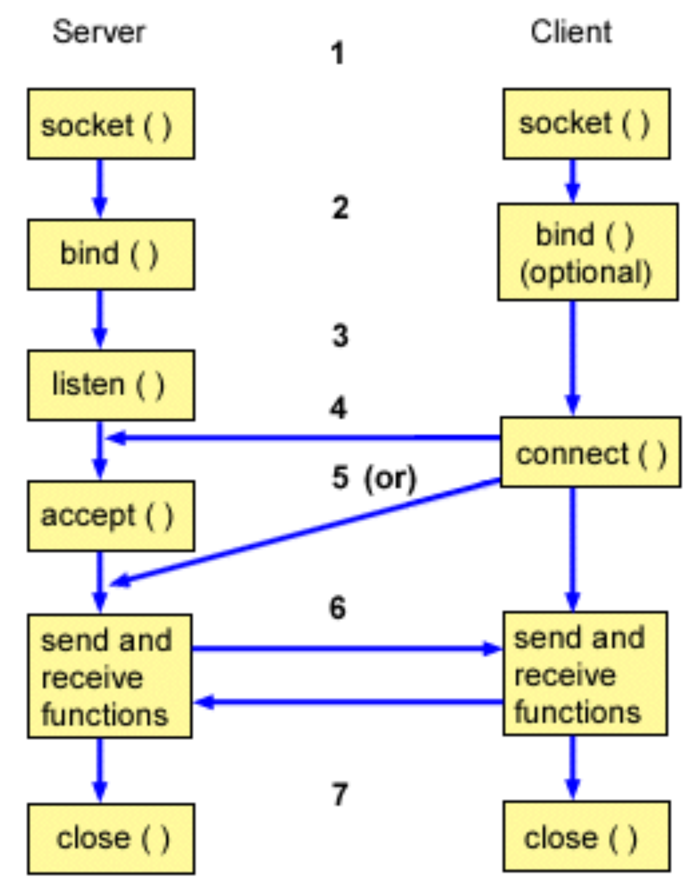
Sockets

- Interprocess connection in a distributed environment.
- Socket communication domains
  - UNIX domain (can be used to implement pipes)
    - names are filenames
  - Internet domain
    - names are IP addresses, names or numbers
    - plus port number
  - NS domain (Xerox communication protocols)
  - ISO OSI protocols
  - etc
- Internet types
  - stream – bidirectional, reliable, sequenced, unduplicated. No record boundaries. Similar to pipes.
  - datagram – bidirectional, but not reliable, sequenced or unduplicated. Record boundaries are preserved. (Packet switched networks like Ethernet.)
  - raw – access to the underlying protocols which support sockets (available in routers and other network equipment)
  - also non-internet sockets (other transport protocols)

Socket calls

- Setting up a socket
  - socket - make a socket, specify the domain and protocol
  - bind - associate a name with the socket
  - listen - now ready to get connections
  - accept - gets a connection and returns a new socket (used for the actual communication)
- another process (for the other end of the socket):
  - socket - make a socket
  - connect - makes the connection between this socket and the named one
- Then normal read and write operations can be performed on the socket.
- Only one process bound to each port.
- select – can be used to read from multiple sockets when data becomes available.



from https://www.ibm.com/support/knowledgecenter/en/ssw_ibm_i_71/rzab6/howdosockets.htm

Communicating via shared resources

Shared resources

- Separate processes can alter the resource.
- Need to check the state of the resource to either receive data or know that some event has occurred.
- Usually need to explicitly coordinate access to the resource.

- What if the information I want isn't there yet?
  When do I try again?

Files

- Easy to use but slow.
- File system may provide synchronization help, e.g. only one writer at a time.

Memory

- Fast
- Synchronization is usually handled explicitly by the processes.

Shared memory

- Different threads in the same process automatically share memory. How can we share memory between different heavyweight processes?
  - define sections of shared memory,
  - attach the shared memory to the process,
  - detach, indicate who can access it etc.
- Both processes need to know the name of the area of shared memory.
- Must make sure the memory is attached to some unused area of the process's address space.
- Usual security checks - can this process attach to this chunk of memory?
- What about if the processes are on separate machines?

