

Network Programming: Introduction to sockets programming

Presented by: Tiberiu Stef-Praun

- Network programming concepts
 - Sockets internals
 - Examples
-

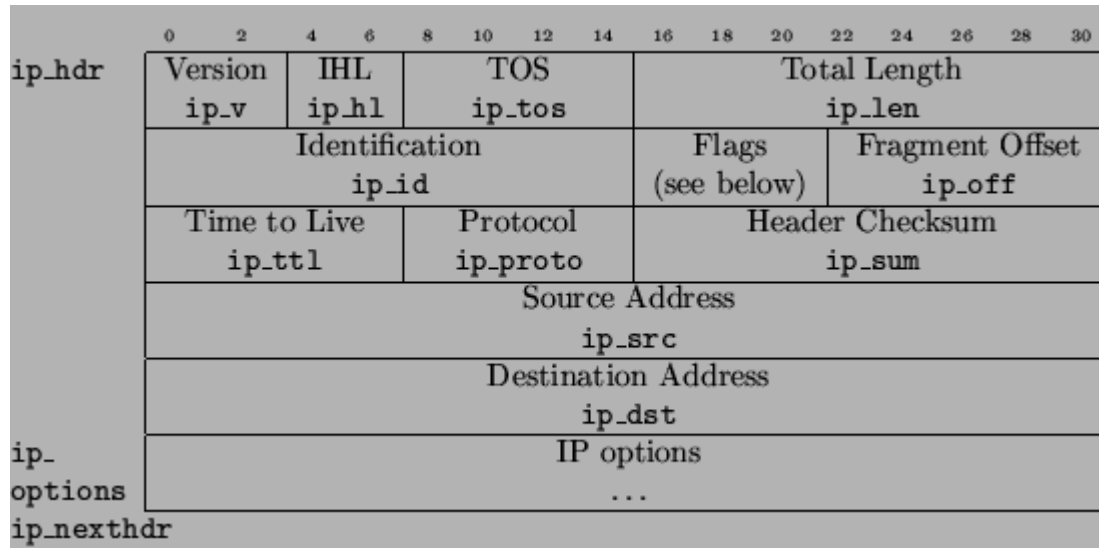
Sockets Programming

Programming Concepts

- In Unix env., all the devices are file abstractions:
 - functions: open, read, write, close
 - The API for the network device is the socket
 - comm. domain PF_INET
 - comm. type SOCK_STREAM (tcp), SOCK_DGRAM (udp)
 - protocol (default)
 - The data to be sent is encapsulated automatically according to the protocol chosen
 - At the API level there are some data structs to be filled out, defining the sending and receiving sockets
 - The network programming paradigm: client-server
 - Server side can choose different data notification mechanisms -sync: select()- and multiple client handling - concurrent, iterative.
-

Sockets Programming

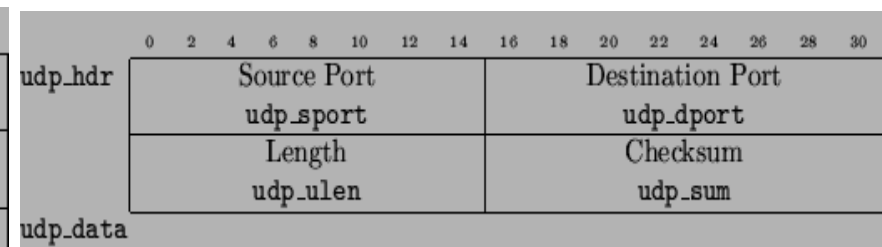
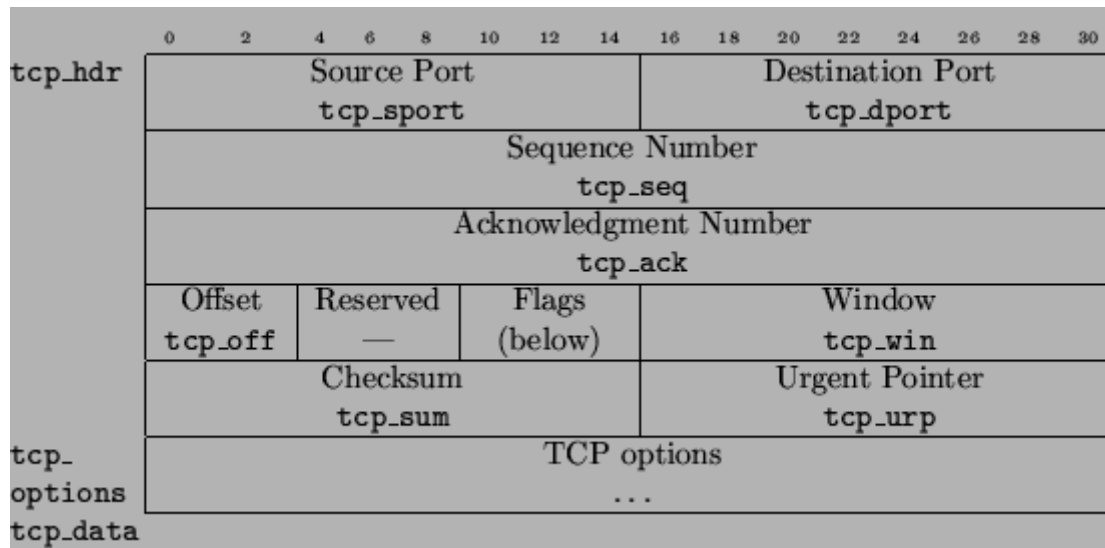
Encapsulation, protocol kernel data structures



