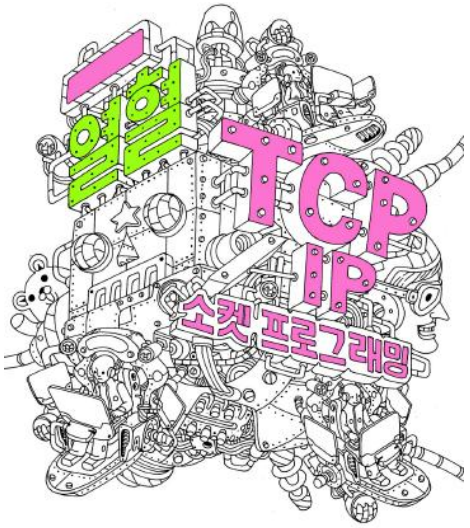
윤성우의 열혈 TCP/IP 소켓 프로그래밍

윤성우 저 열혈강의 TCP/IP 소켓 프로그래밍 개정판

Chapter 21. Asynchronous Notification IO 모델

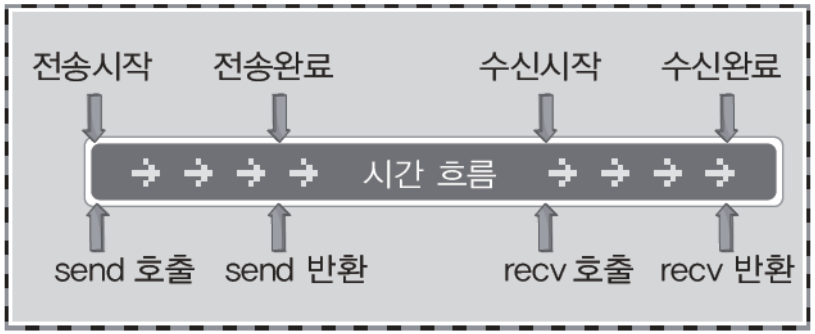


Chapter 21-1. 비동기(Asynchronous) Notification IO 모델의 이해

윤성우 저 열혈강의 TCP/IP 소켓 프로그래밍 개정판

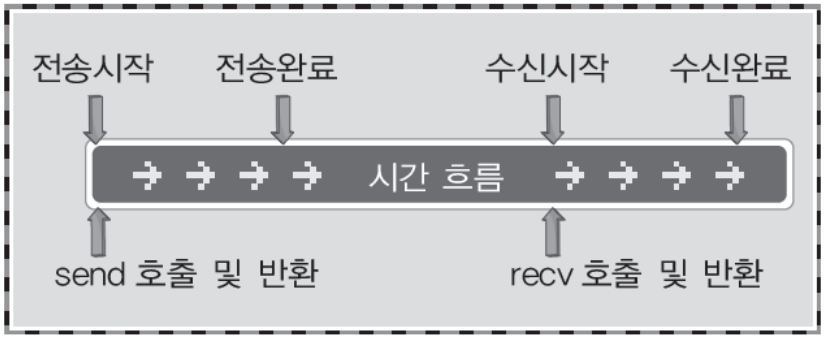
동기와 비동기에 대한 이해

동기 입출력



입출력 함수의 호출 및 반환의 시기가 데이터 전송의 시작 및 완료의 시기와 일치하는 방식의 함수호출, 따라서 **함수가 호출된 동안에는 다른 작업을 할 수가 없다는 단점**이 있다.

비동기 입출력



입출력 함수의 호출 시점이 데이터의 송수신이 시작되는 시기이다. 그러나 호출된 함수는 이내 반환을 하고, 내부적으로는 계속해서 입출력이 진행되는 방식의 함수호출이다. 따라서 **동기 입출력이 지니는 단점의 대안**이 된다.

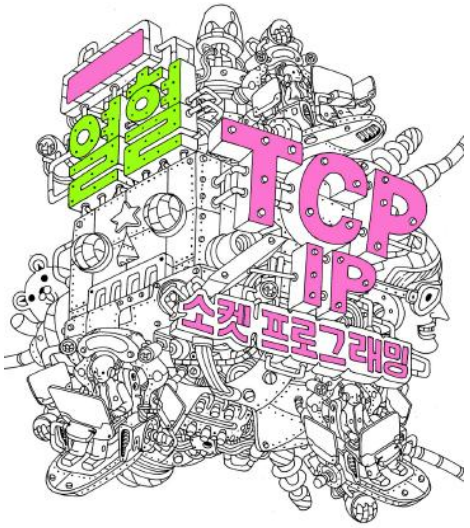
비동기 Notification 입출력 모델에 대한 이해

동기 Notification이란?

- 입출력의 Notification(알림)이 동기적으로 이루어지는 것
- select 함수는 입출력이 완료 또는 가능한 상태가 되었을 때 반환을 하므로, select 함수가 동기 Notification 모델의 대표적인 예이다.

비동기 Notification이란?

