네트워크 응용시스템 개발을 위한 WIN Socket 연구

백승범* · 최재원*

경성대학교 컴퓨터공학과

A Study on WIN Sockets for Development of Network Applicarion Systems

Seng bum Beak* · Jae Weon Choe**

Dept of Computer Enginerring *Kyungsung University

E-mail: covi@hanmir.com

요 약

네트워크 응용 프로그램을 개발하기 위해서는 Socket API를 많이 이용한다. Socket API를 이용해 정상적인 동작을 하는 것은 아주 쉽다. 실제 개발 과정에서 정상적인 상황이 아닌 여러 가지 비정상적인 오류에 대처하게 됨에 따라 속수무책일 때가 많다. 따라서 각 플랫폼별로 비정상적인 오류들의 사례를 연구를 통하여 효율적인 응용시스템 개발의 필요성이 대두되었다. 본 연구에서는 프로그램의 각 CASE별로 네트워크 프로그램의 분석과 구조를 잡아 네트워크 응용 프로그램 개발시 복잡한 하위 계층의 구조를 완벽히 이해하지 않고 쉽게 개발할 수 있도록 한다.

키워드

WINSocket, WASStartup, WSACleanup, socket, bind, listen, accept, connect

Ⅰ. 서 론

이제 더이상 인터넷(Internet)이란 말이 일반인들에게 생소한 말이 아니다. 너 나 할거 없이 인터넷을 알고 싶어하고 인터넷을 알아야 한다는 필요성을 절감한다.

다른 많은 네트워킹 기술을 제치고 이제 인터넷은 컴퓨터 네트워킹과 거의 동의어가 되어버릴 정도로 네트워크 기술의 중심에 서 있다. 인터넷 애플리케이션들은 고전적인 텍스트 기반의 애플리케이션인 이메일, 파일전송, 뉴스그룹 등으로부터 최근의 WWW, 인터넷 전화, 비디오 회의, VOD(Video On Demand) 등 매우 다양하며, 계속해서 그 영역을 확장시켜 가고 있다. 이러한 애플리케이션들은 네트워크의 종단 컴퓨터에서 수행되는 소프트웨어에 의해 구현된다. 이러한 애플리케이션들은 여러 공통의 요구사항을 가지는 것이 보통인데, 예를 들면 신속한, 오류 없는, 순서대로의 데이터 전송을 요구하는 것 등이다.

Socket프로그래밍은 Socket API를 이용해 정상적인 동작을 하는 것은 아주 쉽다. 그러나 실제 개발과정에서는 정상적인 상황이 아닌 여러 가지 비정상적인 오류에 대처하게 됨에 따라 속수 무책일 때가 많다. 따라서 각 플랫폼별로 비정상적인 오류들의 사례연구를 통하여 효율적인 응용시스템 개발의 필요성이 대두되었다고 할 수 있다.

실제 개발 과정에서 이러한 상황에 처해있을 때 개발자들은 당황하기 쉽다. 따라서 이러한 상황을 가정하여 여러 가지 사례별로 오류에 대해 접근하여 그것에 대해 확인해 봄으로 인해 프로그래밍 시간의 단축 및 효율성이 증대되리라 본다.

따라서 본 연구에서는 여러 가지 Socket프로그래밍 중에서도 가장 대표적이라 할 수 있는 BSD Socket과 WIN Socket의 차이점을 알아보고 여러 가지 사례들에 대해 각각의 플랫폼별로 사례연구를 통해 WIN Socket프로그래밍을 이용한 네트워크프로그래밍 응용시스템 개발에 활용하고자 한다.

Ⅱ. 이론적 배경

1. UNIX BSD Socket과 WIN Socket의 차이

WIN Socket의 사용방법과 문법이 BSD Socket의 경우와 유사하기는 하지만 WIN Socket과 BSD Socket용으로 작성된 프로그램은 서로 호환성이 없으며 이 두 가지 Socket 응용 프로그램이 호환성을 갖으

려면 많은 부분을 수정하여야 한다.

먼저 윈도우와 UNIX 운영체제의 차이점을 비교하면 다음과 같다.

첫째, UNIX와 달리 윈도우3.1 이하에서는 멀티태스킹을 지원하지 않는다.

둘째, WIN Socket은 주로 윈도우 프로그래밍으로 구현되는데, 윈도우 프로그래밍이 '메시지 구동형 프로그래밍'이라는 것이 UNIX 프로그램과 크게 다른 점이다.

