Application Programming Interface (Sockets)

- Socket Interface was originally provided by the Berkeley distribution of Unix
  - Now supported in virtually all operating systems
- Each protocol provides a certain set of *services*, and the API provides a syntax by which those services can be invoked in this particular OS

Sockets

- What is a socket?
  - The point where a local application process attaches to the network
  - An interface between an application and the network
  - An application creates the socket
- The interface defines operations for
  - Creating a socket
  - Attaching a socket to the network
  - Sending and receiving messages through the socket
  - Closing the socket

Sockets

- Socket Family
  - PF_INET denotes the Internet family
  - PF_UNIX denotes the Unix pipe facility
  - PF_PACKET denotes direct access to the network interface (i.e., it bypasses the TCP/IP protocol stack)
- Socket Type
  - SOCK_STREAM is used to denote a byte stream
  - SOCK_DGRAM is an alternative that denotes a message oriented service, such as that provided by UDP

Sockets

- Socket Protocol – specifies protocol to use
  - Non zero value specifies a protocol supported by the specified socket family
  - 0 invokes the default protocol used by the socket family and type specified.

Creating a Socket

Chapter 1

```
int sockfd = socket(address_family, type, protocol);
```

- The socket number returned is the socket descriptor (sockfd) for the newly created socket

- `int sockfd = socket (PF_INET, SOCK_STREAM, 0);`
- `int sockfd = socket (PF_INET, SOCK_DGRAM, 0);`

The combination of PF_INET and SOCK_STREAM implies TCP
The combination of PF_INET and SOCK_DGRAM implies UDP

MK MORGAN KAUFMANN 5

Client-Serve Model with TCP

Chapter 1

Server

- Passive open
- Prepares to accept connection, does not actually establish a connection

Server invokes

```
int bind (int socket, struct sockaddr *address,
                                    int addr_len)
int listen (int socket, int backlog)
int accept (int socket, struct sockaddr *address,
                            int *addr_len)
```

MK MORGAN KAUFMANN 6

Client-Serve Model with TCP: Bind - Server

`int bind (int socket, struct sockaddr *address,int addr_len)`

- Binds the newly created socket to the specified address i.e. the network address of the local participant (the server)
- **int socket**: file descriptor of the socket being bound (from socket call)
- **struct sockaddr *address** is a data structure which combines IP and port
- **int addr_len** specifies the length of the sockaddr structure
- Upon successful completion, returns 0. Unsuccessful completion returns -1

Client-Serve Model with TCP: Listen-Server

`int listen (int socket, int backlog)`

- **int socket**: file descriptor of the socket being bound (from socket call)
- **int backlog:** number of outstanding connections in the sockets listen queue. Defines how many connections can be pending on the specified socket
- Return value of 0 indicates success, return value of -1 indicates an error occurred

Client-Serve Model with TCP: Accept-Server

- Accept
  - Carries out the passive open
  - Blocking operation
    - Does not return until a remote participant has established a connection
    - When it does, it returns a new socket that corresponds to the new established connection and the address argument contains the remote participant's address

Client-Serve Model with TCP: Accept-Server


```
int accept (int socket, struct sockaddr
  *address,int *addr_len)
```

- **int socket**: file descriptor of the socket being bound (from socket call)
- **struct sockaddr *address** is a data structure which combines IP and port
- **int *addr_len** sends in the length of the supplied address, on output from the function it contains the length of the stored address
- Return value on successful acceptance is the file descriptor of the accepted socket. Return value of -1 indicates an error occurred.

Client-Serve Model with TCP

Client

- Application performs active open
- It says who it wants to communicate with

Client invokes

```
int connect (int socket, struct sockaddr *address,
                                int addr_len)
```

Connect

- Does not return until TCP has successfully established a connection at which time application is free to begin sending data
- Address contains remote machine's address

Client-Serve Model with TCP: Connect-client


```
int connect (int socket, struct sockaddr *address,
                                int addr_len)
```

- **int socket**: file descriptor of the socket being bound (from socket call)
- **struct sockaddr *address** is a data structure which combines IP and port – address of machine to connect to
- **int *addr_len** the length of the supplied address
- Return value of 0 on successful completion. Return value of -1 indicates an error occurred.

