1. 소켓의 정의와 역할

네트워크 소켓은 컴퓨터 간 통신에서 프로세스가 데이터를 주고받는 종착점 역할을 합니다. 주로 인터넷 소켓으로 사용되며, RFC 147은 1971년에 소켓의 초기 정의를 제시한 문서입니다. 주요 내용은 다음과 같습니다:

1. **소켓 정의**
    - 소켓은 네트워크 상에서 정보 전송을 위한 고유 식별자입니다.
    - 32비트 숫자로 구분되며, 짝수 소켓은 수신, 홀수 소켓은 송신을 의미합니다.
    - 소켓은 송수신 프로세스가 위치한 호스트에 의해 구별됩니다.
2. **포트와의 관계**
    - 소켓은 호스트 운영 체제에서 실행되는 프로세스가 여러 포트에 접근할 수 있도록 합니다.
    - 포트는 물리적 또는 논리적 I/O 장치로, 시스템 호출을 통해 운영 체제에서 관리됩니다.
3. **소켓 사용 방법**
    - 소켓은 ARPA 네트워크에서 기계 간 통신을 위한 포트의 식별자로 사용됩니다.
    - 각 호스트에 할당된 소켓은 고유한 프로세스와 연결되며, 일부 소켓은 전 세계적으로 알려진 이름을 가질 수 있습니다.
4. **소켓 명명 및 사용자 투명성**
    - 사용자는 소켓 이름을 알 필요가 없으며, 소켓 명세는 사용자에게 투명할 수 있습니다.
    - 동일한 목적으로 나중에 사용되는 소켓은 이전에 사용된 소켓과 동일해야 할 수 있습니다.
5. **네트워크 사용 계정**
    - 네트워크 제어 프로그램(NCP)은 각 연결을 기록하고, 연결된 시간, 전송된 메시지와 비트 수, 송수신 호스트, 그리고 해당 소켓 정보를 기록해야 합니다.
6. **32비트 소켓 구조**
    - 소켓은 8비트 홈 필드, 16비트 사용자 필드, 8비트 태그 필드로 나뉩니다.
    - 태그는 7비트 플러그와 1비트 편향으로 구성되며, 0은 수신, 1은 송신 소켓을 나타냅니다.

네트워크 상에서 데이터를 교환하기 위해서는 데이터가 어디에서 어디로 이동할지를 알아야 하는데, 소켓은 이러한 "출발지"와 "목적지"를 지정하는 데 쓰입니다. 소켓을 구성하는 두 가지 필수 정보는 **IP 주소**와 **포트 번호**입니다.

- **IP 주소**: 인터넷 상에서 장치(컴퓨터)를 구분하는 고유한 주소.
- **포트 번호**: IP 주소 상에서 특정 애플리케이션을 구분하는 번호. 예를 들어, 웹 브라우저는 주로 `80`(HTTP) 또는 `443`(HTTPS) 포트를 사용합니다.

2. 소켓의 구성 요소

소켓을 정확하게 이해하려면 아래와 같은 주요 요소들을 파악해야 합니다.

1. **프로토콜**: 소켓이 사용하는 통신 규칙을 정의하는 프로토콜입니다. 대표적으로 **TCP**와 **UDP**가 있습니다.
2. **IP 주소**: 데이터를 송수신하는 컴퓨터의 네트워크 주소입니다.

3. **포트 번호**: 같은 컴퓨터 안에서 여러 프로세스가 소켓을 사용할 수 있도록 구분하는 숫자입니다.
4. **소켓 타입**: 연결 방식에 따라 소켓의 타입이 결정됩니다. 대표적으로는 스트림 소켓과 데이터그램 소켓이 있습니다.

3. TCP와 UDP

TCP(Transmission Control Protocol)와 **UDP**(User Datagram Protocol)는 두 가지 주요 프로토콜로, 각각 다르게 동작합니다.

1) TCP (연결 지향형)

- **연결 성립**: TCP는 데이터 전송 전에 서버와 클라이언트 간의 연결을 확립합니다. 이는 흔히 **3-way Handshake**라고 불리는 과정으로 이루어집니다.
- **신뢰성**: TCP는 전송된 데이터가 수신되었는지 확인하며, 손실된 패킷이 있을 경우 재전송을 수행합니다.
- **순서 보장**: 데이터를 전송한 순서대로 수신할 수 있도록 보장합니다.
- **응용**: 신뢰성이 중요한 애플리케이션(웹 브라우징, 이메일 전송 등)에서 사용됩니다.

3-way Handshake

TCP 연결은 클라이언트와 서버가 서로에게 연결을 확인하고 준비 상태임을 확인하는 세 가지 절차를 거칩니다:

1. **SYN**: 클라이언트가 서버에게 연결을 요청합니다.
2. **SYN-ACK**: 서버가 요청을 받았음을 클라이언트에게 알립니다.
3. **ACK**: 클라이언트가 서버의 응답을 확인하고, 연결이 성립됩니다.

2) UDP (비연결형)

- **연결 없음**: UDP는 연결을 성립하지 않고 데이터를 전송하며, 데이터가 정상적으로 수신되었는지 확인하지 않습니다.
- **빠른 전송**: 연결 과정이 없기 때문에 매우 빠른 전송이 가능합니다.
- **신뢰성 부족**: 데이터가 손실되거나 순서가 뒤바뀌는 등의 상황이 발생할 수 있습니다.
- **응용**: 실시간 성능이 중요한 애플리케이션(스트리밍, 온라인 게임 등)에서 사용됩니다.

