The BSD UNIX Socket Interface
(CS 640 Lecture)

Robert T. Olsen
olsen@cs.wisc.edu
7390CS
Office Hours: 3-5T, 11-12F

Assignment 1

- Work Individually (no groups)
- Due Date: in class, Monday, September 19

  - exception for Ph.D. Qualifying Examinees

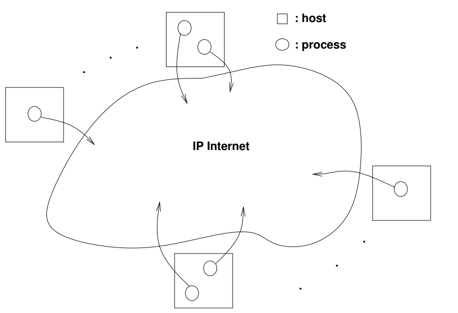
    . due date: September 30
- Other notes

  - everyone is now authorized to register

Interprocess Communication (IPC)

- Goal:

  - write programs that communicate with each other

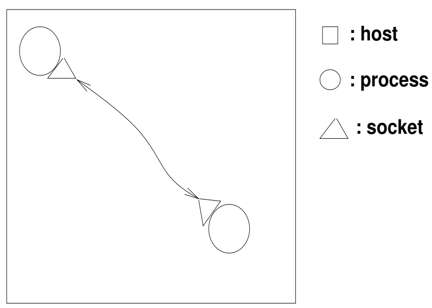


  - examples: telnet, rlogin, ftp, Mosaic

Interprocess Communication (IPC)

- How:

  - use *sockets*: abstract endpoints of communication



Interprocess Communication (IPC)

- Issues

  - creating sockets

  - naming (identifying) sockets

  - sending/receiving data over sockets

- Mechanisms

  - UNIX system calls and library routines (functions called from C programs)

    . **check all return codes**

```
if ((code = syscall()) < 0) {
    perror("syscall");
    exit(1);
}
```

Creating Sockets

int socket(domain, type, protocol)

int domain, type, protocol

- creates and endpoint for communication

- include files: *<sys/types>, <sys/socket>*

- return value: an integer *descriptor* (like a file descriptor) used in future calls, or -1 on error

Types of Sockets

- socket defined by 3 parameters

1) *domain*: address family (protocol family)

. determines address structure

. e.g., AF_UNIX, AF_INET, AF_OSI

(also PF_UNIX, PF_INET, PF_OSI)

. you will use AF_INET

2) *type*: style of communication

. e.g,. SOCK_DGRAM provides unreliable, connectionless service

e.g., SOCK_STREAM provides reliable byte-stream service

. you will use SOCK_DGRAM

3) *protocol*: specifies particular protocol

. usually already defined by domain and type (e.g., TCP for AF_INET and SOCK_STREAM, UDP for SOCK_DGRAM)

. you will use 0 (default protocol)

- Example

```
if ((sd = socket(AF_INET,SOCK_DGRAM,0)) < 0){
    perror("socket");
    exit(1);
}
```

Naming Sockets

int bind(sd, name, namelen)

int sd;
struct sockaddr *name;
int namelen;

- assigns a name to a socket

- include files: *<sys/types>*, *<sys/socket>*, *<netinet/in.h>*

- return value: 0 on success, -1 on error

Socket Addresses

- several types of socket addresses

- you will use *sockaddr_in*, because you use AF_INET

- *sockaddr_in* is a C structure with 3 important fields: *sin_family*, *sin_addr*, *sin_port*

  . *sin_family* determines remaining address structure (e.g., AF_INET)

  . *sin_addr* identifies host

  . *sin_port* identifies process on host

Internet Addresses and Ports

- *sin_addr* values

- four bytes (a.b.c.d)

- e.g., *sun22* is 128.105.40.22

(see */etc/hosts*)

- if you specify *INADDR_ANY* for local host address, all host addresses apply

- *sin_port* values

- 0 - 1024 reserved for system

- well known ports are important

  . finger is port 79, rlogin is 513

  (see */etc/services*)

- if you specify 0, the system picks a port

- Example (binding local address)

```
struct sockaddr_in        addr;

bzero((char *) &addr, sizeof(addr));
addr.sin_family = AF_INET;
addr.sin_port   = htons(0);
addr.sin_addr.s_addr = htonl(INADDR_ANY);

if (bind(sd, (struct sockaddr *) &addr, sizeof(addr)) < 0){

        perror("bind");

        exit(1);

}
```

Sending/Receiving Data (SOCK_DGRAM)

int sendto(sd, msg, len, flags, to, tolen))

int sd;
char *msg;
int len, flags;
struct sockaddr *to;
int tolen;

- return value: number of bytes sent, or -1 on error

- Key Points:

