IO 모델

MM4220 게임서버 프로그래밍

정내훈

다중 접속 서버

- 서버 메인 루프

```
while (true) {
    // Process Client Packet
    recv(A, buf);
    process_packet(A, buf);
    recv(B, buf);
    process_packet(B, buf);

    // Update World
    // do NPC AI, do Heal, …
    …
}
```

– 무엇이 문제인가?

네트워크 I/O 모델

- I/O 모델이 필요한 이유
  - 블럭킹 방지
  - 비 규칙적인 입출력 관리
  - 다중 접속 관리
- 게임 서버의 접속
  - 정해지지 않은 동작 순서
    - 그래도 멈추지 않아야 하는 게임
  - 수 천 개의 접속
  - 상대적으로 낮은 접속당 bandwidth
    - 효율적 자원관리 필요

Windows I/O 모델

- Socket Thread
- Non-blocking I/O
- Select
- WSAAsyncSelect
- WSAEventSelect
- Overlapped I/O (Event)
- Overlapped I/O (Callback)
- I/O Completion Port

Windows I/O 모델

- Socket Thread
  - Thread를 통한 처리

Windows I/O 모델

- Thread를 통한 처리

```
while (!shutdown) {
    new_sock = accept(sock, &addr, &len);
    thread t = thread { do_io, new_sock };
}
```

```
void do_io(mysock) {
    while (true) {
      recv(mysock);
      process_packet(); }}
```

  - 다중 소켓 처리 가능
  - 과도한 thread 개수로 인한 운영체제 Overhead
    - thread당 오버헤드 : Thread Control Block, Stack 메모리

Windows I/O 모델

- Non-blocking I/O

```
unsigned long noblock = 1;
int nRet = ioctlsocket(sock, FIONBIO, &noblock);
```

  - Socket의 모드를 blocking에서 non-blocking으로 변환
  - Socket 함수 호출을 즉시 완료할 수 없을 때.
    - WSAEWOULDBLOCK 에러를 내고 끝난다.
    - 기다리지 않는다

Windows I/O 모델

- Non-blocking I/O
  - 단점 : Busy Waiting
    - 모든 소켓의 recv를 돌아가면서 반복 호출해야 함.
    - 잦은 시스템 Call -> CPU낭비 -> 성능 저하

```
while (true) {
    for (SOCKET s : sockets) {
        recv(s,…);
        if (no_error) 패킷 처리;
}
```

Windows I/O 모델(2021-월수)

- Select

```
int select(
          __in int nfds,
          __inout fd_set* readfds,
          __inout fd_set* writefds,
          __inout fd_set* exceptfds,
          __in const struct timeval* timeout );
```

  - nfds : 검사하는 소켓의 개수, Windows에서는 무시
  - readfds : 읽기 가능 검사용 소켓 집합 포인터
  - writefds : 쓰기 가능 검사용 소켓 집합 포인터
  - exceptfds : 에러 검사용 소켓 집합 포인터
  - timeout : select가 기다리는 최대 시간
  - return value : 사용 가능한 소켓의 개수

Windows I/O 모델

- Select
  - Unix시절부터 내려온 고전적인 I/O 모델
  - unix나 linux에서는 socket 개수의 한계 존재
    - unix: 64, linux: 1024
  - socket의 개수가 많아질수록 성능 저하 증가
    - linear search

```
FD_SET(sock1, &rfds);
FD_SET(sock2, &rfds);

select(0. &rfds, NULL, NULL, &time);

if (FD_ISSET(sock1, &rfds)) recv(sock1, buf, len, 0)
if (FD_ISSET(sock2, &rfds)) recv(sock2, buf, len, 0)
```

Windows I/O 모델

- WSAAsyncSelect
  - 소켓 이벤트를 특정 윈도우의 메시지로 받는다

```
int WSAAsyncSelect(
          __in SOCKET s,
          __in HWND hWnd,
          __in unsigned int wMsg,
          __in long lEvent );
```

