CSE4001 – LAB 10
SYNCHRONIZATION USING MPI PRIMITIVES

Name: Ayush Sharma
Reg. No: 15BCE1335

QUESTION - USE MPI_BARRIER PRIMITIVE IN ORDER TO ACHIEVE SYNCHRONIZATION IN RING PROGRAM

ANSWER - CODE

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

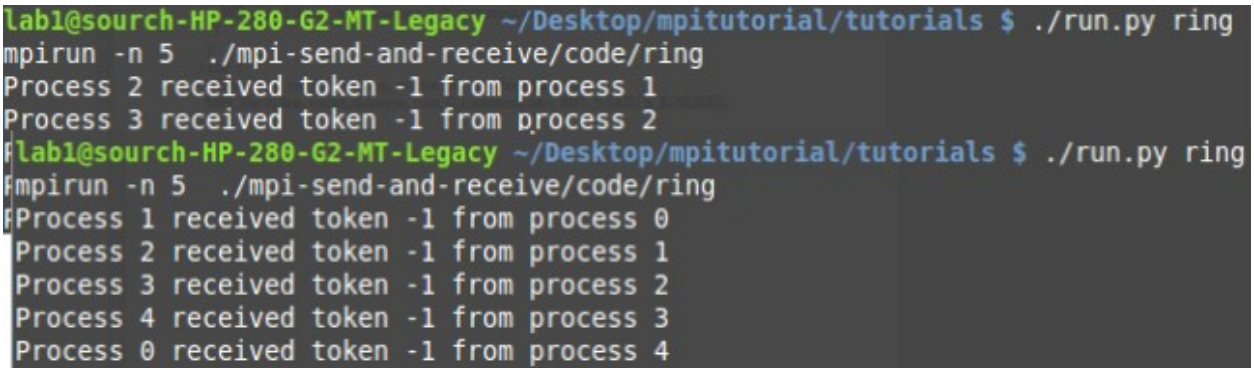
```
        MPI_Recv(data, count, datatype, root, 0, communicator,
MPI_STATUS_IGNORE);
  }
}

int main(int argc, char** argv) {
  // 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);
  MPI_Barrier(MPI_COMM_WORLD);
  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);
    my_bcast(&data, 1, MPI_INT, 0, MPI_COMM_WORLD);
    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;
  }
    my_bcast(&data, 1, MPI_INT, 0, MPI_COMM_WORLD);
  //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
```

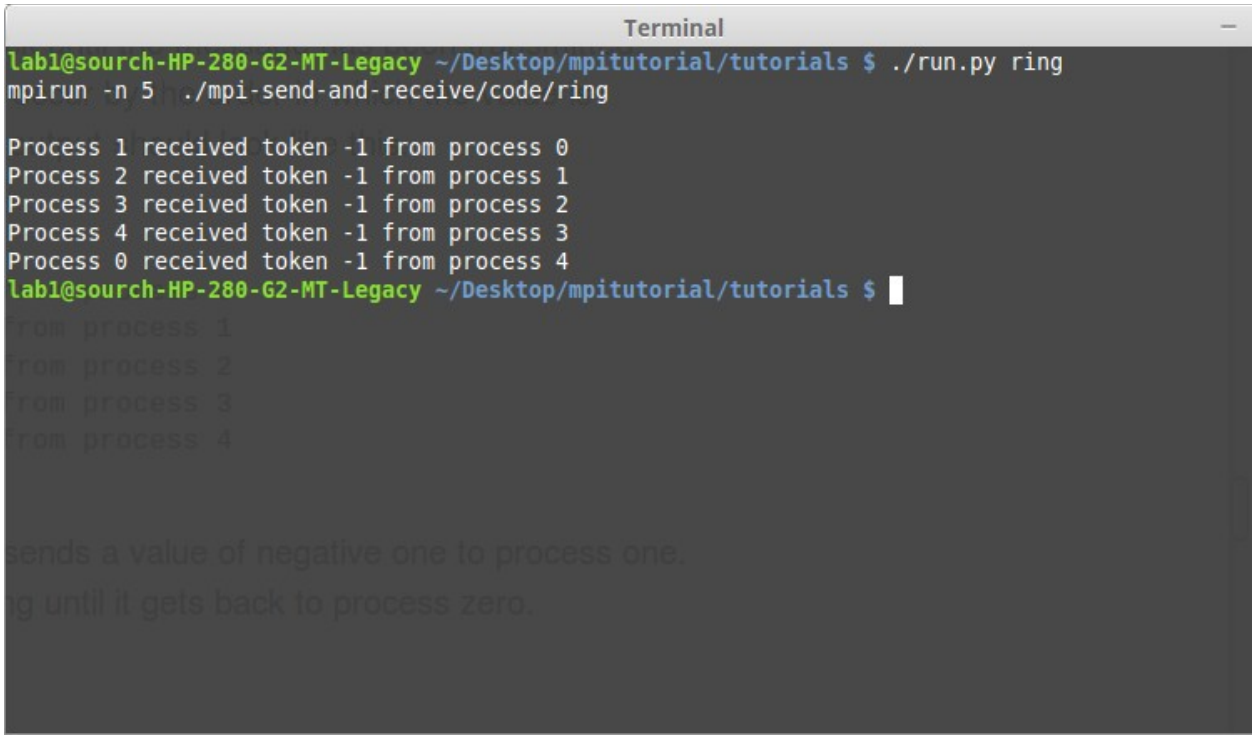
```
  // least one MPI_Send is initialized before all MPI_Recvs (again, to
prevent
  // deadlock)
  if (world_rank == 0) {
        //MPI_Recv(&token, 1, MPI_INT, world_size - 1, 0,
MPI_COMM_WORLD,
        //       MPI_STATUS_IGNORE);
    my_bcast(&data, 1, MPI_INT, 0, MPI_COMM_WORLD);
    printf("Process %d received token %d from process %d\n", world_rank,
token,
        world_size - 1);
  }
  MPI_Finalize();
}
```

RESULT

BEFORE USING MPI_BARRIER



AFTER USING MPI_BARRIER


Terminal
lab1@sourch-HP-280-G2-MT-Legacy ~/Desktop/mpitutorial/tutorials $ ./run.py ring
mpirun -n 5 ./mpi-send-and-receive/code/ring
Process 1 received token -1 from process 0
Process 2 received token -1 from process 1
Process 3 received token -1 from process 2
Process 4 received token -1 from process 3
Process 0 received token -1 from process 4
lab1@sourch-HP-280-G2-MT-Legacy ~/Desktop/mpitutorial/tutorials $