CS 240 Programming in C

Socket Programming

April 27 , 2022

Process communication

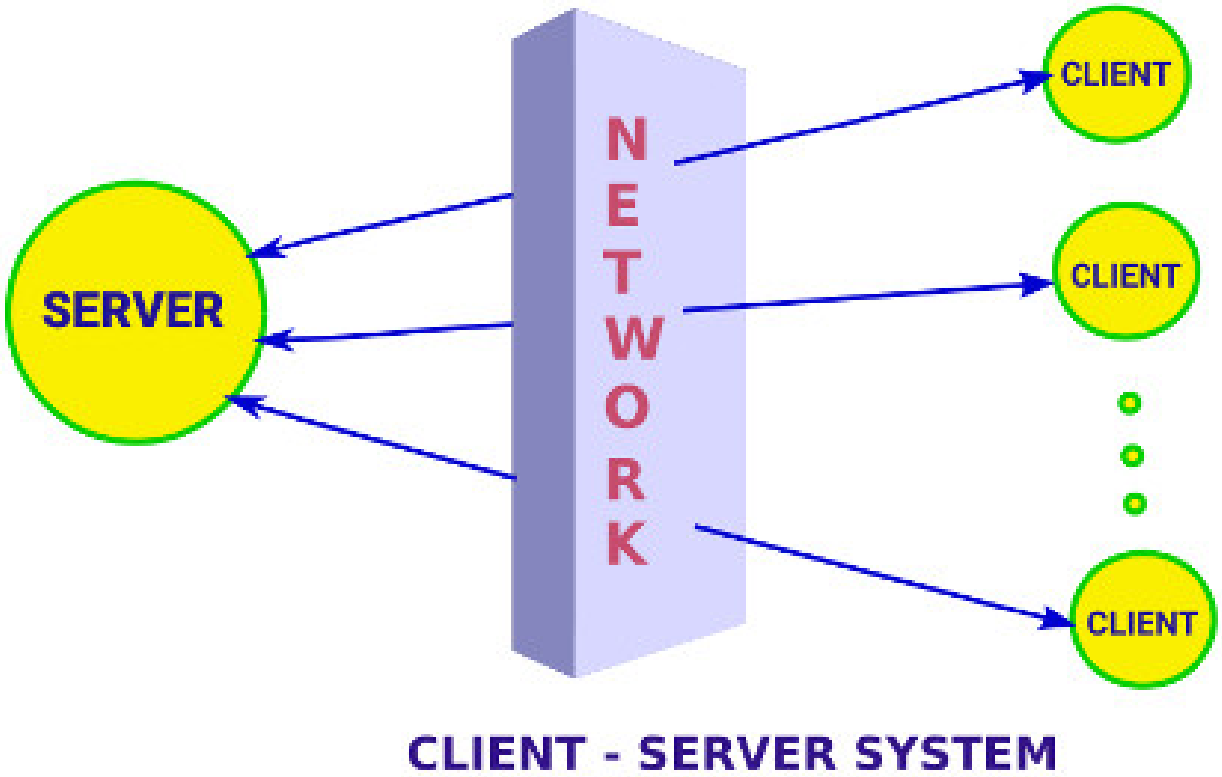
Typically two processes communicate with each other on a single system through one of the following inter process communication techniques.

- Pipes
- Message queues
- Shared memory

There are several other methods. But the above are some of the very common ways of inter-process communication.

Client Server System

Client server model is a software architecture paradigm prevalent in distributed applications. A server has information resources and processes that provide answers to queries and other services to remote clients over the network.



Connection?

How does a client identify the server with which it wants to communicate?

Connection

A client needs to know two things about the server - the Internet Protocol (IP) address and the port number. The IP address can be a traditional 32-bit address or a 128-bit newer address expressing as 2001:db8::1:2:3:4, where each character is a hexadecimal digit representing four bits of address.

