Socket Programming (1/2)
Outline

1. Introduction to Network Programming
2. Network Architecture – Client/Server Model
3. TCP Socket Programming
4. UDP Socket Programming
5. IPv4/IPv6 Programming Migration
Introduction to Network Programming
Introduction to Network Programming

- 1-1. What is Computer Networks?
- 1-2. How to use Computer Network to exchange information?
- 1-3. How to build network applications?
1-1. What is Computer Networks?
What is Computer Networks?

- Communicating Hosts and Network Equipments
- Communicating Links
- Communicating Protocols
1-2. How to use Computer Network to exchange information?
How to use Computer Network to exchange information?

Network Applications

- Protocol Instance

Examples

- World Wide Web (HTTP)
- File Transferring (FTP)
- E-Mail (SMTP)
- VoIP (SIP, H323)
1-3. How to build network applications?
How to build network applications?

- SOCKETS - Network Programming Libraries (Interfaces)
  Examples:
- Linux – BSD Sockets
- Windows – WinSock
- JAVA, ...
2. Socket Overview
Socket Overview

Diagram showing the layers of the network protocol stack, including:
- Application Layer
- Transport Layer
- Network Layer
- Data Link Layer
- Physical Layer

Arrows indicate the flow of data across socket connections.
Network Programming Interfaces - Sockets

- Socket is an interface between application layer and transport layer in programming view.
- Network applications can use socket libraries to build a networked communicating channel and transmit information.
Socket Data Structure

- **Socket Addressing Information**
  - Internet Protocol Address – Network Layer
  - “140.123.101.100” / “140.123.105.25”

- **Transport Service Type – Transport Layer**
  - TCP / UDP / SCTP

- **Transport Service Port – Transport Layer**
  - 80 (http) / 21 (ftp) / 9093 (user defined)
Socket Data Structure

Two important Data Structures (in Ipv4)
- Internet Address Structure : struct addr_in
- Socket Address Structure : struct sockaddr_inr
Internet Address Structure

IPv4 Address Length: 32 bits (4 bytes)
“140.123.101.100” – Dotted-Decimal String

Network Byte Ordered IPv4 Internet Address Structure

```c
struct in_addr {
    in_addr_t     s_addr;
}
```
in_addr_t: uint32_t (Unsigned 32-bit integer)

Conversion of Network Byte Address and Address String

Byte Manipulation Functions

- int inet_aton()
- in_addr_t inet_addr()
- char *inet_ntoa()

Example:
```c
struct in_addr dest;
dest.sin_addr = inet_addr("140.123.101.114");
```
Socket Address Structure

Three Addressing Information
- Transport Layer: Port Number
- Transport Layer: Protocol Type
- Network Layer: IPv4 Address

Socket Address Structure
```
struct sockaddr_in {
    uint8_t   sin_len ;
    sa_family_t  sin_family ;
    in_port_t    sin_port ;
    struct in_addr sin_addr ;
    char        sin_zero[8] ;
}
```
- `sa_family_t`: unsigned short
- `in_port_t`: `uint16_t` (unsigned 16-bit integer)
Socket Address Structure

- Conversion of Network Byte Address and Address String
- Byte Manipulation Functions
  - `int inet_aton()`
  - `in_addr_t inet_addr()`
  - `char *inet_ntoa()`
  - `int inet_ppton()`
  - `const char *inet_ntop()`
- Byte Ordering Functions – Port Number
  - `uint16_t htons()` – Value in Network Byte Order
  - `uint32_t htonl()` – Value in Network Byte Order
  - `uint16_t ntohs()` – Value in Host Byte Order
  - `uint32_t ntohl()` – Value in Host Byte Order
Host Name and Address Translation

- Translation of Host Name and IP Address – DNS
  - struct hostent {}

- IPv4 Host and Address Translation Functions
  - gethostbyname()
    http://www.logix.cz/michal/devel/various/gethostbyname.c.xp
  - gethostbyaddr()
Socket Libraries

- Transport Layer Services
  - Transmission Control Protocol (TCP)
  - User Datagram Protocol (UDP)
  - Stream Control Transmission Protocol (SCTP)

- Socket Libraries
  - Socket Creation/Closing
  - Socket Data Transmission
  - Socket Options

```c
socket() close() bind() accept() connect()
sctp_bindx() sctp_connectx() sctp_getpaddr()
sctp_getladdr() sctp_freeladdr() sctp_freepaddr()
send() sendto() sendmsg() sctp_sendmsg()
recv() recvfrom() recvmsg() sctp_recvmsg()
setsockopt() getsockopt() sctp_opt_info()
```

