게임프로그래밍

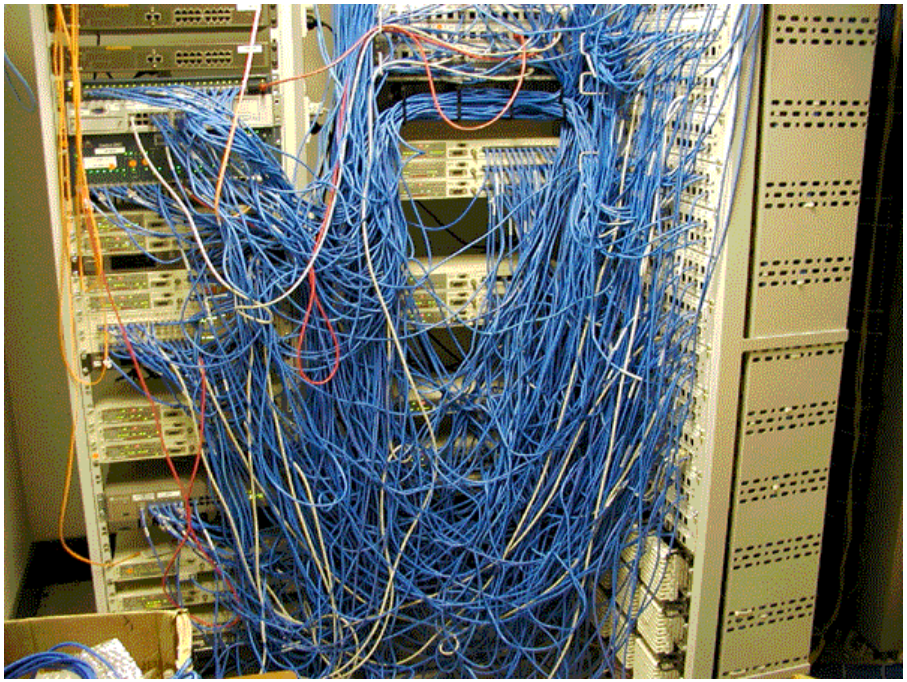
게임 서버

박종승

Dept. of CSE, Incheon Nat. Univ.
jong@inu.ac.kr
http://ecl.inu.ac.kr

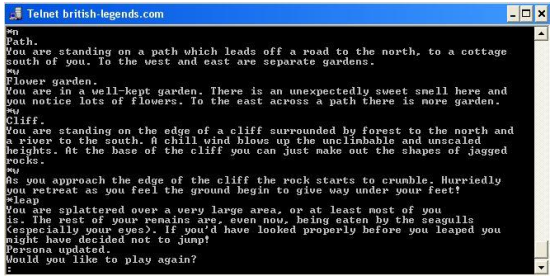
목차

- 게임 서버

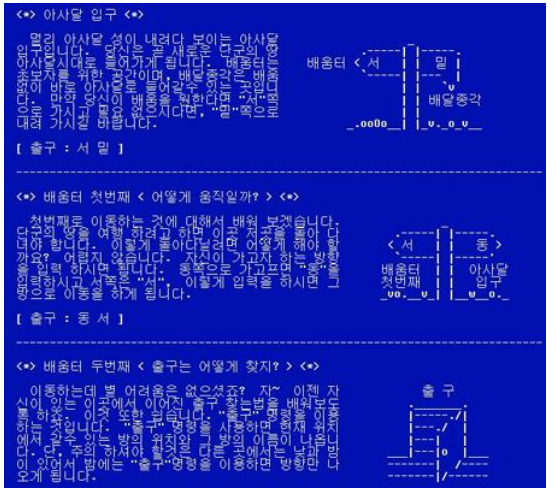


온라인 게임의 역사

- 텍스트 기반 MUD
  - 1978 머드(MUD; Multi-User Dungeon)
    - 1978년 영국 Essex 대학의 학생
  - 국내: '단군의 땅'(1994),...
- 그래픽 기반 MUD
  - 1985 Habitat
  - 국내: '바람의 나라'(1996),...



