CSci 4061
Introduction to Operating Systems

Network Systems Programming

Network Programming

Socket headers

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
```

Creating a socket

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

- Creates a communication end-point, returns a `fd`
- Domain is the communication domain:
  - PF_UNIX is for local Unix communication
  - **PF_INET**/INET6 is for IPv4/v6 Internet communication

Creating a socket (cont'd)

- Type specifies the communication style
- TCP
  - SOCK_STREAM
    - reliable stream of bytes
- UDP
  - SOCK_DGRAM
    - unreliable message delivery

```
fd = socket (PF_INET, SOCK_STREAM, 0);
```

Using sockets: client-server

- Server – 4 steps
  - Creates an open socket with `socket`
  - Bind the socket to a network address (IP, port) with `bind`
  - Set up a queue for incoming connection requests with `listen`
  - `accept` a connection

Using sockets (cont'd)

- Accept connection requests with `accept`
  - Creates a **new** socket connected to the client
  - Returns new socket identifier as an `fd`
- Read and write to/from the new socket fd with our old friends `read/write`
- Close the new socket `fd` with `close`() when you are done
- <picture>

Socket addressing

- `bind()` takes a socket address structure with an address format that depends on domain

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

len? different sizes for different types (IPv4, IPv6, …)

```
struct  sockaddr{
 sa_family_t sin_famly;     // AF_INET, same as PF_INET
 u_intl6_t sin_port;        // port: network byte order
 struct in_addr sin_addr;   // IP address: network byte order
}
```

Using bind

Network byte order

```
int fd;
int port = 6666;
fd = socket (…);
struct sockaddr_in addr:
addr.sin_family= AF_INET;
addr.sin_addr.s_addr= htonl(INADDR_ANY);
addr.sin_port= htons(port); //server picks the port
bind (fd,(struct sockaddr*)&addr, sizeof(addr));
```

`INADDR_ANY`:

OS picks local IP address—if multi-homed, then may want to hand-pick address.

Bind error: this port is already taken on this machine

Avoiding Port Collision

- If you are testing your server… `bind` … then crash …
- OS holds port for a while and you may get a `bind` error next time you run the server

```
int enable = 1;

setsockopt(fd, SOL_SOCKET, SO_REUSEADDR,
           (char*)&enable, sizeof(int));
```

- Gives back ports to OS: call before `bind` so next `bind` call does not fail

Choosing Ports

- You can pick # >1024 (up to 65536)
- *ss* will list active ports

Listen and accept

`listen`() sets up a queue for incoming connection requests
`accept`() gives you a channel to the requester

```
#include <sys/types.h>
#include <sys/socket.h>
int listen (int fd, int backlog);
int accept (int fd, struct sockaddr* addr;
            socklen_t* addrlen);
```

// create a socket … enable/reuse … bind

```
listen (fd, 5);  //queue up 5 pending requests
int accept (fd,(struct sockaddr*)&client_addr
            &addr_len);
```

//can `read` or `write` to `new_fd` so can the other side

Accept

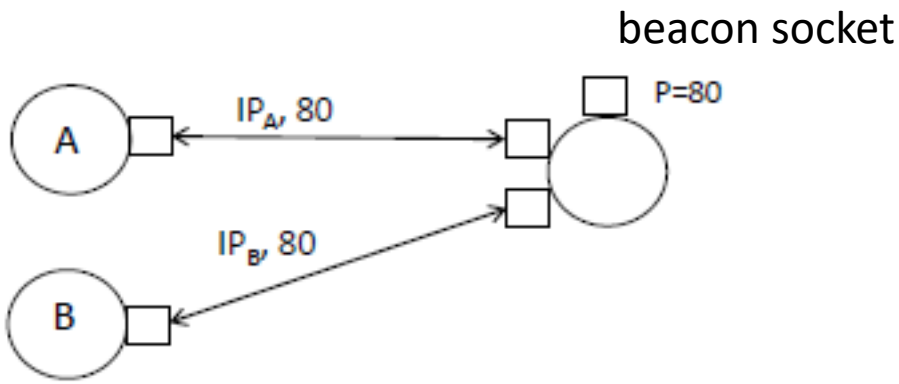
`accept` returns a `fd` for a new socket

`accept` blocks until a request arrives

Server Styles

- Single thread, one request at a time …

- Thread/request
  - Each new request is handled by a new thread
  - Could also have a thread pool



Server Styles (cont'd)

- Multiplexed
  - Each thread does a portion of request handling (i.e. lab #3)
- Server cannot selectively `accept` clients!
- Cannot say, accept from IP 128.118.44.3
  - Can, however, block/pass it through with a firewall

Now to the client …

- First, the client needs to know the IP and port of the server

- It can use DNS to get the IP based on the symbolic name
  - cnn.com, caesar.cs.umn.edu, …

Host Lookup

```
#include <netdb.h>

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

"caesar.cs.umn.edu"

hostent **contains** sockaddr

150.91.115.42

Client Scenario

- Create socket with `socket`
- Resolve IP address, port of server
- No need to `bind`, OS does it automatically (when you `connect`) and finds a free port
- Connect to the server with `connect`
- Read and write to/from the socket with
  - `read/write` or other socket I/O calls
  - close the socket with `close`

Internet Addresses

- Internet addresses
  - IP + port
- Must be understandable by the network layer (routers) and end-host
- Network byte order
- Functions to convert components from host to network formats and *vice-versa*

```
#include <inet.h>
uint32_t htonl(uint32_t hostlong); // long for IP
uint32_t ntohl(uint32_t netlong);
```

- short int conversion for ports