Lecture 3 (Jan 27, 2004)

Outline
- Sockets programming
- IP data forwarding

Berkeley Sockets
- Networking protocols are implemented as part of the OS
  - The networking API exported by most OS’s is the socket interface
  - Originally provided by BSD 4.1c ~1982.
- The principal abstraction is a socket
  - Point at which an application attaches to the network
  - Defines operations for creating connections, attaching to network, sending/receiving data, closing.

Connection-oriented example (TCP)

Connectionless example (UDP)

Socket call
- Means by which an application attached to the network
- int socket(int family, int type, int protocol)
- Family: address family (protocol family)
  - AF_UNIX, AF_INET, AF_NS, AF_IMPLINK
- Type: semantics of communication
  - SOCK_STREAM, SOCK_DGRAM, SOCK_RAW
- Not all combinations of family and type are valid
- Protocol: Usually set to 0 but can be set to specific value.
  - Family and type usually imply the protocol
- Return value is a handle for new socket
**Bind call**

- Binds a newly created socket to the specified address
- int bind(int socket, struct sockaddr *address, int addr_len)
- **Socket**: newly created socket handle
- **Address**: data structure of address of local system
  - IP address and port number (demux keys)
  - Same operation for both connection-oriented and connectionless servers
    - Can use well known port or unique port

**Listen call**

- Used by connection-oriented servers to indicate an application is willing to receive connections
- int listen(int socket, int backlog)
- **Socket**: handle of newly created socket
- **Backlog**: number of connection requests that can be queued by the system while waiting for server to execute accept call.

**Accept call**

- After executing `listen`, the accept call carries out a passive open (server prepared to accept connects).
- int accept(int socket, struct sockaddr *address, int addr_len)
- It blocks until a remote client carries out a connection request.
- When it does return, it returns with a new socket that corresponds with new connection and the address contains the clients address

**Connect call**

- Client executes an active open of a connection
- int connect(int socket, struct sockaddr *address, int addr_len)
- Call does not return until the three-way handshake (TCP) is complete
- Address field contains remote system’s address
- Client OS usually selects random, unused port

**Send(to), Recv(from)**

- After connection has been made, application uses send/recv to data
- int send(int socket, char *message, int msg_len, int flags)
  - Send specified message using specified socket
- int recv(int socket, char *buffer, int buf_len, int flags)
  - Receive message from specified socket into specified buffer

**Socket Implementation**

- Protocol implementation
  - Process per protocol
    - Use a separate process to implement each protocol
    - Messages are passed between processes
  - Process per message
    - Use one process to handle each message/communication
    - Generally more efficient
- Buffer use
  - Applications use buffers as do protocols
    - Copies are VERY expensive
    - Message abstraction enables pointers to be used and minimal copies
Practical issues – using sockets

- You have to be very careful when using these calls
  - Specific data structures and formats
  - Ports cannot be less than 1024
- You can use other tools to see if things are working
  - tcpdump
  - /proc
  - netstat
- Client and server can be on same system
- Think about error handling methods
- Refer to TCP/IP Sockets in C
- Baby steps!!

Data forwarding

Forwarding and Routing

- Routing: involves computation of routes
  - Which path to take
- Forwarding: select an output interface at each hop
  - Assumes routes have been computed
  - Depends only on destination IP address
- They are independent of each other

Forwarding Tables

- Suppose there are \( n \) possible destinations, how many bits are needed to represent addresses in a routing table?
  - \( \log_2 n \)
- So, we need to store and search \( n \times \log_2 n \) bits in routing tables?
  - We’re smarter than that!
Datagram Forwarding

- **Strategy**
  - Every datagram contains destination’s address
  - If directly connected to destination network, then forward to host
  - If not directly connected to destination network, then forward to some router
  - Forwarding table maps network number into next hop
  - Each host has a default router
  - Each router maintains a forwarding table

<table>
<thead>
<tr>
<th>Network</th>
<th>Next Hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>R1</td>
</tr>
<tr>
<td>2</td>
<td>R1</td>
</tr>
<tr>
<td>3</td>
<td>Interface 1</td>
</tr>
<tr>
<td>4</td>
<td>Interface 0</td>
</tr>
<tr>
<td>Default</td>
<td>R3</td>
</tr>
</tbody>
</table>

**Example**

- For router R2 in previous figure

Subnetting and Supernetting

- Fixed network sizes are wasteful
  - What happens if a site asks for 256 IP addresses?
    - Subnetting
- Too many entries at a router can be combined
  - Keep routing tables small
  - Superneting
  - Classless Inter-Domain Routing (CIDR)

Subnetting

- Add another level to address/routing hierarchy: subnet
- Subnet masks define variable partition of host part
- Subnets visible only within site

<table>
<thead>
<tr>
<th>Network number</th>
<th>Host number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class B address</td>
<td>Class B address</td>
</tr>
</tbody>
</table>
| 11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
Forwarding Table Lookup

- Longest prefix match
  - Each entry in the forwarding table is:
    - \(<\text{Network Number} / \text{Num. Bits}> | \text{interface-id}\)

Suppose we have:

- 192.20/16 | i0
- 192.20.12/24 | i1

And destination address is: 192.20.12.7, choose i1