Sockets

Irfan Sheikh

Department of Computer Science, University of Delhi

July 2021

Table of Contents

Introduction to Sockets

Problems in IPC over a Network

- Messages are structured according to the type of network and protocol being used
- Therefore, the methods by which processes over a network establish communications are likely to differ
- Processes need to know what type of network they are talking to - goes against the Unix philosophy

Sockets

Sockets were developed in order to:

- Provide common methods for IPC
- Allow use of sophisticated network protocols

A socket is an **endpoint** of a bidirectional communication path over a network.

The Client-Server Model

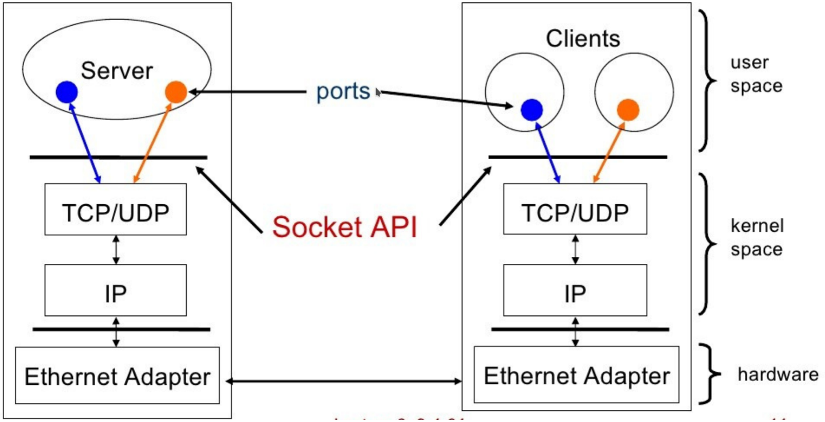


Figure: Client-Server Communication using Sockets

Image Source

Client-Server Communication

Kernel Structure - three layers

1. Sockets Layer: provides the interface between the system calls and the lower layers for network communication
2. Protocol Layer: contains the protocol modules used for communication (e.g. TCP (or UDP) / IP)
3. Device Layer: contains the device drivers that control the network devices

Communication overview

Endpoint 1: A server process creates a socket on startup and listens on that socket for incoming connection requests by clients.
Endpoint 2: A client process creates another socket, and sends either a connection request followed by subsequent messages (TCP), or direct messages (UDP) to the server.

Socket Domains

Domains

Sockets that share common communication properties (naming conventions, protocol address formats, etc) are grouped into **domains**. Two such domains are commonly used.

Internet Domain Sockets

- IPC between processes on different machines over a network

Socket Domains

Domains

Sockets that share common communication properties (naming conventions, protocol address formats, etc) are grouped into **domains**. Two such domains are commonly used.

Internet Domain Sockets

- IPC between processes on different machines over a network
- Uses port numbers, and, IPv4 or IPv6 addresses

Socket Domains

Domains

Sockets that share common communication properties (naming conventions, protocol address formats, etc) are grouped into **domains**. Two such domains are commonly used.

Internet Domain Sockets

- IPC between processes on different machines over a network
- Uses port numbers, and, IPv4 or IPv6 addresses

Unix Domain Sockets

- IPC between processes executing on a host OS on the same machine

Socket Domains

Domains

Sockets that share common communication properties (naming conventions, protocol address formats, etc) are grouped into **domains**. Two such domains are commonly used.

Internet Domain Sockets

- IPC between processes on <u>different machines</u> over a network
- Uses port numbers, and, IPv4 or IPv6 addresses

Unix Domain Sockets

- IPC between processes executing on a host OS on the <u>same machine</u>
- Uses the file system as the address name space

Internet Domain Sockets

- Also called Network Sockets - communication over a network
- Socket address - transport protocol, port number, and IP addr

Types of Network Sockets

Stream Network Sockets

- Connection-oriented sockets - reliable byte stream - sequenced, unique flow of error-free data
- Message boundaries are not preserved
- Default protocol: TCP - high protocol overhead

Datagram Network Sockets

- Connection-less sockets - unreliable delivery and order of datagrams, error checking optional
- Message boundaries are preserved
- Default protocol: UDP - lower protocol overhead

Raw Network Sockets allow direct transfer of IP packets without any protocol-specific transport layer formatting. e.g. Nmap.

