5. 소켓 관련 주요 옵션 및 설정

5.1 `setsockopt`, `getsockopt`

✅ `setsockopt()` 함수

소켓의 **옵션 값을 설정**할 때 사용됨.

```
int setsockopt(int sockfd, int level, int optname,
               const void *optval, socklen_t optlen);
```

📌 매개변수 설명

인자	의미
`sockfd`	대상 소켓 파일 디스크립터
`level`	옵션 레벨 (`SOL_SOCKET`, `IPPROTO_TCP` 등)
`optname`	설정할 옵션 이름
`optval`	설정할 값 포인터
`optlen`	값의 크기 (ex. `sizeof(int)`)

✅ `getsockopt()` 함수

소켓의 **옵션 값을 조회**할 때 사용됨.

```
int getsockopt(int sockfd, int level, int optname,
               void *optval, socklen_t *optlen);
```

🎯 자주 사용하는 옵션

Level	Option	의미
`SOL_SOCKET`	`SO_REUSEADDR`	TIME_WAIT 중인 포트를 즉시 재사용
`SOL_SOCKET`	`SO_RCVBUF`, `SO_SNDBUF`	수신/송신 버퍼 크기 설정
`SOL_SOCKET`	`SO_KEEPALIVE`	TCP 연결 유효성 주기 확인
`SOL_SOCKET`	`SO_RCVTIMEO`, `SO_SNDTIMEO`	수신/송신 타임아웃
`IPPROTO_TCP`	`TCP_NODELAY`	Nagle 알고리즘 비활성화 (지연 없이 송신)

🧪 실습 예제 1: `SO_REUSEADDR` 설정

```
int opt = 1;
setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt));
```

- 바인딩 오류 방지
- 서버 재시작 시 `"Address already in use"` 에러 예방

🧪 실습 예제 2: `SO_RCVBUF` 크기 조정

```
int bufsize = 8192;
setsockopt(sock, SOL_SOCKET, SO_RCVBUF, &bufsize, sizeof(bufsize));
```

조회할 땐:

```
int cur_bufsize;
socklen_t len = sizeof(cur_bufsize);
getsockopt(sock, SOL_SOCKET, SO_RCVBUF, &cur_bufsize, &len);
printf("현재 수신 버퍼 크기: %d 바이트\n", cur_bufsize);
```

🧪 실습 예제 3: 타임아웃 설정 (`SO_RCVTIMEO`)

```
struct timeval tv;
tv.tv_sec = 5;
tv.tv_usec = 0;

setsockopt(sock, SOL_SOCKET, SO_RCVTIMEO, &tv, sizeof(tv));
```

- `recv()`에서 5초 이내에 수신이 없으면 `-1` 리턴 + `errno == EAGAIN`

🧪 실습 예제 4: TCP_NODELAY (지연 없이 전송)

```
int flag = 1;
setsockopt(sock, IPPROTO_TCP, TCP_NODELAY, &flag, sizeof(flag));
```

- Nagle 알고리즘을 끄면 작은 패킷도 바로 전송됨 (레이턴시 감소)

⚠️ 에러 처리

```
if (setsockopt(...) < 0) {
    perror("setsockopt");
    exit(1);
}
```

🧠 요약

함수	용도
`setsockopt()`	소켓 동작을 조정할 수 있는 **설정** 함수
`getsockopt()`	현재 소켓의 옵션 값을 **조회**할 때 사용
`SOL_SOCKET`	대부분의 일반적인 옵션이 정의된 레벨
`IPPROTO_TCP`	TCP 전용 설정은 이 레벨에서 다룸

5.2 `SO_REUSEADDR`, `SO_REUSEPORT`

✅ 1. `SO_REUSEADDR`

🔎 의미

- 포트를 **TIME_WAIT 상태에서도 즉시 재사용 가능**하게 해줌.
- 서버 재시작 시 `"Address already in use"` 에러 방지에 자주 사용.

⚙️ 사용 예시

```
int opt = 1;
setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt));
```

📌 효과

- 이전 연결이 **TIME_WAIT**에 있어도 bind 가능.
- 단, **다른 프로세스와 동시에 같은 포트 사용**은 불가.

✅ 2. `SO_REUSEPORT`

🔎 의미

- **여러 개의 프로세스가 동시에 동일한 포트에 바인딩** 가능하게 함.
- 커널이 자동으로 **로드 밸런싱**해줌 (Linux 3.9+ 지원).