IP hdr `#include <netinet/ip.h>`
`/usr/include/netinet/ip.h`

TCP hdr `#include <netinet/tcp.h>`
`/usr/include/netinet/tcp.h`

UDP hdr `#include <netinet/udp.h>`
`/usr/include/netinet/udp.h`



Source:
<http://linux-ip.net/gl/tcng/node36.html>

Sockets Programming

Kernel socket data structure

Socket data : /usr/include/netinet/in.h

```
#include <netinet/in.h>
#include <sys/socket.h>
struct sockaddr_in {
    short int          sin_family; // Address family: AF_INET
    unsigned short int sin_port;   // Port number
    struct in_addr     sin_addr;   // IP addr: unsigned long s_addr
    unsigned char      sin_zero[8]; // Padding with zero
};
```

Network/Host byte order

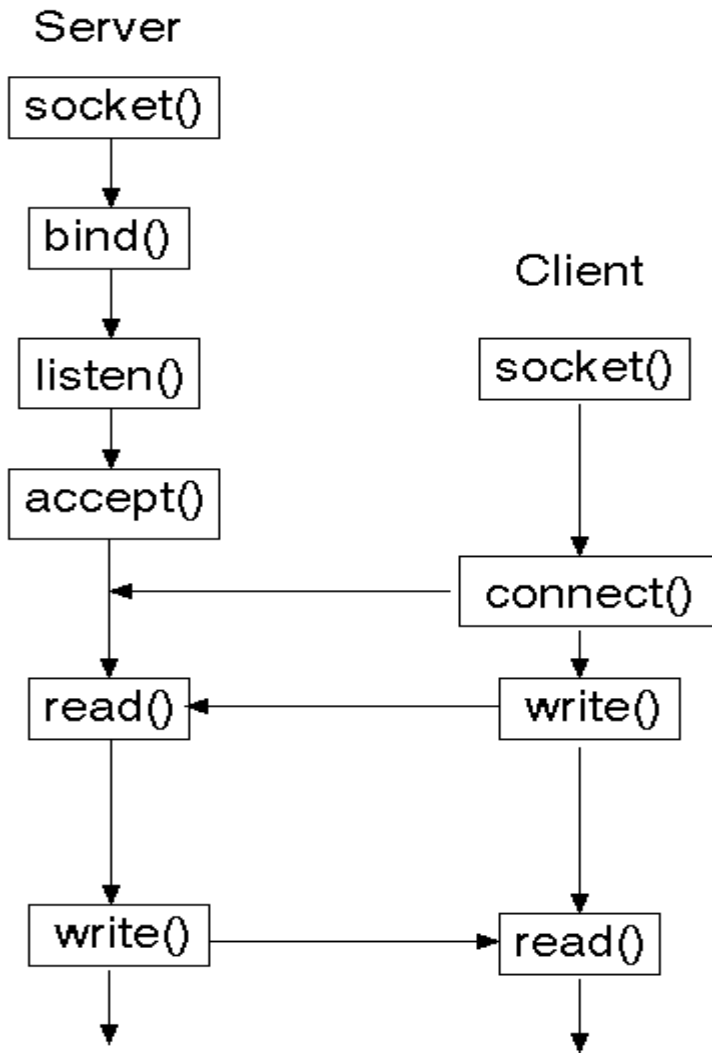
sin_addr and sin_port need to be in proper ordering
(they get encapsulated in the IP and TCP/UDP fields)

Need to change regular string representation to ulong:

- sin_addr.s_addr=inet_addr("128.10.3.62"); //net order
 - sin_port=htons(80); //web port
-

Sockets Programming

TCP communication



Server:

- `socket` : create comm. endpoint
- `bind`: associate the socket with an address
- `listen`: signal acceptance of incoming conn.
- `accept`: accept and incoming conn, create socket for client comm.

