CompSci131

Parallel and Distributed Systems

(c) A. Veidenbaum

Prof. A. Veidenbaum

Today's topics

- **Berkeley sockets**
- **MPI**

- **Reading assignment:**
  - **Today: 4.3 *and* lecture notes**
  - **Next lecture: L10, MPI lecture notes**
    - » **Complete the assignment *before* next class**

(c) A. Veidenbaum

Last Lecture Covered

- **Communication**
- **RPC**

(c) A. Veidenbaum

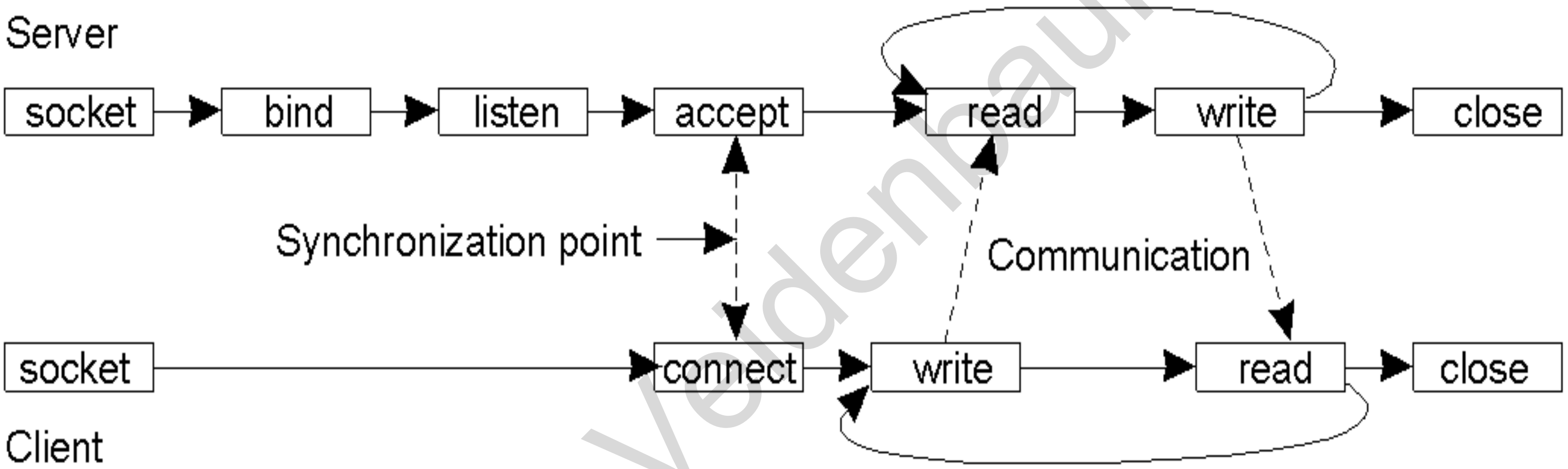
Message-Oriented Transient Communication Protocols

- **Berkeley Socket** primitives for TCP/IP
  - Read/write data and it will appear on the other end

Primitive	Meaning
Socket	**Create a new communication endpoint**
Bind	**Attach a local address (IP+port) to a socket**
Listen	**Announce willingness to accept connections**
Accept	**Block caller until a connection request arrives**
Connect	**Actively attempt to establish a connection**
Send	**Send some data over the connection**
Receive	**Receive some data over the connection**
Close	**Release the connection**

(c) A. Veidenbaum

Berkeley Sockets (2)



- **Server:**
  - Create local address for a socket (bind), reserve buffer (listen)
  - Wait for requests (accept), block until the next one arrives
- **Client:**
  - request connection to server, block until connection is set up

(c) A. Veidenbaum

Berkeley Sockets (3)

- How does a server handle multiple connections?
- *listen* specifies maximum number of connections
  - Allows the OS to reserve enough buffer space
- *accept* "wakes up" on every connection request
  - Allows the OS to fork off a new process to handle the new connection
  - Server goes back and listens for new *connect*
- Client uses transport-level address to connect
  - *connect* blocks until the connection is established

(c) A. Veidenbaum

The Message Passing Interface (MPI)

- Used to program a single application
  - on a multi-computer
- Uses the Single-Program Multiple-Data model
  - SPMD
  - All nodes run the same program on different local data
- Has a number of communication modes, primitives
  - Assumes reliable communication and thus uses transient mode
- Has data distribution primitives

(c) A. Veidenbaum

What exactly is MPI?

- **A Message Passing Interface**
  - **Not a new language!**
  - **A standard for communication between processors**
    - » **The MPI-1 standard was defined in Spring of 1994**
    - » **MPI-2, -3, and 4 have also been defined**
      - **Major additions: parallel I/O, dynamic process management**

- **The standard specifies**
  - **names, calling sequences, and results of its functions**
  - **The functions can be called from Fortran and C/C++**
  - **All implementations of MPI must conform to the standard**
    - » **ensuring portability**

(c) A. Veidenbaum

How to write MPI programs?

