TCP Sockets Outline

- Socket Address Structure
- TCP and UDP call Sequences
- Socket
  - Connect
  - Bind
  - Listen
- Accept
- Close
- Example: TCP Echo Server and Client

[old]
IPv4 Socket Address Structure

The Internet socket address structure is named `sockaddr_in` and is defined by including `<netinet/in.h>` header.

```c
struct in_addr {
    in_addr_t s_addr /* 32-bit IP address */
};

struct sockaddr_in {
    uint8_t sin_len; /* length of structure (16) */
    sa_family_t sin_family; /* AF_INET */
    in_port_t sin_port; /* 16-bit TCP or UDP port number */
    struct in_addr sin_addr; /* 32-bit IPv4 address */
    char sin_zero[8]; /* unused */
};
```
The Socket Interface

Application 1

socket interface

user

kernel

Socket

Underlying communication Protocols

Communications network

Application 2

socket interface

user

kernel

Socket

Underlying communication Protocols
TCP Socket Calls

Server
- socket()
- bind()
- listen()
- accept()

blocks until server receives a connect request from client

Client
- socket()
- connect()
- write()
- read()
- close()
UDP Socket Calls

Server

socket()

bind()

recvfrom()

blocks until server receives data from client

sendto()

close()

Client

socket()

bind()

sendto()

recvfrom()

close()

Not needed
System Calls for Elementary TCP Sockets

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

socket Function

int socket ( int family, int type, int protocol );

family: specifies the protocol family   {AF_INET for TCP/IP}

- SOCK_STREAM  stream socket    TCP
- SOCK_DGRAM    datagram socket   UDP
- SOCK_RAW      raw socket

protocol: set to IPPROTO_TCP for TCP

returns on success:    socket descriptor  {a small nonnegative integer}
                      on error:        -1

Example:

if (( sd = socket (AF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0) err_sys ("socket call error");
```
Connect Function

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

- **sockfd**: a socket descriptor returned by the socket function
- ***servaddr**: a pointer to a socket address structure
- **addrlen**: the size of the socket address structure

The socket address structure must contain the IP address and the port number for the connection wanted.

In TCP `connect` initiates a three-way handshake. `connect` returns only when the connection is established or when an error occurs.

returns on success: 0
on error: -1

**Example:**

```c
if (connect (sd, (struct sockaddr *) &servaddr, sizeof (servaddr)) != 0)
    err_sys("connect call error");
```
TCP Socket Calls

**Server**
- `socket()`
- `bind()`
- `listen()`
- `accept()`
  - blocks until server receives a connect request from client

**Client**
- `socket()`
- `connect()`
- `write()`
- `read()`
- `close()`

**Connect Negotiation**
- Data flow between server and client through `write()` and `read()` operations.
Bind Function

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

**bind** assigns a local protocol address to a socket.

- **protocol address:** a 32 bit IPv4 address and a 16 bit TCP or UDP port number.
- **sockfd:** a socket descriptor returned by the socket function.
- ***myaddr:** a pointer to a protocol-specific address.
- **addrlen:** the size of the socket address structure.

*Servers bind* their “well-known port” when they start.

**returns**
- on success: 0
- on error: -1

**Example:**

```
if (bind (sd, (struct sockaddr *) &servaddr, sizeof (servaddr)) != 0)
    errsys ("bind call error");
```
Listen Function

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

`listen` is called only by a TCP server and performs two actions:

1. Converts an unconnected socket (`sockfd`) into a passive socket.
2. Specifies the maximum number of connections (`backlog`) that the kernel should queue for this socket.

`listen` is normally called before the `accept` function.

returns on success: 0
on error: -1

Example:
```
if (listen (sd, 2) != 0) errsys ("listen call error");
```
Accept Function

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

**accept** is called by the TCP server to return the next completed connection from the front of the completed connection queue.

**sockfd**: This is the same socket descriptor as in **listen** call.

**cliaddr**: used to return the protocol address of the connected peer process (i.e., the client process).

**addrlen**: {this is a value-result argument}

  *before the accept call*: We set the integer value pointed to by **addrlen** to the size of the socket address structure pointed to by **cliaddr**.

  *on return from the accept call*: This integer value contains the actual number of bytes stored in the socket address structure.

**returns**  
**on success**: a new socket descriptor  
**on error**: -1
int accept (int sockfd, struct sockaddr *cliaddr, socklen_t *addrlen);

For `accept` the first argument `sockfd` is the **listening socket** and the returned value is the **connected socket**.

The server will have one connected socket for each client connection accepted.

When the server is finished with a client, the connected socket **must** be closed.

Example:

