

네트워크 프로그래밍

10. 소켓의 다양한 옵션

2

소켓의 옵션과 입출력 버퍼의 크기

소켓 옵션

3

- 소켓 옵션(socket options)
 - ▣ 소켓의 기본 동작을 변경
 - ▣ 소켓 코드와 프로토콜 구현 코드에 대한 세부적인 제어 가능

- Protocol Level
 - ▣ SOL_SOCKET
 - 프로토콜과 무관한 소켓 그 자체
 - ▣ IPPROTO_TCP
 - TCP에 관련된 옵션
 - ▣ IPPROTO_IP
 - IP에 관련된 옵션

Level	Optname	Get	Set	Datatype
SOL_SOCKET	SO_SNDBUF	O	O	int
	SO_RCVBUF	O	O	int
	SO_REUSEADDR	O	O	int
	SO_KEEPALIVE	O	O	int
	SO_BROADCAST	O	O	int
	SO_DONTROUTE	O	O	int
	SO_OOBINLINE	O	O	int
	SO_ERROR	O	X	int
	SO_TYPE	O	X	int
IPPROTO_IP	IP_TOS	O	O	int
	IP_TTL	O	O	int
	IP_MULTICAST_TTL	O	O	unsigned char
	IP_MULTICAST_LOOP	O	O	unsigned char
	IP_MULTICAST_IF	O	O	in_addr{}
IPPROTO_TCP	TCP_KEEPALIVE	O	O	int
	TCP_NODELAY	O	O	int
	TCP_MAXSEG	O	O	int

옵션에 따라서 여러가지 타입이 올 수 있으므로
void로 써놓은것임

옵션정보의 참조에 사용되는 함수

5

```
#include <sys/socket.h>
```

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

→ 성공 시 0, 실패 시 -1 반환

보통 optval을 sizeof해서 넘겨주는것이 좋음

- sock 옵션확인을 위한 소켓의 파일 디스크립터 전달.
- level 확인할 옵션의 프로토콜 레벨 전달.
- optname 확인할 옵션의 이름 전달.
- optval 확인결과를 저장할 버퍼의 주소 값 전달.
- optlen 네 번째 매개변수 optval로 전달된 주소 값의 버퍼크기를 담고 있는 변수의 주소 값 전달, 함수호출이 완료되면 이 변수에는 네 번째 인자를 통해 반환된 옵션정보의 크기가 바이트 단위로 계산되어 저장된다.

옵션정보의 설정에 사용되는 함수

6

```
#include <sys/socket.h>
```

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

➔ 성공 시 0, 실패 시 -1 반환

- sock 옵션변경을 위한 소켓의 파일 디스크립터 전달.
- level 변경할 옵션의 프로토콜 레벨 전달.
- optname 변경할 옵션의 이름 전달.
- optval 변경할 옵션정보를 저장한 버퍼의 주소 값 전달.
- optlen 네 번째 매개변수 optval로 전달된 옵션정보의 바이트 단위 크기 전달.

소켓 타입 정보의 확인

7

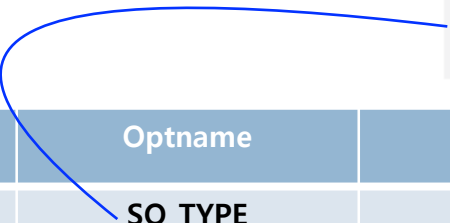
```
tcp_sock=socket(PF_INET, SOCK_STREAM, 0);
udp_sock=socket(PF_INET, SOCK_DGRAM, 0);
printf("SOCK_STREAM: %d \n", SOCK_STREAM);
printf("SOCK_DGRAM: %d \n", SOCK_DGRAM);

state=getsockopt(tcp_sock, SOL_SOCKET, SO_TYPE, (void*)&sock_type, &optlen);
if(state)
    error_handling("getsockopt() error!");
printf("Socket type one: %d \n", sock_type);

state=getsockopt(udp_sock, SOL_SOCKET, SO_TYPE, (void*)&sock_type, &optlen);
if(state)
    error_handling("getsockopt() error!");
printf("Socket type two: %d \n", sock_type);
```

sock_type.c의 일부

```
root@my_linux:/tcip# gcc sock_type.c -o socktype
root@my_linux:/tcip# ./socktype
SOCK_STREAM: 1
SOCK_DGRAM: 2
Socket type one: 1
Socket type two: 2
```