4. 소켓 통신의 구조

소켓 프로그래밍에서 **서버**와 **클라이언트**의 역할을 구체적으로 설명하면 다음과 같습니다:

1) 서버 측

서버는 보통 특정 IP와 포트 번호에 대해 연결 요청을 기다리고 있습니다. 이 과정을 **리스닝**(Listening)이라고 합니다. 클라이언트가 요청을 보내면 서버는 이를 받아들이고, 데이터를 송수신하게 됩니다.

1. **소켓 생성**: 소켓을 생성하여 통신에 사용할 준비를 합니다.
2. **바인딩**: 생성된 소켓을 특정 IP 주소와 포트 번호에 바인딩합니다. 바인딩은 소켓을 특정 네트워크 인터페이스에 묶는 작업입니다.
3. **리스닝**: 클라이언트가 연결을 요청하기를 기다립니다.
4. **연결 수락**: 클라이언트가 요청하면 그 요청을 수락하고, 새로운 소켓을 생성하여 클라이언트와 통신을 시작합니다.
5. **데이터 송수신**: 클라이언트와 데이터를 주고받습니다.

6. **소켓 종료**: 통신이 끝나면 소켓을 닫습니다.

2) 클라이언트 측

클라이언트는 특정 서버의 IP 주소와 포트 번호를 사용하여 연결을 요청합니다. 연결이 성립되면 서버와 데이터를 송수신합니다.

1. **소켓 생성**: 통신할 소켓을 생성합니다.
2. **서버로 연결 요청**: 서버의 IP 주소와 포트를 지정하여 연결 요청을 보냅니다.
3. **데이터 송수신**: 서버와 데이터를 주고받습니다.
4. **소켓 종료**: 통신이 끝나면 소켓을 닫습니다.

5. 소켓 프로그래밍 심화 (C 언어)

서버 예제 (C 언어)

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>

int main() {
    int server_sock, client_sock;
    struct sockaddr_in server_addr, client_addr;
    socklen_t client_addr_size;
    char message[] = "Hello, Client!";

    // 1. 서버 소켓 생성
    server_sock = socket(PF_INET, SOCK_STREAM, 0);
    if (server_sock == -1) {
        printf("Socket creation failed\n");
        exit(1);
    }

    // 2. 서버 주소 정보 설정
    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = htonl(INADDR_ANY);  // 모든 IP에서 접속 허용
    server_addr.sin_port = htons(12345);  // 포트 번호 설정

    // 3. 서버 소켓을 주소에 바인딩
    if (bind(server_sock, (struct sockaddr*)&server_addr, sizeof(server_addr)) ==
-1) {
        printf("Bind failed\n");
        exit(1);
    }

    // 4. 연결 요청 대기
    if (listen(server_sock, 5) == -1) {
```

```
        printf("Listen failed\n");
        exit(1);
    }

    // 5. 클라이언트 연결 수락
    client_addr_size = sizeof(client_addr);
    client_sock = accept(server_sock, (struct sockaddr*)&client_addr,
&client_addr_size);
    if (client_sock == -1) {
        printf("Accept failed\n");
        exit(1);
    }

    // 6. 클라이언트로 메시지 전송
    write(client_sock, message, sizeof(message));

    // 7. 소켓 종료
    close(client_sock);
    close(server_sock);

    return 0;
}
```

클라이언트 예제 (C 언어)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>

int main() {
    int sock;
    struct sockaddr_in server_addr;
    char message[30];
    int str_len;

    // 1. 클라이언트 소켓 생성
    sock = socket(PF_INET, SOCK_STREAM, 0);
    if (sock == -1) {
        printf("Socket creation failed\n");
        exit(1);
    }

    // 2. 서버 주소 정보 설정
    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");  // 서버 IP 주소
    server_addr.sin_port = htons(12345);  // 서버 포트 번호
```

```
    // 3. 서버로 연결 요청
    if (connect(sock, (struct sockaddr*)&server_addr, sizeof(server_addr)) == -1)
{
        printf("Connection failed\n");
        exit(1);
    }

    // 4. 서버로부터 메시지 수신
    str_len = read(sock, message, sizeof(message) - 1);
    if (str_len == -1) {
        printf("Read failed\n");
        exit(1);
    }

    printf("Message from server: %s\n", message);

    // 5. 소켓 종료
    close(sock);

    return 0;
}
```

6. 에러 처리 및 디버깅

소켓 프로그래밍에서 중요한 부분은 에러 처리입니다. 위 예제들에서 확인할 수 있듯이, 각 단계마다 에러가 발생할 수 있습니다. 특히 소켓 생성, 바인딩, 연결, 데이터 송수신 과정에서 발생하는 에러를 처리하는 것이 중요합니다.

주요 에러

- **소켓 생성 실패**: 자원이 부족하거나 권한이 없을 때 발생합니다.
- **바인딩 실패**: IP 주소나 포트가 이미 사용 중일 때 발생합니다.
- **연결 실패**: 네트워크 문제로 인해 서버와의 연결이 이루어지지 않을 때 발생합니다.

소켓 프로그래밍에서 에러 처리는 대부분 반환값을 통해 처리합니다. 예를 들어, `socket()`, `bind()`, `listen()`, `accept()` 등의 함수가 `-1`을 반환하면 에러가 발생한 것입니다.

결론

소켓 프로그래밍은 네트워크 애플리케이션을 개발하는 데 핵심적인 기술입니다. 소켓은 데이터 전송의 출발지와 도착지를 정의하고, 데이터를 어떻게 주고받을지 결정합니다. TCP와 UDP 같은 프로토콜을 이해하고, 이를 기반으로 서버-클라이언트 모델을 구현하는 것이 기본입니다.