2. C 언어와 리눅스 네트워크 환경 준비

2.1 개발 환경 구성 (GCC, GDB, Wireshark, net-tools)

✅ GCC (GNU Compiler Collection)

📌 개요

GCC는 C, C++, Objective-C, Fortran 등 여러 언어를 컴파일할 수 있는 **표준 GNU 컴파일러**다.
네트워크 소켓 프로그래밍에서 가장 많이 쓰이는 **C 컴파일러**이기도 하다.

🔧 설치

```
sudo apt update
sudo apt install build-essential
```

> `build-essential` 패키지는 `gcc`, `g++`, `make` 등을 포함함

⚙️ 사용법

```
gcc -o server server.c
gcc -g -Wall -o client client.c   # 디버깅 정보 포함
```

옵션	설명
`-o`	출력 파일 이름 지정
`-Wall`	모든 경고 출력
`-g`	디버깅 심볼 포함 (GDB용)
`-pthread`	멀티스레드 프로그램 컴파일 시 필요
`-std=c99`	특정 C 표준 지정 가능

✅ GDB (GNU Debugger)

📌 개요

GDB는 C 언어 프로그램의 **실행 흐름을 단계별로 추적하고 변수 상태를 확인**할 수 있는 GNU 디버거다.
TCP 소켓 프로그래밍에서는 연결 대기 중지, SIGPIPE, read/write 오류 등을 디버깅하는 데 필수적이다.

🔧 설치

```
sudo apt install gdb
```

🛠️ 기본 사용법

```
gcc -g -o myserver myserver.c
gdb ./myserver
```

주요 명령어

명령	설명
`break main`	main 함수에서 중단점 설정
`run`	프로그램 실행
`next`	다음 줄 실행
`step`	함수 내부로 진입
`print var`	변수 값 출력
`backtrace`	스택 프레임 출력
`list`	소스코드 표시
`quit`	종료

✅ Wireshark

📌 개요

Wireshark는 네트워크를 통해 오가는 **패킷을 실시간으로 분석하고 시각화**할 수 있는 강력한 도구다.
TCP/UDP 연결 흐름, 패킷 손실, 포트 상태, 프로토콜 구조 확인에 사용됨.

🔧 설치

```
sudo apt install wireshark
sudo usermod -aG wireshark $USER
```

> 재부팅 후 일반 사용자도 패킷 캡처 가능

🔍 주요 기능

기능	설명
필터	`tcp.port == 8080`, `ip.addr == 192.168.0.10`
Follow TCP stream	연결 전체 내용을 재구성
Protocol decoding	HTTP, DNS, TLS 구조를 계층별로 분석 가능
실시간 캡처	인터페이스 선택 후 `Start` 누르면 실시간 패킷 표시

💉 사용 예

1. `sudo wireshark` 실행
2. `eth0` 선택 후 캡처 시작
3. TCP 연결 필터: `tcp.stream eq 0`
4. HTTP 응답 확인: `http.response`

✅ net-tools (ifconfig, netstat 등)

📌 개요

`net-tools`는 **전통적인 네트워크 진단 및 상태 확인 도구 모음**이다.
요즘은 `ip`, `ss`로 대체되고 있지만 여전히 많이 쓰인다.

🔧 설치

```
sudo apt install net-tools
```

🧰 주요 명령어

명령	설명
`ifconfig`	인터페이스 IP 주소 확인
`netstat -anpt`	모든 포트와 PID 확인
`netstat -i`	패킷 통계 보기
`netstat -r`	라우팅 테이블 확인
`arp -a`	ARP 캐시 조회

📦 개발 환경 통합 팁

도구	통합 목적
GCC + GDB	버그 추적 및 데이터 확인
Wireshark	실제 트래픽 확인 (3-way handshake, FIN 등)
net-tools	시스템 상태 확인 (포트 열림 여부, 인터페이스 동작 등)

✅ 요약 정리

도구	용도	명령 예
GCC	컴파일	`gcc -o app app.c`
GDB	디버깅	`gdb ./app`
Wireshark	패킷 분석	GUI 또는 CLI `tshark`
net-tools	인터페이스, 포트 진단	`ifconfig`, `netstat`

2.2 주요 헤더 파일 및 라이브러리 (`<sys/socket.h>`, `<netinet/in.h>`, `<arpa/inet.h>`, `<unistd.h>`)

✅ 1. `<sys/socket.h>` — 소켓 생성과 설정의 중심

📌 역할

- 소켓 관련 **함수와 상수**, **공통 구조체**들을 정의
- `socket()`, `bind()`, `listen()`, `accept()`, `connect()` 함수 선언

