ECE 356/COMPSI 356
Computer Network Architecture

Lab Introduction & Sockets

Monday September 2nd, 2019

Lecture Outline

- Labs: an introduction
- Introduction to sockets
- Socket interface
- Example client-server application
- Host and network byte orders

Lab Overview

- Three labs
  - An echo server (10 pts, individual)
  - A simple router (15 pts, group)
  - Dynamic routing (15 pts, group)
- C/C++

Set up the Lab Environment

1. Download and install VirtualBox
2. Install the provided virtual machine image
   - Wireshark
   - Mininet
3. Write your code in your favorite editor
4. Compile, debug
   - *printf* is your best friend

Labs and Plagiarism

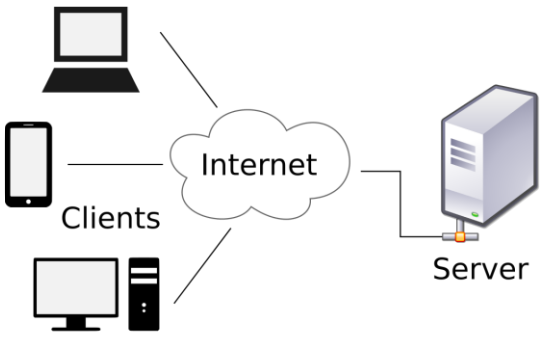
- Discussions are encouraged
- Code needs to be written by the individual/group
- We use code similarity checkers to detect plagiarism
  - Plagiarized assignments result in a failing grade for the course

Lab 1 (1/2)

- This lab needs to be done *individually*
- Reference textbook material: **PD 1.4**
- Submit via Sakai by 11:59 PM Wednesday September 11th
- Hints:
  - Start early
  - Pay attention to the requirements

Lab 1 (2/2)

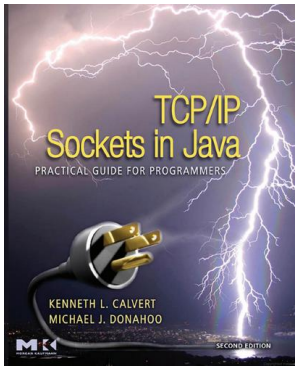
- Write an *echo server* using *TCP sockets*
  - Client/server architecture
  - Client is provided
  - You need to write the server

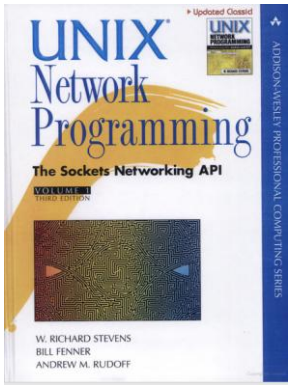


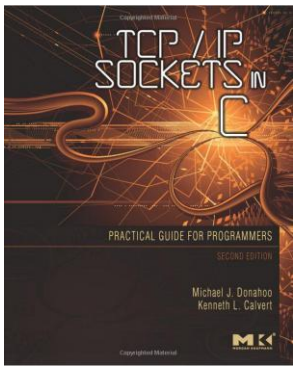
Lecture Outline

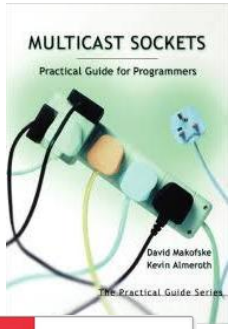
- Labs: an introduction
- **Introduction to sockets**
- Socket interface
- Example client-server application
- Host and network byte orders

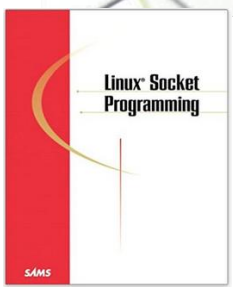
Network Sockets











9

Duke UNIVERSITY

Sockets

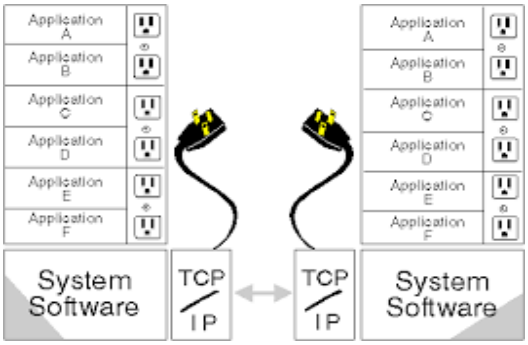
- Provide a form of interprocess communications
- Used to send messages across a network
  - ➢ Most common types of socket applications: client-server applications
- Originated in ARPANET in 1971