Client-Serve Model with TCP

In practice

- The client usually specifies only remote participant's address and let's the system fill in the local information
- Whereas a server usually listens for messages on a well-known port
- A client does not care which port it uses for itself, the OS simply selects an unused one

Client-Serve Model with TCP

Once a connection is established, the application process invokes two operation

```
int send (int socket, char *msg, int msg_len,
                                        int flags)

int recv (int socket, char *buff, int buff_len,
                                          int flags)
```

Client-Serve Model with TCP - send


```
int send (int socket, char *msg, int msg_len,
          int flags)
```

- **int socket**: file descriptor of the socket being bound (from socket call)
- **char *msg** points to the buffer containing the message to send
- **int msg_len** the length of the message in bytes
- **Int flags** specifies type of message transmission
- Return value of -1 if a local error is detected. No guarantee of delivery status.
- If message is too long to send via the specified protocol, send fails and no information is sent

Client-Serve Model with TCP - recv


```
int recv (int socket, char *buff, int buff_len,
          int flags)
```

- **int socket**: file descriptor of the socket being bound (from socket call)
- **char *buff** points to the buffer where the message received is to be stored
- **int buff_len** the length of the storage buffer in bytes
- **Int flags** specifies type of message reception
- Return value is the length of the message in bytes. Return value of 0 if no messages are available and a shutdown has been received. Return value of -1 if an error is detected. No guarantee of delivery status.

Example Application: Client


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

#define SERVER_PORT 5432
#define MAX_LINE 256

int main(int argc, char * argv[])
{
        FILE *fp;
        struct hostent *hp;
        struct sockaddr_in sin;
        char *host;
        char buf[MAX_LINE];
        int s;
        int len;
        if (argc==2) {
                host = argv[1];
        }
        else {
                fprintf(stderr, "usage: simplex-talk host\n");
        exit(1);
        }
```


Example Application: Client


```
        /* translate host name into peer's IP address */
        hp = gethostbyname(host);
        if (!hp) {
                fprintf(stderr, "simplex-talk: unknown host: %s\n", host);
                exit(1);
        }
        /* build address data structure */
        bzero((char *)&sin, sizeof(sin));
        sin.sin_family = AF_INET;
        bcopy(hp->h_addr, (char *)&sin.sin_addr, hp->h_length);
        sin.sin_port = htons(SERVER_PORT);
        /* active open */
        if ((s = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
                perror("simplex-talk: socket");
                exit(1);
        }
        if (connect(s, (struct sockaddr *)&sin, sizeof(sin)) < 0) {
                perror("simplex-talk: connect");
                close(s);
                exit(1);
        }
        /* main loop: get and send lines of text */
        while (fgets(buf, sizeof(buf), stdin)) {
                buf[MAX_LINE-1] = '\0';
                len = strlen(buf) + 1;
                send(s, buf, len, 0);
        }
}
```

Example Application: Server


```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#define SERVER_PORT 5432
#define MAX_PENDING 5
#define MAX_LINE 256

int main()
{
        struct sockaddr_in sin;
        char buf[MAX_LINE];
        int len;
        int s, new_s;
        /* build address data structure */
        bzero((char *)&sin, sizeof(sin));
        sin.sin_family = AF_INET;
        sin.sin_addr.s_addr = INADDR_ANY;
        sin.sin_port = htons(SERVER_PORT);

        /* setup passive open */
        if ((s = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
                perror("simplex-talk: socket");
                exit(1);
        }
```


Example Application: Server


```
        if ((bind(s, (struct sockaddr *)&sin, sizeof(sin))) < 0) {
                perror("simplex-talk: bind");
                exit(1);
        }
        listen(s, MAX_PENDING);
        /* wait for connection, then receive and print text */
        while(1) {
                if ((new_s = accept(s, (struct sockaddr *)&sin, &len)) < 0) {
                perror("simplex-talk: accept");
                exit(1);
        }
        while (len = recv(new_s, buf, sizeof(buf), 0))
                fputs(buf, stdout);
                close(new_s);
        }
}
```

Socket Web pages

- The following are web pages for the various aspects of socket function descriptions:
  - http://pubs.opengroup.org/onlinepubs/009695399/functions/socket.html
  - http://pubs.opengroup.org/onlinepubs/009695399/functions/bind.html
  - http://pubs.opengroup.org/onlinepubs/009695399/functions/send.html
  - http://pubs.opengroup.org/onlinepubs/009695399/functions/listen.html
  - http://pubs.opengroup.org/onlinepubs/009695399/functions/accept.html
  - http://pubs.opengroup.org/onlinepubs/009695399/functions/recv.html
  - http://pubs.opengroup.org/onlinepubs/009695399/functions/connect.html
  - http://beej.us/guide/bgnet/output/html/multipage/htonsman.html
  - Following is a complete tutorial
  - http://beej.us/guide/bgnet/output/html/multipage/index.html