Shared memory in Linux processes

Remember that fork copies data.

```
#include <stdio.h>
#include <unistd.h>
#include <sys/mman.h>

void main() {
    void *shared;
    int *a_number;
    int b_number;

    shared = mmap(NULL, 4096, PROT_READ | PROT_WRITE, MAP_ANONYMOUS | MAP_SHARED, -1, 0);
    printf("Program 1 - shared:%p\n", shared);

    a_number = (int *)shared;
    printf("a_number: %d, b_number: %d\n", *a_number, b_number);

    if (fork() != 0) {
        *a_number = 12345;
        b_number = 54321;
    }
    printf("a_number: %d, b_number: %d\n", *a_number, b_number);
}
```

Distributed shared memory - DSM

- Shared memory between processes running on different machines.
- A natural method to share information.
- Processes don't need to be changed to run on a distributed system.
- Slow.
- May need many messages to transfer the shared memory or parts of it across the network.
- Extra complications to coordinate use of the shared memory.

Message passing	Distributed shared memory
Variables have to be marshalled	Variables are shared directly
Cost of communication is obvious	Cost of communication is invisible
Processes are protected by having private address space	Processes could cause error by altering data
Processes should execute at the same time	Executing the processes may happen with non-overlapping lifetimes

https://en.wikipedia.org/wiki/Distributed_shared_memory

DSM software implementation

- Copy the memory to whichever machine has a process sharing it.
- Mark it read only.
- If the process writes, memory access fault, kernel determines it is shared memory
- and sends a write request to the originating machine, this can broadcast the change to other processors to update their copies.

Optimisations

- Maybe only copy some of the shared memory - copy on **read**.
- Simplified if only one process is allowed to write - readers/writers problem.
- Same benefits from distributed object technology CORBA and RMI.

Distributed concurrency

- Locks, semaphores and monitors require shared memory.
- Doesn’t matter whether a single processor or multiprocessor.
- Sometimes we need locks over resources which are available network wide.
- No shared memory.
- Which means we are going to have to send messages.

Centralized method

- The easiest solution to allocating resources safely is to use one process on one machine to coordinate access to a resource.
- We call this a server or coordinator process.
- Request - Reply - Release
- A process wanting the resource or mutual exclusion requests it with a message to the coordinator and then blocks until it receives a reply.
- When it receives the reply it has the resource and must send a release message when it has finished.

Fully distributed method

- We want decisions made across the entire system.
- So every request must be broadcast to all other processes in case the resource is currently being used.
- The process continues with secure access after it hears back from all other processes in the system.
- If a process is inside the critical section it defers its reply until it leaves the section.
- If a process is not inside the critical section and does not want to enter it replies immediately.
- If it also wants to enter the critical section it checks to see which request *happened earlier*.

What happened first?

- We can't rely on synchronized clocks in a distributed system.
- One clock will run slower than another.
- Use logical clocks instead.
- Each processor keeps timestamps for its processes.
  - The system-wide timestamp is the local timestamp with the processor identifier concatenated on the end (just like in the Bakery algorithm).
- When a message is sent from one processor to another it carries a timestamp.
- If the received timestamp is later than the current logical time of the receiving processor the logical time is bumped up.

Token-passing method

- The fully distributed approach has some fundamental problems:
  - The processes must know all about each other.
  - Processes are assumed not to fail.
- There are solutions to these problems but token-passing is a cleaner method.
  - A token gets passed around the system – one token per critical section.
  - (A logical, if not a physical, ring of processes.)
  - A process can’t enter the critical section until it gets and holds on to the token.
  - Processes pass the token on when they no longer want to enter the critical section.
- Problems
  - Tokens can get lost.
  - Processes can die and rings are broken.

Complications

- Communication can be unreliable or some processes may fail.
- Coordinators may use time-outs if the resources aren't released.
- They can send queries to see if the current owners are still active.
- If the coordinator fails, the using processes need to have an election to see which process should replace it.
  (See the Bully algorithm in the 8th edition of the textbook 18.6.1)
- When a process detects the coordinator is not available it starts an election to see if it should be the coordinator.
  The process with the highest id gets elected.
  A recovering process with a higher id is a bully and becomes the coordinator.
- The new coordinator needs to recreate a wait queue by polling all processes to see if they need the resource.

Before next time

- Read from the textbook

  13.1 File Concept

  13.2 Access Methods