  - s : 소켓
  - hWnd : 메시지를 받을 윈도우
  - wMsg : 메시지 번호
  - lEvent : 반응 event 선택

Windows I/O 모델

- WSAAsyncSelect
  - 클라이언트에 많이 쓰임
    - 윈도우 필요
    - 윈도우 메시지 큐를 사용 -> 성능 느림

lEvent

비트 값	의미
FD_READ	Recv 할 데이터가 있음
FD_WRITE	Send할 수 있는 버퍼 공간 있음
FD_OOB	Out-of-band 데이터 있음
FD_ACCEPT	Accept 준비가 됨
FD_CONNECT	접속이 완료됨
FD_CLOSE	소켓연결이 종료됨

Windows I/O 모델

- WSAAsyncSelect
  - 자동으로 non-blocking mode로 소켓 전환
  - 아래와 같은 함수로 이벤트 처리

```
LRESULT CAsyncSelectDlg::OnSocketMsg(WPARAM wParam, LPARAM lParam)
{
        SOCKET sock=(SOCKET)wParam;
        int nEvent = WSAGETSELECTEVENT(lParam);
        switch(nEvent){
                case FD_READ :
                case FD_ACCEPT :
                case FD_CLOSE :
```

Windows I/O 모델

- WSAEventSelect

```
int WSAEventSelect(
        __in SOCKET s,
        __in WSAEVENT hEventObject,
        __in long lNetworkEvents );
```

  - s : 소켓
  - hEventObject :
  - lNetworkEvents : WSAAsyncSelect와 같음
  - 메시지를 처리할 윈도우가 필요 없음.

Windows I/O 모델

- WSAEventSelect
  - socket과 event의 array를 만들어서 `WSAWaitForMultipleEvents()` 의 리턴값으로 부터 socket 추출
  - 소켓의 개수 64개 제한!
    - 멀티 쓰레드를 사용해서 제한 극복가능

Windows I/O 모델

- WSAEventSelect
  - 다음의 API로 socket대기 상태 검출

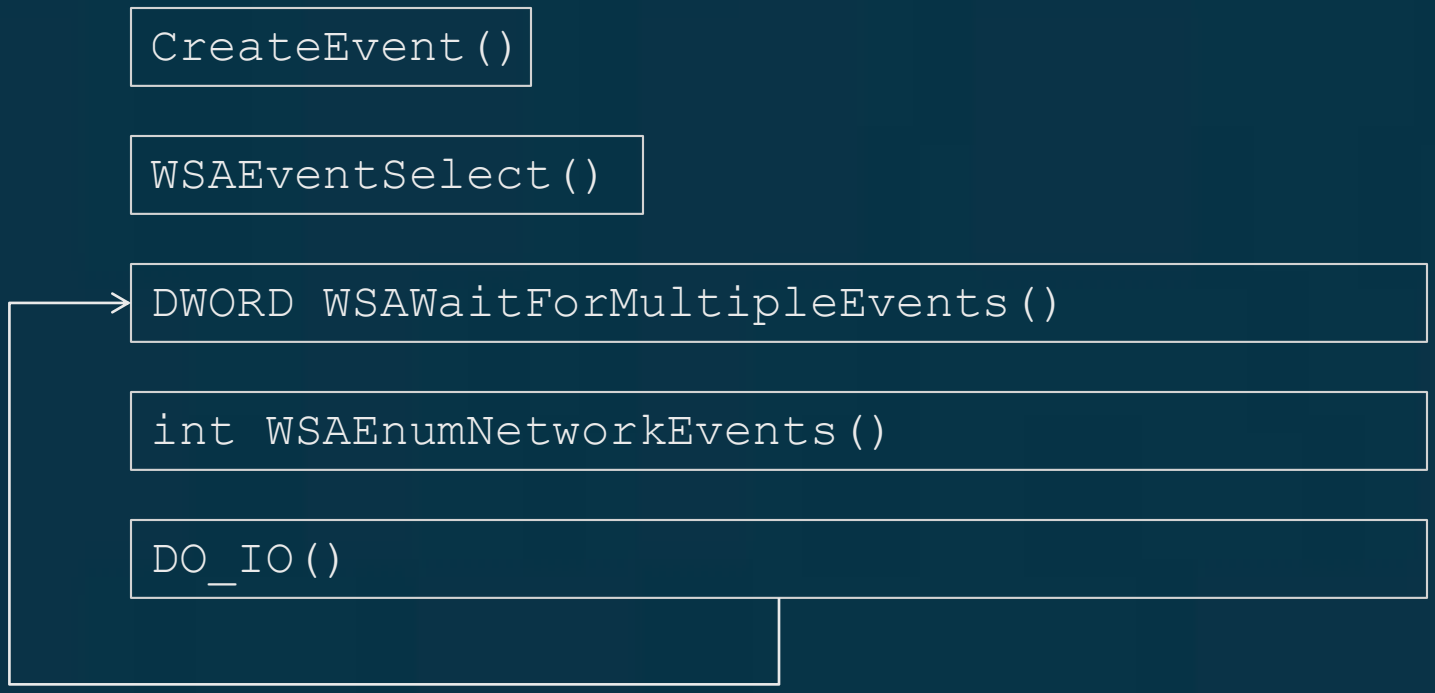
```
DWORD WSAWaitForMultipleEvents(
        DWORD nCount,
        const WSAEVENT *lphEvents,
        BOOL bWaitAll,
        DWORD dwMiliseconds);
```

```
int WSAEnumNetworkEvents(SOCKET s,
        WSAEVENT hEventObject;,
        LPWSANETWORKEVENTS lpNetworkEvents);
```

```
lpNetworkEvents->lNetworkEvents;
```

