Multiplayer Game Programming

Chapter 3

Berkeley Socket API

UDP

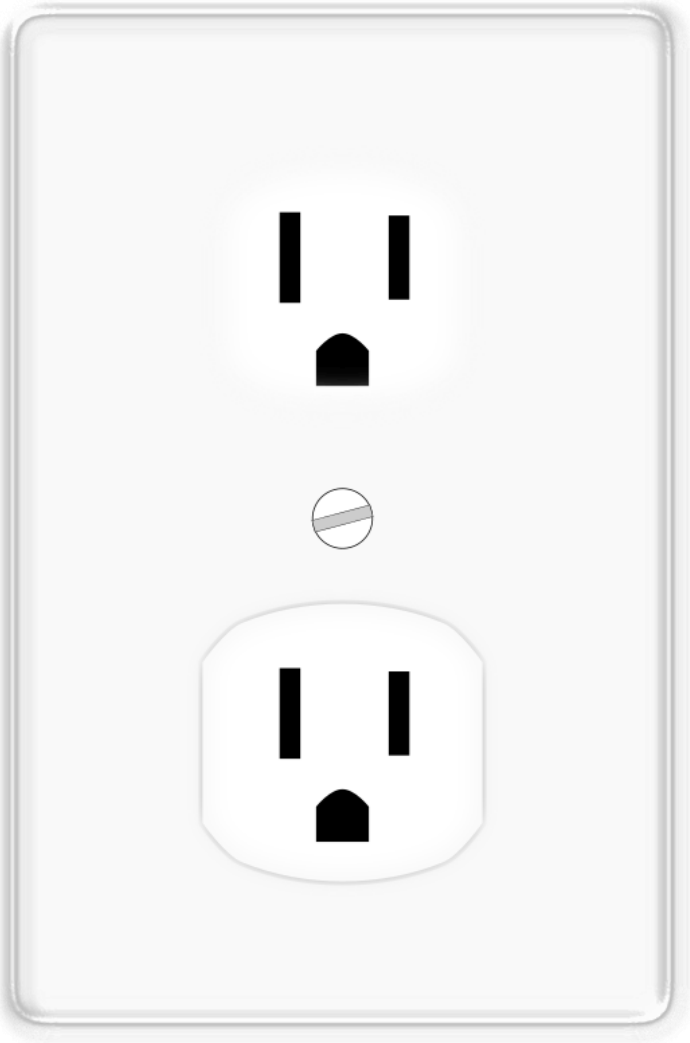
Chapter 3
Objectives

- **Socket creation and initialization**
  - How to create and bind a socket
  - Platform differences

- **Datagram transmission**
  - How to use sockets for UDP

Sockets

- **Berkeley Socket API**
- Originally from BSD UNIX 4.2
- Ported in some way to almost every language and operating system



Operating System Differences

- **POSIX: Linux, Mac OS X, iOS, Android, PlayStation**
  - <sys/socket.h>
  - Socket is a file descriptor (`int`).
- **Windows: Xbox**
  - WinSock2
  - Socket is a `SOCKET`.

```
#define WIN32_LEAN_AND_MEAN
#pragma comment (lib, "ws2_32.lib")

#include <windows.h>
#include <Winsock2.h>
```

Call order

```
//setup WSA
NetworkManager::GetInstance()->init();

// setup the sockets / create an inbound and outbound socket
NetworkManager::GetInstance()->setupSockets();

// bind the receiving socket. Outbound udp does not require binding
NetworkManager::GetInstance()->bindSockets();

// send out the data through udp using sendto()
NetworkManager::GetInstance()->sendData();

// recieve data using UDP rcvfrom()
NetworkManager::GetInstance()->receiveData();

// make sure we close our socket
NetworkManager::GetInstance()->shutdown();
```

Windows Housekeeping

- Windows requires explicit startup of WinSock2

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

- The first param is a WORD (2 bytes data segment)
  - You can populate this using MAKEWORD macro.
    - *MAKEWORD*(2,2);
    - 2.2 is the current version we will use.

