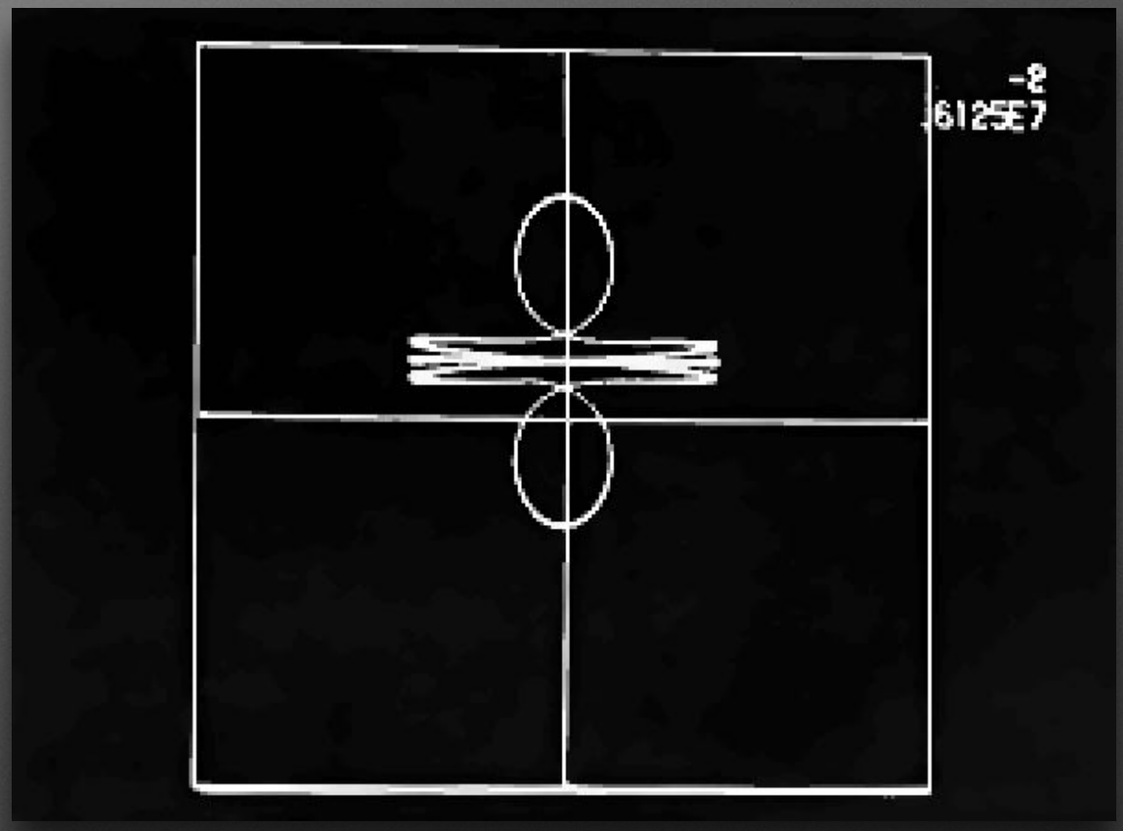
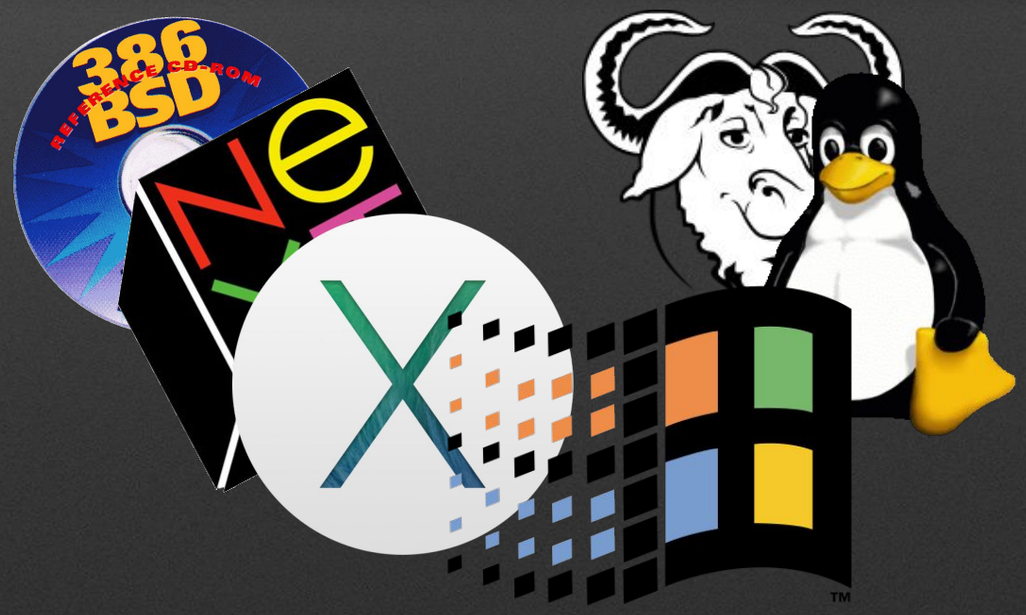
The WinSock API