- 입출력의 Notification(알림)이 비동기적으로 이루어지는 것
- select 함수의 비동기 버전이 WSAEventSelect 함수이다.
- WSAEventSelect 함수는 입출력의 완료에 상관없이 무조건 반환을 한다. 따라서 입출력의 완료를 확인하기 위한 별도의 과정을 거쳐야 한다.
- WSAEventSelect 함수를 이용하면, IO의 상태변화를 명령한 다음, 이후에 기타 작업을 진행한 다음에 IO의 상태변화를 확인할 수 있는 있다.



Chapter 21-2. 비동기(Asynchronous) Notification IO 모델의 이해와 구현

윤성우 저 열혈강의 TCP/IP 소켓 프로그래밍 개정판

WSAEventSelect 함수와 Notification

```
#include <winsock2.h>

int WSAEventSelect(SOCKET s, WSAEVENT hEventObject, long lNetworkEvents);
    ➡ 성공 시 0, 실패 시 SOCKET_ERROR 반환
```

- s — 관찰대상인 소켓의 핸들 전달.
- hEventObject — 이벤트 발생유무의 확인을 위한 Event 오브젝트의 핸들 전달.
- lNetworkEvents — 감시하고자 하는 이벤트의 유형 정보전달.

매개변수 s에 전달된 핸들의 소켓에서 lNetworkEvents에 전달된 이벤트 중 하나가 발생하면, hEventObject에 전달된 핸들의 커널 오브젝트를 signaled 상태로 바꾸는 함수

소켓 하나당 한 번의 함수 호출이 진행된다는 사실에 주목! 한번 등록된 소켓은 select 함수와 달리 매 함수호출 때마다 재등록을 할 필요가 없다!

등록 가능한 이벤트의 종류

• FD_READ	수신할 데이터가 존재하는가?
• FD_WRITE	블로킹 없이 데이터 전송이 가능한가?
• FD_OOB	Out-of-band 데이터가 수신되었는가?
• FD_ACCEPT	연결요청이 있었는가?
• FD_CLOSE	연결의 종료가 요청되었는가?

manual-reset 모드 Event 오브젝트의 또 다른 생성방법


```
#include <winsock2.h>

WSAEVENT WSACreateEvent(void);
```

➜ 성공 시 Event 오브젝트 핸들, 실패 시 WSA_INVALID_EVENT 반환

```
#include <winsock2.h>

BOOL WSACloseEvent(WSAEVENT hEvent);
```

➜ 성공 시 TRUE, 실패 시 FALSE 반환

manual-rest 모드 Event 오브젝트 생성의 편의를 위해서 정의된 두 함수이다.

이벤트 발생유무의 확인에 사용되는 함수

```
#include <winsock2.h>

DWORD WSAWaitForMultipleEvents(
   DWORD cEvents, const WSAEVENT* lphEvents, BOOL fWaitAll, DWORD dwTimeout, BOOL fAlertable);
```

➜ 성공 시 이벤트 발생 오브젝트 관련정보, 실패 시 WSA_INVALID_EVENT 반환

- cEvents — signaled 상태로의 전이여부를 확인할 Event 오브젝트의 개수 정보 전달.
- lphEvents — Event 오브젝트의 핸들을 저장하고 있는 배열의 주소 값 전달.
- fWaitAll — TRUE 전달 시 모든 Event 오브젝트가 signaled 상태일 때 반환, FALSE 전달 시 하나만 signaled 상태가 되어도 반환.
- dwTimeout — 1/1000초 단위로 타임아웃 지정, WSA_INFINITE 전달 시 signaled 상태가 될 때까지 반환하지 않는다.
- fAlertable — TRUE 전달 시, alertable wait 상태로의 진입(이는 다음 Chapter에서 설명한다).
- 반환 값 — 반환된 정수 값에서 상수 값 WSA_WAIT_EVENT_0를 빼면, 두 번째 매개변수로 전달된 배열을 기준으로, signaled 상태가 된 Event 오브젝트의 핸들이 저장된 인덱스가 계산된다. 만약에 둘 이상의 Event 오브젝트가 signaled 상태로 전이되었다면, 그 중 작은 인덱스 값이 계산된다. 그리고 타임아웃이 발생하면 WAIT_TIMEOUT이 반환된다.

이벤트 발생유무의 확인의 모델

아래의 코드를 통해서 manual-reset 모드의 이벤트 오브젝트가 필요한 이유를 알 수 있다.

```
int posInfo, startIdx, i;
. . . . .
posInfo=WSAWaitForMultipleEvents(numOfSock, hEventArray, FALSE, WSA_INFINITE, FALSE);
startIdx=posInfo-WSA_WAIT_EVENT_0;
. . . . .
for(i=startIdx; i<numOfSock; i++)
{
    int sigEventIdx=WSAWaitForMultipleEvents(1, &hEventArray[i], TRUE, 0, FALSE);
    . . . . .
}
```

WSAWaitForMultipleEvents 함수의 반환 값으로 signaled 상태로 전이된 Event 오브젝트의 첫 번째(배열에 저장된 순서를 기준으로) 인덱스 값이 반환, 하지만 여기서 생성하는 Event 오브젝트가 manual-reset 모드이므로 위와 같은 형태로 signaled 상태가 된 Event 오브젝트 모두를 확인할 수 있다.