```c
sfd = accept (sd, NULL, NULL);
if (sfd == -1) err_sys ("accept error");
```
int close (int sockfd);

close marks the socket as closed and returns to the process immediately.

sockfd: This socket descriptor is no longer useable.

Note – TCP will try to send any data already queued to the other end before the normal connection termination sequence.

returns on success: 0
on error: -1

Example:

close (sfd);
TCP Echo Server

#include <stdio.h>        /* for printf() and fprintf() */
#include <sys/socket.h>   /* for socket(), bind(), and connect() */
#include <arpa/inet.h>    /* for sockaddr_in and inet_ntoa() */
#include <stdlib.h>       /* for atoi() and exit() */
#include <string.h>       /* for memset() */
#include <unistd.h>       /* for close() */

#define MAXPENDING 5       /* Maximum outstanding connection requests */

void DieWithError(char *errorMessage);   /* Error handling function */
void HandleTCPClient(int clntSocket);   /* TCP client handling function */
int main(int argc, char *argv[]) {
    int servSock;  /* Socket descriptor for server */
    int clntSock;  /* Socket descriptor for client */
    struct sockaddr_in echoServAddr; /* Local address */
    struct sockaddr_in echoClntAddr; /* Client address */
    unsigned short echoServPort; /* Server port */
    unsigned int clntLen; /* Length of client address data structure */

    if (argc != 2) /* Test for correct number of arguments */ {
        fprintf(stderr, "Usage: %s <Server Port>\n", argv[0]);
        exit(1);
    }
    echoServPort = atoi(argv[1]); /* First arg: local port */
    /* Create socket for incoming connections */
    if ((servSock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0) {
        DieWithError("socket() failed");
    }
TCP Echo Server (cont)

/* Construct local address structure */
memset(&echoServAddr, 0, sizeof(echoServAddr)); /* Zero out structure */

echoServAddr.sin_family = AF_INET; /* Internet address family */

if (bind(servSock, (struct sockaddr *) &echoServAddr, sizeof(echoServAddr)) < 0)
    DieWithError("bind() failed");

/* Mark the socket so it will listen for incoming connections */
if (listen(servSock, MAXPENDING) < 0)
    DieWithError("listen() failed");
for (;;) /* Run forever */
{
    /* Set the size of the in-out parameter */
    clntLen = sizeof(echoClntAddr); /* Wait for a client to connect */
    if ((clntSock = accept (servSock, (struct sockaddr *) &echoClntAddr, &clntLen)) < 0)
        DieWithError("accept() failed");

    /* clntSock is connected to a client! */
    printf("Handling client %s\n", inet_ntoa(echoClntAddr.sin_addr));
    HandleTCPClient(clntSock);
}
/* NOT REACHED */
#include <stdio.h> /* for printf() and fprintf() */
#include <sys/socket.h> /* for socket(), connect(), send(), and recv() */
#include <arpa/inet.h> /* for sockaddr_in and inet_addr() */
#include <stdlib.h> /* for atoi() and exit() */
#include <string.h> /* for memset() */
#include <unistd.h> /* for close() */
#define RCVBUFSIZE 32 /* Size of receive buffer */

void DieWithError(char *errorMessage); /* Error handling function */
int main(int argc, char *argv[]) {

    int sock;                /* Socket descriptor */
    struct sockaddr_in echoServAddr; /* Echo server address */
    unsigned short echoServPort;    /* Echo server port */
    char *servIP;                /* Server IP address (dotted quad) */
    char *echoString;            /* String to send to echo server */
    char echoBuffer[RCVBUFSIZE];  /* Buffer for echo string */
    unsigned int echoStringLen;  /* Length of string to echo */
    int bytesRcvd, totalBytesRcvd; /* Bytes read in single recv() and total bytes read */

    if ((argc < 3) || (argc > 4)) /* Test for correct number of arguments */ {
        fprintf(stderr, "Usage: %s <Server IP> <Echo Word> [<Echo Port>]\n", argv[0]);
        exit(1)
    }
}
servIP = argv[1]; /* First arg: server IP address (dotted quad) */

echoString = argv[2]; /* Second arg: string to echo */

if (argc == 4)
    echoServPort = atoi(argv[3]); /* Use given port, if any */
else
    echoServPort = 7; /* 7 is the well-known port for the echo service */

/* Create a reliable, stream socket using TCP */
if ((sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0)
    DieWithError("socket() failed");

/* Construct the server address structure */
memset(&echoServAddr, 0, sizeof(echoServAddr)); /* Zero out structure */

echoServAddr.sin_family = AF_INET; /* Internet address family */

echoServAddr.sin_addr.s_addr = inet_addr(servIP); /* Server IP address */

echoServAddr.sin_port = htons(echoServPort); /* Server port */
/* Establish the connection to the echo server */
if (connect (sock, (struct sockaddr *) &echoServAddr, sizeof(echoServAddr)) < 0)
    DieWithError("connect() failed");

echoStringLen = strlen(echoString); /* Determine input length */

/* Send the string to the server */
if (send (sock, echoString, echoStringLen, 0) != echoStringLen)
    DieWithError("send() sent a different number of bytes than expected");

/* Receive the same string back from the server */
totalBytesRcvd = 0; /* Count of total bytes received */
printf("Received: "); /* Setup to print the echoed string */
while (totalBytesRcvd < echoStringLen)
{
    /* Receive up to the buffer size (minus 1 to leave space for
       a null terminator) bytes from the sender */
    if ((bytesRcvd = recv (sock, echoBuffer, RCVBUFSIZE - 1, 0)) <= 0)
        DieWithError("recv() failed or connection closed prematurely");
    totalBytesRcvd += bytesRcvd;  /* Keep tally of total bytes */
    echoBuffer[bytesRcvd] = '\0';  /* Terminate the string! */
    printf("%s", echoBuffer);    /* Print the echo buffer */
}
printf("\n");      /* Print a final linefeed */
close (sock);
exit(0);
TCP Sockets Summary

- Sockets Address Structure
- TCP and UDP Call Sequences
- Socket
- Connect
- Bind
- Listen
- Accept
- Close
- TCP Echo Server and Client [old]