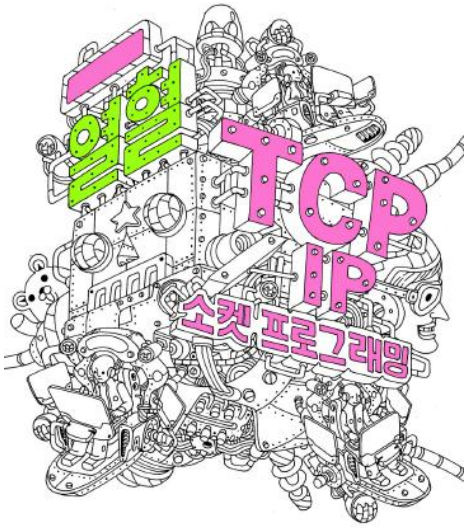
윤성우의 열혈 TCP/IP 소켓 프로그래밍

윤성우 저 열혈강의 TCP/IP 소켓 프로그래밍 개정판

Chapter 23. IOCP(Input Output Completion Port)



Chapter 23-1. Overlapped IO를 기반으로 IOCP 이해하기

윤성우 저 열혈강의 TCP/IP 소켓 프로그래밍 개정판

넌-블로킹 모드의 소켓 구성하기

넌-블로킹 모드의 소켓 구성

```
SOCKET hLisnSock,
int mode=1;
. . . . .
hLisnSock=WSASocket(PF_INET, SOCK_STREAM, 0, NULL, 0, WSA_FLAG_OVERLAPPED);
ioctlsocket(hLisnSock, FIONBIO, &mode);   // for non-blocking socket
. . . . .
```

mode에 저장된 값 1은 넌-블로킹 소켓의 옵션지정에 사용된다.

입출력 모드(FIONBIO)를 변수 mode에 저장된 값으로 바꿔라!

넌-블로킹 모드 소켓의 입출력 이외의 추가적인 특징

- 클라이언트의 연결요청이 존재하지 않는 상태에서 accept 함수가 호출되면 INVALID_SOCKET이 곧바로 반환된다. 그리고 이어서 WSAGetLastError 함수를 호출하면 WSAEWOULDBLOCK가 반환된다.
- accept 함수호출을 통해서 새로 생성되는 소켓 역시 넌-블로킹 속성을 지닌다.

Overlapped IO만 가지고 에코 서버 구현하기1

ReadCompRoutine 함수의 호출을 위해서

SleepEx 함수를 주기적으로 호출한다.

```
while(1)
{
    SleepEx(100, TRUE);    // for alertable wait state
    hRecvSock=accept(hLisnSock, (SOCKADDR*)&recvAdr,&recvAdrSz);
    if(hRecvSock==INVALID_SOCKET)
    {
        if(WSAGetLastError()==WSAEWOULDBLOCK)
            continue;
        else
            ErrorHandling("accept() error");
    }
    puts("Client connected.....");

    lpOvLp=(LPWSAOVERLAPPED)malloc(sizeof(WSAOVERLAPPED));
    memset(lpOvLp, 0, sizeof(WSAOVERLAPPED));

    hbInfo=(LPPER_IO_DATA)malloc(sizeof(PER_IO_DATA));
    hbInfo->hClntSock=(DWORD)hRecvSock;
    (hbInfo->wsaBuf).buf=hbInfo->buf;
    (hbInfo->wsaBuf).len=BUF_SIZE;

    lpOvLp->hEvent=(HANDLE)hbInfo;
    WSARecv(hRecvSock, &(hbInfo->wsaBuf),
        1, &recvBytes, &flagInfo, lpOvLp, ReadCompRoutine);
}
```

예제 CmplRouEchoServ_win.c의 일부

```
typedef struct
{
    SOCKET hClntSock;
    char buf[BUF_SIZE];
    WSABUF wsaBuf;
} PER_IO_DATA, *LPPER_IO_DATA;
```

서버의 클라이언트 연결 루틴!

