10. IPv6 네트워크 프로그래밍

10.1 IPv6 주소 구조 이해

✅ IPv6 도입 배경

IPv6는 기존 IPv4의 주소 고갈 문제를 해결하기 위해 등장한 차세대 인터넷 프로토콜이다.
IPv4가 약 43억 개의 주소(2^{32})를 제공하는 반면, IPv6는 **2^{128}**개의 주소 공간을 제공하여, 사실상 무한에 가까운 고유 주소를 할당할 수 있게 한다.

✅ IPv6 주소 기본 구조

- 총 **128비트** 길이
- **16비트씩 8개의 블록**으로 구분
- **16진수로 표현**하며, 각 블록은 `:`로 구분

```
2001:0db8:0000:0042:0000:8a2e:0370:7334
```

✅ 축약 규칙

1. 앞의 0 생략 가능

```
2001:0db8:0000:0000:0000:0000:0000:0001
→ 2001:db8:0:0:0:0:0:1
```

2. 연속된 0은 "::"으로 한 번만 축약 가능

```
2001:db8::1
(중간 6개의 블록이 0일 때)
```

> `::`는 IPv6 주소에서 단 한 번만 사용할 수 있다.

✅ 주소 유형 (Prefix 기준)

주소 유형	Prefix	설명
유니캐스트	`2000::/3`	단일 호스트 대상
링크로컬	`fe80::/10`	같은 링크 내 통신 (라우터 경유 X)
멀티캐스트	`ff00::/8`	다수 수신자 대상 브로드캐스트
루프백	`::1`	자기 자신 (localhost)

주소 유형	Prefix	설명
언지정 주소	`::`	초기화되지 않은 주소

✅ IPv6 주소 구성 요소

IPv6 주소는 대개 다음과 같이 나뉜다:

구간	설명
Global Routing Prefix	상위 ISP 또는 라우팅 도메인 식별 (ex. `2001:db8::`)
Subnet ID	네트워크 내 서브넷 구분
Interface ID	호스트를 고유하게 식별 (MAC → EUI-64)

✅ Interface Identifier (64비트)

- 일반적으로 **MAC 주소 기반으로 생성** (EUI-64 방식)
- 예: `00:1A:2B:3C:4D:5E` → `021a:2bff:fe3c:4d5e`
    - MAC 주소 48비트 → 중간에 `fffe` 삽입, 앞 비트 조정

✅ IPv6 주소 예시

```
Full : 2001:0db8:0000:0000:0000:ff00:0042:8329
Short: 2001:db8::ff00:42:8329
```

✅ IPv6 vs IPv4 주요 비교

항목	IPv4	IPv6
주소 길이	32비트	128비트
표기법	10진수 4옥텟 (`.` 구분)	16진수 8블록 (`:` 구분)
NAT 필요 여부	필수	필요 없음 (고유 주소 사용)
브로드캐스트	지원	지원하지 않음 (멀티캐스트 사용)
보안	선택적(IPSec)	필수 (기본 포함)

✅ 실전 사용 시 유의점

- IPv6 주소는 주소 자체에 **라우팅/서브넷 정보가 포함**되어 있음
- **ARP를 대신해 NDP(Neighbor Discovery Protocol)** 사용
- IPv4와는 다른 **이중 스택(Dual Stack)** 환경 필요 (IPv4 ↔ IPv6 병행)

10.2 `sockaddr_in6`, `inet_pton`, `inet_ntop` 사용법

✅ 1. `sockaddr_in6` 구조체

IPv6 주소를 표현하기 위해 `struct sockaddr_in6`을 사용한다. 이는 `AF_INET6` 주소 체계에 대응하며 IPv6 통신 시 `bind()`, `connect()`, `accept()` 등의 소켓 함수에서 활용된다.

```
#include <netinet/in.h>

struct sockaddr_in6 {
    sa_family_t     sin6_family;   // 주소 체계: AF_INET6
    in_port_t       sin6_port;     // 포트 번호 (네트워크 바이트 오더)
    uint32_t        sin6_flowinfo; // 플로우 정보 (일반적으로 0)
    struct in6_addr sin6_addr;     // IPv6 주소
    uint32_t        sin6_scope_id; // 링크 로컬 주소의 인터페이스 식별자
};
```

