CSE 124: CLIENT-SIDE PROGRAMMING

George Porter
Oct 9, 2017
ATRIBUTION

• These slides are released under an Attribution-NonCommercial-ShareAlike 3.0 Unported (CC BY-NC-SA 3.0) Creative Commons license

• These slides incorporate material from:
  • Practical TCP/IP Sockets in C, 2nd ed., by Donahoo and Calvert
  • Computer Networks: A Systems Approach, 5e, by Peterson and Davie
Outline

1. Client socket API
2. Demo: writing a simple client
• Socket Interface was originally provided by the Berkeley distribution of Unix
• Now supported in virtually all operating systems

• Each protocol provides a certain set of services, and the API provides a syntax by which those services can be invoked in this particular OS
WHAT IS A SOCKET?

- What is a socket?
  - The point where a local application process attaches to the network
  - An interface between an application and the network
  - An application creates the socket
- The interface defines operations for
  - Creating a socket
  - Attaching a socket to the network
  - Sending and receiving messages through the socket
  - Closing the socket
TRANSPORT PROTOCOLS

- Add services on top of IP
- User Datagram Protocol (UDP)
  - Data checksum
  - Best-effort
- Transmission Control Protocol (TCP)
  - Data checksum
  - Reliable byte-stream delivery
  - Flow control
    - Prevents receiver from being overloaded
  - Congestion control
    - Prevents the network from being overloaded
- IP addresses identify hosts
- Host has many applications
- Ports (16-bit identifier) identify a process
CLIENT OVERVIEW

Steps

1. Create network socket
2. Connect to the server
3. Send a message
4. Receive the response
5. Close the socket

Socket API used

1. socket()
2. connect()
   - struct sockaddr_in
3. send()
4. recv()
5. close()
SOCKET FAMILIES AND TYPES

• **Socket Family**
  - **PF_INET** denotes the Internet family
  - **PF_UNIX** denotes the Unix pipe facility
  - **PF_PACKET** denotes direct access to the network interface (i.e., it bypasses the TCP/IP protocol stack)

• **Socket Type**
  - **SOCK_STREAM** is used to denote a byte stream
  - **SOCK_DGRAM** is an alternative that denotes a message oriented service, such as that provided by UDP
• int sockfd = socket(address_family, type, protocol);

• The socket number returned is the socket descriptor for the newly created socket

• int sockfd = socket (PF_INET, SOCK_STREAM, 0);
• int sockfd = socket (PF_INET, SOCK_DGRAM, 0);

• The combo of PF_INET and SOCK_STREAM implies TCP
### Generic

```c
struct sockaddr {
    unsigned short sa_family; /* Address family (e.g., AF_INET) */
    char sa_data[14];          /* Protocol-specific address information */
};
```

### IP Specific

```c
struct sockaddr_in {
    unsigned short sin_family; /* Internet protocol (AF_INET) */
    unsigned short sin_port;   /* Port (16-bits) */
    struct in_addr sin_addr;   /* Internet address (32-bits) */
    char sin_zero[8];          /* Not used */
};
```

```c
struct in_addr {
    unsigned long s_addr;    /* Internet address (32-bits) */
};
```

<table>
<thead>
<tr>
<th>sockaddr</th>
<th>Family</th>
<th>Blob</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2 bytes</td>
<td>4 bytes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sockaddr_in</th>
<th>Family</th>
<th>Port</th>
<th>Internet address</th>
<th>Not used</th>
</tr>
</thead>
</table>

**Table:**

- **sockaddr**
  - Family: 2 bytes
  - Blob: 4 bytes
  - 8 bytes

- **sockaddr_in**
  - Family: 2 bytes
  - Port: 2 bytes
  - Internet address: 4 bytes
  - Not used: 8 bytes
HANDY HELPER FUNCTIONS FOR ADDRESSES

- Converting IP addr to/from strings
- Printable string to binary:
  - int inet_pton();
- Binary to printable string:
  - const char * inet_ntop();
- Making the byte order consistent
  - uint16_t htons(uint16_t hostshort);
  - uint16_t ntohs(uint16_t netshort);
CONNECTING SOCKETS

- `int connect(int socket, const struct sockaddr *address, socklen_t address_len);`
  - `socket` is the descriptor
  - `address` describes the destination address
  - `len` is the size of address
  - **Blocking call**: waits until a connection is established
  - **Q**: What kinds of errors might occur here?
SSIZE_T SEND

- ssize_t send(socket, buf, len, flags)
  - We’re using blocking semantics of send
  - Always check that the right number of bytes were sent
  - Returns the number of bytes that were copied to the operating system kernel for transmission
RECEIVING RESPONSES

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

- **Note:**
  - `recv()` receives at least one bytes from the socket
  - *It does not receive the same number of bytes* that were sent via ‘send’ (we’ll see why in Chapter 7)
  - Returns 0 when the client has closed the socket

- **What does this mean?**
  - You have to keep reading from the socket until you’ve received all the bytes you need
1. Client socket API
2. Demo: writing a simple client
UC San Diego