MUD



단군의 땅

Habitat

바람의 나라

온라인 게임의 역사

- MMO 게임으로 발전 – 끊임없는 세계
  - 1992: 'Neverwinter Nights' (1991)
  - MMORPG : 'Ultima Online'(1997),
  - 국내: '바람의 나라'(1996), 리니지(1998)
- 현재
  - Aion(2008-), World of Warcraft(2004-),
  - Star Wars: The Old Republic(2011),

Neverwinter Nights


Ultima Online


리니지

World of Warcraft


Star Wars: The Old Republic(2011)

TCP/IP 프로토콜

- TCP/IP 프로토콜
  - 가장 널리 사용되는 데이터 통신 규약
    - 두 컴퓨터간의 안정된 데이터 전송을 지원
  - 포트번호
    - 전체: 0~65,535
      - 공인 기관 관할: 1~1,023
      - 운영체제 관할: 1,024~5,000
      - 운영체제가 임시로 사용 가능: 49,152~
    - 응용프로그램 사용 가능: 5,001~49,151
  - TCP/IP의 사용을 위한 API
    - TCP/IP 프로토콜을 지원하는 API: Socket, Winsock, MacTCP 등
    - Winsock : 윈도우 환경에서의 소켓 API

소켓

- 소켓(socket)
  - 통신 종단의 추상적인 표현
  - 파일 입출력과 같은 공통된 입출력 방법으로 작동함

- 소켓의 역사
  - 1983년 BSD Unix 4.2에 최초로 추가됨
    - Berkeley 소켓 또는 BSD 소켓일고 하며 소켓의 표준으로 간주함
  - 윈속: 1.0(1992), 1.1(1993), 2.0(1994), 2.1(1996), 2.2(1996), 2.2.2(1997)

- 윈속(Winsock) API
  - 소켓 API는 윈도우 SDK에 포함되어 배포됨
  - Berkeley 소켓 표준 함수들 + 윈도우 추가함수들(WSA로 시작)
  - 현재 버전 2.2.x

소켓 사용 준비

- 소켓 프로그래밍 준비
  - 헤더 파일: winsock2.h
  - 링크 파일: ws2_32.lib
  - DLL 파일: ws2_32.dll

- 초기화와 종료
  - 윈속 DLL의 사용을 위한 초기화 : WSAStartup()

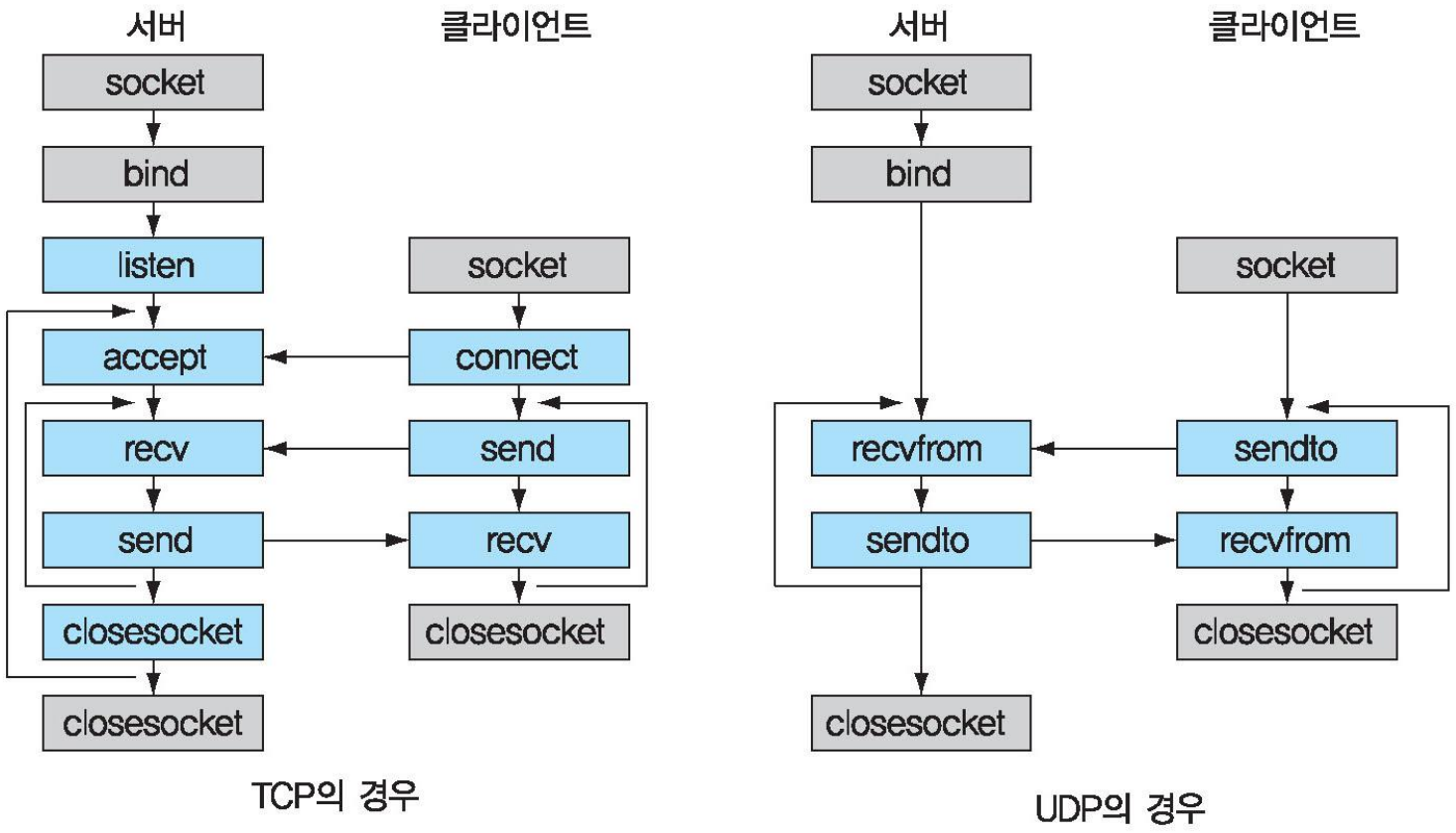
```
WSADATA wsaData;
WSAStartup(MAKEWORD(2,2), &wsaData);
```

