COMP20200 Unix Programming

Lecture 13

CS, University College Dublin, Ireland

Unix Networking

- This section of the course: processes communicating with each other.
- Programming network sockets.
- But first we'll have a look at some networking tools.
- View and edit network connections with `ifconfig`

```
$ ifconfig -a
eth0 Link encap:Ethernet  HWaddr 00:11:43:07:01:12
     inet addr:193.1.133.230 Bcast:193.1.133.255 Mask:255.255.254.0

lo   Link encap:Local Loopback
     inet addr:127.0.0.1  Mask:255.0.0.0
$
```

Other tools/files related to networking GNU/Linux

```
route hostname host
/etc/network/ /etc/hosts
```

Secure Shell (SSH)

- Secure data communication between two networked computers.
- ssh server runs on remote server
- ssh client on local computer.
- Execute remote commands and more.
- Encrypted communication.
- Replacement for rsh and rlogin

```
$ ssh user@csserver.ucd.ie
user@csserver.ucd.ie's password:
[user@csserver ~]$
[user@csserver ~]$ exit
Connection to csserver.ucd.ie closed.
$
```

- Motivation for learning command line: use remote computer as easily as local.

ssh Keys

Authentication without password

- /etc/hosts.equiv or /etc/ssh/shosts.equiv on remote machine: lists local machine and username
- .rhosts or .shosts in user's home directory on remote machine
- Authentication keys (DSA or RSA) make connections easier, more secure.

```
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/john/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in id_rsa.
Your public key has been saved in id_rsa.pub.

$ cat id_rsa.pub >>  ~/.ssh/authorized_keys

$ ssh-copy-id user@csserver.ucd.ie id_rsa.pub
$
```

NFS and File Transfer

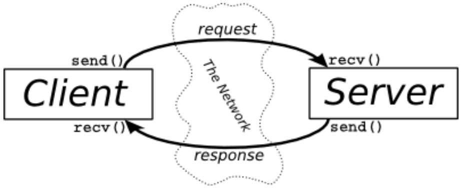
- NFS - Network File System
- scp - remote copying of files. Uses ssh protocol.

```
$ scp user@server:path/remote_file local/path/
$ scp local_file user@server:remote/path/

$
```

Networking and Sockets

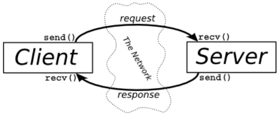
- What is a socket?
  - A way for different processes to communicate.
  - Processes may be on same machine or different machines connected by a network.
  - Since in Unix "everything is a file", so are sockets.
  - Or more technically correct: communication through file descriptors.
  - Call to `socket()` system routine returns socket discriptor.
  - Communicate with `send()` and `recv()` socket calls.



Sockets: Server Sending Data to Client

Client:

- Setup socket address structure
  - Communication domain
  - server address
  - Port number
- Create socket namespace
  - Communication domain
  - Communication type
- Connect to server
- recv data
- print, free, exit



Server:

- Setup server address struct
  - Communication domain
  - all addresses
  - Port number
- Empty client address struct
- Create socket namespace
  - Communication domain
  - Communication type
- Bind socket to address
- listen for incoming connections
- while connections
  - accept connection
  - send data

Client:

```
#define PORTNUM 2343
int main(int argc, char *argv[]){
 char buffer[MAXRCVLEN + 1];
 struct sockaddr_in dest;
 memset(&dest, 0, sizeof(dest));
 dest.sin_family = AF_INET;
 dest.sin_addr.s_addr =
       inet_addr("127.0.0.1");
 dest.sin_port = htons(PORTNUM);

 int mysocket = socket(AF_INET,
       SOCK_STREAM, 0);

 connect(mysocket,
       (struct sockaddr *)&dest,
       sizeof(struct sockaddr));

 len = recv(mysocket, buffer,
     MAXRCVLEN, 0);
 buffer[len] = '\0';

 printf("Received%s (%dbytes).\n"
       , buffer, len);
 close(mysocket);
 return EXIT_SUCCESS; }
```

Server: *(see moodle for full source)

```
#define PORTNUM 2343
int main(int argc, char *argv[]){
 char msg[] ="Hello World!\n";
 struct sockaddr_in dest, serv;
 memset(&serv, 0, sizeof(serv));
 serv.sin_family = AF_INET;
 serv.sin_addr.s_addr=INADDR_ANY;
 serv.sin_port = htons(PORTNUM);

 int mysocket = socket(AF_INET,
     SOCK_STREAM, 0);
 bind(mysocket, &serv,
        sizeof(struct sockaddr));
 listen(mysocket, 1);

 int consocket = accept(mysocket,
      &dest, &socksize);
 while(consocket>0) {
     send(consocket, msg,
         strlen(msg), 0);
     close(consocket);
     consocket = accept(mysocket,
           &dest, &socksize); }
 close(mysocket);
 return EXIT_SUCCESS; }
```

Socket

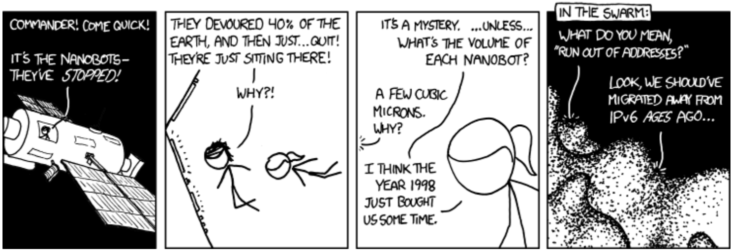
```c
int socket(int domain, int type, int protocol);
```

