System Programming

Week 15: Sockets

Seongjin Lee

Updated: 2016-12-14

12_misc

insight@hanyang.ac.kr

http://esos.hanyang.ac.kr

Esos Lab. Hanyang University

Table of contents

Sockets

Sockets

Socket netwok IPC interface that allow to communicate with other process

- either on the same machine or
- on different machine over a network

Client-Server Setup

Sockets are used as follows:

- Applications create a socket
- Server bind its sockets to well-known addresss
- locate server sockets via its well-known address and communicate with the server

Socket syscall

Sockets are created using the socket syscall which returns a file descriptor to be used for further operations on the underlying socket:

```
#include <sys/socket.h>
fd = socket(domain, type, protocol)
// Returns: file descriptor on success, -1 on error
```

- Domain: `AF_UNIX`, `AF_INET`, `AF_INET6`
- Each communication domain determines
  - how to identify a socket, that is the syntax and semantics of socket well-known addresses
  - the communication range, e.g. single or multiple hosts

Communication Domains

domain	range	transport	address format	address C struct
`AF_UNIX`	same host	kernel	pathname	`sockaddr_un`
`AF_INET`	any host w/ IPv4 connectivity	IPv4 stack	32-bit IPv4 address + 16bit port number	`sockaddr_in`
`AF_INET6`	any host w/ IPv6 connectivity	IPv6 stack	128-bit IPv6 address + 16bit port number	`sockaddr_in6`

Socket Types

Socket types offer different IPC features

Feature	SOCK_STREAM	SOCK_DGRAM
realiable delivery	yes	no
fixed length	no	yes
connection-oriented	yes	no

Stream Sockets : SOCK_STREAM

Stream sockets provide communication channels which are:

- byte-stream: communication happens as a continuous stream of bytes
- reliable: either data transmitted arrive at destination, or the sender gets an error
- bidirectional: between two sockets, data can be transmitted in either direction
- connection-oriented: sockets operate in connected pairs, each connected pair of sockets denotes a communication context, isolated from other pairs

Datagram Sockets : SOCK_DGRAM

Datagram sockets provide communication channels which are:

- message-oriented: data is exchanged at the granularity of messages that peers send to one another; Message length is fixed
- non-reliable: messages can get lost. Also:
  - messages can arrive out of order
  - messages can be duplicated and arrive multiple times

  It is up to applications to detect these scenarios and react (e.g. by re-sending messages after a timeout, adding sequence numbers, etc.).
- connection-less: sockets do not need to be connected in pairs to be used; you can send a message to, or receive a message from, a socket without connecting to it beforehand

Binding sockets to a well-known address

To allow connections from others, we need to bind sockets to well-known addresses using bind:

```
#include <sys/socket.h>
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
// Returns: 0 on success, -1 on error
```

- `sockfd` references the socket we want to bind
- `sockaddr addr` and `addrlen` depends on the socket domain

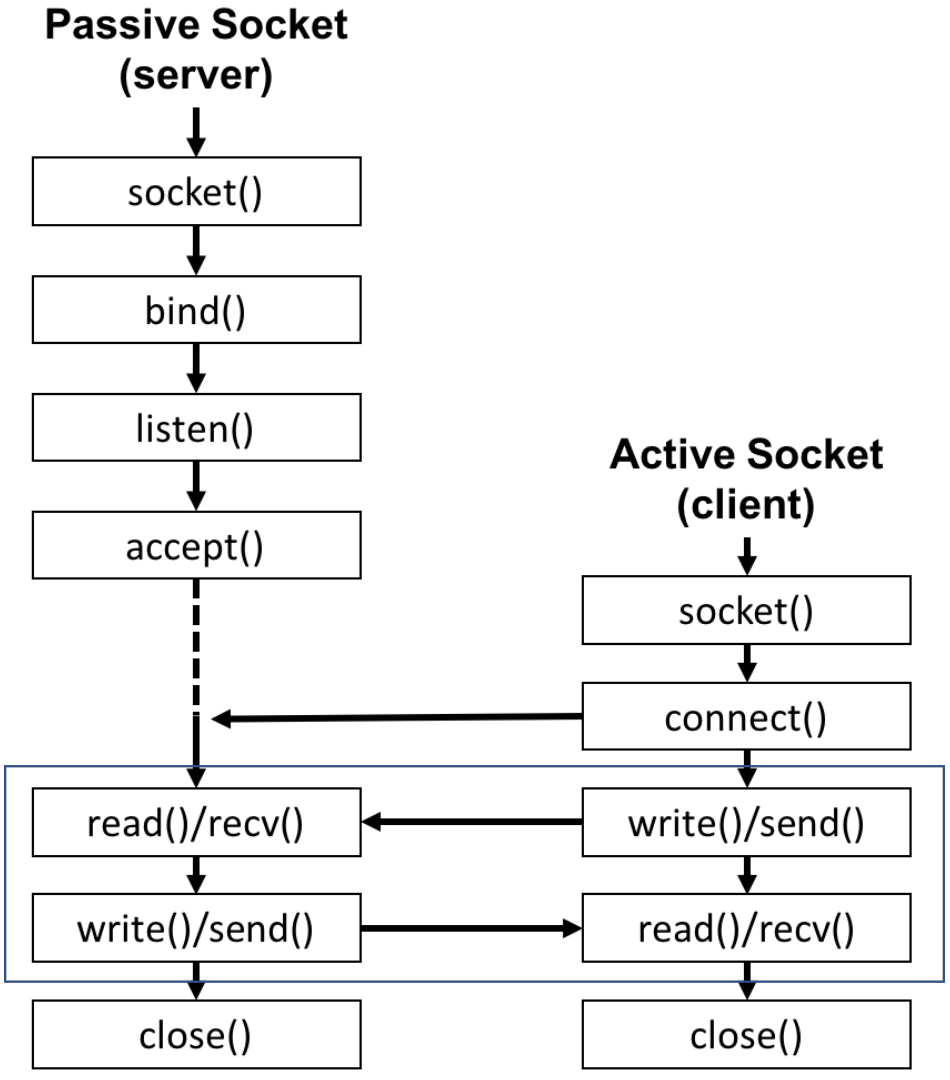
Generic Socket Address Structure

An address identifies a socket endpoint in a particular communication domain. The address format is specific to the particular domain.

```
struct sockaddr {
    sa_family_t sa_family; /* address family */
    char sa_data[14]; /* socket address (size varies with the socke domain) */
};
```

- Each socket domain has its own variant of `sockaddr`
- user has to fill the domain-specific struct and cast it to `struct sockaddr` before passing it to bind

Stream socket syscalls - overview



"Server" and "Client" are ambiguous terms
Passive and Active sockets are more precise

- sockets are created active : `listen()` makes them passive
- `connect()` performs an active open
- `accept()` performs a passive open

Active to passive sockets

`listen()` turns an active socket into a passive one, allowing it to accept incoming connections

```
#include <sys/socket.h>
int listen(int sockfd, int backlog);
// Returns: 0 on success, -1 on error
```

`backlog` specifies the maximum number of pending connections that the passive socket will keep

- active opens may be performed before the matching passive ones
- not yet accepted connections are called pending
- `pending < backlog` connect succeeds immediately
- `pending >= backlog` connect blocks waiting for an accept

Accepting Connections

You can accept connections (i.e. perform a passive open) with:

```
#include <sys/socket.h>
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
// Returns: file descriptor on success, -1 on error
```

- If the corresponding active open hasn't been performed yet, accept blocks waiting for it.
- When the active open happens—or if it has already happened—accept returns a new socket connected to the peer socket.
- The original socket remains available and can be used to accept other connections.

Connecting the socket

you connect (i.e. perform an active open) with:

```
#include <sys/socket.h>
int connect(int sockfd, struct sockaddr *addr, socklen_t addrlen);
// Returns: 0 on success, -1 on error
```

- `sockfd` is your own socket, to be used as your endpoint of the connection
- `addr/addrlen` specify the well-known address of the peer you want to connect to, and are given in the same format of bind parameters

Creating Sockets

```
int listenfd; /* Socket to listen on */
int optval = 1; /* Used by setsockopt() below */
if ((listenfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
     perror("Cannot create socket");
     exit(1);
}

/* Allow this socket to reuse a port number */
if (setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR,
           (const void*)&optval, sizeof(int)) < 0) {
     perror("Cannot set SO_REUSEADDR socket option");
     exit(1);
}
```

- create a socket: `socket()` system call
- use `setsockopt()` to permit socket to reuse the server port number

Binding Listening

```
struct sockaddr_in server_addr;
bzero(&server_addr, sizeof(server_addr)); /* Zero out the server address */
server_addr.sin_family = AF_INET;
/* Listen for connections from any client on the Internet. */
server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
server_addr.sin_port = htons(80); /* Listen for connections on port 80 */
/* Bind socket to address */
if (bind(listenfd, (struct sockaddr *)&server_addr,
                        sizeof(server_addr)) < 0) {
    perror("Cannot bind socket");
    exit(1);
}
/* Listen for incoming connections. Use listen queue length of 10. */
if (listen(listenfd, 10) < 0) {
    perror("Cannot listen");
    exit(1);
}
```

- `bind()` the socket to the port you want to listen on
  - `INADDR_ANY` to accept connections from any IP address
- `listen()` for incoming connections

Accept Connection

```
int clifd;
struct sockaddr_in client_addr;
int client_addr_len = sizeof(client_addr);
while (1) {
     clifd = accept(listenfd,
     (struct sockaddr *)&client_addr,
                  (socklen_t*)&client_addr_len);
     /* Process connection here */
     do_something(clifd);
     close(clifd);
}
```

- The server spins in a loop doing the following:
  - Call accept() to accept an incoming connection from the Internet (blocks until a connection is received)
  - Process the connection close() the client socket
- `accept()` returns a new file descriptor representing the socket for the new connection.

Process the connection

```
rio_t rio; /* Use the RIO libraries to do I/O */
    Rio_readinitb(&rio, clifd);
/* Read lines from the client */
while ((n = Rio_readlineb(&rio, buf, BUFSIZE)) != 0) {
      fprintf(stderr, "Read line from client: %s\n", buf);
      if (strcmp(buf, "\r\n") == 0) {
        // Got a blank line, means request is done.
break; }
}
    /* Send the client some HTML */
    sprintf(buf, "<html><body><b>Hello from my awesome web server.</b><p>You are
coming from %s with IP address %s.<br>Have a nice day!</body></html>\n",
     hostname, hostip);
    /* Send data back to client */
Rio_writen(clifd, buf, strlen(buf));
```

More info on

http://www.binarytides.com/socket-programming-c-linux-tutorial/

Last Words

Last Words

- prepare for the exam