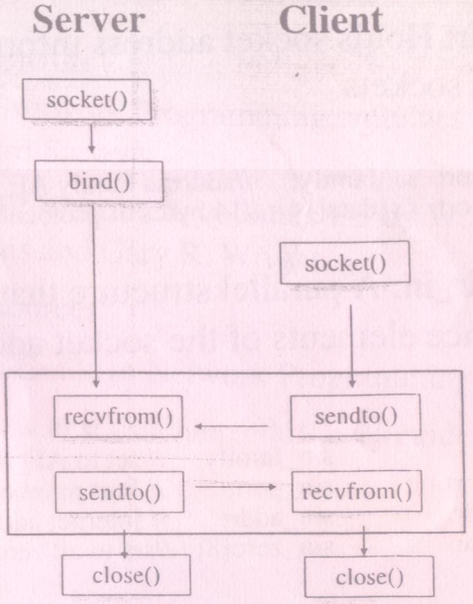
Types of Internet Sockets

- Stream Sockets (SOCK_STREAM)
  - Connection oriented
  - Rely on TCP to provide reliable two-way connected communication
- Datagram Sockets (SOCK_DGRAM)
  - Rely on UDP
  - Connection is unreliable

Connectionless Protocol



What is a socket?

- Socket: An interface between an application process and transport layer
  - The application process can send/receive messages to/from another application process (local or remote)via a socket
- In Unix jargon, a socket is a file descriptor – an integer associated with an open file
- Types of Sockets: **Internet Sockets**, unix sockets, X.25 sockets etc
  - Internet sockets characterized by IP Address (4 bytes) and port number (2 bytes)

Background

- Two types of "Byte ordering"
  - Network Byte Order: High-order byte of the number is stored in memory at the lowest address
  - Host Byte Order: Low-order byte of the number is stored in memory at the lowest address
  - Network stack (TCP/IP) expects Network Byte Order
- Conversions:
  - htons() - Host to Network Short
  - htonl() - Host to Network Long
  - ntohs() - Network to Host Short
  - ntohl() - Network to Host Long

socket structures

- struct sockaddr: Holds socket address information for many types of sockets

```
struct sockaddr {
        unsigned short  sa_family;    //address family AF_xxx
        unsigned short  sa_data[14]; //14 bytes of protocol addr
}
```

- struct sockaddr_in: A parallel structure that makes it easy to reference elements of the socket address

```
struct sockaddr_in {
        short int               sin_family;   // set to AF_INET
        unsigned short int      sin_port;     // Port number
        struct in_addr          sin_addr;     // Internet address
        unsigned char           sin_zero[8]; //set to all zeros
}
```

socket() -- Get the file descriptor

- int socket(int domain, int type, int protocol);
  - domain should be set to AF_INET
  - type can be SOCK_STREAM or SOCK_DGRAM
  - set protocol to 0 to have socket choose the correct protocol based on type
  - socket() returns a socket descriptor for use in later system calls or -1 on error

bind() - what port am I on?

- Used to associate a socket with a port on the local machine
  - The port number is used by the kernel to match an incoming packet to a process
- int bind(int sockfd, struct sockaddr *my_addr, int addrlen)
  - sockfd is the socket descriptor returned by socket()
  - my_addr is pointer to struct sockaddr that contains information about your IP address and port
  - addrlen is set to sizeof(struct sockaddr)
  - returns -1 on error
- my_addr.sin_port = 0; //choose an unused port at random
- my_addr.sin_addr.s_addr = INADDR_ANY; //use my IP addr

Dealing with IP Addresses

- int inet_aton(const char *cp, struct in_addr *inp);
- Example usage:

```
struct sockaddr_in    my_addr;
my_addr.sin_family = AF_INET;
my_addr.sin_port = htons(MYPORT);
inet_aton("10.0.0.5",&(my_addr.sin_addr));
memset(&(my_addr.sin_zero),'\0',8);
```

- inet_aton() gives non-zero on success and zero on failure
- To convert binary IP to string: inet_noa()

```
printf("%s",inet_ntoa(my_addr.sin_addr));
```

close() - Bye Bye!

