# Clients

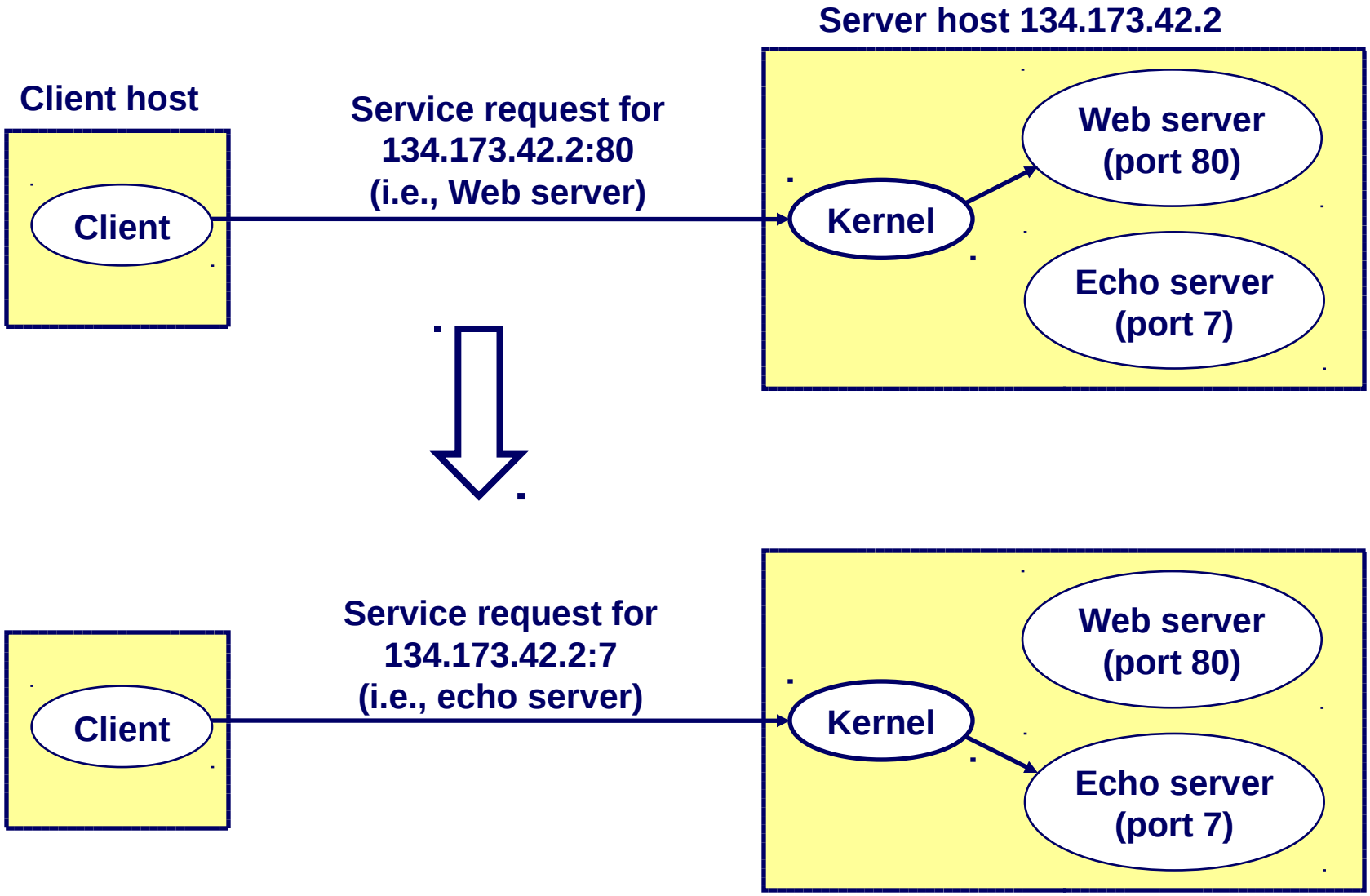

## Examples of client programs

- Web browsers, `ftp`, `telnet`, `ssh`

## How does a client find the server?

- IP address in server socket address identifies host *(more precisely, an adapter on the host)*
- (Well-known) port in server socket address identifies service, and thus implicitly identifies server process that provides it
- Examples of well-known ports
  - Port 7: Echo server
  - Port 22: ssh 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 `init` process (process 1)
- Run continuously until machine is turned off
- Or spawned by `inetd` in response to connection to port

**Each server waits for requests to arrive on well-known port associated with that particular service**

- Port 7: echo server
- Port 22: ssh server
- Port 25: mail server
- Port 80: HTTP server

**Machine that runs a server process is also often referred to as a "server"**

# Echo Client Main Routine


```
/* #include lots of stuff */

/* usage: ./echoclient host port */
int main(int argc, char **argv)
{
    int clientfd;
    size_t n;
    char *host, *port, buf[MAXLINE];

    host = argv[1];
    port = argv[2];

    if ((clientfd = open_clientfd(host, port)) == -1)
        exit(1);
    while (fgets(buf, sizeof buf - 1, stdin) != NULL) {
        write(clientfd, buf, strlen(buf));
        n = read(clientfd, buf, sizeof buf - 1);
        if (n >= 0) {
            buf[n] = '\0';
            fputs(buf, stdout);
        }
    }
    close(clientfd);
    exit(0);
}
```