데이터 수신 시 ReadCompRoutine 함수가 호출되는 구조로 구성

Overlapped IO만 가지고 에코 서버 구현하기2

```
void CALLBACK ReadCompRoutine(
    DWORD dwError, DWORD szRecvBytes, LPWSAOVERLAPPED lpOverlapped, DWORD flags)
{
    LPPER_IO_DATA hbInfo=(LPPER_IO_DATA)(lpOverlapped->hEvent);
    SOCKET hSock=hbInfo->hClntSock;
    LPWSABUF bufInfo=&(hbInfo->wsaBuf);
    DWORD sentBytes;
    if(szRecvBytes==0)
    {
        closesocket(hSock);
        free(lpOverlapped->hEvent); free(lpOverlapped);
        puts("Client disconnected.....");
    }
    else    // echo!
    {
        bufInfo->len=szRecvBytes;
        WSASend(hSock, bufInfo, 1, &sentBytes, 0, lpOverlapped, WriteCompRoutine);
    }
}
```

데이터를 전송한 소켓의 파일 디스크립터와 수신된 데이터 확인

수신된 데이터가 0이라면 연결종료의 요청

에코 서비스 제공 동시에 전송 완료 시 WriteCompRoutine 호출되도록 문장 구성

예제 CmplRouEchoServ_win.c의 일부

Overlapped IO만 가지고 에코 서버 구현하기3

예제 CmplRouEchoServ_win.c의 일부

```
void CALLBACK WriteCompRoutine(
    DWORD dwError, DWORD szSendBytes, LPWSAOVERLAPPED lpOverlapped, DWORD flags)
{
    LPPER_IO_DATA hbInfo=(LPPER_IO_DATA)(lpOverlapped->hEvent);
    SOCKET hSock=hbInfo->hClntSock;
    LPWSABUF bufInfo=&(hbInfo->wsaBuf);
    DWORD recvBytes;
    int flagInfo=0;
    WSARecv(hSock, bufInfo,
        1, &recvBytes, &flagInfo, lpOverlapped, ReadCompRoutine);
}
```

데이터 전송이 완료되고 나면, 계속된 서비스를 위해서 WSARecv 함수가 호출되도록 한다.

즉, ReadCompRoutine 함수와 WriteCompRoutine 함수가 반복 호출되면서 에코 서비스를 제공하는 형태의 서버이다!

클라이언트의 재 구현

예제 StableEchoClnt_win.c의 일부

```
while(1)
{
    fputs("Input message(Q to quit): ", stdout);
    fgets(message, BUF_SIZE, stdin);
    if(!strcmp(message,"q\n") || !strcmp(message,"Q\n"))
        break;
    strLen=strlen(message);
    send(hSocket, message, strLen, 0);
    readLen=0;
    while(1)
    {
        readLen+=recv(hSocket, &message[readLen], BUF_SIZE-1, 0);
        if(readLen>=strLen)
            break;
    }
    message[strLen]=0;
    printf("Message from server: %s", message);
}
```

전송한 데이터를 전부 수신하는 형태로 구현해야 한다! 이것이 TCP의 정상적인 구현 모델이다.

서버가 넌-블로킹 모드로 동작하기 때문에 제대로 된 TCP 클라이언트를 작성해야 한다. 그렇지 않으면 정상적으로 동작하지 않는다.

Overlapped IO 모델에서 IOCP의 모델로

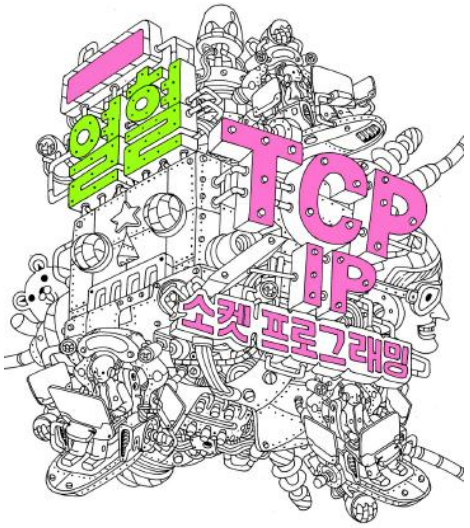
Overlapped IO 모델의 문제점

"넌-블로킹 모드의 accept 함수와 alertable wait 상태로의 진입을 위한 SleepEx 함수가 번갈아 가며 반복 호출되는 것은 성능에 영향을 미칠 수 있다!"

Overlapped IO 모델이 지니는 문제점의 해결책

"accept 함수의 호출은 main 쓰레드가(main 함수 내에서) 처리하도록 하고, 별도의 쓰레드를 추가로 하나 생성해서 클라이언트와의 입출력을 담당하게 한다."

위의 해결책을 토대로 구성이 된 서버모델이 바로 IOCP이다!



Chapter 23-2. IOCP의 단계적 구현

윤성우 저 열혈강의 TCP/IP 소켓 프로그래밍 개정판

IOCP에서의 IO의 완료 확인

IOCP의 기본 모델

IOCP에서는 완료된 IO의 정보가 'Completion Port 오브젝트(CP 오브젝트)'라는 커널 오브젝트에 등록된다. 그런데 이를 위해서는 다음의 요구를 별도로 진행해야 한다.