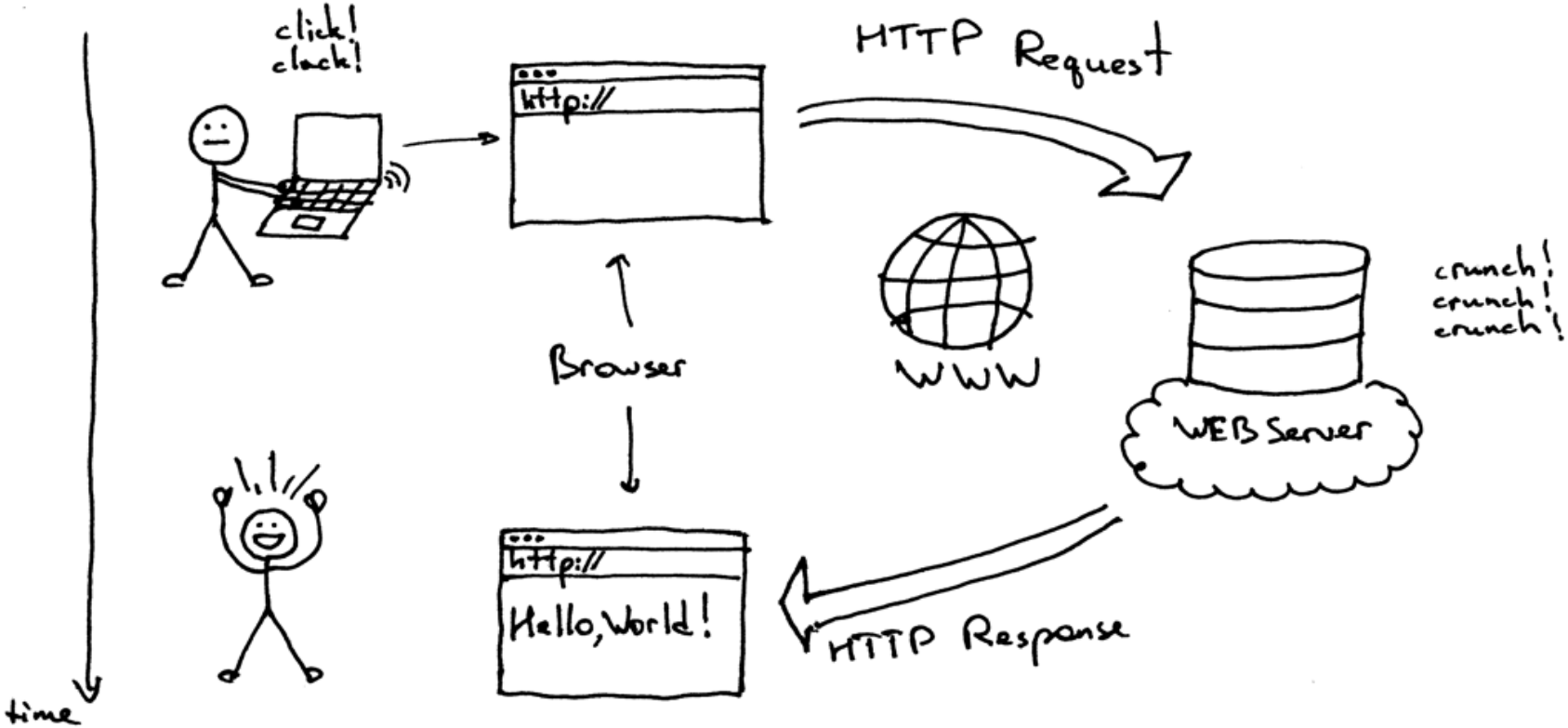
Socket

Sockets are used for communication between a server and a client process. The server's code runs first, which opens a port and listens for incoming connection requests from clients. Once a client connects to the same (server) port, the client or server may send a message. A socket is a network communication end point at a host.