📋 주요 함수

함수	설명
`socket()`	소켓 생성
`bind()`	소켓에 IP/포트 바인딩
`listen()`	연결 요청 대기 상태 진입
`accept()`	클라이언트 요청 수락
`connect()`	서버에 연결 요청 (클라이언트용)
`send()`, `recv()`	데이터 전송/수신
`setsockopt()`	소켓 옵션 설정 (`SO_REUSEADDR` 등)

📋 주요 상수

상수	설명
`AF_INET`	IPv4 주소 체계
`AF_INET6`	IPv6 주소 체계
`SOCK_STREAM`	TCP 연결형
`SOCK_DGRAM`	UDP 비연결형
`SOL_SOCKET`	소켓 옵션 레벨
`SO_REUSEADDR`, `SO_KEEPALIVE`	소켓 옵션 종류

📋 주요 구조체

```
struct sockaddr {
    sa_family_t sa_family;     // 주소 체계 (AF_INET 등)
    char        sa_data[14];   // 주소 + 포트 정보 (실제로는 미사용)
};
```

- 실제 사용은 `sockaddr_in`을 형변환하여 전달

✅ 2. `<netinet/in.h>` — IP와 포트를 위한 구조체 정의

📌 역할

- **IPv4/IPv6 주소 구조체**, 포트 처리, 프로토콜 상수 정의
- 소켓 주소에 **IP/포트 정보 전달** 시 사용

📋 주요 구조체

```
struct sockaddr_in {
    sa_family_t     sin_family;   // AF_INET
    in_port_t       sin_port;     // 포트 (network byte order)
    struct in_addr  sin_addr;     // IP 주소 구조체
    char            sin_zero[8];  // padding
};
```

```
struct in_addr {
    in_addr_t s_addr;   // IP 주소 (network byte order)
};
```

> IPv6의 경우 `struct sockaddr_in6`, `struct in6_addr` 사용

📋 주요 상수

상수	설명
`INADDR_ANY`	0.0.0.0: 모든 인터페이스에서 수신
`INADDR_LOOPBACK`	127.0.0.1
`IPPROTO_TCP`	TCP 프로토콜
`IPPROTO_UDP`	UDP 프로토콜

✅ 3. `<arpa/inet.h>` — IP 주소 변환 전담

📌 역할

- IP 주소와 포트의 **바이트 순서 변환**, **문자열 ↔ 이진 변환** 처리

📋 주요 함수

함수	설명
`inet_pton()`	텍스트 → 바이너리 (presentation → network)
`inet_ntop()`	바이너리 → 텍스트 (network → presentation)
`inet_addr()`	문자열 IP를 `in_addr_t`로 변환 (deprecated)
`inet_ntoa()`	`in_addr` → 문자열 (deprecated)

```
inet_pton(AF_INET, "192.168.0.1", &addr.sin_addr);
inet_ntop(AF_INET, &addr.sin_addr, ip_str, sizeof(ip_str));
```

📋 바이트 순서 변환

함수	역할
`htons()`	호스트 → 네트워크 바이트 (16비트 포트)
`htonl()`	호스트 → 네트워크 바이트 (32비트 IP)
`ntohs()`	네트워크 → 호스트 바이트 (16비트)
`ntohl()`	네트워크 → 호스트 바이트 (32비트)

✅ 4. `<unistd.h>` — POSIX 시스템 호출 선언

📌 역할

- **파일/소켓 디스크립터 관련 함수**, `read()`, `write()`, `close()` 등 선언
- **소켓을 닫을 때 반드시 필요**

📋 주요 함수

함수	설명
`close(fd)`	소켓 또는 파일 디스크립터 종료
`read(fd, buf, n)`	데이터 읽기
`write(fd, buf, n)`	데이터 쓰기
`sleep(n)`	초 단위 슬립
`usleep(us)`	마이크로초 단위 슬립

🧪 전체 예시 코드 스니펫

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

int main() {
    int server_fd = socket(AF_INET, SOCK_STREAM, 0);

    struct sockaddr_in addr;
    addr.sin_family = AF_INET;
    addr.sin_port = htons(8080);
    addr.sin_addr.s_addr = INADDR_ANY;

    bind(server_fd, (struct sockaddr*)&addr, sizeof(addr));
    listen(server_fd, 5);

    struct sockaddr_in client_addr;
    socklen_t len = sizeof(client_addr);
    int client_fd = accept(server_fd, (struct sockaddr*)&client_addr, &len);

    char buffer[1024];
    read(client_fd, buffer, sizeof(buffer));
    write(client_fd, "Hello", 5);

    close(client_fd);
```

```
    close(server_fd);
    return 0;
}
```

✅ 요약 정리

헤더	주요 기능	계층적 역할
`<sys/socket.h>`	소켓 생성 및 통신 함수	소켓 API
`<netinet/in.h>`	IP/Port 구조체 정의	전송/네트워크 계층
`<arpa/inet.h>`	주소 변환, 바이트 순서	네트워크 계층
`<unistd.h>`	소켓 닫기, read/write	POSIX 시스템 API

2.3 `socket`, `bind`, `listen`, `accept`, `connect`, `send`, `recv` 함수 소개

다룰 함수:

