IOCP

MM4220 게임서버 프로그래밍

정내훈

IOCP

- Windows I/O모델 중 최고의 성능
  - 최근 RIO 출시
- 별도의 커널 객체를 통한 구현
  - IOCP객체를 생성해서 핸들을 받아 사용.
- 기본적으로 Overlapped I/O CallBack
  - CallBack 함수들을 멀티쓰레드로 동시에 실행
- IOCP객체 내부 Thread Pool사용
  - Thread생성 파괴 오버헤드 없앰
  - 적은 수의 thread로 많은 연결을 관리
- IOCP객체 내부 Device LIst 사용
  - 등록된 소켓에 대한 I/O는 IOCP가 처리

어려운 이유

- Overlapped I/O로만 동작
  - Overlapped I/O를 모르면 이해할 수 없음.
- 비 직관적인 API
  - 하나의 API를 여러 용도로 사용
    - 파라미터에 따라 완전히 다르게 동작하는 API
    - API이름과 아무 관계도 없는 동작을 하는 경우가 있음.
  - 뜬금없는 API 파라미터
    - 하나의 API를 여러 용도로 사용하기 때문
    - 파라미터로 넘어 오는 정보들이 불완전함 -> 편법으로 보완 필요

Windows I/O 모델

- IOCP – 준비

```
HANDLE CreateIoCompletionPort(
        HANDLE FileHandle,
        HANDLE ExistingCompletionPort,
        ULONG_PTR CompletionKey,
        DWORD NumberOfConcurrentThreads );
```

– IOCP커널 객체 생성

```
HANDLE hIOCP = CreateIoCompletionPort(INVALID_HANDLE_VALUE, NULL, NULL, 0};
```

- 마지막 0 : core 개수 만큼 사용

Windows I/O 모델

- IOCP – 준비
  - IOCP객체와 소켓연결

```
HANDLE CreateIoCompletionPort(socket, hIOCP, key, 0};
```

    - key값은 unique하게 임의로 설정
    - 마지막 값은 무시
  - Worker Thread생성

```
thread { worker_thread };
```

    - 그러나, IOCP는 쓰레드 없이도 동작 가능
    - 다음 챕터에서 쓰레드에 대해 간단히 공부한 후 멀티쓰레드를 사용할 예정

Windows I/O 모델 (2024)

- IOCP – 완료 검사
  - 커널에서 완료 상태 꺼낸다

```
while(true) {
        GetQueuedCompletionStatus(
                hIOCP,
                &dwIOSize,
                &key,
                &lpOverlapped,
                INFINITE
        );
        lpOverlapped에 따른 처리;
}
```

Windows I/O 모델

- IOCP – 완료 검사

```
BOOL GetQueuedCompletionStatus(
        HANDLE CompletionPort,
        LPDWORD lpNumberOfByte,
        PULONG_PTR lpCompletionKey,
        LPOVERLAPPED *lpOverlapped,
        DWORD dwMillisocnds);
```

  - I/O완료 상태를 report
  - Completion Port : 커널 옵젝트
  - lpNumberofByte : 전송된 데이터 양
  - lpCompletionkey : 미리 정해놓은 ID
  - lpOverlapped : Overlapped I/O 구조체