  - 윈속 DLL의 사용 종료 : WSACleanup()

```
WSACleanup();
```

소켓 사용 준비

- 소켓 응용프로그램의 수행 절차



[그림 16-1] 소켓 함수 호출 절차.

서버에서의 소켓 생성, 바인딩

- 소켓 생성: socket()
  - SOCKET socket(int af, int type, int protocol);
    - af : AF_INET (IPv4) 또는 AF_INET6 (IPv6)
    - type : SOCK_STREAM (TCP) 또는 SOCK_DGRAM (UDP)
    - protocol : IPROTO_TCP 또는 IPROTO_UDP
    - 리턴값 : SOCKET(=unsigned int) : 소켓의 번호
  - 예시

```
SOCKET socketListen = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
```

- 소켓 바인딩: bind()
  - int bind(SOCKET s, const struct sockaddr* addr, int addrlen);
    - addr : 소켓에 바인딩할 IP 주소와 포트번호
  - 예시

```
sockaddr_in addr;
addr.sin_family = AF_INET;
addr.sin_port = htons(27015);
addr.sin_addr.s_addr = inet_addr("127.0.0.1");
bind(socketListen, (struct sockaddr*) &addr, sizeof(serveraddr));
```

htons(): u_short를 TCP/IP network byte order(big-endian)로 바꿈
inet_addr(): IPv4 문자열을 IN_ADDR 구조체에 맞도록 바꿈

서버에서의 접속 기다리기, 종료하기

- 접속을 기다리기: listen()
  - int listen(SOCKET s, int backlog);
    - backlog : 대기하는 연결들의 큐의 최대 길이 (SOMAXCONN)
  - 예시

    listen(socketListen, SOMAXCONN);
- 소켓 종료하기: shutdown(), closesocket()
  - int shutdown(SOCKET s, int how);
    - how: SD_SEND, SD_BOTH
    - SD_SEND: 상대방에게 FIN 제어비트를 보냄.
      - 상대방은 이 제어비트를 받고 소켓의 반납작업을 진행할 수 있음.
  - int closesocket(SOCKET s);
    - 소켓을 시스템에 반납하기
  - 예시

    shutdown(socketListen, SD_SEND)
    closesocket(socketListen);

