Programming with Berkeley Sockets

Written by Daniel Stutzbach
Modified by Daya Wimalasuriya
for CIS 432/532

Useful References:
- man pages
- *UNIX Network Programming* by W. Richard Stevens
- *TCP/IP Illustrated* by W. Richard Stevens

Berkeley Sockets

- Standard C functions for network operations
- Standardized in:
  - POSIX 1003.1
  - SUSv2
- Other languages call the C functions
- Present in all major Unix operating systems
  - GNU/Linux, Solaris, BSD, Mac OS X
- Microsoft Windows uses a variant
Overview

- A word about byte order
- Sockets
- How to Create and Manipulate Them
- Datagram versus Stream Protocols
- Datagram Protocol Functions
- Multiplexing with Select

Sockets

- A socket is a 5-tuple:
  - (Protocol, Local Address, Local Port, Remote Address, Remote Port)
- For example:
  - (TCP, salix.cs.uoregon.edu, 32905, www.google.com, 80)
- Socket data structures are stored in the kernel.
- Applications manipulate them with function calls:
  - socket(): Allocate a socket and return a descriptor
  - bind(): Set the local address and port
  - connect(): Set the remote address and port
Byte Order

- Little Endian
  - Lowest-order byte has the lowest address
  - Bytes are ordered in the same way as bits
  - Used on x86 platform
- Big Endian
  - Highest-order byte has the lowest address
  - Used on PowerPC, 68k, Sparc
  - Official network byte order for Internet protocols

Special macros exist for converting between network byte order and host byte order:
- htonl(): convert a 32-bit integer from host to network byte order
- htons(): convert a 16-bit integer from host to network byte order
- ntohl(): convert a 32-bit integer from network to host byte order
- ntohs(): convert a 16-bit integer from network to host byte order
Header Files

- `<sys/socket.h>`: Core BSD socket functions and data structures.
- `<netinet/in.h>`: AF_INET and AF_INET6 address families.
- `<sys/un.h>`: AF_UNIX address family. Used for local communication between programs running on the same computer.
- `<arpa/inet.h>`: Functions for manipulating numeric IP addresses.
- `<netdb.h>`: Functions for translating protocol names and host names into numeric addresses.

Using socket()

```c
int socket(int domain, int type, int protocol);
```

- Creates a socket and returns a descriptor

- Parameters:
  - `domain`: the type of addressing used
    - `PF_INET`, `PF_UNIX`, `PF_INET6`
  - `type`: the type of communication
    - `SOCK_STREAM`, `SOCK_DGRAM`
  - `protocol`: the exact protocol to use, usually zero

- Examples:
  - `s = socket(AF_INET, SOCK_STREAM, 0);`
  - `s = socket(AF_UNIX, SOCK_DGRAM, 0);`
The Socket Address Structure

- Important data structure, used by connect(), bind(), and others
  
  ```c
  struct sockaddr {
      sa_family_t   sa_family;
      char          sa_data[14];
  };
  ```

- Different types are used based on address type
  
  ```c
  struct sockaddr_in {
      sa_family_t   sin_family;
      in_port_t     sin_port;
      struct in_addr sin_addr;
      char          sin_unused; }
  ```

- Another example: Unix domain sockets
  - Host address is implied
  - Port is a path in the filesystem
  
  ```c
  struct sockaddr_un {
      sa_family_t   sun_family;
      char          sun_path[108];
  }
  ```
Using connect()

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

- Used by clients to specify a server
- Parameters:
  - sockfd : A socket previously returned by socket()
  - serv_addr : A socket address structure, specifying the server
  - addrlen : The size of the socket address structure

Connecting to a Web Server

int err;
struct sockaddr_in server;
server.sin_family = AF_INET;
server.sin_port = htons(80);
server.sin_addr.s_addr = htonl (0x7f000001);
err = connect (s, &sockaddr, sizeof sockaddr);
if (err < 0)
    perror ("Error calling connect()");
Connecting to a Unix Domain Server

```c
int err;
struct sockaddr_un server;
server.sun_family = AF_UNIX;
strcpy (server.sun_path, "/tmp/server-sock");
err = connect (s, &server, sizeof server);
if (err < 0)
    perror ("Error calling connect()");
```

Using bind

```c
int bind (int sockfd, const struct sockaddr *my_addr,
          socklen_t addrlen);
```

- Sets the local address and port of the socket
- Not needed for the client side when using the Internet address family
- Parameters are the same as connect()
Connectionless Datagram Protocols

- UDP is the canonical example
- This is what you will use in your first assignment
- Transmitting is done using `send()` or `sendto()`
- Receiving is done using `recv()` or `recvfrom()`

Connection-Oriented Stream Protocols

- TCP is the canonical example
- Transmitting is done with `write()`
- Receiving is done with `read()`
- Additional functions for servers:
  - `listen()`: configures a socket to wait for new connections
  - `accept()`: returns a new socket for a new connection
send() and sendto()

ssize_t send (int s, const void *msg, size_t len, int flags);
ssize_t sendto (int s, const void *msg, size_t len, int flags,
               const struct sockaddr *to, socklen_t tolen);

- These functions are used to send datagrams.
- sendto() is like send() mixed with connect()

Parameters:
- s: A bound socket. For send(), the socket must also be connected.
- msg: The data for the datagram
- len: The size of the data
- flags: Usually zero

recv() and recvfrom()

ssize_t recv(int s, void *buf, size_t len, int flags);
ssize_t recvfrom (int s, void *buf, size_t len, int flags,
                  struct sockaddr *from, socklen_t *fromlen);

- These functions are used to receive datagrams.

Parameters:
- s: A bound socket. For recv(), it should also be connected.
- buf: Memory to store the datagram
- len: The size of buf.
- flags: Usually zero
- from: Memory for a socket address structure
- fromlen: Before calling, the size of from. After calling, the size of from actually used.
Multiplexing with select()

```c
int select (int n, fd_set *readfds, fd_set *writefds,
            fd_set exceptfds, struct timeval *timeout);
```

- select() is used to block on multiple sockets at once.
- It can also be used to block on input from the user.

- Parameters:
  - `n`: The largest file descriptor in use, plus one
  - `readfds`: Wake when any of these descriptors are readable
  - `writefds`: Wake when any of these descriptors are writable
  - `exceptfds`: Wake when any of these descriptors have exceptions, often NULL
  - `timeout`: The maximum duration to block

Defined in `<sys/types.h>

- `fd_set` structures are manipulated with special macros.

```c
FD_CLR (int fd, fd_set *set);
FD_ISSET (int fd, fd_set *set);
FD_SET (int fd, fd_set *set);
FD_ZERO (fd_set *set);
```