CSE 333: Systems Programming

Section 8

UDP Broadcast and Multicast

A very brief introduction…

* No section next week (yay Thanksgiving!), so we'll do something on networking today
* Expect a more formal introduction in the next few lectures

TCP and UDP

- TCP, or transmission control protocol:
  - Guarantees in-order, at-most-once packet delivery
  - Is connection-based. The server opens a socket to which client(s) can connect and then transmit data
- UDP, or user datagram protocol:
  - Makes no guarantees about packet delivery
  - Is connectionless, so no agent needs to "connect" to another to send data
- What are some scenarios where TCP is more appropriate for communication? Vice versa?

UDP Broadcast

- Allows sending of packets to a particular network layer
  - This could mean all devices connected to the local network, all devices at the University of Washington, all devices connected to the Internet, etc.
    - Most ISPs will automatically filter broadcast packets
  - Can be useful, but multicast (slightly more fine-grained) is generally more appropriate

UDP Multicast

- Like UDP broadcast, but sent to a defined group
  - Listening devices subscribe to a particular group, such as “225.0.0.42”, and wait for messages
    - The block of IPv4 addresses from 224.0.0.0 to 239.255.255.255 is reserved for multicast groups
  - Sending devices publish broadcast messages to the group
- Useful for local discovery services, such as iTunes’ library sharing, the first Starcraft’s multiplayer lobby, and so forth

Receiving UDP packets

- To listen for UDP packets, first open a socket with the “socket” function (no need to memorize any of this)

```
// Create a socket using AF_INET (IPv4) and
// SOCK_DGRAM (UDP).
int socketfd = socket(AF_INET, SOCK_DGRAM, 0);
if (socketfd < 0) {
  perror(“Unable to open socket”);
  return false;
}
```

Receiving UDP packets

- Next bind the socket to a particular address and port

```
struct sockaddr_in receive_addr;
// ... Set some parameters for receive_addr,
// such as the port and address.
if (bind(socketfd,
         reinterpret_cast<struct sockaddr *>(&receive_addr),
         sizeof(receive_addr)) != 0) {
  perror("Failed to bind socket to port");
  return false;
}
```

Receiving UDP packets

- If using multicast (see receiver.cc in the section code), subscribe to the multicast group

```
struct ip_mreq mreq;
// ... Set the multicast group and local address.
// See "man 7 ip" for more details.
if (setsockopt(socketfd_, IPPROTO_IP,
               IP_ADD_MEMBERSHIP, &mreq,
               sizeof(mreq)) != 0) {
  perror("Kernel request to join multicast group failed);
  return false;
}
```

Receiving UDP packets

- Finally, the exciting part. Receive messages!

```
string buffer;
buffer.resize(kBufferSize);
struct sockaddr_in from_addr;
socklen_t from_length = sizeof(from_addr);
// This call blocks until a message has been received over
// the network.
ssize_t num_bytes =
    recvfrom(socketfd_, &buffer[0], buffer.size(), 0,
             reinterpret_cast<struct sockaddr*>(&from_addr),
             &from_length);
// If num_bytes != -1, then we've received a message!
```

Sending multicast packets

- Create a socket in the same way as before, then set the broadcast option

```
int so_broadcast = true;
// See "man setsockopt" for more details on socket
// options.
if (setsockopt(socketfd, SOL_SOCKET, SO_BROADCAST,
               &so_broadcast, sizeof(so_broadcast)) != 0) {
  perror("Failed to set broadcast option for socket");
  return false;
}
```

Sending multicast packets

- Send the message!

```
sockaddr_in broadcast_addr;
// ... (Set the broadcast address parameters, such as the
// multicast group).
if (sendto(socketfd_, buffer.c_str(), buffer_length, 0,
           reinterpret_cast<struct sockaddr*>(
               &broadcast_addr),
           sizeof(broadcast_addr)) != 0) {
  perror("Failed to send message.");
  return false;
}
```

Section exercise

- Finish implementing a chat program
- Write the code for constructing and sending messages (broadcaster.cc) and receiving and parsing messages (receiver.cc)
- As a suggestion, start by writing your receiver and then test it out by sending messages to it with the sample solution binary
- Make sure to validate received messages! I'll be sending out malicious packets to try to crash your programs ☺
- Submit broadcaster.cc and receiver.cc to the Dropbox when done