

The Berkely API (Application Protocol Interface)

Helpful Notes

By

Pallab Datta

Feb 20, 2001

The Berkeley API

An application program interface(API) allows Application programs to access certain resources through a predefined and preferably consistent interface.

The socket interface is now available on many UNIX machines.

Another popular socket interface that was derived from BK-socket is called the [Windows Socket](#) or [Winsock](#).

The socket mechanism allows programmers to write appl programs easily without worrying about the underlying networking details.

In a typical communication session one application operates as a server and the other application acts as the client. The server provides services upon request from the client.

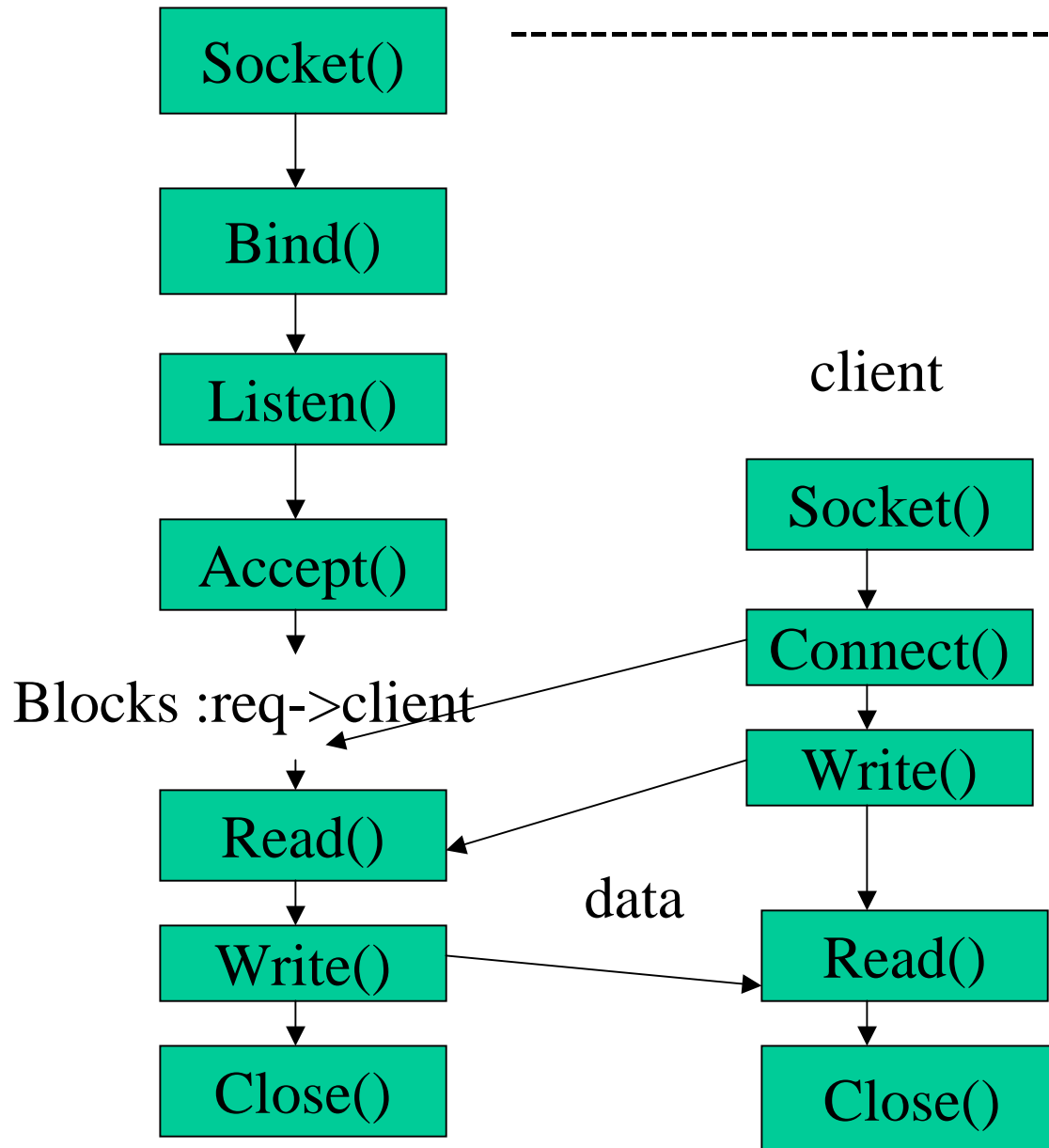
Two modes of service : Connection-oriented ,Connectionless