이벤트 종류의 구분을 위한 함수

hEventObject와 연결된 s 소켓에 발생한 이벤트의 정보를 lpNetworkEvents를 통해 전달하라!

```
#include <winsock2.h>

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

➜ 성공 시 0, 실패 시 SOCKET_ERROR 반환

- s — 이벤트가 발생한 소켓의 핸들 전달.
- hEventObject — 소켓과 연결된(WSAEventSelect 함수호출에 의해), signaled 상태인 Event 오브젝트의 핸들 전달.
- lpNetworkEvents — 발생한 이벤트의 유형정보와 오류정보로 채워질 WSANETWORKEVENTS 구조체 변수의 주소 값 전달.

```
typedef struct _WSANETWORKEVENTS
{
    long lNetworkEvents;
    int iErrorCode[FD_MAX_EVENTS];
} WSANETWORKEVENTS, *LPWSANETWORKEVENTS;
```

lnetworkEvent에는, 수신할 데이터가 존재하면 FD_READ가 저장되고, 연결요청이 있는 경우에는 FD_ACCEPT가 담긴다. 그리고 iErrorCode에는 이벤트 FD_XXX 관련 오류가 발생하면 iErrorCode[FD_XXX_BIT]에 0 이외의 값이 저장된다.

이벤트 종류의 구분과 오류검사의 예

이벤트 종류의 구분

```
WSANETWORKEVENTS netEvents;
. . . . .
WSAEnumNetworkEvents(hSock, hEvent, &netEvents);
if(netEvents.lNetworkEvents & FD_ACCEPT)
{
    // FD_ACCEPT 이벤트 발생에 대한 처리
}
if(netEvents.lNetworkEvents & FD_READ)
{
    // FD_READ 이벤트 발생에 대한 처리
}
if(netEvents.lNetworkEvents & FD_CLOSE)
{
    // FD_CLOSE 이벤트 발생에 대한 처리
}
```

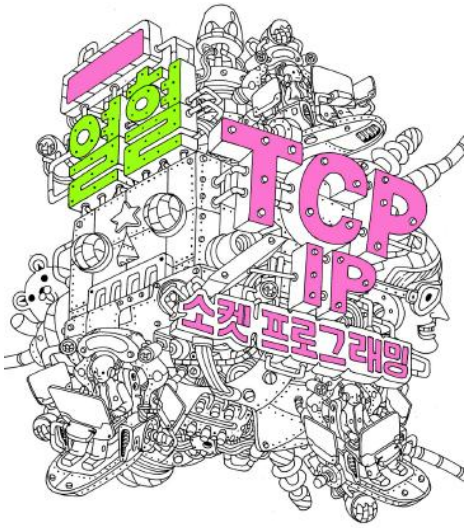
오류의 검사

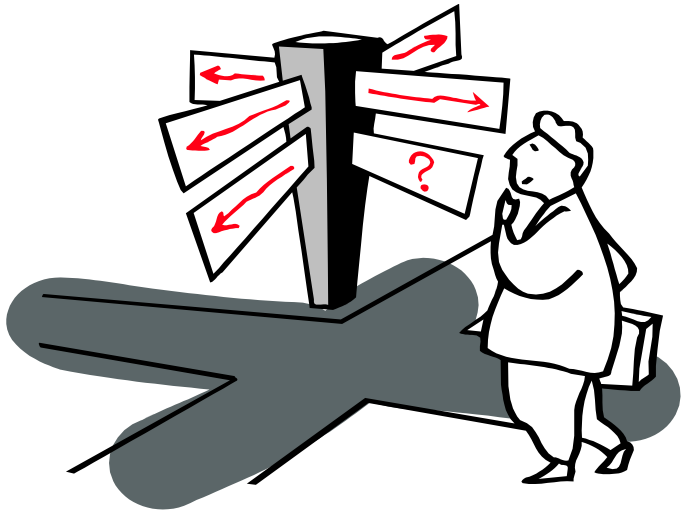
```
WSANETWORKEVENTS netEvents;
. . . . .
WSAEnumNetworkEvents(hSock, hEvent, &netEvents);
. . . . .
if(netEvents.iErrorCode[FD_READ_BIT]!=0)
{
    // FD_READ 이벤트 관련 오류발생
}
```

비동기 Notification IO 모델의 에코 서버 구현

코드의 분량이 상당하므로, 예제 AsynNotiEchoServ_win.c를 보면서 강의를 진행합니다. 특히, 소켓 생성시마다 추가로 생성하는 이벤트 오브젝트의 관리 방법을 관찰하기 바라며, select 함수와의 차이점으로 무엇이 있는지 관찰을 합니다. 특히 select 함수와의 가장 큰 차이점 둘은 다음과 같습니다.

1. 입출력을 확인하는 방법
2. 입출력의 대상에 소켓을 등록하는 방법





Chapter 21이 끝났습니다. 질문 있으신지요?