서버에서의 연결 수락하기, 데이터 주고받기

- 연결을 수락하기: accept()
  - SOCKET accept(SOCKET sListen, struct sockaddr* addr, int* addrlen);
    - addr : 연결을 요청한 상대방 소켓의 주소를 리턴
    - 리턴 : 새로 생성한 소켓 번호
  - 예시

```
SOCKET socketClient = accept(socketListen, NULL, NULL);
```

- 데이터 주고받기: recv(), send()
  - int recv(SOCKET sock, char* buf, int len, int flags);
  - int send(SOCKET sock, const char* buf, int len, int flags);
    - buf : 수신된 또는 송신할 데이터를 가지는 버퍼
  - 예시

```
char recvbuf[512];
int numBytesReceived = recv(socketClient, recvbuf, 512, 0);
if (numBytesReceived > 0) {
    send( socketClient, recvbuf, numBytesReceived, 0 );
} else if (numBytesReceived == 0) { /* 연결종료함 */ }
} else { /* 수신에러 */ }
```

에코 - 서버쪽 코드

01.EchoTCP

```
WSADATA wsaData;
SOCKET socketListen, socketClient;
struct sockaddr_in serverAddr;

::WSAStartup( 0x202, &wsaData );
socketClient = INVALID_SOCKET;

socketListen = ::socket( AF_INET, SOCK_STREAM, IPPROTO_TCP );
if( socketListen == INVALID_SOCKET ) {
  printf( "Socket create error !!₩n" );
  return;
}

::memset( &serverAddr, 0, sizeof( serverAddr ) );
serverAddr.sin_family            = AF_INET;
serverAddr.sin_addr.s_addr       = ::htonl( INADDR_ANY );
serverAddr.sin_port              = ::htons( 8600 );
if ( ::bind( socketListen, ( struct sockaddr* )&serverAddr,
                                 sizeof( serverAddr ) ) == SOCKET_ERROR ){
  return false;
}
if( ::listen( socketListen, SOMAXCONN ) == SOCKET_ERROR ){
  return false;
}
//--next page--
::WSACleanup();
```

에코 - 서버쪽 코드'

```
while ( 1 ) {
  if( socketClient == INVALID_SOCKET )        {
    fd_set fds;
    struct timeval tv = { 0, 100 }; // 0.1 초
    FD_ZERO( &fds );
    FD_SET( socketListen, &fds );
    ::select( 0, &fds, 0, 0, &tv );
    if ( FD_ISSET( socketListen, &fds ) )          {
      struct sockaddr_in fromAddr;
      int size = sizeof( fromAddr );
      socketClient = ::accept( socketListen,
                    ( struct sockaddr* )&fromAddr, &size );
      printf("Accepted a client : %s₩n",
                              ::inet_ntoa(fromAddr.sin_addr));
    }
  } else{
    char recvBuffer[127];
    int recvBytes;
    recvBytes = ::recv( socketClient, recvBuffer, 127, 0 );
    printf( "%d bytes received : %s₩n", recvBytes, recvBuffer );
    ::send( socketClient, recvBuffer, recvBytes, 0 );
    ::shutdown( socketClient, SD_BOTH );
    ::closesocket( socketClient );
    socketClient = INVALID_SOCKET;
  }
}
```

클라이언트 프로그램

- 소켓 생성 : socket() – 이전과 동일
- 서버 연결 : connect()
  - int connect(SOCKET s, const struct sockaddr* name, int namelen);
  - 소켓 생성 후에 바로 connect() 함수를 호출
    - addr : 연결하고자 하는 상대방 소켓의 주소
  - 예시

```
sockaddr_in addr;
addr.sin_family = AF_INET;
addr.sin_port = htons(27015);
addr.sin_addr.s_addr = inet_addr("127.0.0.1");
connect( socketConnect, (struct sockaddr*)&addr, sizeof(addr) );
```

- 연결된 후에는 send(),recv() 사용 – 이전과 동일
- 소켓 반납은 shutdown(), closesocket() 사용 – 이전과 동일