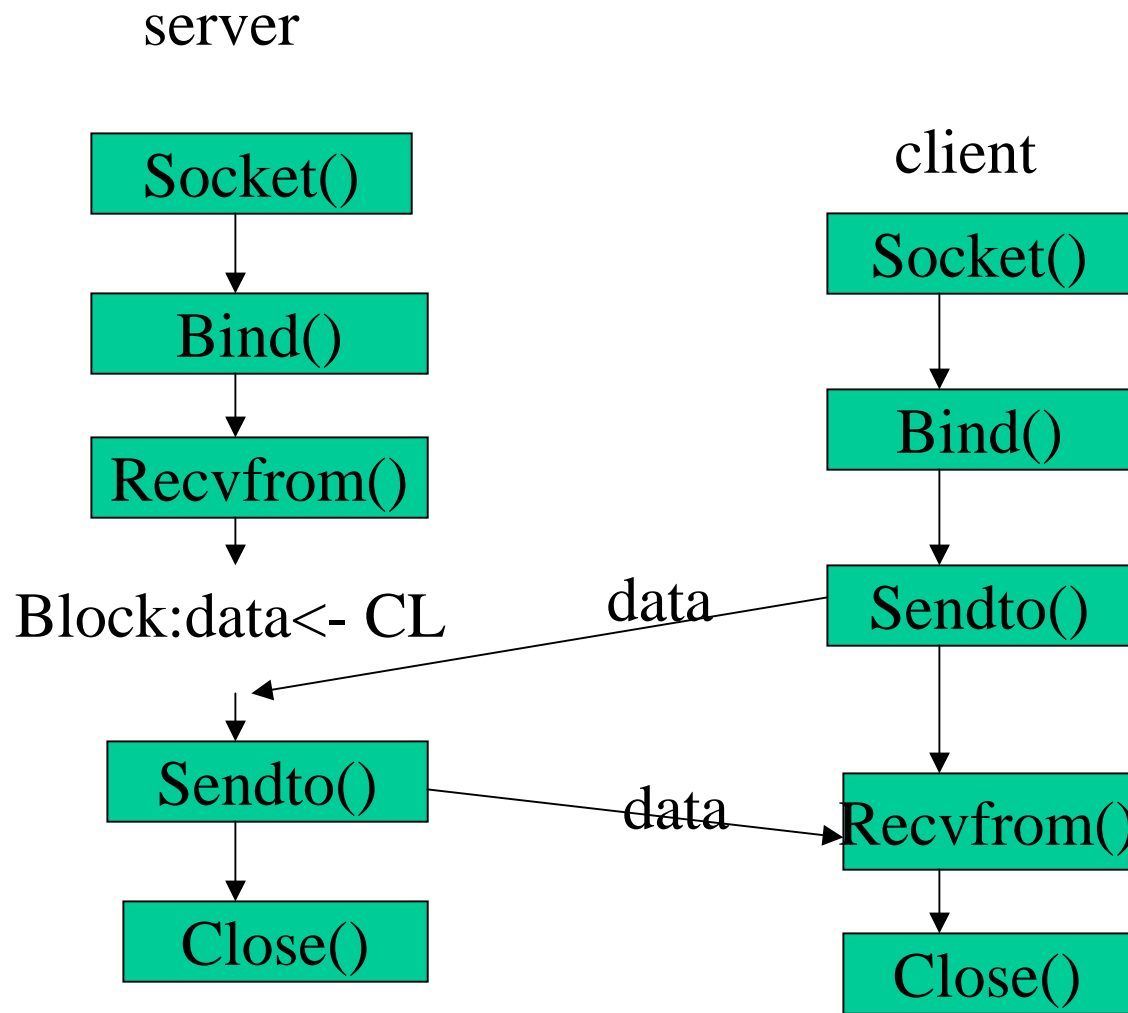
Connection-oriented : Reliable ,setup overhead

