

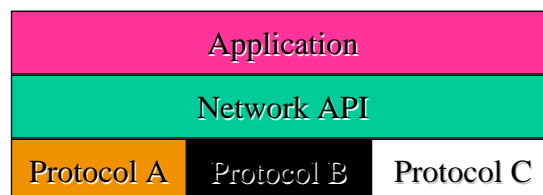
Windows Socket API

Seong Jong Choi
chois@uos.ac.kr

[Multimedia Lab.](#)
Dept. of Electrical and Computer Eng.
University of Seoul
Seoul, Korea

Network Application Programming Interface (API)

- The services provided by the operating system that provide **the interface between application and protocol software**.
- Support multiple communication protocol suites (families).
- Address (endpoint) representation independence.



TCP Sockets Programming

1. Create a passive mode (server) socket.
2. Create a client socket
3. The client establishes a connection.
4. send/receive data.
5. Terminate the connection

TCP/IP

- TCP/IP does not include an API definition.
- There are a variety of API s for use with TCP/IP:
 - Sockets
 - TLI , XTI
 - Winsock
 - MacTCP

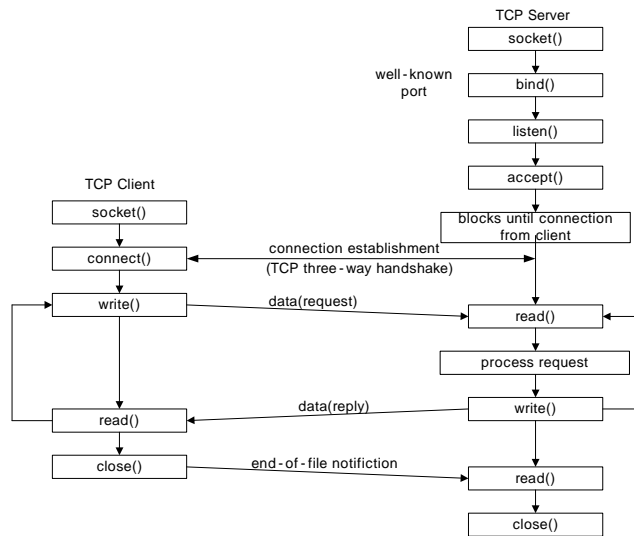
Functions needed

- Specify local and remote communication endpoints
- Initiate a connection
- Wait for incoming connection
- Send and receive data
- Terminate a connection gracefully
- Error handling

Berkeley Socket API

- Developed by Berkeley Software Distribution (BSD)
- Generic:
 - support for multiple protocol families.
 - address representation independence
- Uses existing I/O programming interface as much as possible.

Sockets Functions for TCP Client-server



Socket functions for elementary TCPclient-server

2005-11-11

Seong Jong Choi

Winsock API-7

Socket

- A socket is an abstract representation of a communication endpoint.
- Sockets work with Unix I/O services just like files, pipes & FIFOs.
- Sockets (obviously) have special needs:
 - establishing a connection
 - specifying communication endpoint addresses

2005-11-11

Seong Jong Choi

Winsock API-8

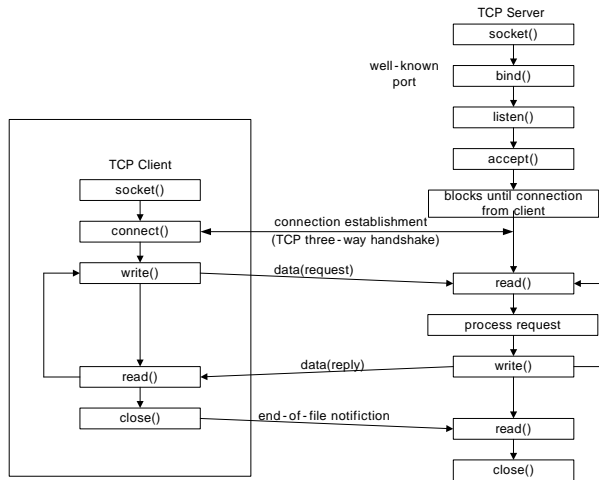
Windows Socket (or Winsock) API

- Network API by Microsoft.
- Nearly the same as Berkeley Socket API.
- Support both IPv4 and IPv6.
- Support overlapped I/O.
- Sockets are treated as handles to the kernel objects.