```
socket(), bind(), listen(), accept(), connect(), send(), recv()
```

이 함수들은 크게 **서버 측과 클라이언트 측으로 나눠서** 사용하는 구조이다.

✅ [1] socket()

개요

소켓을 생성하는 함수. 네트워크 통신을 위한 **파일 디스크립터**를 만든다고 보면 된다.

함수 원형

```
int socket(int domain, int type, int protocol);
```

매개변수	설명
`domain`	주소 체계: `AF_INET` (IPv4), `AF_INET6` (IPv6)
`type`	통신 방식: `SOCK_STREAM` (TCP), `SOCK_DGRAM` (UDP)
`protocol`	일반적으로 0 (자동 선택)

반환값

- 성공: 파일 디스크립터 (0 이상의 정수)
- 실패: -1

✅ [2] bind()

개요

소켓을 **특정 IP 주소와 포트 번호에 연결**하는 함수 (서버 전용).

함수 원형

```
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

> `addr`은 일반적으로 `struct sockaddr_in` 구조체를 `(struct sockaddr *)`로 캐스팅해서 전달

✅ [3] listen()

개요

TCP 서버 소켓을 **수신 대기 상태로 전환**시킴. 커널에 연결 대기 큐를 생성.

함수 원형

```
int listen(int sockfd, int backlog);
```

파라미터	설명
`sockfd`	socket()으로 생성한 소켓
`backlog`	대기 연결 수 (5~128 사이 추천)

✅ [4] accept()

개요

클라이언트의 연결 요청을 수락하고, **새로운 연결 소켓**을 생성함.

함수 원형

```
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

- `addr`: 접속한 클라이언트의 주소 정보
- `addrlen`: 주소 구조체의 크기

> 반환값은 **클라이언트와의 통신 전용 소켓 디스크립터**

✅ [5] connect()

개요

클라이언트가 서버에 접속 요청을 보냄.

함수 원형

```
int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

> 서버의 `bind()` + `listen()` 상태에서만 성공적으로 연결됨

✅ [6] send(), recv()

개요

TCP 소켓을 통한 데이터 전송/수신 함수. (연결형만 해당)

함수 원형

```
ssize_t send(int sockfd, const void *buf, size_t len, int flags);
ssize_t recv(int sockfd, void *buf, size_t len, int flags);
```

파라미터	설명
`sockfd`	통신 중인 소켓
`buf`	전송/수신할 데이터 버퍼
`len`	버퍼 크기
`flags`	일반적으로 0 (또는 `MSG_DONTWAIT`, `MSG_PEEK` 등)

✅ 전체 흐름 요약

```
// 서버 측
int server_fd = socket(AF_INET, SOCK_STREAM, 0);
bind(server_fd, ...);        // IP/PORT 바인딩
listen(server_fd, 5);        // 연결 대기 상태 진입
int client_fd = accept(server_fd, ...);  // 접속 수락
recv(client_fd, buf, ...);   // 데이터 수신
send(client_fd, buf, ...);   // 데이터 송신

// 클라이언트 측
int client_fd = socket(AF_INET, SOCK_STREAM, 0);
connect(client_fd, ...);     // 서버에 연결
send(client_fd, buf, ...);   // 데이터 송신
recv(client_fd, buf, ...);   // 데이터 수신
```

✅ 오류 처리 팁

함수	실패 시 errno 예시	의미
`bind()`	`EADDRINUSE`	포트 중복
`connect()`	`ECONNREFUSED`	서버가 안 떠 있음
`send()`	`EPIPE`, `ECONNRESET`	연결 종료됨
`recv()`	0	연결 종료됨

✅ 간단한 예제: TCP 클라이언트

```
int sock = socket(AF_INET, SOCK_STREAM, 0);
struct sockaddr_in serv_addr;
serv_addr.sin_family = AF_INET;
serv_addr.sin_port = htons(1234);
inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr);
connect(sock, (struct sockaddr*)&serv_addr, sizeof(serv_addr));
send(sock, "Hello", 5, 0);
recv(sock, buffer, 1024, 0);
close(sock);
```

