1 MPI Hands-On; Sending and Receiving Messages III

1. **Synchronous sending**. MPI example code6.c using synchronous send. Modify the code and try to see what may cause to deadlock.

```c
/* A simple MPI example program using synchronous send                */

/* The program consists of one sender process and one receiver        */
/* The sender process sends a message containing its identifier       */
/* to the receiver. This receives the message and sends it back       */
/* Both processes use synchronous send operations (MPI_Ssend)         */

/* Compile the program with 'mpicc -o code6 code6.c'                  */
/* Run the program with 'mpirun -np 2 code6                           */

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

int main(int argc, char* argv[]) {
  int x, y, np, me;
  int tag = 42;
  MPI_Status  status;

  MPI_Init(&argc, &argv);                 /* Initialize MPI */
  MPI_Comm_size(MPI_COMM_WORLD, &np);     /* Get number of processes */
  MPI_Comm_rank(MPI_COMM_WORLD, &me);     /* Get own identifier */

  x = me;
  if (me == 0) {     /* Process 0 does this */
    printf("Sending to process 1\n");
    MPI_Ssend(&x, 1, MPI_INT, 1, tag, MPI_COMM_WORLD);  /* Synchronous
     send */
    printf("Receiving from process 1\n");
    MPI_Recv (&y, 1, MPI_INT, 1, tag, MPI_COMM_WORLD, &status);
    printf("Process %d received a message containing value %d\n", me,
    y);
  }
  else
  {          /* Process 1 does this */
    /* Since we use synchronous send, we have to do the receive-
    operation */
    /* first, otherwise we will get a deadlock */
    MPI_Recv (&y, 1, MPI_INT, 0, tag, MPI_COMM_WORLD, &status);
    MPI_Ssend (&x, 1, MPI_INT, 0, tag, MPI_COMM_WORLD);  /*
    Synchronous send */
  }
  MPI_Finalize();
}
```

2. **Buffered sending**. MPI example code7.c using buffered send to pass a message between two processes.

```c
/* A simple MPI example program using buffered send                   */
/* The program does exactly the same as code6.c                       */

/* The program consists of one sender process and one receiver        */
/* The sender process sends a message containing its identifier       */
```

```c
/* to the receiver. This receives the message and sends it back     */
/* Both processes use buffered send operations (MPI_Bsend)          */

/* Compile the program with 'mpicc -o code7 code7.c'                */
/* Run the program with 'mpirun -np 2 code7                         */

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

#define BUFFSIZE 100     /* Size of the message buffer */

int main(int argc, char* argv[]) {
  int x, y, np, me;
  int buff[BUFFSIZE];    /* Buffer to be used in the communication */
  int size = BUFFSIZE;
  int tag = 42;
  MPI_Status  status;

  MPI_Init(&argc, &argv);               /* Initialize MPI */
  MPI_Comm_size(MPI_COMM_WORLD, &np);   /* Get number of processes */
  MPI_Comm_rank(MPI_COMM_WORLD, &me);   /* Get own identifier */

  MPI_Buffer_attach(buff, size);        /* Create a buffer */

  x = me;

  if (me == 0) {     /* Process 0 does this */
    printf("Sending to process 1\n");
    MPI_Bsend(&x, 1, MPI_INT, 1, tag, MPI_COMM_WORLD);  /* Buffered
    send */
    printf("Receiving from process 1\n");
    MPI_Recv (&y, 1, MPI_INT, 1, tag, MPI_COMM_WORLD, &status);
    printf("Process %d received a message containing value %d\n", me,
    y);
  }
  else
  {          /* Process 1 does this */
    /* This program would work even though we changed the order of
    */
    /* the send and receive calls here, because the messages are
    */
    /* buffered and the processes can continue the execution without
    */
    /* waiting for the other process to receive the message
    */
    MPI_Recv (&y, 1, MPI_INT, 0, tag, MPI_COMM_WORLD, &status);
    MPI_Bsend (&x, 1, MPI_INT, 0, tag, MPI_COMM_WORLD);  /* Buffered
    send */

  }
  MPI_Buffer_detach(&buff, &size);  /* Detach the buffer */
  MPI_Finalize();
  exit(0);
}
```