Windows I/O 모델

- WSAEventSelect : 동작



Windows I/O 모델

- WSAEventSelect : 예제
  - http://perfectchoi.blogspot.com/2009/09/wsaeventselect.html

Windows I/O 모델(2021, 수금)

- Overlapped I/O 모델
  - Windows에서 추가된 고성능 I/O 모델
  - 다른 이름으로는 Asynchronous I/O 또는 비동기 I/O
    - 리눅스의 경우 boost/asio 라이브러리로 사용 가능
  - 대용량 고성능 네트워크 서버를 위해서는 필수 기능
  - IOCP도 Overlapped I/O를 사용
  - 사용 방법이 select style의 I/O 모델과 다르다.
    - I/O요청을 먼저하고 I/O의 종료를 나중에 확인한다.
      - 요청은 즉시 처리
      - Non-Blocking과는 다르게 거의 실패하지 않는다.
    - I/O요청 후 기다리지 않고 다른 일을 할 수 있다.
    - 여러 개의 I/O요청을 동시에 할 수 있다.

Windows I/O 모델

- Overlapped I/O 모델

```
while (true) {
    select(&recv_ready_sockets);
    for (i : recv_ready_sockets) {
        recv(socket[i], buf, …);
        패킷 처리 for i;
    }
}
```

non-overlapped I/O

```
for (sock : all_sockets)
    recv(sock, buf[sock]);
while (true) {
    sock = wait_for_completed_recv();
    패킷 처리 for sock;
    recv(sock, buf[sock], …);
}
```

overlapped I/O

Windows I/O 모델

- Overapped I/O 모델
  - 소켓 내부 버퍼를 사용하지 않고 직접 사용자 버퍼에서 데이터를 보내고 받을 수 있다. (옵션)

```
int result;
int buffsize = 0;
result = setsockopt(s, SOL_SOCKET, SO_SNDBUF, &buffsize, sizeof(buffsize));
```

