Sockets 2: Basic Syscalls

1. Introduction and Addresses
2. **Basic System Calls**
   - **basic syscalls**
   - **server and client syscall sequences**
   - **socket()**
   - **bind()**
   - **listen()**
   - **accept()**
   - **connect()**
3. Socket I/O Syscalls
4. IP Addresses and Hostnames
5. Endianess, Options, Servers

Basic Network Programming Calls

There are a relatively large number of calls related to sockets and network programming.

The most important are:

- **socket** – create a socket
- **bind** – associate (bind) an *address* (name) to a socket
- **listen** – allow connection requests on a socket
- **accept** – accept a connection on a (listening) socket
- **connect** – initiate a connection via a socket
- **write/send/sendto/sendmsg** – send data through the socket
- **read/recv/recvfrom/recvmsg** – retrieve data from the socket

Client-Server System Calls

The standard calls for a stream-based *server* are:

1. `socket()`
2. `bind()`
3. `listen()`
4. `accept()`
5. `read()'s/write()'s`

The standard calls for a stream-based *client* are:

1. `socket()`
2. `connect()`
3. `write()'s/read()'s`

Client-Server System Calls (contd.)

The standard calls for a datagram-based *server* are:

1. `socket()`
2. `bind()`
3. `recvfrom()'s/sendto()'s`

The standard calls for a datagram-based *client* are:

1. `socket()`
2. `sendto()'s/recvfrom()'s`

socket

The `socket` call creates a socket:

```
int socket(int domain, int type, int protocol)
```

- `domain` specifies the *communication domain*, which determines the **address family** (address format) and the types of sockets/connections that are available
- `type` defines the *communication semantics*
- `protocol` is usually 0 (zero), meaning use the default protocol for the specified socket type
- returns a *file descriptor* **handle** for the new socket

socket (contd.)

`domain` will be one of:

- `AF_INET` – IPv4;
- `AF_INET6` – IPv6
- `AF_UNIX/AF_LOCAL` – UNIX sockets
- `AF_PACKET` – raw packets

`type` will be one of

- `SOCK_STREAM` – stream (e.g., TCP)
- `SOCK_DGRAM` – datagram (e.g., UDP)
- `SOCK_SEQPACKET` – stream-message (e.g., SCTP)
- `SOCK_RDM` – reliable datagram (e.g., DCCP)
- `SOCK_RAW` – raw packets

socket (contd.)

Standard call to setup an IPv4 TCP socket:

```
int sockfd = socket(AF_INET, SOCK_STREAM, 0));
```

bind

`bind` associate an address with a (open) socket:

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

- `sockfd` is the file descriptor for the socket
- `addr` is the address to associate
- `addrlen` is the length of the particular address struct
- returns 0 on success else -1

`bind()` is always used when initializing a server, but may also be used with clients.

If `bind()` is not called, the kernel assigns an address (e.g., sets the port number).

bind (contd.)

Standard calls to setup IPv4 TCP socket address for a *server*:

```
struct sockaddr_in servaddr;

memset(&servaddr, 0, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(1234);
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);

bind(sockfd, (struct sockaddr *) &servaddr, sizeof(servaddr));
```

One could set up the address directly in the struct declaration:

```
struct sockaddr_in servaddr =
    {AF_INET, htons(1234), htonl(INADDR_ANY)};
```

listen

`listen` tells the kernel to accept connections to a socket:
`int listen(int sockfd, int backlog)`

- `sockfd` is the file descriptor for the socket
- the socket type must be `SOCK_STREAM` or `SOCK_SEQPACKET`
- `backlog` is a suggestion about the maximum size for the **queue** of *completed by not yet accepted connections*
- returns 0 on success else -1

One of the standard calls to set up a *server* for a *connection-oriented protocol*.

The standard call is:

```
listen(sfd, 10);
```

accept

`accept` completes a connection with a *listening* socket:

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

- `sockfd` is the file descriptor for the listening socket
- `addr` is the address of the connecting *client*
- `addrlen` is the length of the particular address struct
- returns a file descriptor for a socket for the connection, which will be different from the listening socket FD `sockfd`
- if client address is not required, `addr` and `addrlen` should be set to `NULL`

accept (contd.)

The standard call when not interested in the caller's address is:

```
accept(sockfd, (struct sockaddr *) NULL, NULL);
```

If one is interested in getting information about the caller:

```
struct sockaddr_in clientaddr;
socklen_t clientlen;
accept(sockfd, (struct sockaddr *) &clientaddr, &clientlen);
```

connect

`connect()` is used in a *client* program, to establish a connection to the server:

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

- `sockfd` is the file descriptor for the socket
- `serv_addr` is the address of the target server socket
- `addrlen` is the length of the particular address struct
- returns 0 on success else -1

`connect()` *must* be used with TCP connections, but can also be used to create a "connected UDP socket." (though this is less common).

connect (contd.)

The standard setup and call for an IPv4 TCP socket:
(assuming a socket has already been created)

```
struct sockaddr_in servaddr;

//Set up server address struct:
memset(&servaddr, 0, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(1234);
inet_aton("131.230.133.20",&servaddr.sin_addr);

connect(sockfd, (struct sockaddr *)&servaddr, sizeof(servaddr))
```

Example: Server (receives a message)

```
char msg[201];
int listen_fd, connect_fd, nread;
struct sockaddr_in servaddr;

//Create main server socket:
listen_fd = socket(AF_INET, SOCK_STREAM, 0);

//Set up server address struct and give socket address:
servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(1234);
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
bind(listen_fd, (struct sockaddr *) &servaddr, sizeof(servaddr));

//Start socket listening:
listen(listen_fd, 10);

//Main server loop:
while(1) {
  //Accept a new connection and get socket to use for client:
  connect_fd = accept(listen_fd, (struct sockaddr *) NULL, NULL);

  //Read client message and print it:
  nread = read(connect_fd,msg,200); msg[nread] = '\0';
  printf("Client message: %s\n",msg); }

  //Close connection to client:
  close(connect_fd); }
```

Example: Client (sends a message)

```
//Get server's IP address and message from arguments:
char *server_ip = argv[1];
char *msg = argv[2];

int sock_fd;
struct sockaddr_in servaddr;

//Create socket:
sock_fd = socket(AF_INET, SOCK_STREAM, 0);

//Set up server address struct:
servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(1234);
inet_aton(server_ip,&servaddr.sin_addr);

//Connect to server:
connect(sock_fd, (struct sockaddr *)&servaddr, sizeof(servaddr));

//Send message to server:
write(sock_fd,msg,strlen(msg));

//Close connection:
close(sock_fd);
```