- `sin6_family`: `AF_INET6`로 설정
- `sin6_port`: `htons(port)`로 설정
- `sin6_addr`: `inet_pton()`을 통해 설정
- `sin6_scope_id`: 링크-로컬 주소일 경우 인터페이스 지정 필요

✅ 2. `inet_pton()` 함수

IPv6 주소를 **문자열 → 바이너리**로 변환

```
#include <arpa/inet.h>

int inet_pton(int af, const char *src, void *dst);
```

매개변수	설명
`af`	주소 체계 (`AF_INET`, `AF_INET6`)
`src`	문자열 형태의 주소 (예: `"::1"`)
`dst`	`struct in6_addr` 포인터 (바이너리 형태로 저장됨)

```
struct sockaddr_in6 addr6;
inet_pton(AF_INET6, "2001:db8::1", &addr6.sin6_addr);
```

✅ 3. `inet_ntop()` 함수

IPv6 주소를 **바이너리 → 문자열**로 변환

```
const char *inet_ntop(int af, const void *src, char *dst, socklen_t size);
```

매개변수	설명
`af`	주소 체계 (`AF_INET`, `AF_INET6`)
`src`	`struct in6_addr` 포인터
`dst`	출력 문자열 버퍼
`size`	버퍼 크기 (보통 `INET6_ADDRSTRLEN`)

```
char ipstr[INET6_ADDRSTRLEN];
inet_ntop(AF_INET6, &addr6.sin6_addr, ipstr, sizeof(ipstr));
printf("IPv6 주소: %s\n", ipstr);
```

✅ 예제 코드 (IPv6 주소 설정 및 출력)

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

int main() {
    struct sockaddr_in6 addr6;
    char ip_str[INET6_ADDRSTRLEN] = "2001:db8::1";
    char ip_output[INET6_ADDRSTRLEN];

    memset(&addr6, 0, sizeof(addr6));
    addr6.sin6_family = AF_INET6;
    addr6.sin6_port = htons(8080);

    if (inet_pton(AF_INET6, ip_str, &addr6.sin6_addr) <= 0) {
        perror("inet_pton 실패");
        return 1;
    }

    if (inet_ntop(AF_INET6, &addr6.sin6_addr, ip_output, sizeof(ip_output)) == NULL) {
        perror("inet_ntop 실패");
        return 1;
    }

    printf("IPv6 주소 출력: %s\n", ip_output);
    return 0;
```

```
}
```

출력 예시:

```
IPv6 주소 출력: 2001:db8::1
```

✅ 주의사항

항목	설명
`inet_pton` 실패 시 `<= 0` 반환	입력 주소 형식이 잘못된 경우 포함
`inet_ntop` 실패 시 `NULL` 반환	메모리 오류 혹은 포인터 문제
`sin6_scope_id`	`fe80::` 등 링크로컬 주소에서는 설정 필수 (예: `en0`, `eth0`의 ifindex 사용)

10.3 IPv6 서버/클라이언트 구현

IPv6 기반의 소켓 통신은 기존 IPv4와 매우 유사하지만, 주소 체계(`AF_INET6`), 구조체(`sockaddr_in6`), 포맷 함수(`inet_pton`, `inet_ntop`) 등이 IPv6에 맞게 변경되어야 한다.

✅ 서버 측 구현 예제

```c
// ipv6_server.c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <netinet/in.h>

#define PORT 8080
#define BUFFER_SIZE 1024

int main() {
    int server_fd, client_fd;
    struct sockaddr_in6 server_addr, client_addr;
    char buffer[BUFFER_SIZE];
    socklen_t client_len = sizeof(client_addr);

    server_fd = socket(AF_INET6, SOCK_STREAM, 0);
    if (server_fd < 0) {
        perror("socket 실패");
        exit(EXIT_FAILURE);
    }