10

Duke UNIVERSITY

Socket

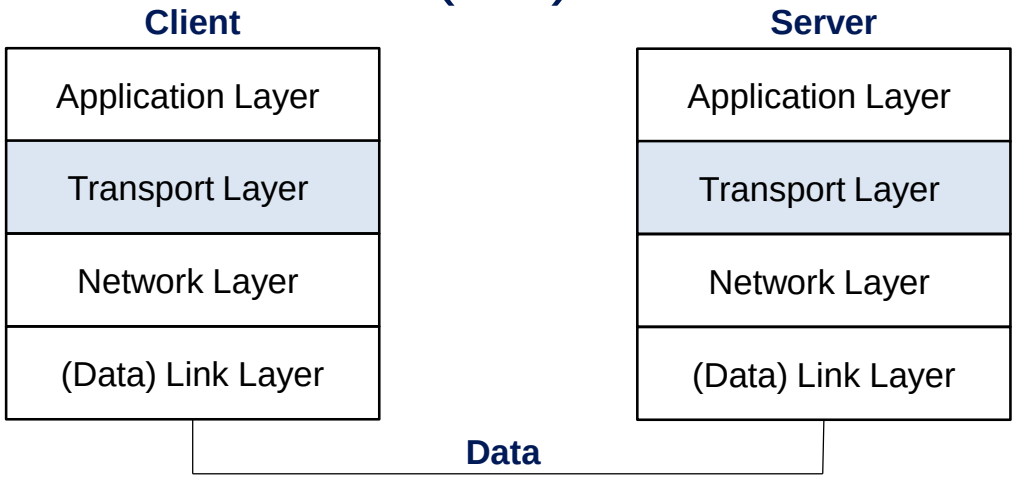
- What is a socket?
  - An interface between an application and the network
  - The point where a local application process attaches to the network
- An application creates the socket



- Sockets are specific to a node and are not externally addressable

Duke UNIVERSITY

Primarily Used in the Transport Layer (1/2)



Duke UNIVERSITY

Primarily Used in the Transport Layer (2/2)

- Lower-layer capabilities often keep track of active socket pairs
  - Firewalls

Lecture Outline

- Labs: an introduction
- Introduction to sockets
- **Socket interface**
- Example client-server application
- Host and network byte orders

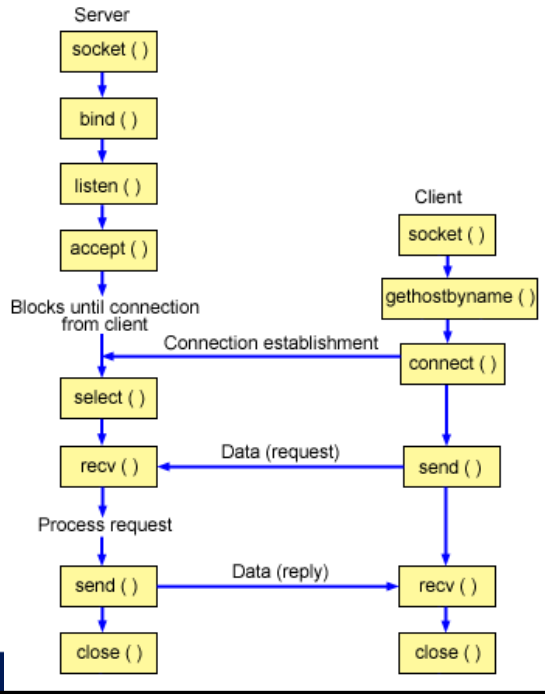
Application Programming Interface (Sockets)

- Each protocol provides a certain set of *services*, and the API provides a syntax by which those services can be invoked in this particular OS
- **Socket Interface** was originally provided by the Berkeley Software Distribution (BSD) of Unix
  - Now supported in virtually all operating systems
  - Easier to port applications between different OSs

Socket Interface

- The interface defines operations for
  - Creating a socket
  - Attaching a socket to the network
  - Sending and receiving messages through the socket
  - Closing the socket

Connection-oriented Example (TCP)



Sockets

- Socket Family
  - PF_INET denotes the Internet family
  - PF_UNIX denotes the Unix pipe facility
  - PF_PACKET denotes direct access to the network interface (i.e., it bypasses the TCP/IP protocol stack)
- Socket Type
  - SOCK_STREAM is used to denote a byte stream
  - SOCK_DGRAM is an alternative that denotes a message oriented service, such as that provided by UDP

Creating a Socket (1/2)

```
int sockfd = socket(address_family, type,
  protocol);
```

- The socket number returned is the socket descriptor for the newly created socket

Creating a Socket (2/2)

