Socket Programming

Dr. Geetha V
Assistant Professor
Dept of IT,
NITK Surathkal

Socket Programming

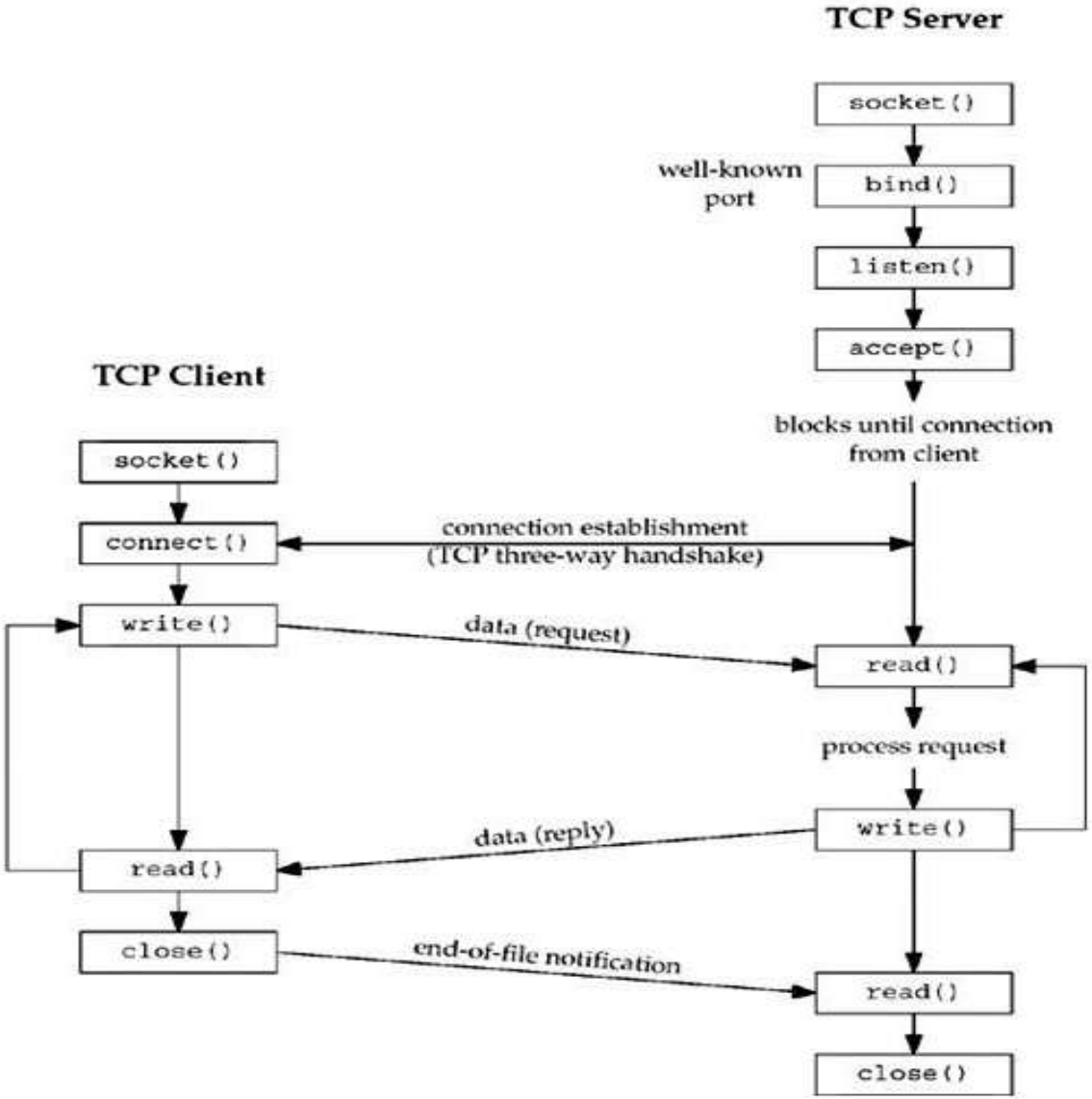
- What is a socket?
- Using sockets
  - Types (Protocols)
  - Associated functions
  - Styles

  - We will look at using sockets in C

What is a socket?

