네트워크 응용시스템 개발을 위한 BSD Socket 연구

최준원* · 최재원*

*경성대학교 컴퓨터공학과

A Study On BSD Socket for Development of Network Application Systems.

Jun-won Choi* · Jae-weon Choe**

*Dept. of Computer Engineering KyungSung University

E-mail : imacess@yahoo.co.kr

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 처하게 됨에 따라 속수무책일 때가 많다. 따라서 각 플랫폼별로 비정상적인 오류들의 사례를 연구를 통하여 효율적인 응용시스템 개발의 필요성이 대두되었다.

키워드

Socket, bind, listen, accept, connect, send, recv

Ⅰ. 서 론

국내 초고속 인터넷 서비스 시장은 다양한 서비스와 폭발적인 가입자 증가로 전 세계에서도 그 유례를 찾아볼 수 없을 정도로 활성화되어 있다. 국내 초고속 인터넷 서비스 가입자는 2000년 1월까지 만해도 30만 명에 불과했다. 하지만 2001년 12월말 기준으로 국내 초고속 인터넷 서비스 가입자 는 이미 780만 명을 넘어섰다.

다른 많은 네트워킹 기술을 제치고 이제 인터넷은 컴퓨터 네트워킹과 거의 동의어가 되어버릴 정도로 네트워크 기술의 중심에 서 있다. 네트워크 애플리케이션은 인터넷의 존재 이유라 할 수 있다. 최종 사용자의 관점에서 인터넷은 인터넷에 연결된 자신의 컴퓨터를 통하여 통신하고자 하는 하나의 또는 다수의 상대방 컴퓨터와 의미 있는 데이터를 주고받는 수단일 뿐이다. 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.

소켓프로그래밍은 Socket API를 이용해 정상적인 동작을 하는 것은 아주 쉽다. 그러나 실제 개발과정에서는 정상적인 상황이 아닌 여러 가지 비정상적인 오류에 처하게 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 소켓프로그래밍 중에서도 가장 대표적이라 할 수 있는 BSD 소켓프로그래밍의 여러 가지 사례들에 대해 각각의 플랫폼별로 사례연구를 통해 소켓프로그래밍을 이용한 네트워크 프로그래밍 응용시스템 개발에 활용하고자 한다.

II. 이론적 배경

1. OSI 참조 모델

OSI 참조 모델은 네트워크 통신을 위한 기본 모델이다. OSI 참조 모델의 의미는 앞으로 나올 네트워크 기기들의 개발을 가속화하기 위한 기술 발전의 축이 된다. TCP/IP는 OSI 모델과는 다른 네트워크 구조를 가지고 있다.

OSI 참조 모델은 네트워크가 제공하는 기능을 7개의 계층으로 나누어 식별하며, 각 계층은 독특한 기능을 가지고, 계층들간에 서로 의존한다. 각 계층은 자신의 바로 상위 계층이나 하위 계층과 통신하며 한 계층에서 문제가 발생하면 그 계층이 주위의 계층에 피드백을 제공해야 한다.

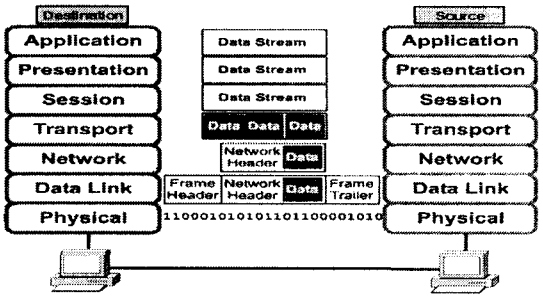

그림 1. OSI 참조모델과 캡슐화

2. TCP 프로토콜

TCP는 IP 데이터 그램에 올바른 데이터가 저장되게끔 보장한다. TCP와 UDP를 포함한 모든 전송 계층 프로토콜이 기본적인 배달 서비스에 IP를 사용하지만, IP는 데이터 그램이나 패킷이 목적지에 올바르게 전달된다고 보장하지 않는 신뢰성이 결여된 프로토콜이다. 패킷이 목적지로 향하는 과정에서 다른 네트워크를 통과하면서 순서가 바뀔 수 있으며, 인터넷에서 운반되면서 무수한 문제가 발생할 수 있다.

