네트워크 II

(14주차)

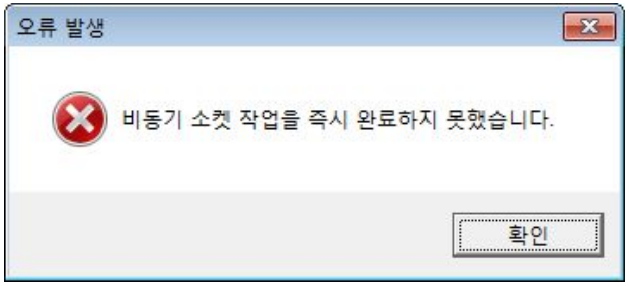
학습개요

- 학습 목표
  - 동기 방식과 비동기 방식의 차이를 이해한다.
  - MFC를 이용한 GUI 기반 소켓 프로그래밍 기법을 익힌다.

- 학습 내용
  - 동기와 비동기
  - MFC 소켓 프로그래밍 (GUI)
  - 실습

동기와 비동기 개념

- 비동기 함수 오류 메시지



- 동기와 비동기 함수의 특성
  - 동기(Synchronous = Blocking) 함수
    - 작업 처리 조건이 만족될 때까지 함수가 리턴하지 않고 대기한다.
  - 비동기(Asynchronous = Nonblocking) 함수
    - 작업 처리 조건이 만족될 때까지 함수가 대기하지 않고 리턴한다.

동기와 비동기 개념

- 동기 함수의 문제점

```
void CMyTestView::OnLButtonDblClk(UINT nFlags, CPoint point)
{
    ...
    char buf[256];
    socket.Recv(buf, 256); // 받은 데이터가 없으면 여기서 메시지 루프가 정지된다.
}
```

  - 해결 ⇨ 별도의 스레드를 만들어서 동기 함수를 호출하거나 비동기 함수를 사용해야 한다.

동기와 비동기 개념

- CSocket 클래스
  - 중요 함수의 경우 동기적으로 동작하되, 함수 호출 시 메시지 루프가 정지하는 일이 없도록 내부적으로 윈도우 메시지를 처리
    - (예) Accept(), Receive(), Send(), Connect(), ReceiveFrom(), SendTo()
  - 동기 함수를 호출해야 할 시점을 가상 함수 호출로 알려줌. CAsyncSocket 클래스가 제공하는 On~ 함수를 정의해 두면 자동으로 해당 On~ 함수가 호출됨.
    - (예) 클라이언트가 접속하면 가상 함수 OnAccept()가 자동으로 호출됨. 따라서 OnAccept() 함수를 재정의하고 여기서 Accept() 함수를 호출

MFC 소켓 프로그래밍 - GUI

- CSocket 클래스를 이용한 응용 프로그램 작성
  - ① CSocket 클래스를 상속받아 새로운 소켓 클래스를 생성
  - ② CAsyncSocket 클래스가 제공하는On~ 가상 함수를 필요에 따라 재정의
  - ③ 재정의한 On~ 가상 함수에서 적절한 소켓 함수를 호출

    (예) OnAccept() 함수에서는 Accept() 함수를 호출하고,
    OnReceive() 함수에서는 Receive() 함수를 호출

실습

직렬화를 이용한 소켓 입출력

- 데이터를 보내는 경우

```
CSocket sock;
int data1;
double data2;
…
sock.Send(&data1, sizeof(data1));
sock.Send(&data2, sizeof(data2));
```

```
CSocket sock;
int data1;
double data2;
…
CSocketFile file(&sock);
CArchive ar(&file, CArchive::store);
ar << data1;
ar << data2;
```