에코 - 클라이언트쪽 코드

```
WSADATA wsaData;
SOCKET socketConnect;
struct sockaddr_in serverAddr;

::WSAStartup( 0x202, &wsaData );

socketConnect = ::socket( AF_INET, SOCK_STREAM, IPPROTO_TCP );
if ( socketConnect == INVALID_SOCKET ) return false;

::memset( &serverAddr, 0, sizeof( serverAddr ) );
serverAddr.sin_family                   = AF_INET;
serverAddr.sin_addr.s_addr              = ::inet_addr( "127.0.0.1" );
serverAddr.sin_port                     = ::htons( 8600 );
if ( ::connect( socketConnect, ( struct sockaddr* )&serverAddr,
                sizeof( serverAddr ) ) == SOCKET_ERROR )
  return false;

char sendBuffer[127] = "Test client message...",  recvBuffer[127];
int sentBytes, recvBytes;
sentBytes = ::send( socketConnect, sendBuffer,
                ::strlen( sendBuffer ) + 1, 0 );
printf( "%d bytes sent.₩n", sentBytes );

recvBytes = ::recv( socketConnect, recvBuffer, 127, 0 );
printf( "%d bytes Received₩n%s₩n", recvBytes, recvBuffer );

::shutdown( socketConnect, SD_BOTH );
::closesocket( socketConnect );
::WSACleanup();
```

UDP를 사용하는 서버 및 클라이언트 프로그램

- UDP에서는 소켓을 생성하고 바로 데이터를 송수신함
  - listen(),accept(),connect() 함수를 사용하지 않음
- 소켓의 생성과 바인딩 : socket(), bind()
  - type : SOCK_DGRAM
  - protocol : IPROTO_UDP
- 데이터의 송수신 : sendto(), recvfrom()
  - sendto()에서 수신자를 지정해야 함
  - recvfrom()은 송신자와 무관하게 수신됨.
    - 수신 후에 송신자를 알 수 있음

에코 - 서버쪽 코드

02.EchoUDP

```
WSADATA wsaData;
SOCKET listenSocket;
struct sockaddr_in echoServerAddr, echoClientAddr;
char echoBuffer[ECHOMAX];
int receiveSize, clientAddrLen;

::WSAStartup( 0x202, &wsaData );
listenSocket = ::socket( AF_INET, SOCK_DGRAM, IPPROTO_UDP );
if( listenSocket == INVALID_SOCKET ) return;
::memset( &echoServerAddr, 0, sizeof( echoServerAddr ) );
echoServerAddr.sin_family                = AF_INET;
echoServerAddr.sin_addr.s_addr    = ::htonl( INADDR_ANY );
echoServerAddr.sin_port                  = ::htons( 8599 );
::bind( listenSocket, ( struct sockaddr* )&echoServerAddr,
            sizeof( echoServerAddr ) ) ;
while( 1 )   {
  clientAddrLen = sizeof( echoClientAddr );
  receiveSize = ::recvfrom( listenSocket, echoBuffer, ECHOMAX, 0,
            ( struct sockaddr* )&echoClientAddr, &clientAddrLen );
  if( receiveSize < 0 ) continue;
  printf( "Handling client - %s₩n%d Bytes : %s",
            ::inet_ntoa( echoClientAddr.sin_addr ),
                        receiveSize, echoBuffer );
  if( ::sendto( listenSocket, echoBuffer, receiveSize, 0,
            ( struct sockaddr* )&echoClientAddr,
            sizeof( echoClientAddr ) ) != receiveSize )
  break;
}
::shutdown( listenSocket, SD_BOTH );
::closesocket( listenSocket );
::WSACleanup();
```

에코 - 클라이언트쪽 코드

```
::WSAStartup( 0x202, &wsaData );

socketValue = ::socket( AF_INET, SOCK_DGRAM, IPPROTO_UDP );
if( socketValue == INVALID_SOCKET ) return false;
::memset( &echoServerAddr, 0, sizeof( echoServerAddr ) );
echoServerAddr.sin_family                = AF_INET;
echoServerAddr.sin_addr.s_addr    = ::inet_addr( "127.0.0.1" );
echoServerAddr.sin_port                  = ::htons( 8599 );
echoStringLen = ::strlen( echoString ) + 1;
if( ::sendto( socketValue, echoString, echoStringLen, 0,
                ( struct sockaddr* )&echoServerAddr,
                sizeof( echoServerAddr ) ) != echoStringLen )
  return;

fromSize = sizeof( fromAddr );
if( ( recvStringLen = ::recvfrom( socketValue, echoBuffer,
                ECHOMAX, 0, ( struct sockaddr* )&fromAddr,
                &fromSize ) ) != echoStringLen )
  return;

if( echoServerAddr.sin_addr.s_addr != fromAddr.sin_addr.s_addr ){
  printf( "Received a packet from unknown source." );
  return;
}
echoBuffer[recvStringLen] = '₩0';
printf( "Received %d bytes : %s", recvStringLen, echoBuffer );
::shutdown( socketValue, SD_BOTH );
::closesocket( socketValue );
::WSACleanup();
```