"이 소켓을 기반으로 진행되는 IO의 완료 상황은 저 CP 오브젝트에 등록해 주세요"

CP 오브젝트에 IO의 완료가 등록될 수 있도록
다음 두 가지가 선행되어야 한다.

IOCP의 기본 조건

- Completion Port 오브젝트의 생성
- Completion Port 오브젝트와 IO의 완료를 등록할 소켓과의 연결

위의 두 가지 작업을 위해서 정의된 함수가 CreationIOCompletionPort이다.
즉, CreationIOCompletionPort 함수는 CP 오브젝트의 생성을 목적으로, 그리고 생성된 CP 오브젝트와 소켓과의 연결을 목적으로 사용된다.

CreateIoCompletionPort 함수의 두 가지 기능

```
#include <windows.h>

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

➜ 성공 시 CP 오브젝트의 핸들, 실패 시 NULL 반환

- FileHandle — CP 오브젝트 생성시에는 INVALID_HANDLE_VALUE를 전달.
- ExistingCompletionPort — CP 오브젝트 생성시에는 NULL 전달.
- CompletionKey — CP 오브젝트 생성시에는 0 전달.
- NumberOfConcurrentThreads — CP 오브젝트에 할당되어 완료된 IO를 처리할 쓰레드의 수를 전달, 예를 들어 2가 전달되면 CP 오브젝트에 할당되어 동시 실행 가능한 쓰레드의 수는 최대 2개로 제한된다. 그리고 이 인자에 0이 전달되면 시스템의 CPU 개수가 동시 실행 가능한 쓰레드의 최대수로 지정된다.

CP 오브젝트의 생성방법, CP 오브젝트의 생성을 목적으로 함수를 호출할 경우, 마지막 매개변수만이 의미를 갖는다.

```
HANDLE hCpObject;
. . . . .
hCpObject=CreateIoCompletionPort(INVALID_HANDLE_VALUE, NULL, 0, 2);
```

CP 오브젝트에 할당되어 IO의 처리를 담당할 쓰레드의 수를 최대 2개로 제한!

```
HANDLE hCpObject;
SOCKET hSock;
. . . . .
CreateIoCompletionPort((HANDLE)hSock, hCpObject , (DWORD)ioInfo, 0);
```

다음의 CreateIoCompletionPort 함수가 호출된 이후부터는 hSock을 대상으로 진행된 IO 완료 시, 해당 정보가 hCpObject에 해당하는 CP 오브젝트에 등록된다.

Completion Port의 완료된 IO 확인

```
#include <windows.h>

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

➡ 성공 시 TRUE, 실패 시 FALSE 반환

- CompletionPort — 완료된 IO 정보가 등록되어 있는 CP 오브젝트의 핸들 전달.
- lpNumberOfBytes — 입출력 과정에서 송수신 된 데이터의 크기정보를 저장할 변수의 주소 값 전달.
- lpCompletionKey — CreateIoCompletionPort 함수의 세 번째 인자로 전달된 값의 저장을 위한 변수의 주소 값 전달.
- lpOverlapped — WSASend, WSARecv 함수호출 시 전달하는 OVERLAPPED 구조체 변수의 주소 값이 저장될, 변수의 주소 값 전달
- dwMilliseconds — 타임아웃 정보전달, 여기서 지정한 시간이 완료되면 FALSE를 반환하면서 함수를 빠져나가며, INFINITE를 전달하면 완료된 IO가 CP 오브젝트에 등록될 때까지 블로킹 상태에 있게 된다.

왼편의 함수호출을 통해서, CreateIoCompletionPort 함수호출 시 전달된 정보와 WSASend, WSARecv 함수호출 시 전달된 정보를 얻게 된다. 그리고 이 정보를 통해서 클라이언트에게 서비스를 제공하게 된다.

함수의 반환결과를 통해서 다음 두 가지 정보를 얻어야 한다.

- ✈ 입출력이 발생한 소켓의 핸들 정보
- ✈ 입출력과 관련된 데이터 송수신 버퍼의 정보

GetQueued~ 함수호출과 데이터의 전달관계

hClntSock과 hComPort와의 연결!
handleInfo는 IO 완료 시 전달할 데이터!

```
CreateIoCompletionPort(
    (HANDLE)hClntSock,
    hComPort,
    (DWORD)handleInfo,
    0
);
```

소켓 정보를
담아서 전달한다.

```
GetQueuedCompletionStatus(
    hComPort,
    &bytesTrans,
    (LPDWORD)&handleInfo,
    (LPOVERLAPPED*)&ioInfo,
    INFINITE
);
```

CP 오브젝트와 연결된 소켓을 대상으로
입출력 함수의 호출

```
WSARecv(
    handleInfo->hClntSock,
    &(ioInfo->wsaBuf),
    1,
    &recvBytes,
    &flags,
    &(ioInfo->overlapped),
    NULL
);
```

버퍼 정보를 담아서
전달한다.

