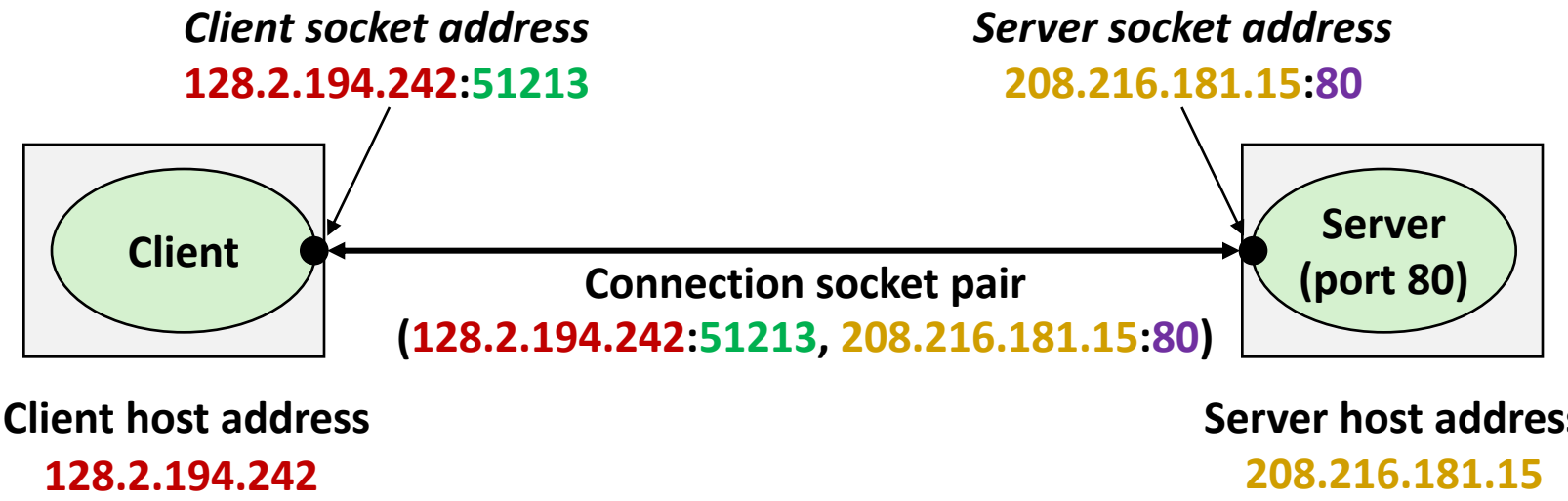
Network Programming

Eunji Lee
(ejlee@ssu.ac.kr)

Putting it all Together: Anatomy of an Internet Connection



51213 is an ephemeral port allocated by the kernel

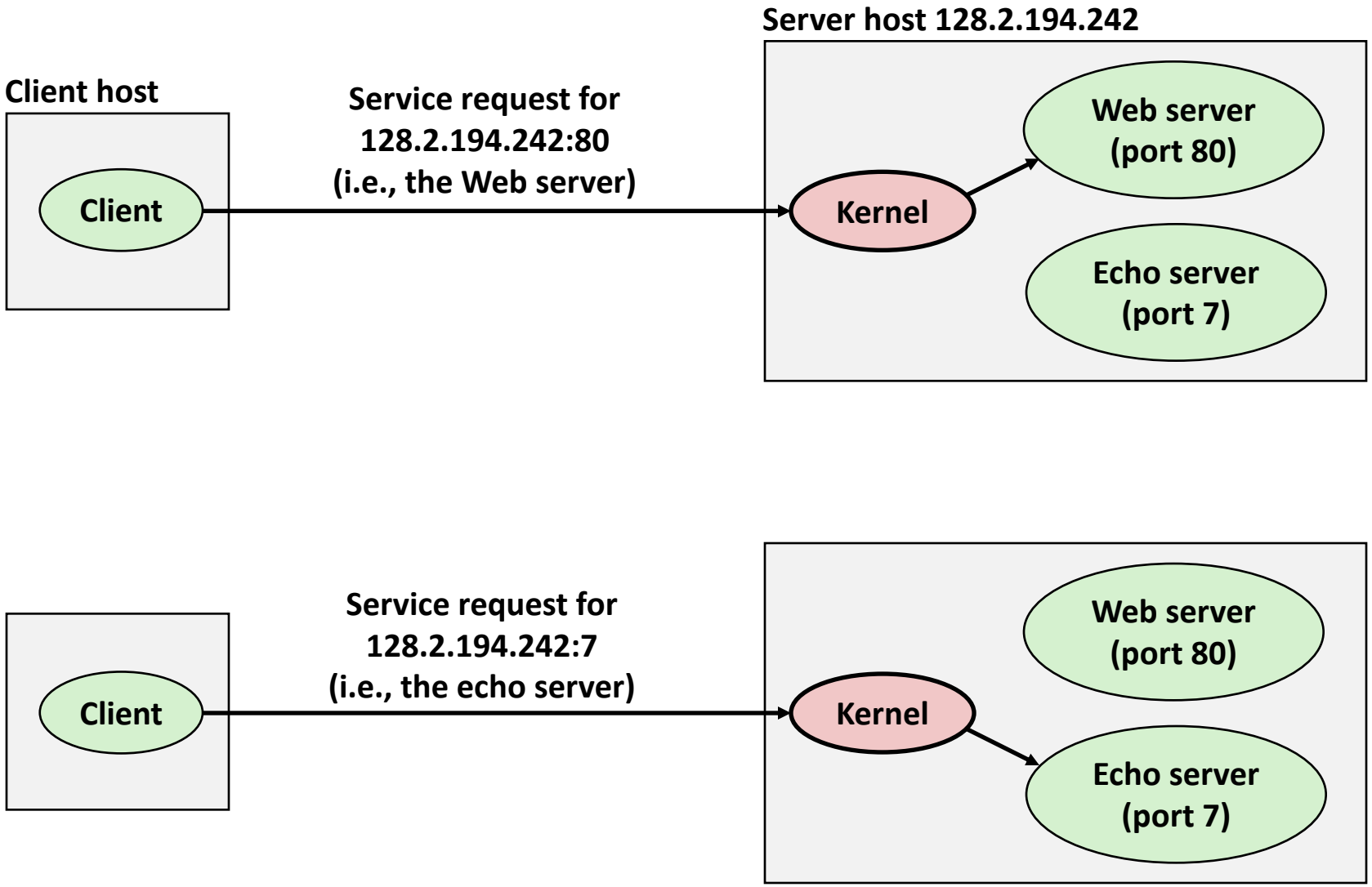
80 is a well-known port associated with Web servers

