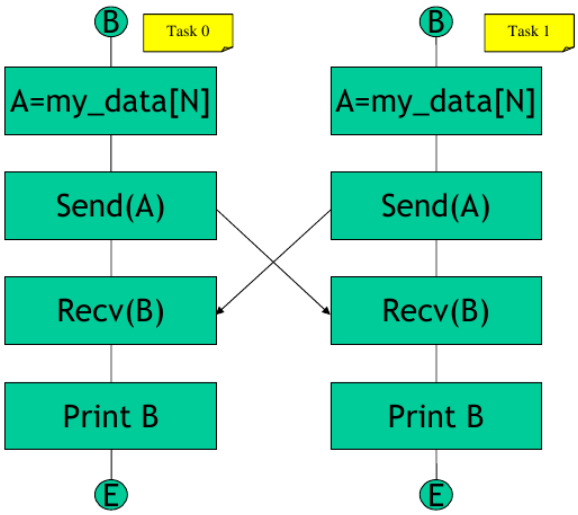
Q/A Exercise 2

ANSWERS:

Q- What happens if the order of the send/receive in task 1 is inverted?

A- Both processes try to send their buffer to the other process through blocking sends that do not return until the buffer is free from being used (i.e. a matching receive is posted or a system-dependent buffer is provided). Thus the execution of the following program



cannot go further the send calls and results in a deadlock.

Q- Try to reduce the SEND buffer A to 1000 elements and invert the order of send/receive in task 1. What happens when you run the code?

A- This exploits the system-dependent buffer provided for the blocking send; even if the program should terminate in a deadlock the send calls are able to return thanks to that buffer (the send buffer has been transfered to the system buffer so the program can continue with the next instruction).

Q- Rewrite the code without any IF statement and any deadlock. What do you need to use instead of the blocking SEND?

A- You can use the non blocking ISEND:

C

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

#define ndata 10000

int main(int argc, char* argv[]){

    int ierr, me, nprocs, i=0, you;
    MPI_Status status;
    MPI_Request req;

    float a[ndata];
    float b[ndata];

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &me);


    for(i=0;i<ndata;++i){
    a[i] = me;
    }

    if(nprocs>2 && me==0){
    printf("\n\tThis program must run on 2 processors");
    return 0;
    }

    you = 1-me;

    MPI_Isend(a, ndata, MPI_REAL, you, 0, MPI_COMM_WORLD, &req);
    MPI_Recv(b, ndata, MPI_REAL, you, 0, MPI_COMM_WORLD, &status);

    printf("\n\tI am task %d and I have received b(0) = %1.2f \n", me, b[0]);

    MPI_Finalize();
```

```
}
```

FORTRAN

```
program hello
  implicit none
  include 'mpif.h'

  integer ierr,me,nprocs,you,req
  integer status(MPI_STATUS_SIZE)

  integer,parameter :: ndata = 10000
  real :: a(ndata)
  real :: b(ndata)

  call MPI_INIT(ierr)
  call MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
  call MPI_COMM_RANK(MPI_COMM_WORLD, me, ierr)

  a = me

  if (nprocs /= 2) then
     print *,'This program must run on 2 processors'
     call MPI_ABORT(ierr)
     stop
  endif

  you = 1-me

  call MPI_ISEND(a,ndata,MPI_REAL,you,0,MPI_COMM_WORLD,req,ierr)
  call MPI_RECV(b,ndata,MPI_REAL,you,0,MPI_COMM_WORLD,status,ierr)

  print *,'I am task ',me,' and I have received b(0) = ',b(0)
  call MPI_FINALIZE(ierr)

end program hello
```

‹ Solution 2 up Exercise 3 ›