- must have created a socket with descriptor *sd*

and bound local address to it, if a reply is desired

- must know destinations address and port and have

filled *to* structure appropriately

- *flags* parameter will be 0 for you

Sending/Receiving Data (SOCK_DGRAM)

int recvfrom(sd, buf, len, flags, from, fromlen))

int sd;
char *buf;
int len, flags;
struct sockaddr *from;
int *tolen;

- return value: number of bytes received,

or -1 on error

- Key Points:

- must have created a socket with descriptor *sd*

and bound appropriate address to it

- source address is filled in the *from* structure

- *flags* parameter will be 0 for you

- by default, call blocks until data arrives

```
/*
 * Example of client using UDP protocol.
 */

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

main(argc, argv)
int       argc;
char      *argv[];
{
          int                          sockfd;
          struct sockaddr_in   cli_addr, serv_addr;

          /*
           * Fill in the structure "serv_addr" with the address of the
           * server that we want to send to.
           */

          bzero((char *) &serv_addr, sizeof(serv_addr));
          serv_addr.sin_family     = AF_INET;
          serv_addr.sin_addr.s_addr = inet_addr(SERV_HOST_ADDR);
          serv_addr.sin_port       = htons(SERV_UDP_PORT);

          /*
           * Open a UDP socket (an Internet datagram socket).
           */

          if ( (sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
                    perror("client: cannot open socket");
                    exit(1)
          }

          /*
           * Bind any local address for us.
           */

          bzero((char *) &cli_addr, sizeof(cli_addr));         /* zero out */
          cli_addr.sin_family     = AF_INET;
          cli_addr.sin_addr.s_addr = htonl(INADDR_ANY);
          cli_addr.sin_port       = htons(0);
          if (bind(sockfd, (struct sockaddr *) &cli_addr, sizeof(cli_addr)) < 0) {
                    perror("client: cannot bind local address");
                    exit(1);
          }


          if (sendto(sockfd, msg, len, 0, &serv_addr, sizeof(serv_addr) != len) {
                    perror("client: sendto");
                    exit(1);
          }

          n = recvfrom(sockfd, buf, bufsize, 0, (struct sockaddr *) 0, (int *) 0);
          if (n < 0) {
                    perror("client: recvfrom");
                    exit(1);
          }

          close(sockfd);
          exit(0);
}
```

```
/*
 * Example of server using UDP protocol.
 */

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

main(argc, argv)
int       argc;
char      *argv[];
{
          int                          sockfd;
          struct sockaddr_in   serv_addr, cli_addr;


          /*
           * Open a UDP socket (an Internet datagram socket).
           */

          if ( (sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
                    perror("server: cannot open socket");
                    exit(1)
          }

          /*
           * Bind our local address so that the client can send to us.
           */

          bzero((char *) &serv_addr, sizeof(serv_addr));
          serv_addr.sin_family     = AF_INET;
          serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
          serv_addr.sin_port       = htons(SERV_UDP_PORT);

          if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0) {
                    perror("server: cannot bind local address");
                    exit(1);
          }


          while (1) {

                    n = recvfrom(sockfd, msg, msgsize, 0, &cli_addr, &clilen);
                    if (n < 0) {
                              perror("server: recvfrom");
                              exit(1);
                    }

                    if (sendto(sockfd, msg, n, 0, &cli_addr, clilen) != n)
                              perror("server: sendto");
                              exit(1);
                    }
          }



}
```

Other System Calls

int connect(sd, name, namelen)

int sd;
struct sockaddr *name;
int namelen;

- specifies peer with which *sd* is to be associated

int listen(sd, backlog)

int sd, backlog;

- specifies maximum backlog of connections a server will allow

int accept(sd, addr, addrlen)

int sd;
struct sockaddr *addr;
int *addrlen

- extracts first connection off queue of pending connections

int close(sd)

int sd;

- deletes descriptor from system tables

int select(width, rfds, wfds, efds, timeout)

int width;
fd_set *rfds, *wfds, *efds;
struct timeval *timeout;

- allows server to wait for data on multiple sockets at the same time

- can be blocking, non-blocking, or timeout

Library Routines

struct hostent *gethostbyname(name)

char *name;

- returns pointer to structure from which host address can be obtained

int gethostname(name, namelen)

char *name;
int namelen;

- returns name of current processor

int getsockname(sd, name, namelen)

int sd;
struct sockaddr *name;
int *namelen;

- returns address structure for *sd* (can get port information, etc.)

- **htonl(), htons(), ntohl(), ntohs()**

- convert between network byte order and host byte order

- required since some machines are big endian and others are little endian (network order is big endian)

- big endian

address	byte
0	0 1 2 3
4	4 5 6 7

- little endian

address	byte
0	3 2 1 0
4	7 6 5 4

- convert to network order before sending, and back to host order after receiving

How to Get Help

- Unix Programmer's Manual (man pages)

  - man *function-name*

- Document located on Web Server

  (*Berkeley UNIX System Calls and IPC*)

- TA's (e-mail or office hours)