The WinSock API

NOW WITH TCP!

Client's sequence of calls

- Initialize
- Make a socket
- Make a remote address
- **`connect()`** the socket to the address
- As desired:
  - **`recv()`** inbound bytes
  - **`send()`** outbound bytes
- Cleanup and shutdown

Create socket

```
SOCKET CreateSocket(int protocol)
{
  SOCKET result = INVALID_SOCKET;
  int type = SOCK_DGRAM;
  if (protocol == IPPROTO_TCP)
    type = SOCK_STREAM;

  result = socket(AF_INET, type, protocol);

  return result;
}
```

Create address

```
sockaddr_in* CreateAddress(char* ip, int port)
{
  sockaddr_in* result =
(sockaddr_in*)calloc(sizeof(*result), 1);

  result->sin_family = AF_INET;
  result->sin_port = htons(port);

  if (ip == NULL)
    result->sin_addr.S_un.S_addr = INADDR_ANY;
  else
    result->sin_addr.S_un.S_addr = inet_addr(ip);

  return result;
  // Caller will be responsible for free()
}
```

Connect

```
int Connect(SOCKET sock, sockaddr_in* address)
{
  if (connect(sock, (sockaddr*)address,
              sizeof(sockaddr_in)) == SOCKET_ERROR)
    return WSAGetLastError();
  else
    return 0;
}
```

For a TCP socket, `connect()` is ***blocking***!

Create non-blocking socket

```
SOCKET CreateSocket(int protocol)
{
   SOCKET result = INVALID_SOCKET;
   int type = SOCK_DGRAM;
   if (protocol == IPPROTO_TCP)
      type = SOCK_STREAM;

   result = socket(AF_INET, type, protocol);
   if (result != INVALID_SOCKET && protocol == IPPROTO_TCP)
   {
      int mode = 1;
      int err = ioctlsocket(result, FIONBIO, &mode);
      if (err != NO_ERROR)
      {
         closesocket(result);
         result = INVALID_SOCKET;
      }
   }
   return result;
}
```

Bind

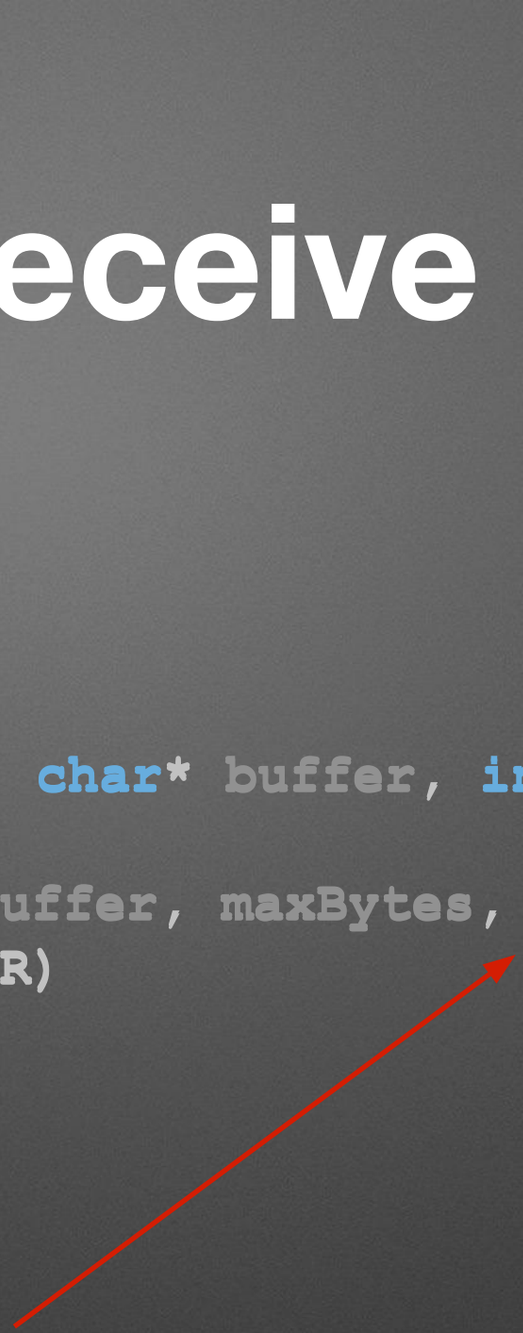
Don’t.

