Berkeley Sockets

Prof. Indranil Sen Gupta
Department of Computer Science & Engg.
I.I.T. Kharagpur

What is a socket?

- The ***socket*** is the BSD method for accomplishing inter-process communication (IPC).
- It is used to allow one process to speak to another (on same or different machine).
  - ***Analogy***: Like the telephone is used to allow one person to speak to another.
- Works very similar to files.
  - Socket descriptor ➔ very similar to file descriptor.
  - Read/write on a socket and file are very similar.

Basic Idea

- When two processes located on the same or different machines communicate, we define association and socket.
  - ***Association***: basically a 5-tuple
    - Protocol
    - Local IP address
    - Local port number
    - Remote IP address
    - Remote port number
  - ***Socket***: also called half-association (a 3-tuple)
    - Protocol, local IP address, local port number
    - Protocol, remote IP address, remote port number

More about sockets

- Creating a socket is the first step in network programming using BSD socket interface.
  - Using the *socket()* system call.
- Two main addressing formats of a socket:
  - ***AF_UNIX***: uses Unix pathnames to identify sockets, and are very useful for IPC between processes on the same machine.
  - ***AF_INET***: uses IP addresses.
    - In addition to machine address, there is also a port number that allows more than one *AF_INET* socket on each machine.

Types of socket

- Two most common types:
  - ***SOCK_STREAM***: Stream sockets, which provide reliable, two-way, connection-oriented communication streams. *<Uses TCP>*
  - ***SOCK_DGRAM***: Datagram sockets, which provide connectionless, unreliable service, used for packet-by-packet transfer of information. *<Uses UDP>*
- Other types like SOCK_RAW also exist.
  - Beyond the scope of the present discussion.

Systems calls for using sockets

- socket()
- bind()
- connect()
- listen()
- accept()
- send() & recv()
- sendto() & recvfrom()
- close() & shutdown()
- getpeername()
- gethostname()
- gethostbyname()

socket() :: Get the Socket Descriptor

- ***General syntax:***

```
#include <sys/types.h>
#include <sys/socket.h>
int  socket (int domain, int type, int protocol)
```

  - *domain*: should be set to AF_INET (typically)
  - *type*: should be set to SOCK_STREAM or SOCK_DGRAM
  - *protocol*: set to zero (typically)
- ***Returns***: socket descriptor; -1 on error

bind():: What Port am I on?

- Used to associate the socket with an address
- ***General syntax***:

```
#include <sys/types.h>
#include <sys/socket.h>
int  bind (int sockfd, struct sockaddr *my_addr, int addrlen);
```

  - *sockfd*: socket file descriptor returned by *socket()*
  - *my_addr*: pointer to a structure that contains information about the local IP address and port number.
  - *addrlen*: typically set to *sizeof(struct sockaddr)*
- ***Returns***: -1 on error

The *sockaddr* Structure

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

sockaddr_in is a parallel structure to ***sockaddr*** which a programmer uses in the program for convenience.

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

```
struct  in_addr
 {
  unsigned  long  s_addr;
 }
```


connect(): Connect to a Remote Socket

- ***General syntax***:

```
#include  <sys/types.h>
#include  <sys/socket.h>
int  connect (int sockfd, struct sockaddr *serv_addr, int addrlen);
```

  - *sockfd*: socket file descriptor returned by *socket()*
  - *serv_addr*: pointer to a structure that contains the destination IP address and the port number
  - *addrlen*: typically set to *sizeof(struct sockaddr)*
- ***Returns***: -1 on error

listen(): Get Set for Incoming Connections

- Here, we wish to wait for incoming connections and handle them in some way.
  - Two steps, first you *listen(),* then you *accept().*
- ***General syntax***:

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

  - *sockfd*: socket file descriptor returned by *socket().*
  - *backlog*: used to set the maximum number of requests (up to a maximum of about 20) that will be queued up before requests start being denied.
- ***Returns***: -1 on error

accept(): Waiting for Incoming Connections

- Basic concept:
  - Someone far away will try to *connect()* to your machine on a port that you are *listen()*'ing on.
  - Such connections will be queued up waiting to be *accept()*'ed.
  - *accept()* returns a ***brand new socket file descriptor*** to use for every single connection.
- Two socket file descriptors!!
  - The original one is still listening on your port.
  - Newly created one is finally ready to *send()* and *recv().*

accept(): contd..

- ***General syntax***:

```
#include <sys/socket.h>
int accept (int sockfd, void *addr, int *addrlen);
```

  - *sockfd*: *listen()*'ing socket descriptor
  - *addr*: pointer to a local *struct sockaddr_in* (This is where the information about the incoming connection will go)
  - *addrlen*: local integer variable that should be set to *sizeof(struct sockaddr_in)* before *accept()* is called.
- ***Returns***: -1 on error

send() and recv(): Sending/receiving Data

- Used for communicating over stream sockets or connected datagram sockets.
- ***General syntax***:

```
int send (int sockfd, const void *mesg, int len, int flags);
int recv (int sockfd, void *buf, int len, unsigned int flags);
```

  - *mesg*: a pointer to the data you want to send
  - *len*: length of the data in bytes
  - *buf*: buffer to read the information into
  - *flags*: typically set to 0
- *send()* returns the number of bytes actually sent out, and *recv()* returns the number of bytes actually read into the buffer.

sendto() and recvfrom()

- Used to transmit and receive data packets over unconnected datagram sockets.
- ***General syntax***:

```
int sendto (int sockfd, const  void *msg, int len, unsigned int flags,
        const struct sockaddr *to, int tolen);
int recvfrom (int sockfd, void *buf, int len, unsigned int flags,
        struct sockaddr *from, int *fromlen);
```

