Lab 5 - Sockets: Master/Slave Voting

ECE 4220/7220
Real Time Embedded Computing
Spring 2016
University of Missouri
OSI Model
**TCP/IP**

<table>
<thead>
<tr>
<th>OSI</th>
<th>TCP/IP</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Application</td>
</tr>
<tr>
<td>6</td>
<td>Presentation</td>
</tr>
<tr>
<td>5</td>
<td>Session</td>
</tr>
<tr>
<td>4</td>
<td>Transport</td>
</tr>
<tr>
<td>3</td>
<td>Network</td>
</tr>
<tr>
<td>2</td>
<td>Data link</td>
</tr>
<tr>
<td>1</td>
<td>Physical</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Application (FTP, SMTP, HTTP, etc.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>TCP (host-to-host)</td>
</tr>
<tr>
<td>6</td>
<td>IP</td>
</tr>
<tr>
<td>5</td>
<td>IP</td>
</tr>
<tr>
<td>4</td>
<td>Network access (usually Ethernet)</td>
</tr>
</tbody>
</table>
Protocols
Layers of TCP/IP

• **Network interface (layer 1):** Deals with all physical components of network connectivity between the network and the IP protocol

• **Internet (layer 2):** Contains all functionality that manages the movement of data between two network devices over a routed network

• **Host-to-host (layer 3):** Manages the flow of traffic between two hosts or devices, ensuring that data arrives at the application on the host for which it is targeted

• **Application (layer 4):** Acts as final endpoints at either end of a communication session between two network hosts
Sockets

• Used for communication across networks
  – Also can be used on same machine

• Two types of communication
  – Connection based (e.g. TCP)
  – Connectionless (e.g. UDP)
Connection based

- When setting up a socket this is denoted by the SOCK_STREAM type
- Perform handshaking to set up a connection between two machines
- Sets up a bi-directional communication (acknowledgement from the receiver)
- Considered reliable since packets will not be inverted
Connectionless

- When setting up a socket this is denoted by the SOCK_DGRAM type
- No setup to create a connection
- Uni-directional communication (no acknowledgment)
- Considered unreliable because packets may be received in the incorrect order
- However, this type of connection makes broadcast available
Socket Structures

- There are three different structures we will encounter when using sockets.
- These structures are used in many socket functions such as binding to ports and sending and receiving messages.
- The three types are sockaddr_in, in_addr, and sockaddr.
sockaddr Structure

• struct sockaddr {
    unsigned short sa_family;
    char sa_data[14];
}

• The sockaddr structure can be seen as a universal structure (notice the 14 bytes)

• By having this universal structure we allow different implementations to be created by allowing users to determine how to use the 14 bytes
sockaddr_in Structure

- struct sockaddr_in {
  short sin_family;
  unsigned short sin_port;
  struct in_addr sin_addr;
  char sin_zero[8];
}

- This structure is a more defined version of the previous sockaddr struct (notice that some of the 14 bytes are used and the rest are left as padding)

- This structure contains all of the information that we need to fill out such as the port and address
in_addr Structure

- struct in_addr{
  unsigned long s_addr;
}
- This is a structure that is located inside our sockaddr_in structure
- This structure has a single field that holds the address variable
Useful Conversion Functions

• htons htonl – goes from host byte order to network byte order
• ntohs ntohl – goes from network to host
• Used for assigning values to setup socket such as addresses and port values
• in_addr inet_addr(char addr)
  – takes string in IPV4 format and converts to a long value
• char* inet_ntoa(struct in_addr in)
  – Converts long to a string readable by the person
Setting up a Socket

int socket(int domain, int type, int protocol)
− Domain is the family type (AF_INET)
− Type: either SOCK_STREAM or SOCK_DGRAM
− Protocol is 0 for default
  This creates a socket, however it is not bound to any machine interface and is useless at this point

int bind(int desc, struct sockaddr *addr, int addrilen)
− This function takes the previously created socket and associates it with a specific device
− The device is described by filling out the fields in a sockaddr_in structure
Using a Socket

- At this point a socket has been set up and bound to a specific device
- We will use `sendto` and `recvfrom` to send and receive messages using our socket.
- These functions also require a `sockaddr_in` structure to be filled out in order to tell the socket where to send the information or who sent you the information
Useful link

http://www.linuxhowtos.org/C_C++/socket.htm

This is a simple (but helpful) tutorial on Sockets