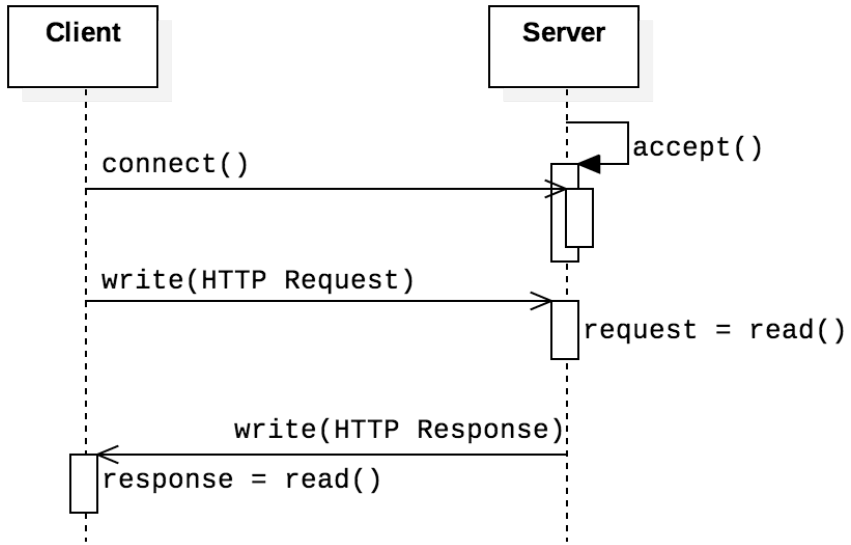
Socket

Sockets allow you to exchange information between processes on the same machine or across a network, distribute work to the most efficient machine, and they easily allow access to centralized data. Socket application program interfaces (APIs) are the network standard for TCP/IP.