Unix Domain Sockets

- IPC between processes on the same machine - two processes can communicate by opening the same socket (can be thought of as a full duplex pipe)
- Similar to Network Sockets, but rather than using an underlying network protocol, all communication occurs entirely within the OS kernel

Types of Unix Domain Sockets

Stream Unix Sockets

- Stream-oriented socket - provides a continuous stream of bytes (relate to TCP)
- Reliable as usual
- Message boundaries are not preserved

Datagram Unix Sockets

- Datagram-oriented socket - transfers discrete chunks of data (relate to UDP)
- Reliable - they don't reorder datagrams
- Message boundaries are preserved

Unix Sockets vs Network Sockets

Comparison

Unix Sockets

- let you use file permissions to restrict access to unix sockets (services)
- let you transfer special things like open file descriptors
- both processes execute on the same system - no need to perform checks and operations as in network sockets

Network Sockets

- regulating access to network sockets is difficult - need packet filtering, firewall rules
- they have network protocol overhead since things like network routing and error detection have to be performed

Time to Think

Question

Is it possible to use Network Sockets for processes executing on the same system?

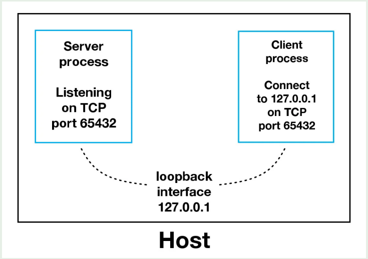
Time to Think

Question

Is it possible to use Network Sockets for processes executing on the same system?

Answer

Yes, by using the Virtual Loopback Interface (`lo` or `lo0` on Unix-like systems). Image Source.


Server process
Listening on TCP port 65432
Client process
Connect to 127.0.0.1 on TCP port 65432
loopback interface 127.0.0.1
Host

Berkeley / POSIX Sockets

- Berkeley sockets - API for Network Domain and Unix Domain Sockets
- Evolved with little modification into a component of the POSIX specification.
- BSD and POSIX sockets are essentially synonymous with each other.

Communication using Stream Network Sockets

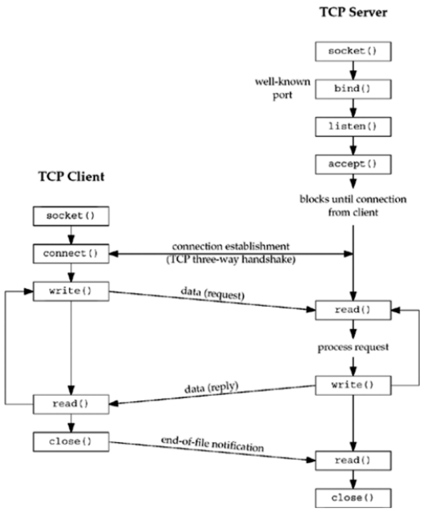


Figure: Socket functions for TCP client/server. Taken from Fig 4.1, Unix Network Programming, Vol. 1, 3rd edition

Communication using Datagram Network Sockets

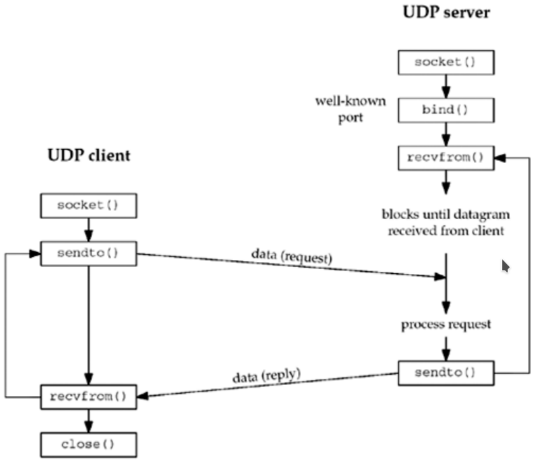
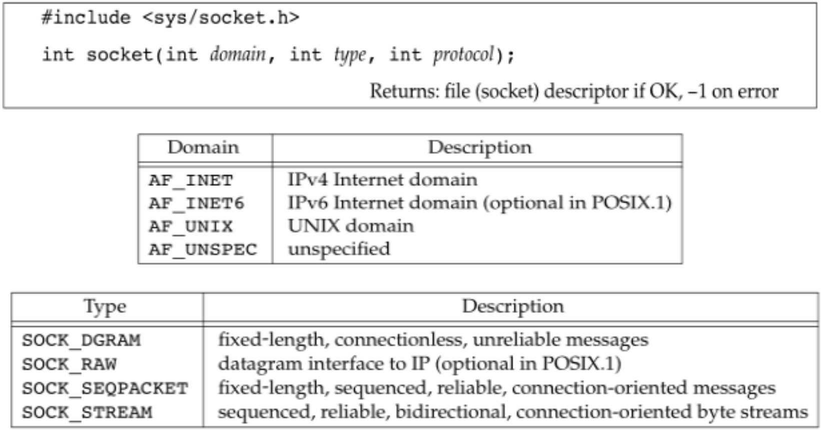