Windows I/O 모델

- IOCP 그리고
  - 이벤트 추가함수

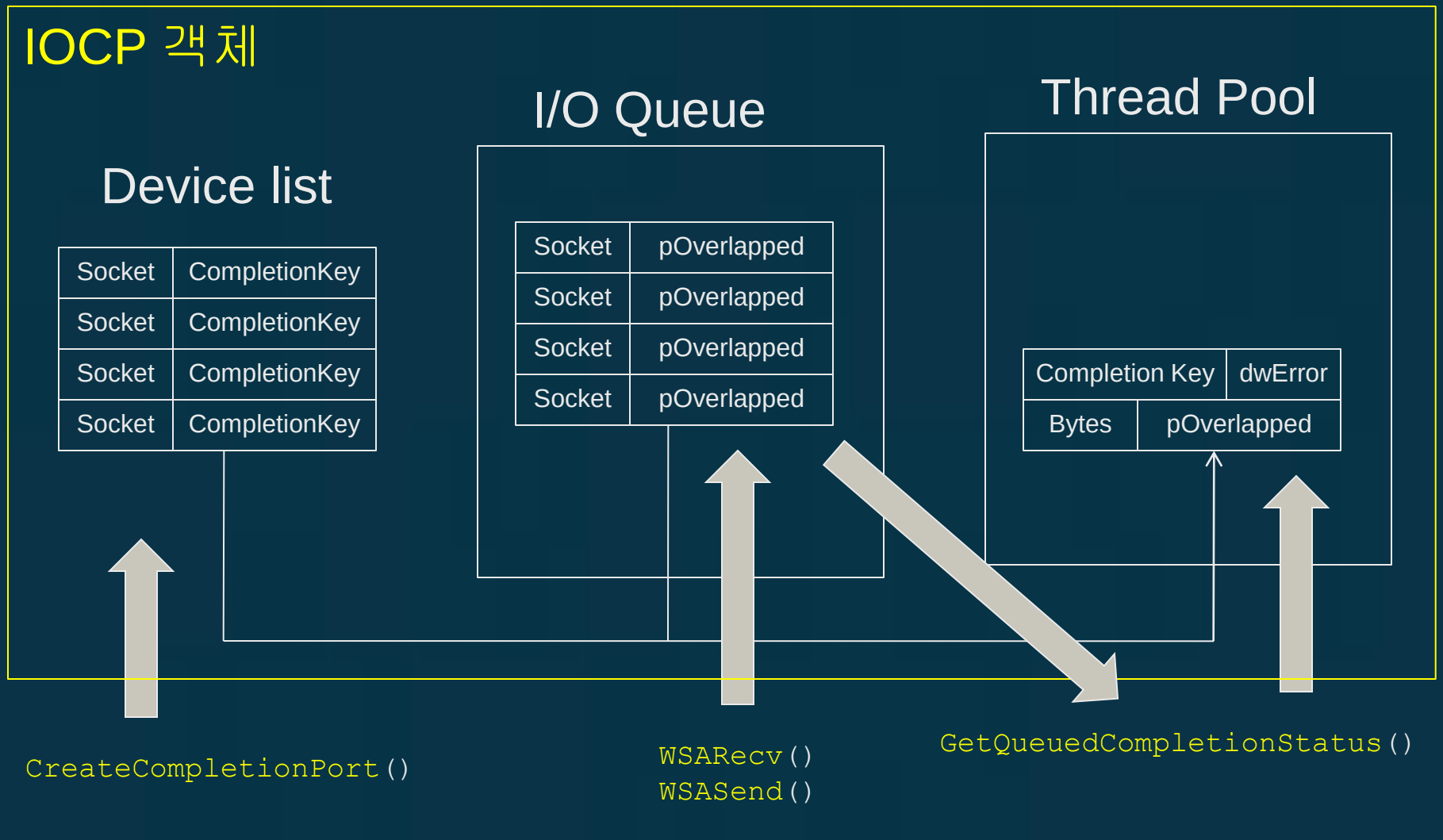
```
BOOL PostQueuedCompletionStatus(
        HANDLE CompletionPort,
        DWORD NumberOfByte,
        ULONG_PTR dwCompletionKey,
        LPOVERLAPPED lpOverlapped);
```

  - 커널의 Queue에 이벤트 추가
  - Completion Port : 커널 옵젝트
  - NumberofByte : 전송된 데이터 양
  - dwCompletionkey : 미리 정해놓은 ID
  - lpOverlapped : Overlapped I/O 구조체

Windows I/O 모델

- 그리고...
  - `PostQueuedCompletionStatus()`의 용도
    - IOCP를 사용할 경우 IOCP가 main loop가 되기 때문에 socket I/O 이외에도 모든 다른 작업할 내용을 추가 할 때 쓰인다.
      - 예) timer

Windows I/O 모델



Windows I/O 모델

I/O Model	장점	단점
select	호환성	불편하고, 느리다
WSAAsyncSelect	윈도우 메시지 기반이라 친숙하다	성능이 떨어지고 윈도우가 필수
WSAEventSelect	비교적 사용하기 쉽고 성능이 좋다	64개 소켓 제한
Overlapped I/O (Event)	IO 동시작업으로 성능 향상	64개 이벤트 제한, 개별 작업 결과 확인 필요
Overlapped I/O (Callback)	64개 이벤트제한 없음. 프로그램 간단	대규모 연결에는 아직 부족
IOCP	대규모 연결 처리 가능	사용하기 어렵다

