

# Socket Programming (CSE 573S)

Ken Wong  
Washington University

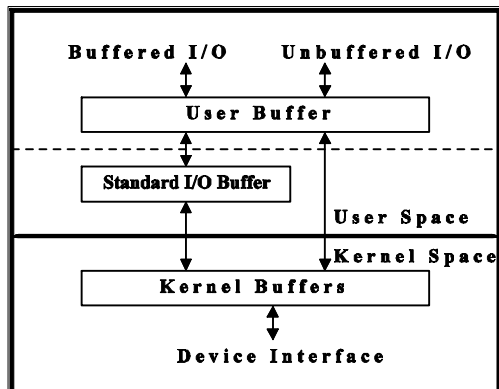
kenw@wustl.edu  
www.arl.wustl.edu/~kenw

## Overview

- Unix I/O is based on the concept of a byte stream file
- Basic Idea (Writing to a File)
  - » A file descriptor is a handle for a file

```
int fd;  
char msg[] = "hello\n";  
fd = open ("myfile", O_RDWR);  
write (fd, msg, 6);  
close (fd);
```

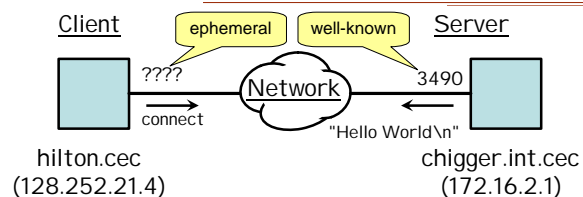
## I/O Buffering Options



## Client-Server Model

- References
  - » Donahoo and Calvert, Chapters 1-3 and Part 2 (API Ref.)
  - » <http://www.ecst.csuchico.edu/~beej/guide/net>
- A server waits to be contacted by a client
- Typical Scenario
  - » Start server on some host
    - Server initializes and waits for client to contact it for service
  - » Start client on any host
    - Client sends request for service to server
  - » Server provides service to client and waits for next client
    - *Iterative Server*: Server handles short service requests
    - *Concurrent Server*: Server invokes child process to handle request

## Example



- Well-Known Port Numbers (Services)
  - » Internet Assigned Numbers Authority (IANA)
  - » [www.iana.org/assignments/port-numbers](http://www.iana.org/assignments/port-numbers)
  - » /etc/services: Host file listing services and ports
- Ephemeral Port Number
  - » Port number assigned on demand by OS

5 - Ken Wong, 1/29/2004

Washington University in St. Louis

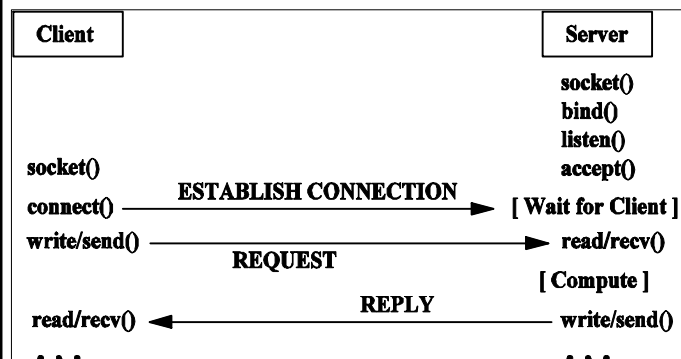
## Internet Sockets

- Socket: An IP address and a port number
  - » A connection is a socket pair
- Stream Socket
  - » Connection-oriented
  - » Reliable, ordered two-way communication stream
  - » Based on TCP (Transmission Control Protocol)
  - » SOCK\_STREAM packets
- Datagram Socket
  - » Connectionless
  - » Unreliable, unordered, packet-by-packet communication
  - » Based on UDP (User Datagram Protocol)
  - » SOCK\_DGRAM packets

