

Sockets

Socket

- Unrelated to process communication
- Bidirectional
- Client/Server
- Returns file descriptor

Socket Address

- Interprocess communication
- Socket pair
- TCP socket pair is a 4-tuple:
 - IP address and port number for each socket

Socket Address Data Types

- Generic data type:

```
struct sockaddr {           // generic socket address
    sa_family_t sa_family; // address family
    char sa_data[];         // endpoint address in family
}
```

- Internet (TCP, UDP): `sockaddr_in`
- Unix: `sockaddr_un`

struct sockaddr_in

```
struct sockaddr_in {
    short            sin_family; // e.g. AF_INET
    unsigned short   sin_port;   // e.g. htons(3490)
    struct in_addr   sin_addr;   // see struct in_addr, below
    char            sin_zero[8]; // zero this if you want to
};

struct in_addr {
    unsigned long s_addr; // load with inet_aton()
}
```

Socket Address Structures as Arguments

- Passed by reference
- Function definitions: generic socket structure
- Function calls: cast to specific `sockaddr` structure
- How is this done?

Endianness

- Read left-to-right or right-to-left?
- Big-endian: most significant byte on the left
- Little-endian: most significant byte on the right
- Example: $91329 = 0x164c1$
 - 00 01 64 C1
 - C1 64 01 00

Network Byte Order

- Big-endian
- Functions:
 - `htonl()`: host to network long
 - `htons()`: host to network short
 - `ntohl()`: network to host long
 - `ntohs()`: network to host short

Socket Connection Using TCP

- Passive open
- Active open
- Passive close - receive FIN packet
- Active close - close socket

Client/Server Actions

■ Server

- Create a socket (`socket()`)
- Assign an address to the socket (`bind()`)
- Make socket a passive socket and listen (`listen()`)
- Accept incoming connections

■ Client

- Create a socket (`socket()`)
- Connect socket (`connect()`)

Socket System Call

- `int socket(int domain, int type, int protocol)`
 - domain: address family
 - type: `SOCK_STREAM` or `SOCK_DGRAM`
 - protocol: typically 0 (determined by system)
- Return value:
 - Socket descriptor on success
 - -1 on failure

Assign Address to Socket

- `int bind(int s, const struct sockaddr *name, int addrlen);`
 - `s`: socket descriptor
 - `name`: address for socket (e.g. TCP = IP address and port number)
 - `addrlen`: length of address in `name`
- Return value:
 - 0 on success
 - -1 on failure

Listen for Incoming Connections

- `int listen(int s, int backlog)`
 - `s`: socket descriptor
 - `backlog`: number of connection requests in queue
- Return value:
 - 0 on success
 - -1 on failure

Accept Connection

- `int accept(int s, struct sockaddr *name, int *addrlen);`
 - `s`: socket descriptor
 - `name`: address of client (or NULL)
 - `addrlen`: maximum length and actual length of address in `name` (in/out argument)
- Return value:
 - Connected socket descriptor on success
 - -1 on failure

Address and Host System Calls

- `gethostname()`: retrieves hostname of current system
- `gethostbyname()`: obsolete
- `getaddrinfo()`
- `inet_addr()`
- `inet_aton()`
- `inet_ntoa()`

struct addrinfo

```
struct addrinfo {
    int                ai_flags;
    int                ai_family;    // AF_INET
    int                ai_socktype; // SOCK_STREAM/SOCK_DGRAM
    int                ai_protocol; // 0 - address for any protocol
    socklen_t          ai_addrlen;
    struct sockaddr *ai_addr;
    char               *ai_canonname;
    struct addrinfo *ai_next;
};
```


Client Actions

- Create socket
- Connect socket to server socket
- `int connect(int s, struct sockaddr *name, int addrlen);`
 - `s`: socket descriptor
 - `name`: server address (`sockaddr_in`)
 - `addrlen`: length of address in `name`
- Return value:
 - 0 on success
 - -1 on failure

Transfer Data

- `read()`
- `write()`
- `recv()`
- `send()`

Buffering

- TCP and UDP buffer data
- Successful call to `write()` or `send()`
 - where is the data written?
- Receive buffer
 - `read()` or `recv()`
- What happens on program exit or crash?

Connectionless vs. Connection-Oriented

- Connection

- sever must listen for connections
- client just connects to server
- SOCK_STREAM
- TCP protocol

- Connectionless

- client must create a socket and bind its local address to that socket
- SOCK_DGRAM
- UDP protocol

UDP Sockets

- How is socket programming over UDP different from socket programming over TCP?
- From the program perspective, how is UDP socket programming different from TCP?
- What should the client do if the packet does not reach the server?

Transferring Data over UDP Sockets

- `int sendto(int sockfd, const char *buf, int len, int flags, struct sockaddr *dest_addr, int addrlen)`
- `int recvfrom(int sockfd, char *buf, int len, int flags, struct sockaddr *src_addr, int addrlen)`
- Less common:
 - `sendmsg()`
 - `recvmsg()`

UDP Sockets

■ Server

- `socket()`
- `bind()`
- `recvfrom()` / `sendto()`
- `close()`

■ Client

- `socket()`
- `sendto()` / `recvfrom()`
- `close()`