Programming UDP

Peter Chapin
Vermont State University
CIS-3152: Network Programming

UDP and Sockets

- The sockets API doesn't directly talk about particular transport protocols.
  - Sockets works with multiple protocols
    - TCP, UDP, etc, are in the TCP/IP protocol stack.
    - The OSI protocols are different.
    - etc...
- TCP is a `SOCK_STREAM` protocol
- UDP is a `SOCK_DGRAM` protocol

Creating a Socket

- Use the same `socket` function as with TCP
  - ```
    if ((socket_handle =
         socket(PF_INET, SOCK_DGRAM, 0)) == -1) {
      perror("Unable to create socket");
      return error_code;
    }
    ```
- The combination `PF_INET` and `SOCK_DGRAM` means UDP.
    ```

Prepare Address

- As with TCP you must prepare a `sockaddr_in` structure.
  - Contains address where you want to send the datagram (UDP packet).
  - Nothing different here.

No Connection!

- No call to `connect` is necessary.
  - Each datagram must be addressed individually.
    - Like a traditional postal envelope.
  - With TCP, `connect` is told the address of the other endpoint.
    - ... thus it is *not* necessary to specify the address for each write operation.
  - UDP is different.

Prepare a Buffer

- You must format the datagram yourself.
  - `char buffer[512];`
    - Put any data you want into the buffer.
- *Size of the datagram is an application issue.*
  - Should not be too large (IP protocol has limits!)
  - Note that packet structure is now exposed to the application. This is different than with TCP.
    - UDP is not a streaming protocol. Application must manage the datagrams.
    - This is an important distinction between stream and datagram protocols in general.

Send the Buffer

- Use the sendto function to send the datagram.
  - ```
    if (sendto(socket_handle,
               buffer,  // Pointer to data
               length,  // Number of bytes to send
               0,       // Flags (see man page)
               (struct sockaddr *) &server_address,
               sizeof(server_address)) == -1) {
      perror("Unable to send");
      return error_code;
    }
    ```
  - Notice that the destination address must be given.
  - Be sure to specify an appropriate length.
    - In some cases you won't want to send the entire buffer (application dependent).
    ```

Receive Reply

- Receives not only data, but address of sender.
  - ```
    int address_length = sizeof(struct sockaddr_in);
    if ((count = recvfrom(socket_handle,
            buffer, // Buffer to store incoming data
            512,    // Size of buffer
            0,      // Flags (see man page)
            (struct sockaddr *) &server_address,
            &address_length)) == -1) {
      perror("Error during packet receive");
      return error_code;
    }
    ```
  - The `address_length` is passed as an in/out parameter.
  - `recvfrom` returns number of bytes actually received.
    ```

Address Handling

- Each call to `recvfrom` returns the address of the sender.
- To reply turn that address around.
  - Use the `sockaddr_in` structure returned by `recvfrom` in the next call to `sendto`.
- When sending to a UDP server...
  - Send initial request to the server's “well known port”
  - Send subsequent datagrams (if any) to the address in the server's reply (probably a different port).
  - Server uses a new port for each client.

One To One

- Each `sendto` call produces exactly one datagram.
  - Calls to `sendto` are not combined or split.
- Each `recvfrom` call returns exactly one datagram.
  - Incoming datagrams are not combined or split.
- *This is another aspect of a datagram protocol.*

Timeout

- UDP is unreliable
  - When sending a request to the server, there may never be a reply:
    - Server is off-line (you don't know until you try!)
    - Request lost on network.
    - Reply lost on network.
      - QUIZ: Does it matter which of the two cases above happened?
- `recvfrom` will normally wait forever.
  - That's bad if the reply never comes.

SIGALRM

- On Unix you can timeout with `SIGALRM`.
  - Just before calling `recvfrom`, call `alarm`.
    - The `alarm` function takes a count of seconds as an argument.
    - Raises the `SIGALRM` signal after that time elapses.
  - Install a signal handling function that does nothing.
  - When `SIGALRM` is raised, `recvfrom` will return with the `EINTR` error code (interrupted system call).
    - In that case, you timed out.
  - Use `alarm(0)` to cancel the alarm if `recvfrom` returns normally.

select

- On Windows use the `select` function.
  - This function can wait for multiple sockets.
    - ... But we will use it to wait for just one.
    - We use it because it provides a timeout option.
  - Note: `select` also available on Unix.
  - ```
    fd_set handles;
    struct timeval timeout = { 10, 0 };
    ...
    FD_ZERO(&handles);
    FD_SET(socket_handle, &handles);
    if (select(1, &handles, NULL, NULL, &timeout)
                                              == 0) {
      // Timed out (zero sockets ready)
    }
    ```
    ```

Socket Option

- Another approach is to use `setsockopt`
  - The `SO_RCVTIMEO` option sets a receiver timeout on a specified socket.