This Week

- No Quiz
- Tests handed back yesterday
  - See your TA to get your test
- No Class Thursday – Carnival
- Next Quiz
  - Network Programming
IP Addresses

- IPv4
  - 32-bit addresses
  - 128.2.134.11 (gs3082.sp.cs.cmu.edu)

- IPv6
  - 128-bit addresses

- Lab only deals with IPv4
IP Address Struct

```
#include <netinet/in.h>

struct in_addr {
    unsigned int s_addr; /* network byte order (big-endian) */
};
```

- Addresses are always stored in network byte order (big-endian)
- Machine you are working on may be little-endian
- Conversion functions

```
#include <netinet/in.h>

unsigned long int htonl(unsigned long int hostlong);
unsigned short int htons(unsigned short int hostshort);

unsigned long int ntohl(unsigned long int netlong);
unsigned short int ntohs(unsigned short int netshort);
```
Domain Naming System (DNS)

- Mapping from easy(er) to remember names to IP addresses
- gs3082.sp.cs.cmu.edu ↔ 128.2.134.11

```c
#include <netdb.h>

/* DNS host entry structure */
struct hostent {
    char   *h_name;       /* official domain name of host */
    char   **h_aliases;   /* null-terminated array of domain names */
    int    h_addrtype;    /* host address type (AF_INET) */
    int    h_length;      /* length of an address, in bytes */
    char   **h_addr_list; /* null-terminated array of in_addr structs */
};
```
DNS Queries

```c
#include <netdb.h>
struct hostent *gethostbyname(const char *name);

gethostbyname(gs3082.sp.cs.cmu.edu) = 128.2.134.11
```

```c
#include <netdb.h>
struct hostent *gethostbyaddr(const char *addr, int len, 0);

gethostbyaddr(128.2.134.11, 4, 0) = gs3082.sp.cs.cmu.edu
```
Sockets

- Endpoints of connections between hosts
- Socket address
  - IP address (32 bits)
  - Port (16 bits)
  - address:port
- Connection identified by socket pairs
  - (clientAddr:clientPort, serverAddr:serverPort)
Ports

- **Clients**
  - Use any port given to them by the socket interface

- **Servers**
  - Host specific services on specific ports
    - Port 7: echo server
    - Port 23: telnet server
    - Port 25: mail server
    - Port 80: HTTP server
Sockets Interface

Client

- `socket`
- `connect`
- `rio_readlineb`
- `rio_writen`
- `close`

Server

- `socket`
- `bind`
- `listen`
- `accept`
- `rio_readlineb`
- `rio_writen`
- `close`

Client / Server Session

- `open_clientfd`
- `open_listenfd`

Await connection request from next client
Socket Address Structs

- Generic socket address
  - Passed to `connect`, `bind`, and `accept`

```c
#include <sys/socket.h>

struct sockaddr {
    unsigned short  sa_family;    /* protocol family */
    char            sa_data[14];  /* address data. */
};
```

`sa_family`
Socket Address Structs

- Internet-specific socket address
- Must be cast to (sockaddr *) before passing to connect, bind or accept

```c
#include <netinit/in.h>

struct sockaddr_in {
    unsigned short sin_family; /* address family (always AF_INET) */
    unsigned short sin_port;  /* port num in network byte order */
    struct in_addr sin_addr;  /* IP addr in network byte order */
    unsigned char   sin_zero[8]; /* pad to sizeof(struct sockaddr) */
};
```

```
AF_INET  0 0 0 0 0 0 0 0 0 0
 sin_port  sin_addr
 sin_family
```

Family Specific
The socket Function

- Used to create a socket descriptor
  - Very similar to a standard file descriptor
  - Open procedure differs for clients or server

```c
#include <sys/types.h>
#include <sys/socket.h>

int socket(int domain, int type, int protocol);
```

- For all applications in this class:
  - domain = AF_INET
  - type = SOCK_STREAM
  - protocol = 0
The connect Function