⚙️ 사용 예시

```
int opt = 1;
setsockopt(sockfd, SOL_SOCKET, SO_REUSEPORT, &opt, sizeof(opt));
```

📌 특징

항목	내용
커널 분산 방식	연결 요청을 프로세스 간에 분산 처리
활용 예시	Nginx 다중 워커 프로세스
조건	**모든 프로세스가 동시에** `SO_REUSEPORT` **설정**해야 동작

🧪 비교 요약

옵션	포트 재사용	다중 프로세스 바인딩	TIME_WAIT 처리	사용 시기
`SO_REUSEADDR`	O	X	O	서버 재시작 대비
`SO_REUSEPORT`	O	O	O	고성능 멀티 프로세스 서버 구현

🧠 참고: `bind()` 이전에 반드시 설정해야 함

```
int sockfd = socket(AF_INET, SOCK_STREAM, 0);
int opt = 1;
setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt));
setsockopt(sockfd, SOL_SOCKET, SO_REUSEPORT, &opt, sizeof(opt));  // 리눅스 3.9 이상에서만 가능

struct sockaddr_in addr = { ... };
bind(sockfd, (struct sockaddr*)&addr, sizeof(addr));
```

🧱 실습 환경 주의

- `SO_REUSEPORT` 는 Linux 3.9+에서만 지원됨.
  → 확인 명령: `uname -r`
- `setsockopt()` 는 `bind()` 보다 **먼저 호출**해야 효과 있음

5.3 소켓 타임아웃 설정 (`SO_RCVTIMEO`, `SO_SNDTIMEO`)

✅ 개요

옵션	의미
`SO_RCVTIMEO`	`recv()`, `recvfrom()` 함수의 **읽기 타임아웃** 설정
`SO_SNDTIMEO`	`send()`, `sendto()` 함수의 **쓰기 타임아웃** 설정

둘 다 **blocking 소켓 함수의 대기 시간을 제한**하는 데 사용된다.
설정한 시간이 초과되면 해당 함수는 `-1`을 리턴하고 `errno == EAGAIN` 또는 `EWOULDBLOCK`이 됨.

🔧 사용 방법

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

int sockfd = socket(AF_INET, SOCK_STREAM, 0);

// 타임아웃 시간 설정 (3초)
struct timeval timeout;
timeout.tv_sec = 3;
timeout.tv_usec = 0;

// 수신 타임아웃
setsockopt(sockfd, SOL_SOCKET, SO_RCVTIMEO, &timeout, sizeof(timeout));

// 송신 타임아웃
setsockopt(sockfd, SOL_SOCKET, SO_SNDTIMEO, &timeout, sizeof(timeout));
```

🔍 동작 예시

```c
char buf[1024];
int n = recv(sockfd, buf, sizeof(buf), 0);

if (n < 0) {
    if (errno == EWOULDBLOCK || errno == EAGAIN) {
        printf("수신 타임아웃 발생\n");
    } else {
        perror("recv 오류");
    }
}
```

🧠 참고사항

항목	설명
유효 범위	`setsockopt()`를 호출한 **해당 소켓에만 적용**됨
비동기와 차이	이건 여전히 **blocking 소켓**이지만, **시간 제한이 있는 blocking**
주의	일부 시스템에서는 `tv_usec`을 무시하거나 근사치로 처리함

⏱️ 타임아웃과 비슷한 다른 설정과의 차이

방법	설명
`SO_RCVTIMEO`, `SO_SNDTIMEO`	제한 시간 동안 blocking 시도
`O_NONBLOCK` (비동기)	즉시 반환, 대기하지 않음
`select()`, `poll()`, `epoll()`	다중 소켓을 대상으로 타임아웃 관리

📌 언제 쓰나?

- **응답 없는 서버나 느린 클라이언트**로부터의 대기 시간 제한
- **일부 네트워크 환경에서 연결 시도나 전송 지연**을 방지
- **심플한 단일 소켓 프로그램**에 유용

5.4 TCP_NODELAY, 버퍼 크기 조정

✅ `TCP_NODELAY` 옵션

🔹 목적

- **Nagle 알고리즘**을 비활성화함
- **작은 패킷 전송 지연 없이 즉시 전송**하고 싶을 때 사용

🔸 Nagle 알고리즘이란?

- 여러 개의 작은 TCP 패킷을 하나로 묶어서 전송하여 **네트워크 혼잡을 줄이는 알고리즘**
- 그러나 실시간 응답이 중요한 애플리케이션(채팅, 게임, RPC 등)에서는 **오히려 지연**을 유발함

🔧 사용 예시

```
int flag = 1;
setsockopt(sockfd, IPPROTO_TCP, TCP_NODELAY, (char *)&flag, sizeof(int));
```

✅ 버퍼 크기 조정

🔹 기본 설명

- 소켓은 내부적으로 **송신 버퍼**와 **수신 버퍼**를 갖고 있고,
- 이 크기를 **운영체제 커널에서 동적으로 조절하거나 수동 조정**할 수 있다.

🔧 설정 방법