Receive

```
int Receive(SOCKET sock, char* buffer, int maxBytes, sockaddr* filterAddress)
{
   sockaddr sender;
   int size = sizeof(sockaddr);

   int bytes = recvfrom(sock, buffer, maxBytes, 0, &sender, &size);
   if (bytes == SOCKET_ERROR)
      return -1;

   if (bytes > 0)
   {
      if (filterAddress != NULL)
      {
         if (sender.sa_family != filterAddress->sa_family)
            return 0;

         for (int i = 0; i < size; i++)
         {
            if (sender.sa_data[i] != filterAddress->sa_data[i])
               return 0;
         }
      }
   }
   return bytes;
}
```

TCP Receive

```
int ReceiveTCP(SOCKET sock, char* buffer, int maxBytes)
{
   int bytes = recv(sock, buffer, maxBytes, 0);
   if (bytes == SOCKET_ERROR)
      return -1;
   return bytes;
}
```



- ~~MSG_WAITALL~~
- ~~MSG_PEEK~~
- ~~MSG_OOB~~

Send To

```
int Send(SOCKET sock, char* buffer,
                      int bytes, sockaddr_in* dest)
{
  int result = sendto(sock, buffer, bytes, 0,
            (sockaddr*)dest, sizeof(sockaddr_in));
  if (result == SOCKET_ERROR)
    return -1;
  else
    return result;
}
```

TCP Send

```
int SendTCP(SOCKET sock, char* buffer, int bytes)
{
  int result = send(sock, buffer, bytes, 0);
  if (result == SOCKET_ERROR)
    return -1;
  else
    return result;
}
```

Could be less than **bytes**!

Closure

```
void Close(SOCKET sock, boolean now)
{
  if (now)
    closesocket(sock);
  else
    shutdown(sock, SD_SEND);
}
```

Client's sequence of calls

- Initialize
- Make a socket
- Make a remote address
- **`connect()`** the socket to the address
- As desired:
  - **`recv()`** inbound bytes
  - **`send()`** outbound bytes
- Cleanup and shutdown

Server's sequence of calls

- Initialize
- Make a socket
- Make a local address
- **`bind()`** the socket to the address
- **`listen()`** for incoming connections
- As desired:
  - **`accept()`** an incoming connection on the listening socket, getting a new socket
  - **`recv()`** inbound bytes on an accepted socket
  - **`send()`** outbound bytes on an accepted socket
- Close the accepted socket
- Cleanup and shutdown

Create non-blocking socket

```
SOCKET CreateSocket(int protocol)
{
   SOCKET result = INVALID_SOCKET;
   int type = SOCK_DGRAM;
   if (protocol == IPPROTO_TCP)
      type = SOCK_STREAM;

   result = socket(AF_INET, type, protocol);
   if (result != INVALID_SOCKET && protocol == IPPROTO_TCP)
   {
      int mode = 1;
      int err = ioctlsocket(result, FIONBIO, &mode);
      if (err != NO_ERROR)
      {
         closesocket(result);
         result = INVALID_SOCKET;
      }
   }
   return result;
}
```

Create address

```
sockaddr_in* CreateAddress(char* ip, int port)
{
  sockaddr_in* result =
(sockaddr_in*)calloc(sizeof(*result), 1);

  result->sin_family = AF_INET;
  result->sin_port = htons(port);

  if (ip == NULL)
    result->sin_addr.S_un.S_addr = INADDR_ANY;
  else
    result->sin_addr.S_un.S_addr = inet_addr(ip);

  return result;
  // Caller will be responsible for free()
}
```

Bind

```
int Bind(SOCKET sock, sockaddr_in* addr)
{
  int size = sizeof(sockaddr_in);
  int result = bind(sock, (sockaddr*)addr, size);
  if (result == SOCKET_ERROR)
    return GetLastError();

  return 0;
}
```

Listen

```
int Listen(SOCKET sock, int backlog)
{
  int max = backlog;
  if (max < 1)
    max = SOMAXCONN;

  int result = listen(sock, max);
  if (result == SOCKET_ERROR)
    return GetLastError();

  return 0;
}
```

Accept

```
SOCKET Accept(SOCKET sock, sockaddr_in* incoming)
{
   int size = sizeof(sockaddr_in);
   SOCKET sock = accept(sock, incoming, &size);
   if (sock == INVALID_SOCKET)
      return -1; // WSAGetLastError will probably
                 // return WSAEWOULDBLOCK

   return sock;
}
```

TCP Send and Receive

```
int ReceiveTCP(SOCKET sock, char* buffer, int maxBytes)
{
   int bytes = recv(sock, buffer, maxBytes, 0);
   if (bytes == SOCKET_ERROR)
      return -1;
   return bytes;
}

int SendTCP(SOCKET sock, char* buffer, int bytes)
{
   int result = send(sock, buffer, bytes, 0);
   if (result == SOCKET_ERROR)
      return -1;
   else
      return result;
}
```

Always the result from an `accept()`, *never* your `listen()` socket! [`WSAENOTCONN`]

select()

Closure

```
void Close(SOCKET sock, boolean now)
{
  if (now)
    closesocket(sock);
  else
    shutdown(sock, SD_SEND);
}
```

Server's sequence of calls

- Initialize
- Make a socket
- Make a local address
- **`bind()`** the socket to the address
- **`listen()`** for incoming connections
- As desired:
  - **`accept()`** an incoming connection on the listening socket, getting a new socket
  - **`recv()`** inbound bytes on an accepted socket
  - **`send()`** outbound bytes on an accepted socket
- Close the accepted socket
- Cleanup and shutdown


easy