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

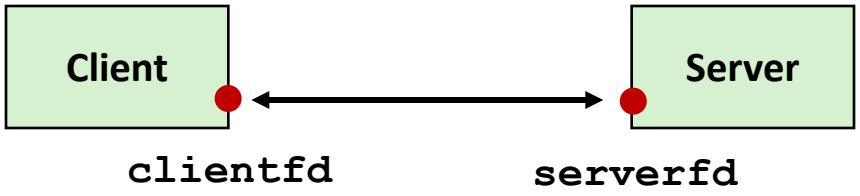
See `/etc/services` for a comprehensive list of the port mappings on a Linux machine

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

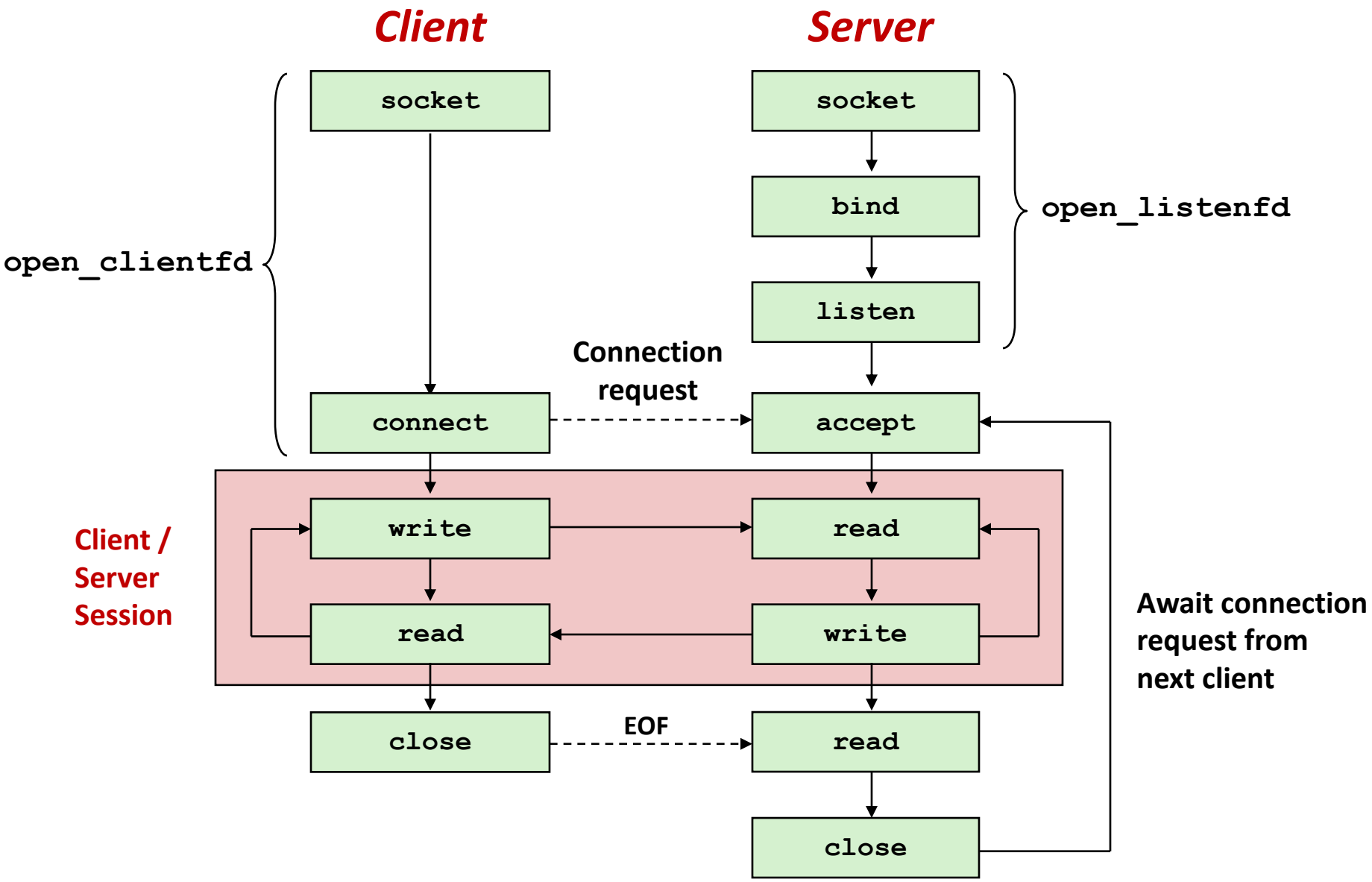
Sockets

- **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**

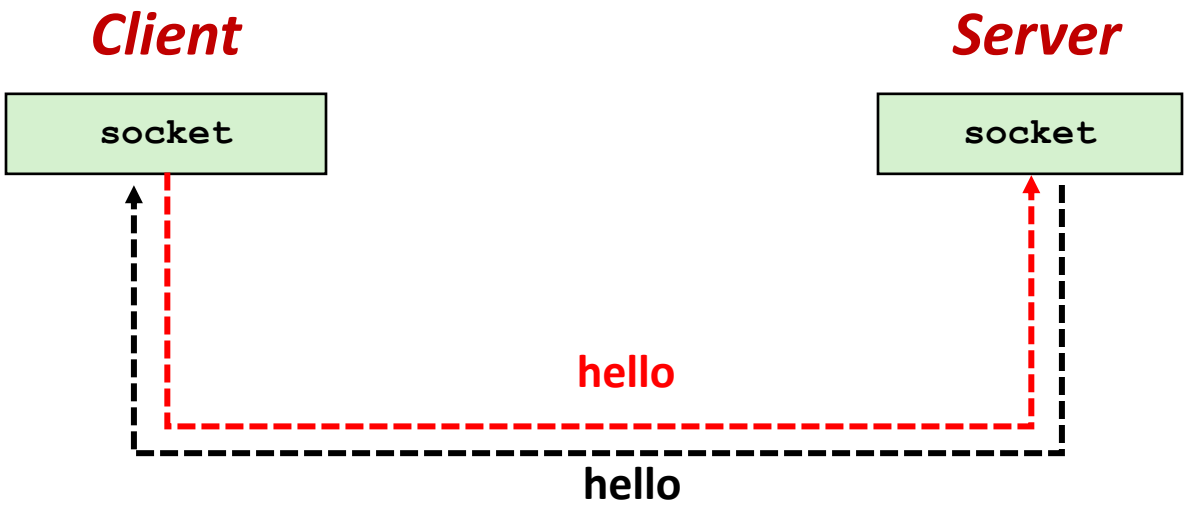


- **The main distinction between regular file I/O and socket I/O is how the application "opens" the socket descriptors**