6 - Ken Wong, 1/29/2004

Washington University in St. Louis

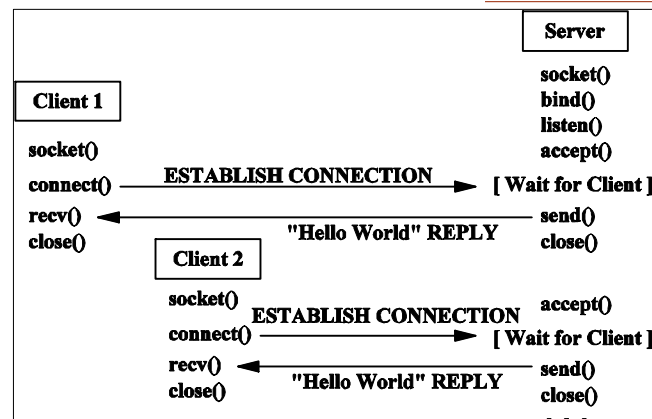
## Connection-Oriented Client-Server



7 - Ken Wong, 1/29/2004

Washington University in St. Louis

## Iterative Server (Example 1)



8 - Ken Wong, 1/29/2004

Washington University in St. Louis

## System Call Semantics

- socket: Create unnamed socket (data structure)
- bind: (Server) Bind name (IP address) to socket
- listen: Server is willing to accept connections
- accept: (Server) Get connect request from queue, complete new socket, and allocate file descriptor
- send: Send *n* bytes
- recv: Receive *at most* *n* bytes
- connect: (Client) I nitiate a connection on a socket

9 -Ken Wong, 1/29/2004

Washington University in St.Louis

## Example (Pseudo-Code)

- Client:        Create a socket;  
                  Connect to the server's port;  
                  Read message from server;  
                  Display message;  
                  Close socket;
- Server:        Create a socket;  
                  Bind name to socket;  
                  Listen for client connections;  
                  while (accept new connection) {  
                      Send message to client;  
                      Close socket;  
                  }

10 -Ken Wong, 1/29/2004

Washington University in St.Louis

## Standard Include Code (./stdinc.h)

```
#include <unistd.h>        // symbolic constants, sys svc protos
#include <stdio.h>        // standard I/O library
#include <stdlib.h>        // utility function protos
#include <errno.h>        // error handling (perror())
#include <string.h>        // char string functions
#include <strings.h>       // function protos like bzero()
#include <netdb.h>        // gethostbyname
#include <sys/types.h>    // socket, bind, accept, ... , send, recv
#include <netinet/in.h>   // socket address struct
#include <sys/socket.h>   // same as sys/types.h, listen
#include <arpa/inet.h>    // function protos like inet_ntoa()
static inline void
Fatal (char *msg) { fprintf(stderr, "%s", msg); exit(1); }
static inline void
FatalError (char *msg) { perror(msg); exit(1); }
```

11 -Ken Wong, 1/29/2004

Washington University in St.Louis

## Structures (1)

- <sys/socket.h>, a general socket address

```
struct sockaddr {
    u_short    sa_family;     // address family
    char    sa_data[14];     // up to 14 bytes of direct address
};
```
- <netinet/in.h>, a TCP/IP socket address

```
struct in_addr {            // Internet address
    u_long    S_addr;        // 32-bit netid/hostid (NBO)
};
struct sockaddr_in {
    short     sin_family;     // AF_INET
    u_short   sin_port;       // 16-bit port number (NBO)
    struct in_addr sin_addr;   // 32-bit netid/hostid (NBO)
    char      sin_zero[8];    // unused
};
```

12 -Ken Wong, 1/29/2004

Washington University in St.Louis

## Structures (2)

### ■ Returned by gethostbyname()

// <netdb.h>: Structures returned by network data base library.  
 // All addresses are supplied in host order, and returned in  
 // network order (suitable for use in system calls).

```
struct hostent {
    char    *h_name;    // official name of host
    char    **h_aliases; // alias list
    int     h_addrtype; // host address type
    int     h_length;   // length of address
    char    **h_addr_list; // list of addresses from name server
#define h_addr h_addr_list[0]
                // address (backward compatibility)
};
```

13 -Ken Wong, 1/29/2004

Washington University in St. Louis

## Client Code (Part 1)

```
#include "stdinc.h"
const int    PORT = 3490;    // server's port
const int    MAXDATASIZE = 100; // input buffer size
int main (int argc, char *argv[]) {
    char    *usage = "Usage: client Hostname\n";
    int     sd, numbytes;
    char    buf[MAXDATASIZE+1];
    struct hostent    *he;
    struct sockaddr_in    srvAddr;    // server's address
    if (argc != 2)    Fatal(usage);
    if ((he=gethostbyname(argv[1])) == NULL)
        Fatal("gethostbyname");
    if ((sd=socket(AF_INET,SOCK_STREAM,0)) == -1)
        FatalError("socket");
}
```

