1. **Introduction to Network Programming**

Network applications are everywhere. Any time you browse the Web, send an email message, or pop up an X window, you are using a network application. Interestingly, all network applications are based on the same basic programming model, have similar overall logical structures, and rely on the same programming interface.

Every network application is based on the client-server model. With this model, an application consists of a server process and one or more client processes. A server manages some resource, and it provides some service for its clients by manipulating that resource. For example, a Web server manages a set of disk files that it retrieves and executes on behalf of clients. An FTP server manages a set of disk files that it stores and retrieves for clients. Similarly, an email server manages a spool file that it reads and updates for clients. Clients and servers often run on separate hosts and communicate using the hardware and software resources of a computer network [1].

The fundamental operation in the client-server model is the transaction (Figure 1). A client-server transaction consists of four steps:

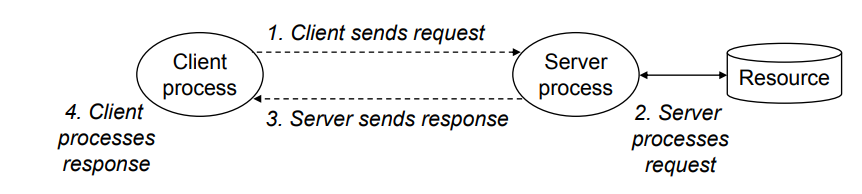


Figure Client server transaction

1. When a client needs service, it initiates a transaction by sending a request to the server. For example, when a Web browser needs a file, it sends a request to a Web server.
2. The server receives the request, interprets it, and manipulates its resources in the appropriate way. For example, when a Web server receives a request from a browser, it reads a disk file.
3. The server sends a response to the client, and then waits for the next request. For example, a Web server sends the file back to a client.
4. The client receives the response and manipulates it. For example, after a Web browser receives a page from the server, it displays it on the screen.
   1. What is a Socket?

A network socket is one endpoint in a communication flow between two programs running over a network. Sockets are created and used with a set of programming requests or "function calls" sometimes called the sockets application programming interface (API). The most common sockets API is the Berkeley UNIX C interface for sockets. Sockets can also be used for communication between processes within the same computer [2].

Sockets allow communication between two different processes on the same or different machines. To be more precise, it's a way to talk to other computers using standard Unix file descriptors. In Unix, every I/O action is done by writing or reading a file descriptor. A file descriptor is just an integer associated with an open file and it can be a network connection, a text file, a terminal, or something else. To a programmer, a socket looks and behaves much like a low-level file descriptor. This is because commands such as read() and write() work with sockets in the same way they do with files and pipes.

* + 1. **Socket Types**

There are four types of sockets available to the users. The first two are most commonly used and the last two are rarely used.

Processes are presumed to communicate only between sockets of the same type but there is no restriction that prevents communication between sockets of different types [3].

* **Stream Sockets** − Delivery in a networked environment is guaranteed. If you send through the stream socket three items "A, B, C", they will arrive in the same order − "A, B, C". These sockets use TCP (Transmission Control Protocol) for data transmission. If delivery is impossible, the sender receives an error indicator. Data records do not have any boundaries.
* **Datagram Sockets** − Delivery in a networked environment is not guaranteed. They're connectionless because you don't need to have an open connection as in Stream Sockets − you build a packet with the destination information and send it out. They use UDP (User Datagram Protocol).
* **Raw Sockets** − these provide users access to the underlying communication protocols, which support socket abstractions. These sockets are normally datagram oriented, though their exact characteristics are dependent on the interface provided by the protocol. Raw sockets are not intended for the general user; they have been provided mainly for those interested in developing new communication protocols, or for gaining access to some of the more cryptic facilities of an existing protocol.
* **Sequenced Packet Sockets** − they are similar to a stream socket, with the exception that record boundaries are preserved. This interface is provided only as a part of the Network Systems (NS) socket abstraction, and is very important in most serious NS applications. Sequenced-packet sockets allow the user to manipulate the Sequence Packet Protocol (SPP) or Internet Datagram Protocol (IDP) headers on a packet or a group of packets, either by writing a prototype header along with whatever data is to be sent, or by specifying a default header to be used with all outgoing data, and allows the user to receive the headers on incoming packets.

References

1. <http://csapp.cs.cmu.edu/2e/ch11-preview.pdf>
2. <http://whatis.techtarget.com/definition/sockets>
3. <https://www.tutorialspoint.com/unix_sockets/what_is_socket.htm>