두 애플리케이션이 통신할 때마다 가상 회선을 제공하는 TCP를 사용하여 신뢰성을 얻을 수 있다. 두 TCP 기반 애플리케이션에 데이터를 전송할 때 둘 사이에 가상 회선이 개설된다. 모든 데이터가 성공적으로 전송되어 수신되면 둘 사이의 연결은 종결된다.

두 애플리케이션이 TCP를 사용하여 통신할 때 두 TCP 사이에 가상 회선이 개설된다. 가상 회선은 신뢰성, 흐름 제어, 그리고 I/O 관리 기능을 제공하여 TCP를 UDP와 구분 짓게 한다.

네트워크의 호스트들은 처리 능력, 메모리, 네트워크 대역폭 및 기타 자원을 포함하여 서로 다른 특성을 갖는다. 따라서 모든 호스트가 같은 속도로 데이터를 송수신할 수 없으므로 TCP는 이런 호스트의 속도 차

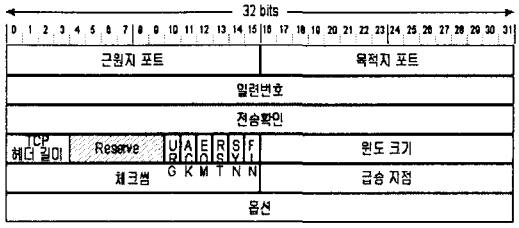

그림 2. TCP 헤더

이에 대응할 수 있어야 한다. TCP는 전송 데이터를 감시하여 신뢰할 만한 전송 서비스를 제공한다. TCP는 일련 번호를 사용하여 데이터의 개별적인 바이트를 감시하며, acknowledgment 플래그를 통해 일부 바이트가 분실되었는지 여부를 판별한다. 그리고 체크섬을 통해 데이터의 유효성을 검사한다.

3. BSD Socket의 기본개념과 API

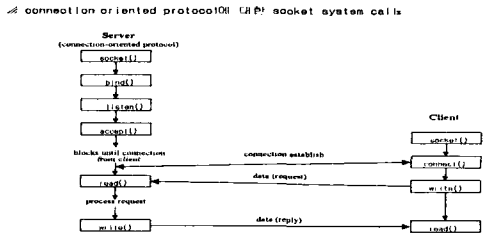

그림 3. TCP 프로토콜의 Socket System Call

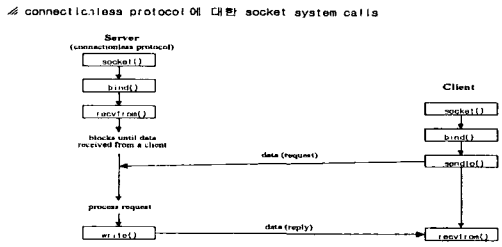

그림 4. UDP 프로토콜의 Socket System Call

```
#include <sys/socket.h>
int socket{
  int domain, /* 프로토콜 체계 */
  int type,     /* 서비스 타입 */
  int protocol /* 소켓에서 사용할 프로토콜 */
 };
struct sockaddr_in {
  short sin_family;  /* 주소 체계 */
  u_short sin_port;  /* 16비트 포트 번호 */
  struct in_addr sin_addr; /* 32비트 IP 주소 */
  char sin_zero[8];
 };
                    ... 중략 ...
```

그림 5. 소켓 구조체와 헤더파일

표 1. TCP 소켓의 데이터 송수신 함수 비교

문법	인수
int send(int s, char* buf, int length, int flags);	s : 소켓 번호
int write(int s, const void* buf, int length);	buf : 전송할 데이터가 저장된 버퍼
int recv(int s, char* buf, int length, int flags);	length : buf 버퍼의 크기
int read(int s, void* buf, int length);	flags : 보통 0

