UNIX Systems Programming

Networking with Sockets

(Curry, chp.14)

Dr. Kivanç Dinçer

CENG-332 Lectures

Spring 2000

Network Protocol Suites

Nearly every UNIX system is connected to some type of network

- **TCP/IP** (Transmission Control Protocol/Internet Protocol)
  - The de facto standard network protocol suite in use today is TCP/IP (Transmission Control Protocol/Internet Protocol)
  - developed by the Internet Engineering Task Force
  - funded by the US Defense Advanced Research Projects Agency (DARPA)
    - DARPA also provided principal funding for development of Berkeley UNIX –
    - BSD Unix was the first O/S that supports internetworking via TCP/IP- Berkeley networking paradigm: socket interface
  - used world-wide by hosts connected to the Internet
- **OSI** (Open Systems Interconnect)
  - standardized by International Standards Organization (ISO)
  - fairly popular in Europe, never caught in US.

Networking Concepts – Host Names

- Each host on the network has a unique host name.
  - On the Internet, a host name must be a *fully qualified domain name*.

Internet Domain Name System (DNS) allows the host name space to be subdivided into a number of logical areas, or domains.

  - easy to administer: each organization can administer its own name space.
    - In old days, entire host name space was controlled by the *Network Information Center*
    - 9M hosts on the Internet in 1996.
  - allow host names to be reused in different areas of the name space.
- On top-level each country has a two-letter domain
  - edu, mil, gov, com are also top-level domains.

Networking Concepts – Host Addresses

```
int gethostname(char* name, int len)
```

Host addresses, network addresses or Internet addresses:

- are usually written in dotted-quad notation
  - each byte of the address is converted to an unsigned decimal number and separated from the next by a period.
  - e.g., 0x7b2d4359 → 123.45.67.89
- consists of two parts:
  - a network number: used by the nw routing sw to decide how to deliver data from one network to another
    - subnetwork number: which part of the network
  - and a host number

Network Address Classes

- Class A: 1 byte network # and 3 bytes host #
- **Class B:** 2 byte network # and 2 bytes host #
- **Class C:** 3 byte network # and 1 bytes host #

/etc/hosts

- lists host name and address pairs
- usually used for local area addresses
- Network Information Service (Yellow Pages) provides a different interface to this file

Translation bw host names and addresses

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <netinet/in.h>
struct hostent* gethostent(void)
struct hostent* gethostbyname(const char *name)
struct hostent* gethostbyaddr(const char *addr,int len,int type)
int sethostent(int stayopen)      //open db and sets ptr to first entry
int endhostent(void)              //close db
```

```
struct hostent {
   char        *h_name;        <-- official host name
   char       **h_aliases;     <-- pointers to other names of host
   int          h_addrtype;
   int          h_length;
   char       **h_addr_list;   <-- list of addresses for that host
}
```

Networking Concepts –Services and Port Numbers

- On any given host on the network, a number of network services may be provided
  - remote login, file transfer, e-mail delivery, …
- Port number:
  - a small integer used to identify the service to which data is to be delivered
  - each service is assigned a port number
  - In order for two hosts to communicate using some service, they must agree on the port number to be used for that service.

Well-known ports:

- All standards Internet protocols use.
- FTP: 21, HTTP: 80, …
- stored in /etc/services

```
#include <netdb.h>
struct servent* getservent(void)            udp or udp
struct servent* getservbyname(const char *name, char *proto)
struct hostent* getservbyport(int port,char* proto)
int setservent(int stayopen)      //open db and sets ptr to first entry
int endservent(void)              //close db
```

```
struct servent {
   char        *s_name;              official name of service
   char       **s_aliases;           pointers to other names of service
   int          s_port;
   char        *s_proto;             protocol to use
}
```

Network Byte Order

When implementing integer storage on a computer, manufacturers have two choices:

- Big-endian:
  - most significant byte in the lowest memory address
  - SUN, Motorola
- Little-endian
  - Intel, DEC
- A 32-bit integer value as stored on a big-endian machine looks different than one stored on a little-endian machine.
  - To copy data from one type of host to the other, it is necessary to transform the data into the proper format.

- Network byte order (big endian)
  - insures that all traffic arriving at a host from the network will be in the same format
  - Berkeley networking paradigm specifies that each network program must perform these byte order conversions itself.
    - These are usually implemented as C preprocessor macros