Overview of the Sockets Interface



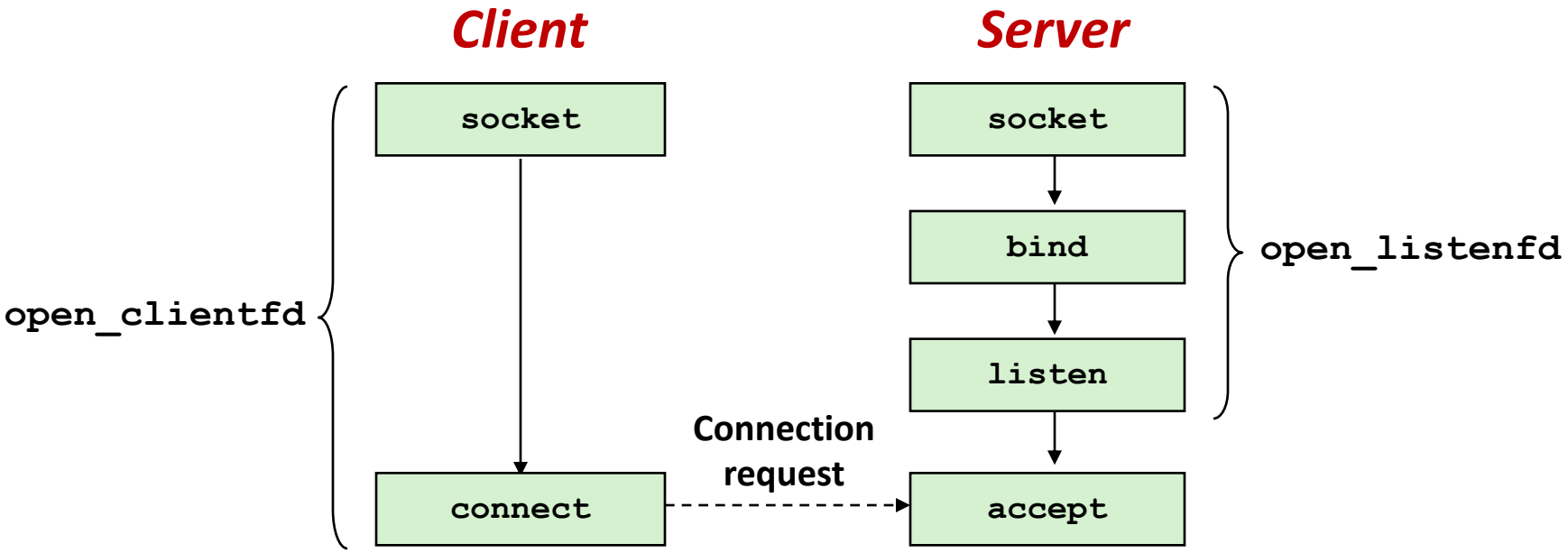
Example: Echo Client / Server




```
(base) ejlee@datos-SYS-7049GP-TRT:~/lecture/sp_prof/net_lab$ ./client 127.0.0.1 8080
fd = 3
hello
hello
world
world
What a small world
What a small world
```

```
(base) ejlee@datos-SYS-7049GP-TRT:~/lecture/sp_prof/net_lab$ ./server 8080
Waiting for request ...
Connected to 127.0.0.1
server received 6 bytes
server received 6 bytes
server received 19 bytes
```

Overview of the Sockets Interface



- **Office Telephone Analogy for Server**
  - Socket: Buy a phone
  - Bind: Tell the local administrator what number you want to use
  - Listen: Plug the phone in
  - Accept: Answer the phone when it rings

Echo Server

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<arpa/inet.h>

#define LISTENQ 1024
#define MAXLINE 1024

int main(int argc, char **argv)
{
    int listenfd, connfd, port, clientlen;
    struct sockaddr_in serveraddr, clientaddr;
    char buf[MAXLINE];

    // Create a socket
    if ((listenfd = socket(AF_INET, SOCK_STREAM, 0)) < 0){
        perror("socket");
        exit(-1);
    }

    // Associate the socket with a pair of address and port number
    port = atoi(argv[1]);
    serveraddr.sin_family = AF_INET;
    serveraddr.sin_addr.s_addr = htonl(INADDR_ANY);
    serveraddr.sin_port = htons((unsigned short)port);

    if (bind(listenfd, (struct sockaddr*)&serveraddr, sizeof(serveraddr)) == -1){
        perror("Failed to bind");
        return -1;
    }

    // Start waiting
    if(listen(listenfd, LISTENQ) < 0)
       return -1;

    // Wait for connection request
    printf("Waiting for request ... \n");

    clientlen = sizeof(clientaddr);
    if((connfd = accept(listenfd, (struct sockaddr*)&clientaddr, &clientlen)) < 0){
        perror("accept");
        exit(-1);
    }
```

```c

    char* caddrp = inet_ntoa(clientaddr.sin_addr);
    printf("Connected to %s\n", caddrp);

    int n;
    while(1) {
        n = read(      , buf, MAXLINE);
        if (n == 0) break;
        printf("server received %d bytes\n", n);
        write(      , buf, n);
    }
    return 0;
}
```

`socket` function

- **Create a socket descriptor**

```
#include <sys/types.h>
#include <sys/socket.h>

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

Returns: nonnegative descriptor if OK, −1 on error

- **Client**

```
clientfd = Socket(AF_INET, SOCK_STREAM, 0);
```

  - Just allocates & initializes some internal data structures
  - AF_INET: indicates that the socket is associated with Internet protocols
  - SOCK_STREAM: selects a reliable byte stream connection provided by TCP

`bind` function

- Associate socket address in `my_addr` with the socket descriptor sockfd

```
#include <sys/socket.h>

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

Returns: 0 if OK, −1 on error