III. BSD Socket 프로그래밍 연구

1. TCP 통신 프로그램의 기본 구조와 동작.

```
                        ... 중략 ...
  if ((Rsock = socket(AF_INET, SOCK_STREAM, 0))
==-1) {
      printf("Server Error! socket() error");
      exit(-1);
  }
  printf("[Server] Socket created.\n");
                        ... 중략 ...
  if (bind(Rsock, (struct sockaddr *) &SockAddr,
  sizeof(SockAddr))) {
    printf("[Server] Server Error! bind() error");
    exit(-1);
  }
  printf("[Server] Socket binded.\n");
  if (listen(Rsock, 5)) {
    printf("[Server] Server Error! listen() error");
    exit(-1);
   }
   printf("[Server] Socket listened.\n");
                        ... 중략 ...
  if ((Csock = accept(Rsock, (struct sockaddr *)
  &ClientAddr, &ClientAddrLen)) < 0) {
    printf("[Server] Server Error! accept() error\n");
    printf("Rsock = %d  Csock = %d\n", Rsock,
    Csock);
    exit(-1);
  }
                        ... 중략 ...
  if ((num = recv(Csock, buf, sizeof(buf), 0)) < 0) {
    printf("[Child] Receive Error!\n");
    exit(-1);
  }
                        ... 중략 ...
```

그림 6. TCP 통신 기본 프로그램

그림 7. TCP 소켓 서버 실행 상태

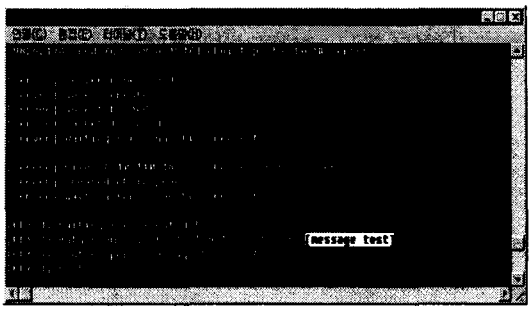

그림 8. TCP 소켓 클라이언트로부터 data 전달받은 서버 상태

BSD의 기본 API와 기본 루틴을 사용하여 프로그램을 작성한 후 확인한 결과 정상적인 동작이 이루어짐을 확인할 수 있다. 실행동작을 살펴보면 서버는 소켓이 생성되고, TCP 계층의 인터페이스인 포트가 소켓에 바인드 되어진다. Queue size로 동시 접속 가능한 클라이언트의 수를 지정하고, accept()에서 블록킹 상태로 머물면서 클라이언트의 접속을 기다린다. 클라이언트는 소켓이 생성되고 서버에게 접속 요청을 보내서 연결을 확립한다.

연결이 확립되면 서버 측에선 Rsock이 Csock을 fork()하여 클라이언트에게 서비스를 한다.

2. TCP의 신뢰성 있는 통신.

TCP 통신 프로토콜을 이용하여 데이터를 전송 시 신뢰성을 보장한다. 4만라인 정도의 데이터를 전달하는 과정에서 데이터의 손실 및 변경이 없었다. 이는 TCP 통신 프로토콜이 데이터를 전송할 때 안정성을 보장하기 위하여 오류제어, 흐름제어, 시퀀스제어를 하고 있다.

3. 동시 서비스 요청에 대한 서버의 순차적 서비스

세 개의 클라이언트로부터 서비스 요청을 받아 처리하는 과정을 보면 가장 먼저 접속을 한 클라이언트부터 서비스를 순차적으로 한다. 세 개의 클라이언트로부터 서비스 요청을 받아 data를 전달하였으나 서버의 서비스 처리 순서는 첫 번째 입력 요청을 받은 클라이언트부터 처리하였다.

4. 동시 서비스 요청에 대한 서버의 병렬적 서비스

세 개의 클라이언트로부터 서비스 요청을 받아 처리하는 과정을 보면 가장 먼저 데이터를 보낸 클라이언트부터 서비스를 하지만 동시에 나머지 두 개의 클라이언트도 서비스를 하고 있다. 서버의 서비스 처리 순서는 가장 먼저 데이터를 보낸 클라이언트부터 처리함과 동시에 나머지 두 개의 클라이언트도 서비스하는 과정을 볼 수 있다.

5. 서버/클라이언트의 포트 바인딩

서버에서 필히 바인드를 해야 하지만 클라이언트는 바인드를 하지 않는 것을 원칙으로 한다. 클라이언트에는 바인드 하거나 하지 않거나 상관이 없이 프로그램은 정상적으로 동작을 한다. 그러나 클라이언트에서 바인드를 하지 않는 이유는 여러 개의 클라이언트가 같은 포트로 서버에 접속을 할 경우 첫 번째 클라이언트는 정상적인 동작을 하지만 두 번째 클라이언트부터는 같은 포트를 사용하기 때문에 바인드 에러가 발생하기 때문이다.