Client Code

- TCP clients can call `connect()` which:
 - takes care of establishing an endpoint address for the client socket (we don't need to call `bind`).
 - Attempts to establish a connection to the specified server.

Again



Socket functions for elementary TCPclient-server

EchoCli.C

```

int main(int argc, char *argv[])
{
    WSADATA WsaData;
    SOCKET sock;
    struct sockaddr_in serv_addr;
    int err;

```

```

    err = WSASStartup(0x0101, &WsaData);
    if (err == SOCKET_ERROR)
        FatalError("WSASStartup Failed");

```

```

    /*
     * Bind our local address
     */
    memset(&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    // Use the local host
    serv_addr.sin_addr.s_addr =
        inet_addr(SERVER_ADDRESS);
    serv_addr.sin_port =
        htons(SERVER_TCP_PORT);

```

```

    /*
     * Open a TCP socket (an Internet stream socket)
     */
    sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock < 0)
        FatalError("socket() failed -- do you have TCP/IP
        networking installed?");

```

```

    if (connect(sock, (struct sockaddr *) &serv_addr,
        sizeof(serv_addr)) < 0)
        FatalError("connect() failed -- is the server running?");

```

```

    DoClientLoop(stdin, sock);

```

```

    closesocket(sock);

    return EXIT_SUCCESS;
}

```

Winsock Initialization

```
int WSAStartup(  
    WORD wVersionRequired,  
    LPWSADATA lpWSAData,  
)
```

Return Value

- zero : system support the Winsock API
- non zero: Error

Note:

- WSA stands for "Windows Sockets Asynchronous".
- WSAStartup must be the first Winsock function a program calls
- When returns error, WSAGetLastError() or GetLastError() can be used

Specifying an Endpoint Address

```
struct sockaddr_in {  
    short sin_family; //Address family (must be AF_INET)  
    unsigned short sin_port; //IP port  
    struct in_addr sin_addr; //IP address  
    char sin_zero[8]; //Padding  
};  
typedef struct sockaddr_in SOCKADDR_IN
```

Note:

- Internet address family, the SOCKADDR_IN structure is used by Windows Sockets to specify a local or remote endpoint address to which to connect a socket.

in_addr

```
struct in_addr {  
    union {  
        struct { u_char s_b1,s_b2,s_b3,s_b4; } S_un_b;  
        struct { u_short s_w1,s_w2; } S_un_w;  
        u_long S_addr;  
    } S_un;  
};  
#define s_addr S_un.S_addr
```

inet_addr()

- Converting an IP address text string into the form required.

```
serv_addr.sin_addr.s_addr =  
inet_addr("203.249.107.60");
```

Network Byte Order

- All values stored in a `sockaddr_in` must be in network byte order (big-endian).
 - `sin_port` a TCP/IP port number.
 - `sin_addr` an IP address.

Network Byte Order Functions

- Provide conversion from host-dependent byte order to network byte order.

'h' : host byte order 'n' : network byte order
's' : u_short (16bit) 'l' : u_long (32bit)

- Host to network
 - u_short **htons** (u_short *hostshort*);
 - u_long **htonl** (u_long *hostlong*);
- Network to host
 - u_short **ntohs** (u_short *netshort*);
 - u_long **ntohl** (u_long *netlong*);

Creating a socket

```
SOCKET socket (int family, int type, int proto);
```

- `family` specifies the protocol family (PF_INET for TCP/IP).
- `type` specifies the type of service
 - SOCK_STREAM for TCP
 - SOCK_DGRAM for UDP.
- `protocol` specifies the specific protocol (usually 0 which means the default).
- `socket()` allocates resources needed for a communication endpoint - but it does not deal with endpoint addressing.
- Returns INVALID_SOCKET on failure.

