14 - Lecture - Sockets and HTTP

Preparation

UNIX I/O using file descriptors

- review lecture note 11 - Introduction to UNIX

Big-endian vs. Little-endian

- /home/jae/cs3157-pub/sample-code/misc/endian-demo.c

Sockets API

Recommended reading & reference:

- Beej's Guide to Network Programming - http://beej.us/guide/bgnet/
    - Chapter 2: What is a socket?
    - Chapter 3: IP Addresses, structs, and Data Munging
    - Chapter 5: System Calls or Bust

Lecture slides: Sockets API and HTTP 1.0

- http://www.cs.columbia.edu/%7Ejae/3157/files/overview-sockets-http.pdf

TCP Client & Server code examples:

- /home/jae/cs3157-pub/sample-code/tcp/tcp-sender.c
- /home/jae/cs3157-pub/sample-code/tcp/tcp-recver.c

IPv4 address structures:

```
struct sockaddr {
    sa_family_t sa_family;
    char        sa_data[14];
}

struct sockaddr_in {
    sa_family_t    sin_family; /* address family: AF_INET */
    in_port_t      sin_port;   /* port in network byte order */
    struct in_addr sin_addr;   /* internet address */
};

struct in_addr {
    uint32_t       s_addr;     /* address in network byte order */
};
```

```
send(int socket, const void *buffer, size_t length, int flags)
```

- normally, send() blocks until it sends all bytes requested
- returns num bytes sent or -1 for error
- send(sock,buf,len,0) is equivalent to write(sock,buf,len)

recv(int socket, void *buffer, size_t length, int flags)

    - normally, recv() blocks until it has received at least 1 byte
    - returns num bytes received, 0 if connection closed, -1 if error
    - recv(sock,buf,len,0) is equivalent to read(sock,buf,len)

    - With TCP sockets, we can receive less data than we requested;
      MSG_WAITALL flag changes this behavior -- it requests that the
      operation block until the full request is satisfied.

HTTP

Recommended reading & reference:

    HTTP Made Really Easy: http://www.jmarshall.com/easy/http/

Lecture slides: Sockets API and HTTP 1.0

    - http://www.cs.columbia.edu/%7Ejae/3157/files/overview-sockets-http.pdf

HTTP protocol in action:

    An example of HTTP exchange using netcat.

HTTP 1.0 v. HTTP 1.1

  - persistent connection

Dynamic web page

  - HTML form:

```
<FORM METHOD=GET ACTION="/mdb-lookup">
    Lookup: <INPUT TYPE="text" NAME="key">
    <p>
    <INPUT TYPE="submit" VALUE="Lookup">
</FORM>
```

    when the text box is filled with "abc" and button clicked, the
    browser sends:

```
GET /mdb-lookup?key=abc HTTP/1.1
```

  - How do dynamic web sites manage sessions?

    Server "feeds" cookie at the start of session:

```
HTTP/1.1 200 OK
Content-type: text/html
Set-Cookie: name=value
```

    Browser stores the cookies in a local file, indexed by web sites,
    and on subsequent visits, it brings the corresponding cookie
    along:

```
GET /index.html HTTP/1.1
Cookie: name=value
```