- int close(int sockfd);
  - Closes connection corresponding to the socket descriptor and frees the socket descriptor
  - Will prevent any more sends and recvs

sendto() and recvfrom() - DGRAM style

- int sendto(int sockfd, const void *msg, int len, int flags, const struct sockaddr *to, int tolen);
  - *to* is a pointer to a struct sockaddr which contains the destination IP and port
  - *tolen* is sizeof(struct sockaddr)
- int recvfrom(int sockfd, void *buf, int len, int flags, struct sockaddr *from, int *fromlen);
  - *from* is a pointer to a local struct sockaddr that will be filled with IP address and port of the originating machine
  - *fromlen* will contain length of address stored in *from*

References

- Books:
  - Unix Network Programming, volumes 1-2 by W. Richard Stevens.
  - TCP/IP Illustrated, volumes 1-3 by W. Richard Stevens and Gary R. Wright
- Web Resources:
  - Beej's Guide to Network Programming
    - www.ecst.csuchico.edu/~beej/guide/net/

Summary

- Sockets help application process to communicate with each other using standard Unix file descriptors
- Two types of Internet sockets: SOCK_STREAM and SOCK_DGRAM
- Many routines exist to help ease the process of communication

Socket Programming Instruction Sheet

In this lab, our goal is to make two machines talk to each other. You will have to write two programs: ***listener.c*** and ***talker.c*** based on the concepts you have learned so far.

listener essentially sits on a machine waiting for incoming messages on a specific port (say 5000). It prints whatever message it gets on that port onto the screen. **talker** reads whatever the user types on the command line and sends this message to the **listener**.

To make things easy for you, we have provided the skeleton of the program. Your job is to fill in the blanks. The provided data sheet contains the syntax of all the functions you would need for this exercise. Note that the variables used in the syntax may be very different from those used in the skeleton, so use caution.

Initially, you will run both **listener** and **talker** on the same machine utilizing the loop back feature. This will help in debugging the programs. For this, you will need to specify the destination address (listener's) as 127.0.0.1. The source address (talker's) can be obtained by typing "ifconfig eth0" at the command prompt. Once the programs are running correctly, you can pair with one of the other teams and test that your **talker** runs with their **listener** and vice-versa. In this case change the destination IP address appropriately.

```
/******** listener.c – listens for incoming messages on port 5000 ********/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define MYPORT _____  // the port talker will be connecting to
#define MAXBUFLEN 100

int main(void) {
    int sockfd;
    struct sockaddr_in my_addr;    // my address information
    struct sockaddr_in their_addr; // talker's address information
    int addr_len, numbytes;
    char buf[MAXBUFLEN];

    if ((sockfd = socket(_____, _____, _____)) == -1) {//exit on error
        perror("socket");
```

```
        exit(1);
    }

    my_addr.sin_family = _____;
    my_addr.sin_port = htons(_____);
    my_addr.sin_addr.s_addr = _____; // automatically fill with my IP
    memset(&(my_addr.sin_zero), 0, 8); // zero the rest of the struct
    if (bind(_____, _____,_____) == -1) {//exit on error
        perror("bind");
        exit(1);
    }

    addr_len = _____;
    if ((numbytes=recvfrom(_____,_____,_____, _____,
                _____, _____)) == -1) {//exit on error
        perror("recvfrom");
        exit(1);
    }

    printf("got packet from %s IP address\n",inet_ntoa(_____));
    printf("packet is %d bytes long\n",_____);
    buf[numbytes] = '\0';
    printf("packet contains the message \"%s\"\n",_____);

    close(_____);
    return 0;
}

/***** talker.c reads input from command line and sends message to listener ******/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

#define LPORT _____  // the port listener is waiting on

int main(int argc, char *argv[]) {
    int sockfd;
```

```
struct sockaddr_in their_addr; // listener's address information
int numbytes;

if (argc != 3) {
   fprintf(stderr,"usage: talker hostname message\n");
   exit(1);
}

if ((sockfd = socket(_____, _____, _____)) == -1) {
   perror("socket");
   exit(1);
}

their_addr.sin_family = _____;
their_addr.sin_port = htons(_____);
inet_aton(_____, _____); // fill the IP address in their_addr.sin_addr
memset(&(their_addr.sin_zero), '\0', 8); // zero the rest of the struct

if ((numbytes=sendto(_____, _____, _____, 0,
   _____, _____)) == -1) { //exit on error
   perror("sendto");
   exit(1);
}
printf("sent %d bytes to %s IP address \n", _____, inet_ntoa(_____));
close(_____);
return 0;
}
```

Extra lab: UDP socket programming

- Consider three machines A, B, C arranged logically in a cyclical order (clock-wise: A, B, C)
- You have to write an application using UDP socket programming to achieve the following functionality
- You can write the program in any of the following languages: C, C++, Java, Perl; but I recommend C or C++

1. The application should have a header of its own, with at least the following fields: an 8-bit field representing a colour logically, and an 8-bit field representing the TTL (time-to-live), an 8-bit sequence number, an 8-bit source ID field
2. The application-level source IDs for the three machines can be 10, 20, and 30
3. Packets can be of colours red, blue, green (say integer values 0, 1, 2)
4. The application uses the following rules: