CSC209: Software Tools and Systems Programming

Week 10: Sockets [1]

Kianoosh Abbasi

[1] Slides are mostly taken from Andi Bergen's in summer 2021.

Human Communication

Human communication is governed by rules: vocabulary, sentence structure, spelling, grammar. . .

How would you like it if a l l th elect ur e sli des we r e

fo r ma ttt ed like thiiiiiis ?!.$

Computer Communication

Computer communication is also governed by rules:

- *Encoding* of information, e.g., text, signed/unsigned integers, floating point
- *Ordering* of bytes, e.g., big endian, little endian
- *Message sequencing*, e.g., first send username, then send password
- *Message start and end boundaries*, e.g., CRLF (`\r\n`) to terminate messages

Transport Protocols

Two widespread models of *transport protocols* for computer communication over a network:

1. Connectionless: Exemplified by `UDP` protocol
2. Connection-oriented: Exemplified by `TCP` protocol

Protocols are a set of *rules*. Both TCP and UDP protocols are implemented by the *operating system*.

CSC209 vs. CSC358

- In CSC209, we learn what is necessary to *use* TCP to communicate over a network
  - No UDP, due to time constraints
- In CSC358, you will learn *how TCP and UDP work*

notes: Just like how CSC209 involves *using* system calls, and CSC369 involves how system calls work "under the hood"

UDP

- UDP is used for sending a *datagram* from one machine to another
- A datagram is a self-contained message with a beginning and end
- The OS sends the datagram, but doesn't follow up to make sure that it got delivered

TCP

- TCP is used to establish a *socket* (similar to a pipe) to communicate between two processes
  - Processes may be on the same computer, or two different computers connected by a network
- The socket is created using a system call
- The process sending the data writes a sequence of bytes to the socket
- The OS guarantees that the bytes will be delivered over the network, in the correct order, to the receiving process

UDP vs. TCP

- Comparing UDP and TCP is like comparing SMS and WhatsApp
- If you send an SMS to your friend, you have no way of knowing if they received your message
  - Perhaps they may reply back to you confirming that they received your message
- If you send a message over WhatsApp, **the app itself tells you whether or not the message was successfully delivered**

We were planning to tell a UDP joke on this slide...

But we weren't sure if you would get it.

Internet Protocol (IP) Addresses

- An *IP address* identifies a specific host (computer) on a specific network.
- IPv4 addressing (most widespread) identifies hosts by four decimal 8-bit integers separated by dots, e.g., `128.100.3.30`
- IPv6 addressing (slowly being adopted) identifies hosts by eight groups of four hexadecimal digits, separated by colons, e.g., `fe80:1234:0432:a2d8:61ff:fe8b:8924:c23f`

TCP and UDP Ports

- An IP address only identifies a host, but not the program running on the host
- To communicate with a specific program on a host, you must specify a *port number* between 0 and 65535
- For Assignment 3, your client and server programs must use the same port number; otherwise, they cannot communicate

Port Number Conventions

- Ports in range 0–1023 are *well-known* or *reserved* (e.g., 22 for SSH, 80 for HTTP, 443 for HTTPS)
- Ports in range 1024–49151 are *registered* (e.g., 3724 for World of Warcraft)
- Ports in range 49152–65535 are *dynamic*
  - These are the ones you should typically pick, to avoid conflict

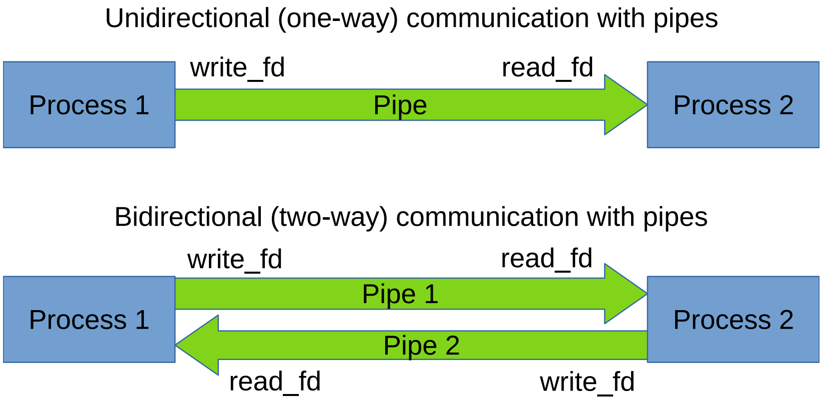
See IANA for list of port assignments

`localhost`

- You may test your client and server programs by running both on the same computer
- `127.0.0.1` is the "loopback" IP address, for when your program needs to communicate with another program on the same computer
- `localhost` is a *hostname* "aliased" to `127.0.0.1`
- To test your client and server on different PCs, obtain the PC's IP address by running `ifconfig`
  - Lab PCs have IP address in the range `142.1.X.Y`

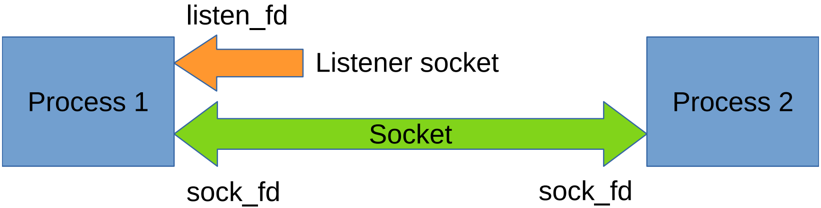

There's no place
like 127.0.0.1

Pipes: Unidirectional vs. Bidirectional Communication



Sockets

Bidirectional (two-way) communication with sockets



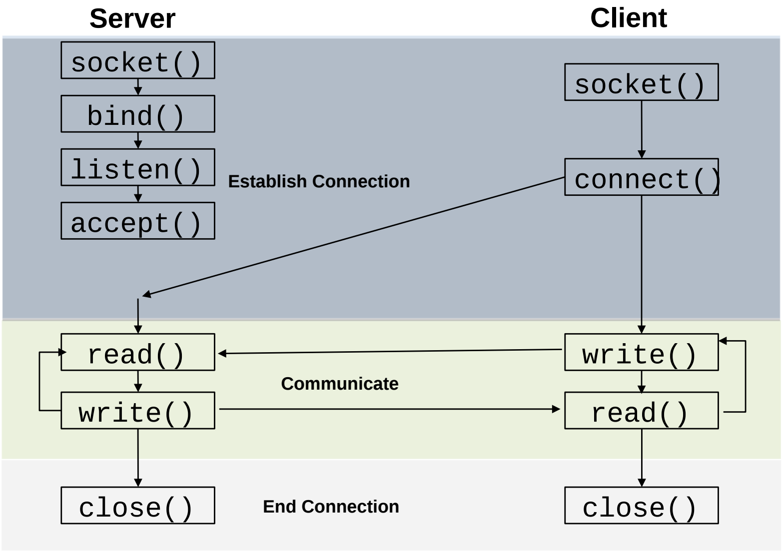
- A server must have a *listener socket* to accept new connections
- A separate socket is created to communicate with each client

System Calls for Setting up a Server

1. `socket`: Creates a socket.
2. `bind`: Assigns an address and port to the socket. (must assign an IP address that actually belongs to your systems).
3. `listen`: Establish a queue for incoming connections.
4. `accept`: Accept queued incoming connection and create a new socket.
5. `read`/`write`: Receive/send data on socket.

System Calls for Setting up a Client

1. `socket`: Creates a socket
2. `connect`: Connects to a remote server using an IP address and port.
3. `read`/`write`: Receive/send data on socket.


Server
Client
socket()
bind()
listen()
accept()
socket()
connect()
Establish Connection
read()
write()
write()
read()
Communicate
close()
close()
End Connection

IP Address Struct

- System calls expect the IP address to be passed in as an `in_addr` struct
- There are system calls to help you convert back and forth between structs and human-readable strings, e.g.
  - Use `inet_aton` to convert IP address from a string of the form `a.b.c.d` into an `in_addr` struct
  - Use `inet_ntoa` to convert `in_addr` struct into string of the form `a.b.c.d`

Endianness

- By convention, all data being sent over the network must first be converted into *big endian*, known as *network byte order*
- The endianness of the host is referred to as *host byte order*
- `htonl()`, `htons()`, `ntohl()`, and `ntohs()` are used for converting between *host byte order* and *network byte order*
  - See man pages for usage
  - Even port numbers must be converted
  - ASCII text does not require conversion (why?)

Defining Message Boundaries

- Assume that a sender sends the sequence of bytes "Hello world" to a receiver over a TCP socket
- TCP guarantees that the receiver will receive the entire sequence, eventually
- But it's possible that when the receiver calls `read()` on the socket:
  - The entire message wasn't received yet
  - The `read()` call was interrupted (e.g., see `EINTR` in `man 2 read`)

Question

How do we know when we have received a complete message, and not a partial message?

Answer

We define a byte sequence that indicates the end of a message. In text-based protocols, the most common convention is to signify an end-of-message with a CRLF (carriage-return + newline, or `\r\n`) sequence. Actual message content must not contain any instances of this sequence.

Question

Are there alternative techniques for determining that we have received a complete message?

Answer

Yes. Two common techniques:

1. Define a fixed-length message format (i.e., every message must be identical in length).
2. Define a fixed-length "header" that contains an integer representing the length of the remainder of the message.

Question

Is it possible for the server (or client) to call `read()` on a socket and receive more than one message?

Answer

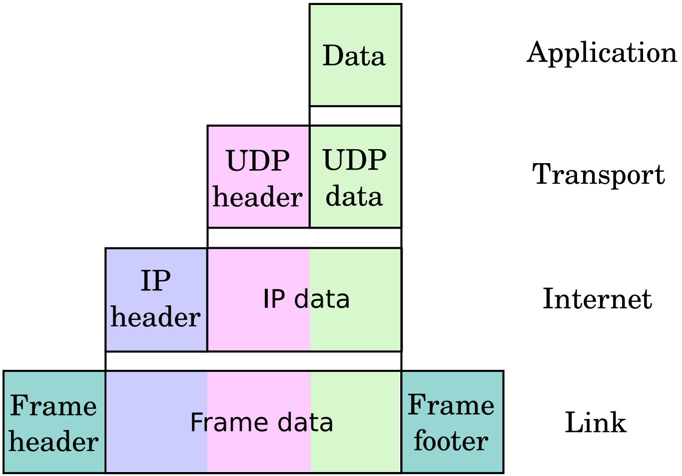
Yes. This might happen if the sender is sending the messages faster than you are reading them. In this case, you must save the messages in a *buffer* and handle them one at a time.

Buffering

- Buffering is an extremely common technique, especially in networking.
- The Operating System also does its own buffering
- What happens if your PC receives data from the network, but your program isn't ready to call `read()` yet, because it is busy doing something else?
  - Answer: The OS saves it in a buffer, until your program calls `read()`

Extra Slides

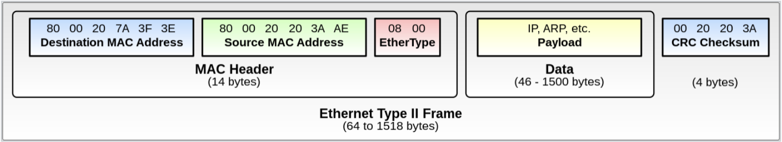
The TCP/IP Model



Link Layer

Link-layer protocols deal with how your device physically transmits the data, e.g., wirelessly, or over a copper or fibre-optic cable

The Ethernet header



The most common Ethernet Frame format, type II

Internet Layer

Internet protocols such as IP, RIP, and OSPF govern how your data gets transferred from one Internet Service Provider (ISP; e.g., Bell, Rogers) to another

