6. Write a MPI program to demonstration of deadlock using point to point communication and avoidance of deadlock by altering the call sequence

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

int main(int argc, char** argv) {
    MPI_Init(&argc, &argv);                 // Initialize MPI environment

    int rank, size;
    int message = 100;                      // single integer used for sending/receiving
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);   // get this process' rank (ID)
    MPI_Comm_size(MPI_COMM_WORLD, &size);   // get total number of processes

    // Ensure only 2 processes are used
    if (size != 2) {
        if (rank == 0)
            printf("Please run this program with 2 processes.\n");
        MPI_Finalize();                     // finalize MPI and exit
        return 0;
    }

    // Default mode = "deadlock"
    char mode[10] = "deadlock";

    // Optional: check for argument to choose mode (argv[1] = "deadlock" or "safe")
    if (argc > 1) {
        /* strncpy used to copy the argument safely into the fixed-size buffer.
           We ensure the last byte is '\0' to avoid overflow issues. */
        strncpy(mode, argv[1], sizeof(mode));
        mode[sizeof(mode) - 1] = '\0';     // force null-termination
    }

    if (strcmp(mode, "deadlock") == 0) {
        /* DEADLOCK mode: both processes do a blocking send first, then a blocking recv.
           If both MPI_Send calls block waiting for a matching receive, both processes
           will be stuck (deadlock). */
        if (rank == 0) {
            MPI_Send(&message, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
            MPI_Recv(&message, 1, MPI_INT, 1, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
            printf("Process 0 completed communication in deadlock mode.\n");
        } else if (rank == 1) {
            MPI_Send(&message, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
            MPI_Recv(&message, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
            printf("Process 1 completed communication in deadlock mode.\n");
        }

    } else if (strcmp(mode, "safe") == 0) {
        /* SAFE mode: rank 0 does Send then Recv; rank 1 does Recv then Send.
           This ordering guarantees at least one side is ready to receive the other's message,
           so blocking sends can complete and no deadlock occurs. */
        if (rank == 0) {
            MPI_Send(&message, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
            MPI_Recv(&message, 1, MPI_INT, 1, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
            printf("Process 0: Sent and received safely.\n");
        } else if (rank == 1) {
            MPI_Recv(&message, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
            MPI_Send(&message, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
            printf("Process 1: Received and sent safely.\n");
        }

    } else {
        if (rank == 0)
            printf("Unknown mode: %s. Use 'deadlock' or 'safe'.\n", mode);
    }

    MPI_Finalize();                          // Clean up MPI
    return 0;
}
```

```
mpicc deadlock_vs_safe.c -o deadlock_vs_safe
```

1.
   ```
   mpirun -np 1 ./deadlock_vs_safe
   Please run this program with 2 processes.
   ```
2.
   ```
   mpirun -np 2 ./deadlock_vs_safe  deadlock
   Process 0 completed communication in deadlock mode.
   Process 1 completed communication in deadlock mode.
   ```
3.
   ```
   mpirun -np 2 ./deadlock_vs_safe  safe
   Process 0: Sent and received safely.
   Process 1: Received and sent safely.
   ```