IOCP

- IOCP API

```
CreateIoCompletionPort();
```

  - IOCP객체 생성
  - Socket을 IOCP에 연결

```
GetQueuedCompletionStatus();
```

  - IOCP로 부터 I/O결과를 얻어옴
  - 결과가 없으면 Thread를 IOCP의 thread-pool에 등록하고 멈춤

IOCP 서버 설계

- 1 : 1 통신
  - 초기화
    - IOCP 핸들 생성
  - Bind/listen/accept호출
  - 클라이언트 소켓을 IOCP에 등록 후 Recv 호출
  - 서버 메인 루프
    - GQCS호출
      - WSARecv가 완료 되면 패킷 처리, 다시 WSARecv호출
      - WSASend가 완료 되면 메모리 반환

IOCP 서버 설계

- 다중 접속

Accept를 어디에??

- 1 : 1 통신
  - 초기화
    - IOCP 핸들 생성
  - Bind/listen/accept호출
  - 클라이언트 소켓을 IOCP에 등록 후 Recv 호출
  - 서버 메인 루프
    - GQCS호출
      - WSARecv가 완료 되면 패킷 처리, 다시 WSARecv호출
      - WSASend가 완료 되면 메모리 반환

IOCP

- Accept 처리
  - 비동기 호출 필요!!! GQCS를 통한 완료 처리 필수
  - AcceptEx함수를 사용해서 비동기로 처리
  - Listen Socket을 IOCP에 등록 후, AcceptEx호출
- Accept 완료 처리
  - 새 클라이언트가 접속했으면 **클라이언트 객체**를 만든다.
  - IOCP에 소켓을 등록한다.
  - WSARecv()를 호출한다.
  - AcceptEx()를 다시 호출한다.

IOCP 서버 설계

- 1 : n 통신
  - 초기화
    - IOCP 핸들 생성
  - Bind/listen 소켓 IOCP 등록
  - 비동기 accept 호출
  - 서버 메인 루프
    - GQCS호출
      - WSARecv가 완료 되면 패킷 처리, 다시 WSARecv호출
      - WSASend가 완료 되면 메모리 반환
      - 비동기 accept 가 완료되면 IOCP등록, 세션 생성, WSARecv 호출, 다시 비동기 accept호출.

IOCP

- **클라이언트 객체 (SESSION 객체)**
  - 서버는 클라이언트의 정보를 갖고 있는 객체가 필요
    - 최대 동접과 같은 개수가 필요
    - 필요한 정보 : ID, 네트워크 접속 정보, 상태, 게임정보(name, HP, x, y)
- GetQueuedCompletionStatus를 받았을 때 클라이언트 객체를 찾을 수 있어야 한다.
  - IOCP에서 오고 가는 것은 completion_key와 overlapped I/O pointer, number of byte 뿐
  - Completion_key를 클라이언트 객체의 포인터로 하거나 클라이언트 객체의 ID 혹은 index로 한다.

IOCP

- **Overlapped 구조체**
  - 모든 Send, Recv는 Overlapped 구조체가 필요.
  - 하나의 구조체를 동시에 여러 호출에서 사용하는 것을 불가능
  - 소켓당 Recv 호출은 무조건 한 개여야 한다.
    - Recv 호출 용 Overlapped 구조체 한 개가 있어서 계속 재사용하는 것이 바람직 (new/delete overhead 제거)
  - 소켓당 Send 호출은 동시에 여러 개가 될 수 있다.
    - Send 버퍼도 같은 개수가 필요하다.
    - 개수의 제한이 없으므로 new/delete로 사용
      - Send 할 때 new, Send가 complete되었을 때 delete
    - 성능을 위해서는 공유 Pool을 만들어서 관리할 필요가 있다.