3. **Non-blocking sending I**. MPI example code8.c using non-blocking send and receive to pass a message between two processes.

```c
/* A simple MPI example program using non-blocking send
   */
```

```c
/* The program does exactly the same as code6.c */

/* The program consists of one sender process and one receiver */
/* The sender process sends a message containing its identifier */
/* to the receiver. This receives the message and sends it back */
/* Both processes use non-blocking send and receive operations */
/* (MPI_Isend and MPI_Irecv, and MPI_Wait to wait until the message */
/* has arrived) */

/* Compile the program with 'mpicc -o code8 code8.c' */
/* Run the program with 'mpirun -np 2 code8 */

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

int main(int argc, char* argv[]) {
  int x, y, np, me;
  int tag = 42;
  MPI_Status status;
  MPI_Request send_req, recv_req;    /* Request object for send and receive */

  MPI_Init(&argc, &argv);                /* Initialize MPI */
  MPI_Comm_size(MPI_COMM_WORLD, &np);  /* Get number of processes */
  MPI_Comm_rank(MPI_COMM_WORLD, &me);  /* Get own identifier */

  x = me;
  if (me == 0) {    /* Process 0 does this */
    printf("Process %d sending\n",me);
    MPI_Isend(&x, 1, MPI_INT, 1, tag, MPI_COMM_WORLD, &send_req);
    printf("Process %d receiving\n", me);
    MPI_Irecv (&y, 1, MPI_INT, 1, tag, MPI_COMM_WORLD, &recv_req);
    /* We could do computations here while we are waiting for communication */
    MPI_Wait(&send_req, &status);
    MPI_Wait(&recv_req, &status);
    printf("Process %d received a message containing value %d\n", me, y);
  }
  else
  {
    MPI_Irecv (&y, 1, MPI_INT, 0, tag, MPI_COMM_WORLD, &recv_req);
    MPI_Isend (&x, 1, MPI_INT, 0, tag, MPI_COMM_WORLD, &send_req);
    /* We could do computations here while we are waiting for communication */
    MPI_Wait(&recv_req, &status);
    MPI_Wait(&send_req, &status);

  }
  MPI_Finalize();
  exit(0);
}
```

4. **Non-blocking sending II**. A simple MPI example code9.c using non-blocking send and receive. The sender process sends a message to all other processes. They receive the message and send an answer back.

```
/* processes. The sender process sends a message containing its */
/* identifier to all the other processes. These receive the message */
/* and replies with a message containing their own identifier */
/* Both processes use non-blocking send and receive operations */
/* (MPI_Isend and MPI_Irecv, and MPI_Waitall) */

/* Compile the program with 'mpicc -o code9 code9.c' */
/* Run the program with 'mpirun -np 4 code9 */

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

#define MAXPROC 8    /* Max number of procsses */

int main(int argc, char* argv[]) {
  int i, x, np, me;
  int tag = 42;

  MPI_Status status[MAXPROC];
  /* Request objects for non-blocking send and receive */
  MPI_Request send_req[MAXPROC], recv_req[MAXPROC];
  int y[MAXPROC];  /* Array to receive values in */

  MPI_Init(&argc, &argv);                 /* Initialize */
  MPI_Comm_size(MPI_COMM_WORLD, &np);     /* Get nr of processes */
  MPI_Comm_rank(MPI_COMM_WORLD, &me);     /* Get own identifier */

  x = me;   /* This is the value we send, the process id */
  if (me == 0) {    /* Process 0 does this */
    /* First check that we have at least 2 and at most MAXPROC
    processes */
    if (np<2 || np>MAXPROC) {
      printf("You have to use at lest 2 and at most %d processes\n",
    MAXPROC);
      MPI_Finalize();
      exit(0);
    }
    printf("Process %d sending to all other processes\n",me);
    /* Send a message containing the process id to all other processes
     */
    for (i=1; i<np; i++) {
      MPI_Isend(&x, 1, MPI_INT, i, tag, MPI_COMM_WORLD, &send_req[i]);
    }
    /* While the messages are delivered, we could do computations here
     */
    /* Wait until all messages have been sent */
    /* Note that we use requests and statuses starting from position 1
     */
    MPI_Waitall(np-1, &send_req[1], &status[1]);
```

```
46      printf("Process %d receiving from all other processes\n", me);
47      /* Receive a message from all other processes */
48      for (i=1; i<np; i++) {
49        MPI_Irecv (&y[i], 1, MPI_INT, i, tag, MPI_COMM_WORLD, &recv_req[
        i]);
50      }
51      /* While the messages are delivered, we could do computations here
         */
52      /* Wait until all messages have been received */
53      /* Requests and statuses start from position 1 */
54      MPI_Waitall(np-1, &recv_req[1], &status[1]);
55
56      /* Print out one line for each message we received */
57      for (i=1; i<np; i++) {
58        printf("Process %d received message from process %d\n", me, y[i
        ]);
59      }
60      printf("Process %d ready\n", me);
61    }
62    else
63    { /* all other processes do this */
64
65      /* Check sanity of the user */
66      if (np<2 || np>MAXPROC) {
67        MPI_Finalize();
68        exit(0);
69      }
70      MPI_Irecv (&y, 1, MPI_INT, 0, tag, MPI_COMM_WORLD, &recv_req[0]);
71      MPI_Wait(&recv_req[0], &status[0]);
72      MPI_Isend (&x, 1, MPI_INT, 0, tag, MPI_COMM_WORLD, &send_req[0]);
73      /* Lots of computations here */
74      MPI_Wait(&send_req[0], &status[0]);
75    }
76
77    MPI_Finalize();
78    exit(0);
79  }
```