- An interface between application and network
  - **The application creates a socket**
  - The socket *type* dictates the style of communication
    - **reliable vs. best effort**
    - **connection-oriented vs. connectionless**
- Once configured the application can
  - **pass data to the socket for network transmission**
  - **receive data from the socket (transmitted through the network by some other host)**

Socket – client server functions



Server - Client

```
//fileserver.cc
#include<stdio.h>
#include<unistd.h>
#include<string.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<sys/types.h>
#define SERV_PORT 5576
main(int argc,char **argv) {
int i,j;
ssize_t n;
FILE *fp;
char s[80],f[80];
struct sockaddr_in servaddr,cliaddr;
int listenfd,connfd,clilen;
```

```
//fileclient.cc
 #include<stdio.h>
#include<unistd.h>
#include<string.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<sys/types.h>
#define SERV_PORT 5576
main(int argc,char **argv)
{
int i,j;
ssize_t n;
char filename[80],recvline[80];
struct sockaddr_in servaddr;
int sockfd;
```

Socket header files

#include<sys/socket.h>

Defines socket prototypes, macros, variables, and the following structures: **sockaddr, msghdr, linger**

#include<netinet/in.h>

Defines prototypes, macros, variables, and the **sockaddr_in** structure to use with Internet domain sockets.

File:NETINET **Member:**IN

Socket header files

#include<sys/types.h>

Defines various data types. Also includes prototypes, macros, variables, and structures that are associated with the **select()** function. You must include this file in all socket applications.

File:SYS **Member:**TYPES

#include<unistd.h>

Contains macros and structures that are defined by the integrated file system. Needed when the system uses the **read()** and **write()** system functions. **File:**H **Member:**UNISTD

The struct sockaddr

- The generic:

defined in sys/socket.h

```
struct sockaddr {
    u_short sa_family;
    char sa_data[14];
};
```

  - sa_family
    - specifies which address family is being used
    - determines how the remaining 14 bytes are used

- The Internet-specific:

Defined in netinet/in.h

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

  - sin_family = AF_INET
  - sin_port: port # (0-65535)
  - sin_addr: IP-address
  - sin_zero: unused

Server - Client

```
//fileserver.cc
#include<stdio.h>
#include<unistd.h>
#include<string.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<sys/types.h>
#define SERV_PORT 5576
main(int argc,char **argv) {
int i,j;
ssize_t n;
FILE *fp;
char s[80],f[80];
struct sockaddr_in servaddr,cliaddr;
int listenfd,connfd,clilen;
```

```
//fileclient.cc
 #include<stdio.h>
#include<unistd.h>
#include<string.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<sys/types.h>
#define SERV_PORT 5576
main(int argc,char **argv)
{
int i,j;
ssize_t n;
char filename[80],recvline[80];
struct sockaddr_in servaddr;
int sockfd;
```

Server - Client

```
listenfd=socket(
AF_INET,
SOCK_STREAM,0);
```

```
sockfd=socket(
AF_INET,
SOCK_STREAM,0);
```

- **int s = socket(domain, type, protocol);**
  - s: socket descriptor, an integer (like a file-handle)
  - domain: integer, communication domain
    - e.g., **PF_INET (IPv4 protocol)** – typically used
    - e.g. **AF_INET (Ipv4 – Internet Protocol)**
  - type: communication type
    - **SOCK_STREAM**: reliable, 2-way, connection-based service
    - **SOCK_DGRAM**: unreliable, connectionless,
  - protocol: specifies protocol (see file /etc/protocols for a list of options) - usually set to 0

Server - Client

```
listenfd=socket(
AF_INET,
SOCK_STREAM,0);
```

```
sockfd=socket(
AF_INET,
SOCK_STREAM,0);
```

- **int s = socket(domain, type, protocol);**
  - s: socket descriptor, an integer (like a file-handle)
  - domain: integer, communication domain
    - e.g., **PF_INET (Protocol Family – socket- IPv4 protocol)**
    - e.g. **AF_INET (Address Family- sockaddr_in Ipv4)**
  - type: communication type
    - **SOCK_STREAM**: reliable, 2-way, connection-based service
    - **SOCK_DGRAM**: unreliable, connectionless,
  - protocol: specifies protocol (see file /etc/protocols for a list of options) - usually set to 0