Figure: Socket functions for UDP client/server. Taken from Fig 8.1, Unix Network Programming, Vol. 1, 3rd edition

Socket Descriptors

- To create a socket, we call the `socket` function.



```
#include <sys/socket.h>

int socket(int domain, int type, int protocol);
```

Returns: file (socket) descriptor if OK, –1 on error

Domain	Description
`AF_INET`	IPv4 Internet domain
`AF_INET6`	IPv6 Internet domain (optional in POSIX.1)
`AF_UNIX`	UNIX domain
`AF_UNSPEC`	unspecified

Type	Description
`SOCK_DGRAM`	fixed-length, connectionless, unreliable messages
`SOCK_RAW`	datagram interface to IP (optional in POSIX.1)
`SOCK_SEQPACKET`	fixed-length, sequenced, reliable, connection-oriented messages
`SOCK_STREAM`	sequenced, reliable, bidirectional, connection-oriented byte streams

Figure: `socket()` syscall Taken from Sec 16.2, APUE, 3rd edition

Recall the Protocols

In the `socket()` function, if we pass 0 in the `protocol` field, then a default protocol gets chosen for us for the specified `domain` and `type`.

Question

What would be the default protocol for the following combinations of Socket domains and types?

1. domain: `AF_INET`, type: `SOCK_STREAM`
2. domain: `AF_INET`, type: `SOCK_DGRAM`

Recall the Protocols

In the `socket()` function, if we pass 0 in the `protocol` field, then a default protocol gets chosen for us for the specified `domain` and `type`.

Question

What would be the default protocol for the following combinations of Socket domains and types?

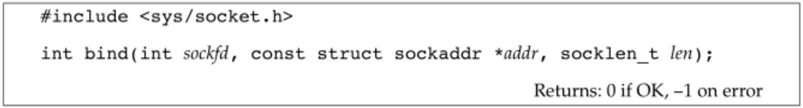
1. domain: `AF_INET`, type: `SOCK_STREAM`
2. domain: `AF_INET`, type: `SOCK_DGRAM`

Answer

1. TCP (`IPPROTO_TCP`)
2. UDP (`IPPROTO_UDP`)

Addressing Sockets

We use the `bind` function to associate an address with a socket.



```
#include <sys/socket.h>

int bind(int sockfd, const struct sockaddr *addr, socklen_t len);
```

Returns: 0 if OK, –1 on error

- A server needs to associate a well-known address with the its socket on which client requests will arrive. e.g. port 80 (HTTP), port 22 (SSH)
- For a client, binding is not necessary. We can let the system choose a default socket address by calling `connect()` or `listen()` without calling `bind()`.

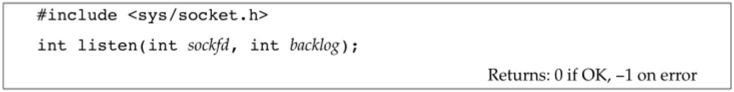
Establishing Connections : `connect()`

- For stream network socket, before we can exchange data, we need to create a connection between the socket of the process requesting the service (the client) and the process providing the service (the server). The `connect` function creates a connection.
- For a datagram network socket, if we call `connect`, the destination address of all messages we send is set to the address we specified in the connect call, relieving us from having to provide the address every time we transmit a message. In addition, we will receive datagrams only from the address we've specified.

```
#include <sys/socket.h>

int connect(int sockfd, const struct sockaddr *addr, socklen_t len);

                                         Returns: 0 if OK, −1 on error
```

Establishing Connections : `listen()`

- A server announces that it is willing to accept connect requests by calling the `listen()` function.



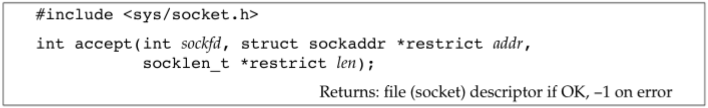
```
#include <sys/socket.h>

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

Returns: 0 if OK, –1 on error

- `backlog` specifies the number of outstanding connection requests that can be enqueued.
- Once the queue is full, the system will reject additional connect requests.

Establishing Connections : `accept()`

- Once a server has called `listen()`, the socket can receive connection requests.
- The `accept()` function retrieves a connection request and converts it into a connection.



```
#include <sys/socket.h>

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

Returns: file (socket) descriptor if OK, −1 on error

- `accept()` returns a socket descriptor that is connected to the client that called `connect`.
- The original socket passed to `accept()` is not associated with the connection, but instead remains available to receive additional connection requests.