- The second param points to a structure to be filled with data about the version you specify.
  - We don't really use or need this data, but it must be passed to the function and populated.

Windows Housekeeping

- **SHUTDOWN**:
  - Windows WinSock2 also requires explicit shutdown.
    - WSACleanup();
  - I created a Shutdown function which is called from the second last line of int main() to handle this.

```
int init()
{
    srand(time(NULL));

    WSADATA wsaInfo;
    // Initialize Winsock
    int errCode = WSAStartup(MAKEWORD(2, 2), &wsaInfo);

    if (errCode != 0)
    {
        cout << "WSAStartup failed with error: " << errCode;
        return 1;
    }

    return 0;
}

void shutdown()
{
    WSACleanup();
    exit(0);
}
```

Creating a Socket

```
SOCKET socket( int af, /*address family*/
               int type, /*socket type*/
               int protocol );
```

- Returns new socket if successful
- -1 if unsuccessful `INVALID_SOCKET`

```
// Assigning UDP socket so we have something to listen for
UDPINSocket = socket(AF_INET, SOCK_DGRAM, 0);
if (UDPINSocket == INVALID_SOCKET)
{
    cout << "Failed to create in socket." << endl;
    shutdown();
}
```

Address Family

Table 3.1 Address Family Values for Socket Creation

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	Internet Protocol Version 6

```
// Assigning UDP socket so we have something to listen for
UDPINSocket = socket(AF_INET, SOCK_DGRAM, 0);
if (UDPINSocket == INVALID_SOCKET)
{
    cout << "Failed to create in socket." << endl;
    shutdown();
}
```

Socket Type

- How will data be passed

Table 3.2 Type Values for Socket Creation

Macro	Meaning	
SOCK_STREAM	Packets represent segments of an ordered, reliable stream of data	TCP
SOCK_DGRAM	Packets represent discrete datagrams	UDP
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	

```
// Assigning UDP socket so we have something to listen for
UDPINSocket = socket(AF_INET, SOCK_DGRAM, 0);
if (UDPINSocket == INVALID_SOCKET)
{
    cout << "Failed to create in socket." << endl;
    shutdown();
}
```

Protocol

Table 3.3 Protocol Values for Socket Creation

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

```
// Assigning UDP socket so we have something to listen for
UDPINSocket = socket(AF_INET, SOCK_DGRAM, 0);
if (UDPINSocket == INVALID_SOCKET)
{
    cout << "Failed to create in socket." << endl;
    shutdown();
}
```

? Error Reporting

- When an API call returns a value of -1 or `INVALID_SOCKET` this indicates an error
  - Retrieve true error code in platform dependent manner
  - Do it quickly; another error in same thread will change value
- **Windows**:

```
int WSAGetLastError();
```

- POSIX:

```
int errno;
```

Binding Socket to Address

```
int bind( SOCKET sock, /* created this previously */
      const sockaddr *address, /*need to create*/
      int address_len );
```

- Registers address for use by socket.
  - Unrelated to the address to transmit the data through.
- Actual type of `sockaddr` can change based on protocol family and so on:
  - From a time before classes and inheritance.
  - Different types must be cast to `sockaddr`.

SOCKADDR

```
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];
};
```

- You may need to perform a reinterpret_cast<> to down convert a sockaddr_in to a sockaddr.

IN_ADDR IPv4 Address

```
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;
};
```

- Specify IP address as a 32-bit number, two 16-bit numbers, or four 8-bit numbers (most common).
- `INADDR_ANY` allows binding to all local interfaces on the host.