The IPv4 Header

IPv4 header format

<table>
<tr><th>Offsets</th><th>Octet</th><th colspan="8">0</th><th colspan="8">1</th><th colspan="8">2</th><th colspan="8">3</th></tr>
<tr><th>Octet</th><th>Bit</th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th><th>16</th><th>17</th><th>18</th><th>19</th><th>20</th><th>21</th><th>22</th><th>23</th><th>24</th><th>25</th><th>26</th><th>27</th><th>28</th><th>29</th><th>30</th><th>31</th></tr>
<tr><th>0</th><th>0</th><td colspan="4">Version</td><td colspan="4">IHL</td><td colspan="6">DSCP</td><td colspan="2">ECN</td><td colspan="16">Total Length</td></tr>
<tr><th>4</th><th>32</th><td colspan="16">Identification</td><td colspan="3">Flags</td><td colspan="13">Fragment Offset</td></tr>
<tr><th>8</th><th>64</th><td colspan="8">Time To Live</td><td colspan="8">Protocol</td><td colspan="16">Header Checksum</td></tr>
<tr><th>12</th><th>96</th><td colspan="32">Source IP Address</td></tr>
<tr><th>16</th><th>128</th><td colspan="32">Destination IP Address</td></tr>
<tr><th>20</th><th>160</th><td colspan="32" rowspan="3">Options (if IHL > 5)</td></tr>
<tr><th>⋮</th><th>⋮</th></tr>
<tr><th>56</th><th>448</th></tr>
</table>

The IPv6 Header

<table>
<caption>Fixed header format</caption>
<tr><th>Offsets</th><th>Octet</th><th colspan="8">0</th><th colspan="8">1</th><th colspan="8">2</th><th colspan="8">3</th></tr>
<tr><th>Octet</th><th>Bit</th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th><th>16</th><th>17</th><th>18</th><th>19</th><th>20</th><th>21</th><th>22</th><th>23</th><th>24</th><th>25</th><th>26</th><th>27</th><th>28</th><th>29</th><th>30</th><th>31</th></tr>
<tr><th>0</th><th>0</th><td colspan="4">Version</td><td colspan="8">Traffic class</td><td colspan="20">Flow label</td></tr>
<tr><th>4</th><th>32</th><td colspan="16">Payload length</td><td colspan="8">Next header</td><td colspan="8">Hop limit</td></tr>
<tr><th>8</th><th>64</th><td colspan="32" rowspan="4">Source address</td></tr>
<tr><th>12</th><th>96</th></tr>
<tr><th>16</th><th>128</th></tr>
<tr><th>20</th><th>160</th></tr>
<tr><th>24</th><th>192</th><td colspan="32" rowspan="4">Destination address</td></tr>
<tr><th>28</th><th>224</th></tr>
<tr><th>32</th><th>256</th></tr>
<tr><th>36</th><th>288</th></tr>
</table>

Transport Layer

Transport protocols, such as TCP and UDP, govern how your OS "packages up" your application data to send it to another host over the network, and check to make sure that it arrived at the destination.

The TCP Header

TCP segment header

<table>
<tr><th>Offsets</th><th>Octet</th><th colspan="8">0</th><th colspan="8">1</th><th colspan="8">2</th><th colspan="8">3</th></tr>
<tr><th>Octet</th><th>Bit</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><th>0</th><th>0</th><td colspan="16">Source port</td><td colspan="16">Destination port</td></tr>
<tr><th>4</th><th>32</th><td colspan="32">Sequence number</td></tr>
<tr><th>8</th><th>64</th><td colspan="32">Acknowledgment number (if ACK set)</td></tr>
<tr><th>12</th><th>96</th><td colspan="4">Data offset</td><td colspan="3">Reserved
000</td><td>NS</td><td>CWR</td><td>ECE</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td><td colspan="16">Window Size</td></tr>
<tr><th>16</th><th>128</th><td colspan="16">Checksum</td><td colspan="16">Urgent pointer (if URG set)</td></tr>
<tr><th>20</th><th>160</th><td colspan="32" rowspan="3">Options (if data offset > 5. Padded at the end with "0" bits if necessary.)</td></tr>
<tr><th>⋮</th><th>⋮</th></tr>
<tr><th>60</th><th>480</th></tr>
</table>

The UDP Header

UDP datagram header

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port															
4	32	Length																Checksum															

Application Layer

In this course, when we define a message format, what we are really doing is defining an *application-layer protocol* that governs how our server and client communicate with each other

Remark about Layering

Layering is done for a good reason: Imagine, when writing your code for Assignment 3, that you had to write separate code based on whether your client is connected over a WiFi connection or an Ethernet cable!

TCP: Additional Features

- TCP has many more features that are beyond the scope of our discussions for this course
- Flow control: If a computer is sending data too fast for the receiver to handle, TCP will automatically slow down to avoid data loss
- Congestion control: If the network is too congested, TCP will automatically slow down to avoid data loss