# Echo Client: open_clientfd


```
int open_clientfd(char *hostname, char *port)
{
  int clientfd, error;
  struct addrinfo hints, *hostaddresses = NULL;

  /* Find out the server's IP address and port */
  memset(&hints, 0, sizeof hints);
  hints.ai_flags = AI_ADDRCONFIG | AI_V4MAPPED;
  hints.ai_family = AF_INET6;
  hints.ai_socktype = SOCK_STREAM;
  if (getaddrinfo(hostname, port, &hints, &hostaddresses) != 0)
    return -2;
  }

  /* We take advantage of the fact that AF_* and PF_* are identical */
  clientfd = socket(hostaddresses->ai_family,
   hostaddresses->ai_socktype, hostaddresses->ai_protocol);
  if (clientfd == -1)
    return -1; /* check errno for cause of error */
  /* Establish a connection with the server */
  if (connect(clientfd, hostaddresses->ai_addr, hostaddresses->ai_addrlen)
   == -1)
    return -1;
  freeaddrinfo(hostaddresses);
  return clientfd;
}
```

# Echo Server: Main Routine


```
int main(int argc, char **argv) {
    int listenfd, connfd, clientlen, error;
    char * port;
    union {struct sockaddr_in client4; struct sockaddr_in6 client6;
      } clientaddr;
    char hostname[NI_MAXHOST], hostaddr[NI_MAXHOST];

    listenfd = open_listenfd(argv[1]);
    if (listenfd < 0)
        exit(1);
    while (1) {
        clientlen = sizeof clientaddr;
        connfd = accept(listenfd, (struct sockaddr*)&clientaddr, &clientlen);
        if (connfd == -1)
            continue;
        error = getnameinfo((struct sockaddr*)&clientaddr, clientlen,
          hostname, sizeof hostname, NULL, 0, 0);
        if (error != 0)
          continue;
        getnameinfo((struct sockaddr*)&clientaddr, clientlen,
          hostaddr, sizeof hostaddr, NULL, 0, NI_NUMERICHOST);
        printf("server connected to %s (%s)\n", hostname, hostaddr);
        echo(connfd);
        close(connfd);
    }
}
```

# Echo Server: open_listenfd

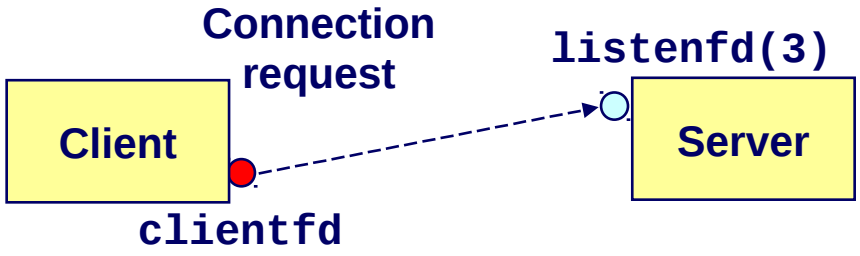

```
int open_listenfd(char *port)
{
    int listenfd, optval=1, error;
    struct addrinfo hints;
    struct addrinfo *hostaddresses = NULL;

    /* Find out the server's IP address and port */
    memset(&hints, 0, sizeof hints);
    hints.ai_flags = AI_ADDRCONFIG | AI_V4MAPPED | AI_PASSIVE;
    hints.ai_family = AF_INET6;
    hints.ai_socktype = SOCK_STREAM;
    error = getaddrinfo(NULL, port, &hints, &hostaddresses);
    if (error != 0)
        return -2;
    if ((listenfd = socket(hostaddresses->ai_family,
      hostaddresses->ai_socktype, hostaddresses->ai_protocol)) == -1)
        return -1;
    /* Eliminates "Address already in use" error from bind. */
    if (setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR,
                   (const void *)&optval , sizeof optval) == -1)
        return -1;
    /* Listenfd will be an endpoint for all requests to port */
    if (bind(listenfd, hostaddresses->ai_addr,
      hostaddresses->ai_addrlen) == -1)
        return -1;
    /* Make it a listening socket ready to accept
       connection requests */
    if (listen(listenfd, LISTEN_MAX) == -1)
        return -1;
    return listenfd;
}
```

# 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 server

## Connected descriptor

- End point of connection between client and server
- New descriptor is created each time server accepts 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 or spawn a thread to handle it
- Can be closed to break connection to particular client

# Testing Echo Server With `telnet`


```
mallet> ./echoserver 5000
server connected to bow-vpn.cs.hmc.edu (::ffff:192.168.6.5)
server received 5 bytes
server connected to bow-vpn.cs.hmc.edu (::ffff:192.168.6.5)
server received 8 bytes

bow> telnet mallet-vpn 5000
Trying 192.168.6.1...
Connected to mallet-vpn.
Escape character is '^]'.
123
123
Connection closed by foreign host.
bow> telnet mallet-vpn 5000
Trying 192.168.6.1...
Connected to mallet-vpn.
Escape character is '^]'.
456789
456789
Connection closed by foreign host.
bow>
```

# Running Echo Client and Server


```
mallet> echoserver 5000
server connected to bow-vpn.cs.hmc.edu (::ffff:192.168.6.5)
server received 4 bytes
server connected to bow-vpn.cs.hmc.edu (::ffff:192.168.6.5)
server received 7 bytes
...

bow> echoclient mallet-vpn 5000
123
123

bow> echoclient mallet-vpn 5000
456789
456789
bow>
```