Chapter 6
Client Software Design
Program arguments

Arguments from the command line are sent to the program as parameters.

`int main(int argc, char* argv[])`
(Example call: `a.out Hi there    joe x-y-z`)

`argc` is the number of parameters the command line.
Example: `argc` is 5. Arguments are separated by space(s).
The command is counted.

`argv` is an array of strings.
Example: `argv[2]` is "there".
Example: `argv[0]` is "a.out" (the name of the command).

Commandline arguments are used freguently

```
telnet cheetah
g++ a.cc b.cc
```

Example program code (from assignment 1):

```
 src_fd = open(argv[1],0);
```

Domain Name Service

Given a name you can get the internet number(s)
Given an internet number you can get the name.
Limited nicknames available.

```
struct hostent *gethostbyname(const char *name);
```

given an internet name, returns a host structure

```
struct hostent *gethostbyaddr(const void *addr,
                        int len, int type);
```

given an internet number, returns a host structure

You get `NULL` (not found) or
a pointer to a `hostent` structure

```
struct  hostent {
  char    *h_name;        /* official name */
  char    **h_aliases;    /* alias list */
  int     h_addrtype;     /* host address type */
  int     h_length;       /* length of address */
  char    **h_addr_list;  /* list of addresses */
#define h_addr  h_addr_list[0]  /* compatiblity */
};
```

type is `AF_INET`, length is 4 (4 byte integer).
`h_addr` is the address (IP number)
`char *` type to allow any length address
array of addresses applies to gateway machines

Byte Order

Byte order for integers varies.

High byte first: SPARC, 68000, MIPS, RS6000

Low byte first: Pentium, MIPS

Needed for network communication: a standard order.
(High byte first)

Needed for programming: conversion routines from host order to network order.

`htonl` (host to network–long): convert a number from host order (whatever that is) to network order.

`ntohl` (network to host–long)
`htons` (host to network–short)
`ntohs` (network to host–short)

Domain Name Service Example

```
struct hostent *hp, *hp2;
struct in_addr addr;
hp = gethostbyname("cheetah.cecs.csulb.edu");
printf("%s\n",hp->h_name);
printf("%d\n",hp->h_length);
memcpy(&addr, hp->h_addr, hp->h_length);
printf("%x\n",ntohl(addr.s_addr));
printf("%s\n",inet_ntoa(addr));
hp2 = gethostbyaddr(&addr, 4, AF_INET);
```

`h_length` is always 4 for IP.
copy is necessary to unfool c about length (1 for `char`)
`h_addr` is in network byte order
`inet_ntoa` converts an internet number (in network byte order, in an `in_addr` structure) to ascii dot notation.

The contents of `*hp2` should be the same as `*hp`.

Port and Services

```
struct servent *getservbyname(
       const char *name, const char *proto)
struct servent *getservbyport(
             int port, const char *proto)
```

Given a name (or a port number) and protocol (UDP/TCP), return a service structure.

```
struct servent {
  char *s_name;  /* name of service */
  char **s_aliases;   /* alias list */
  int  s_port;              /* port */
  char *s_proto; /* protocol to use */
};
```

`port` is `int` here, `u_short` in `sockaddr_in`
proto and name usually echo what you entered

```
 struct servent *sp, *sp2;
 sp = getservbyname("telnet", "tcp");
 printf("%s\n",sp->s_name); /* telnet */
 printf("%d\n",ntohs(sp->s_port)); /* 23 */
 printf("%s\n",sp->s_proto); /* tcp */
 sp2 = getservbyport(htons(23), "tcp");
```

`s_port` is in network byte order
`*sp` and `*sp2` should contain identical data.

More Tricks

Getting integers from a command line argument requires an extra step.

For example, if you run the command:

```
lab99> a.out 33
```

`argv[1]` will be `"33"`, that is, it will be a string and not an integer.

To get an integer you use the ASCII to integer conversion function `atoi`.

Warning: `atoi(argv[1])` will return an `int`, not a `short`, if you need a `short` you either need to store the value in a variable that is a `short` or cast.

Some implementations will allow you to replace the tcp/udp string with the value `NULL` as in:

```
getservbyport(htons(23), NULL);
```

This will match either TCP or UDP. Which value is returned if the service has both TCP and UDP is not specified.

TCP Algorithm (6.1)
(Client)

1. Get server address and service port number
2. Get socket
3. Set up `sockaddr_in` struct (for 4.)
4. Get connection (use any local port)
5. Talk with server
6. Close the connection

Message delivery guaranteed (or error is signalled)

Connection establishment is ensured by protocol

Close/shutdown is communicated to other end-point

UDP Algorithm (6.2)
(Client)

1. Get server address and service port number
2. Get socket
3. Set up `sockaddr_in` struct (for 4.)
4. Set socket for server (use any local port)
5. Send/receive messages to/from server
6. Close the socket

No guarantee that messages will get there.

Application must handle missing message problem

System calls (review)

`socket`: set up a socket, you specify TCP or UDP.

`connect`: client only. Must pass this an `sockaddr_in` with the address of the server. For TCP sockets, this forms a connection. For UDP sockets, this sets up a default destination address.

`accept`: TCP server only. Accept a client's connection request. You get a new socket for each accept.

`write`: send a message. Params are socket, buffer, count.
`send`: Params are socket, buffer, count, flagbits. If flagbits is 0, it is identical to write.

`read`: receive a message. Params are socket, buffer, buffersize.
`recv`: Params are socket, buffer, buffersize, flagbits. If flagbits is 0, it is identical to read.

Variations (UDP only):

`sendto`: Message destination given as an argument.

`sendmsg`: Message structure contains destination.

`recvfrom`: Message source given as an argument.

`recvmsg`: Message structure contains source.