

Introduction to Socket Programming

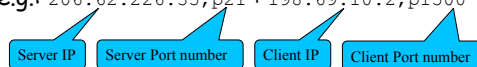


Agenda

- Basics of socket programming
 - Use of port numbers
 - Data structures for socket addressing
- TCP communication through socket programming
- UDP communication through socket programming

Socket Basics

- End point determined by two things:
 - **Host address**: IP address is *Network Layer*
 - **Port number**: is *Transport Layer*
- Two end-points determine a connection: socket pair
 - e.g.: 206.62.226.35, p21 + 198.69.10.2, p1500



Page 4

Quick question

- If a **Web browser** is engaged in a TCP communication with a **Web server**, what is the pair of sockets that can be used to represent their TCP connection?

Ports

- Numbers (typical, since vary by OS):
 - 0-1023 "reserved"
 - 1024 - 5000 "ephemeral"
 - Above 5000 for general use
- Well-known, reserved services (see /etc/services in Unix):
 - ftp 21/tcp
 - telnet 23/tcp
 - http 80/tcp
 - snmp 161/udp

Page 6

Transport Layer

- **UDP**: User Datagram Protocol
 - no acknowledgements
 - no retransmissions
 - out of order, duplicates possible
 - connectionless
- **TCP**: Transmission Control Protocol
 - reliable (in order, all arrive, no duplicates)
 - flow control
 - Connection-based

Page 7

Addresses and Sockets

- Structure to hold address information
- Functions pass address from user to OS
 - bind()
 - connect()
 - sendto()
- Functions pass address from OS to user
 - accept()
 - recvfrom()

Page 9

Socket Address Structure

```

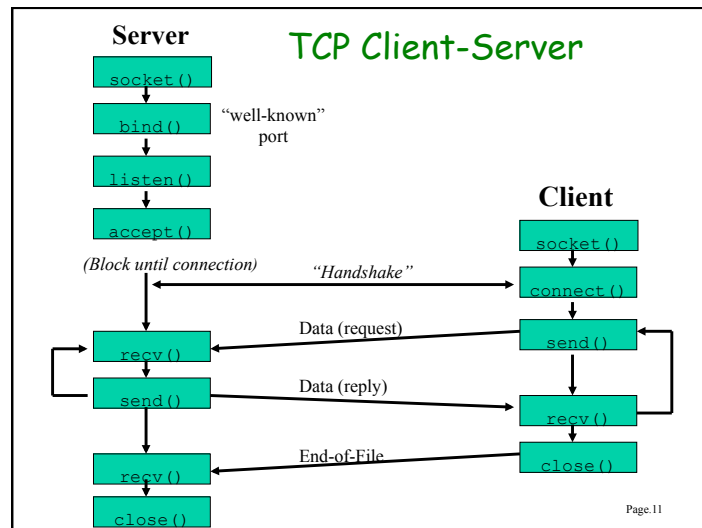
struct in_addr {
    in_addr_t s_addr;      /* 32-bit IPv4 addresses */
};

struct sockaddr_in {
    sa_family_t sin_family; /* AF_INET */
    in_port_t sin_port;     /* TCP/UDP Port num */
    struct in_addr sin_addr; /* IPv4 address (above) */
}

uint32_t htonl(uint32_t hostlong);
uint16_t htons(uint16_t hostshort);
uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);

```

Page 10



socket ()

`int socket(int family, int type, int protocol);`
 Create a socket, giving access to transport layer service.

- *family* is one of
 - `AF_INET` (IPv4), `AF_INET6` (IPv6), `AF_LOCAL` (local Unix),
 - `AF_ROUTE` (access to routing tables), `AF_KEY` (new, for encryption)
- *type* is one of
 - `SOCK_STREAM` (TCP), `SOCK_DGRAM` (UDP)
 - `SOCK_RAW`
- *protocol* is 0 (used for some raw socket options)
- upon success returns **socket descriptor**
 - Integer, like file descriptor
 - Return **-1** if failure

Page 12

bind()

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

Assign a local protocol address ("name") to a socket.

- *sockfd* is socket descriptor from `socket ()`
- *myaddr* is a pointer to address struct with:
 - port number and IP address (`INADDR_ANY`)
 - If port is 0, then host will pick ephemeral port
- *addrlen* is length of structure
- returns 0 if ok, -1 on error
 - `errno=EADDRINUSE` ("Address already in use")

Page 13

listen()

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

Change socket state for TCP server.

- *sockfd* is socket descriptor from `socket ()`
- *backlog* is maximum number of *incomplete* connections
 - historically **5**
 - rarely above **15** on an even moderate Web server!

Page 14

accept()

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

Return next completed connection.