  - 앞의 다중 I/O 모델들은 recv와 send의 가능 여부만 검사 후 I/O수행, Overlapped는 I/O 요청 후 실행 완료 통보
- Non-Blocking IO와의 차이
  - I/O 호출과 완료의 분리
    - Non-Blocking IO는 실패와 성공이 있고 거기서 끝
    - Overlapped I/O는 동시 진행, 버퍼가 커널에 등록되어 커널에서 전송 수행
  - I/O 다중 수행

Windows I/O 모델

- Overapped I/O 모델
  - Send와 Recv를 호출했을 때 패킷 송수신의 완료를 기다리지 않고 Send, Recv함수 종료
  - 이때 Send와 Recv는 송수신의 시작을 지시 만하는 함수
    - 이미 도착한 데이터가 있으면 받을 수도 있지만, 그렇게 하지 않을 것임.
  - 여러 개의 Recv, Send를 겹쳐서 실행함으로써 여러 소켓에 대한 동시 다발적 Recv, Send도 가능
    - 하나의 socket은 하나의 recv만 가능!!!

Windows I/O 모델

- Overlapped I/O
  - SOCKET WSASocket(int af, int type, int protocol, LPWSAPROTOCOL_INFO lpProtocolInfo, GROUP g, DWORD dwFlags)
    - af : address family
      - AF_INET만 사용 (AF_NETBIOS, AF_IRDA, AF_INET6)
    - type : 소켓의 타입
      - tcp를 위해 SOCK_STREAM사용 (SOCK_DGRAM)
    - protocol : 사용할 프로토콜 종류
      - IPPROTO_TCP (IPPROTO_UDP)
    - lpProtocolInfo : 프로토콜 정보
      - 보통 NULL
    - g : 예약
    - dwFlags : WSA_FLAG_OVERLAPPED

Windows I/O 모델

- Overlapped I/O
  - int WSARecv(SOCKET s, LPWSABUF lpBuffers, DWORD dwBufferCount, LPDWORD lpNumberofBytesRecvd, LPDWORD lpFlags, LPWSAOVERLAPPED lpOverlapped, LPWSAOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine)
    - s : 소켓
    - lpBuffers : 받은 데이터를 저장할 버퍼
    - dwBufferCount : 버퍼의 개수
    - lpFlags : 동작 옵션(MSG_PEEK, MSG_OOB)
    - lpNumberofBytesRecvd : 받은 데이터의 크기 => NULL
    - lpOverlapped, lpCompletionRoutine : NULL이 아닌 경우 overlapped I/O로 동작

Windows I/O 모델

- Overlapped I/O
  - LPWSAOVERLAPPED lpOverlapped

```
typedef struct WSAOVERLAPPED {
        DWORD Internal;
        DWORD InternalHigh;
        DWORD Offset;
        DWORD OffsetHigh;
        WSAEVENT hEvent;
} WSAOVERLAPPED, FAR *LPWSAOVERLAPPED;
```

    - Internal, InternalHigh, Offset, OffsetHigh : 0으로 초기화 후 사용
    - hEvent I/O가 완료 되었음을 알려주는 event 핸들
  - LPWSAOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine
    - callback 함수, 뒤에 설명

Windows I/O 모델

- Overlapped I/O 모델
  - Overlapped I/O가 언제 종료되었는지를 프로그램이 알아야 함
  - 두 가지 방법이 존재
    - Overlapped I/O Event모델
    - Overlapped I/O Callback모델

Windows I/O 모델

- Overlapped I/O Event 모델
  - WSARecv의 LPWSAOVERLAPPED lpOverlapped 구조체의 WSAEVENT hEvent 사용
  - 작업 결과 확인
    - WSAGetOverlappedResult()

Windows I/O 모델

- Overlapped I/O Event 모델
  - WSAGetOverlappedResult()

```
BOOL WSAGetOverlappedResult(
        SOCKET s,
        LPWSAOVERLAPPED lpOverlapped,
        LPDWORD lpcbTransfer,
        BOOL fWait,
        LPDWORD lpdwFlags);
```