Connect()

```
int connect( SOCKET s,  
             const struct sockaddr *server,  
             socklen_t addrlen);
```

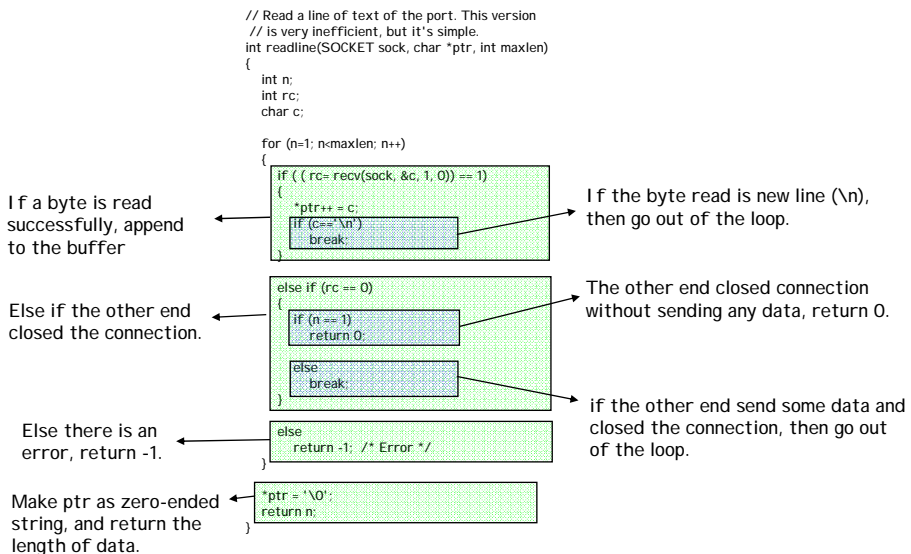
- `s` is an already created TCP socket.
- `server` contains the address of the server (IP Address and TCP port number)
- `connect()` returns 0 if OK, -1 on error

Closesocket()

```
int closesocket(SOCKET s);
```

- closes an existing socket
- If no error occurs, `closesocket` returns zero. Otherwise, a value of `SOCKET_ERROR` is returned, and a specific error code can be retrieved by calling [WSAGetLastError](#).

Example: Receiving a line with a socket



recv()

- The `recv()` receives data from a connected socket.
- **int** `recv`(
 SOCKET `s`,
 char FAR *`buf`, // [out] Buffer for the incoming data.
 int `len`, // [in] Length of `buf`.
 int `flags`
);
- Returns value
 - No error: the number of bytes received
 - **Connection closed: zero**
 - Error: `SOCKET_ERROR`
- Notes
 - `len` (length of `buf`) may be different from return value (length of received data)

Example: Sending a buffer with a socket

```
//Write bytes to the port with proper block size handling.
int writen(SOCKET sock, char *ptr, int nBytes)
{
    int nleft;
    int nwritten;

    nleft = nBytes;
    while (nleft > 0)
    {
        nwritten = send(sock, ptr, nBytes, 0);

        if (nwritten == SOCKET_ERROR)
        {
            fprintf(stdout, "Send Failed\n");
            exit(1);
        }

        nleft -= nwritten;
        ptr += nwritten;
    }

    return nBytes - nleft;
}
```

Send `nBytes` of data. The actual length of data is `nwritten`

If there is an error, exit program

`nleft`: length of data yet to be sent
`ptr`: starting address of the data to be sent

The program go out of the while loop only if there is no data to be sent, i.e., `nleft = 0`.

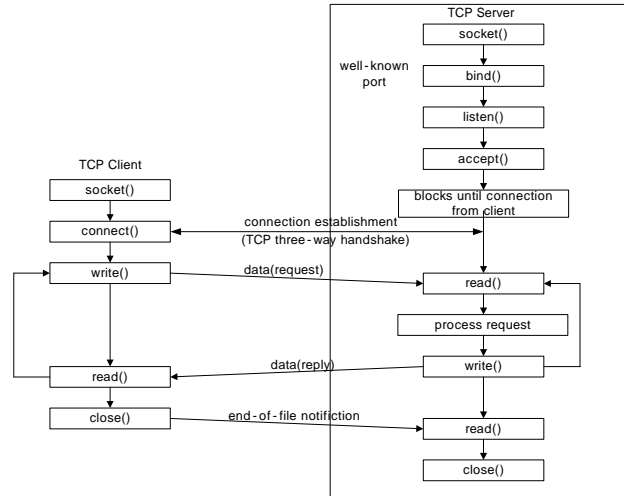
send()