- Domain
  - AF_UNIX, AF_LOCAL Local communication unix(7)
  - AF_INET IPv4 Internet protocols ip(7)
  - AF_INET6 IPv6 Internet protocols ipv6(7)
  - ... and more...
- Type
  - SOCK_STREAM Reliable, two-way, connection-based byte streams.
  - SOCK_DGRAM Datagrams for connectionless, unreliable messages.
  - SOCK_RAW Provides raw network protocol access.
  - ... and more...
- Protocal: 0 gives default protocol for domain & type.

IPv4 IPv6

IP Addresses, versions 4 and 6

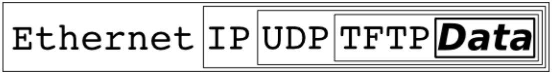
- Version 4: 32-bit
  - eg. 192.168.1.1
  - approximately 4.294 billion addresses
  - Running out!
- Version 6: 128-bit
- eg. 2001:0db8:c9d2:aee5:73e3:934a:a5ae:9551
- 3.4×10^{38} (million IPv4 Internets for every single star in Universe)


COMMANDER! COME QUICK!
IT'S THE NANOBOTS– THEY'VE STOPPED!
THEY DEVOURED 40% OF THE EARTH, AND THEN JUST...QUIT! THEY'RE JUST SITTING THERE!
WHY?!
IT'S A MYSTERY.
...UNLESS... WHAT'S THE VOLUME OF EACH NANOBOT?
A FEW CUBIC MICRONS. WHY?
I THINK THE YEAR 1998 JUST BOUGHT US SOME TIME.
IN THE SWARM:
WHAT DO YOU MEAN, "RUN OUT OF ADDRESSES?"
LOOK, WE SHOULD'VE MIGRATED AWAY FROM IPv6 AGES AGO...

http://xkcd.com/865/

SOCK_STREAM SOCK_DGRAM

- SOCK_STREAM
  - Stream sockets are reliable two-way connected communication streams
  - Output in the order "1, 2", will arrive in the order "1, 2"
  - The Transmission Control Protocol (TCP)
  - SSH, HTTP protocols
- SOCK_DGRAM
  - Connectionless, may arrive out of order, data may be lost.
  - User Datagram Protocol (UDP)
  - Examples DHCP, video streaming



Connect

- Called by client:

```
int connect(int sockfd, const struct sockaddr *server_addr,
    socklen_t addrlen);
```

  - Connect creates connection with socket descriptor to server address.

- Called by Server:

```
int bind(int sockfd, struct sockaddr *my_addr, socklen_t
    addrlen);
int listen(int sockfd, int backlog);
int accept(int sockfd, struct sockaddr *client_addr, socklen_t
     *addrlen);
```

  - Bind connects socket descriptor to local address and port.
  - Listen allows socket to accept incoming connections.
  - Accept connection request and create new socket to client.

send / recv

```
ssize_t send(int sockfd, const void *buf, size_t len, int flags);
ssize_t recv(int sockfd, void *buf, size_t len, int flags);
```

- Only call send / recv when in connected state.
- One side of communication calls send, other calls recv.
- Buffer `buf` is allocated memory of size `len`.
- On host calling send, buf contains the data to be sent.
- On host calling recv, buf will contain data after call.

Client:

```
#define PORTNUM 2343

int main(int argc, char *argv[]){
 char buffer[MAXRCVLEN + 1];
 struct sockaddr_in dest;
 memset(&dest, 0, sizeof(dest));
 dest.sin_family = AF_INET;
 dest.sin_addr.s_addr =
      inet_addr("127.0.0.1");
 dest.sin_port = htons(PORTNUM);

 int mysocket = socket(AF_INET,
      SOCK_STREAM, 0);

 connect(mysocket, &dest,
      sizeof(struct sockaddr));

 len = recv(mysocket, buffer,
     MAXRCVLEN, 0);

 buffer[len] = '\0';
 printf("Received%s (%dbytes).\n"
      , buffer, len);
 close(mysocket);
 return EXIT_SUCCESS; }
```

Server: *(see moodle for full source)

```
#define PORTNUM 2343
int main(int argc, char *argv[]){
 char msg[] ="Hello World!\n";
 struct sockaddr_in dest, serv;
 memset(&serv, 0, sizeof(serv));
 serv.sin_family = AF_INET;
 serv.sin_addr.s_addr=INADDR_ANY;
 serv.sin_port = htons(PORTNUM);

 int mysocket = socket(AF_INET,
     SOCK_STREAM, 0);
 bind(mysocket, &serv,
        sizeof(struct sockaddr));
 listen(mysocket, 1);

 int consocket = accept(mysocket,
     &dest, &socksize);
 while(consocket>0) {
     send(consocket, msg,
        strlen(msg), 0);
     close(consocket);
     consocket = accept(mysocket,
          &dest, &socksize); }
 close(mysocket);
 return EXIT_SUCCESS; }
```