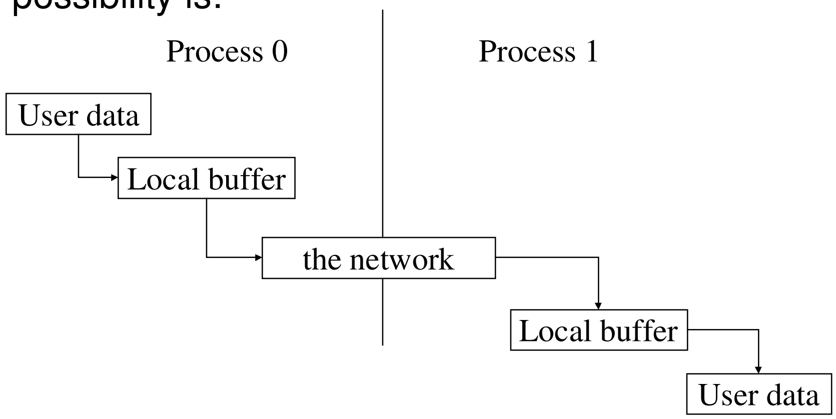
Buffers

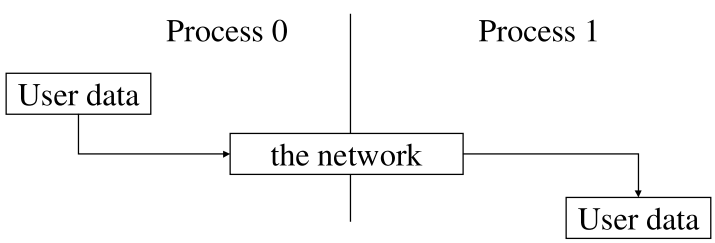
- When you send data, where does it go? One possibility is:



1

Avoiding Buffering

- It is better to avoid copies:



- This requires that **`MPI_Send`** wait on delivery, or that **`MPI_Send`** return before transfer is complete, and we wait later.

2

Blocking and Non-blocking Communication

- So far we have been using *blocking* communication:
  - » `MPI_Recv` does not complete until the buffer is full (available for use).
  - » `MPI_Send` does not complete until the buffer is empty (available for use).
- Completion depends on size of message and amount of system buffering.

3

Sources of Deadlocks

- Send a large message from process 0 to process 1
  - » If there is insufficient storage at the destination, the send must wait for the user to provide the memory space (through a receive)
- What happens with this code?

Process 0	Process 1
Send(1)	**Send(0)**
Recv(1)	**Recv(0)**

- This is called "unsafe" because it depends on the availability of system buffers

4

Some Solutions to the "unsafe" Problem

- Order the operations more carefully:

Process 0	Process 1
`Send(1)`	`Recv(0)`
`Recv(1)`	`Send(0)`

- Supply receive buffer at same time as send:

Process 0	Process 1
`Sendrecv(1)`	`Sendrecv(0)`

5

More Solutions to the "unsafe" Problem

- Supply own space as buffer for send

Process 0	Process 1
`Bsend(1)`	`Bsend(0)`
`Recv(1)`	`Recv(0)`

- Use non-blocking operations:

Process 0	Process 1
`Isend(1)`	`Isend(0)`
`Irecv(1)`	`Irecv(0)`
`Waitall`	`Waitall`

6

MPI's Non-blocking Operations

- Non-blocking operations return (immediately) "request handles" that can be tested and waited on.

```
MPI_Isend(start, count, datatype,
    dest, tag, comm, request)
MPI_Irecv(start, count, datatype,
    dest, tag, comm, request)
MPI_Wait(&request, &status)
```

- One can also test without waiting:

```
MPI_Test(&request, &flag, status)
```

7

Multiple Completions

- It is sometimes desirable to wait on multiple requests:

```
MPI_WAITALL(count, array_of_requests,
  array_of_statuses)
MPI_Waitany(count, array_of_requests,
  &index, &status)
MPI_Waitsome(count, array_of_requests,
  array_of indices, array_of_statuses)
```

- There are corresponding versions of **`test`** for each of these.

8

Communication Modes

- MPI provides multiple *modes* for sending messages:
  - » Synchronous mode (**`MPI_Ssend`**): the send does not complete until a matching receive has begun. (Unsafe programs deadlock.)
  - » Buffered mode (**`MPI_Bsend`**): the user supplies a buffer to the system for its use. (User allocates enough memory to make an unsafe program safe.
  - » Ready mode (**`MPI_Rsend`**): user guarantees that a matching receive has been posted.
    - – Allows access to fast protocols
    - – undefined behavior if matching receive not posted
- Non-blocking versions (**`MPI_Issend`**, etc.)
- **`MPI_Recv`** receives messages sent in any mode.

Buffered Mode

- When MPI_Isend is awkward to use (e.g. lots of small messages), the user can provide a buffer for the system to store messages that cannot immediately be sent.

```
int bufsize;
char *buf = malloc( bufsize );
MPI_Buffer_attach( buf, bufsize );
...
MPI_Bsend( ... same as MPI_Send ... )
...
MPI_Buffer_detach( &buf, &bufsize );
```

- **`MPI_Buffer_detach`** waits for completion.
- Performance depends on MPI implementation and size of message.

Exercise: Buffered Sends

- What is wrong with this code?

```
Call MPI_Buffer_attach( buf, &
          bufsize+MPI_BSEND_OVERHEAD, ierr )
Do i=1,n
    ...
    Call MPI_Bsend( bufsize bytes )
    ...
    Enough MPI_Recvs( )
enddo
call MPI_Buffer_detach( buf, bufsize, ierr )
```


Buffering is limited

- Processor 0
  i=0
  MPI_Bsend
  MPI_Recv
  i=1
  MPI_Bsend

- i=1 Bsend fails because first Bsend has not been able to deliver the data

- Processor 1
  i=0
  MPI_Bsend
  … delay due to computing, process scheduling,...
  MPI_Recv

6

Correct Use of MPI_Bsend

- Fix: Attach and detach buffer in loop
- ```
  Do i=1,n
      Call MPI_Buffer_attach( buf, &
            bufsize+MPI_BSEND_OVERHEAD, ierr )
      ...
      Call MPI_Bsend( bufsize bytes )
      ...
      Enough MPI_Recvs( )
      call MPI_Buffer_detach( buf, bufsize, ierr )
  enddo
  ```
- Buffer detach will wait until messages have been delivered

13

Other Point-to Point Features

- **`MPI_Sendrecv`**
- **`MPI_Sendrecv_replace`**
- **`MPI_Cancel`**
- Persistent requests

14
  ```

MPI_Sendrecv

- Allows simultaneous send and receive
- Everything else is general.
  - » Send and receive datatypes (even type signatures) may be different
  - » Can use Sendrecv with plain Send or Recv (or Irecv or Ssend_init, …)
  - » More general than “send left”

Process 0	Process 1
`SendRecv(1)`	**`SendRecv(0)`**