    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin6_family = AF_INET6;
    server_addr.sin6_port = htons(PORT);
```

```
    server_addr.sin6_addr = in6addr_any;

    if (bind(server_fd, (struct sockaddr*)&server_addr, sizeof(server_addr)) < 0) {
        perror("bind 실패");
        exit(EXIT_FAILURE);
    }

    listen(server_fd, 5);
    printf("[서버] 대기 중 (IPv6, 포트 %d)...\n", PORT);

    client_fd = accept(server_fd, (struct sockaddr*)&client_addr, &client_len);
    if (client_fd < 0) {
        perror("accept 실패");
        exit(EXIT_FAILURE);
    }

    char client_ip[INET6_ADDRSTRLEN];
    inet_ntop(AF_INET6, &client_addr.sin6_addr, client_ip, sizeof(client_ip));
    printf("[서버] 연결됨: %s\n", client_ip);

    ssize_t len = read(client_fd, buffer, sizeof(buffer) - 1);
    if (len > 0) {
        buffer[len] = '\0';
        printf("[서버] 받은 메시지: %s\n", buffer);
    }

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

✅ 클라이언트 측 구현 예제

```
// ipv6_client.c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <netinet/in.h>

#define SERVER_ADDR "2001:db8::1"  // 또는 ::1 (localhost)
#define PORT 8080

int main() {
    int sock;
    struct sockaddr_in6 server_addr;
    char *message = "안녕하세요, 서버!";

    sock = socket(AF_INET6, SOCK_STREAM, 0);
    if (sock < 0) {
```

```
        perror("socket 실패");
        exit(EXIT_FAILURE);
    }

    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin6_family = AF_INET6;
    server_addr.sin6_port = htons(PORT);
    if (inet_pton(AF_INET6, SERVER_ADDR, &server_addr.sin6_addr) <= 0) {
        perror("inet_pton 실패");
        exit(EXIT_FAILURE);
    }

    if (connect(sock, (struct sockaddr*)&server_addr, sizeof(server_addr)) < 0) {
        perror("connect 실패");
        exit(EXIT_FAILURE);
    }

    write(sock, message, strlen(message));
    printf("[클라이언트] 메시지 전송 완료\n");

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

✅ 주요 차이점 요약

항목	IPv4 (`AF_INET`)	IPv6 (`AF_INET6`)
주소 구조체	`struct sockaddr_in`	`struct sockaddr_in6`
주소 필드	`sin_addr.s_addr`	`sin6_addr`
포트 필드	`sin_port`	`sin6_port`
주소 해석 함수	`inet_pton`, `inet_ntop`	동일하게 사용 (`AF_INET6`)
바인딩 주소	`INADDR_ANY`	`in6addr_any` (`::`)
특이사항	없음	링크-로컬 시 `sin6_scope_id` 설정 필요

✅ 실전 주의사항

- **IPv6만 사용 시** 클라이언트/서버 모두 IPv6 소켓을 사용해야 연결 가능
- **듀얼 스택** 사용 시 IPv6 소켓이 IPv4를 수용하도록 설정할 수 있지만 OS 설정 필요
- 링크 로컬 주소(`fe80::`)를 사용할 경우 반드시 `scope_id`를 지정해야 한다 (예: `sin6_scope_id = if_nametoindex("eth0")`)

10.4 듀얼 스택 서버 구현

듀얼 스택 서버란 하나의 서버에서 **IPv4와 IPv6 요청을 모두 수신**할 수 있는 구조를 의미한다.
운영체제와 소켓 설정에 따라 다음 두 가지 방식이 있다.

✅ 1. 방법 개요

방식	설명
① 단일 IPv6 소켓	IPv6 소켓 하나로 IPv4도 처리 (OS가 매핑 지원)
② IPv4 + IPv6 각각 생성	각각 소켓을 만들어 동시에 `listen()`

✅ 2. 기본 전제

- 소켓 생성 시 `AF_INET6` 사용
- `setsockopt()`으로 `IPV6_V6ONLY` 옵션을 적절히 설정
- 운영체제에 따라 `dual-stack socket` 지원 여부 확인 필요 (Linux 기본은 **off**)

✅ 방법 1: IPv6 소켓 하나로 IPv4도 처리