```
772
773     /* Listenfd will be an endpoint for all requests to port
774        on any IP address for this host */
775     bzero((char *) &serveraddr, sizeof(serveraddr));
776     serveraddr.sin_family = AF_INET;
777     serveraddr.sin_addr.s_addr = htonl(INADDR_ANY);
778     serveraddr.sin_port = htons((unsigned short)port);
779     if (bind(listenfd, (SA *)&serveraddr, sizeof(serveraddr)) < 0)
780         return -1;
```

`listen` function

- `listen` indicates that this socket will accept connection (`connect`) requests from clients

```
#include <sys/socket.h>

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

Returns: 0 if OK, −1 on error

- `LISTENQ` is constant indicating how many pending requests allowed

- We're finally ready to enter the main server loop that accepts and processes client connection requests.

`accept` function

- **`accept()` blocks waiting for a connection request**

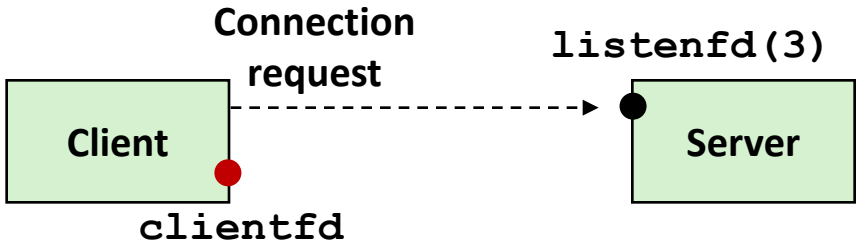
```
#include <sys/socket.h>

int accept(int listenfd, struct sockaddr *addr, int *addrlen);
                    Returns: nonnegative connected descriptor if OK, −1 on error
```

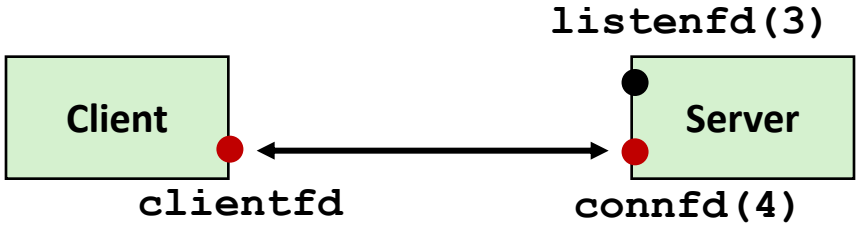
- **`accept` returns a *connected descriptor* (`connfd`) with the same properties as the *listening descriptor* (`listenfd`)**
  - Returns when the connection between client and server is created and ready for I/O transfers
  - All I/O with the client will be done via the connected socket
- **`accept` also fills in client's IP address**

Echo Server: `accept` Illustrated

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`

Connected vs. Listening Descriptors

- **Listening descriptor**
  - End point for client connection requests
  - Created once and exists for lifetime of the server

- **Connected descriptor**
  - End point of the connection between client and server
  - A new descriptor is created each time the server accepts a connection request from a client
  - Exists only as long as it takes to service client

- **Why the distinction?**
  - Allows for concurrent servers that can communicate over many client connections simultaneously
    - E.g., Each time we receive a new request, we fork a child to handle the request

Echo Client

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<unistd.h>
#include<arpa/inet.h>

#define MAXLINE 1024

int main(int argc, char** argv)
{
    int clientfd, port;
    struct sockaddr_in serveraddr;

    char *host, buf[MAXLINE];

    host = argv[1];
    port = atoi(argv[2]);

    if((clientfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("socket");
        exit(1);
    }

    serveraddr.sin_family = AF_INET;
    serveraddr.sin_port = htons((unsigned short)port);
    serveraddr.sin_addr.s_addr = inet_addr(host);

    if(connect(clientfd, (struct sockaddr*)&serveraddr, sizeof(serveraddr)) < 0) {
        perror("connect");
        exit(1);
    }

    int rbytes, wbytes;
    while(fgets(buf, MAXLINE, stdin) != NULL) {
        wbytes = write(clientfd, buf, strlen(buf));
        if(wbytes < strlen(buf))
            printf("Failed to send message\n");
        rbytes = read(clientfd, buf, MAXLINE);
        fputs(buf, stdout);
    }

    close(clientfd);
    return 0;

}
```

`connect` function

- Establish an Internet connection with the server at socket address `serv_addr`

```
#include <sys/socket.h>

int connect(int sockfd, struct sockaddr *serv_addr, int addrlen);
```

Returns: 0 if OK, −1 on error

sockaddr: socketbits.h (included by socket.h), sockaddr_in: netinet/in.h

```
/* Generic socket address structure (for connect, bind, and accept) */
struct sockaddr {
    unsigned short  sa_family;   /* Protocol family */
    char            sa_data[14]; /* Address data.  */
};

