ECE 344: Operating Systems

Lecture 26

Sockets
1.0.1

Jon Eyolfson

November 17, 2022

Sockets are Another Form of IPC

We've seen pipes and signals
We also talked about shared memory

These forms of IPC assume that the processes are on the same physical machine

Sockets enable IPC between physical machines, typically over the network

Servers Follow 4 Steps to Use Sockets

These are all system calls, and have the usual C wrappers:

1. `socket`
   Create the socket
2. `bind`
   Attach the socket to some location (a file, IP:port, etc.)
3. `listen`
   Indicate you're accepting connections, and set the queue limit
4. `accept`
   Return the next incoming connection for you to handle

Clients Follow 2 Steps to Use Sockets

Clients have a much easier time, they use one socket per connection

1. `socket`
     Create the socket
2. `connect`
     Connect to some location, the socket can now send/receive data

The socket System Call Sets the Protocol and Type of Socket

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

domain is the general protocol, further specified with protocol (mostly unused)
- AF_UNIX is for local communication (on the same physical machine)
- AF_INET is for IPv4 protocol using your network interface
- AF_INET6 is for IPv6 protocol using your network interface

type is (usually) one of two options: stream or datagram sockets

Stream Sockets Use TCP

All data sent by a client appears in the same order on the server

Forms a persistent connection between client and server

Reliable, but may be slow

Datagram Sockets Use UDP

Sends messages between the client and server

No persistent connection between client and server

Fast but messages may be reordered, or dropped

The bind System Call Sets a Socket to an Address

```
int bind(int socket, const struct sockaddr *address,
         socklen_t address_len);
```

socket is the file descriptor returned from the socket system call

There's different sockaddr structures for different protocols
    struct sockaddr_un for local communcation (just a path)
    struct sockaddr_in for IPv4, a IPv4 address (e.g. 8.8.8.8)
    struct sockaddr_in6 for IPv6, a IPv6 address (e.g. 2001:4860:4860::8888)

The `listen` System Call Sets Queue Limits for Incoming Connections

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

socket is still the file descriptor returned from the socket system call

backlog is the limit of the outstanding (not accepted) connections
    The kernel manages this queue, and if full will not allow new connections

We'll set this to 0 to use the default kernel queue size

The accept System Call Blocks Until There's a Connection

```
int accept(int socket, struct sockaddr *restrict address,
           socklen_t *restrict address_len);
```

socket is *still* the file descriptor returned from the socket system call

address and address_len are locations to write the connecting address
    Acts as an optional return value, set both to NULL to ignore

This returns a new file descriptor, we can read or write to as usual

The connect System Call Allows a Client to Connect to an Address

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

sockfd is the file descriptor returned by the socket system call
    The client would need to be using the same protocol and type as the server

addr and addrlen is the address to connect to, exactly like bind

If this call succeeds then sockfd is may be used as a normal file descriptor

Our Example Server Sends "Hello there!" to Every Client and Disconnects

Please see examples/lecture-26 in your examples repository
  Relevant source files: client.c and server.c

We use a local socket just for demonstration, but you could use IPv4 or IPv6
  We use example.sock in the current directory as our socket address

Our server uses signals to clean up and terminate from our infinite accept loop

Instead of `read`/`write` There's Also `send`/`recv` System Calls

These system calls are basically the same thing, except they have `flags`

Some examples are:
    `MSG_OOB` — Send/receive out-of-band data
    `MSG_PEEK` — Look at data without reading
    `MSG_DONTROUTE` — Send data without routing packets

Except for maybe `MSG_PEEK`, you do not need to know these

`sendto`/`recvfrom` take an additional address
    The kernel ignores the address for stream sockets (there's a connection)

You Perform Networking Through Sockets

Sockets are IPC across physical machines, the basics are:

- Sockets require an address (e.g. local and IPv4/IPv6)
- There are two types of sockets: stream and datagram
- Servers need to bind to an address, listen, and accept connections
- Clients need to connect to an address