Transferring Data : `read()` and `write()`

- Socket endpoint is represented as a file descriptor
- Therefore, we can use `read()` and `write()` to communicate with a socket, as long as it is connected.
- This means that we can pass socket descriptors to functions that were originally designed to work with local files. We can also arrange to pass the socket descriptors to child processes that execute programs that know nothing about sockets.
- If we want to specify options, receive packets from multiple clients, or send out-of-band data, we need to use one of the six socket functions designed for data transfer.

Transferring Data : specialized functions

- `send` - similar to `write`, but allows us to specify some options to control how we send the data.
- `sendto` - is similar to `send`, but it allows us to specify a destination address to be used with connection-less sockets.
- `sendmsg` - allows us to specify multiple buffers from which to transmit data, using a `msghdr` structure.
- `recv` - similar to `read`, but allows us to specify some options to control how we receive the data.
- `recvfrom` - similar to `recv`, but allows us to retrieve the identity (address) of the sender. Used with connection-less sockets.
- `recvmsg` - allows us to specify multiple input buffers to be used to receive the data, again using a `msghdr` structure.

Clean up

- `close()` frees up the socket descriptor for reuse and deallocates the endpoint if the last active reference is closed.
- `shutdown()` allows us to disable data transmission and / or reception on the socket. It deactivates a socket independtly of the number of active file descriptors referencing it.

Example : MySQL

MySQL

- The server `mysqld` listens for connections from client programs and manages access to databases on behalf of those clients.
- Clients (`mysql` command-line client, `phpMyAdmin`, etc) on Unix can connect to the `mysqld` server in two different ways.
  1. Using a Unix socket to connect through a file in the file system (default `/tmp/mysql.sock`). It is faster than a TCP/IP socket, but can be used only when connecting to a server on the same computer.
  2. Using a TCP/IP socket, which connects through a port number.

Example : MySQL

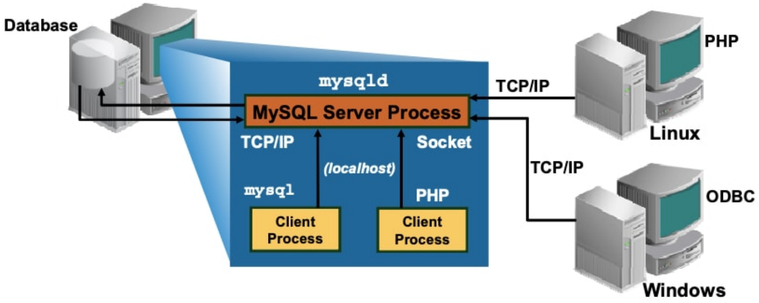


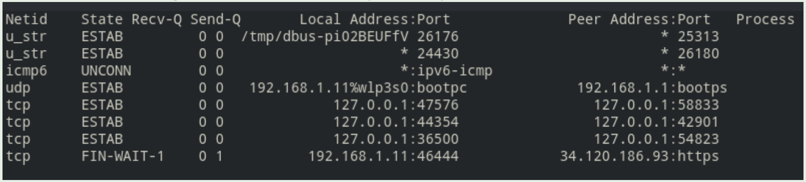
Figure: MySQL Client-Server

Image Source

Investigating Sockets

ss: socket statistics

Used to investgate sockets on your system.



```
Netid     State Recv-Q Send-Q       Local Address:Port              Peer Address:Port   Process
u_str     ESTAB         0  0  /tmp/dbus-pi02BEUFfV 26176                       * 25313
u_str     ESTAB         0  0                     * 24430                       * 26180
icmp6     UNCONN        0  0                     *:ipv6-icmp                 *:*
udp       ESTAB         0  0   192.168.1.11%wlp3s0:bootpc           192.168.1.1:bootps
tcp       ESTAB         0  0           127.0.0.1:47576              127.0.0.1:58833
tcp       ESTAB         0  0           127.0.0.1:44354              127.0.0.1:42901
tcp       ESTAB         0  0           127.0.0.1:36500              127.0.0.1:54823
tcp       FIN-WAIT-1    0  1        192.168.1.11:46444          34.120.186.93:https
```

References

1. The Design of the Unix Operating System, Maurice J. Bach
2. Advanced Programming in the Unix Environment, 3rd edition, W. Richard Stevens, Stephen A. Rago
3. Unix Network Programming, Volume 1, 3rd edition, W. Richard Stevens, Bill Fenner, Andrew M. Rudoff
4. Wikipedia links: Unix Domain Socket, Network Socket, Berkeley Sockets API