직렬화를 이용한 소켓 입출력

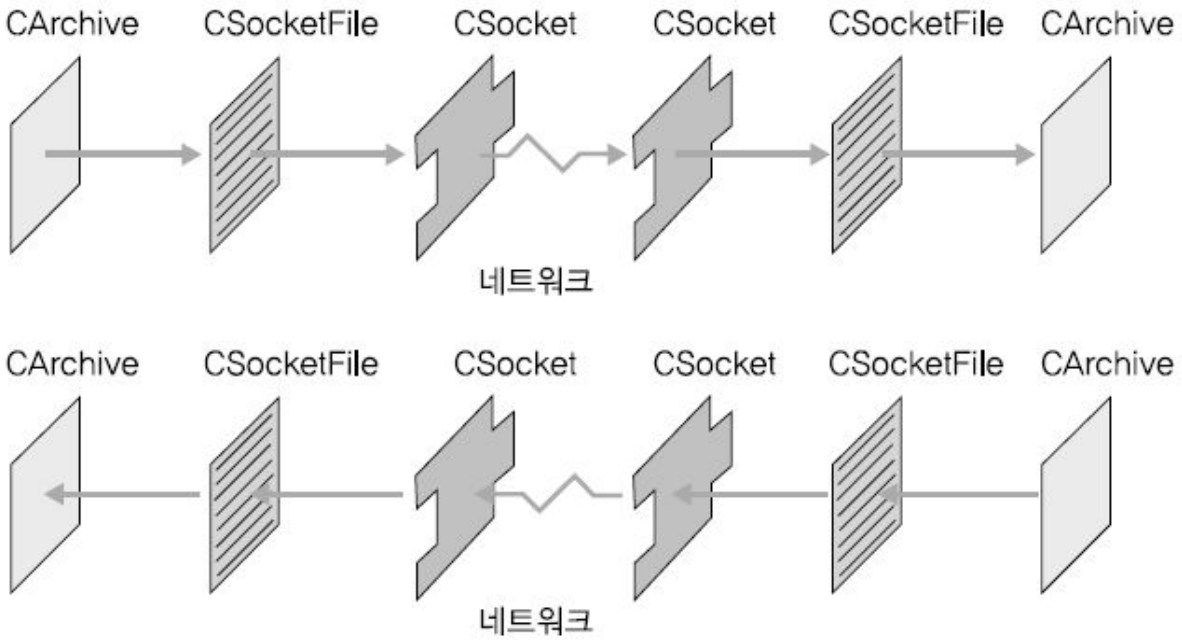
- 데이터를 받는 경우

```
CSocket sock;
int data1;
double data2;
…
sock.Recv(&data1, sizeof(data1));
sock.Recv(&data2, sizeof(data2));
```

```
CSocket sock;
int data1;
double data2;
…
CSocketFile file(&sock);
CArchive ar(&file, CArchive::load);
ar >> data1;
ar >> data2;
```

직렬화를 이용한 소켓 입출력

- 직렬화를 이용한 소켓 입출력 과정



직렬화를 이용한 소켓 입출력

- 장점
  - 입출력 함수를 직접 사용하지 않으므로 프로그래머가 입출력과 관련된 세부 사항을 신경쓰지 않아도 됨
  - 통신 양단의 문자열 저장 방식이 달라도 내부적으로 변환하여 올바로 처리
  - 통신 양단의 바이트 정렬(Byte Ordering)이 달라도 내부적으로 변환하여 올바로 처리

직렬화를 이용한 소켓 입출력

- 단점
  - 통신 양단이 모두 MFC의 직렬화를 이용한 소켓 입출력을 하는 경우에만 올바른 데이터 교환이 보장됨
  - CSocket 클래스 대신 CAsynSocket 클래스를 직접 사용할 경우에는 직렬화를 쓸 수 없음

학습정리

- 동기 함수는 작업 처리 조건이 만족될 때까지 함수가 리턴하지 않고 대기하며, 비동기함수는 작업 처리 조건이 만족될 때까지 함수가 대기하지 않고 바로 리턴합니다.
- 동기함수의 블로킹으로 인해 메시지 루프가 정지되므로 별도의 스레드를 만들어서 동기 함수를 호출하거나 비동기 함수를 사용해 문제를 해결해야 합니다.
- MFC 소켓 응용 프로그램은 CSocket 클래스를 상속받아 새로운 소켓 클래스를 생성하고, CAsyncSocket 클래스가 제공하는On 가상 함수를 필요에 따라 재정의해 적절한 소켓 함수를 호출해 사용합니다.
- 통신 양단이 MFC의 직렬화를 이용한 소켓 입출력을 하는 경우 통신 양단의 문자열 저장 방식, Byte Ordering이 달라도 내부적으로 변환하여 올바로 처리됩니다.
- CSocket 클래스 대신 CAsynSocket 클래스를 직접 사용할 경우에는 직렬화를 쓸 수 없습니다.