Client:

- initiate connection to the remote addr. specified as a param

Sockets Programming

TCP comm, socket setup code

```
int passiveTCPsocket(int port, int qlen)
{
    struct sockaddr_in sin;
    int sockd;

    bzero((char *)&sin, sizeof(sin));
    sin.sin_family = AF_INET;
    sin.sin_addr.s_addr = INADDR_ANY;
    sin.sin_port = htons((u_short)port);

    if ((sockd = socket(PF_INET, SOCK_STREAM, 0)) < 0)
    {
        printf("can't create socket: %s\n", sys_errlist
[errno]);
        exit(1);
    }

    if (bind(sockd, (struct sockaddr *)&sin, sizeof(sin)) <
0) {
        printf("can't bind to port %d\n", port);
        exit(1);
    }

    if (listen(sockd, qlen) < 0) {
        printf("can't listen::%s\n", sys_errlist[errno]);
        exit(1);
    }
    return sockd;
}
```

```
int connectTCP(char *host, int port)
{
    struct sockaddr_in sin;
    int sockd;

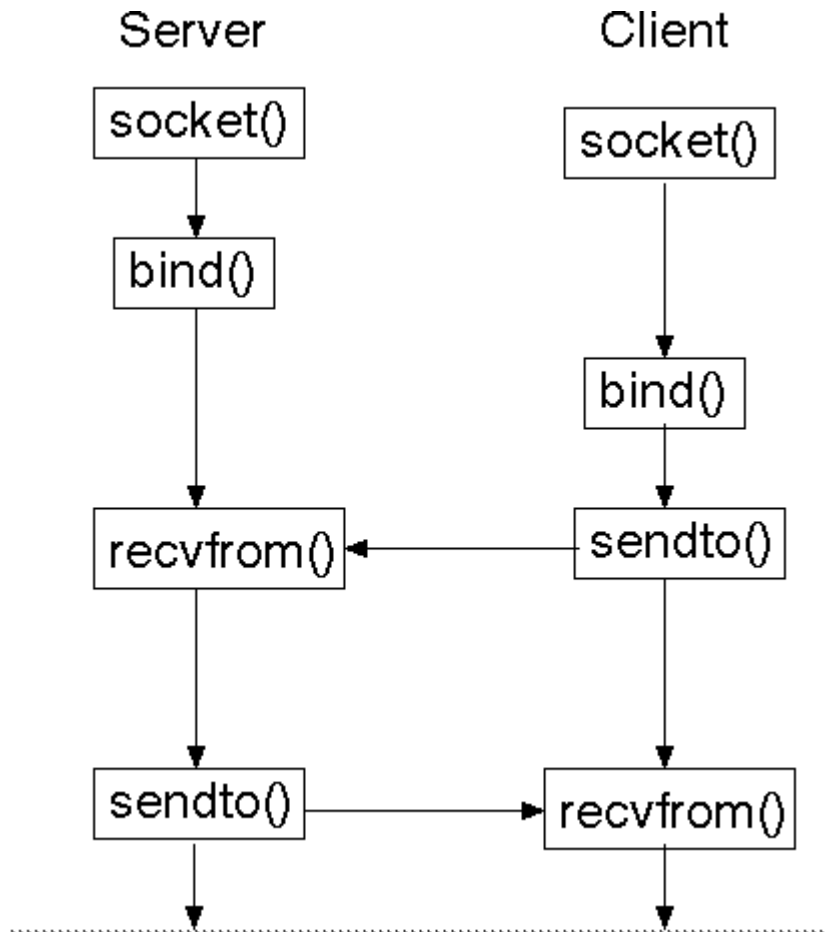
    bzero((char *)&sin, sizeof(sin));
    sin.sin_family = AF_INET;
    sin.sin_port = htons((u_short)port);
    sin.sin_addr.s_addr = inet_addr(host);

    if ((sockd = socket(PF_INET, SOCK_STREAM, 0)) <
0) {
        printf("can't create socket: %s\n", sys_errlist
[errno]);
        exit(1);
    }
    if (connect(sockd, (struct sockaddr *)&sin, sizeof
(sin)) < 0) {
        printf("can't connect to %s:%d: %s\n", host, port,
sys_errlist[errno]);
        exit(1);
    }
    return sockd;
}
```