14 -Ken Wong, 1/29/2004

Washington University in St. Louis

## Client Code (Part 2)

```
srvAddr.sin_family = AF_INET;
srvAddr.sin_port = htons(PORT);    // short, NBO
srvAddr.sin_addr = *((struct in_addr *)he->h_addr); //NBO
bzero(&(srvAddr.sin_zero), 8);    // 0 rest of struct
```

```
if (connect(sd, (struct sockaddr *)&srvAddr,
            sizeof(struct sockaddr)) == -1)
    FatalError("connect");
if ((numbytes=recv(sd, buf, MAXDATASIZE, 0)) == -1)
    FatalError("recv");
buf[numbytes] = '\0';
printf("Received: %s", buf);
close(sd);
return 0;
```

15 -Ken Wong, 1/29/2004

Washington University in St. Louis

## Network Byte Order (NBO)

<b>A</b>	<b>BIG</b>	<b>A+1</b>
	<b>High-Order</b>	<b>Low-Order</b>
<b>A+1</b>	<b>A</b>	<b>LITTLE</b>
	<b>High-Order</b>	<b>Low-Order</b>

- Big Endian Byte Order (Network Byte Order)
  - » Used in protocol headers
  - » Deals with shorts (2-bytes) and longs (4-bytes) and the unsigned versions
- Examples: Sun (Big Endian), Intel (Little Endian)

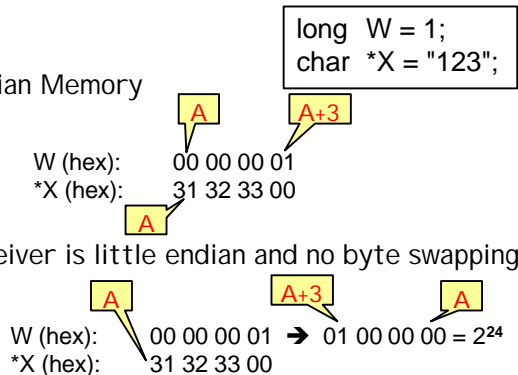
16 -Ken Wong, 1/29/2004

Washington University in St. Louis

## Network Byte Order (NBO)

- Consider the following on a big endian host:

- Big Endian Memory



- If receiver is little endian and no byte swapping:

17 - Ken Wong, 1/29/2004

Washington University in St. Louis

## Iterative Server Code (Part 1)

```
#include <./stdinc.h>
const int MYPORT = 3490; // client connects to this port
const int BACKLOG = 10; // max # pending connections
main() {
    int sd; // listen on sd
    int new_fd; // new connection on new_fd
    struct sockaddr_in myAddr, // my address
                    clAddr; // client's address
    char *msg = "Hello World\n"; // reply msg
    socklen_t sinSz = sizeof(struct sockaddr_in); // value-result arg

    if ((sd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
        FatalPerror("socket");
    myAddr.sin_family = AF_INET; // HBO
    myAddr.sin_port = htons(MYPORT); // short, NBO
    myAddr.sin_addr.s_addr = htonl(INADDR_ANY);
        // any interface; auto-fill with my IP; NBO
    bzero(&(myAddr.sin_zero), 8); // 0 rest of the struct
}
```

18 - Ken Wong, 1/29/2004

Washington University in St. Louis

## Iterative Server Code (Part 2)

```
if (bind(sd, (struct sockaddr *)&myAddr,
        sizeof(struct sockaddr)) == -1)
    FatalPerror("bind");
if (listen(sd, BACKLOG) == -1) FatalPerror("listen");

while(1) { // main accept() loop
    new_fd = accept(sd, (struct sockaddr *) &clAddr, &sinSz);
    if (new_fd == -1) FatalPerror("accept");
    printf ("server: got connection from %s\n",
            inet_ntoa(clAddr.sin_addr));
    if (send(new_fd, msg, strlen(msg), 0) == -1)
        FatalPerror("send");
    close(new_fd);
}
```

19 - Ken Wong, 1/29/2004

Washington University in St. Louis

## Server Code Using Wrappers

```
#include <./stdinc.h>
#include <./myinc.h> // wrappers and constants (e.g., MYPORT)
const int BACKLOG = 10; // max # pending connections
main() {
    ... Declarations (as before) ...

    Socket(AF_INET, SOCK_STREAM, 0);
    SocketAddr(MYPORT, INADDR_ANY);
    Bind(sd, (struct sockaddr *)&myAddr, sizeof(struct sockaddr));
    Listen(sd, BACKLOG) == -1);

    while(1) { // main accept() loop
        new_fd = Accept(sd, (struct sockaddr *) &clAddr, &sinSz);
        printf ("server: got connection from %s\n",
                inet_ntoa(clAddr.sin_addr));

        Send(new_fd, msg, strlen(msg), 0);
        close(new_fd);
    }
}
```