Connectionless : Unreliable, “Best –Effort” Service, No setup overhead

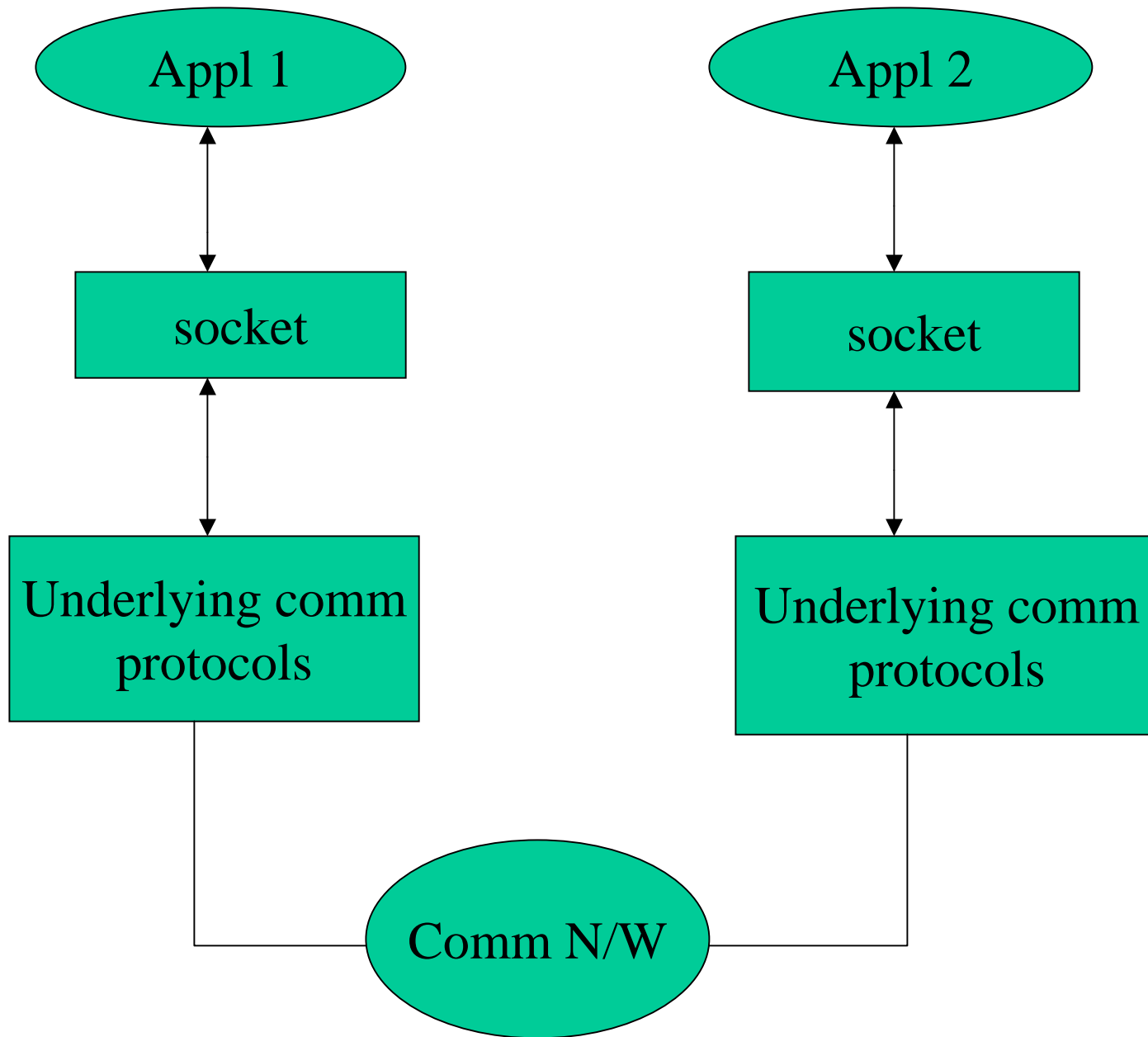
server

“socket calls for connection-oriented com”





Socket calls for connectionless communication



Communication through the socket interface

SOCKET CALLS FOR CONNECTION-ORIENTED COMM

* Server -> calls `socket()` -> creates a TCP socket.

The `bind()` then binds the port address of the server to the socket.

The `listen()` call then turns the socket into a listening socket that can accept incoming connections from clients.

Finally the `accept()` call puts the server process to sleep until a client request arrives..!!

NOW Client calls `socket()` -> creates a active socket on the client side

`Connect()` call-> establishes the TCP connection to the server with the specified destination socket address. When the TCP connection is completed the `accept()` function at the server wakes up and returns the descriptor for the given connection.

