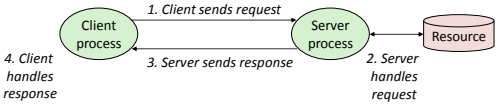
Lab 12

Network Programming

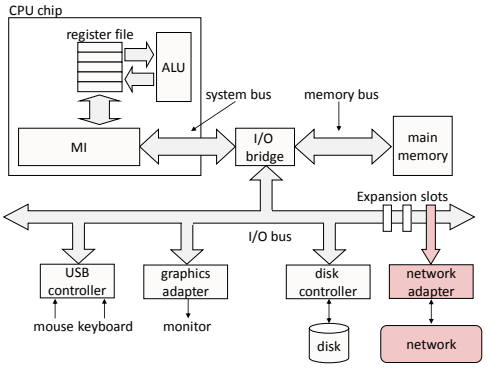
CPSC 275
Introduction to Computer Systems

Client-Server Transaction

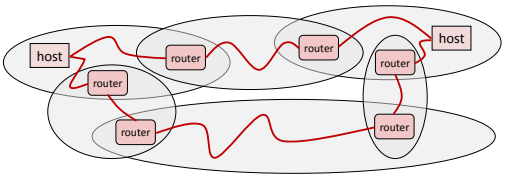


Note: clients and servers are processes running on hosts (can be the same or different hosts)

Hardware Organization of a Network Host



Logical Structure of an internet



Internet Connections

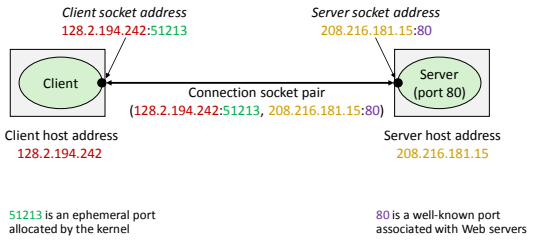
- Clients and servers communicate by sending streams of bytes over *connections*:
  - Point-to-point, full-duplex (2-way communication), and reliable.
- *A socket* is an endpoint of a connection
  - Socket address is an `IPaddress:port` pair

Internet Connections, cont'd

- A *port* is a 16-bit integer that identifies a process:
  - ***Ephemeral port*:** Assigned automatically on client when client makes a connection request
  - ***Well-known port:*** Associated with some service provided by a server (e.g., port 80 is associated with Web servers)
- A connection is uniquely identified by the socket addresses of its endpoints (*socket pair*)

  `(cliAddr:cliPort, servAddr:servPort)`

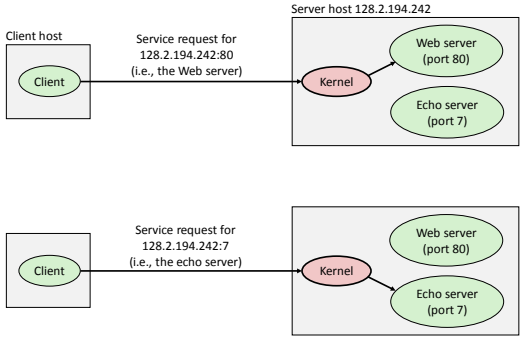
Putting it all Together:



Clients

- Examples of client programs
  - Web browsers, **`ftp`**, **`telnet`**, **`ssh`**
- How does a client find the server?
  - The IP address in the server socket address identifies the host (more precisely, an adapter on the host)
  - The (well-known) port in the server socket address identifies the service, and thus implicitly identifies the server process that performs that service.
  - Examples of well know ports
    - Port 7: Echo server
    - Port 23: Telnet server
    - Port 25: Mail server
    - Port 80: Web server

Using Ports to Identify Services



Servers

- Servers are long-running processes (*daemons*)
  - Created at boot-time (typically) by the **`init`** process (process 1)
  - Run continuously until the machine is turned off
- Each server waits for requests to arrive on a well-known port associated with a particular service
  - Port 7: echo server
  - Port 23: telnet server
  - Port 25: mail server
  - Port 80: HTTP server
- A machine that runs a server process is also often referred to as a "server"

Server Examples

- Web server (port 80)
  - Resource: files/compute cycles (CGI programs)
  - Service: retrieves files and runs CGI programs on behalf of the client
- FTP server (20, 21)
  - Resource: files
  - Service: stores and retrieve files
- Telnet server (23)
  - Resource: terminal
  - Service: proxies a terminal on the server machine
- Mail server (25)
  - Resource: email "spool" file
  - Service: stores mail messages in spool file

