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

- host
- process
- socket
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);
        }

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
     - 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)

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

Example

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

Goal:
- write programs that communicate with each other

IP Internet.
- 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
    ```c
    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
   - 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
  - 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
  - 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)

```c
int sendto(sd, msg, len, flags, to, tolen);
```

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

- Key Points:
  - must have created a socket with descriptor `sd`
  - must have created a socket with descriptor `sd`
  - and bound local address to it, if a reply is desired
  - and bound local address to it, if a reply is desired
  - must know destinations address and port and have filled to structure appropriately
  - filled to structure appropriately
  - flags parameter will be 0 for you
  - flags parameter will be 0 for you

```
int sockfd;
struct sockaddr_in cli_addr, serv_addr;
```

- Fill in the structure `serv_addr` with the address of the server that we need to send to.

```
bind(sockfd, (struct sockaddr *) &addr, sizeof(addr));
```

- Opens a UDP socket (an Internet datagram socket).

- `socket` returns a descriptor (a file-like object) that is used to communicate with a server.

- `socket` returns a descriptor (a file-like object) that is used to communicate with a server.

```
sockaddr_in serv_addr, cli_addr;
```

- `sockaddr_in` is a C structure with 3 important fields:
  - `sin_family`: specifies particular protocol
  - `sin_port`: identifies process on host
  - `sin_addr`: identifies host

```
sockaddr_in addr;
```

- `sockaddr_in` is a C structure with 3 important fields:
  - `sin_family`: specifies particular protocol
  - `sin_port`: identifies process on host
  - `sin_addr`: identifies host

```
struct sockaddr_in serv_addr, cli_addr;
```

- `sockaddr_in` is a C structure with 3 important fields:
  - `sin_family`: specifies particular protocol
  - `sin_port`: identifies process on host
  - `sin_addr`: identifies host

```
serv_addr.sin_port = htons(SERV_UDP_PORT);
```

- Use `htons` to convert an integer from host byte order to network byte order.

- Use `htons` to convert an integer from host byte order to network byte order.

```
serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
```

- Use `htonl` to convert an integer from host byte order to network byte order.

- Use `htonl` to convert an integer from host byte order to network byte order.

```
bzero((char *) &serv_addr, sizeof(serv_addr));
```

- zeroes the structure `serv_addr` (also called a sockaddr)

- zeroes the structure `serv_addr` (also called a sockaddr)

```
main(argc, argv)
```

- `main` is the entry point of the program.

- `main` is the entry point of the program.

```
#include <sys/socket>
```

- `#include` is a preprocessor directive that includes the specified header file.

- `#include` is a preprocessor directive that includes the specified header file.

```
#include <netinet/in.h>
```

- `#include` is a preprocessor directive that includes the specified header file.

- `#include` is a preprocessor directive that includes the specified header file.

```
char *argv[];
```

- `argv` is an array of characters that contains the command line arguments.

- `argv` is an array of characters that contains the command line arguments.

```
struct sockaddr *name;
```

- `struct sockaddr` is a C structure that represents a network address.

- `struct sockaddr` is a C structure that represents a network address.

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

- `sendto` is a system call that sends data to a specified socket.

- `sendto` is a system call that sends data to a specified socket.

```
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

```
sockaddr_in arg)
```

- `sockaddr_in` is a C structure that represents a network address.

- `sockaddr_in` is a C structure that represents a network address.

```
sockaddr_in cli_addr, serv_addr;
```

- `sockaddr_in` is a C structure with 3 important fields:
  - `sin_family`: specifies particular protocol
  - `sin_port`: identifies process on host
  - `sin_addr`: identifies host

```
cli_addr.sin_addr.s_addr = htonl(INADDR_ANY);
```

- Use `htonl` to convert an integer from host byte order to network byte order.

- Use `htonl` to convert an integer from host byte order to network byte order.