IOCP

- Overlapped I/O pointer를 확장
  - Overlapped I/O 구조체 자체는 쓸만한 정보가 없다.
  - 따라서 정보들을 더 추가할 필요가 있다.
    - 뒤에 추가하면 IOCP는 있는지 없는지도 모르고 에러도 나지 않는다. (pointer만 왔다 갔다 하므로)
  - 꼭 필요한 정보
    - 지금 이 I/O가 send인지 recv인지????
    - I/O Buffer의 위치 (Send할 때 버퍼도 같이 생성되므로)

IOCP

- 완료 처리
  - GetQueuedCompletionStatus를 부른다.
    - 에러처리/접속종료처리를 한다.
    - Send/Recv/Accept처리를 한다.
      - 확장 Overlapped I/O 구조체를 유용하게 사용한다.
      - Recv
        - 패킷이 다 왔나 검사 후 다 왔으면 패킷 처리
        - 여러 개의 패킷이 한번에 왔을 때 처리
        - 계속 Recv호출
      - Send
        - Overlapped 구조체, 버퍼의 free(혹은 재사용)
      - Accept
        - 새 클라이언트 객체 등록

IOCP

- 버퍼관리
  - Recv
    - 하나의 소켓에 대해 Recv호출은 언제나 하나이기 때문에 하나의 버퍼를 계속 사용할 수 있다.
    - 패킷들이 중간에 잘려진 채로 도착할 수 있다.
      - 모아 두었다가 다음에 온 데이터와 붙여주어야 한다.
      - 남은 데이터를 저장해 두는 버퍼 필요, 또는 Ring Buffer를 사용할 수도 있다.
    - 패킷들이 여러 개 한꺼번에 도착할 수 있다.
      - 잘라서 처리해야 한다.

IOCP

- 버퍼관리
  - Send
    - 하나의 Socket에 여러 개의 send를 동시에 할 수 있다.
      - 다중 접속! Broadcasting
      - overlapped구조체와 WSABUF는 중복 사용 불가능!
    - Windows는 send를 실행한 순서대로 내부 버퍼에 넣어놓고 전송한다.
    - 내부 버퍼가 차서 Send가 중간에 잘렸다면??
      - 나머지를 다시 보내면 된다.
      - 다시 보내기 전 다른 Send가 끼어들었다면??
      - 해결책
        - 모아서 차례 차례 보낸다. send 데이터 버퍼 외에 패킷 저장 버퍼를 따로 둔다. (성능 저하)
        - 또는, 이런 일이 벌어진 소켓을 끊어버린다.

IOCP

- IOCP 소개
- IOCP API
- IOCP 사용법
- IOCP 서버 구현 실습

IOCP 서버 구현

- 먼저 할 일
  - 다중 접속 관리
    - 클라이언트 접속 시 마다 ID 부여
  - 패킷 포맷 및 프로토콜 정의
    - 기본 패킷 포맷
      - 길이 (Byte) + 타입 (Byte) + Data (….)
    - Client -> Server
      - Login 요청, 이동 패킷
    - Server -> Client
      - Login 수락, 위치 지정, ID 접속 알림, ID 로그아웃 알림

IOCP 서버 구현

- 프로토콜 정의 TIP
  - #pragma pack(push, 1) 사용

```
struct sc_packet_login_ok {
   char size;
   char type;
   int  id;
   short x, y;
   short hp;
   short level;
   int   exp;
};

struct sc_packet_move {
   char size;
   char type;
   int id;
   short x, y;
};
```

```
struct sc_packet_enter {
   char size;
   char type;
   int  id;
   char name[MAX_ID_LEN];
   char o_type;
   short x, y;
};

struct sc_packet_leave {
   char size;
   char type;
   int  id;
};
```

```
struct cs_packet_login {
   char  size;
   char  type;
   char  name[MAX_ID_LEN];
};

struct cs_packet_move {
   char  size;
   char  type;
   char  direction;
};
```

IOCP 서버 구현

- 먼저 할 일
  - 패킷 처리 루틴 작성

```
bool process_packet(unsigned char *buf, int client_id);
```

IOCP 서버 구현

- Recv의 구현
  - overapped 구조체의 확장

```
struct OVER_EX{
   WSAOVERLAPPED m_wsaOver;
   WSABUF        m_wsaBuf;
   unsigned char m_netbuf[MAX_BUF_SIZE];   // IOCP send/recv 버퍼
   enumOperation m_Operation;              // Send/Recv/Accept 구별
};
```