/* Internet-style socket address structure */
struct sockaddr_in  {
    unsigned short  sin_family;  /* Address family (always AF_INET) */
    unsigned short  sin_port;    /* Port number in network byte order */
    struct in_addr  sin_addr;    /* IP address in network byte order */
    unsigned char   sin_zero[8]; /* Pad to sizeof(struct sockaddr) */
};
```

sockaddr: socketbits.h (included by socket.h), sockaddr_in: netinet/in.h

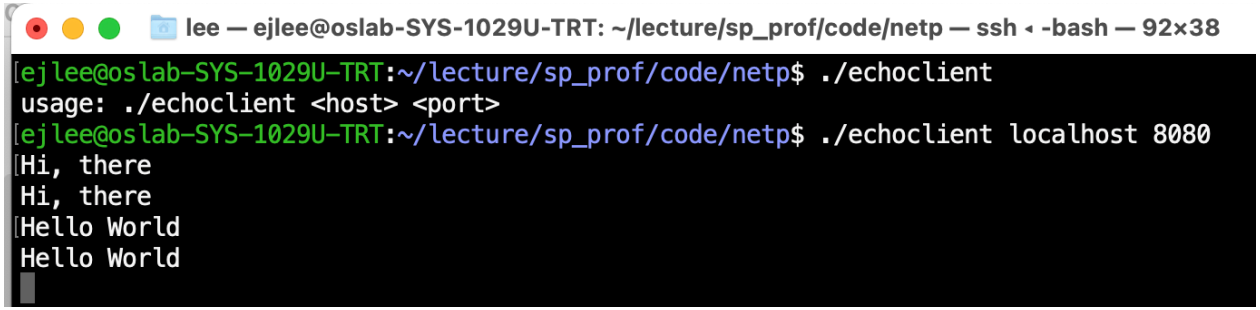
For More Information

- **W. Richard Stevens, "Unix Network Programming: Networking APIs: Sockets and XTI", Volume 1, Second Edition, Prentice Hall, 1998**
  - THE network programming bible
- **Unix Man Pages**
  - Good for detailed information about specific functions
- **Complete versions of the echo client and server are developed in the text**
  - Updated versions linked to course website
  - Feel free to use this code in your assignments

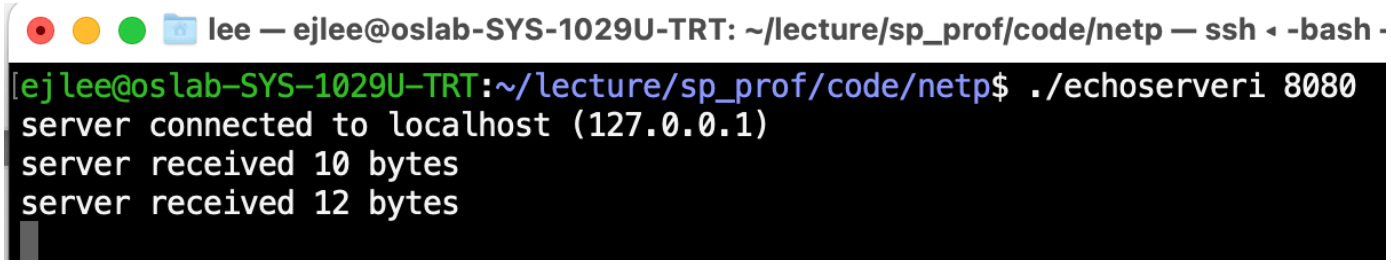
(Backups) Execution of Echo Server / Client

- Source codes
  - http://csapp.cs.cmu.edu/3e/code.html

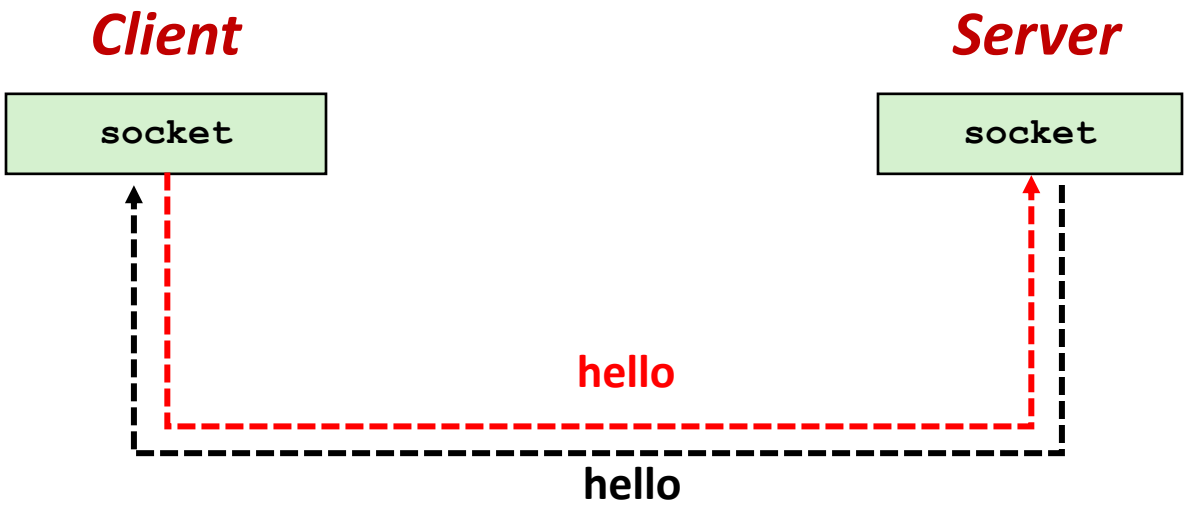
- Client



- Server



(Backups) Example: Echo Client / Server