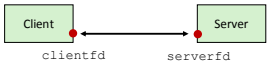
See `/etc/services` for a comprehensive list of the port mappings on a Linux machine

Sockets Interface

- Created in the early 80's as part of the original Berkeley distribution of Unix that contained an early version of the Internet protocols
- Provides a user-level interface to the network
- Underlying basis for all Internet applications
- Based on client/server programming model

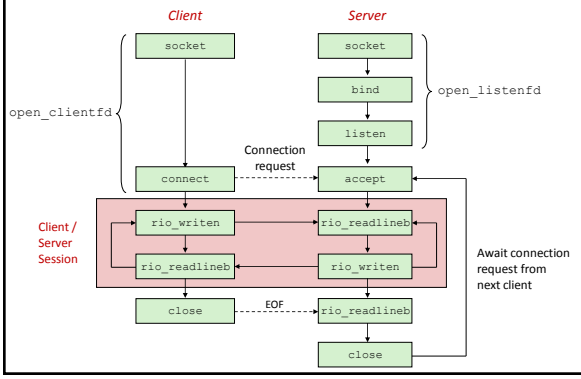
Sockets

- What is a socket?
  - To the kernel, a socket is an endpoint of communication
  - To an application, a socket is a file descriptor that lets the application read/write from/to the network
    - ***Remember:*** All Unix I/O devices, including networks, are modeled as files
- Clients and servers communicate with each other by reading from and writing to socket descriptors



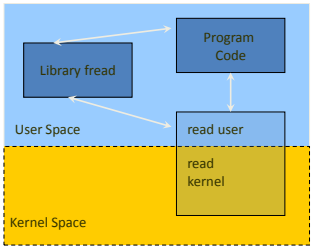
- The main distinction between regular file I/O and socket I/O is how the application "opens" the socket descriptors

Overview of the Sockets Interface



System Calls

- System calls
  - Interface to the kernel



Low-Level System I/O

- Processes keep a list of open files
- Files can be opened for reading, writing
- Each file is referenced by a *file descriptor* (integer)
- Three files are opened automatically
  - 0: standard input
  - 1: standard output
  - 2: standard error

File I/O System Calls: `read()`

```
bytes_read = read(fd, buffer, count)
```

- Read up to count bytes from file and place into buffer
- fd: file descriptor
- buffer: pointer to array
- count: number of bytes to read
- Returns number of bytes read or -1 if error

File I/O System Call: `write()`

```
nbytes = write(fd, buffer, count)
```

- Write count bytes from buffer to a file
- fd: file descriptor
- buffer: pointer to array
- count: number of bytes to write
- Returns number of bytes written or -1 if error

Example: A Simple Copy Program, Version 1

```
#define BUFSIZE 1
main()
{
   char buf[BUFSIZE];
   int n;
   while ((n = read(0, buf, BUFSIZE)) > 0)
       write(1, buf, n);
}
```

File I/O system call: `open()`

`fd = open(path, flags, mode)`

- path: string, absolute or relative path
- flags:
  - O_RDONLY - open for reading
  - O_WRONLY - open for writing
  - O_RDWR - open for reading and writing
  - O_CREAT - create the file if it doesn't exist
  - O_TRUNC - truncate the file if it exists
  - O_APPEND - only write at the end of the file
- mode: specify permissions if using O_CREAT

File I/O system call: `close()`

`retval = close(fd)`

- Close an open file descriptor
- Returns 0 on success, -1 on error

Copy Program

Version 2

```
#define BUFSIZE 1
main()
{
   char buf[BUFSIZE];
   int fdr, fdw, n;

   if ((fdr = open("foo.txt", O_RDONLY, 0)) < 0)
       exit(-1);
   if ((fdw = open("bar.txt", O_WRONLY, 0)) < 0)
       exit(-1);
   while ((n = read(fdr, buf, BUFSIZE)) > 0)
       write(fdw, buf, n);
   close(fdr);
   close(fdw);
}
```

Copy Program

Version 3

```
#define BUFSIZE 1
#define mode S_IRUSR|S_IWUSR|S_IRGRP|S_IWGRP|S_IROTH|S_IWOTH

main()
{
   char buf[BUFSIZE];
   int fdr, fdw, n;

   if ((fdr = open("foo.txt", O_RDONLY, 0)) < 0)
       exit(-1);
   if ((fdw = open("bar.txt", O_CREAT|O_WRONLY, mode)) < 0)
       exit(-1);
   while ((n = read(fdr, buf, BUFSIZE)) > 0)
       write(fdw, buf, n);
   close(fdr);
   close(fdw);
}
```