Bind Socket Example

- inAdd is of type `SOCKADDR_IN inAddr;`
  - Fill with data for communication
- htos & htonl: endian conversion (Next slide)

```
void NetworkManager::bindSockets()
{
    cout << "about to bind sockets" << endl;
    // using IP version 4
    inAddr.sin_family = AF_INET;
    // Port 8889
    inAddr.sin_port = htons(8889);
    // From any available address ( note computers can have multiple)
    inAddr.sin_addr.s_addr = htonl(INADDR_ANY);

    // Associate the address information with the socket using bind.
    // At this point you can receive datagrams on your bound socket.
    if (bind(UDPINSocket, reinterpret_cast<SOCKADDR *>(&inAddr), sizeof(inAddr)) == SOCKET_ERROR)
    {
        cout << "[ERROR] bind Error: " << WSAGetLastError() << endl;

        shutdown();
    }
}
```

Byte Order

- Network byte order is **big endian**.
- Our platform (windows) is **little endian**.
- Use conversion functions when manually filling in or reading address structures:

```
uint16_t htons( uint16_t hostshort );
uint16_t ntohs( uint16_t netshort);

uint32_t htonl( uint32_t hostlong );
uint32_t ntohl( uint32_t netlong);
```

Datagram Transmission

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

- Returns number of bytes sent or -1 for error
- to: Destination host address
  - Same format as when creating socket
- Most common way to send UDP Packets

Send Data Example

- inet_pton() is used to create the address to send to.
  - This takes in the family, address, and place to store it

```
#include <WS2tcpip.h> // needed for inet_pton

void NetworkManager::sendData()
{
    // setup where/how to send the data
    outAddr.sin_family = AF_INET;
    outAddr.sin_port = htons(8889);
    inet_pton(AF_INET, "127.0.0.1", &outAddr.sin_addr);

    cout << "about to sendData" << endl;
    // setup message data
    const char* messageText = "This is a test of UDP Transmission";
    int messageLen = MAX_MESSAGE_SIZE;

    //send some data
    int TotalByteSent = sendto(UDPOUTSocket, messageText, messageLen, 0,
                               reinterpret_cast<SOCKADDR *>(&outAddr), sizeof(outAddr));
    cout << " sent : " << TotalByteSent << " of data" << endl;
}
```

Datagram Reception

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

- Returns number of bytes received or -1 for error
- `from`: Output param assigned address of sending host
  - Do not fill with an address before invoking.
  - **Does not** select incoming datagram based on address.
    - **ONLY USED AS OUTPUT PARAM**
- Most common way to receive UDP packets.

Receive Data Example

```
void NetworkManager::receiveData()
{
    cout << "about to receiveData" << endl;
    // setup buffer to store data received when it comes in.
    int ByteReceived = 0;
    char ReceiveBuf[65535]; // this needs to be large enough for all the data
    int inAddrSize = sizeof(ReceiveBuf);

    // loop until we receive data
    while (ByteReceived <= 0)
    {
        cout << "waiting on Data: " << ByteReceived << endl;
        // call to recieve data on the socket previously bound
        ByteReceived = recvfrom(UDPINSocket, ReceiveBuf, sizeof(ReceiveBuf), 0,
                                reinterpret_cast<SOCKADDR *>(&inAddr), &inAddrSize);
        // check for error
        if (ByteReceived == SOCKET_ERROR)
        {
            cout << "[Error] receive error: " << WSAGetLastError() << endl;
        }
        cout << "received Data: " << ByteReceived << endl;
    }
}
```

Call order

```
//setup WSA
NetworkManager::GetInstance()->init();

// setup the sockets / create an inbound and outbound socket
NetworkManager::GetInstance()->setupSockets();

// bind the receiving socket. Outbound udp does not require binding
NetworkManager::GetInstance()->bindSockets();

// send out the data through udp using sendto()
NetworkManager::GetInstance()->sendData();

// recieve data using UDP rcvfrom()
NetworkManager::GetInstance()->receiveData();

// make sure we close our socket
NetworkManager::GetInstance()->shutdown();
```