CompletionPort에 할당되는 쓰레드의 의미

CP 오브젝트에 할당되는 쓰레드의 의미를 이해하기 위한 질문 1.

"그럼 IO에 어떻게 쓰레드를 할당해야 하나요?"

코드상에서 쓰레드의 CP 오브젝트 할당과정은 별도로 존재하는 않는다. CP 오브젝트에 할당된 쓰레드는 'CP 오브젝트를 대상으로 GetQueued~ 함수를 호출하는 쓰레드'를 뜻한다. 즉, CP 오브젝트에 할당가능 한 쓰레드의 수를 2개로 제한한다는 것은 GetQueued~ 함수를 동시에 호출할 수 있는 쓰레드의 수를 2개로 제한한다는 뜻이다.

CP 오브젝트에 할당되는 쓰레드의 의미를 이해하기 위한 질문 2.

"혹시 쓰레드가 자동으로 생성되어서 IO를 처리하게 되나요?"

GetQueued~ 함수를 호출하는 쓰레드는 IO를 처리하는 쓰레드이다. IO 처리에 필요한 결과를 GetQueued~ 함수가 반환하기 때문이다. 그리고 이러한 역할을 담당하는 쓰레드는 프로그래머가 직접 생성해야 한다.

IOCP 기반의 에코 서버 1

```
typedef struct     // socket info
{
    SOCKET hClntSock;
    SOCKADDR_IN clntAdr;
} PER_HANDLE_DATA, *LPPER_HANDLE_DATA;

typedef struct     // buffer info
{
    OVERLAPPED overlapped;
    WSABUF wsaBuf;
    char buffer[BUF_SIZE];
    int rwMode;    // READ or WRITE
} PER_IO_DATA, *LPPER_IO_DATA;
```

예제 IOCPEchoServ_win.c의 일부

오른쪽의 두 구조체중 PER_HANDLE_DATA는 클라이언트의 소켓 정보를, PER_IO_DATA는 버퍼 정보를 담고 있다.

```
hComPort=CreateIoCompletionPort(INVALID_HANDLE_VALUE, NULL, 0, 0);
GetSystemInfo(&sysInfo);
for(i=0; i<sysInfo.dwNumberOfProcessors; i++)
    _beginthreadex(NULL, 0, EchoThreadMain, (LPVOID)hComPort, 0, NULL);

hServSock=WSASocket(AF_INET, SOCK_STREAM, 0, NULL, 0, WSA_FLAG_OVERLAPPED);
servAdr.sin_family=AF_INET;
servAdr.sin_addr.s_addr=htonl(INADDR_ANY);
servAdr.sin_port=htons(atoi(argv[1]));

bind(hServSock, (SOCKADDR*)&servAdr, sizeof(servAdr));
listen(hServSock, 5);
```

IOCP 서버에서의 CP 오브젝트의 생성과 리스닝 소켓의 생성과정을 보이고 있다. 그리고 CPU(정확히는 코어)의 수만큼 쓰레드를 생성하여 EchoThreadMain 함수를 호출하도록 하고 있다.

IOCP 기반의 에코 서버 2

예제 IOCPEchoServ_win.c의 일부

```
while(1)
{
    SOCKET hClntSock;
    SOCKADDR_IN clntAdr;
    int addrLen=sizeof(clntAdr);

    hClntSock=accept(hServSock, (SOCKADDR*)&clntAdr, &addrLen);
    handleInfo=(LPPER_HANDLE_DATA)malloc(sizeof(PER_HANDLE_DATA));
    handleInfo->hClntSock=hClntSock;
    memcpy(&(handleInfo->clntAdr), &clntAdr, addrLen);

    CreateIoCompletionPort((HANDLE)hClntSock, hComPort, (DWORD)handleInfo, 0);

    ioInfo=(LPPER_IO_DATA)malloc(sizeof(PER_IO_DATA));
    memset(&(ioInfo->overlapped), 0, sizeof(OVERLAPPED));
    ioInfo->wsaBuf.len=BUF_SIZE;
    ioInfo->wsaBuf.buf=ioInfo->buffer;
    ioInfo->rwMode=READ;
    WSARecv(handleInfo->hClntSock, &(ioInfo->wsaBuf),
        1, &recvBytes, &flags, &(ioInfo->overlapped), NULL);
}
```

```
typedef struct      // socket info
{
    SOCKET hClntSock;
    SOCKADDR_IN clntAdr;
} PER_HANDLE_DATA, *LPPER_HANDLE_DATA;
```

```
typedef struct      // buffer info
{
    OVERLAPPED overlapped;
    WSABUF wsaBuf;
    char buffer[BUF_SIZE];
    int rwMode;     // READ or WRITE
} PER_IO_DATA, *LPPER_IO_DATA;
```

