Overview

• Design Principles
• Basic Paradigm
• Getting the needed information
  • Addressing info
    • Host
    • Service
    • Protocol Info
• Making a connection

Client Design Principles

• Open a connection at the latest possible time
• Close a connection at the earliest possible time
• Prepare for:
  • Network failures
  • Server process unavailable
• Allow server location to be specified:
  • Provide default information
  • Let user supply address and port at run time
  • Assist the user by converting all forms to the required form
Client Algorithm

- Determine the address scheme to be used
- Look up IP address for the service
- Look up the service number for the service
- Form an address structure of the appropriate type
- Allocate a socket
- Connect the client socket to the service
- Communicate with server (write & read)
- Close the connection

Client Calls

Simple
- socket
- connect
- write
- read
- close

More Complete
- Allocate sockaddr_in
- getservbyname or atoi
- gethostbyname or inet_addr
- Fill sockaddr_in
- getprotobyname
- socket
- connect
- write-read-close

Getting Ready

- From the header files
  - Deciding the address family
  - Deciding on the protocol family
  - Choosing the address structure
  - Specifying the specific protocol
- From the network
  - Finding the address of a machine
- From the /etc/services file
  - Finding a service
Addresses

- Creating a socket does not provide enough information to allow for communication
- An endpoint address must be specified
  - Multiple address families, each with different addressing schemes, have been created
  - Today, for all practical purposes, only the internet address family is used – AF_INET.
  - AF_INET is a symbolic constant.
  - Under AF_INET, an endpoint address consists of:
    - An IP address
    - A port number

Address Families

- #define AF_UNSPEC 0 /* unspecified */
- #define AF_UNIX 1 /* local to host(pipes) */
- #define AF_INET 2 /* internetwork:UDP,etc*/
- #define AF_PUP 4 /* pup proto: e.g. BGP */
- #define AF_CHAOS 5 /* mit CHAOS protocols */
- #define AF_NS 6 /* XEROX NS protocols */
- #define AF_NBS 7 /* nbs protocols */
- #define AF_CCITT 10 /* CCITT protocols */
- #define AF_SNA 11 /* IBM SNA */
- #define AF_DECnet 12 /* DECnet */
- #define AF_LAT 14 /* LAT */
- #define AF_SYSLINK 15 /* NSC Hyperchannel */
- #define AF_APPLETALK 16 /* Apple Talk */
- #define AF_802 18 /* IEEE 802.2 & ISO 8802 */

Address Structures

- The general structure for an address is:

```c
struct sockaddr {
    u_char sa_len;
    u_short sa_family; /* type of address */
    char sa_data[14]; /* value of address */
};
```

- The AF_INET family has an address structure defined as follows:

```c
struct sockaddr_in {
    u_char sin_len;
    u_short sin_family; /* address family */
    u_short sin_port; /* port number */
    u_long sin_addr; /* IP address */
    char sin_zero[8]; /* unused set zero */
};
```
Looking up a Machine Address

- The address of machines on the internet are binary addresses
- To make addresses easier to remember, two alternate forms are provided
  - Domain names (www.sis.pitt.edu)
  - Dotted notations (136.142.116.2)
  - to 32 bit integer IP address is simple
- Library routines exist to do these conversions:
  - Inet_addr - converts dotted decimal to 32 bit IP
  - gethostbyname - converts domain name to 32 bit IP and puts it in a part of a structure called hostent (host entity)

Host Addresses and hostent

- hostent * = gethostbyname(char * name)
  char * name is the domain name
- hostent * = gethostbyaddr(char * addr, int alen, int atype)
  char * addr is the binary address
  int alen = 4 for IP
  int atype = AF_INET for IP
- struct hostent
  struct hostent {
    char *h_name; /* host name */
    char **h_aliases; /* aliases */
    int h_addrtype; /* address type */
    int h_length; /* address length */
    char **h_addr_list; /* list of addresses */
  }
  #define h_addr h_addr_list[0]

Domain Name Conversion

```c
struct hostent *hptr;
char *examenam = "icarus.lis.pitt.edu";

if (hptr = gethostbyname(examenam)) {
  /* IP address is now in hptr->h_addr */
}
else {
  /* name was in error - handle it */
}
```
Finding a Port