Server

```
bzero(&servaddr,sizeof(servaddr));
  //bzero(char* c, int n): 0's n bytes starting at c
servaddr.sin_family=AF_INET;
servaddr.sin_port=htons(SERV_PORT);
bind(listenfd,(struct sockaddr *)&servaddr,
sizeof(servaddr));
```

- associates and (can exclusively) reserves a port for use by the socket
- **int status = bind(sockid, &addrport, size);**
  - status: error status, = -1 if bind failed
  - sockid: integer, socket descriptor
  - addrport: struct sockaddr, the (IP) address and port of the machine (address usually set to INADDR_ANY – chooses a local address)
  - size: the size (in bytes) of the addrport structure

Server

listen(listenfd,1);

- Called by passive participant
- **int status = listen(sock, queuelen);**
  - status: 0 if listening, -1 if error
  - sock: integer, socket descriptor
  - queuelen: integer, # of active participants that can "wait" for a connection
  - listen is **non-blocking**: returns immediately

Client

```
bzero(&servaddr,sizeof(servaddr));
servaddr.sin_family=AF_INET;
servaddr.sin_port=htons(SERV_PORT);
inet_pton(AF_INET,argv[1],&servaddr.sin_addr);
connect(sockfd,(struct
sockaddr*)&servaddr,sizeof(servaddr));
```

- **int status = connect(sock, &name, namelen);**
  - status: 0 if successful connect, -1 otherwise
  - sock: integer, socket to be used in connection
  - name: struct sockaddr: address of passive participant
  - namelen: integer, sizeof(name)
- connect is **blocking**

Client

```
bzero(&servaddr,sizeof(servaddr));
servaddr.sin_family=AF_INET;
servaddr.sin_port=htons(SERV_PORT);
inet_pton(AF_INET,argv[1],&servaddr.sin_addr);
connect(sockfd,(struct
sockaddr*)&servaddr,sizeof(servaddr));
```

#include <netinet/in.h>

uint32_t htonl(uint32_t hostlong); Host to network long

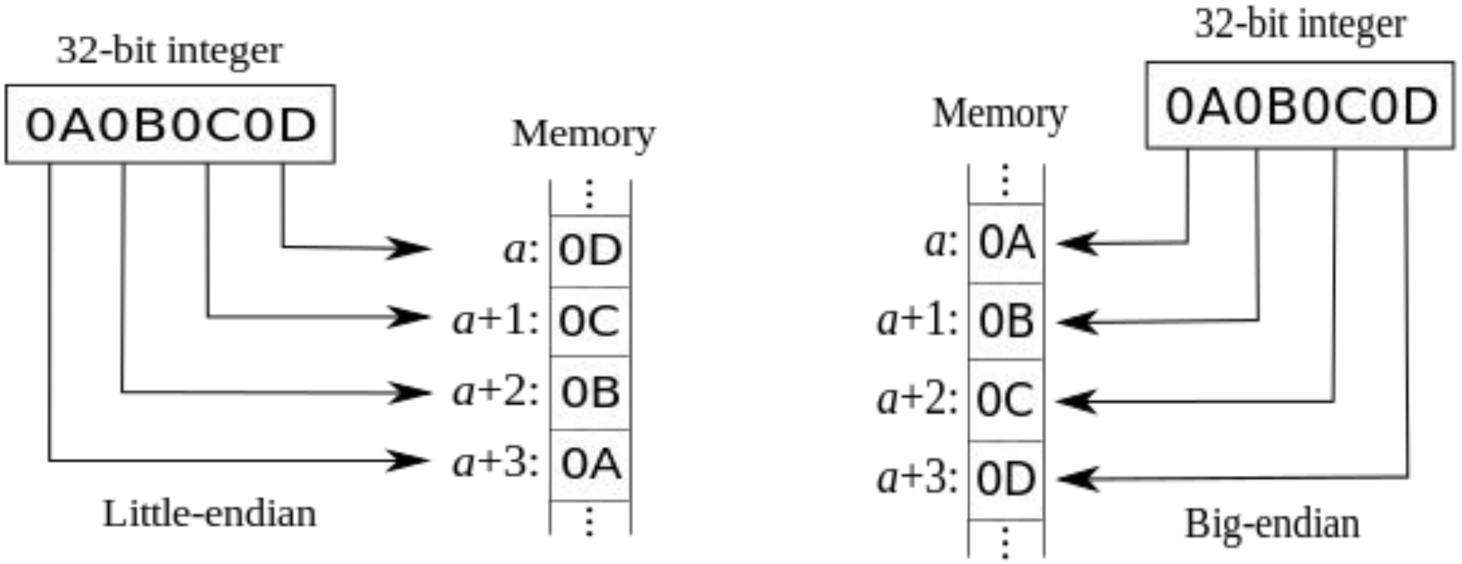
uint16_t htons(uint16_t hostshort); Host to network Short