2. WIN Socket의 동작모드

1) Blocking 모드

socket() 시스템 콜을 호출하여 하나의 Socket을 만들면 이것은 디폴트로 blocking 모드로 동작하는 Socket이 된다.

2) Non-blocking 모드

함수가 호출되었을 때 함수의 원하는 동작이 완료되는 것과 무관하게 일단 함수가 즉시 리턴 되는 Socket을 말한다.

3) Asynchronous(비동기) 모드

비동기 모드에서도 non-blocking 모드에서처럼 Socket 관련 함수의 호출이 바로 리턴 된다. 그러나 비동기 모드에서는 non-blocking 모드와 달리, 함수의 동작이 완료되는 시점, 또는 함수의 실행이 시작되지 못한 경우 다음에 다시 재 시도하여야 하는 시점을 시스템(Winsock.dll)이 메시지 처리 방식으로 나중에 응용 프로그램에게 알려준다.

3. WIN Socket의 기본개념

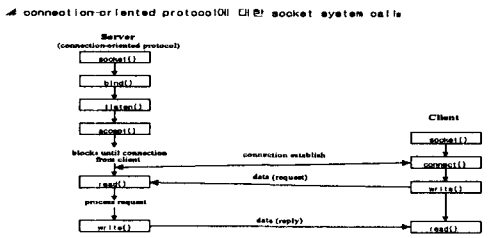

그림 1. TCP의 Socket System Call

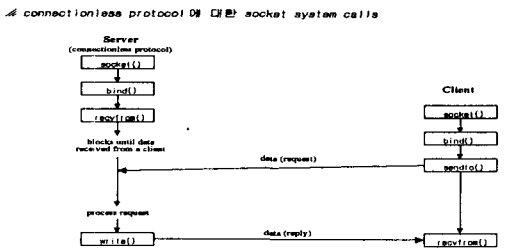

그림 2. UDP의 Socket System Call

4. Wiin Socket의 API

1) WSAStartup()

WINSocket을 사용하기 위해 먼저 WSAStartup을 실행하게 된다. 이 함수는 winsock.dll을 확인하기 위해 사용된다. Winsock.dll의 버전 체크와 DLL Load를 위해서 반드시 사용해야 하는 함수이다.

2) WSACleanup()

사용이 끝난 winsock.dll을 해제한다.

3) socket()

socket(int af, int type, int protocol)

Socket은 socket을 만드는 함수로서 UNIX의 표준 포맷을 따른다. 통신에 필요한 Socket 디스크립터 생성한다.

4) bind()

int bind (SOCKET s, const struct sockaddr FAR* name, int namelen);

Socket에 IP 주소와 포트 번호 할당한다.

5) connect()

int PASCAL connect(SOCKET s, const struct sockaddr *name, int namelen);

연결을 요청하는 함수이다.

6) listen()

int PASCAL listen(SOCKET s, int backlog);

바인드된 socket을 가지고 외부에서 접속 요청을 기다리는 함수이다.

7) accept()

SOCKET PASCAL accept(SOCKET s, LPSOCKETADDR lpAddr, LPINT lpLength);

외부에서 접속 요청이 있을 때 이를 수락하고 접속을 하는 함수이다.

8) closesocket()

사용이 끝난 socket을 close한다. Listening socket은 서버프로그램이 서비스 제공이 완전히 종료될 때만 닫는다.

9) select()

FD_READ, FD_WRITE, FD_LISTEN, FD_CLOSE 등의 event가 발생했는지 체크한다.

10) recv()

Socket으로부터 도착한 데이타를 버퍼로 읽어들인다.

11) send()

Socket을 통하여 데이타를 전송한다.

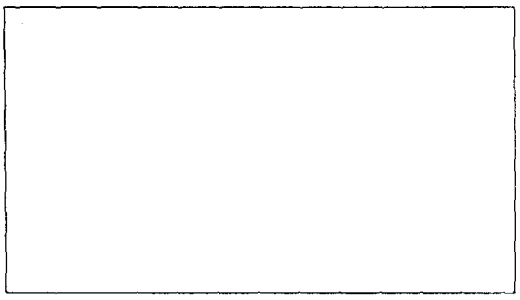

그림 3 WIN Socket 서버와 WIN Socket 클라이언트의 연결관계

III. WIN Socket 프로그래밍 환경

1. 프로그래밍 환경

Chatserver.cpp , Chatclinet.cpp의 포트번호를 변경한다. 서버 프로그램 실행 시에 포트번호의 중복을 방지하기 위해서이다. ftp를 이용하여 임의의 두 머신의 Chatclient.cpp와 Chatserver.cpp를 ASICII 모드로 전송한다.