```c
// dual_stack_server.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <netinet/in.h>

#define PORT 8080

int main() {
    int server_fd;
    struct sockaddr_in6 addr;
    int opt = 0;

    server_fd = socket(AF_INET6, SOCK_STREAM, 0);
    if (server_fd < 0) {
        perror("socket");
        exit(1);
    }

    // 중요한 부분: IPV6_V6ONLY = 0 → IPv4도 허용
    opt = 0;
    if (setsockopt(server_fd, IPPROTO_IPV6, IPV6_V6ONLY, &opt, sizeof(opt)) < 0) {
        perror("setsockopt IPV6_V6ONLY");
        exit(1);
```

```
    }

    memset(&addr, 0, sizeof(addr));
    addr.sin6_family = AF_INET6;
    addr.sin6_port = htons(PORT);
    addr.sin6_addr = in6addr_any;

    if (bind(server_fd, (struct sockaddr*)&addr, sizeof(addr)) < 0) {
        perror("bind");
        exit(1);
    }

    listen(server_fd, 5);
    printf("듀얼 스택 서버 시작 (IPv4 + IPv6 수신)\n");

    while (1) {
        struct sockaddr_storage client_addr;
        socklen_t client_len = sizeof(client_addr);
        int client_fd = accept(server_fd, (struct sockaddr*)&client_addr, &client_len);

        char ipstr[INET6_ADDRSTRLEN];
        void *addr_ptr;

        if (client_addr.ss_family == AF_INET) {
            struct sockaddr_in *v4 = (struct sockaddr_in*)&client_addr;
            addr_ptr = &(v4->sin_addr);
        } else {
            struct sockaddr_in6 *v6 = (struct sockaddr_in6*)&client_addr;
            addr_ptr = &(v6->sin6_addr);
        }

        inet_ntop(client_addr.ss_family, addr_ptr, ipstr, sizeof(ipstr));
        printf("클라이언트 접속: %s\n", ipstr);

        close(client_fd);
    }

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

✅ 방법 2: IPv4와 IPv6 소켓을 각각 생성

```
// 핵심 개념만 정리

int ipv4_fd = socket(AF_INET, SOCK_STREAM, 0);
int ipv6_fd = socket(AF_INET6, SOCK_STREAM, 0);

// 각자 bind(), listen() → select/epoll로 처리
```

- 두 소켓 모두 `bind()` 하고

- `select()`, `poll()`, `epoll()`로 **동시에 모니터링**
- 구현 복잡도는 증가하지만 OS 제한 없이 확장성 높음

✅ Linux 동작 특징

- 기본적으로 **IPv6 소켓이 IPv4를 수신하지 못함**
- `IPV6_V6ONLY` 옵션을 **0으로 명시적으로 설정**해야 듀얼 스택 동작 가능
- `/proc/sys/net/ipv6/bindv6only` 값이 0일 경우, 시스템 전역 설정으로 dual-stack 허용

📌 비교 요약

항목	단일 소켓 방식	이중 소켓 방식
OS 설정 의존	있음 (Linux: 기본 비허용)	없음
구현 난이도	낮음	높음
성능/유연성	보통	우수
권장 사용 시점	개인 서버, 테스트 용도	실무, 고성능, 멀티포트 환경

✳️ 참고

- Windows는 기본적으로 dual-stack을 지원하며, `IPV6_V6ONLY`가 기본 `false`
- IPv4 주소가 `::ffff:127.0.0.1` 형식으로 IPv6 주소로 맵핑됨 (IPv4-mapped IPv6 address)