SCOKET Libraries
Operation of Socket Libraries

1. Creating Socket Handle
   - Protocol Type
   - Protocol Family
2. At Server - Binding Port Number and specific IP address
3. At Client - Connecting Server’s Socket with specific Port Number and IP Address
4. Data Transmission
   - Sending and Receiving Data
5. Closing Socket Handle
### Socket Categories

sockfd = socket (domain, type, protocol)

<table>
<thead>
<tr>
<th>Service</th>
<th>Domain</th>
<th>Type</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
<td>AF_INET</td>
<td>SOCK_STREAM</td>
<td>IPPROTO_IP</td>
</tr>
<tr>
<td></td>
<td>PF_INET</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UDP</td>
<td>AF_INET</td>
<td>SOCK_DGRAM</td>
<td>IPPROTO_IP</td>
</tr>
<tr>
<td></td>
<td>PF_INET</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCTP</td>
<td>AF_INET</td>
<td>SOCK_SEQPACKET</td>
<td>IPPROTO_SCTP</td>
</tr>
<tr>
<td></td>
<td>PF_INET</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Categories of Socket Libraries

<table>
<thead>
<tr>
<th>Category</th>
<th>Functions</th>
</tr>
</thead>
</table>
| **Common** | socket(), close(), setsockopt(), getsockopt()  
gethostbyname(), htons(), htonl(), bind() |
| **TCP** | connect(), listen(), accept(), send(), recv() |
| **UDP** | sendto(), recvfrom() |
| **SCTP** | sctp_sendto(), sctp_bindx()  
sctp_sendmsg(), sctp_recvmsg() |
TCP Socket Programming
TCP Socket Programming

- 3-1. General Issues
- 3-2. Elementary TCP Socket Functions
- 3-3. TCP Client/Server Example
3-1. General Issues
General Issues

- Connection-Oriented
- Point-to-Point
- Reliable Data Transfer
- Flow Control
Connection-Oriented

- Before two communicating TCPs can exchange data, they must first agree upon the willingness to communicate.
- IP does not use “connections” - each datagram is sent independently.
Point-to-Point

- A TCP connection has two endpoints.
  - No broadcast/multicast
TCP guarantees that data will be delivered without loss, out of order, duplication or transmission errors.
TCP uses the ACK packets together with the sliding window mechanism.
3-2. Elementary TCP Socket Functions
Elementary of TCP Socket System Calls

**Server**
- (connection-oriented protocol)
  - `socket()`
  - `bind()`
  - `listen()`
  - `accept()` blocks until connection from client

**Client**
- (connection-oriented protocol)
  - `socket()`
  - `connect()` connection establish
  - `send()` data (request)
  - `recv()` data (reply)

**Initialization**
- `socket()`

**Connection**
- `bind()`
- `listen()`
- `accept()` blocks until connection from client

**Transmission**
- `recv()`
- `send()` process request
- `send()` data (reply)
- `recv()`
Elementary of TCP Socket System Calls

- **socket()**
  Create an endpoint for communication.

- **bind()**
  Assign a local protocol address to a socket.

- **listen()**
  Listen for connections on a socket.

- **accept()**
  Accept actual connection from some client process.

- **connect()**
  Initiate a connection on a socket.

- **send() & recv()**
  Send/Receive a message from a socket.
TCP Program Operation Flow

Server

- **Initialization**
  - `socket()`
  - `bind()`
  - `listen()`
  - `accept()` (blocks until connection from client)

Connection

- **Connection**
  - `recv()` (process request)
  - `send()` (data (reply))

Transmission

- Client
  - `socket()`
  - `connect()` (connection establish)
  - `send()` (data (request))
  - `recv()` (data (reply))
TCP Functions Discussion

- **Socket()**

  format: `SOCKET socket(  int domain,
                   int type,
                   int protocol );`

  argument:  
  - domain  PF_INET(AF_INET)
  - type    Socket type (SOCK_STREAM, SOCK_DGRAM)
  - protocol default 0

  return value:  
  - success  Socket number
  - fail     INVALID_SOCKET

- **example:**

  ```c
  tcp_sock = socket(PF_INET, SOCK_STREAM, 0);
  udp_sock = socket(PF_INET, SOCK_DGRAM, 0);
  ```
```

- **Bind()**

  format :  int bind( SOCKET s,
                     const struct sockaddr *name,
                     int namelen );

  argument :  s Socket number
              name Socket address

  return value :  success  0
                 fail      SOCKET_ERROR

- **example :**

  struct sockaddr_in sa;
  sa.sin_family = AF_INET;
  sa.sin_port = htons(5001);  /* host to network for short int */
  sa.sin_addr.s_addr = INADDR_ANY;
  bind(sock, (struct sockaddr *)&sa, sizeof(sa));
```
Listen()