- If you *connect()* a datagram socket, you can then simply use *send()* and *recv()* for all your transactions.

close() and shutdown()

- Used to close the connection on the socket descriptor.

```
close (sockfd);
```

  - This prevents any more reads and writes to the socket.

```
int shutdown (int sockfd, int how);
```

  - how=0 ➔ further receives are disallowed
  - how=1 ➔ further sends are disallowed
  - how=2 ➔ further sends and receives are disallowed (like *close*())

getpeername()

- This function will tell you who is at the other end of a connection stream socket.

```
#include  <sys/socket.h>
int  getpeername (int sockfd, struct sockaddr *addr, int *addrlen);
```

  - *sockfd*: descriptor of the connected stream socket
  - *addr*: pointer to a structure that will hold the information about the other side of the connection
  - *addrlen*: pointer to an *int* that should be initialized to *sizeof(struct sockaddr)*

gethostname()

- This function returns the name of the computer that your program is running on.
  - This name can be used by *gethostbyname()* to determine the IP address of the local machine.

```
#include  <unistd.h>
int  gethostname (char *hostname, size_t size);
```

  - *hostname*: pointer to an array of *char*s that will contain the host name upon the function's return.
  - *size*: length in bytes of the *hostname* array.

gethostbyname()

- Returns the IP address of a host given its name.
  - Invokes the *Domain Name Server (DNS).*

```
#include <netdb.h>

struct hostent *gethostbyname (const char *name);
```

  - Returns a pointer to a struct hostent:

```
struct hostent
  {
    char    *h_name;        /* official name of the host */
    char    **h_aliases;    /* NULL terminate array of alternate names */
    int     haddrtype;      /* Type of address being returned (AF_INET) */
    int     h_length;       /* Length of the address in bytes */
    char    **h_addr_list;  /* Zero terminated array of network addresses */
  };
#define  h_addr  h_adr_list[0];
```

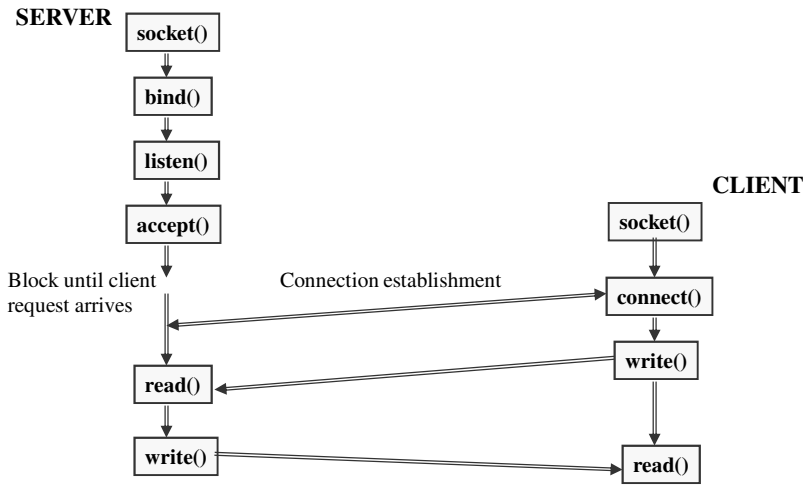

Client-server Model

- Standard model for network applications.
  - A ***server*** is a process that is waiting to be contacted by a ***client*** process so as to provide some service.
- ***Typical scenario***:
  - The server process is started on some computer system.
    - Initializes itself, then goes to sleep waiting for a client request.
  - A client process is started, either on the same system or on some other system.
    - Client sends a request (across the network) to the server.
  - When the server process has finished providing its service to the client, the server goes back to sleep, waiting for the next client request to arrive.

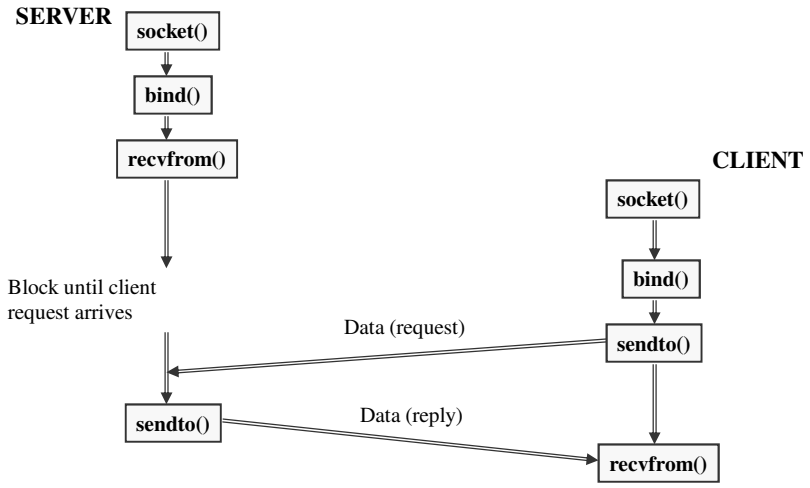
Client-server Model (contd.)

- Roles of the client and the server processes are asymmetric.
- Two types of servers:
  - ***Iterative servers***: Used when the server process knows in advance how long it takes to handle each request and it handles each request itself.
  - ***Concurrent servers***: Used when the amount of work required to handle a request is unknown; the server starts another process to handle each request.

System Calls for Connection-oriented Protocol



System Calls for Connectionless Protocol



References

- **Unix Network Programming**
  W.R.Stevens, Prentice-Hall of India, 1992.
- **Internetworking with TCP/IP (Volume I,II,III)**
  D.E.Comer and D.L.Stevens, Prentice-Hall of India, 1995.
- **http://www.ecst.csuchico.edu/~beej/guide/net**