Client and server are now **READY** to exchange information. Finally both the server and client closes connection through `close()`.

SOCKET CALLS FOR CONNECTIONLESS COMMUNICATION

* NO connection is established prior to communication.

In the same way the server calls `socket()`-> active socket is created at the server side. Then it binds the port no of the server to the socket by using the `bind()` call.

Then it calls `recvfrom()` -> puts the server to a waiting state till the arrival of some request from the client.

* At the client end `socket()`-> creates an active socket at the client end. `Bind()` -> binds the port # at the client end to the active socket created.

The `recvfrom()` call at the server end returns, when a complete UDP datagram has been received from the client side.

Finally both the client and the server terminates their socket connection through the `close()` call.

SOCKET SYSTEM CALLS

* Include header files `<sys/types.h>` and `<sys/socket.h>`

* `int socket(int family, int type, int protocol);`

family -> identifies the family by address or by protocol. The address family identifies a collection of protocols with the same address format. The protocol family identifies a collection of protocol having same architecture. e.g: `AF_UNIX` –used for comm on UNIX machines. `AF_INET`- used for internet comm using TCP/IP protocol. The protocol family is identified using the `PF_` prefix.

type -> identifies the semantics of communication.

`SOCK_STREAM` : provides data delivery service as a sequence of bytes and does not preserve message boundaries. `SOCK_DGRAM` : provides data delivery in blocks of bytes..!!

protocol-> protocol to be used . e.g: default protocol for each family and type. Default protocol for `SOCK_STREAM` type with `AF_INET` family is TCP. For `SOCK_DGRAM` with `AF_INET` is UDP.

SOCKET SYSTEMS CALLS(contd.....)

`int bind(int sd, struct sockaddr *name, int namelen);` This call is used to assign an address to a socket.

`sd` -> socket descriptor returned by socket call.

`name` -> pointer to an address structure that contains the local IP address and the port #.

`namelen`-> is the size of the address structure in bytes

Return value : 0 on success and -1 on failure.

```
sockaddress structure : struct sockaddr {  
    u_short sa_family; /*address family */  
    char    sa_data[14] /* address */ }  
}
```

SOCKET SYSTEMS CALLS (contd...)

An application program using the Internet family should use the `sockaddr_in` structure to assign values and should use the `sockaddr` structure only for casting purposes in function arguments.

```
struct sockaddr_in {  
    u_short sin_family;    /* AF_INET */  
    u_short  sin_port;     /* TCP or UDP port */  
    struct   in_addr sin_addr /* 32-bit IP address */  
    char    sin_zero[8]    /* unused */  
}
```

`sin_addr` -> local IP add. For a host with multiple IP addresses, `sin_addr` is typically set to `INADDR_ANY` to indicate that the server is willing to accept comm through any of its IP addresses.

`sin_zero` -> Is used to fill out struct `sockaddr_in` to 16 bytes.

MORE SOCKET CALLS ..!!

Client establishes a connection on a socket by calling `connect()`. The prototype for `connect` is :

```
int connect (int sd, struct sockaddr *name , int namelen);
```

`sd`-> socket descriptor returned by the `socket` call.

`name` -> points to the server address structure.

`namelen`-> amount of space in bytes specified by `name`.

For connection oriented comm `connect` attempts to setup a virtual ckt between the server and the client.

For connectionless comm `connect` stores the server's add, so that client can use the socket descriptor when sending datagrams instead of specifying the server's add each time..!!!

Return value :0 on success and -1 on failure.

MORE SYS CALLS....

The server can accept a conn-req from client after issuing `listen()`

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

sd-> socket descriptor returned by the `socket()` call.

backlog-> max no of connection requests that the sys should queue while it waits for the server to accept them(usually 5..!!).

Return value : success is 0, failure is -1.

After this the server accepts the connection request by calling `accept()`.

```
int accept(int sd, struct sockaddr *addr, int *addrlen)
```

addr -> pointer to an add structure that `accept` fills in with the clients IP addr and Port No. `addrlen`-> specifies the amount of space pointed to by `addr` before the call. If no conn req's are pending `accept` will block the caller until a connection arrives. It returns a new socket descriptor.

SYS CALLS FOR DATA TRANSFER ..!!

Clients and servers can exchange data using `write` and `sendto`.