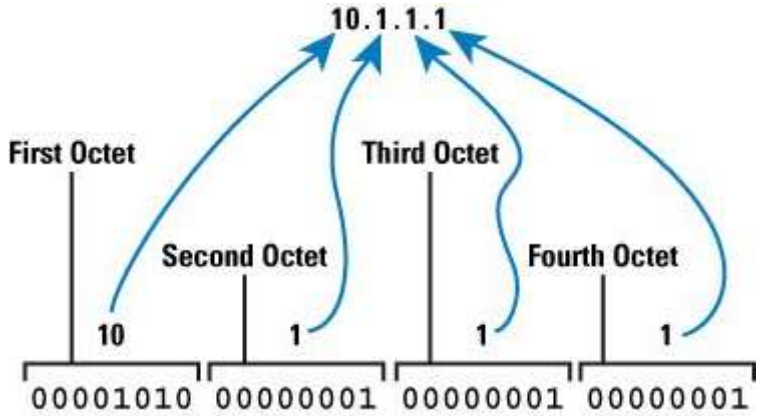
uint32_t ntohl(uint32_t netlong); Network to host long

uint16_t ntohs(uint16_t netshort); Network to host short

- **Host Byte-Ordering**: the byte ordering used by a host (big or little) Intel: little endian. Motorola IBM –Big Endian
- **Network Byte-Ordering**: the byte ordering used by the network – always **big-endian**

Little and Big Endian





Client

```
bzero(&servaddr,sizeof(servaddr));
servaddr.sin_family=AF_INET;
servaddr.sin_port=htons(SERV_PORT);
inet_pton(AF_INET,argv[1],&servaddr.sin_addr);
connect(sockfd,(struct
sockaddr*)&servaddr,sizeof(servaddr));
```

- **n=>network p=>presentation**
- const char *inet_ntop(int af, const void *src, char *dst, socklen_t size);

  Network format to printable or dotted format
- int inet_pton(int af, const char *src, void *dst);

  Dotted format of address to network format

Server

```
clilen=sizeof(cliaddr);
connfd=accept(listenfd,(struct sockaddr*)&cliaddr, &clilen);
printf("\n clinet connected");
```

- **int s = accept(sock, &name, &namelen);**
  - s: integer, the new socket (used for data-transfer)
  - sock: integer, the orig. socket (being listened on)
  - name: struct sockaddr, address of the active participant
  - namelen: sizeof(name): value/result parameter
    - must be set appropriately before call
    - adjusted by OS upon return
  - accept is **blocking**: waits for connection before returning

Server - Client

```
read(connfd,f,80);
fp=fopen(f,"r");
printf("\n name of the file:
%s",f);
while(fgets(s,80,fp)!=NULL)
{
printf("%s",s);
write(connfd,s,sizeof(s));
}
close(listenfd);
fclose(fp); }  //close main
```

```
printf("enter the file name");
scanf("%s",filename);
write(sockfd,filename,sizeof(
        filename));
printf("\n data from server: \n");
while(read(sockfd,recvline,80)!=0)
{
fputs(recvline,stdout);
}
close(sockfd);

}
```