```
int buf_size = 65536; // 64KB

// 송신 버퍼 크기 조정
setsockopt(sockfd, SOL_SOCKET, SO_SNDBUF, &buf_size, sizeof(buf_size));

// 수신 버퍼 크기 조정
setsockopt(sockfd, SOL_SOCKET, SO_RCVBUF, &buf_size, sizeof(buf_size));
```

> ⚠️ 실제 설정된 값은 커널의 최소/최대 값 제한에 따라 변경될 수 있음.

🔍 확인 방법

```
int actual_size;
socklen_t optlen = sizeof(actual_size);

// 수신 버퍼 크기 조회
getsockopt(sockfd, SOL_SOCKET, SO_RCVBUF, &actual_size, &optlen);
printf("수신 버퍼 크기: %d bytes\n", actual_size);
```

> 리눅스에서는 이 값이 실제 요청한 값의 **2배**로 반환되는 경우가 있음. (헤더 공간 포함)

🧠 참고 비교

옵션	목적	권장 사용 사례
`TCP_NODELAY`	지연 없이 바로 전송	채팅, 게임, 실시간 제어
`SO_SNDBUF`	송신 큐 확보	대용량 전송 시 전송 블로킹 방지
`SO_RCVBUF`	수신 큐 확보	높은 처리량 수신기, 패킷 손실 줄이기

📌 실무 적용 예시

상황	적용
게임/실시간 제어	`TCP_NODELAY = 1`
동영상 스트리밍	`SO_SNDBUF`, `SO_RCVBUF` 크게 조정
서버가 수많은 동시 접속을 수용할 때	송신/수신 버퍼를 상황에 맞게 튜닝

5.5 `fcntl`을 이용한 소켓 비동기 설정

✅ `fcntl` 함수란?

`fcntl`(file control)은 파일 디스크립터의 **속성 설정/제어**를 위한 시스템 콜이다.
소켓은 파일 디스크립터 기반으로 동작하므로 `fcntl`로 비동기 모드 설정이 가능하다.

✅ Non-blocking 모드란?

- 기본적으로 소켓 함수(`read`, `recv`, `accept` 등)는 **blocking 모드**로 동작함.
- **비동기 모드**에서는 I/O 작업이 바로 완료되지 않아도 **즉시 반환**됨.
  예: `accept()` → 대기 클라이언트 없으면 `-1` 반환하고 `errno == EAGAIN`.

✅ `fcntl`로 non-blocking 설정하기

🔸 사용법

```
#include <fcntl.h>
#include <unistd.h>

int flags = fcntl(sockfd, F_GETFL, 0);          // 현재 플래그 읽기
fcntl(sockfd, F_SETFL, flags | O_NONBLOCK);     // 비동기 플래그 설정
```

> `O_NONBLOCK`은 플래그를 **덮어쓰는 게 아니라 추가**해야 함. (`|` 연산 사용)

🔹 비동기 소켓 해제 (Blocking 모드 복원)

```
fcntl(sockfd, F_SETFL, flags & ~O_NONBLOCK);
```

✅ 적용 예시

예: `accept()`가 블로킹되지 않도록 하기

```
int listen_fd = socket(...);
fcntl(listen_fd, F_SETFL, fcntl(listen_fd, F_GETFL, 0) | O_NONBLOCK);

while (1) {
    int client_fd = accept(listen_fd, NULL, NULL);
    if (client_fd == -1) {
        if (errno == EAGAIN || errno == EWOULDBLOCK) {
            // 대기 중인 클라이언트 없음 → sleep 또는 continue
            continue;
        } else {
            perror("accept");
            break;
        }
```

```
    }

    // 정상적인 클라이언트 연결 처리
}
```

✅ 비동기 처리가 필요한 이유

상황	설명
이벤트 기반 서버 (`select`, `poll`, `epoll`)	모든 I/O를 non-blocking으로 설정해야 이벤트 루프가 멈추지 않음
GUI 앱, 로봇 등 실시간성 요구 시스템	I/O 대기 중 블로킹되면 전체 시스템 응답 지연
수천 개의 동시 접속 처리 서버	블로킹 모드로는 연결 수 제한 및 성능 저하 발생

📌 관련 참고

함수	설명
`fcntl(fd, F_GETFL)`	파일 상태 플래그 읽기
`fcntl(fd, F_SETFL, flags)`	플래그 설정
`O_NONBLOCK`	비동기 설정 플래그