Connected UDP

- A UDP socket can be used in a call to connect()
- This simply tells the OS the address of the peer.
  - No handshake is made to establish that the peer exists.
  - No data of any kind is sent on the network as a result of calling connect() on a UDP socket.
- Once a UDP socket is connected:
  - can use sendto() with a null destination address
  - can use write() and send()
  - can use read() and recv()
    - only datagrams from the peer will be returned.

다중스레딩

- 다중스레딩
  - 스레드 생성 함수 : CreateThread()
    - 스레드 생성 시에 실행할 함수를 지정함
  - 스레드를 생성하는 함수
    - 메인 스레드의 빠른 종료를 금지시켜야 함 : WaitForMultipleObjects()
- 참고
  - 대기 함수: WaitForSingleObject(), WaitForMultipleObjects()
    - Thread 뿐만 아니라 Process, Event, Mutex, Semaphore 등에서도 사용됨

스레드 예제

03.MultiThreading

```
DWORD __stdcall ThreadRunner( LPVOID parameter )
{
  int* argument;
  int count=3;

  argument = ( int* )parameter;
  while ( count-- > 0 )
    printf( "I'm %d Thread !!₩n", *argument );
  return 0;
}

void main()
{
  HANDLE handleThread[5];
  int array[5] = { 0, },  i;

  for ( i = 0 ; i < 5 ; i++ ) {
    array[i] = i;
    handleThread[i] = ::CreateThread( 0, 0, ThreadRunner, &array[i], 0, 0 );
  }
  for ( i = 0 ; i < 5 ; i++ )
    printf( "I'm Main thread !!₩n" );
    ::WaitForMultipleObjects( 5, handleThread, TRUE, INFINITE );
}
```

스레드 관리

- 스레드 클래스와 스레드 관리 클래스
  - Thread : 스레드의 쉬운 구현을 위한 스레드 기반 클래스
    - run() : 스레드가 담당해야 할 응용에 따른 작업들을 수행
    - begin() : ThreadManager::spawn()을 호출함
  - ThreadManager : Thread 객체들을 관리하는 클래스
    - Thread 객체의 생성, 소멸을 담당함
    - spawn() : 스레드를 생성하고 Thread::run()을 호출함
    - join() : 모든 스레드가 종료되기를 기다림

스레드 동기화

- 스레드 동기화 : 임계구역에 대한 동기화 처리
  - CriticalSection
  - Event
  - Mutex
  - Semaphore
- Problems in synchronization
  - Dead lock
  - Bottleneck

비동기식 입출력

- 입출력 방식
  - 동기식: 입출력 함수를 호출하면 작업이 완료될 때까지 리턴하지 않음
  - 비동기식: 입출력 함수의 호출 시에 바로 리턴
    - 다른 일들을 할 수 있음
    - 입출력과 관련된 일을 별도의 전담 스레드가 맡는 방식의 구현이 가능
- 비동기식 입출력
  - 윈도우 : 비동기식 입출력 기능
    - 중첩된 입출력(overlapped I/O)으로 제공함
    - Driver가 알아서 처리하고, 사용자에게 IO가 종료되었음을 알려줌
  - 중첩된 입출력을 지원하는 함수들
    - 파일 입출력을 위한 ReadFile(), WriteFile() 함수들
    - 소켓 입출력을 위한 WSASend(), WSARecv() 함수들

