Non-Blocking
Endianness & Class memory

Chapter 3-5

GAME 311- Network Programming

Chapter 3 & 4 Objectives

- **Blocking and nonblocking I/O**
  - How to use sockets in real time

- **Endian**
  - Describe what endianness is and why it's important to networking games.

Blocking

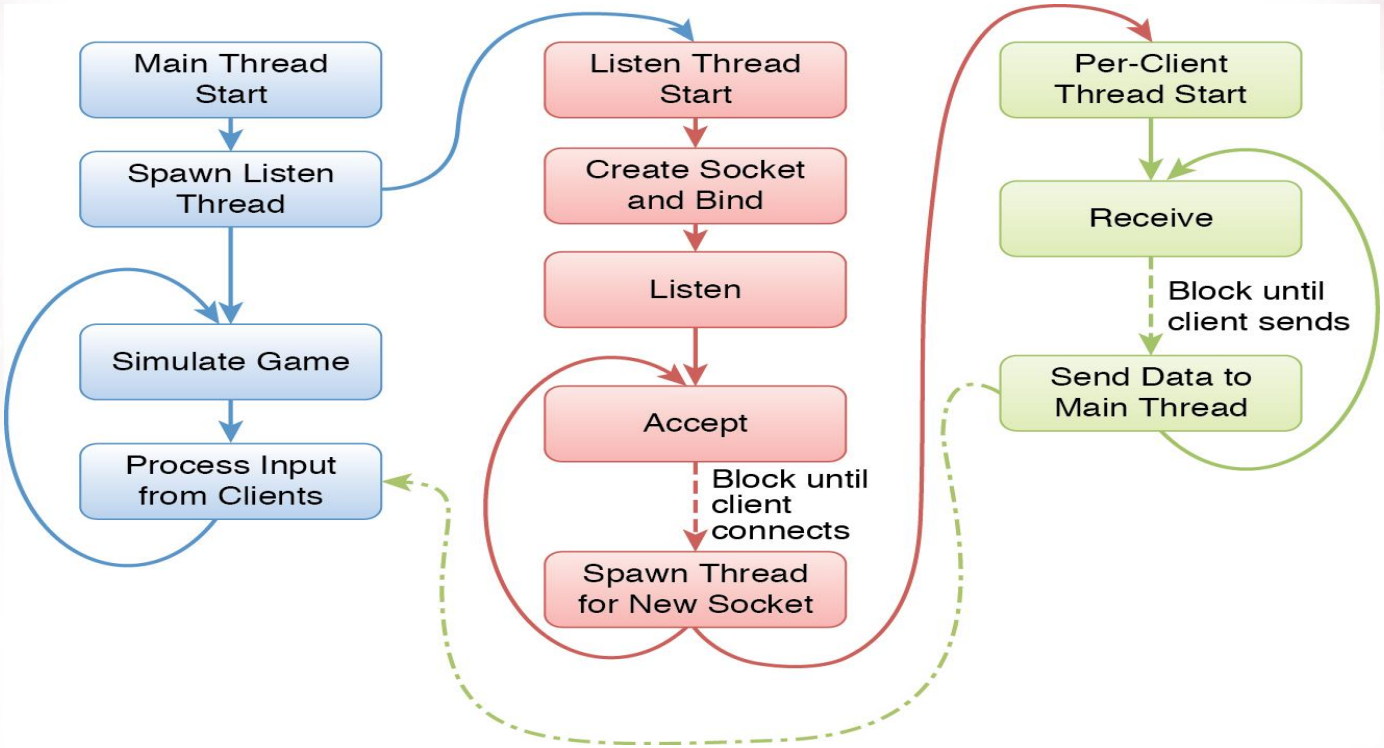
- Some actions halt thread or **block** waiting for completion:
  - UDP:
    - Receiving: When no incoming data available
  - TCP:
    - Sending: When no room left in the send buffers
    - Connecting: Until handshake complete
    - Accepting: Until handshake complete
- Problem for any non turn based game
- Problem when communicating with multiple remote hosts sending data

Working Around Blocking

- **Multithreading**
  - Use separate thread for each connection.