`write()` call is used for connection-oriented comm. A connectionless client may also call `write` if the client has executed a `connect` call..!!

`sendto()` is used in connectionless communication.

```
int write( int sd, char *buf , int buflen);
```

```
int sendto(int sd, char *buf, int buflen, int flags, struct sockaddr *addrp,  
int addrlen);
```

Return value : returns the no of bytes transmitted on success and `-1` on failure.

```
int read(int sd, char *buf , int buflen);
```

```
int recvfrom(int sd, char *buf, int buflen, int flags, struct sockaddr  
*addrp, int addrlen);
```

Return value : returns the no of bytes received on success and `-1` on failure.

Terminate the socket connection `-close(int sd)`.

NETWORK UTILITY FUNCTIONS

To convert a domain name to an IP add include <sys/socket.h>, <sys/types.h> and <netdb.h>.

```
struct hostent {  
  
    char    *h_name; /* official name or host*/  
    char    **h_aliases; /*alias name this host uses*/  
    int     h_addrtype ; /* address type */  
    int     h_length    ; /* length of address */  
    char    **h_addr_list; /* list of addr's from name server*/  
}
```

NAME TO ADDRESS TRANSLATION FUNCTIONS

* struct hostent *gethostbyname (char *name);

Returns NULL on error. It obtains information from the file /etc/hosts or from the name server.

* struct hostent *gethostbyaddr(char *addr, int len, int type);

Returns the same information as the above.

IP ADDRESS MANIPULATION FUNCTIONS

For IP address manipulation the `<sys/types.h>`, `<sys/socket.h>`, `<netinet/in.h>` and `<arpa/inet.h>` are to be included.

The two functions used for this purpose are :

`char *inet_ntoa(struct in_addr in)` -> Takes a 32 bit-IP address and returns the corresponding IP address in dotted format.

`unsigned long inet_addr(char *cp)` -> Takes the host address in dotted format and returns the IP address as a 32 bit IP address in network byte order.

IMPLEMENTING AN ECHO SERVER..

Program at the “server –end”

* Create a stream socket

```
if((sd= socket(AF_INET,SOCK_STREAM,0) ) == -1) {  
    fprintf(stderr, “cant create a socket”);  
    exit (1); }
```

* Bind some add to the socket

```
bzero((char *)& server ,sizeof(struct sockaddr_in));  
server.sin_family =AF_INET;  
server.sin_port = htons(port);  
server.sin_addr.s_addr = htons(INADDR_ANY);
```

* Queue up to 5 connect requests

```
listen(sd ,5);
```

* Code to handle connection req from client

```
if ((new_sd = accept(sd, (struct sockaddr *)&client, &client_len)) == -1)
{ fprintf(stderr, "can't accept client");
exit(1); }
```

* Reading bytes

```
bytes_to_read = MAXBUFLLEN; bp = buf ;
while ((n= read(new_sd, bp, bytes_to_read)) > 0 ) {
bp += n;
bytes_to_read -= n; }
write(new_sd, buf, MAXBUFLLEN); }
```

* Closing socket connection at client and server

```
close(new_sd);
close(sd);
```

Program Executing at the Client-end

* Create a socket stream

```
if((sd= socket(AF_INET,SOCK_STREAM ,0) == -1) {  
    fprintf(stderr, “can’t create a socket”);  
    exit(1); }
```

* Bind address to the socket at the client end

```
bzero((char *)&server ,sizeof(struct sockaddr_in));  
server.sin_family =AF_INET;  
server .sin_port = htons(port);  
if((hp = gethostbyname(host)) == NULL) {  
    fprintf( stderr, “Can’t get server’s address..!!”);  
    exit(1); }
```

* Connecting to the server

```
if(connect(sd , (struct sockaddr *)&server, sizeof(server)) == -1){  
    fprintf(stderr, “can’t connect..!!”); exit(1); }
```

* Input from the user ..

```
gets(sbuf); /* sbuf -> receive input buffer */
```

```
write(sd, sbuf , MAXBUFLEN); /* send the buf to the server*/
```

* Receive data from the server side

```
bp =rbuf; bytes_to_read = MAXBUFLEN;  
while((n = read(sd, bp , bytes_to_read)) > 0) {  
    bp += n; bytes_to_read -= n; }
```

* Closing connection at client side

```
close(sd);
```

SOME IMP WEB-SITES FOR REFERENCE

<http://www.cs.rpi.edu/courses/netprog/lectures/pphtml/sockets/>

http://orca.stm.edu/~sayfarth/network_pgm/net-6-6-1.html

<http://www.linuxgazette.com/issue47/bueno.html>

Reading references : UNIX Network Programming by
W.Richard Stevens.

