# EECS 678: Introduction to Operating Systems
# Sockets: Course Notes For Reference

## Socket Abstraction

In Unix, the *open* function creates a file descriptor (integer value) that can be used to access the file. Like files, each socket is identified by a integer called its socket descriptor. Unix allocates socket descriptors in the same descriptor table as file descriptors. Reading / writing to a socket can be done the same way they are done with files.

Once a socket is created, it can be used to wait for an incoming connection (server operation) or to initiate a connection (client operation).

A socket is an end point for communication. The end point is identified by the machine's IP address and the protocol port number. Before the sockets are used for communication, information such as end point address, protocol type, etc. should be provided. (these parameters are discussed under system calls)

Figure 1 illustrates how the socket system calls are used in a client-server communication session.

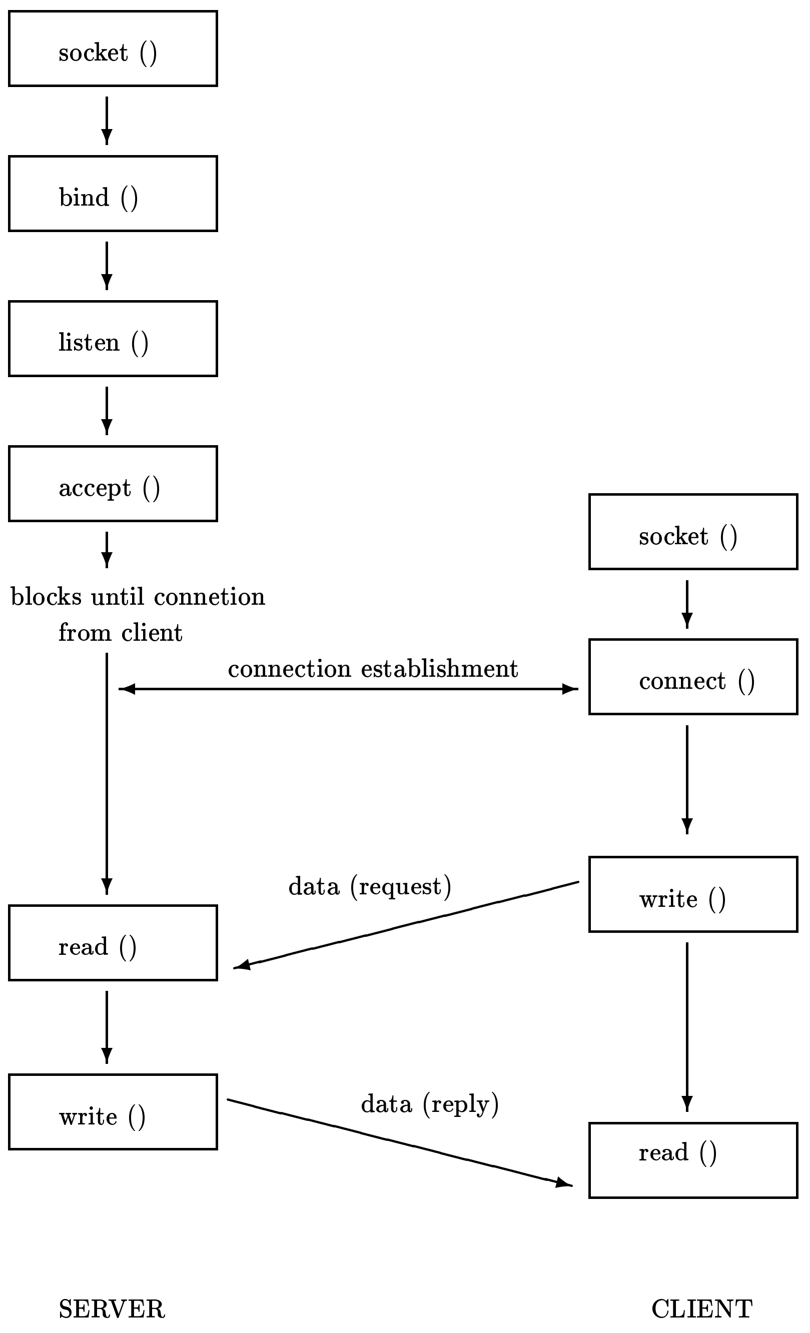

Figure 1. Socket system calls in a connection oriented client-server communication session

## Elementary Socket System Calls

Described below are some of the elementary system calls required to perform network programming using sockets.

