CS 5651 Computer Networks

Application Layer Socket Programming

Pete Willemsen

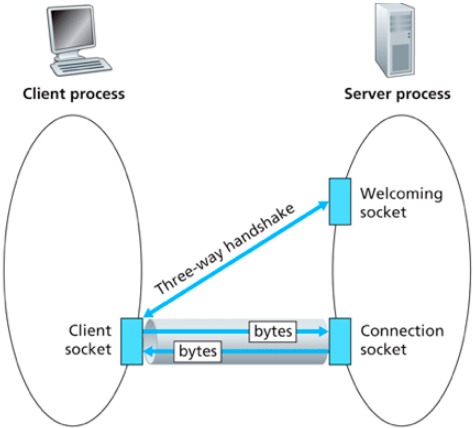
University of Minnesota Duluth

September 24 2012

Socket Programming

What happens when you create and open sockets?

- Create a socket - *socket* call
- Bind sockets to addresses and ports - *bind* call
- Wait for a connection - *listen* and *accept* calls
- Connect to sockets - *connect* call
- Transmit the data - *send* and *recv* calls
- Close the socket - *close* call



Walk through a TCP Server and Client system.

- Both programs will have to create sockets
- One will act as server (remain on, primarily receive connections and respond)
- Other will act as client (connect to server, send and receive data)

Ideal Socket Program

What would an ideal socket program for a server look like? Perhaps, like this:

```
int main(int argc, char *argv[])
{
    processArguments(argc, argv);

    // Port number on which server listens
    ServerSocket ss(args.portNumber);
    while (!done) {
        ss.welcomeClient();
        ss.processClientRequest();
        ss.disconnectClient();
    }
    ss.close();
}
```

Ideal Socket Program

How about for the client? Perhaps, like this:

```
int main(int argc, char *argv[])
{
    processArguments(argc, argv);

    // Hostname and port of server!
    ClientSocket cs(args.hostname, args.portNumber);

    cs.sendRequest(``GET / HTTP/1.1'');
    cs.receiveAndProcessResponse();

    cs.close();
}
```

Now, let's see what it's really like!

Important Socket API Functions - Opening and Closing

- socket - Creates a socket for communication.

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

Important Socket API Functions - Opening and Closing

- socket - Creates a socket for communication.

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

- Example Usage:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
...
  int socket_fd = socket(AF_INET, SOCK_STREAM, 0);
  if (socket_fd < 0)
  {
      // throw exception or generate error
  }
```

Important Socket API Functions - Opening and Closing

- socket - Creates a socket for communication.

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

- Example Usage:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
...
  int socket_fd = socket(AF_INET, SOCK_STREAM, 0);
  if (socket_fd < 0)
  {
     // throw exception or generate error
  }
```

- close - Closes the file descriptor associated with socket.

```
int close(int fd);
```

- Example Usage:

```
#include <unistd.h>
...
   close(socket_fd);
```

Important Socket API Functions - Binding

- bind - Associates an address with a socket. Bind allows you to set the port and local addresses of the socket using the socket address structures.

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

Important Socket API Functions - Binding

- bind - Associates an address with a socket. Bind allows you to set the port and local addresses of the socket using the socket address structures.

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

- Example Usage:

```
struct sockaddr_in servaddr;
memset(&servaddr, 0, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port = htons(9034);

if (bind(socket_fd, (const sockaddr*)&servaddr, sizeof(servaddr)) < 0)
  {
    // throw exception or generate error
  }
```

- Notice that *sin_port* is set to *htons(9034)*; you can set the port to be what you want, but it must be in network byte order! This server accepts connections on port 9034.
- Note also that *s_addr* is set to *INADDR_ANY*. What is that?

Important Socket API Functions - Binding

- bind - Associates an address with a socket. Bind allows you to set the port and local addresses of the socket using the socket address structures.

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

- Example Usage:

```
struct sockaddr_in servaddr;
memset(&servaddr, 0, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port = htons(9034);

if (bind(socket_fd, (const sockaddr*)&servaddr, sizeof(servaddr)) < 0)
  {
    // throw exception or generate error
  }
```

- Notice that *sin_port* is set to *htons(9034)*; you can set the port to be what you want, but it must be in network byte order! This server accepts connections on port 9034.
- Note also that *s_addr* is set to *INADDR_ANY*. What is that?
- *INADDR_ANY* allows this socket to accept connections on any of the interfaces available on the machine (*e.g.*127.0.0.1, 131.212.41.X, 192.168.0.2).

