Introduction to Sockets in C

CST 357/457 – Systems Programming

Michael Ruth, Ph.D.
Associate Professor
Computer Science & I.T.
mruth@roosevelt.edu

Objectives

- Discuss sockets concepts including IPC
- Explain socket creation including domains, types, protocols, and addresses
- Discuss socket functions including send/receive, close, and IO relation
- Explain binding sockets to addresses, listening for connections, making connections, and transferring data
- Discuss client/server operation using sockets

Introduction

- Sockets are the ultimate form of IPC
  - Network IPC
    - Goals was to achieve the same interface for
      - Intermachine communication
      - Intramachine communication
  - Sockets are abstractions of a communication endpoint
  - Socket Descriptors
    - Handle to a socket
    - Very much like file descriptors
      - In fact, the majority of functions that work with files work with sockets

Sockets

- To create a socket we use the socket function:
  - `#include <sys/socket.h>`
  - `int socket(int domain, int type, int protocol)`
    - Protocol is usually zero to select the default protocol for the given domain and socket type
    - Socket Type determines the type of communication mechanism
    - Socket Domain determines the nature of the communication

Domains & Types

- Socket Domains:
  - **AF_INET**
  - AF_INET6
  - AF_UNIX
- Socket Type:
  - SOCK_DGRAM
    - UDP interface
  - SOCK_RAW
    - Datagram interface to IP
  - SOCK_SEQPACKET
    - Fixed-length, reliable, sequenced, connection-oriented messages
  - **SOCK_STREAM**
    - TCP interface

Sockets & Functions

- socket and open are very related
- Some important functions and their operations
  - close
    - deallocates the socket
  - read
    - equivalent to recv w/o flags
  - write
    - equivalent to send w/o flags

Sockets and Bidirectional I/O

- Communication via Sockets is bi-directional
- We can disable the I/O on a socket with the following:
  - `int shutdown(int sockfd, int how)`
  - How parameter–
    - SHUT_RD – disables reading
    - SHUT_RW – disables writing
    - SHUT_RDRW – disables both
- Why would we wish to use this:
  - Close deallocates the endpoint only when the last active reference is closed, but shutdown allows us to deactivate the socket
  - Sometimes it is convenient to shut down I/O in a direction to inform the other side that we are done (in that way)

Byte Ordering

- Little/Big Endian
  - Big-endian – the highest byte address occurs in the least significant byte (LSB)
    - EX: Solaris 9 – Sun Sparc
  - Little-Endian – the least significant byte (LSB) contains the lowest byte address
    - EX: Linux 2.4.22 - Intel Pentium

- Host/Network
  - Host implies the computer you're on
  - Network refers to the fact that TCP uses big-endian

Host/Network Byte Order Translation

- `#include <arpa/inet.h>`
- `uint32_t htonl(uint32_t hostint32)`
- `uint16_t htons(uint16_t hostint16)`
- `uint32_t ntonl(uint32_t netint32)`
- `uint16_t ntons(uint16_t netint16)`

Address Format

- A set of structs allow us to perform 90% of what we need to do
  - **sockaddr**
  - **sockaddr_in**
  - **in_addr**

struct sockaddr

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

- This structure holds socket address information for many types of sockets:
  - *sa_family*
    - AF_INET
  - *a_data* contains a ***destination address*** and ***port number*** for the socket

struct sockaddr_in

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

- programmers created a parallel structure for sockaddr – can be substituted
  - **sin_port** – must be in network byte order
  - **sin_addr** – must be in network byte order

IP Address and port in NBO

- A simple and quick mechanism to perform this is:
  - `in_addr_t inet_addr(const char *cp);`
    - converts an IP address in numbers-and-dots notation (cp) into an unsigned long
    - Already in Network Byte Order
  - Port must be converted
    - EX: `htons(13);`

ntoa (network to address)

- `char *inet_ntoa(struct in_addr in);`
  - Converts the address specified by in_addr to a human readable string in numbers-and-dots notation

Address Resolution

- Ideally, we shouldn't be forced to know the internal structures
  - Additionally, we wish to be able to perform hostname and server lookups
    - w/o scanning /etc/hosts or /etc/services
- `struct hostent *gethostbyname(char *hostname);`
  - Resolves the IP name for the given hostname

struct hostent

```
struct hostent {
   char* h_name;
   char** h_aliases;
   int h_addrtype;
   int h_length;
   char** h_addr_list;
   #define h_addr h_addr_list[0]
};
```

- h_name: This is the official name of the host, i.e. the full address.
- h_aliases: a pointer to the list of aliases (other names) the host might have.
- h_addrtype: The type of address this host uses.
- h_length: The length of the address. Different address types might have different lengths.
- h_addr_list: A pointer to the list of addresses of the host. Note that a host might have more than one address, as explained earlier.
- h_addr: In older systems, there was only the h_addr field, so it is defined here so old programs could compile without change on newer systems.