- *sockfd* is socket descriptor from `socket()`
- *cliaddr* and *addrlen* return protocol address from client
- returns brand new descriptor, created by OS

Page 15

connect()

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

Connect to server.

- *sockfd* is socket descriptor from `socket()`
- *servaddr* is a pointer to a structure with:
 - port number and IP address
 - must be specified (unlike `bind()`)
- *addrlen* is length of structure
- client doesn't need `bind()`
 - OS will pick *ephemeral port*
- returns 0 if successful, -1 on error

Page 16

Question

- Function `connect()` can be used both for TCP and UDP communication.
- TCP communication must use `connect()`.
- Function `connect()` is optional for UDP communication.
- What's the benefits of using `connect()` for UDP communication?

Page 17

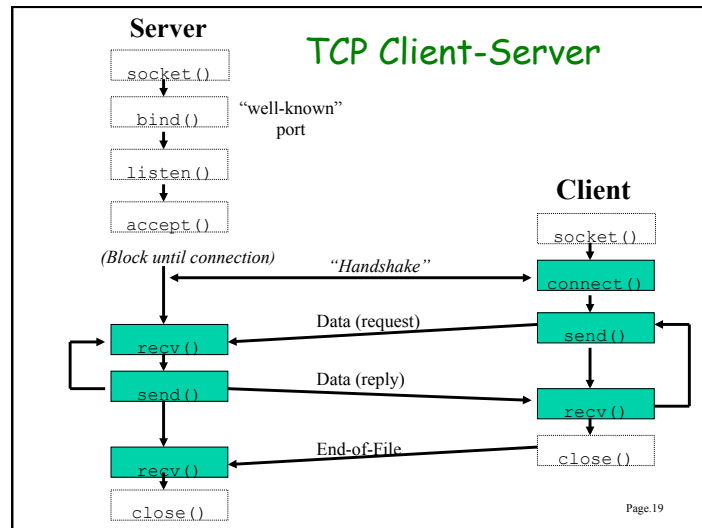
close()

```
int close(int sockfd);
```

Close socket for use.

- *sockfd* is socket descriptor from `socket()`
- closes socket for reading/writing
 - returns -1 if error
- By default, `close()` is not blocking.
 - socket option *SO_LINGER*
 - + block until data sent
 - + or discard any remaining data

Page 18



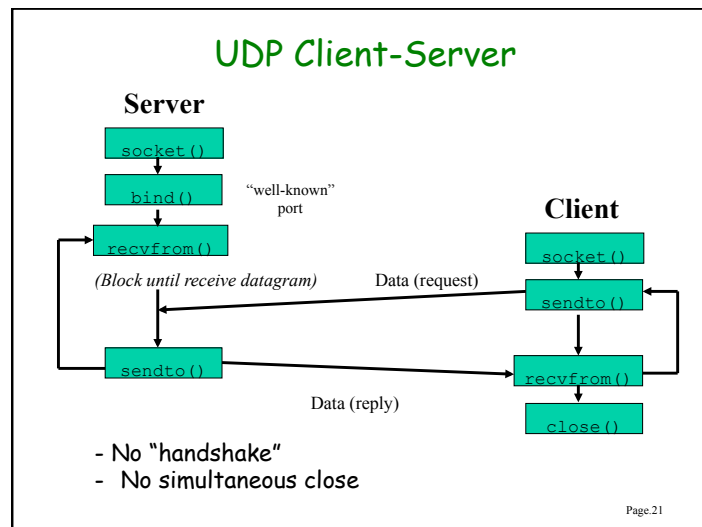
Sending and Receiving

```

int recv(int sockfd, void *buff, size_t
mbytes, int flags);
int send(int sockfd, void *buff, size_t
mbytes, int flags);
  
```

- Same as `read()` and `write()` but for *flags*
 - MSG_DONTWAIT
 - MSG_OOB
 - MSG_DONTROUTE

Page 20



Sending and Receiving

```

int recvfrom(int sockfd, void *buff, size_t mbytes,
int flags, struct sockaddr *from, socklen_t
*addrlen);
int sendto(int sockfd, void *buff, size_t mbytes, int
flags, const struct sockaddr *to, socklen_t
addrlen);
  
```

- Same as `recv()` and `send()` but for *addr*
 - `recvfrom` fills in address of where packet came from
 - `sendto` requires address of where sending packet to

Page 22

Concurrent Servers

Text segment

```
sock = socket()
/* setup socket */
while (1) {
    newsock = accept(sock)
    fork()
    if child
        read(newsock)
        until exit
}
```

Parent

```
int sock;
int newsock;
```

Child

```
int sock;
int newsock;
```

- Close sock in child