Communication via Network

- Clients and servers communicate by sending streams of bytes over connections:
  - Point-to-point, full-duplex, and reliable
- A socket is an endpoint of a connection
  - Socket address is an \(<\text{IP address} : \text{port}>\) pair
- 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 socket addresses of its endpoints (socket pair)
  - \(<\text{client IP}:\text{client port}, \text{server IP}:\text{server port}>\)
Communication via Network

Client socket address
128.2.194.242:51213

Server socket address
208.216.181.15:80

Client host address
128.2.194.242

Server host address
208.216.181.15

51213 is an ephemeral port allocated by OS

80 is a well-known port associated with Web servers
Most network applications are built on client-server model

- A server and one or more clients
- Clients and servers are processes running on hosts (can be the same or different hosts)
- Server provides service by manipulating resource for clients

Client process

1. Client sends request

2. Server handles request

3. Server sends response

4. Client handles response
Examples of client programs
- Web browsers, ftp, telnet, ssh

How does a client find servers?
- IP address in server socket address identifies host
- The (well-known) port in server socket address identifies service, and thus implicitly identifies server process that performs service

Examples of well-known ports (cf. /etc/services)
- Port 21: ftp
- Port 23: telnet
- Port 25: mail
- Port 80: web
Using Ports

Client host

Service request for 128.2.194.242:80 (i.e., the Web server)

Server host 128.2.194.242

Web server (port 80)

Echo server (port 7)

Client

Service request for 128.2.194.242:7 (i.e., the echo server)

Web server (port 80)

Echo server (port 7)
Servers

- Servers are long-running processes or daemons
  - Usually initiated during booting procedure
  - 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 21: ftp 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”
Sockets

- **Sockets interface**
  - Introduced in BSD4.1 UNIX, 1981.
  - Provides a user-level interface to network.
  - Based on client-server paradigm
  - Two types of transport service
    - Unreliable datagram (UDP)
    - Reliable and connection-oriented byte stream (TCP)
  - Underlying basis for all Internet applications
Sockets

- **What is a socket?**
  - An interface to network from applications (a “door”)
    - To an OS, a socket is an endpoint of communication
    - To an application, a socket is a file descriptor
      - Applications read/write from/to network using file descriptor
  - Clients and servers communicate with each by reading from and writing to socket descriptors
    - Main distinction between regular file I/O and socket I/O is how applications “open” file or socket descriptors
Hardware/software organization of networking applications

Sockets interface (system calls)

Hardware interface (interrupts)

User code

Kernel code

Hardware and firmware

Global IP Internet
Sockets

Connection-oriented service

Server
- socket()
- bind()
- listen()
- accept()
- read()
- write()

Client
- socket()
- connect()
- write()
- read()

Connectionless service

Server
- socket()
- bind()
- sendto()

Client
- socket()
- bind()
- sendto()
- recvfrom()
- read()
Creation of a Socket

- `socket(2)` creates a socket and returns a socket descriptor, which is an ordinary file descriptor

- **Prototype**

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

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

- `protocol` is usually 0 to let system choose an appropriate protocol for `domain`

- You can close SD with `close()`
Socket Address Structure

- **Generic socket address**
  - For address arguments to `connect()`, `bind()`, and `accept()`
    ```c
    struct sockaddr {
      unsigned short sa_family;  /* protocol family */
      char sa_data[14];          /* address data. */
    };
    ```

- **Internet-specific socket address**
  - Must cast `(sockaddr_in *)` to `(sockaddr *)` for `connect()`, `bind()`, and `accept()`
    ```c
    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) */
    };
    ```
Connecting to a Server