20 - Ken Wong, 1/29/2004

Washington University in St. Louis

## Testing The Simple Stream Server

### ■ Compiling

```
g++ -g -o myserver myserver.c -lsocket -lnsl
g++ -g -o myclient myclient.c -lsocket -lnsl
```

» Can use gcc also

» "-lsocket -lnsl" not needed on some UNIX (e.g., Linux, NetBSD)

### ■ Testing using telnet

» Start server on chigger.int.cec (172.16.2.1) ['myserver']

» Issue command "telnet 172.16.2.1 3490" from any host

» Output is the message "Hello World" interspersed with telnet messages each time you invoke telnet.

### ■ Testing using client

» Procedure is the same as using telnet except now issue the command "client 172.16.2.1" from any host

21 -Ken Wong, 1/29/2004

Washington University in St. Louis

## Extending the Basic Example

### ■ Client sends a request

» Return the value of gettimeofday() (in "struct timeval" format) from the remote host OR

» Return a list of who is logged on the remote host; i.e., the list of login names from the 'who' command

### ■ Issues

» Message (request and reply) format and alignment

» Data representation

» CPU architecture mismatch

22 -Ken Wong, 1/29/2004

Washington University in St. Louis

## Message Formats

### ■ General

» Message boundary

» No alignment issues

### ■ Request

» Fixed length msg

### ■ Reply

» Msg length depends on request type

» 'who\_reply' size is dynamic

```
short request; // operation
```

```
struct timeval gettimeofday;
```

```
struct who_reply { // Fake
    unsigned short nchars;
    char who[1];
} who_reply;
```

23 -Ken Wong, 1/29/2004

Washington University in St. Louis

## Caveats (1)

```
struct in_addr dst, src;
printf ("Dst: %s, Src: %s\n", inet_ntoa(dst), inet_ntoa(src));
```

### ■ Some Functions Are Non-Reentrant (e.g., gethostbyname(), inet\_ntoa())

» Prints the same value twice even if dst and src are different!!!

» Non-reentrant calls use a single static area to store results

» Solaris has reentrant forms (e.g., gethostbyname\_r())

» Older OSes do not have reentrant forms → Copy value before recalling

24 -Ken Wong, 1/29/2004

Washington University in St. Louis

## Caveats (2)

- Recv (Receive at most MAX\_DATA\_SIZE bytes)  
if ((nBytes = recv(sd, buf, MAX\_DATA\_SIZE, 0)) == -1) ...

- TCP can decide to deliver any non-zero number of bytes → Should really read something like this:

```
char *p = buf;
int  msgSz;           // number of bytes in message
int  nRead = 0;      // number of bytes read so far
int  nBytes = 0;     // number of bytes returned by recv()
while (nRead < msgSz) {
    nBytes = recv(sd, p, msgSz-nRead, 0);
    if (nBytes <= 0) Fatal("recv"); // <0: error; 0: EOF
    p = p+nBytes;
    nRead = nRead + nBytes;
}
```

25 -Ken Wong, 1/29/2004

Washington University in St. Louis

## Common Questions (1)

- How can a client and server tell each other which ports it is using?
  - » The client has to know the server's IP address and port number (it's "well known")
  - » The server gets the client's port from the SYN packet (connect()); the programmer doesn't need to know it
- When I run a client/server test it works the first time but later, I get "bind: Address already in use"
  - » 2MSL (Maximum Segment Life) TCP timer can be 2 minutes

26 -Ken Wong, 1/29/2004

Washington University in St. Louis

## Common Questions (2)

- When I try to run the server (client) I get the error message "invalid command"
  - » Check that you compiled the source on the correct host
    - In the example, the server must be compiled on chigger.cec (or like host) and the client must be compiled on hilton.cec (or like host)!
- How do I find out what ports are in use?
  - » Solaris,BSD: netstat -a -f inet
  - » Linux: netstat -a -A inet
- Why do we convert some header fields and not others?
  - » Only the non-byte fields need converting because of their "endianness"

27 -Ken Wong, 1/29/2004

Washington University in St. Louis

## Advice

- Write short, simple code
- Test code incrementally
- Don't write more code than you have to
- Provide wrappers for all system calls that test the return code!!!
- Use stderr for verbose status or flush your print buffer after each newline
- Verify the evolution of the computation state
- Use defensive functions/features (hex pkt hdr dump; Pkt snd/rcv trace)

28 -Ken Wong, 1/29/2004

Washington University in St. Louis