## The *socket* System Call

`Usage`

```
int socket(int FAMILY, int TYPE, int PROTOCOL);
```

For TCP/IP, PF_INET FAMILY is used as the protocol family; type of service can be stream or datagram, denoted by SOCK_STREAM and SOCK_DGRAM respectively. For a socket that uses the Internet protocol family, the protocol or type of service argument determines whether the socket will use TCP or UDP.

### The *bind* System Call

The *bind* system call assigns a name to an unnamed socket.

`Usage`

```
int bind(int SOCKFD, struct sockaddr *MYADDR, int ADDRLEN);
```

The second argument is a pointer to a protocol-specific address and the third argument is the size of the address structure.
Using *bind*:

- Servers register their well-known address with the system. It tells the system "this is my address and any messages received for this address are to be given to me." Servers need to do this before accepting client requests.
- A client can register a specific address for itself.

### The *connect* System Call

A client process connects a socket descriptor by using the *connect* system call to establish a connection with a server.

`Usage`

```
int connect(int SOCKFD, struct sockaddr *SERVADDR, int ADDRLEN);
```

The second and third arguments are a pointer to a socket address and its size as described earlier.

## The *listen* System Call

This system call is used by a connection-oriented server to indicate that it is willing to receive connections.

Usage

```
  int listen(int SOCKFD, int BACKLOG);
```

The *listen* system call is usually executed after both the *socket* and *bind* system calls are executed, and immediately before the *accept* system call is executed. The BACKLOG argument specifies how many connection requests can be queued by the system while it waits for the server to execute the *accept* system call.

## The *accept* System Call

After a connection-oriented server executes the *listen* system call, an actual connection from a client process is waited for by having the server execute the *accept* system call.

Usage

```
  int accept(int SOCKFD, struct sockaddr, *PEER, int *ADDRLEN);
```

The *accept* system call takes the first connection request on the queue and creates another socket with the same properties as SOCKFD. If there are no connection requests pending, this call blocks the caller until one arrives.

The PEER and ADDRLEN arguments are used to return the address of the connected peer process (the client).

## The *close* System Call

The *close* system call is used to close a socket.

Usage

```
  int close(int SOCKFD);
```

## System Data Structures

The socket software provides corresponding structure declarations. Each TCP/IP endpoint address consists of a 2-byte field that identifies the address type, a 2-byte port number field, a 4-byte IP address field, and an 8-byte field that remains unused. Predefined structure *sockaddr_in* specifies the format.

```
struct sockaddr_in {            /* struct to hold an address */
  u_short sin_family;           /* type of address           */
  u_short sin_port;             /* protocol port number      */
  struct in_addr sin_addr;      /* 32 bit netid/hostid       */
                                /* network byte ordered      */
  char sin_zero[8];             /* unused                    */
};

struct in_addr {
  u_long s_addr;                /* 32 bit netid/hostid       */
};
```

## Looking Up A Domain Name

A client must specify the address of a server using structure *sockaddr_in.* Doing this requires converting an address in dotted decimal notation or a domain name in text form into a 32 bit IP address represented in binary. The *inet_addr* and *gethostbyname* library routines perform this conversion. The former takes an ASCII string that contains a dotted decimal address and returns an IP address in binary. The latter takes an ASCII string that contains the domain name for a machine and returns the address of a *hostent* structure that contains, among other things, the host's IP address in binary. The *hostent* structure is defined below.

```
struct hostent {
  char *h_name;                /* official host name               */
  char **h_aliases             /* other aliases                    */
  int h_addrtype;              /* host address type                */
  int h_length;                /* length of address                */
  char **h_addr_list;          /* list of addresses from name server */
                               /* a NULL terminates the list       */
};
#define h_addr h_addr_list[0]   /* first address in list            */
```

For Internet addresses, the array of pointers `h_addr_list[0]`, `h_addr_list[1]`, and so on, are not pointers to characters, but are pointers to structures of type `in_addr` defined above.

## Network Byte Order

TCP/IP specifies a standard representation for binary integers used in protocol headers. The representation, known as *network byte order*, represents integers with the most significant byte first (big endian). Sending machines are required to translate from the local integer representation to network byte order, and receiving machines are required to translate from network byte order to the local machine representation.

The following four functions handle the potential byte order differences between different computer architectures.