 Prototype

```c
#include <sys/types.h>    /* See NOTES */
#include <sys/socket.h>
int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

- To establish a TCP connection to a server
  - `servaddr` contains `<IP address, port number>` of server
  - The client does not have to call `bind()` before calling `connect()`
    - The kernel will choose both an ephemeral port and source IP address if necessary
  - Client process blocks until connection establishes
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>

#define MAXLINE 80

int main (int argc, char *argv[]) {
    int n, cfd;
    struct hostent *h;
    struct sockaddr_in saddr;
    char buf[MAXLINE];
    char *host = argv[1];
    int port = atoi(argv[2]);

    if ((cfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("socket() failed.\n");
        exit(1);
    }
if ((h = gethostbyname(host)) == NULL) {
    printf("invalid hostname %s\n", host);
    exit(2);
}

bzero((char *)&saddr, sizeof(saddr));
saddr.sin_family = AF_INET;
bcopy((char *)h->h_addr, (char *)&saddr.sin_addr.s_addr, h->h_length);
saddr.sin_port = htons(port);

if (connect(cfd,(struct sockaddr *)&saddr,sizeof(saddr)) < 0) {
    printf("connect() failed.\n");
    exit(3);
}

while ((n = read(0, buf, MAXLINE)) > 0) {
    write(cfd, buf, n);
    n = read(cfd, buf, MAXLINE);
    write(1, buf, n);
}
close(cfd);
Binding a Port to a Socket

- `bind(2)` call assigns a local address (or name) to socket descriptor
- Client process will use this same address to connect to this socket
- Prototype

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

int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```
Activation of a Socket

- To activate a socket, use `listen(2)`

**Prototype**

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

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

- This sets maximum number of pending connections the system allows before refusing connection
- `listen()` is not need for UDP
Accepting a Client

Prototype

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

int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

- **family** specifies the protocol family.
  - AF_UNIX: Local Unix domain protocols
  - AF_INET: IPv4 Internet protocols

- **type** specifies the communication semantics.
  - SOCK_STREAM
  - SOCK_DGRAM
  - SOCK_RAW: provides raw network protocol access

- **protocol** specifies a particular protocol for the socket
Accepting a Client

- `accept()` blocks until a client connects
- It fills in a `sockaddr` structure with client address
- It returns a new socket, communication socket
- A communication socket will be assigned to each client
  - A single listening descriptor can fork many connected descriptors
- All actual data transfer will be done via communication sockets
Accepting a Client

1. Server blocks in `accept`, waiting for connection request on listening descriptor `listenfd`

2. Client makes connection request by calling and blocking in `connect`

3. Server returns `connfd` from `accept`. Client returns from `connect`. Connection is now established between `clientfd` and `connfd`
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <arpa/inet.h>

#define MAXLINE 80

int main (int argc, char *argv[]) {
    int n, listenfd, connfd, caddrlen;
    struct hostent *h;
    struct sockaddr_in saddr, caddr;
    char buf[MAXLINE];
    int port = atoi(argv[1]);

    if ((listenfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("socket() failed.\n");
        exit(1);
    }
bzero((char *)&saddr, sizeof(saddr));
saddr.sin_family = AF_INET;
saddr.sin_addr.s_addr = htonl(INADDR_ANY);
saddr.sin_port = htons(port);
if (bind(listenfd, (struct sockaddr *)&saddr,
        sizeof(saddr)) < 0) {
    printf("bind() failed.\n");
    exit(2);
}
if (listen(listenfd, 5) < 0) {
    printf("listen() failed.\n");
    exit(3);
}
while (1) {
    caddrlen = sizeof(caddr);
    if ((connfd = accept(listenfd, (struct sockaddr *)&caddr,
                   &caddrlen)) < 0) {
        printf ("accept() failed.\n");
        continue;
    }
h = gethostbyaddr((const char *)&caddr.sin_addr.s_addr, sizeof(caddr.sin_addr.s_addr), AF_INET);
printf("server connected to %s (%s)\n", h->h_name, inet_ntoa(*(struct in_addr *)&caddr.sin_addr));

// echo
while ((n = read(connfd, buf, MAXLINE)) > 0) {
    printf ("got %d bytes from client.\n", n);
    write(connfd, buf, n);
}

printf("connection terminated.\n");
close(connfd);
Echo Service

Client
- socket
- connect
- write
- read
- close

Server
- socket
- bind
- listen
- accept
- read
- write
- read
- close

Await connection request from next client