SEE WHAT I DID THERE

EXTREMELY

Brief History of UNIX





`recvfrom()` = Berkeley

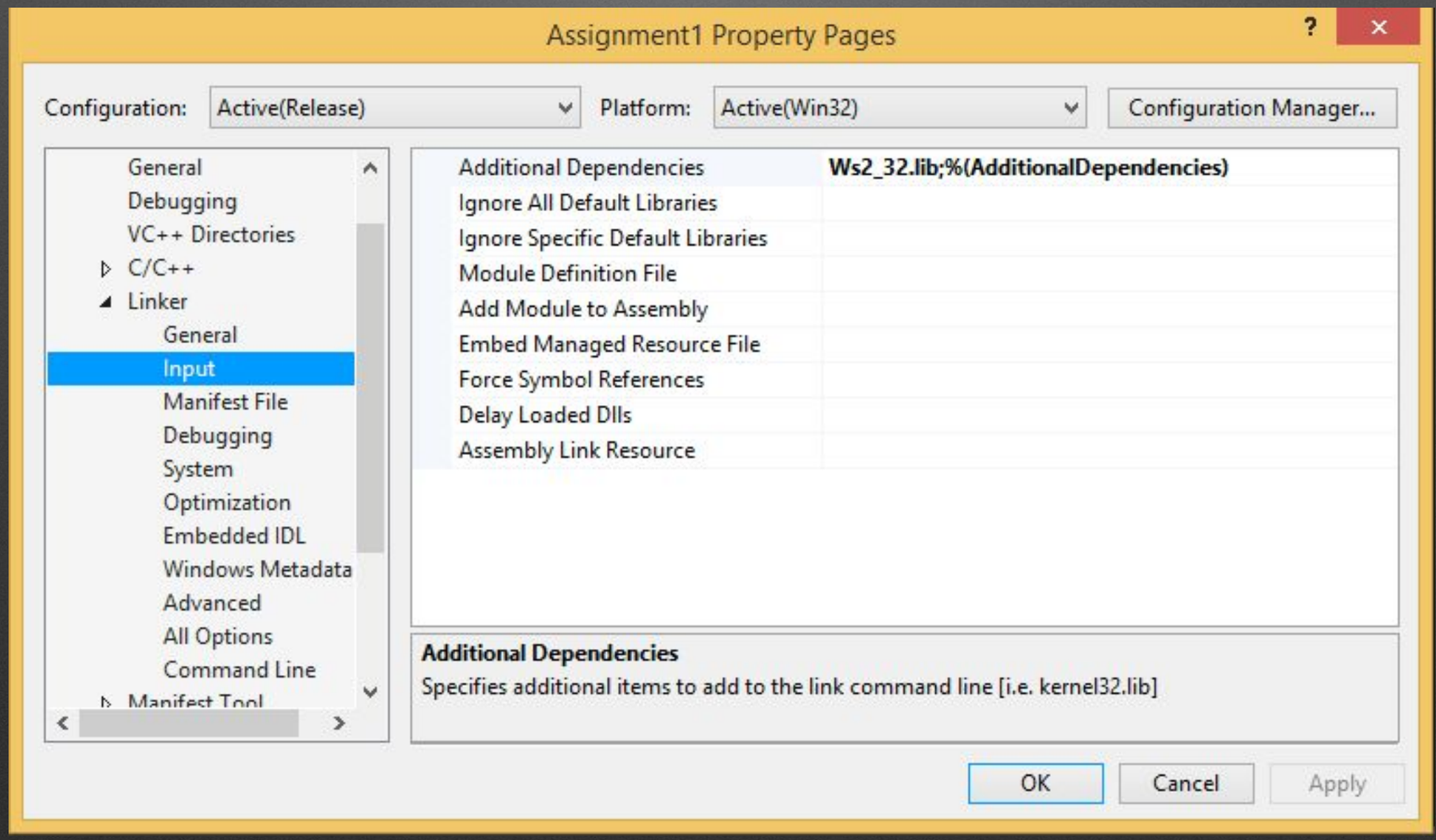
`WSAGetLastError()` = Windows

Basic sequence of calls

- Initialize
- Make a socket
- Make an address
- Bind the socket to the address
- As desired:
  - Check for inbound packets
  - Send outbound packets
- Cleanup and shutdown

Include Library

```
#include <WinSock2.h>
#include <Ws2tcpip.h>
```



WSAStartup

```
int WSAStartup(WORD wVersionRequested, LPWSADATA lpWSAData);
```

WSAGetLastError

```
int WSAGetLastError();
```

http://msdn.microsoft.com/en-us/library/windows/desktop/ms740668(v=vs.85).aspx

Create socket

```
SOCKET socket(int af, int type, int protocol);
```

Macro	Meaning
AF_UNSPEC	Unspecified
AF_INET	Internet Protocol Version 4
AF_IPX	Internetwork Packet Exchange: An early network layer protocol popularized by Novell and MS-DOS
AF_APPLETALK	Appletalk: An early network suite popularized by apple computer for use with its Apple and Macintosh computers
AF_INET6	I

```
SOCKET udpSocket = socket(AF_INET, SOCK_DGRAM, 0);
```

Macro	Meaning
SOCK_STREAM	Packets represent segments of an ordered, reliable stream of data