format: int listen( SOCKET s,
               int backlog );

argument: s      Socket number
          backlog maxima listen connection number

return value: success 0
              fail    SOCKET_ERROR

eexample:
  listen(sock, 1)
Connect()

format : int connect(
    SOCKET s,
    const struct sockaddr *name,
    int namelen );

argument : s Socket number
ame Socket address
            namelen name length

return value : success 0
               fail SOCKET_ERROR

example :
struct sockaddr_in sa;
sa.sin_family = AF_INET;
sa.sin_port = htons(5001);  /* server's port number*/
sa.sin_addr.s_addr = htonl(serverip);
connect( sock, (struct sockaddr *)&sa, sizeof(sa);
Accept()

format: SOCKET accept( SOCKET s,
            struct sockaddr *addr,
            int *addrlen );

argument: s       Socket number
            addr    Socket address
            addrlen addr length

return value: success 0
              fail    SOCKET_ERROR

easy: example:
struct sockaddr_in sa;
int sa_len = sizeof(sa);
new_sock = accept(sock, (struct sockaddr far *)&sa, &sa_len)
**Send()**

format:  int send( SOCKET s,  
        const void *buf,  
        int len,  
        int flags );

argument:  s     Socket number  
            buf    the data buffer for transmission  
            len    the data buffer length  
            flags  the way function is called

return value:  success  the sent data length  
               fail      SOCKET_ERROR
recv()

format: int recv( SOCKET s,
                   void *buf,
                   int len,
                   int flags);

argument:
  s    Socket number
  buf  Receiving data buffer
  len  Buffer length
  flags The way function is called

return value:
  success Received data length (it is 0 if socket on the other side is closed)
  fail    SOCKET_ERROR
3-3. TCP Client/Server Example
TCP Client/Server Example
Server Initialization

```c
/* tcpserver.c */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h> /* INADDR_ANY */
#include <ctype.h> /* toupper() */

define PORT 10000 /* server port value */

int main()
{
    int accept_sock;
    int client_sock;
    struct sockaddr_in serv_addr;
    char ch;

    /* create INTERNET,TCP socket */
    accept_sock = socket(PF_INET, SOCK_STREAM, 0);
```
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = INADDR_ANY;
serv_addr.sin_port = htons(PORT); /* specific port */
/* bind protocol to socket */
if(bind(accept_sock, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0)
{
    perror("bind");
    exit(1);
}
listen(accept_sock,5);
Server Connection
31
32    for(;;)
33    {
34        /* accept one connection, will block here. */
35        client_sock = accept(accept_sock, 0, 0);

Server Transmission
36    for(;;)
37    {
38        if(recv(client_sock, &ch, sizeof ch, 0) <= 0)
39            break;
40        ch = toupper(ch);
41        send(client_sock, &ch, sizeof ch, 0);
42        if(ch == '\0') // end of string
43            break;
44    }
45    close(client_sock); // close a client socket
46    }
47    // close(accept_sock); // unreachable
48}
Client Initialization

01/* tcpclient.c */
02
03#include <stdio.h>
04#include <sys/types.h> /* basic system data types */
05#include <sys/socket.h> /* basic socket definitions */
06#include <netinet/in.h> /* sockaddr_in{} and other Internet defns */
07
08#define PORT 10000 /* server port value */
09
10int main()
11{
12  int           connect_sock;
13  char          input[100], output[100], *s = output;
14  struct sockaddr_in serv_addr;
15  int           n;
16
17  connect_sock = socket(PF_INET, SOCK_STREAM, 0);
19  serv_addr.sin_family = AF_INET;
20  serv_addr.sin_port = htons(PORT);
21  serv_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
Client Connection

22
23  if (connect(connect_sock, (struct sockaddr *) &serv_addr, sizeof serv_addr) <0)
24  {
25    perror("connect");
26    exit(1);
27  }
28

Client Transmission

29  printf("Input:");
30  scanf("%s", input);
31  send(connect_sock, input, strlen(input) + 1, 0);  // including ending '\0'
32  while((n = recv(connect_sock, s, sizeof *s, 0)) > 0) {
33    s += n;                                         // move pointer for recv()
34  }
35  printf("Output:%s\n", output);
36  close(connect_sock);
37}