- **Use standard C programming + parallelism, communication**

```
#include <mpi.h>
 void main (int argc, char *argv[]) {
 int err;
 err = MPI_Init(&argc, &argv);   /* join the Collective */
       ...   /*compute smth*/
       err = MPI_Finalize();                 /* exit */
```

(c) A. Veidenbaum

How to write MPI programs?

- **After MPI_Init each process becomes part of the MPI *world***
  - **Is ready to execute and communicate with others**
  - **Each participating process starts the same program on its node**
    - » **Can find out its own id (rank), total number of nodes**

(c) A. Veidenbaum

What does MPI provide?

- **Point-to-point communication**
  - **Send, Recv primitives**
  - **Communicates <u>variables</u> of the MPI types**
    - » **including aggregated types**
- **Collective communication**
  - **One-to-many and many-to-one**
    - » **Broadcast and "reduce" (with different operators)**
      - **Reductions: X = SUM(A[I]), 0<I<N**
    - » **Same syntax on all processors**

(c) A Veidenbaum

- **Synchronization**
  - **Barrier**
    - » ***Every* node has to reach this point in a program before *all* continue**
    - » **What about other types of synchronization?**
- **Static process creation (in MPI-1)**
    - » **Can be dynamic in MPI-2 and later**

(c) A. Veidenbaum

Point-to-point communication

- Explicit Send() and Recv() primitives
  - Parameters:
    - » user buffer, data type, count (address, type, length)
      - Data type can be user defined
  - Can hide type conversion (different byte ordering, floats)
- Basic Send/Recv are Blocking
  - The Send function only returns when data is received by the remote task
  - The Recv blocks until it gets data
    - » This is <u>synchronous</u> communication
- Now can describe a simple MPI subset
  - *sufficient to start programming…*

(c) A. Veidenbaum.

A 6-function MPI

- **MPI_Init(&argc, &argv) - start MPI**
- **MPI_Finalize() - end MPI**
- **MPI_COMM_SIZE - number of nodes**
- **MPI_COMM_RANK - my id (rank)**
- **MPI_SEND**
  - **Parameters: (start, count, datatype, dest, tag, comm)**
- **MPI_RECV**
  - **Parameters: (start, count, datatype, source, tag, comm, status)**

(c) A. Veidenbaum

- **6-function MPI uses**
  - **Blocking send/receive**
  - **Synchronization explicit in send/receive**
    - » **Deadlock possible**

(c) A. Veidenbaum

Other types of MPI primitives

- **Has a number of communication modes, primitives**

Primitive	Meaning
MPI_bsend	**Buffered send - append outgoing message to a local send buffer**
MPI_send	**Send a message and wait until copied to local or remote buffer**
MPI_ssend	**Synchronous send a message and wait until receipt starts**
MPI_sendrecv	**Send a message and wait for reply**
MPI_isend	**Pass reference to outgoing message, and continue**
MPI_issend	**Pass reference to outgoing message, and wait until receipt starts**
MPI_recv	**Receive a message; block if there are none**
MPI_irecv	**Check if there is an incoming message, but do not block**

(c) A. Veidenbaum

- **MPI_Irecv(&buf, count, datatype, source, tag, comm, &request)**
- **MPI_Wait(&request, &status)**
  - **Now block until Irecv is finished**

- **MPI_Test(&request, &flag, &status)**
  - **Flag=1 when finished, else 0. status={src,tag}**
- **MPI_Get_count(&status, datatype, &count)**
  - **Returns the count of elements received**

(c) Veidenbau

Buffering

- **Where is the message buffered?**
- **A critical issue for good performance**
- **The are three data exchange mechanisms**
  - **Eager (MPICH default)**
    - » **data is sent to the destination immediately**
  - **Rendezvous**
    - » **When a receive is posted**
      - ***Control is always sent***
  - **Get**
    - » **Receiver reads data directly**
      - **Best on PGAS systems**

(c) A. Veidenbaum

Sending Modes

- **Synchronous mode (MPI_Ssend):**
  - **the send does not complete until a matching receive has begun**
- **Buffered mode (MPI_Bsend)**
  - **the user supplies the buffer to system for its use**
- **Ready mode (MPI_Rsend)**
  - **user guarantees that matching receive has been posted.**
    - » ***undefined behavior if the matching receive is not posted***
- **Non-blocking versions**
  - **MPI_Issend, MPI_Irsend, MPI_Ibsend**

- ***Note that an MPI_Recv may receive messages sent with any send mode.***

(c) A. Veidenbaum