SOCK_DGRAM	Packets represent discrete datagrams
SOCK_RAW	Packet headers may be custom crafted by the application layer
SOCK_SEQPACKET	Similar to SOCK_STREAM but packets may need to be read in their entirety upon receipt

Macro	Required Type	Meaning
IPPROTO_UDP	SOCK_DGRAM	Packets wrap UDP datagrams
IPPROTO_TCP	SOCK_STREAM	Packets wrap TCP segments
IPPROTO_IP / 0	Any	Use the default protocol for the given type

Create address

```
struct sockaddr {
    uint16_t  sa_family;
    char      sa_data[14];
};
```

```
struct sockaddr_in {
    short     sin_family;
    uint16_t  sin_port;
    struct    in_addr sin_addr;
    char      sin_zero[8];
};
```

```
sockaddr_in myAddr;
memset(myAddr.sin_zero, 0, sizeof(myAddr.sin_zero));
myAddr.sin_family = AF_INET;
myAddr.sin_port = htons(80);
myAddr.sin_addr.S_un.S_un_b.s_b1 = 65;
myAddr.sin_addr.S_un.S_un_b.s_b2 = 254;
myAddr.sin_addr.S_un.S_un_b.s_b3 = 248;
myAddr.sin_addr.S_un.S_un_b.s_b4 = 180;
```

```
struct in_addr {
  union {
    struct {
      uint8_t s_b1,s_b2,s_b3,s_b4;
    } S_un_b;
    struct {
      uint16_t s_w1,s_w2;
    } S_un_w;
    uint32_t S_addr;
  } S_un;
};
```