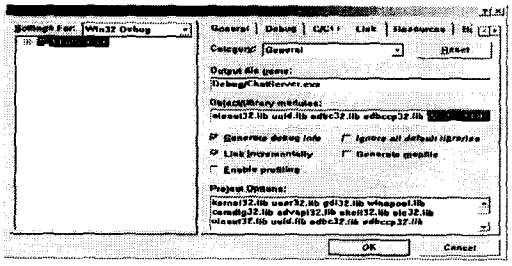

그림 4 Visual C++의 Project Setting

WS2_32.lib를 project 의 link 부분에 등록한다.

2. Standard WIN Socket 프로그램

```
                    ... 중략 ...
DWORD WINAPI ClientThread(LPVOID lpParam)
{   SOCKET sock = (SOCKET)lpParam;
    char szBuff[DEFAULT_BUFFER];
                    ... 중략 ...
    ret = recv (sock, szBuff, DEFAULT_BUFFER, 0);
                    ... 중략 ..}
int main ()
{                   ... 중략 ...
sListen = socket(AF_INET, SOCK_STREAM, 0);
                    ... 중략 ...
if(bind (sListen, (struct sockaddr *)&local, sizeof(local))
== SOCKET_ERROR)
{
    cout << "Bind fail : " << WSAGetLastError();
    return -1; }
listen (sListen, 8);
while(1)
{ iAddrSize = sizeof(client);
    sClient = accept (sListen, (struct
sockaddr *)&client, &iAddrSize);
                    ... 중략 ...
    closesocket(sClient);
    closesocket(sListen);
    WSACleanup();
                    ... 중략 ..}
```

표 1 Chatserver.cpp 소스 요약

```
                    ... 중략 ...
int main ()
{                   ... 중략 ...
if (WSAStartup (MAKEWORD (2, 2), &wsd) != 0)
{ cout << "[Client] : Failed to load WINSocket library"
<< endl;
    return -1; }
                    ... 중략 ...
Csock = socket (AF_INET, SOCK_STREAM, 0);
                    ... 중략 ...
cin.getline(sServerIP, 128);
cout << "Input Connect Server Port Number: ";
                    ... 중략 ...
server.sin_family = AF_INET;
server.sin_addr.s_addr = inet_addr(sServerIP);
server.sin_port = htons(iPort);
                    ... 중략 ...
else cout<<"[Client] : Connect success" << endl;
senddata=send(Csock, sendline,
strlen(sendline), 0);
                    ... 중략 ...
recvdata = recv(Csock, recvline, MAXLINE, 0);
                    ... 중략 ...
closesocket (Csock);
WSACleanup ();
                    ... 중략 ...
}
```

표 2 Chatclinet.cpp 소스 요약

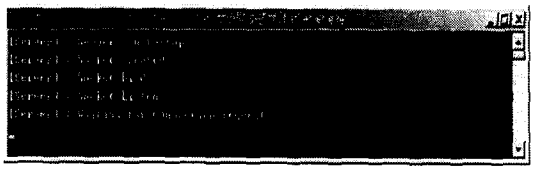

그림 5 소켓 서버 실행 상태

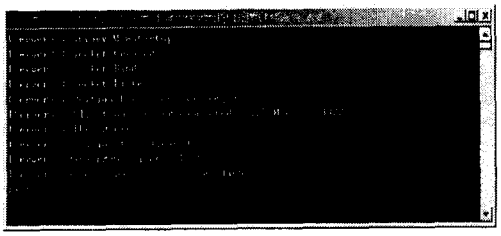

그림 6 소켓 클라이언트로부터 data 전달받은 서버 상태

IV. WIN Socket 프로그래밍 연구

1. TCP의 신뢰성 있는 통신

TCP 프로토콜은 신뢰성 있는 통신을 보장한다. 연결이 확립되고 data를 전달하는 과정에서 오류제어, 흐름제어, 시퀀스제어를 하기 때문이다.