Port Numbers!

What port numbers can you use? Recall that:

- Port numbers are 16-bit integer values, so range from 0 to 65535.
- However, Port numbers 0 - 1023 are reserved as the well-known ports!
- See IANA.org for listings of the well-known ports.
- Also, look at /etc/services on OS X or Linux systems.

You should use ports in the range 1024 - 65535!

Using Well Known Ports

Note: You can use the well-known ports in your application. However, you will need to run your application with special permissions (sudo or root or some variant). For assignments, do *not* use the well-known reserved port space.

Important Socket API Functions - Listening

- listen - sets the socket state as ready for incoming connections and provides information about the maximum size for queued pending connections.

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

- Example Usage:

```
if (listen(socket_fd, 64) < 0)
  {
    // throw exception or generate error
  }
```

- Listen prepares the operating system to receive connections on this socket.
- Notice the second argument. It's called the *backlog* and relates to the size of the queue associated with this socket. It must be greater than 0, with a maximum of 128 (or the value in /proc/sys/net/core/somaxconn, for Linux that is).

Important Socket API Functions - Accepting

- accept - Accept a connection on a socket. This function *blocks* and waits for a connection to be initiated by a client.

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

- Example Usage:

```
int clientSocket_fd;
struct sockaddr_in clientaddr;
clientSocket_fd = accept(socket_fd, (sockaddr*)&clientaddr, &length);
if (clientSocket_fd == 0)
{
   // throw exception or generate error
}
```

- Accept blocks if no connections are made. Once a connection is made the process is awakened and processes the accept call.
- Make sure you understand what it means for a function to *block*!

Important Socket API Functions - Sending and Receiving

- send - sends a message into a socket.
- recv - receives a message from a socket.

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

- Example Usage:

```
std::string message = ''MESSAGE'';
int sentBytes = 0;
sentBytes = send(socket_fd, message.c_str(), message.length(), 0);
if (sentBytes == -1)
   // throw error or generate error
...
char recvBuffer[1024];
int recvBytes = 0;
recvBytes = recv(socket_fd, recvBuffer, 1024, 0);
```

- Note that *recv* blocks if nothing has been sent to the socket!
- These examples use nice formatted output and input, but send and recv are very basic functions that handle bytes. In other words, the data may be binary too, not just ASCII as in these examples.

Putting it Together - TCP Server

```
int listen_fd = socket(AF_INET, SOCK_STREAM, 0);

struct sockaddr_in servaddr;
memset(&servaddr, 0, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port = htons(9023);
bind(listen_fd, (const sockaddr*)&servaddr, sizeof(servaddr));

listen(listen_fd, 64);

int clientSocket_fd;
struct sockaddr_in clientaddr;
socklen_t length = sizeof(clientaddr);
clientSocket_fd = accept(listen_fd, (sockaddr*)&clientaddr, &length);

// create an output stream, write the time to it and send it to the client
char CR = '\r';
char LF = '\n';
std::ostringstream outputStream("");
outputStream << ``MESSAGE `` << time(NULL) << CR << LF;

send(clientSocket_fd, outputStream.str().c_str(), outputStream.str().length(), 0);

close(clientSocket_fd);
close(listen_fd);
```

Important Socket API Functions - Connecting

- connect - connects two sockets together based on the address of the recipient host.

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

- Example Usage:

```
if (connect(socket_fd, (const sockaddr*)&servaddr, sizeof(servaddr)) < 0)
  {
    // throw exception
  }
```

- Actually causes the TCP three-way-handshake to initiate!
- *servaddr* MUST contain the IP Address and Port Number to which you want to connect!
- Will cause the *accept* function on the server side to unblock if it was blocked.

Putting it Together - TCP Client

```
int socket_fd = socket(AF_INET, SOCK_STREAM, 0);

struct sockaddr_in servaddr;
memset(&servaddr, 0, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(9023);

std::string ipaddr = ``127.0.0.1'';
inet_pton(AF_INET, ipaddr.c_str(), &servaddr.sin_addr);

connect(socket_fd, (const sockaddr*)&servaddr, sizeof(servaddr));

char buffer[1024];
int recvBytes = recv(socket_fd, buffer, 1024, 0);

std::string messageType;
time_t timeFromServer;
char CR, LF;
std::istringstream inputStream(buffer);

inputStream >> messageType >> timeFromServer;
CR = inputStream.get();
LF = inputStream.get();

close(socket_fd);
```

TCP Server and TCP Client

Some questions...

