Discussion Session #10

EE450: Computer Networks

Socket Programming Supporting Slides

Introduction

- Sources:
  - http://beej.us/guide/bgnet/
  - http://man7.org/linux/man-pages/
- Connect to nunki.usc.edu using some X-Window software
- To compile some C program

```
cc -o my_program my_source_name.c -lnsl -lsocket –
lresolv
```

- Socket Types
  - **Stream Sockets (TCP): today's discussion**
    - Also called SOCK_STREAM
  - Datagram Sockets (UDP)
    - Also called SOCK_DGRAM

Client Stream-Socket

- The steps involved in establishing a socket on the **client** side are as follows:
  - Create a socket with the **socket**() system call
  - Connect the socket to the address of the server using the **connect**() system call
  - Send and receive data. There are a number of ways to do this, but the simplest is to use the **read**() and **write**() system calls.

Server Stream-Socket

- The steps involved in establishing a socket on the **server** side are as follows:
  - Create a socket with the **socket**() system call
  - Bind the socket to an address using the **bind**() system call. For a server socket on the Internet, an address consists of a port number on the host machine.
  - Listen for connections with the **listen**() system call
  - Accept a connection with the **accept**() system call. This call typically blocks until a client connects with the server.
  - Send and receive data

Structures and Functions Used for Socket Programming in C

addrinfo

- Used to prepare the socket address structures for subsequent use. It's also used in host name lookups, and service name lookups
  - Call to getaddrinfo() to fill out your struct addrinfo for you

```
struct addrinfo {
    int              ai_flags;     // AI_PASSIVE, AI_CANONNAME, etc.
    int              ai_family;    // AF_INET, AF_INET6, AF_UNSPEC
    int              ai_socktype;  // SOCK_STREAM, SOCK_DGRAM
    int              ai_protocol;  // use 0 for "any"
    size_t           ai_addrlen;   // size of ai_addr in bytes
    struct sockaddr *ai_addr;      // struct sockaddr_in or _in6
    char            *ai_canonname; // full canonical hostname
    struct addrinfo *ai_next;      // linked list, next node
};
```

sockaddr

- Holds socket address information for many types of sockets.

```
struct sockaddr {
    unsigned short    sa_family;
            // address family, AF_xxx
    char              sa_data[14];
            // 14 bytes of protocol address (dest ip and
port)
};
```

sockaddr_in

- A parallel structure: struct sockaddr_in ("in" for "Internet") to be used with IPv4
- a pointer to a struct sockaddr_in can be cast to a pointer to a struct sockaddr and vice-versa

```
struct sockaddr_in {
    short int          sin_family;  // Address family, AF_INET
    unsigned short int sin_port;    // Port number
    struct in_addr     sin_addr;    // Internet address
    unsigned char      sin_zero[8]; // Same size as struct sockaddr
};
```

in_addr

- 4-byte IP

```
struct in_addr {
    uint32_t s_addr; // that's a 32-bit int (4 bytes)
};
```

sockaddr_storage

- Designed to be large enough to hold both IPv4 and IPv6 structures. (For some calls, sometimes you don't know in advance if it's going to fill out your struct sockaddr with an IPv4 or IPv6 address).

```
struct sockaddr_storage {
    sa_family_t  ss_family;     // address family
    // all this is padding, implementation specific, ignore it:
    char      __ss_pad1[_SS_PAD1SIZE];
    int64_t   __ss_align;
    char      __ss_pad2[_SS_PAD2SIZE];
};
```

getaddrinfo()

- Converts human-readable text strings representing hostnames or IP addresses into a dynamically allocated linked list of struct addrinfo structures
  - allocates and initializes a linked list of addrinfo structures, one for each network address that matches node and service, subject to any restrictions imposed by hints, and returns a pointer to the start of the list in res. The items in the linked list are linked by the ai_next field
- There are several reasons why the linked list may have more than one
  - addrinfo structure, including: the network host is multihomed, accessible over multiple protocols (e.g., both AF_INET and AF_INET6);
  - or the same service is available from multiple socket types (one SOCK_STREAM address and another SOCK_DGRAM address, for example).
  - Normally, the application should try using the addresses in the order in which they are returned.

getaddrinfo()

```
int getaddrinfo(const char *node,
        // e.g. "www.example.com" or IP
        const char *service,  // e.g. "http" or port number
        const struct addrinfo *hints,
        struct addrinfo **res);
```

socket()

- Simply returns to you a socket descriptor that you can use in later system calls, or -1 on error

```
int s;
struct addrinfo hints, *res;
// do the lookup
// [pretend we already filled out the "hints" struct]
getaddrinfo("www.example.com", "http",
&hints,  &res);
// [again, you should do error-checking on getaddrinfo(),
and walk
// the "res" linked list looking for valid entries instead of just
// assuming the first one is good (like many of these
examples do.)
// See the section on client/server for real examples.]
s = socket(res->ai_family, res->ai_socktype, res-
>ai_protocol);
```

bind()

- Associate that socket with a port on your local machine

int bind(int sockfd, struct sockaddr *my_addr, int addrlen);

connect()

- Connect to a remote host

```
int connect(int sockfd, struct sockaddr *serv_addr, int addrlen);
```

listen()

- Listen for incoming connections

```
int listen(int sockfd, int backlog);
```

accept()

- Accept an incoming connection

int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);

Send() and receive()

- Send and receive data over stream sockets

```
int send(int sockfd, const void *msg, int len, int flags);
int recv(int sockfd, void *buf, int len, int flags);
```

close() and shutdown()

- Close a connection on the socket descriptor (both directions)

  close(sockfd);

- Shutdown a connection on the socket descriptor (you can choose direction)

  int shutdown(int sockfd, int how);

getpeername()

- Tells you who is at the other end of a connected stream socket

int getpeername(int sockfd, struct sockaddr *addr, int *addrlen);

gethostname()

- Returns the name of the computer that your program is running on

```
int gethostname(char *hostname, size_t size);
```

waitpid()

- Wait for process to change state

int waitid(idtype_t idtype, id_t id, siginfo_t *infop, int options);

  - "WNOHANG" argument
    - Prevents the parent process from being blocked

http://linux.die.net/man/2/waitpid

sigaction()

- Examine and change a signal action

int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);

http://linux.die.net/man/2/sigaction