The POSIX Socket API

Giovanni Agosta

Piattaforme Software per la Rete – Modulo 2

Outline

1. Sockets & TCP Connections
2. Socket API
3. UNIX-domain Sockets
4. TCP Sockets

TCP Connections

Preliminaries

TCP Application Interfaces

- Loosely specified
- Multiple implementations (Berkeley Sockets, System V TLI)
- Finally, POSIX socket API

POSIX Sockets API

- (Mostly) Unix networking interface abstraction
- Bidirection communication device
- Allows many different underlying protocols (not just TCP)
- Also abstracts inter process communication (IPC)

Socket Concepts

Communication style

- Data is sent in *packets*
- Communications style determines packet handling and addressing

Communication styles

- Connection (Stream and sequential sockets)
  - In-order delivery
  - Automatic request for retrasmission of lost/reordered packets
- Reliably Delivered Messages
  - No in-order delivery guarantee
  - Automatic request for retrasmission of lost packets
- Datagram
  - No in-order delivery guarantee
  - Actually, no delivery guaranty at all
- Raw

Socket Concepts
Namespaces & Protocols

Namespaces

Define how socket addresses are written

- Local namespace
  - Socket addresses are filenames
- Internet namespace
  - Socket addresses are IP addresses plus port numbers
  - Port numbers allow multiple sockets on the same host

Protocols

Specify the underlying protocol for transmitting data

- IP protocol family
- IP version 6
- UNIX local communication

Socket Concepts

Protocol-Style Combinations

Protocol	**Style**				
	SOCK_STREAM	SOCK_DGRAM	SOCK_RAW	SOCK_RDM	SOCK_SEQPACKET
PF_LOCAL	†	†			
PF_INET	TCP	UDP	IPv4		
PF_INET6	TCP	UDP	IPv6		
PF_NETLINK		†	†		
PF_X25					†
PF_APPLETALK		†	†		
PF_PACKET		†	†		

†Valid combination, with no special name

Socket API

Socket Representation and System Calls

Representation

- File descriptors are employed to represent sockets
- Once communication is established, POSIX I/O calls are used

System Calls

socket	Creates a socket
close	Destroys a socket
connect	Creates a connection between two sockets
bind	Labels a server socket with an address
listen	Configures a socket to accept conditions
accept	Accepts a connection and creates a new socket for the connection

Socket API
socket

Prototype

```
#include <sys/socket.h> int socket(int domain, int
type, int protocol)
```

Operation

- Creates a socket (data structure in the file table)
- Takes three parameters

domain	Socket domain (protocol family, e.g., PF_LOCAL, PF_INET)
type	Socket type (communication style)
protocol	Protocol (generally implicit)

- Returns a file descriptor (positive integer) if successful, -1 otherwise

Socket API

close

Prototype

```
#include <unistd.h> int close(int f)
```

Operation

- Closes the socket
- Actually, since it is a file descriptor, this is just the usual `close` call
- Returns 0 if successful

Socket API

connect

Prototype

```
#include <sys/socket.h>
int connect(int sockfd, const struct sockaddr
*serv_addr, socklen_t addrlen);
```

Operation

- Connects the socked `sockfd` to the specified (remote) address
- `addrlen` is an integer (size of the `sockaddr` structure)
- Connectionless sockets can use `connect` multiple times, to change the associated address

Socket API
bind

Prototype

```
#include <sys/socket.h>
int bind(int sockfd, const struct sockaddr *addr,
socklen_t addrlen);
```

Operation

- Assigns a local address to the socket
- Necessary to make a socket visible outside the process
- The `sockaddr` structure depends on the address family
- Returns 0 if successful, -1 otherwise

Socket API

listen

Prototype

```
#include <sys/socket.h>
int listen(int sockfd, int backlog);
```

Operation

- Marks the socket as *passive*
- The socket must use the SOCK_STREAM or SOCK_SEQPACKET styles
- It will then be used to accept incoming connections
- backlog is the maximum length of the pending connections queue
- Returns 0 if successful, -1 otherwise

Socket API

accept

Prototype

```
#include <sys/socket.h>
int accept(int sockfd, struct sockaddr *addr,
socklen_t *addrlen);
```

Operation

- Used in passive connection-based sockets
- `addr` is filled with the peer socket address
- `addrlen` initially contains the size of `addr` in memory, replaced with the actual size
- Returns the file descriptor (positive integer) for the accepted socket if successful, -1 otherwise

Socket Address

Generic data structure

Rationale

- Addresses differ for the various protocols
- Different structures must be used
- Sockets are rather old (1982), and a non-ANSI C workaround was used instead of **void** ∗

The sockaddr structure

```
struct sockaddr {
  sa_family_t sa_family ; /* AF_xxx */
  char sa_data [14]; /* address */
}
```

Socket Address

Data types for sockaddr structures

Type	Description	Header
`int8_t`	signed 8 bit integer	`sys/texttts.h`
`uint8_t`	unsigned 8 bit integer	`sys/texttts.h`
`int16_t`	signed 16 bit integer	`sys/texttts.h`
`uint16_t`	unsigned 16 bit integer	`sys/texttts.h`
`int32_t`	signed 32 bit integer	`sys/texttts.h`
`uint32_t`	unsigned 32 bit integer	`sys/texttts.h`
`sa_family_t`	address family	`sys/socket.h`
`socklen_t`	address struct length (`uint32_t`)	`sys/socket.h`
`in_addr_t`	IPv4 address (`uint32_t`)	`netinet/in.h`
`in_port_t`	TCP or UDP port (`uint16_t`)	`netinet/in.h`

Local Sockets

Using Sockets as IPC

Why?

- Provide communication between programs/processes
- Use the same socket abstraction

How to use local/UNIX sockets

- Namespace: `PF_LOCAL` or `PF_UNIX`
- Use the **struct** sockaddr_un
- The filename must be up to 108 bytes
- The actual length is computed using SUN_LEN

Local/UNIX Socket Address

The sockaddr structures

```
#define UNIX_PATH_MAX    108

struct sockaddr_un {
    /* AF_UNIX */
    sa_family_t  sun_family;
    /* pathname */
    char         sun_path[UNIX_PATH_MAX];
};
```

Values

- sun_family = AF_LOCAL or sun_family = AF_UNIX
- sun_path must be a file *pathname*

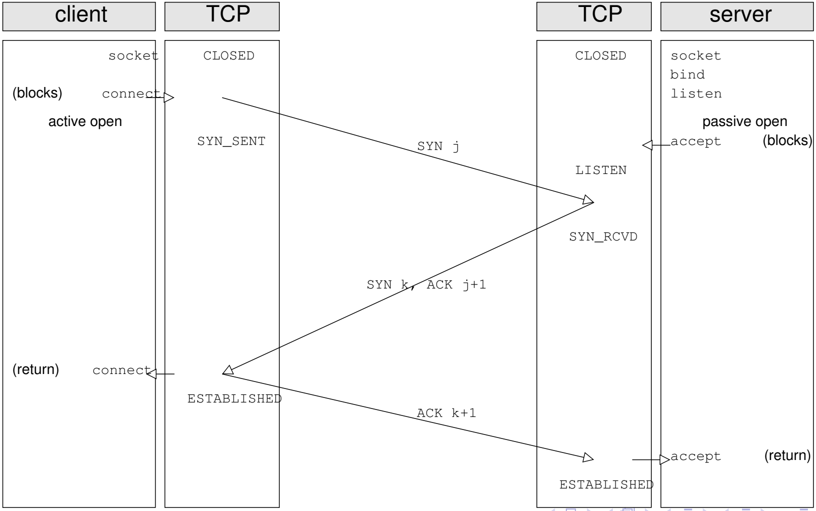
TCP Sockets

Generalities

- Connection-based socket
- Two sockets are involved, with different roles
  - One socket (*server*) accept the connection
  - The other socket (*client*) establishes the connection
- The client needs to know the server in advance, but not vice versa

TCP Connections

Three Way Handshake



IPv4 Socket Address

The sockaddr structures

```
struct sockaddr_in {
    /* address family: AF_INET */
    sa_family_t       sin_family;
    /* port in network byte order (big-endian) */
    in_port_t         sin_port;
    /* internet address */
    struct in_addr    sin_addr;
};
/* Internet address. */
struct in_addr {
    /* address in network byte order */
    in_addr_t         s_addr;
};
```

TCP Connections

Well-known ports

- TCP (and UDP) define *well-known* ports
- Well-known ports are assigned numbers from 0 to 1023
- The remaiing ports are divided in registered (up to 49151) and dynamic (up to 65535)
- In Linux and BSD, well-known ports are *reserved* to processes with administration rights (only root can start the ssh server, e.g.)
- Also, ports 1024–4999 and 61000–65535 are used as ephemeral ports (i.e., for client sockets)