```
cli_addr.sin_family = AF_INET;
```

- Use `AF_INET` to specify the address family (IP version 4).

- Use `AF_INET` to specify the address family (IP version 4).

```
if (bind(sockfd, (struct sockaddr *) &cli_addr, sizeof(cli_addr)) < 0) {
```

- `bind` is a system call that binds a socket to a local address.

- `bind` is a system call that binds a socket to a local address.

```
exit(1)
```

- `exit` is a system call that terminates the program

- `exit` is a system call that terminates the program

```
if (sendto(sockfd, msg, len, 0, &serv_addr, sizeof(serv_addr) != len) {
```

- `sendto` is a system call that sends data to a specified socket.

- `sendto` is a system call that sends data to a specified socket.

```
exit(1)
```

- `exit` is a system call that terminates the program

- `exit` is a system call that terminates the program

```
if (recvfrom(sockfd, buf, bufsize, 0, (struct sockaddr *) 0, (int *) 0));
```

- `recvfrom` is a system call that receives data from a specified socket.

- `recvfrom` is a system call that receives data from a specified socket.

```
exit(1)
```

- `exit` is a system call that terminates the program

- `exit` is a system call that terminates the program

```
char *argv[];
```

- `argv` is an array of characters that contains the command line arguments.

- `argv` is an array of characters that contains the command line arguments.

```
#include <sys/socket>
```

- `#include` is a preprocessor directive that includes the specified header file.

- `#include` is a preprocessor directive that includes the specified header file.

```
#include <netinet/in.h>
```

- `#include` is a preprocessor directive that includes the specified header file.

- `#include` is a preprocessor directive that includes the specified header file.

```
char *argv[];
```

- `argv` is an array of characters that contains the command line arguments.

- `argv` is an array of characters that contains the command line arguments.

```
int main(argc, argv)
```

- `main` is the entry point of the program.

- `main` is the entry point of the program.

```
#include <sys/socket>
#include <netinet/in.h>
#include <stdio.h>
#include <string.h>
```

- `#include` is a preprocessor directive that includes the specified header file.

- `#include` is a preprocessor directive that includes the specified header file.

- `#include` is a preprocessor directive that includes the specified header file.

- `#include` is a preprocessor directive that includes the specified header file.

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

- `Example` is a placeholder for a program example.

- `Example` is a placeholder for a program example.

```
Internet Addresses and Ports
```

- `Internet Addresses and Ports` is a section title.

- `Internet Addresses and Ports` is a section title.

```
Types of Sockets
```

- `Types of Sockets` is a section title.

- `Types of Sockets` is a section title.

```
Mechanisms
```

- `Mechanisms` is a section title.

- `Mechanisms` is a section title.

```
Creating Sockets
```

- `Creating Sockets` is a section title.

- `Creating Sockets` is a section title.

```
C programs)
```

- `C programs` is a subheading.

- `C programs` is a subheading.

```
include files:
```

- `include files:` is a subheading.

- `include files:` is a subheading.

```
return value: an integer
```

- `return value:` is a description of the function's return value.

- `return value:` is a description of the function's return value.

```
return value: number of bytes received,
```

- `return value:` is a description of the function's return value.

- `return value:` is a description of the function's return value.

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

- `return value:` is a description of the function's return value.

- `return value:` is a description of the function's return value.

```
descriptor
```

- `descriptor` is a term used in the explanation.

- `descriptor` is a term used in the explanation.

```
Key Points:
```

- `Key Points:` is a section title.

- `Key Points:` is a section title.

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

- `return value:` is a description of the function's return value.

- `return value:` is a description of the function's return value.

```
return value: number of bytes received,
```

- `return value:` is a description of the function's return value.

- `return value:` is a description of the function's return value.

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

- `Example` is a placeholder for a program example.

- `Example` is a placeholder for a program example.
### 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(), htonl(), 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 0123
      4 4567
  - little endian
    - address          byte
      0 3210
      4 7654
  - convert to network order before sending.
  - and back to host order after receiving