2. 동시 서비스 요청에 대한 서버의 순차적 서비스

세 개의 클라이언트로부터 서비스 요청을 받아 처리하는 과정을 보면 가장 먼저 접속을 한 클라이언트부터 서비스를 순차적으로 한다. WIN Socket에서는 concurrent 서비스를 위하여 쓰레드를 생성하는 반면 BSD Socket에서는 fork과정을 통해 process를 생성한다.

3. 동시 서비스 요청에 대한 서버의 병렬적 서비스

세 개의 클라이언트로부터 서비스 요청을 받아 처리하는 과정을 보면 가장 먼저 Data를 보낸 클라이언트부터 서비스를 하지만 동시에 나머지 두 개의 클라이언트도 서비스를 하고 있다.WIN Socket에서는 concurrent 서비스를 위하여 스레드를 생성하는 반면 BSD Socket에서는 fork과정을 통해 process를 생성한다.

4. 서버/클라이언트의 포트 바인딩

서버는 바인딩이 필요 하지만 클라이언트는 바인딩을 하지 않는 것을 원칙으로 한다. BSD Socket에서는 서버 프로그램을 실행 시에는 아무런 문제가 발생하지 않고 클라이언트가 connect 요청하는 시점에서 서버 프로그램에 에러가 발생한다. 하지만 WIN Socket에서는 클라이언트의 요청과 무관하게 서버 프로그램의 실행만으로도 에러를 출력한다.

5. 바인드 에러의 발생에 관해.

BSD Socket에서는 서버 프로그램을 실행시에는 아무런 문제가 발생하지 않는다. 클라이언트가 connect 요청하는 시점에서 서버 프로그램에 에러가 발생한다. 하지만 WIN Socket에서는 클라이언트의 요청과 무관하게 서버 프로그램의 실행만으로도 에러를 출력한다.

클라이언트도 바인드를 하지 않았을 경우 바인드를 해도 위와 같이 에러가 발생하지는 않지만 같은 클라이언트에서 여러 어플리케이션 동시에 실행되고 모두 같은 포트로 바인드가 된다면 문제는 발생할 것이다. 그래서 클라이언트에서는 원칙적으로 바인드를 실행하지 않는다.

6. 큐의 크기에 따른 동시연결 요청의 한계

WIN Socket과 달리 BSD Socket에서는 listen의 큐 사이즈를 '1'로 하였을 때 두 번째 클라이언트의 접속 요청까지 받고 세 번째부터 접속을 할 수가 없었다. 그 차이는 WIN Socket 에서는 첫 번째 클라이언트의 접속 요청도 큐에 저장되지만 BSD Socket에서는 저장되지 않고 바로 접속됨을 알 수 있다.

7. UDP 통신의 기본 구조와 동작, 신뢰성

비연결지향형이므로 connect(), listen()등이 사용되지 않는다. 동일 LAN 상이지만 버퍼크기를 넘는 양의 Data는 신뢰성을 얻을 수 없었다. UDP는 모든 데이터를 한번에 접근하기 때문이다.

V. 결 론

소켓프로그래밍은 Socket API를 이용해 정상적인 동작을 하는 것은 아주 쉽다. 그러나 실제 개발과정에서는 정상적인 상황이 아닌 여러 가지 비정상적인 오류에 대처하게 됨에 따라 속수 무책일 때가 많다. 따라서 각 플랫폼별로 비정상적인 오류들의 사례연구를 통하여 효율적인 응용시스템 개발의 필요성이 대두되었다고 할 수 있다.

실제 개발 과정에서 이러한 상황에 처해있을 때 개발자들은 당황하기 쉽다. 따라서 이러한 상황을 가정하여 여러 가지 사례별로 오류에 대해 접근하여 그것에 대해 확인해 봄으로 인해 프로그래밍 시간의 단축 및 효율성이 증대되리라 본다.

본 연구에서는 BSD Socket과 WIN Socket프로그래밍의 차이점과 여러 가지 사례들에 대해 각각의 플랫폼별로 사례연구를 통해 소켓프로그래밍을 이용한 네트워크프로그래밍 응용시스템 개발에 활용하고자 한다.

향후 JAVA Socket의 연구도 해야 할 것이다.

참고문헌

[1] Douglas E. Comer, "Computer Network and Internets" PRENTICE HALL, 1997.

[2] Derek Hamner, Merlin Hughes, Michael Shoffner, Umesh Bellur, "Java Network Programming 2/E", MANNING, 1999.

[3] Dove Roberts, "DEVELOPING FOR THE INTERNET WITH Winsock 2", 에프원, 1999.