- Why didn't the client use the bind call?

TCP Server and TCP Client

Some questions...

- Why didn't the client use the bind call?
  - Client doesn't care about the address that is bound to the socket
  - Operating system will set an address structure up with a default port for the client
  - Does this make sense as to why this would occur?

Important Socket API Functions - Sending Datagrams

- sendto - sends a message into a socket.

```
ssize_t sendto(int sockfd, const void *buf, size_t len, int flags,
               const struct sockaddr *dest_addr, socklen_t addrlen);
```

- Example Usage:

```
struct sockaddr_in sendToAddr;
memset(&sendToAddr, 0, sizeof(sendToAddr));
sendToAddr.sin_family = AF_INET;
sendToAddr.sin_port = htons( 12003 );

std::string ipaddr = ``127.0.0.1'';
inet_pton(AF_INET, ipaddr.c_str(), &sendToaddr.sin_addr);

std::string message = ``MESSAGE'';
int sentBytes = sendto(socket_fd, message.str().c_str(), message.str().length(), 0,
                       (const struct sockaddr*)&sendToAddr, sizeof(sendToAddr));
if (sentBytes == -1)
   // throw error or generate error
```

Important Socket API Functions - Receiving Datagrams

- recvfrom - receives a message from a socket whether it is connected or not.

```
ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,
                 struct sockaddr *src_addr, socklen_t *addrlen);
```

- Example Usage:

```
struct sockaddr_in sentFromAddr;
socklen_t sentFromAddressLen = sizeof(sentFromAddr);
memset(&sentFromAddr, 0, sizeof(sentFromAddr));

char recvBuffer[1024];
int recvBytes = recvfrom(socket_fd, recvBuffer, 1024, 0,
                         (struct sockaddr*)&sentFromAddr, &sentFromAddressLength);
if (recvBytes == -1)
   // throw error or generate error
```

- Note that *recvfrom* blocks if nothing has been sent to the socket!

Putting it Together - UDP Sender

```
int socket_fd = socket(AF_INET, SOCK_DGRAM, 0);

struct sockaddr_in sendto_addr;
memset(&sendto_addr, 0, sizeof(sendto_addr));
sendto_addr.sin_family = AF_INET;
sendto_addr.sin_port = htons(12003);

std::string ipaddr = ''127.0.0.1'';
inet_pton(AF_INET, ipaddr.c_str(), &sendto_addr.sin_addr);

long frameCounter = 1;
std::ostringstream sendToStream(");
sendToStream << frameCounter;

sendto(socket_fd, sendToStream.str().c_str(), sendToStream.str().length(), 0,
       (const struct sockaddr *)&sendto_addr, sizeof(sendto_addr));

close(socket_fd);
```

Putting it Together - UDP Receiver

```
int socket_fd = socket(AF_INET, SOCK_DGRAM, 0);

struct sockaddr_in recv_addr;
memset(&recv_addr, 0, sizeof(recv_addr));
recv_addr.sin_family = AF_INET;
recv_addr.sin_port = htons(12003);
recv_addr.sin_addr.s_addr = htonl(INADDR_ANY);

bind(socket_fd, (const sockaddr *)&recv_addr, sizeof(recv_addr));

socklen_t sender_addr_length;
struct sockaddr_in sender_addr;
memset(&sender_addr, 0, sizeof(sender_addr));

char data[1024];
recvfrom(socket_fd, data, 1024, 0,
         (struct sockaddr*)&sender_addr, &sender_addr_length);

long frameCounter = -1;
std::istringstream inputStream(data);
inputStream >> frameCounter;
std::cout << ``Received: `` << frameCounter << std::endl;

close(socket_fd);
```

Important Socket API Functions - Socket Options

- setsockopt - sets various options for how sockets behave

```
int setsockopt(int sockfd, int level, int optname,
               const void *optval, socklen_t optlen);
```

- Example Usage:

```
int on = 1;
setsockopt(socket_fd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on));
```

- Note that this option sets the socket so that addresses can be reused immediately
- Socket options are described in the socket man page: *man 7 socket*