- `int sockfd = socket(PF_INET, SOCK_STREAM, 0);`
- `int sockfd = socket(PF_INET, SOCK_DGRAM, 0);`

- The combination of PF_INET and SOCK_STREAM implies TCP
- The combination of PF_INET and SOCK_DGRAM implies UDP

Client-Server Model with TCP: Server

- Server: passive open
  - Prepares to accept connection, does not actually establish a connection
- Server invokes:

```
int bind(int socket, struct sockaddr
*address, int addr_len)
int listen(int socket, int backlog)
int accept(int socket, struct sockaddr
*address,int *addr_len)
```


Client-Server Model with TCP: Bind

- Binds the newly created socket to the specified address, i.e. the network address of the local participant (the server)
  - Socket address: a combination of an IP address and a port number

Client-Server Model with TCP: Listen

```
int listen(int socket, int backlog)
```

- Defines how many connections can be pending on the specified socket

Client-Server Model with TCP: Accept (1/2)

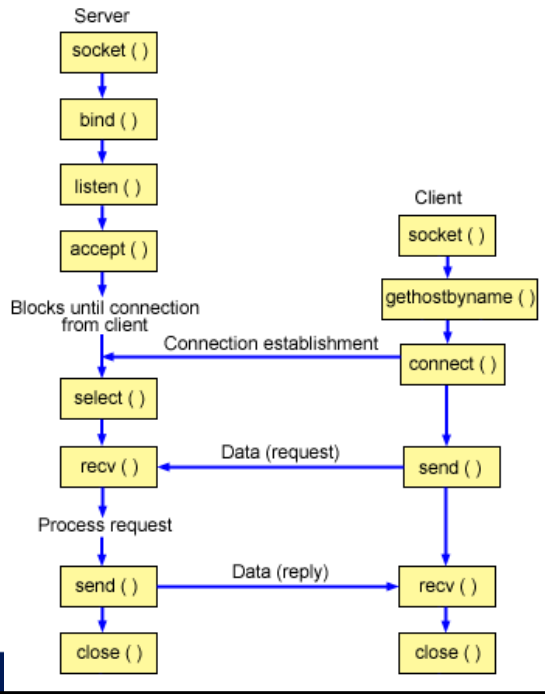
```
int accept(int socket, struct sockaddr
*address,int *addr_len)
```

- Carries out the passive open
- Blocking operation
  - Does not return until a remote participant has established a connection

Client-Server Model with TCP: Accept (2/2)

- Returns a *new socket* that corresponds to the new established connection
  - The address argument contains the remote participant's address
- Original socket still exists, used in future invocations of accept

Connection-oriented Example (TCP): Client



Client-Server Model with TCP: Client

Client: application performs active open

- It says who it wants to communicate with

Client invokes

```
int connect(int socket, struct sockaddr
*address, int addr_len)
```


Client-Server Model with TCP: Connect

- Does not return until TCP has successfully established a connection at which application is free to begin sending data
- Address contains remote machine's address

Client-Server Model with TCP: In practice

- The client usually specifies only remote participant's address and lets the system fill in the local information
- A server usually listens for messages on a well-known port
- A client does not care which port it uses for itself, the OS simply selects an unused one

Client-Server Model with TCP: Sending and Receiving

Once a connection is established, the application process invokes two operations:

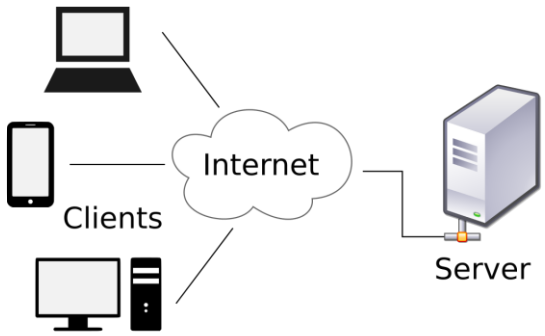
```
int send(int socket, char *msg, int
msg_len, int flags)
int recv(int socket, char *buff, int
buff_len, int flags)
```

Lecture Outline

- Labs: an introduction
- Introduction to sockets
- Socket interface
- **Example client-server application**
- Host and network byte orders

A Simple Talk Program

- Display at the server what is transmitted by the client



Example Application: Client

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

#define SERVER_PORT 5432
#define MAX_LINE 256

int main(int argc, char * argv[])
{
    FILE *fp;
    struct hostent *hp;
```

1/4

```
    struct sockaddr_in sin;
    char *host;
    char buf[MAX_LINE];
    int s;
    int len;
    if (argc==2) {
     host = argv[1];
    }
    else {
     fprintf(stderr, "usage: simplex-talk
    host\n");
    exit(1);
    }
```

2/4

Duke UNIVERSITY

Example Application: Client