    - s :socket
    - lpOverlapped : WSARecv에 넣었던 구조체
    - lpcbTransfer : 전송된 데이터 크기
    - fWait : 대기 여부
    - lpdwFlags : Recv의 lpFlag의 결과

Windows I/O 모델

- Overlapped I/O Event 모델
  - 1. WSACreateEvent()를 사용해서 이벤트 생성
  - 2. WSAOVERLAPPED구조체 변수선언, 0으로 초기화 hEvent에 1의 이벤트
  - 3. WSASend(), WSARecv()
    - 2의 구조체를 WSAOVERLAPPED에
      - 중복 사용 불가능!! 호출 완료 후 재사용
    - lpCompletionROUTINE에 NULL
  - 4. WSAWaitForMultipleEvents()함수로 이벤트 감지
  - 5. WSAGetOverlappedResult()함수로 이벤트완료 확인

Windows I/O 모델

- Overlapped I/O Callback 모델
  - 이벤트 제한 개수 없음
  - 사용하기 편리
  - WSARecv와 WSASend의 LPWSAOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine 함수 사용
  - Overlapped I/O가 끝난후 lpCompletionRoutine이 호출됨

Windows I/O 모델

- Overlapped I/O Callback 모델
  - Callback함수

```
void CALLBACK CompletionROUTINE(
        DWORD dwError,
        DWORD cbTransferred,
        LPWSAOVERLAPPED lpOverlapped,
        DWORD dwFlags);
```

  - dwError : 작업의 성공 유무
  - cbTransferred : 전송된 바이트 수
  - lpOverlapped : WSASend/WSARecv에서 사용한 구조체
  - dwflags : WSASend/WSARecv에서 사용한 flag

Windows I/O 모델

- Overlapped I/O
  - Overlapped I/O 구분

lpOverlapped	hEvent	lpCompletionRoutine	Completion 여부 식별
NULL	세팅 불가	무시됨	동기적 실행
Not-NULL	NULL	NULL	Overlapped 동작, 완료 검사 불가능
Not-NULL	Not-NULL	NULL	Overlapped 동작, Event객체로 완료 검사
Not-NULL	무시됨	Not-NULL	Overlapped 동작, completion routine을 통해서 완료 관리

실습

- 예제 프로그램 :
  - http://myblog.opendocs.co.kr/archives/1204
  - 64비트 용으로 수정한 것을 E-Class 자료실에서 다운로드.
    - [3주차] I/O 모델 : ECHO_SeverClient.zip

MMOServer 다중접속

- 클라이언트 변경
  - Avatar와 PC가 존재
    - Avatar : 나
    - PC : Playing Character, 다른 사람
    - 그래픽 상 구분 필요 (카메라는 아바타 연동)
- 서버 변경
  - 여러 개의 Socket관리
  - 여러 개의 Client Object 관리
  - 객체 상태 변화 => BroadCast <= ID 필요
- 프로토콜 정의 필요
  - 종류 : Object 추가/이동/삭제
  - 구성 : Size, Type, DATA(id, 좌표, Avatar여부)

숙제 (#3)

- 게임 서버/클라이언트 프로그램 작성
  - 내용
    - 숙제 (#2)의 프로그램의 다중 사용자 버전
    - Client/Server 모델, 서버는 반드시 Overlapped I/O callback 을 사용할 것
    - 클라이언트 10개 까지 접속 가능 하게 수정
      - 모든 클라이언트 에서 다른 모든 클라이언트의 말의 움직임이 보임
  - 목적
    - Windows 다중 접속 Network I/O 습득
    - Overlapped I/O 습득
  - 제약
    - Windows에서 Visual Studio로 작성 할 것
    - 그래픽의 우수성을 보는 것이 아님
  - 제출
    - Zip으로 소스를 묶어서 eclass로 제출
      - 컴파일 및 실행이 가능해야 함, 필요 없는 중간 파일들 포함 시키지 말것