2.4 네트워크 디버깅 도구: `netstat`, `ss`, `lsof`, `tcpdump`

✅ 1. `netstat` (Network Statistics)

📌 개요

- 리눅스 전통적인 네트워크 정보 도구 (지금은 `ss`가 대체 중)
- 열려 있는 **포트, 연결 상태, 라우팅 테이블, 인터페이스 정보** 확인 가능

📋 사용 예시

```
netstat -an             # 모든 연결 및 포트 정보 (숫자)
netstat -tulnp          # TCP/UDP 리스닝 포트 + 프로세스 정보
netstat -rn             # 라우팅 테이블 출력
```

옵션	설명
`-a`	모든 연결 (listening + established)
`-n`	숫자로 출력 (DNS 조회 생략)
`-t`	TCP
`-u`	UDP

옵션	설명
`-l`	Listening 상태
`-p`	소켓 사용 중인 프로세스 표시
`-r`	라우팅 테이블

> 🔒 권한 없는 유저는 `-p` 옵션으로 프로세스 정보 확인 불가

✅ 2. `ss` (Socket Statistics)

📌 개요

- `netstat`보다 더 빠르고 강력한 네트워크 상태 확인 도구
- **/proc/net** 기반으로 실시간 소켓 정보 출력

📋 사용 예시

```
ss -tuln                    # TCP/UDP listening 포트 확인
ss -s                       # 소켓 통계 요약
ss -p state established     # 연결 완료된 세션과 PID 확인
```

옵션	설명
`-t`	TCP 소켓
`-u`	UDP 소켓
`-l`	Listening 중인 소켓
`-n`	숫자 IP, 포트 출력
`-p`	PID/프로세스 정보 표시
`-s`	통계 요약
`state established`	현재 연결된 상태만 출력

✅ 3. `lsof` (List Open Files)

📌 개요

- **열려 있는 파일/소켓/디스크/포트 등 모든 리소스** 확인 가능
- 특히 포트와 프로세스 관계를 추적할 때 매우 유용

📋 사용 예시

```
lsof -i :8080              # 8080 포트를 점유 중인 프로세스
lsof -i                    # 전체 네트워크 연결 확인
lsof -iTCP -sTCP:LISTEN    # TCP 리스닝 포트만 보기
```

옵션	설명
`-i`	네트워크 파일 (TCP, UDP)
`-sTCP:LISTEN`	리스닝 상태만
`:포트번호`	특정 포트 검색
`-nP`	IP/포트 숫자 출력

> `kill -9 $(lsof -t -i :8080)`로 포트를 점유한 프로세스를 죽일 수도 있어

✅ 4. `tcpdump`

📌 개요

- **로우 레벨 패킷 분석 도구**
- 네트워크 인터페이스에서 실제로 오가는 패킷을 **헤더 단위로 캡처**

📋 기본 구조

```
tcpdump -i 인터페이스 [필터]
```

📋 예시

```
tcpdump -i lo               # 로컬 루프백 인터페이스 감시
tcpdump -i eth0 port 8080   # 8080 포트로 주고받는 패킷만
tcpdump -nn -X port 80      # 숫자 출력 + payload 바이트 덤프
tcpdump -w dump.pcap        # 패킷을 pcap 파일로 저장
tcpdump -r dump.pcap        # 저장한 pcap 파일 다시 보기
```

옵션	설명
`-i`	감시할 인터페이스 지정
`-nn`	DNS, 포트 이름 해석 생략
`-X`	패킷의 payload까지 헥사 + ASCII로 출력
`-w`, `-r`	저장 및 읽기
필터 예시	`port 80`, `tcp`, `udp`, `host 192.168.0.1`, `src`, `dst` 등

> `wireshark`과 함께 `.pcap` 파일을 분석하면 GUI 기반 분석도 가능해

🧪 실전 활용 흐름

문제	사용할 도구	활용 예
특정 포트 누가 쓰고 있는지 확인	`lsof`, `ss`, `netstat`	`lsof -i :포트`
서버에 연결된 클라이언트 확인	`ss`, `netstat`	`ss -t state established`
패킷이 오가는지 실시간 확인	`tcpdump`	`tcpdump -i eth0 port 80`
시스템 콜 수준 추적	`strace`, `lsof`	`strace -p PID`

✅ 요약 정리

도구	주요 기능
`netstat`	연결 상태, 포트, 라우팅 정보 확인 (legacy)
`ss`	빠르고 강력한 실시간 소켓 분석
`lsof`	열린 파일/소켓과 프로세스 매핑 추적
`tcpdump`	로우 레벨 패킷 분석, 필터링, 저장