to prevent deadlock.
  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 {
    // Set the token's value if you are process 0
    token = -1;
  }
  MPI_Send(&token, 1, MPI_INT, (world_rank + 1) % world_size, 0,
           MPI_COMM_WORLD);
  // Now process 0 can receive from the last process. This makes sure that at
  // least one MPI_Send is initialized before all MPI_Recvs (again, to prevent
  // deadlock)
  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();
}
```