```
    /* translate host name into peer's IP
    address */
    hp = gethostbyname(host);
    if (!hp) {
     fprintf(stderr, "simplex-talk:
    unknown host: %s\n", host);
     exit(1);
    }
    /* build address data structure */
    bzero((char *)&sin, sizeof(sin));
    sin.sin_family = AF_INET;
    bcopy(hp->h_addr, (char
    *)&sin.sin_addr, hp->h_length);
    sin.sin_port = htons(SERVER_PORT);
}
```

3/4

```
    /* active open */
    if ((s = socket(PF_INET, SOCK_STREAM,
    0)) < 0) {
     perror("simplex-talk: socket");
     exit(1);
    }
    if (connect(s, (struct sockaddr *)&sin,
    sizeof(sin)) < 0) {
     perror("simplex-talk: connect");
     close(s);
     exit(1);
    }
    /* main loop: get and send lines of text
    */
    while (fgets(buf, sizeof(buf), stdin)) {
     buf[MAX_LINE-1] = '\0';
     len = strlen(buf) + 1;
     send(s, buf, len, 0);
    }
}
```

4/4

Duke UNIVERSITY

Example Application: Server

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#define SERVER_PORT 5432
#define MAX_PENDING 5
#define MAX_LINE 256

int main()
{
     struct sockaddr_in sin;
     char buf[MAX_LINE];
     int len;
     int s, new_s;
```

1/4

```
    /* build address data structure */
    bzero((char *)&sin, sizeof(sin));
    sin.sin_family = AF_INET;
    sin.sin_addr.s_addr = INADDR_ANY;
    sin.sin_port = htons(SERVER_PORT);

    /* setup passive open */
    if ((s = socket(PF_INET, SOCK_STREAM, 0)) <
    0) {
     perror("simplex-talk: socket");
     exit(1);
    }
```

2/4

Duke UNIVERSITY

Example Application: Server

```
    if ((bind(s, (struct sockaddr *)&sin,
    sizeof(sin))) < 0) {
     perror("simplex-talk: bind");
     exit(1);
    }
    listen(s, MAX_PENDING);
    /* wait for connection, then receive and
    print text */
    while(1) {
     if ((new_s = accept(s, (struct sockaddr
    *)&sin, &len)) < 0) {
     perror("simplex-talk: accept");
     exit(1);
    }
```

3/4

```
    while (len = recv(new_s, buf,
    sizeof(buf), 0))
     fputs(buf, stdout);
     close(new_s);
    }
}
```

4/4

Duke UNIVERSITY

Lecture Outline

- Labs: an introduction
- Introduction to sockets
- Socket interface
- Example client-server application
- **Host and network byte orders**

Socket Address Structs

- Internet-specific socket address

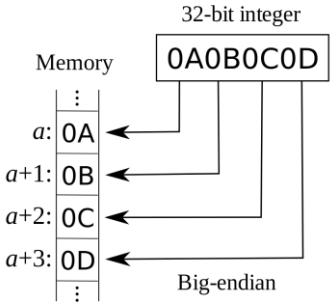
```
#include <netinit/in.h>

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) */
};
```

Big and Little Endian (1/2)

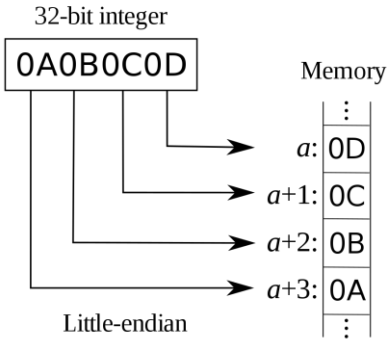
- Describe the order in which a sequence of bytes is stored in memory

- Big Endian Byte Order
  - The **most significant** byte (the "big end") of the data is placed first
  - IBM mainframes, some microcontrollers
  - **Network byte order in TCP/IP**



Big and Little Endian (2/2)

- Little Endian Byte Order
  - The **least significant** byte (the "little end") of the data is placed first
  - Most modern computers
  - **Host byte order** is usually different from network byte order



Converting Between Host and Network Byte Orders

- `uint32_t htonl(uint32_t hostlong);`
- `uint16_t htons(uint16_t hostshort);`
- `uint32_t ntohl(uint32_t netlong);`
- `uint16_t ntohs(uint16_t netshort);`

Lecture Summary

- Labs: an introduction
- Introduction to sockets
- Socket interface
- Example client-server application
- Host and network byte orders