Here is a list of links to Sockets information as provided by students to Pallab:

<http://www.ecst.csuchico.edu/~beej/guide/net/>
http://www.cs.utah.edu/dept/old/texinfo/glibc-manual-0.02/library_15.html
<http://compnetworking.about.com/compute/compnetworking/cs/socketprogramming/>
<http://fag.grm.hia.no/it2200/ovinger/Csourcesocket.htm>
<http://www.freesoft.org/CIE/Topics/3.htm>
<http://www.ece.wpi.edu/courses/ee535/hwk97/hwk4cd97/murti/node3.html>
<http://www.cs.rpi.edu/courses/netprog/lectures/pphtml/sockets/>
<http://www.linuxgazette.com/issue47/bueno.html>
http://orca.stm.edu/~sayfarth/network_pgm/net-6-6-1.html
<http://www.cs.mun.ca/~rod/Winter98/cs4759/berkeley.html>
http://www.stardust.com/winsock/ws1.1_api/
<http://guir.berkeley.edu/projects/javadoc/help-doc.html>
<http://www2.cc.unca.edu/BerkeleyDB/>
http://eel.st.usm.edu/~seyfarth/network_pgm/net-6-6-1.html
http://www.sockets.com/ms_icmp.htm
<http://bmrc.berkeley.edu/research/publications/1996/112/node6.html>
<http://www.cis.ohio-state.edu/htbin/rfc/rfc2292.html>
<http://casaturn.kaist.ac.kr/~sikang/os/socket/>
http://www3.tsl.uu.se/~micke/WASA/Control_via_internet/control_via_net.htm

1

http://www.openvms.compaq.com:8000/721final/6523/6523pro_016.html
http://www.classic.be.com/developers/may_dev_conf/transcripts/approachingnetworking/
<http://now.cs.berkeley.edu/Fastcomm/fastcomm.html>
<http://www.cs.tamu.edu/course-info/cpsc463/tutorial/>
<http://www.cs.berkeley.edu/~dmartin/qt/topicals.html>
<http://www.ieng.com/univercd/cc/td/doc/product/software/ioss390/ios390sk/sklibfun.htm>
<http://www.mtholyoke.edu/acad/compsc/Honors/Hu-Imm-Lee/ch2.htm>
<http://www.internet2.edu/qos/may98Workshop/html/apiprop.html> The
<http://www.uwo.ca/its/doc/courses/notes/socket/>
<http://www.pcquest.com/may99/socket.asp>
<http://sunsite.iisc.ernet.in/collection/virlib/tcpip/parker/tyt14fi.htm>
http://kitap.selcuk.edu.tr/tcpip/Teach_Yourself_TCPIP/tyt14fi.htm
<http://blondie.mathcs.wilkes.edu/~sullivan/sockets/>
<http://www.netcon.com/docs/socket.htm>
<http://www.ecst.csuchico.edu/~beej/guide/net/>
<http://www-users.cs.umn.edu/~bentlema/unix/advipc/ipc.html>
<http://world.std.com/~jimf/papers/sockets/sockets.html>
<http://www.s390.ibm.com/products/tpf/v3n2a4.html>
<http://www.cs.utexas.edu/users/dragon/cs378/resources/sockets.html>
<http://www.cis.temple.edu/~ingargio/cis307/readings/unix4.html>
<http://www.fortunecity.com/skyscraper/arpanet/6/cc.htm>
<http://www.extreme.indiana.edu/~gannon/c212.f96/c212w14.html>
<http://troyda.eas.muohio.edu/623/L1-14/L1-14.html>
http://wwwcs.dongguk.ac.kr/~jiwon/main/network_tp/SOCKET1/

<http://www.acornsw.com/cujcd/HTML/15.05/ROSS/ROSS.HTM>

<http://www.sparc.spb.su/jjb/Docs/internet/sockets>

<http://www.sockaddr.com/TheSocketsParadigm.html>

<http://www.whisqu.se/per/docs/general28.htm>

http://www.adetti.iscte.pt/ADETTI/Security/HowTo/net_programming/winsick3.htm