`Usage`

```
  u_long htonl(u_long hostlong);
  u_short htons(u_short hostshort);
  u_long ntohl(u_long netlong);
  u_short ntohs(u_short netshort);
```

- `htonl` : converts host to network, long integer.
- `htons` : converts host to network, short integer.
- `ntohl` : converts network to host, long integer.
- `ntohl` : converts network to host, short integer.

The example below shows a client program and a server program.

NOTE: The error handling is omitted from the code in order to make the program easier to understand. Students, however are advised to include the error handling in their programs.

NOTE: The library procedure *bzero* is used to place bytes containing zeros in a block of memory. It is a fast way to zero a large structure of array.

For more information read *man* pages.

```
/*
* desc : example client program.
*/

#define PORTNO 3456

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

static int str2IpAddr(char *anIpName){
  /* input   : hostname in ip format such as wigner.tisl.ukans.edu
   * returns : 0 if any error ipaddr otherwise
   * output  : -
   * desc    : resolves an ip name.
   */

  struct hostent *hostEntry;
  struct in_addr *scratch;

  if ((hostEntry = gethostbyname ( anIpName )) == (struct hostent*) NULL)
    return 0;

  scratch = (struct in_addr *) hostEntry->h_addr;

  return (ntohl(scratch->s_addr));

}
```

```
int main (int argc, char *argv[]){

  struct sockaddr_in peerAddress, ownAddress;
  int sessionSocket;
  char data[256] ="";
  char hostname[32];

  bzero((char *)&peerAddress, sizeof(peerAddress));
  bzero((char *)&ownAddress, sizeof(ownAddress));

  sessionSocket = socket (AF_INET, SOCK_STREAM, 0);

  gethostname (hostname, 32);
  ownAddress.sin_port = htons(0);
  ownAddress.sin_family = AF_INET;
  ownAddress.sin_addr.s_addr = htonl(str2IpAddr (hostname));
  bind (sessionSocket, (struct sockaddr *)&ownAddress, sizeof(ownAddress));

  peerAddress.sin_addr.s_addr = htonl (str2IpAddr(argv[1]));
  peerAddress.sin_port = htons (PORTNO);
  peerAddress.sin_family = AF_INET;
  connect (sessionSocket, (struct sockaddr *)&peerAddress, sizeof (peerAddress));
  read (sessionSocket, data, 256);

  printf ("%s\n",data);

  close (sessionSocket);

  return 0;

}


/*
 * desc : example server program
 */

#define PORTNO 3456

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
```

```
static int str2IpAddr(char *anIpName){
  /* input   : hostname in ip format such as wigner.tisl.ukans.edu
   * returns : 0 if any error ipaddr otherwise
   * output  : -
   * desc    : resolves an ip name.
   */

  struct hostent *hostEntry;
  struct in_addr *scratch;

  if ((hostEntry = gethostbyname ( anIpName )) == (struct hostent*) NULL)
        return 0;

  scratch = (struct in_addr *) hostEntry->h_addr;

  return (ntohl(scratch->s_addr));

}

int main (int argc, char *argv[]){

  struct sockaddr_in ownAddress, peerAddress;
  int serverSocket, sessionSocket, size;
  char data[256] = "Hi, This is server speaking...\n";
  char hostname[32];

  bzero((char *)&peerAddress, sizeof(peerAddress));
  bzero((char *)&ownAddress, sizeof(ownAddress));
  serverSocket = socket (AF_INET, SOCK_STREAM, 0);

  gethostname (hostname, 32);
  ownAddress.sin_port = htons(PORTNO);
  ownAddress.sin_family = AF_INET;
  ownAddress.sin_addr.s_addr = htonl(str2IpAddr (hostname));
  bind (serverSocket, (struct sockaddr *)&ownAddress, sizeof(ownAddress));

  sessionSocket = accept (serverSocket, (struct sockaddr *)&peerAddress, &size);

  close (serverSocket);

  write (sessionSocket, data, 256);

  close (sessionSocket);

  return 0;

}
```

## Reference

Stevens, W. Richard, 1990, *UNIX NETWORK PROGRAMMING*, Prentice Hall, Englewood Cliffs, N.J., 1990.
Comer, Douglas and Stevens, David, 1996, *INTERNETWORKING WITH TCP/IP - Volume 3*, Prentice Hall, Upper Saddle River, N.J., 1996.