```
$ ./echoclient localhost 8080
Hi, there
Hi, there
Hello World
Hello World
```

```
$ ./echoserveri 8080
server connected to localhost
(127.0.0.1)
server received 10 bytes
server received 12 bytes
```

(Backups) Echo Client: Main Routine

```c
/*
 * echoclient.c - An echo client
 */
/* $begin echoclientmain */
#include "csapp.h"

int main(int argc, char **argv)
{
    int clientfd, port;
    char *host, buf[MAXLINE];
    rio_t rio;

    if (argc != 3) {
        fprintf(stderr, "usage: %s <host> <port>\n", argv[0]);
        exit(0);
    }
    host = argv[1];
    port = atoi(argv[2]);

    clientfd = Open_clientfd(host, port);
    Rio_readinitb(&rio, clientfd);

    while (Fgets(buf, MAXLINE, stdin) != NULL) {
        Rio_writen(clientfd, buf, strlen(buf));
        Rio_readlineb(&rio, buf, MAXLINE);
        Fputs(buf, stdout);
    }
    Close(clientfd); //line:netp:echoclient:close
    exit(0);
}
/* $end echoclientmain */
```

(Backups) Echo Client: open_clientfd

```
721  ********************************/
722 /*
723  * open_clientfd - open connection to server at <hostname, port>
724  *   and return a socket descriptor ready for reading and writing.
725  *   Returns -1 and sets errno on Unix error.
726  *   Returns -2 and sets h_errno on DNS (gethostbyname) error.
727  */
728 /* $begin open_clientfd */
729 int open_clientfd(char *hostname, int port)
730 {
731     int clientfd;
732     struct hostent *hp;
733     struct sockaddr_in serveraddr;
734
735     if ((clientfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
736     return -1; /* check errno for cause of error */
737
738     /* Fill in the server's IP address and port */
739     if ((hp = gethostbyname(hostname)) == NULL)
740     return -2; /* check h_errno for cause of error */
741     bzero((char *) &serveraddr, sizeof(serveraddr));
742     serveraddr.sin_family = AF_INET;
743     bcopy((char *)hp->h_addr_list[0],
744       (char *)&serveraddr.sin_addr.s_addr, hp->h_length);
745     serveraddr.sin_port = htons(port);
746
747     /* Establish a connection with the server */
748     if (connect(clientfd, (SA *) &serveraddr, sizeof(serveraddr)) < 0)
749     return -1;
750     return clientfd;
751 }
752 /* $end open_clientfd */
```

Create socket

Create address

Establish connection

(Backups) accept function

```c
/*
 * echoserveri.c - An iterative echo server
 */
/* $begin echoserverimain */
#include "csapp.h"

void echo(int connfd);

int main(int argc, char **argv)
{
    int listenfd, connfd, port, clientlen;
    struct sockaddr_in clientaddr;
    struct hostent *hp;
    char *haddrp;
    if (argc != 2) {
        fprintf(stderr, "usage: %s <port>\n", argv[0]);
        exit(0);
    }
    port = atoi(argv[1]);

    listenfd = Open_listenfd(port);
    while (1) {
        clientlen = sizeof(clientaddr);
        connfd = Accept(listenfd, (SA *)&clientaddr, &clientlen);

        /* determine the domain name and IP address of the client */
        hp = Gethostbyaddr((const char *)&clientaddr.sin_addr.s_addr,
               sizeof(clientaddr.sin_addr.s_addr), AF_INET);
        haddrp = inet_ntoa(clientaddr.sin_addr);
        printf("server connected to %s (%s)\n", hp->h_name, haddrp);

        echo(connfd);
        Close(connfd);
    }
    exit(0);
}
/* $end echoserverimain */
```