Real world use: Web request

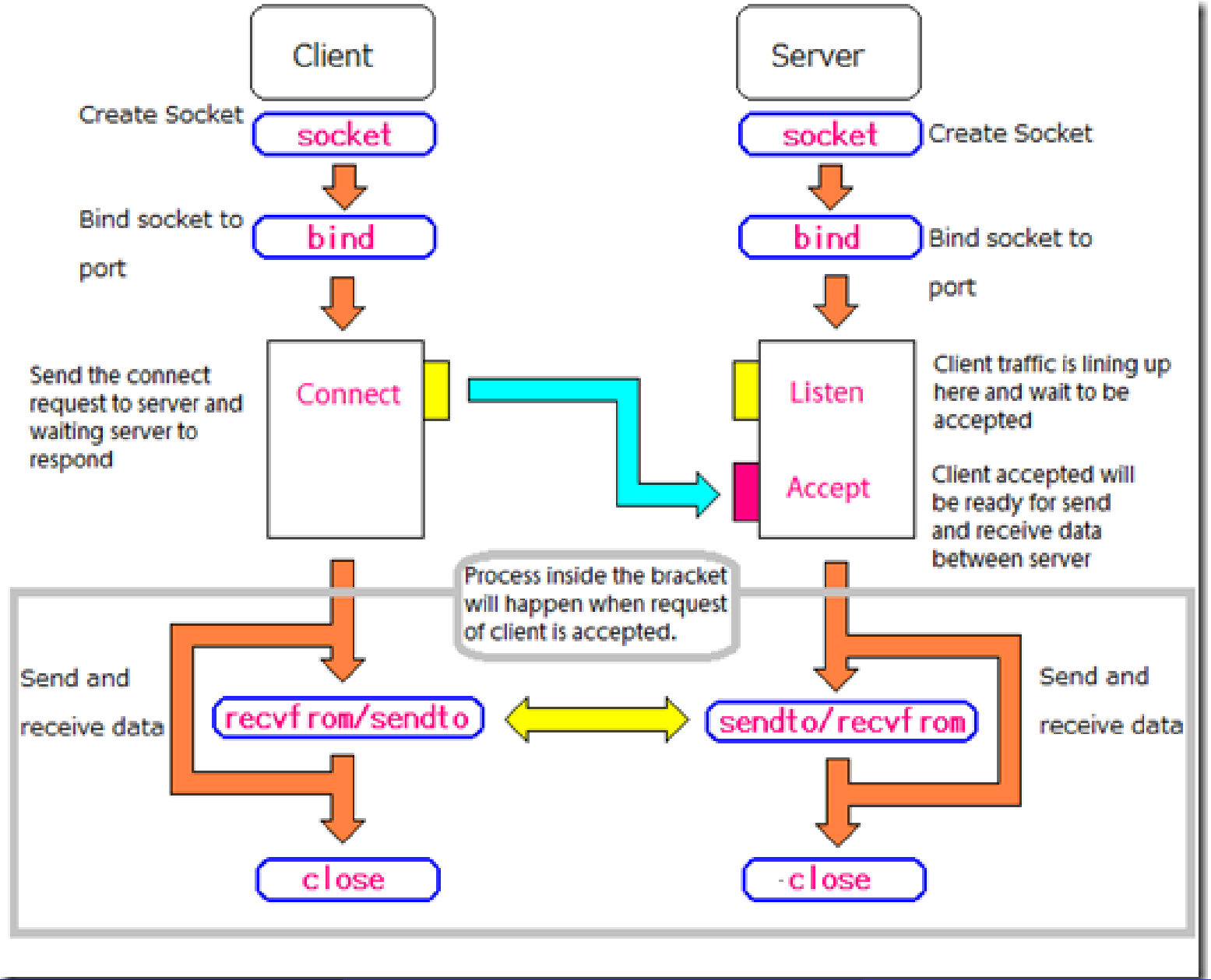


Hypertext Transfer Protocol (HTTP)



HTTP is a simple request-response protocol, defined in RFC 2616. It defines communication for web browsers and servers. Figure shows the basic structure of HTTP in relation to the functions that establish the socket connection. The client—a web browser sends an HTTP request to the server and receives a response. HTTP applications use TCP connections for their transport layer.

Socket programming in C: flowchart



File Descriptor

- In Unix and Unix-like computer operating systems, a file descriptor (FD, less frequently fildes) is a process-unique identifier (handle) for a file or other input/output resource, such as a pipe or network socket.
- File descriptors typically have non-negative integer values, with negative values being reserved to indicate "no value" or error conditions.

int	Name	symbolic const	file stream
0	Standard input	STDIN_FILENO	stdin
1	Standard output	STDOUT_FILENO	stdout
2	Standard error	STDERR_FILENO	stderr

Socket creation

- sockfd: socket descriptor, an integer (like a file-handle)
- domain: integer, specifies communication domain.
  - AF_LOCAL: Same host local machine
  - AF_INET: IPv4 Host
  - AF_INET: IPv6 Host
  - SOCK_STREAM: TCP(reliable, connection oriented)
  - SOCK_DGRAM: UDP(unreliable, connectionless)
- protocol: Protocol value for Internet Protocol(IP), which is 0. This is the same number which appears on protocol field in the IP header of a packet.

Code

```
int sockfd = socket(domain, type, protocol)
```

Setsockopt

This helps in manipulating options for the socket referred by the file descriptor sockfd. This is completely optional, but it helps in reuse of address and port. Prevents error such as: “address already in use”.

Setsockopt

- sockfd: socket descriptor
- level: integer, specifies the protocol level at which the option resides.
  - SOL_SOCKET: Socket options
  - SOL_IP: IP options
  - SOL_TCP: TCP options
  - SOL_UDP: UDP options
- optname: integer, specifies a single option to set
- optval: pointer to buffer in which the value for the requested option(s) are to be returned
- optlen: integer, specifies the size of the buffer pointed to by the optval argument

Code

```
int setsockopt(int sockfd, int level, int optname, const void *optval,
socklen_t optlen);
```