Create address

```
int inet_pton(int af, const char* src, void* dst);
int InetPton(int af, const PCTSTR src  void* dst);
```

```
sockaddr_in myAddr;
myAddr.sin_family = AF_INET;
myAddr.sin_port = htons( 80 );
InetPton(AF_INET, "65.254.248.180", &myAddr.sin_addr);
```

Create address

```
class SocketAddress
{
public:
    SocketAddress(uint32_t inAddress, uint16_t inPort)
    {
        GetAsSockAddrIn()->sin_family = AF_INET;
        GetAsSockAddrIn()->sin_addr.S_un.S_addr = htonl(inAddress);
        GetAsSockAddrIn()->sin_port = htons(inPort);
    }
    SocketAddress(const sockaddr& inSockAddr)
    {
        memcpy(&mSockAddr, &inSockAddr, sizeof( sockaddr) );
    }

    size_t GetSize() const {return sizeof( sockaddr );}

private:
    sockaddr mSockAddr;

    sockaddr_in* GetAsSockAddrIn()
        {return reinterpret_cast<sockaddr_in*>( &mSockAddr );}
};
typedef shared_ptr<SocketAddress> SocketAddressPtr;
```

Byte Ordering

```
htond() double ntohd()

htonf() float  ntohf()

htonl() long ntohl()

htons() short  ntohs()


  WSAHtons(SOCKET s, u_short h, u_short*

pn)
```

Bind

```
int bind(SOCKET sock, const sockaddr *address, int address_len);
```

Send To

```
int sendto(SOCKET sock, const char *buf, int len, int flags,
const sockaddr *to, int tolen);
```

Receive From

```
int recvfrom(SOCKET sock, char *buf, int len, int flags,
sockaddr *from,
int *fromlen);
```


TOYOTA

Blocking Calls (UDP Version)

- Initialize
- Make a socket
- Make an address
- Bind the socket to the address
- As desired:
  - Check for inbound packets
  - Send outbound packets*…sometimes*
- Cleanup and shutdown

UDPSocket

```
class UDPSocket
{
public:
    ~UDPSocket();
    int Bind(const SocketAddress& inToAddress);
    int SendTo(const void* inData, int inLen, const SocketAddress& inTo);
    int ReceiveFrom(void* inBuffer, int inLen, SocketAddress& outFrom);
private:
    friend class SocketUtil;
    UDPSocket(SOCKET inSocket) : mSocket(inSocket) {}
    SOCKET mSocket;
};
typedef shared_ptr<UDPSocket> UDPSocketPtr;

int UDPSocket::Bind(const SocketAddress& inBindAddress)
{
    int err = bind(mSocket, &inBindAddress.mSockAddr,
            inBindAddress.GetSize());
    if(err != 0)
    {
        SocketUtil::ReportError(L"UDPSocket::Bind");
        return SocketUtil::GetLastError();
    }
    return NO_ERROR;
}

int UDPSocket::SendTo(const void* inData, int inLen,
                      const SocketAddress& inTo)
{
    int byteSentCount = sendto( mSocket,
                                static_cast<const char*>( inData),
                                inLen,
                                0, &inTo.mSockAddr, inTo.GetSize());
    if(byteSentCount >= 0)
    {
        return byteSentCount;
    }
    else
    {
        //return error as negative number
        SocketUtil::ReportError(L"UDPSocket::SendTo");
        return -SocketUtil::GetLastError();
    }
}

int UDPSocket::ReceiveFrom(void* inBuffer, int inLen,
                           SocketAddress& outFrom)
{
    int fromLength = outFromAddress.GetSize();
    int readByteCount = recvfrom(mSocket,
                                 static_cast<char*>(inBuffer),
                                 inMaxLength,
                                 0, &outFromAddress.mSockAddr,
                                 &fromLength);
    if(readByteCount >= 0)
    {
        return readByteCount;
    }
    else
    {
        SocketUtil::ReportError(L"UDPSocket::ReceiveFrom");
        return -SocketUtil::GetLastError();
    }
}

UDPSocket::~UDPSocket()
{
    closesocket(mSocket);
}
```

Closure

```
int closesocket( SOCKET sock );
```

WSACleanup

```
int WSACleanup();
```


easy