- **Nonblocking I/O**
  - Enable nonblocking mode and poll for updates.
  - Does not require new threads

Multithreading



- Does not scale well for large numbers of clients

Nonblocking I/O

- We can tell our socket to be non-blocking by using the ioctlsocket call:

```
// SERVER ONLY
void NetworkManager::AcceptConnections()
{
    int clientSize = sizeof(peerAddr);

    peerSocket = accept(listenSocket, reinterpret_cast<SOCKADDR *>(&peerAddr), &clien
    if (peerSocket != INVALID_SOCKET)
    {
        char ipConnected[32];
        inet_ntop(AF_INET, &peerAddr.sin_addr, ipConnected, 32);
        // print out who connected
        cout << ipConnected << " Just connected into the server" << endl;
    }

    unsigned long b = 1;
    ioctlsocket(peerSocket, FIONBIO, &b);
}
```

Nonblocking I/O

```
unsigned long b = 1;
ioctlsocket(peerSocket, FIONBIO, &b);
```

- **The ioctlsocket function takes in 3 params:**
  - **The socket to change the IO mode on.**
  - **The type of ctl to change**
    - **Our case here needs to be FIONBIO**
      - **IO – (NB) non-blocking IO**
  - The last param is a reference to a 1 or 0.
    - If the value is anything other than 0 it is non-blocking
    - 0 ensure it is blocking.
    - Without this call the accept and connect calls are blocking by default.

Nonblocking I/O

- **Nonblocking sockets** return control immediately when asked to do operations that would block.
  - Return value is -1 from previously blocking calls
    - Error code is indicative.
      - Windows: **WSAEWOULDBLOCK**
      - POSIX: **EAGAIN**
  - Can try again next frame to:
    - Send
    - Receive
    - Accept
    - Connect

Endian Compatibility

- Different CPUs store multibyte numbers in different formats.

- **Little endian**:
  - Least significant bytes first in memory
  - Intel x86, most iOS hardware, Xbox One, PS4

- **Big endian**:
  - Most significant bytes first in memory
  - PowerPC, XBox360, PS3

Endian Examples: 0x12345678

Little endian

Value =>	0x78	0x56	0x34	0x12
Address =>	0x01000000	0x01000001	0x01000002	0x01000003

Big endian

Value =>	0x12	0x34	0x56	0x78
Address =>	0x01000000	0x01000001	0x01000002	0x01000003

Byte Swapping

- For compatibility between hosts of different endianness.
  - Decide on consistent endianness of stream
  - If host endianness does not match stream endianness
    - Swap order of bytes in each multibyte number **before writing**

ONLY SWAP MULTIBYTE, ATOMIC UNITS.

- For example, don't swap an array of single byte characters.)