- Used by the **client** to create a connection to a server

```c
#include <sys/socket.h>

int connect(int sockfd, struct sockaddr *srv_addr, int addrlen);
```

- **srv_addr** is the address of the server
- **addrlen** is the `sizeof(sockaddr_in)`
- If `connect` returns successfully, **sockfd** can be read and written just like any other file descriptor
The `bind` Function

- Used by the **server** to grab hold of a specific socket address on the server and tie it to the listed file descriptor

```c
#include <sys/socket.h>

int bind(int sockfd, struct sockaddr *my_addr, int addrlen);
```

- **sockfd** is file descriptor to associate with socket address
- **my_addr** is local IP address and unused port
- **addrlen = sizeof(sockaddr_in)**
The `listen` Function

- By default, sockets are assumed active.
- `listen` converts `sockfd` from an active socket to a listening socket, waiting for connections.

```c
#include <sys/socket.h>

int listen(int sockfd, int backlog);
```

- `backlog` is a hint to the kernel about how many requests should be queued before requests are refused.
The `accept` Function

- `accept` waits for an incoming connection on the listed file descriptor
- Once connected, it fills in the client’s socket address and returns a connected descriptor
  - Descriptor can then be used to communicate with the client

```c
#include <sys/socket.h>

int accept(int listenfd, struct sockaddr *addr, int *addrlen);
```
- Server blocks in `accept`, waiting for connection request on listening file descriptor `listenfd`

- Client makes connection request by calling and blocking in `connect`

- Server returns `connfd` from `accept`. Client returns from `connect`. Connection is now established between `clientfd` and `connfd`
Listening vs. Connected

- **Listening descriptor**
  - End point for client connection requests.
  - Created once and exists for lifetime of the server.

- **Connected descriptor**
  - End point of the connection between client and server.
  - A new descriptor is created each time the server accepts a connection request from a client.
  - Exists only as long as it takes to service client.

- **Why the distinction?**
  - Allows for concurrent servers that can communicate over many client connections simultaneously.
    - E.g., Each time we receive a new request, we fork a child to handle the request.
**telnet**

- Great testing tool for servers that send ASCII over Internet connections

- **Usage:**
  - `unix> telnet <host> <portnumber>`
  - Creates a connection with a server running on `<host>` and listening on port `<portnumber>`
Echo client & server

On Server

```
bass> echoserver 5000
server established connection with KITTYHAWK.CMCL (128.2.194.242)
server received 4 bytes: 123
server established connection with KITTYHAWK.CMCL (128.2.194.242)
server received 7 bytes: 456789
...
```

On Client

```
kittyhawk> echoclient bass 5000
Please enter msg: 123
Echo from server: 123

kittyhawk> echoclient bass 5000
Please enter msg: 456789
Echo from server: 456789
kittyhawk>
```
Echo Client Main Routine

```
#include "csapp.h"

/* usage: ./echoclient host port */
int main(int argc, char **argv)
{
    int clientfd, port;
    char *host, buf[MAXLINE];
    rio_t rio;
    host = argv[1];  port = atoi(argv[2]);
    clientfd = Open_clientfd(host, port);
    Rio_readinitb(&rio, clientfd);
    printf("type:"); fflush(stdout);
    while (Fgets(buf, MAXLINE, stdin) != NULL) {
        Rio_writen(clientfd, buf, strlen(buf));
        Rio_readlineb(&rio, buf, MAXLINE);
        printf("echo:");
        Fputs(buf, stdout);
        printf("type:"); fflush(stdout);
    }
    Close(clientfd);
    exit(0);
}
```
int open_clientfd(char *hostname, int port) {
    int clientfd;
    struct hostent *hp;
    struct sockaddr_in serveraddr;

    if ((clientfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
        return -1; /* check errno for cause of error */