Sending / Receiving Data

- With a connection (SOCK_STREAM):
  - **int count = send(sock, &buf, len, flags);**
    - count: # bytes transmitted (-1 if error)
    - buf: char[], buffer to be transmitted
    - len: integer, length of buffer (in bytes) to transmit
    - flags: integer, special options, usually just 0
  - **int count = recv(sock, &buf, len, flags);**
    - count: # bytes received (-1 if error)
    - buf: void[], stores received bytes
    - len: # bytes received
    - flags: integer, special options, usually just 0
  - Calls are **blocking** [returns only after data is sent (to socket buf) / received]

close

- When finished using a socket, the socket should be closed:
- status = close(s);
  - status: 0 if successful, -1 if error
  - s: the file descriptor (socket being closed)
- **Closing a socket**
  - closes a connection (for SOCK_STREAM)
  - frees up the port used by the socket

Solution: Network Byte-Ordering

- Defs:
  - **Host Byte-Ordering**: the byte ordering used by a host (big or little)
  - **Network Byte-Ordering**: the byte ordering used by the network – always big-endian
- Any words sent through the network should be converted to **Network Byte-Order** prior to transmission (and back to Host Byte-Order once received)

Dealing with blocking calls

- Many of the functions we saw block until a certain event
  - accept: until a connection comes in
  - connect: until the connection is established
  - recv: until a packet (of data) is received
  - send: until data is pushed into socket's buffer
    - Q: why not until received?
- For simple programs, blocking is convenient
- What about more complex programs?
  - multiple connections
  - simultaneous sends and receives
  - simultaneously doing non-networking processing

Other useful functions

- bzero(char* c, int n): 0's n bytes starting at c
- gethostname(char *name, int len): gets the name of the current host
- gethostbyaddr(char *addr, int len, int type): converts IP hostname to structure containing long integer
- inet_addr(const char *cp): converts dotted-decimal char-string to long integer
- inet_ntoa(const struct in_addr in): converts long to dotted-decimal notation

- Warning: check function assumptions about byte-ordering (host or network). Often, they assume parameters / return solutions in network byte-order

Server Program Client Program

- Define Headers
- Main() :
  - Define variables
  - Socket()
  - Bind()
  - Listen()
  - accept()
  - Read/write
  - close

- Define Headers
- Main()
  - Define variables
  - Socket()
  - Connect
  - Write/read
  - close

headers

```
//server
#include<stdio.h>
#include<unistd.h>
#include<string.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<sys/types.h>
#define SERV_PORT
5576
```

```
//client
#include<stdio.h>
#include<unistd.h>
#include<string.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<sys/types.h>
#define SERV_PORT
5576
```

pgm

```
//server
main(int argc,char **argv)
{
int i,j;
ssize_t n;
FILE *fp;
char s[80],f[80];
struct sockaddr_in servaddr,cliaddr;
int listenfd,connfd,clilen;
```

```
//client
main(int argc,char **argv)
{
int i,j;
ssize_t n;
char filename[80],recvline[80];
struct sockaddr_in servaddr;
int sockfd;
```

pgm

//server

```
listenfd=socket(AF_INET,SOCK_STREAM,0);
bzero(&servaddr,sizeof(servaddr));
servaddr.sin_family=AF_INET;
servaddr.sin_port=htons(SERV_PORT);
bind(listenfd,(struct sockaddr *)&servaddr,sizeof(servaddr));
listen(listenfd,1);
```

//client

```
sockfd=socket(AF_INET,SOCK_STREAM,0);
```

pgm

```
//server
clilen=sizeof(cliaddr);
connfd=accept(listenfd,(struct sockaddr*)&cliaddr,&clilen);
```

```
//client
bzero(&servaddr,sizeof(servaddr));
servaddr.sin_family=AF_INET;
servaddr.sin_port=htons(SERV_PORT);
inet_pton(AF_INET,argv[1],&servaddr.sin_addr);
connect(sockfd,(struct sockaddr*)&servaddr,sizeof(servaddr));
```

Read/write

//server

```
printf("\n clinet connected");
read(connfd,f,80);
fp=fopen(f,"r");
printf("\n name of the file: %s",f);
while(fgets(s,80,fp)!=NULL)
{
printf("%s",s);
write(connfd,s,sizeof(s));
}
close(listenfd);
fclose(fp);
}
```

//client

```
printf("enter the file name");
scanf("%s",filename);
write(sockfd,filename,sizeof(filename));
printf("\n data from server: \n");
while(read(sockfd,recvline,80)!=0)
{
fputs(recvline,stdout);
}
close(sockfd);


}
```