Sockets Programming in C using TCP/IP

TA: Awad A Younis
Class: CS457
Fall 2014
**Computer Networks:**

- Consists of **Machines** Interconnected by **communication channels**

---

- **Machines** are Hosts and Routers
  - **Hosts** run applications
  - **Routers** forward *information* among communication channels

- **Communication channels** is a means of conveying sequences of bytes from one host to another (*Ethernet, dial-up, satellite, etc.*)
Packets:
• Sequences of **bytes** that are constructed and interpreted by programs

A packet contains
  - **Control information:**
    - Used by routers to figure out how to forward every packet.
    - e.g. packet destination
  - **User data**
Protocol:

- An agreement about the \textit{packets exchanged} by communicating programs and \textit{what they mean}.

- A protocol tells
  - how packets are structured
    - where the distention information is located in the packet
    - how big it is

- Protocols are designed to solve specific problems
  - TCP/IP is such collection of solutions (protocol suite or family):
    - IP, TCP, UDP, DNS, ARP, HTTP, and many more

- How can we access the services provided by TCP/IP suite?
  - Sockets API.
Addresses:

• Before one program can communicate with another program, it has to tell the network where to find the other program.

• In TCP/IP, it takes two pieces of information:
  - Internet Address, used by IP (e.g. Company’s main phone number)
  - Port Number, interpreted by TCP & UDP (extension number of an individual in the company)
Client and server

- **Server**: *passively* waits for and responds to clients
- **Client**: initiates the communication
  - must know the address and the port of the server

- **Socket()**: endpoint for communication
- **Bind()**: assign a unique number
- **Listen()**: wait for a caller
- **Connect()**: dial a number
- **Accept()**: receive a call
- **Send()** and **Receive()**: Talk
- **Close()**: Hang up
- **Server**
  1. Create a TCP socket using `socket()`
  2. Assign a port number to the socket with `bind()`
  3. Tell the system to allow connections to be made to that port using `listen()`
  4. Repeatedly do the following:
     - Call `accept()` to get a new socket for each client connection
     - Communicate with the client using `send()` and `recv()`
     - Close the client connection using `close()`

- **Client**
  1. Create a TCP socket using `socket()`
  2. Establish a connection to server using `connect()`
  3. Communicate using `send()` and `recv()`
  4. Close connection using `close()`
Why socket programming?

- To build network applications.
  - Firefox, google chrome, etc.
  - Apache Http server

What is a socket?

- It is an abstraction through which an application may send and receive data
- File is an analogy: read (receive) and write (send)

Types of sockets

- Stream sockets (TCP): reliable byte-stream service
- Datagram sockets (UDP): best effort datagram service
• **What is a socket API?**
  
  – An interface between application and network
  
  – **Applications** access the services provided by **TCP** and **UDP** through the sockets API
Specifying Addresses

- Applications need to be able to specify Internet address and Port number. How?
- Use **Address Structure**
  1. **Sockaddr**: generic data type
  2. **in_addr**: internet address
  3. **sockaddr_in**: another view of Sockaddr

```c
struct sockaddr_in{
    unsigned short sin_family; /* Internet protocol (AF_INET) */
    unsigned short sin_port; /* Address port (16 bits) */
    struct in_addr sin_addr; /* Internet address (32 bits) */
    char sin_zero[8]; /* Not used */
}
```

<table>
<thead>
<tr>
<th>sockaddr</th>
<th>sa_family</th>
<th>sa_data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Family</td>
<td>Blob</td>
<td>(14 bytes)</td>
</tr>
<tr>
<td>2 bytes</td>
<td>2 bytes</td>
<td>4 bytes</td>
</tr>
<tr>
<td>8 bytes</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sockaddr_in</th>
<th>Family</th>
<th>Port</th>
<th>Internet address</th>
<th>Unused</th>
</tr>
</thead>
<tbody>
<tr>
<td>sin_family</td>
<td>sin_port</td>
<td>sin_addr</td>
<td></td>
<td>sin_zero</td>
</tr>
</tbody>
</table>
Create a socket

int socket(int protocolFamily, int type, int protocol)

- **protocolFamily**: Always PF_INET for TCP/IP sockets
- **type**: Type of socket (SOCK_STREAM or SOCK_DGRAM)
- **protocol**: Socket protocol (IPPROTO_TCP or IPPROTO_UDP)

socket () returns the descriptor of the new socket if no error occurs and -1 otherwise.

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

int servSock;
if ((servSock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0)
```

Bind to a socket

```
int bind(int socket, struct sockaddr *localAddress, unsigned int addressLength)
```

- `socket`: Socket (returned by socket ())
- `localAddress`: Populated sockaddr structure describing local address
- `address Length`: Number of bytes in sockaddr structure--usually just size of (localAddress)

- bind() returns 0 if no error occurs and -1 otherwise.
- Example:

```c
struct sockaddr_in ServAddr;
ServAddr.sin_family = AF_INET; /* Internet address family
ServAddr.sin_addr.s_addr = htonl(INADDR_ANY); /* Any incoming interface
ServAddr.sin_port = htons(ServPort); /* Local port */
```

```c
if (bind(servSock, (struct sockaddr *) &ServAddr, sizeof(echoServAddr)) < 0)
```
Listen to incoming connections

int listen(int socket, int backlog)

- socket: Socket (returned by socket ())
- backlog: Maximum number of new connections (sockets) waiting

- listen() returns 0 if no error occurs and - 1 otherwise.
- Example:

#define MAXPENDING 5

if (listen(servSock, MAXPENDING) < 0)
Accept new connection

```c
int accept(int socket, struct sockaddr * clientAddress, int * addressLength )
```

- **socket**: Socket (listen() already called)
- **clientAddress**: Originating socket IP address and port
- **addressLength**: Length of sockaddr buffer (in), returned address (out)

• accept () returns the newly connected socket descriptor if no error occurs and -1 otherwise.

• Example:

```c
#define MAXPENDING 5

if ((clientSock=accept(servSock,(structsockaddr*)&ClntAddr,&clntLen))<0)
```
Constricting a Message

1. Encoding data: array vs struct
2. Byte ordering: htonl/htons vs ntohl/ntohs
3. Alignment and Padding: int/unsigned short and int/unsigned short
Some helpful resources:

   (Beej’s Guide to Network Programming Using Internet Sockets)

   (Book: TCP/IP Sockets in C Practical Guide for Programmers)


Thank You

Reference

• Pocket Guide to TCP/IP Socket, by Michael J. Donahoo and Kenneth L. Calvert

• Beej’s Guide to Network Programming Using Internet Sockets, by Brian "Beej" Hall. (http://www.cs.columbia.edu/~danr/courses/6761/Fall00/hw/pa1/6761-sockhelp.pdf)