    /* Fill in the server's IP address and port */
    if ((hp = gethostbyname(hostname)) == NULL)
        return -2; /* check h_errno for cause of error */
    bzero((char *) &serveraddr, sizeof(serveraddr));
    serveraddr.sin_family = AF_INET;
    bcopy((char *) hp->h_addr, (char *)&serveraddr.sin_addr.s_addr, hp->h_length);
    serveraddr.sin_port = htons(port); /* Establish a connection with the server */
    if (connect(clientfd, (SA *) &serveraddr, sizeof(serveraddr)) < 0)
        return -1;
    return clientfd;
}
bcopy Arguments

/* DNS host entry structure */
struct hostent {
  . . .
  int    h_length;    /* length of an address, in bytes */
  char   **h_addr_list; /* null-terminated array of in_addr structs */
};

struct sockaddr_in  {
  . . .
  struct in_addr    sin_addr;  /* IP addr in network byte order */
};

/* Internet address structure */
struct in_addr  {
  unsigned int    s_addr;  /* network byte order (big-endian) */
};

struct hostent  *hp;    /* DNS host entry */
struct sockaddr_in  serveraddr; /* server’s IP address */
...
bcopy((char *)hp->h_addr_list[0], /* src, dest */
   (char *)&serveraddr.sin_addr.s_addr, hp->h_length);
int main(int argc, char **argv) {
    int listenfd, connfd, port, clientlen;
    struct sockaddr_in clientaddr;
    struct hostent *hp;
    char *haddrp;

    port = atoi(argv[1]); /* the server listens on a port passed
                             on the command line */
    listenfd = open_listenfd(port);

    while (1) {
        clientlen = sizeof(clientaddr);
        connfd = Accept(listenfd, (SA *)&clientaddr, &clientlen);
        hp = Gethostbyaddr((const char *)&clientaddr.sin_addr.s_addr,
                            sizeof(clientaddr.sin_addr.s_addr), AF_INET);
        haddrp = inet_ntoa(clientaddr.sin_addr);
        printf("server connected to %s (%s)\n", hp->h_name, haddrp);
        echo(connfd);
        Close(connfd);
    }
}
int open_listenfd(int port)  
{  
    int listenfd, optval=1;  
    struct sockaddr_in serveraddr;  

    /* Create a socket descriptor */  
    if ((listenfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)  
        return -1;  

    /* Eliminates "Address already in use" error from bind. */  
    if (setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR,  
                     (const void *)&optval, sizeof(int)) < 0)  
        return -1;  

    ... (more)
/* Listenfd will be an endpoint for all requests to port 
on any IP address for this host */
bzero((char *) &serveraddr, sizeof(serveraddr));
serveraddr.sin_family = AF_INET;
serveraddr.sin_addr.s_addr = htonl(INADDR_ANY);
serveraddr.sin_port = htons((unsigned short)port);
if (bind(listenfd, (SA *)&serveraddr, sizeof(serveraddr)) < 0)
    return -1;

/* Make it a listening socket ready to accept 
connection requests */
if (listen(listenfd, LISTENQ) < 0)
    return -1;

return listenfd;
setsockopt

- The socket can be given some attributes.

```c
... /* Eliminates "Address already in use" error from bind(). */
if (setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR,
               (const void *)&optval, sizeof(int)) < 0)
    return -1;
```

- Handy trick that allows us to rerun the server immediately after we kill it.
  - Otherwise we would have to wait about 15 secs.
  - Eliminates “Address already in use” error from bind().

- Strongly suggest you do this for all your servers to simplify debugging.
Testing with `telnet`

```
bass> echoserver 5000
server established connection with KITTYHAWK.CMCL (128.2.194.242)
server received 5 bytes: 123
server established connection with KITTYHAWK.CMCL (128.2.194.242)
server received 8 bytes: 456789

kittyhawk> telnet bass 5000
Trying 128.2.222.85... Connected to BASS.CMCL.CS.CMU.EDU.
Escape character is '^[]'.
123
123
Connection closed by foreign host.
kittyhawk> telnet bass 5000
Trying 128.2.222.85... Connected to BASS.CMCL.CS.CMU.EDU.
Escape character is '^[]'.
456789
456789
Connection closed by foreign host.
kittyhawk>
```