비동기식 입출력의 구현

- 비동기식 입출력의 구현
  - 완료 이벤트를 어떻게 확인하는가에 따라서 달라짐
  - 가장 단순한 방법: 입출력이 완료될 때까지 상태를 반복해서 확인
    - 확인을 위한 함수: WSAGetOverlappedResult()
  - 더 나은 방법: 특정 이벤트가 발생될 때까지 대기
    - 이벤트를 대기하는 함수: WaitForSingleObject(), WaitForMultipleObjects()
  - 가장 세련된 방법: 입출력 완료포트
    - IOCP; Input/Output Completion Port
    - 입출력 완료포트 = 하나의 이벤트 통지 큐
      - 여러 입출력 핸들들에서 발생되는 완료 이벤트들을 큐로 관리함
    - 포트의 생성: CreateIoCompletionPort()
    - 이벤트 큐의 확인: GetQueuedCompletionStatus()

입출력 완료포트

- 완료포트 핸들의 생성
  - CreateIoCompletionPort() – 인자값을 NULL로
- 입출력 핸들 만들기, 제거
  - CreateFile(), CloseHandle()
  - 소켓의 경우 CreateFile() 대신 socket(),accept()
- 입출력 핸들 등록하기
  - CreateIoCompletionPort() – 인자값에 완료포트 핸들을 지정함
- 데이터 읽기와 쓰기
  - ReadFile(),WriteFile()
  - 소켓의 경우 recv(),send() 대신 ReadFile(),WriteFile()
- 완료 정보 얻기
  - GetQueuedCompletionStatus()

게임 패킷

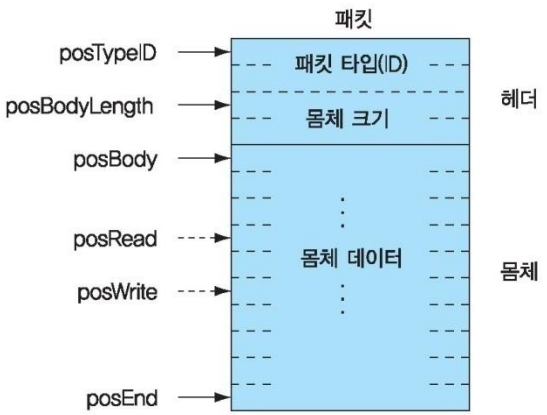
- Client-Server Functions
  - Initialization - Client, Sever
  - Rendering - Client
  - User Interaction - Client
  - Networking - Client, Server
  - A.I. Calculation for NPC - Client, Server
    - Physics, Math., Script interpreter, etc
  - Scene Graph Update - Client, Server
    - Position, Score, Status, Items, etc.
- Dependent on Game Design
- Bandwidth-Efficiency-Implementation Tradeoff
- Server cluster
  - Patch(update), Authentication(login), Lobby, Game, DB, Synch …

패킷 디자인

- Packet Protocol Design
  - Length+Header+Data
    - Unable to handle damaged Multiple packets
    - Malicious usage makes "Segment Fault" – Core dump
  - Length+Header+Data+Ending
  - Length+Header+Data+Check+Ending
  - Multiple of 4 bytes
  - Encryption
- Principles
  - Reliability – robust
  - Efficiency – save bandwidth
  - Meaningful
  - Expandable

패킷 클래스의 정의

- 패킷 클래스 : Packet
  - 하나의 패킷을 표현
- 패킷의 정의 : 헤더, 몸체
  - 바이트버퍼 : packBuffer
- 클래스 포인터 변수들
  - 헤더 : 패킷의 타입 ID와 몸체의 길이
    - posTypeID, posBodyLength
  - 몸체의 시작, 끝, 읽기, 쓰기 위치
    - posBody, posEnd, posRead, posWrite
- 패킷 데이터 읽기 및 쓰기
  - writeData()
  - readData()



[그림 16-2] 패킷 구조.