Bind

- sockfd: socket descriptor
- myaddr: pointer to a sockaddr structure containing the address to be bound to the socket
- addrlen: integer, size of the address structure

Code

```
int bind(int sockfd, const struct sockaddr *myaddr, socklen_t addrlen);
```

Listen

- sockfd: socket descriptor
- backlog: integer, specifies the maximum length to which the queue of pending connections for sockfd may grow

Code

```
int listen(int sockfd, int backlog);
```

Accept

- sockfd: socket descriptor
- addr: pointer to a sockaddr structure. This structure is filled in with the address of the peer socket, as known to the communications layer. The exact format of the address returned addr is determined by the socket's address family
- addrlen: integer, size of the address structure

Code

```
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

Connect

- sockfd: socket descriptor
- serv_addr: pointer to a sockaddr structure containing the address of the server to which the connection should be established
- addrlen: integer, size of the address structure

Code

```
int connect(int sockfd, const struct sockaddr *serv_addr, socklen_t addrlen);
```

Send

- sockfd: socket descriptor
- buffer: pointer to a buffer containing the message to be sent
- length: integer, length of the message in bytes
- flags: integer, specifies the type of message transmission

Code

```
int send(int sockfd, const void *buffer, size_t length, int flags);
```

Recv

- sockfd: socket descriptor
- buffer: pointer to a buffer where the message should be stored
- length: integer, length of the buffer in bytes
- flags: integer, specifies the type of message reception

Code

```
int recv(int sockfd, void *buffer, size_t length, int flags);
```

Server Program: time_server.c

- In time_server.c program, we have created a server. In the code :
- The call to the function 'socket()' creates an UN-named socket inside the kernel and returns an integer known as socket descriptor
- This function takes domain/family as its first argument. For Internet family of IPv4 addresses we use AF_INET
- The second argument 'SOCK_STREAM' specifies that the transport layer protocol that we want should be reliable ie it should have acknowledgement techniques. For example : TC
- The third argument is generally left zero to let the kernel decide the default protocol to use for this connection. For connection oriented reliable connections, the default protocol used is TCP
- The call to the function 'bind()' assigns the details specified in the structure 'serv_addr' to the socket created in the step above. The details include, the family/domain, the interface to listen on(in case the system has multiple interfaces to network) and the port on which the server will wait for the client requests to come

Server Program: time_server.c

- The call to the function 'listen()' with second argument as '10' specifies maximum number of client connections that server will queue for this listening socket
- After the call to listen(), this socket becomes a fully functional listening socket
- In the call to accept(), the server is put to sleep and when for an incoming client request, the three way TCP handshake* is complete, the function accept () wakes up and returns the socket descriptor representing the client socket
- The call to accept() is run in an infinite loop so that the server is always running and the delay or sleep of 1 sec ensures that this server does not eat up all of your CPU processing
- As soon as server gets a request from client, it prepares the date and time and writes on the client socket through the descriptor returned by accept().

Client Program: time_client.c

- In the time_client.c program, we create a client which will connect to the server and receive date and time from it. In the above piece of code :
- We see that here also, a socket is created through call to socket() function
- Information like IP address of the remote host and its port is bundled up in a structure and a call to function connect() is made which tries to connect this socket with the socket (IP address and port) of the remote host
- Note that here we have not bind our client socket on a particular port as client generally use port assigned by kernel as client can have its socket associated with any port but In case of server it has to be a well known socket, so known servers bind to a specific port like HTTP server runs on port 80 etc while there is no such restrictions on clients
- Once the sockets are connected, the server sends the data (date+time) on clients socket through clients socket descriptor and client can read it through normal read call on the its socket descriptor.

Resource Links

- https://www.geeksforgeeks.org/socket-programming-cc/
- https://www.tutorialspoint.com/unix_sockets/network_addresses.htm
- https://www.softprayog.in/programming/network-socket-programming-using-tcp-in-c
- https://www.thegeekstuff.com/2011/12/c-socket-programming/