- An address requires a port as well as a machine address.
- Ports are numbers or names – http, ftp, etc.
- Different ports are allocated for the same service using different transport (e.g. UDP, TCP)
- The function getservbyname can be used if a service name is provided, else, simply use the number provided

/etc/services

# Network services, Internet style
echo 7/tcp
echo 7/udp
sysstat 11/tcp users
daytime 13/tcp
daytime 13/udp
ftp-data 20/tcp
ftp 21/tcp
telnet 23/tcp
smtp 25/tcp
whois 43/tcp mail
pop2 109/tcp pop-2 POP V2
sunrpc 111/udp rpcbind
sunrpc 111/tcp rpcbind
imap 143/tcp

Service Name Conversion

```c
struct servent *sptr;
char *examplenam = "http";
if (sptr = getservbyname(examplenam))
{
    /* Service port is now in sptr->s_port */
}
else
{
    /* name was not found - handle it */
}"
```
Protocol Families

#define PF_UNSPEC AF_UNSPEC
#define PF_UNIX AF_UNIX
#define PF_INET AF_INET
#define PF_IMPLINK AF_IMPLINK
#define PF_PUP AF_PUP
#define PF_CHAOS AF_CHAOS
#define PF_NS AF_NS
#define PF_NBS AF_NBS
#define PF_ECMA AF_ECMA
#define PF_DATAKIT AF_DATAKIT
#define PF_CCITT AF_CCITT
#define PF_SNA AF_SNA
#define PF_DECnet AF_DECnet
#define PF_DLI AF_DLI

Protocol Identifier Lookup

struct protoent *pptr;
if (pptr = getprotobyname("udp"))
{ /*official number is in pptr->p_proto */
} else
{ /* error occurred - handle it */
};

struct protoent {
    char *p_name; /* protocol name */
    char **p_aliases; /* list of aliases */
    int p_proto; /* protocol number */
};

Allocate a socket

#include <sys/types.h>
#include <sys/socket.h>

int s; /* socket descriptor */
s = socket(PF_INET, SOCK_STREAM, ppe->proto)
/* arguments */ protocol family
               service type
               protocol identifier */
Connecting to a Remote Server

• CONNECT system call initiates a TCP 3-way handshake
  • Validates socket is valid and not already in use
  • Fills in the remote endpoint address from 2nd argument
  • Chooses a local endpoint address (IP and port number)
  • Initiates a TCP connection and returns a value
  • Blocks until connection is made or timeout occurs
    
    \[
    \text{retcode} = \text{connect}(s, \text{remaddr}, \text{remaddrlen});
    \]
    
    /* s = socket descriptor
    remaddr = address of structure of type sockaddr_in
    that specifies remote endpoint address
    remaddrlen = length in bytes of remote address */

Communicating with a Server

#define BLEN 120 /* buffer length */
char *req="request of some type";
char *buf[BLEN]; /* buffer for answer */
char *bptr; /* pointer to buffer */
int n; /* number bytes read */
int buflen; /* space left in buffer */
bptr = buf;
buflen = BLEN
write(s, req, strlen(req)); /* send request */
while ((n=read(s, bptr, buflen) > 0) {
    bptr += n;
    buflen -= n;
}

Reading Server Responses

• TCP is a stream-oriented protocol (sequence of bytes) and may segment or block data into packets
• Iteration on the read is necessary until no more data is sent for a server response
• One write by a server application may require one or more reads by the client
Closing a Connection

- Terminate connection and deallocate a socket
- Partial Close:
  - Server can not terminate connection until it knows all client data has been sent
  - Client needs to know that all server response data has been received
  - Thus, a direction for a close can be specified since each side knows when it is done. This is a partial close.

```c
errcode = shutdown(s, direction)
/* s = socket descriptor
   direction = 0 - no further input is allowed
               1 - no further output is allowed
               2 - no input or output is allowed */
```

Other Socket Utility Calls

- Host information
  - int getpeername(socket, sockaddr *remote, int length)
  - extracts remote machine endpoint address from socket structure
  - int = gethostid()
  - gets local host id -- 32 bit identifier
- Conversion of data format
  - htons - host-to-network byte order
  - ntohs - network-to-host byte order

Utility Functions From Comer

- Comer provides a set of utility functions that make the process of connecting easier
  - connectTCP(machine,service)
    - make a TCP connection
  - connectUDP(machine,service)
    - make a UDP connection
  - connectsock(host,service,protocol)
    - create the socket and connect