  - 클라이언트 정보에 추가될 내용

```
Class SESSION{
   …
   OVER_EX      m_recv_over;
   unsigned char m_prev_recv;     // 지난 번 처리 후 남은 패킷 조각의 크기
};
```

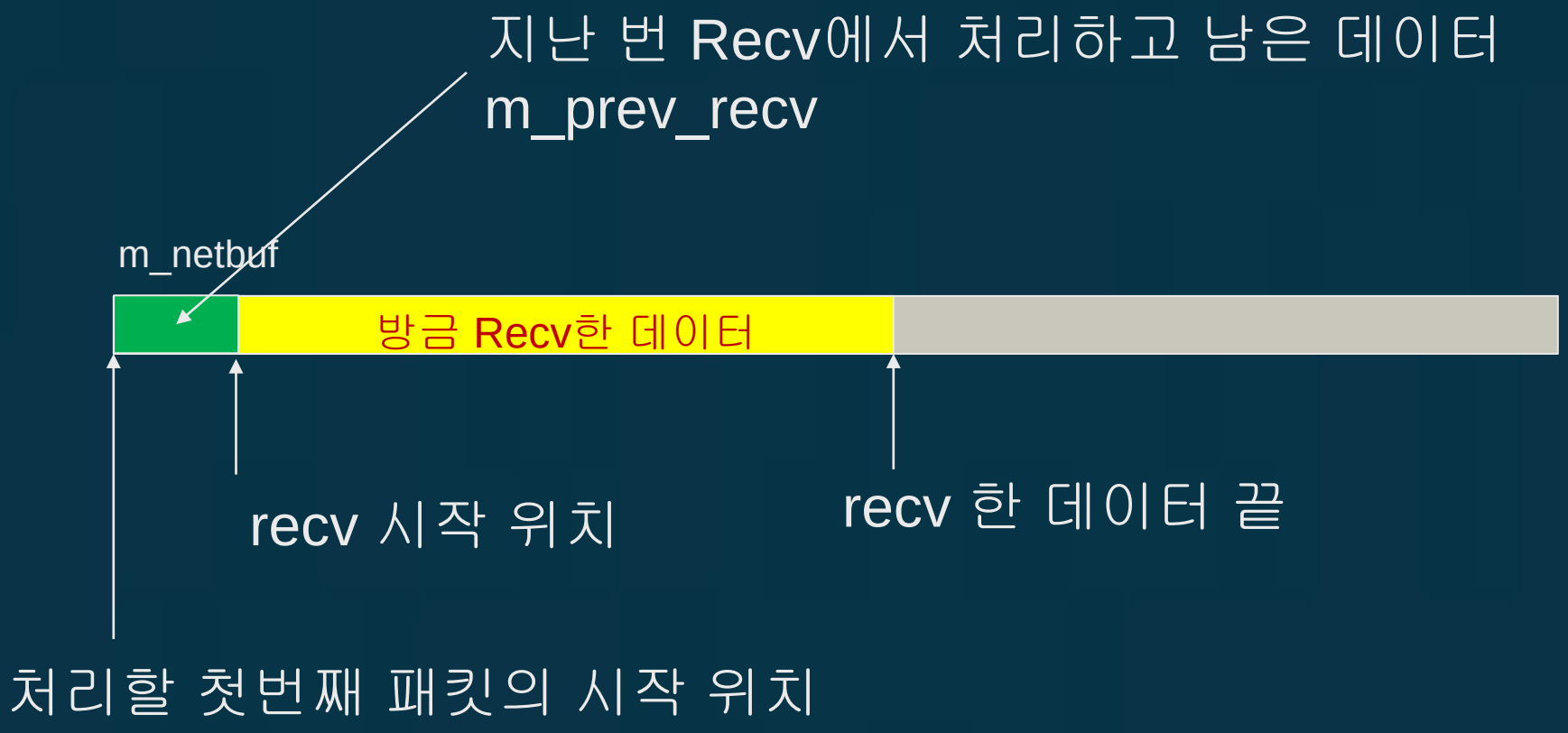
IOCP 서버 구현

- Recv의 구현 : 패킷 재조립

Start :
남은 데이터로 패킷 하나를 완성할 수 있나?
예 : 패킷 처리 함수 호출
Goto Start
종료:
Recv를 호출

IOCP 서버 구현

- Recv의 구현



IOCP 서버 구현

- Recv의 구현

```
unsinged char *packet_start = exover->m_netbuf;
int remain_data = client->m_prev_recv + num_byte;
int packet_size = packet_start[0];
while(packet_size <= remain_data) {
        process_packet(packet_start, key);
        packet_start += packet_size;
        remain_data -= packet_size;
        if (0 == remain_data) break;
        packet_size = packet_start[0];
}
client->m_prev_recv = remain_data;
if (0 != remain_data)
   memcpy(exover->m_netbuf, packet_start, remain_data);

Do_Recv (…);
```

IOCP 서버 구현

- WSASend의 사용
  - Send는 여러 클라이언트 에서 동시 다발적으로 발생
  - Send하는 overlapped 구조체와, buffer는 send가 끝날 때 까지 유지되어야 한다.
    - 개수를 미리 알 수 없으므로 Dynamic하게 관리해야 한다.
  - 부분 send인 경우
    - 버퍼가 비워지지 않은 경우
    - 에러처리하고 끊어버려야 한다.
      - 현재 운영체제의 메모리가 꽉 찬 경우
      - 이러한 일이 벌어지지 않으려면 send하는 데이터의 양을 조절해야 한다.

IOCP 서버 구현

- Send완료의 구현
  - Overapped구조체와 Buffer를 해제 시켜야 한다.
    - 메모리 재사용.
    - 모든 자료구조를 확장 Overalapped 구조체에 넣었으므로.

```
if (dwIoSize < pOverEx->m_IOCPbuf[0]) Disconnect(client);
delete pOverlappedEx;
```

IOCP 서버 구현

- 체스에서 클라이언트 객체

```
struct SESSION {
	int id;
	char name[MAX_ID_LEN];