6. 바인드 에러의 발생에 관해

서버 프로그램을 실행하면 바인드 에러가 발생할 수도 있고 발생하지 않을 수도 있다. 네트워크의 상태를 확인해 보면 서버가 TIME_WAIT 상태에 놓여 있기 때문이다.

TIME_WAIT는 회선의 종결 절차가 완료되었으나 TCP는 분실되었을지 모르는 느린 세그먼트를 위해 당분간 소켓을 열어 놓은 상태로 유지시킨다. 이 상태는 새로운 연결을 기존의 연결에서 사용된 일련 번호로 다시 사용하는 것을 막는다. 이 현상이 바인드 에러로 나타난다.

7. 큐의 크기에 따른 동시연결 요청의 한계.

서버의 Queue Size를 1로 설정하였으므로 동시 접속자는 2명이 된다. 첫 번째 접속 요청을 한 클라이언트는 바로 서버에 접속이 되고 두 번째 접속 요청을 한 클라이언트는 Queue Size가 1이므로 대기 큐에 들어가서 접속이 이루어진다. 세 번째 접속 요청을 한 클라이언트는 대기 큐가 가득 찼으므로 접속은 허용되지 않는다. 그렇지만 앞의 클라이언트가 서비스되고 나면 남은 대기 큐에 들어가 서비스를 받을 수 있게 된다.

8. 주요 API 함수의 동작 모드

accept(), connect(), send(), recv()는 블럭킹 모드로 동작한다. 서버는 서비스를 하기 위해 클라이언트와 연결이 확립되어야 하며 클라이언트가 데이터를 보낼 때까지 기다려야 한다. 이는 접속요청과 데이터를 전달받기 위함이다.

9. 복수의 통신로를 이용한 통신

하나의 소켓을 생성하여서는 복수의 통신로를 사용할 수가 없다. 이는 하나의 Socket은 하나의 포트에만 바인드 할 수 있고 하나의 프로세서는 하나의 연결만을 가질 수 있기 때문이다.

10. UDP 통신의 기본 구조와 동작, 신뢰성

비연결지향형이므로 connect(), listen()등이 사용되지 않는다. 서버는 socket을 생성하고 바인드후 클라이언트로부터 데이터를 받아 서비스한다. 같은 LAN 상에서 신뢰성을 확인해본 결과 한번 접근할 수 있는 버퍼의 크기보다 적은 양의 데이터는 신뢰성이 있음을 확인할 수 있었고, 동일 LAN 상이지만 버퍼크기를 넘는 양의 데이터는 신뢰성을 얻을 수 없었다. UDP는 모든 데이터를 한번에 접근하기 때문이다.

IV. 결 론

소켓프로그래밍은 socket API를 이용해 정상적인 동작을 하는 것은 아주 쉽다. 그러나 실제 개발과정에서는 정상적인 상황이 아닌 여러 가지 비정상적인 오류에 처하게 됨에 따라 속수 무책일 때가 많다.

따라서 각 플랫폼별로 비정상적인 오류들의 사례연구를 통하여 효율적인 응용시스템 개발의 필요성이 대두되었다고 할 수 있다.

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 소켓프로그래밍 중에서도 가장 대표적이라 할 수 있는 BSD 소켓프로그래밍의 여러 가지 사례들에 대해 각각의 플랫폼별로 사례연구를 통해 소켓프로그래밍을 이용한 네트워크 프로그래밍 응용시스템 개발에 활용하고자 한다.

향후 WIN Socket과 JAVA Socket등의 연구도 해야 할 것이다.

참고문헌

[1] 박준철, "TCP/IP 소켓 프로그래밍(C Version)", 사이텍미디어, 2001.

[2] 김혜영, "다중 쓰레드 프로그래밍 구조에서의 경량 프로세스 관리에 관한 연구", 서울대학교, 1996.

[3] Derek Hamner, Merlin Hughes, Michael

Shoffner, Umesh Bellur, "Java Network Programming 2/E", MANNING, 1999.
[4] Dove Roberts, "DEVELOPING FOR THE INTERNET WITH Winsock 2", 에프원, 1999.
[5] Douglas E. Comer, "Computer Network and Internets" PRENTICE HALL, 1997.