Ref: [4]

Sockets Programming

UDP communication



Server:

- `socket` : create comm. endpoint
- `bind`: associate the socket with an address

Client:

- `bind`: associate with an address (because connectionless, per packet delivery)
- `(connect)` – special case of UDP, use with `read/write` when same dest. address

Sockets Programming

UDP sockets setup code

```
int passiveUDPsocket(int port)
{
    int sockd;          /* socket descriptor */
    struct sockaddr_in sin; /* server address */

    bzero((char *)&sin, sizeof(sin));
    sin.sin_family = AF_INET;
    sin.sin_addr.s_addr = INADDR_ANY;
    sin.sin_port = htons((u_short)port);

    if ((sockd = socket(PF_INET, SOCK_DGRAM, 0)) < 0)
    {
        printf("can't create socket: %s\n", sys_errlist
[errno]);
        exit(1);
    }
    if (bind(sockd, (struct sockaddr *)&sin, sizeof(sin)) <
0) {
        printf("can't bind to port %d: %s\n", port,
sys_errlist[errno]);
        exit(1);
    }

    return sockd;
}
```

```
int connectUDP(char *host, int port)
{
    struct sockaddr_in sin;
    int sockd;

    bzero((char *)&sin, sizeof(sin));
    sin.sin_family = AF_INET;
    sin.sin_port = htons((u_short)port);
    sin.sin_addr.s_addr = inet_addr(host);

    if ((sockd = socket(PF_INET, SOCK_DGRAM, 0)) <
0) {
        printf("can't create socket: %s\n", sys_errlist
[errno]);
        exit(1);
    }
    if (connect(sockd, (struct sockaddr *)&sin, sizeof
(sin)) < 0) {
        printf("can't connect to %s:%d\n", host, port);
        exit(1);
    }
    return sockd;
}
```

Ref: [4]

Sockets Programming

Server Code: the loop

TCP Server

```
msock = passiveTCPsocket(port,5);
while(1) {
    alen = sizeof(fsin);
    if ((sock = accept(msock, (struct sockaddr *)
&fsin, &alen)) < 0) {
        if (errno == EINTR)
            continue;
        printf("accept failed: %s\n", sys_errlist
[errno]);
        exit(1);
    }
    switch(fork()) {
        case 0: /* child */
            close(msock);
            PROCESS(sock);
        default:
            close(sock);
            break;
        case -1:
            printf("fork failed: %s\n", sys_errlist
[errno]);
            exit(1);
    }
}
```

UDP Server

```
char *data={"dddd"}

sockd = passiveUDPsocket(port);
while(1) {
    alen = sizeof (fsin);
    if (recvfrom(sockd, buf, sizeof(buf), 0,
(struct sockaddr *)&fsin, &alen)
< 0) {
        printf("error - recvfrom: %s\n",
sys_errlist[errno]);
        exit(1);
    }

    sendto(sockd, (data, sizeof(data), 0,
(struct sockaddr *)&fsin, sizeof(fsin));
}
```

Ref: [4]

Sockets Programming

Additional coding tips

- for concurrent servers, take care of zombie children (use signal)

```
int sigChldHandler()
{
    pid_t pid;
    int stat;
    while ((pid = waitpid(-1, &stat, WNOHANG)) > 0)
        printf("child %d terminated\n", pid);
}

signal(SIGCHLD, sigChldHandler);
```

- I/O multiplexing: for higher speed of the server, use select()/poll() - avoids blocking on the read(), will loop on FD_ISSET()
 - signal-driven receiving (SIGIO)
-

Sockets Programming

Useful resources

- [1] Google
 - [2] <http://pandonia.canberra.edu.au/ClientServer/socket/socket.html>
 - [3] R.W. Stevens: Unix Network Programming, vol.1
 - [4] <http://www.cs.purdue.edu/homes/rf/cs536/index.html>
 - unix man pages
-