	short x, y;

	SOCKET		m_sock;
	OVER_EX		m_recv_over;
	unsigned int m_prev_size;
};

SESSION g_clients[MAX_USER];
```

IOCP 서버 실습

- Chatting Server와의 차이
  - Callback함수 대신 GQCS
  - Accept대신 AcceptEX
  - 패킷 종류가 다양해 진다.
  - 패킷이 중간에 잘렸을 때 이어 붙이는 기능

IOCP 서버 실습

- 비동기 Accept
  - 메인루프는 GQCS로 대기해야 하므로 동기식 Accept는 처리할 수 없음
    - Accept도 실행해야 하고 GQCS도 실행해야 하고 진퇴양난에 빠짐
  - Accept또한 IOCP로 수신해야 함
    - AcceptEx 필요

AcceptEx

```
BOOL AcceptEx(
  SOCKET        sListenSocket,              // 서버 메인 소켓
  SOCKET        sAcceptSocket,              // 클라이언트 연결 소켓
  PVOID         lpOutputBuffer,             // 클라이언트 주소 저장 버퍼
  DWORD         dwReceiveDataLength,        // recv할 데이터 크기
  DWORD         dwLocalAddressLength,       // size of server address
  DWORD         dwRemoteAddressLength,      // size of client address
  LPDWORD       lpdwBytesReceived,          // size of information
  LPOVERLAPPED lpOverlapped                 // OVERLAPPED 구조체
);
```

- 서버 메인 소켓은 IOCP에 등록되어 있어야 한다.
- 클라이언트 연결 소켓은 미리 생성해 두어야 한다.
- 클라이언트 주소 저장 버퍼의 크기는 32byte 이상
- dwReceiveDataLength는 0으로, 여기서 패킷을 받지는 않을 것임
- server address와 client address의 크기는 최소 **주소크기 + 16**
- lpdwBytesReceived에는 받은 패킷 크기 + 서버 주소 크기 + 클라이언트 주소 크기가 들어가는데 여기서 받지 않을 예정이므로 NULL 포인터를 넣을 것

AcceptEx

- 완료 처리
  - 완료된 socket을 사용해 새 클라이언트 생성
    - + 초기화
  - 새로운 socket을 생성
  - AccepEx 호출
- 헤더와 라이브러리 추가 필요

```
#include <MSWSock.h>

#pragma comment(lib, "MSWSock.lib")
```