```
#include <sys/types.h>
#include <netinet/in.h>
u_long  htonl(u_long  hostlong); //converts 32-bit value to nw byte order
u_short htons(u_short hostshort);
u_long  ntohl(u_long  netlong);
u_long  ntohs(u_short netshort);
```

  - Character strings do not need to be converted
  - Floating-point numbers are converted to integers or strings and then exchanged.

Creating a Socket

```
#include <sys/types.h>
#include <sys/socket.h>
int socket(int domain, int type, int protocol)
```

- domain specifies the domain, or address family, in which addresses should be interpreted.
  - It imposes certain restrictions on the length of addresses, and what they mean.
  - AF_INET domain is used for Internet addresses.
- protocol specifies the protocol number that should be used on the socket, usually the same as address family.
  - PF_INET protocol is used
  - if 0, the system will figure it out.

- type specifies the communications channels supported by sockets:
  - SOCK_STREAM (virtual circuit)
    - a bi-directional continuous byte stream that guarantees the reliable delivery of data in the order in which it was sent.The circuit remains intact until the conversation is complete. Implemented in the Internet domain using *Trasmission Control Protocol* (TCP).
  - SOCK_DGRAM
    - used to send distinct packets of info called datagrams. No guarantees on order or delivery. Implemented in the Internet domain using *Datagram Protocol* (UDP).

> returns a socket descriptor (a non-negative integer similar to a fd) or –1 and errno.

Server-Side Functions: bind, listen, accept

1- Naming a socket

- A server process must provide a socket with a name, so that client programs can access it.

```
int bind(int s, const struct sockaddr* name, int addlen)
```

Note: address must not be already in use!

```
struct sockaddr_in {
       short          sin_family;   <----  always AF_INET
       u_short        sin_port      <----  port number
       struct in_addr sin_addr;     <----  host address assoc'd w/port
};
```


Server-Side Functions: bind, listen, accept

2- Waiting for Connections

- Server must notify the O/S when it is ready to accept connections from clients on that socket.

```
int listen(int s, int backlog)
```

backlog specifies the # of connection requests that may be pending at any given time.

3- Accepting Connections

```
int accept(int s, struct sockaddr *name, int *addrlen)
```

- returns a new sd to communicate with the client.
  - old sd continue to accept additional connections.
- When connection is accepted, if name is not null, O/S stores the address of the client there and length in addrlen.

returns –1 and errno if fails.

Client-Side: Connecting to a Server

```
int connect(int s, struct sockaddr *name, int addrlen)
```

- connects the socket ref'd by s to the server at addr described by name.
- addrlen specifies the length of addr in name.

returns 0 or –1 and errno.

A client may use connect to connect to a datagram socket to the server as well.

- Not necessary
- But it does enable the client to send datagrams on the socket w/o having to specify destination addr for each datagram.

Client-Side: Transferring Data

1- simply use read and write.

2- use send and recv

```
int recv(int s, char *buf,       int len, int flags)
int send(int s, const char *buf, int len, int flags)
```

flag effect how the data is sent or received.

MSG_OOB: The data is sent as out-of-band data. This data "jumps over" any other data that has been sent and not received.

- e.g., to handle interrupt characters in a remote login session.

MSG_PEEK: If specified in a call to recv, the data is copied into buf as usual, but it is not consumed. Another call to recv will return the same data.

Transferring Data using Datagram-Based Sockets

- client does not (generally)call connect
  - There is no way for the O/S to determine automatically where data on these sockets is to be sent.
- server does not call listen or accept

The sender must tell the O/S each time where the data is to be delivered, and the receiver must ask where it came from.

```
int recvfrom(int s, char *buf, int len, int flags,
         struct sockaddr *from, int *fromlen)
int send(int s, const char *buf, int len, int flags,
         struct sockaddr *to,   int tolen)
```

returns # bytes actually received/sent or –1.

Destroying the Communications Channel

1- Close a socket with the `close` function

- if the socket refers to a stream-based socket, the `close` will block until all data has been transmitted.

2- Use `shutdown` function

```
int shutdown(int s, int how)
```

shuts down either or both sides of the communications channel

- how is 0: shut down for reading, all further reads from the socket return eof.
- how is 1: shut down for writing, all further writes to the socket will fail.
- how is 2: shut down both sides

See Examples 14-1, 14-2, 14-3.