```
"echoserveri.c" 37L, 912C written
```

```c
#include <sys/socket.h>

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

Returns: nonnegative connected descriptor if OK, −1 on error

(Backups) connect function

```
#include <sys/socket.h>

int connect(int sockfd, struct sockaddr *serv_addr, int addrlen);
                                                    Returns: 0 if OK, −1 on error
```

```
/* Generic socket address structure
struct sockaddr {
    unsigned short  sa_family;   /*
    char            sa_data[14]; /*
};
```

```
struct sockaddr_in  {
    unsigned short  sin_family;
    unsigned short  sin_port;
    struct in_addr  sin_addr;
    unsigned char   sin_zero[8];
};
```

```
t <hostname, port>
reading and writing.

stbyname) error.
```

```
728 /* $begin open_clientfd */
729 int open_clientfd(char *hostname, int port)
730 {
731     int clientfd;
    struct hostent *hp;
    struct sockaddr_in serveraddr;

    if ((clientfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    return -1; /* check errno for cause of error */

738     /* Fill in the server's IP address and port */
739     if ((hp = gethostbyname(hostname)) == NULL)
740     return -2; /* check h_errno for cause of error */
741     bzero((char *) &serveraddr, sizeof(serveraddr));
742     serveraddr.sin_family = AF_INET;
743     bcopy((char *)hp->h_addr_list[0],
744       (char *)&serveraddr.sin_addr.s_addr, hp->h_length);
745     serveraddr.sin_port = htons(port);
746
747     /* Establish a connection with the server */
748     if (connect(clientfd, (SA *) &serveraddr, sizeof(serveraddr)) < 0)
749     return -1;
750     return clientfd;
751 }
752 /* $end open_clientfd */
```

(Backups) Echo Server : Main Routine

```
 1 /*
 2  * echoserveri.c - An iterative echo server
 3  */
 4 /* $begin echoserverimain */
 5 #include "csapp.h"
 6
 7 void echo(int connfd);
 8
 9 int main(int argc, char **argv)
10 {
11     int listenfd, connfd, port, clientlen;
12     struct sockaddr_in clientaddr;
13     struct hostent *hp;
14     char *haddrp;
15     if (argc != 2) {
16         fprintf(stderr, "usage: %s <port>\n", argv[0]);
17         exit(0);
18     }
19     port = atoi(argv[1]);
20
21     listenfd = Open_listenfd(port);
22     while (1) {
23         clientlen = sizeof(clientaddr);
24         connfd = Accept(listenfd, (SA *)&clientaddr, &clientlen);
25
26         /* determine the domain name and IP address of the client */
27         hp = Gethostbyaddr((const char *)&clientaddr.sin_addr.s_addr,
28                sizeof(clientaddr.sin_addr.s_addr), AF_INET);
29         haddrp = inet_ntoa(clientaddr.sin_addr);
30         printf("server connected to %s (%s)\n", hp->h_name, haddrp);
31
32         echo(connfd);
33         Close(connfd);
34     }
35     exit(0);
36 }
37 /* $end echoserverimain */

"echoserveri.c" 37L, 912C written
```

(Backups) Echo Server: open_listenfd

```
754 /*
755  * open_listenfd - open and return a listening socket on port
756  *     Returns -1 and sets errno on Unix error.
757  */
758 /* $begin open_listenfd */
759 int open_listenfd(int port)
760 {
761     int listenfd, optval=1;
762     struct sockaddr_in serveraddr;
763
764     /* Create a socket descriptor */
765     if ((listenfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
766         return -1;
767
768     /* Eliminates "Address already in use" error from bind. */
769     if (setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR,
770            (const void *)&optval , sizeof(int)) < 0)
771         return -1;
772
773     /* Listenfd will be an endpoint for all requests to port
774        on any IP address for this host */
775     bzero((char *) &serveraddr, sizeof(serveraddr));
776     serveraddr.sin_family = AF_INET;
777     serveraddr.sin_addr.s_addr = htonl(INADDR_ANY);
778     serveraddr.sin_port = htons((unsigned short)port);
779     if (bind(listenfd, (SA *)&serveraddr, sizeof(serveraddr)) < 0)
780         return -1;
781
782     /* Make it a listening socket ready to accept connection requests */
783     if (listen(listenfd, LISTENQ) < 0)
784         return -1;
785     return listenfd;
786 }
787 /* $end open_listenfd */
```

Create socket

Create address

bind & listen

(Backups) Echo Server: echo

- The server uses RIO to read and echo text lines

```
 1 /*
 2  * echo - read and echo text lines until client closes connection
 3  */
 4 /* $begin echo */
 5 #include "csapp.h"
 6
 7 void echo(int connfd)
 8 {
 9     size_t n;
10     char buf[MAXLINE];
11     rio_t rio;
12
13     Rio_readinitb(&rio, connfd);
14     while((n = Rio_readlineb(&rio, buf, MAXLINE)) != 0) {
15         printf("server received %ld bytes\n", n);
16         Rio_writen(connfd, buf, n);
17     }
18 }
19 /* $end echo */
20
```