IOCP 기반의 에코 서버 3

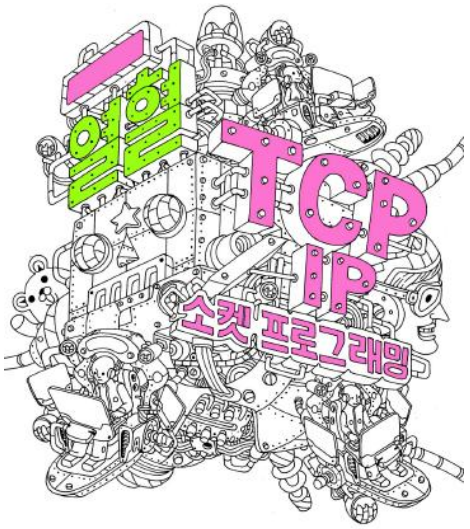
예제 IOCPEchoServ_win.c의 일부

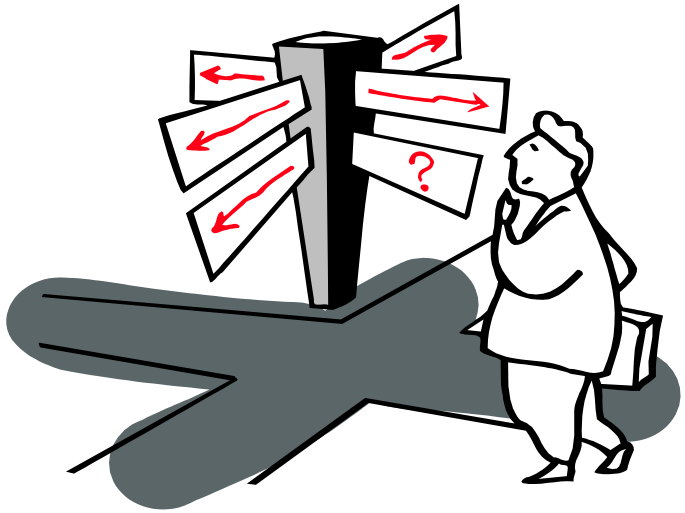
```
DWORD WINAPI EchoThreadMain(LPVOID pComPort)
{
    HANDLE hComPort=(HANDLE)pComPort;
    SOCKET sock;
    DWORD bytesTrans;
    LPPER_HANDLE_DATA handleInfo;
    LPPER_IO_DATA ioInfo;
    DWORD flags=0;
    while(1)
    {
        GetQueuedCompletionStatus(hComPort, &bytesTrans,
            (LPDWORD)&handleInfo, (LPOVERLAPPED*)&ioInfo, INFINITE);
        sock=handleInfo->hClntSock;
        if(ioInfo->rwMode==READ)
        {
            puts("message received!");
            if(bytesTrans==0)   // EOF 전송 시
            {
                closesocket(sock);
                free(handleInfo); free(ioInfo);
                continue;
            }
            memset(&(ioInfo->overlapped), 0, sizeof(OVERLAPPED));
            ioInfo->wsaBuf.len=bytesTrans;
            ioInfo->rwMode=WRITE;
```

이어짐

```
            WSASend(sock, &(ioInfo->wsaBuf),
                1, NULL, 0, &(ioInfo->overlapped), NULL);
            ioInfo=(LPPER_IO_DATA)malloc(sizeof(PER_IO_DATA));
            memset(&(ioInfo->overlapped), 0, sizeof(OVERLAPPED));
            ioInfo->wsaBuf.len=BUF_SIZE;
            ioInfo->wsaBuf.buf=ioInfo->buffer;
            ioInfo->rwMode=READ;
            WSARecv(sock, &(ioInfo->wsaBuf),
                1, NULL, &flags, &(ioInfo->overlapped), NULL);
        }
        else
        {
            puts("message sent!");
            free(ioInfo);
        }
    }
```

실제 클라이언트에 대한 서비스는 다음의 쓰레드 함수를 통해 완성된다. IOCP의 쓰레드는 GetQueued~ 함수의 반복호출을 통해서 클라이언트의 데이터를 수신하고 이에 대한 서비스를 제공한다.





Chapter 23이 끝났습니다. 질문 있으신지요?