패킷 클래스 예제

06.PacketTester

```
//Packet.h
class Packet{
private:
		UCHAR packetBuffer[PACKETBUFFERSIZE];
		// 위치포인터들.
		USHORT* posTypeID; //헤더의packet Type ID.
		USHORT* posBodyLength; //헤더의packet body의length.
		UCHAR* posBody; //몸체시작위치.
		UCHAR* posEnd; //몸체끝위치.
		UCHAR* posRead;
		UCHAR* posWrite;
public:
		Packet(WORD typeID);
		void clear();
		void setTypeID(USHORT typeID) { *posTypeID = typeID; }
		USHORT getTypeID() { return *posTypeID; }
		USHORT getBodyLength() { return *posBodyLength; }
		int getPacketLength() { return (*posBodyLength + PACKETHEADERSIZE); }
		UCHAR* getPacketBuffer() { return packetBuffer; }
		void readData(void* buffer, int size);
		void writeData(void* buffer, int size);
};
```

클라이언트 프로그램의 구현

- 클라이언트 소켓 클래스 : ClientSocket
  - 구현이 단순함
- 윈속의 초기화와 종료
  - WSAStartup(), WSACleanup()
- 이벤트 처리
  - 자신의 데이터 통신만을 처리하면 되므로 간단한 방법 사용
  - WSAEVENT 구조체를 사용한 이벤트 방식
    - WSACreateEvent(), WSACloseEvent()
    - WSAEventSelect()
      - 네트워크 이벤트 상수 심볼 : FD_XXX
        - » FD_READ, FD_WRITE, FD_ACCEPT, FD_CONNECT, FD_CLOSE
    - WSAEnumNetworkEvents()
      - WSANETWORKEVENTS
- 소켓 통신
  - socket(), connect(), shutdown(), closesocket()

클라이언트 프로그램의 구현'

- 비블로킹 모드로 지정하기
  - WSAAsyncSelect()
- 소켓 입출력
  - FD_READ 이벤트가 발생하면 ReadFile() 호출
    - 패킷 데이터를 읽어오도록 함
  - 송신이 필요하면 WriteFile() 호출
    - 송신이 완료되면 FD_WRITE 이벤트가 발생함
- 전체적인 흐름
  - 메인 함수 : ClientSocket::selectEvent()를 반복해서 호출함
  - selectEvent() : 수신 이벤트 완료시에 onReceive()를 호출함
  - onReceive() : 메인 함수의 parsePacket() 함수를 호출함
  - parsePacket() : 패킷의 ID에 따라서 해당하는 패킷 처리 함수를 호출함
  - 패킷 처리 함수 : 응용에 따라 패킷 데이터를 처리하고,
    - ClientSocket::sendPacket()를 호출하여 새로운 패킷을 서버로 보냄

서버 프로그램의 구현

- 클래스들
  - 서버 메인프로그램
  - Accepter : 클라이언트와의 연결 수락을 담당
  - ServerSocket : 서버 소켓의 기능을 담당
  - SessionManager : 서버의 모든 세션들을 관리하는 세션 관리자
  - Session : 한 클라이언트와의 연결에 대한 접속 제어 및 패킷 송수신
    - 응용에 따라 구현이 달라질 수 있음
  - CompletionHandler : 완료포트를 사용한 비동기 통신
  - WorkerThread : 소켓의 이벤트들을 처리하는 일꾼 스레드

- 서버 메인 프로그램
  - Accepter::initialize()
  - CompletionHandler::initialize()
  - ThreadManager::join()

서버 프로그램의 구현'

- Accepter
  - 클라이언트와의 연결 수락을 담당
  - 독립된 단일 스레드로 동작
  - initialize()
    - ServerSocket::initialize()
    - 자신의 스레드를 생성하고 실행을 시작함. 다음을 반복함
      - ServerSocket::acceptConnection()
      - 접속되면, SessionManager::createSession()
- SessionManager
  - createSession()
    - Session::onCreate()
- CompletionHandler
  - 정해진 개수의 WorkerThread를 생성함

서버 프로그램의 구현”

- WorkerThread
  - 통신에 있어서의 이벤트들을 처리함
  - 반복적으로 완료 큐를 감시하고, 완료 이벤트를 처리함
    - 연결 종료 : SessionManager::removeSession()
    - 송수신 : Session::dispatch()
      - 송수신 완료 사실을 통지함
- Session
  - onCreate()
    - 세션이 생성된 후에 가장 먼저 해야 할 일을 수행
    - 연결된 클라이언트에게 연결 접속의 성공을 알림
  - dispatch()
    - 수신된 패킷을 인자로 parsePacket()을 호출
  - parsePacket()
    - 수신된 패킷을 보고 이에 따른 답장 패킷을 보냄

예제

07.NetworkModule