휴강

- 2023년 1학기 월화반
  - 4월 3일 예비군 훈련으로 휴강

IOCP 서버 실습(2023-화목)

- IOCP 서버를 제작해 보자
  - Visual studio 2022를 여시오.
  - 기존 다중접속 Overlapped I/O서버 수정
  - Protocol.h 작성
  - 컴파일 에러 수정
- 테스트용 클라이언트
  - 각자 자신의 클라이언트 사용
  - 또는 eclass에서 다운로드
    - [실습자료] SINGLE THREAD IOCP 실습자료
      - CLIENT_SOURCE.ZIP

IOCP 서버 실습

- Sample 클라이언트 연동
  - SFML로 그래픽 구현
    - 수 많은 시행착오 끝에 선택
      - DirectDraw -> DirectX9 -> SFML
      - 중간에 OpenGL과 SDL도 시도
      - Direct2D를 사용할 뻔.
    - NUGET에서 간단 설치
    - 제대로 사용하려면 VCPKG 필요
      - vcpkg : visual studio 패키지 관리 프로그램
        - 필요한 라이브러리를 가져와서 컴파일해서 설치
        - 디렉토리나 라이브러리 세팅을 자동으로 해줌
        - MS에서 호환성 관리

디버깅 팁

- 오동작이 의심될 때에는 에러 검사 필요

```
int ret = WSASend(u.m_s, &exover->wsabuf, 1, NULL,
                  0, &exover->over, NULL);
if (0 != ret) {
   int err_no = WSAGetLastError();
   if (WSA_IO_PENDING != err_no)
   error_display("Error in SendPacket:", err_no);
}
```

디버깅 팁

- 네트워크 관련 에러 검출

```
void error_display(const char *msg, int err_no )
{
	WCHAR *lpMsgBuf;
	FormatMessage(
		FORMAT_MESSAGE_ALLOCATE_BUFFER |
		FORMAT_MESSAGE_FROM_SYSTEM,
		NULL, err_no,
		MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
		(LPTSTR)&lpMsgBuf, 0, NULL );
	std::cout << msg;
	std::wcout << L"에러 " << lpMsgBuf << std::endl;
	while(true);
	LocalFree(lpMsgBuf);
}
```

디버깅 팁

- 한글이 나오지 않는데???
  - 다음 추가

```
std::wcout.imbue(std::locale("korean"));
```

  - VisualStudio -> 솔루션탐색기 -> 프로젝트 -> 오른클릭 -> 속성 -> 구성속성 -> 고급 -> 프로젝트 기본값 -> 문자 집합 -> 유니코드 문자 집합 사용

멀티플레이어 구현

- 요구 사항
  - 내가 접속했을 때 상대방이 보인다.
  - 내가 접속했을 때 상대방에게 내가 보인다.
  - 새로 접속하는 상대방이 보인다.
  - 나의 움직임이 상대방에게 보인다.
  - 접속을 끊은 상대방이 사라진다.
  - 내가 접속을 끊으면 상대방에게서 사라진다.

멀티플레이어

- 다중 접속 구현
  - CS_LOGIN 패킷을 받았을 때
    - 새 플레이어의 위치를 다른 모든 플레이어에게 전송
    - 다른 플레이어들의 위치를 새 플레이어에게 전송
  - do_move() 에서
    - 이동할 때 마다 새 좌표를 모든 플레이어에게 전송
  - GetQueuedCompletionStatus() 에서
    - 접속 종료 시 다른 모든 플레이어에게 알림

정리

- IOCP MMOG 서버의 뼈대 완성
- 완벽하지 않음
  - 최적화 필요
  - id 재사용 금지
  - 등등등

정리

- 이후의 내용
  - 멀티쓰레드
  - 멀티쓰레드를 사용한 IOCP
  - 부하 테스트
  - 컨텐츠 구현
    - Quest, Item, Skill, 전투... => 별다른 것 없으므로 PASS
    - 시야
      - 성능, 맵핵 방지
    - 충돌 처리
      - 해킹 방지
    - AI
      - 몇 십만 마리의 몬스터..., 스크립트 연동