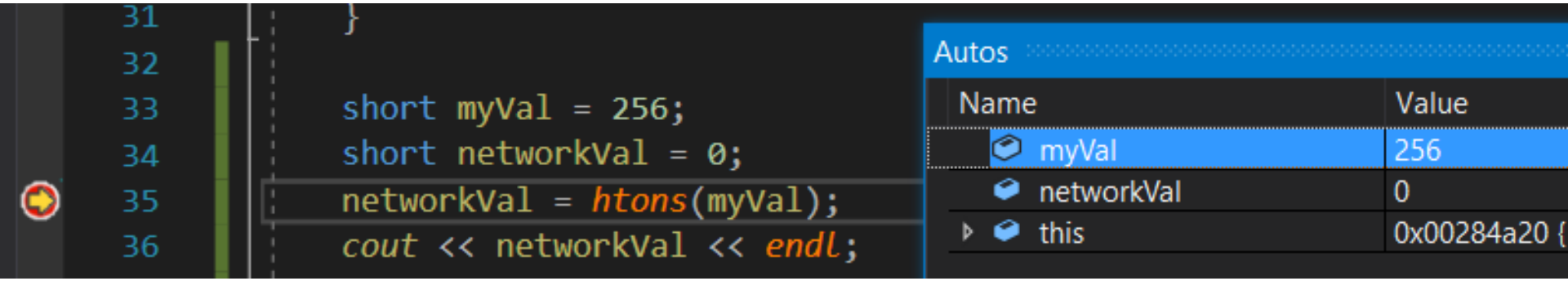
hto* conversion

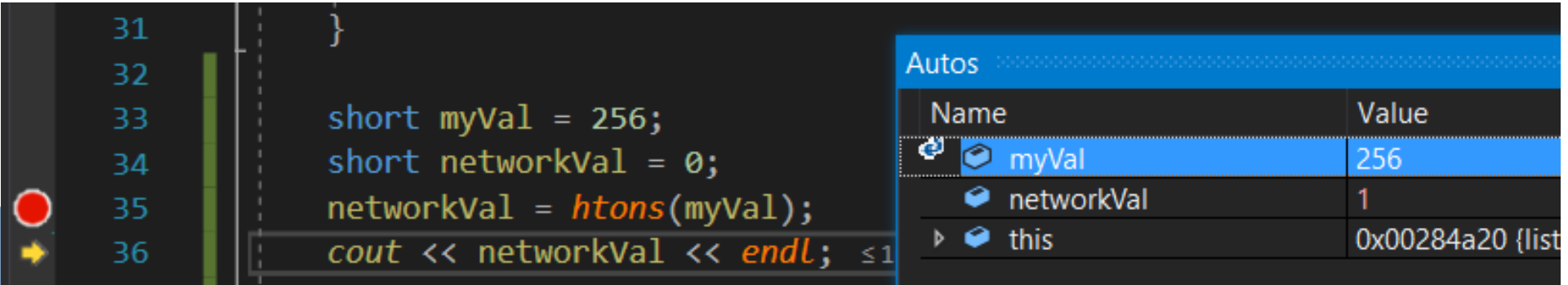
- We convert from host to “network” order using a series of conversion functions available from the Winsock2 API:
  - *Htons - short*
  - *Htonll – long long*
  - *Htonf - float*
  - *Htond – double*
  - *Etc…*

```
listenAddr.sin_family = AF_INET;
listenAddr.sin_port = htons(8890);
inet_pton(AF_INET, "10.105.156.53", &listenAddr.sin_addr);
```

Analysis of htons

- We can see the byte swapping taking place if we debug the values before and after:
- Note that the 256 value(00000001 00000000) is swapped to become 1 (00000000 00000001)





Ntoh* conversion

- In addition to converting data used for transfer we can also convert received data to its proper byte order using:
  - *ntohs - short*
  - *ntohll – long long*
  - *ntohf - float*
  - *ntohd – double*
  - *Etc…*

```
short myVal = 256;
short networkVal = 0;
networkVal = htons(myVal);
cout << networkVal << endl;
short receivedVal = ntohs(networkVal);
cout << receivedVal << endl;
```

Memory and Object Review

```
struct StructExample
{
    int a = 0;
    float c = 0.0f;
    bool b = false;
    short d = 0;
}
```

- int
  - 4
- float
  - 4
- bool
  - 1
- short
  - 2
  - 1 + padding
- ==========
  - Total 12(32 bit)

Class Size Evaluation

- Functions don't occupy memory
- Is there Static Variables?
  - They don't count
- Padding / Alignment
  - Same as the Structs
- Does the Class inherit members from a parent class?
  - Those need to be added too.
- Is there Virtual functions or virtual inheritance within the class?
  - The class will then have a hidden ptr assigned to it to reference these.
  - 4 byte virtual table Ptr
- More here:
- http://www.cprogramming.com/tutorial/size_of_class_object.html

Planning for Adjustments

- Presently our code is sending simple data to just update the paddles Y coordinates.
- What if we needed to send more than just a single position value?
  - What if we wanted to change the colour of the paddle based off events and allow the paddle to move in the x and y?
  - What if we added powerups to our pong game?
    - What would the reprocutions be?

Replicating Objects

- If we send each piece of data off one by one for the X, Y and paddle info like colour and size we could get offset in communication.
- All of a sudden we start accidentally mixing packets intended for the X location as the Y and data mixed up.
- The better option is to replicate entire objects at a time.
- Send all the information about one particular object as required.
- Instead of worrying about byte swapping, we can turn all of our data into a char array.

Chapter 3 & 4 Summary

- Explain why multithreading or non-blocking is a requirement of networked games
- Describe what endianness is and why it's important to networking games.
- Reviewed memory consumed by an instance of an object.