Associating Addresses With Sockets

- **`int bind(int sockfd, struct sockaddr *addr, socklen_t len)`**
  - Returns 0 if ok, -1 on error
  - Some restrictions:
    - Address specified must be valid for the machine this is running on
    - Address must match the format specified by address family
    - The port number cannot be less than 1024 w/o appropriate permissions
    - Only one socket per address (there are exceptions)
  - **portnumber == 0**
    - Chooses the first unused port
  - **INADDR_ANY**
    - automatically fill in the IP address of the machine the process is running on

Defaults and Discovery

- Since we have some default modes, let’s discuss how we can find out what we actually are bound to
  - `int getsockname(int sockfd, struct sockaddr *addr, socklen_t alenp)`
    - Discover the address bound to the socket
  - `int getpeername(int sockfd, struct sockaddr *addr, socklen_t len)`
    - If the socket is connected, return the address of its peer

Time to actually Connect

- If we are dealing with connection-oriented network service (SOCK_STREAM or SOCK_SEQPACKET) we need to establish a connection
  - **`int connect(int sockfd, struct sockaddr *addr, socklen_t len)`**
    - There may be many reasons for failure
  - Additionally, can connect be used with SOCK_DGRAM ?
    - Yes, but….

Server Mechanisms

- A server announces that it is willing to accept connect requests by calling:
  - `int listen(int sockfd, int backlog)`
    - Backlog is a queue size

- We retrieve the connections from the server by calling
  - `int accept(int sockfd, struct sockaddr *addr, socklen_t *len)`
    - The descriptor returned is the descriptor connected to the process which called connect
    - Accept blocks!

Transferring data

- Connection Oriented
  - Sock presented by sockfd must have already been opened and connected
    - send
    - recv
- Connectionless
  - No need to open connection, simply use the following methods:
    - sendto
    - recvfrom

send

- **`ssize_t send(int sockfd, void *buf, size_t nbytes, int flags)`**

- Identical to write except for flags argument:
  - `MSG_DONTROUTE`
  - `MSG_EOR`
  - `MSG_DONTWAIT`

recv

- **`ssize_t recv(int sockfd, void *buf, size_t nbytes, int flags)`**
  - Again same as read except for flags:
    - `MSG_PEEK`
    - `MSG_WAITALL`
    - `MSG_TRUNC`

sendto & recvfrom

- ssize_t sendto(int sockfd, void *buf, size_t nbytes, int flags, struct sockaddr *destaddr, socklen_t destlen)
- ssize_t recvfrom(int sockfd, void *buf, size_t nbytes, int flags, struct sockaddr *addr, socklen_t addrlen)

Socket Options

- We can get and set socket options
  - Three kinds of options:
    - Generic options for all socket types
    - Options that are managed at the socket level, but depend on the underlying protocols for support
    - Protocol-specific options unique to each individual protocol
  - **`int setsockopt(int sockfd, int level, int option, void *val, socklen_t len)`**
  - **`int getsockopt(int sockfd, int level, int option, void *val, socklen_t lenp)`**
    - Level identifies the protocol
      - If generic then level == SOL_SOCKET

Socket Options Table

Options	Description
SO_ACCEPTCONN	Return whether a socket listening
SO_BROADCAST	Broadcast datagrams if val is nonzero
SO_DONTROUTE	Bypass normal routing
SO_KEEPALIVE	Periodic keep alive messages enabled if val is nonzero
SO_REUSEADDR	Reuse addresses in bind if val is nonzero
SO_SNDBUF	The size in bytes of send buffer
SO_RCVBUF	The size in bytes of receive buffer

Client/Server Model

- The client-server model is used to divide the work of Internet programs into two parts:
  - Server:
    - Passive (slave)
    - Waits for requests
    - Upon receipt of requests, processes them and then serves replies
  - Client:
    - Active (master)
    - Sends requests
    - Waits for and receives server replies

Client Psuedocode

```
get the server's address
form a working address that can be used to talk over
Internet.

connect to the server

while (!done) do:
   wait until there's either information from the server,
or
         from the user.

   If (information from server) do
      parse information
      show to user, update local state information, etc.

   else {we've got a user command}
      parse command
      send to server, or deal with locally.
done
```

Server pseudocode

```
bind a port on the computer, so Clients will be able
to    connect
forever do:
   listen on the port for connection requests.
   accept an incoming connection request
   if (this is an authorized Client)
      while (connection still alive) do:
         receive request from client
         handle request
         send results of request, or error messages
      done
   else
      abort the connection
done
```

Summary

- Discussed sockets concepts including IPC
- Explained socket creation including domains, types, protocols, and addresses
- Discussed socket functions including send/receive, close, and IO relation
- Explained binding sockets to addresses, listening for connections, making connections, and transferring data
- Discussed client/server operation using sockets

Questions?