- The `recv()` sends data on a connected socket.
- `int send(`
 - `SOCKET s,`
 - `const char FAR *buf,`
 - `int len,` // [in] Length of the data in *buf*.
 - `int flags``);`
- Returns value
 - No error: the number of bytes sent
 - Error: `SOCKET_ERROR`
- Notes
 - `len` (length of `buf`) may be different from return value (length of sent data)

Server Code

- Passive mode:
 - Address already determined.
 - Tell the kernel to accept incoming connection requests directed at the socket address (port number).
 - Tell the kernel to queue incoming connections for us.

Again



Socket functions for elementary TCP client-server

Example: EchoSrvSimple

```

int main(int argc, char *argv[])
{
    SOCKET listener;
    SOCKET newsocket;
    WSADATA WsaData;
    struct sockaddr_in serverAddress;
    struct sockaddr_in clientAddress;
    int clientAddressLength;
    int err;
    char pBuf[200];

    CheckOsVersion();

    err = WSASStartup(0x0101, &WsaData);
    if (err == SOCKET_ERROR)
    {
        FatalError("WSASStartup Failed");
        return EXIT_FAILURE;
    }

    listener = socket(AF_INET, SOCK_STREAM, 0);
    if (listener < 0)
    {
        FatalError("socket() failed");
        return EXIT_FAILURE;
    }

    /* Bind our local address */
    memset(&serverAddress, 0, sizeof(serverAddress));
    serverAddress.sin_family = AF_INET;
    serverAddress.sin_addr.s_addr = htonl(INADDR_ANY);
    serverAddress.sin_port = htons(SERV_TCP_PORT);

    err = bind(listener, (struct sockaddr *)&serverAddress,
              sizeof(serverAddress));
    if (err < 0)
        FatalError("bind() failed");

    listen(listener, 2);

    fprintf(stderr, "Simple Echo Server\n");
    fprintf(stderr, "Running on TCP port %d\n", SERV_TCP_PORT);
    fprintf(stderr, "nrpress Ctrl+C to stop the server\n");

    // Loop forever accepting requests new connections
    // and starting reading from them.
    for (;;)
    {
        clientAddressLength = sizeof(clientAddress);
        newsocket = accept(listener, (struct sockaddr *)&clientAddress,
                          &clientAddressLength);
        if (newsocket < 0)
        {
            FatalError("accept() Failed");
            return EXIT_FAILURE;
        }

        for (;;)
        {
            int len;
            len = read(newsocket, pBuf, 200);
            if (len == 0)
                break;
            else if (len == SOCKET_ERROR)
                exit(1);
            write(newsocket, pBuf, len);
        }

        closesocket(newsocket);
    }

    closesocket(listener);
    return 0;
}
  
```

bind()

- The **bind()** associates a local address (port number) with a socket.
- **int bind**(
 SOCKET *s*,
 const struct sockaddr FAR **name*,
 int *namelen*
);
- Returns value
 - No error: zero
 - Error: SOCKET_ERROR

listen()

- The **listen()** notifies a protocol that the process is prepared to accept incoming connection on a socket.
- **int listen**(
 SOCKET *s*,
 int **backlog**
);
- Returns value
 - No error: zero
 - Error: SOCKET_ERROR
- Note
 - The **backlog** specifies a limit on the number of connections that can be queued on the socket, after which the socket refuses to queue additional connection request

accept()

- The `accept()` waits for incoming connection request and **returns a connected socket**.
- SOCKET `accept`(
 SOCKET `s`, // listening socket
 struct sockaddr FAR *`addr`, // **[out] address of the other end**
 int FAR *`addrlen`
);
- Returns value
 - No error: new socket connected to a client
 - Error: `INVALID_SOCKET`
- Note
 - `accept()` is a blocking call.