Level	Optname	Get	Set	Datatype
SOL_SOCKET	SO_TYPE	O	X	int

소켓의 입출력 버퍼 크기 확인

8

```
sock=socket(PF_INET, SOCK_STREAM, 0);
len=sizeof(snd_buf);
state=getsockopt(sock, SOL_SOCKET, SO_SNDBUF, (void*)&snd_buf, &len);
if(state)
    error_handling("getsockopt() error");

len=sizeof(rcv_buf);
state=getsockopt(sock, SOL_SOCKET, SO_RCVBUF, (void*)&rcv_buf, &len);
if(state)
    error_handling("getsockopt() error");

printf("Input buffer size: %d \n", rcv_buf);
printf("Output buffer size: %d \n", snd_buf);
```

get_buf.c의 일부

```
root@my_linux:/tcip# gcc get_buf.c -o getbuf
root@my_linux:/tcip# ./getbuf
Input buffer size: 87380
Output buffer size: 16384
```


소켓의 입출력 버퍼 크기 변경

9

set_buf.c의 일부

```
int snd_buf=1024*3, rcv_buf=1024*3;
```

```
sock=socket(PF_INET, SOCK_STREAM, 0);
state=setsockopt(sock, SOL_SOCKET, SO_RCVBUF, (void*)&rcv_buf, sizeof(rcv_buf));
state=setsockopt(sock, SOL_SOCKET, SO_SNDBUF, (void*)&snd_buf, sizeof(snd_buf));
len=sizeof(snd_buf);
state=getsockopt(sock, SOL_SOCKET, SO_SNDBUF, (void*)&snd_buf, &len);
len=sizeof(rcv_buf);
state=getsockopt(sock, SOL_SOCKET, SO_RCVBUF, (void*)&rcv_buf, &len);
printf("Input buffer size: %d \n", rcv_buf);
printf("Output buffer size: %d \n", snd_buf);
```

내가 요청한것이 우선이 아니라
os레벨에서 알아서 관리를 한다.
참고용으로 보고 랜잡으면
바꿔주지만 아니라면 그냥
os판단하에 값을 넣는다

```
root@my_linux:/tcpip# gcc set_buf.c -o setbuf
root@my_linux:/tcpip# ./setbuf
Input buffer size: 6144
Output buffer size: 6144
```

소켓의 입출력 버퍼 크기 변경

10

- 연결설정(3-way handshake) 후에는 버퍼 크기 변경이 불가
 - ▣ 서버의 경우 listen() 호출 이전에 설정
 - ▣ 클라이언트의 경우 connect() 호출 이전에 설정

확인해야함

찾기로는 accept으로도 될 것 같지만
listen으로 가르쳐 주신 것으로 통일

11

SO_REUSEADDR

bind() error

12

□ 시나리오-1: 클라이언트를 restart할 경우

- 1) 서버 실행하고 클라이언트가 서버에 연결된다
 - 2) 클라이언트를 Ctrl+C로 강제 종료시킨다.
 - 3) 클라이언트를 재 실행시킨다.
- 아무 문제 없음!

클라의 경우 > 임의의 남은 포트를 할당 해 주므로
재시작 할 경우 time_wait 상태인 이전에 사용했던 포트가 아닌
새로운 포트를 할당 해 줌

□ 시나리오-2: 서버를 restart할 경우

- 1) 서버 실행하고 클라이언트가 서버에 연결된다
- 2) 서버를 Ctrl+C로 강제 종료시킨다.
- 3) 서버를 재 실행시킨다.

□ bind() error 발생!!

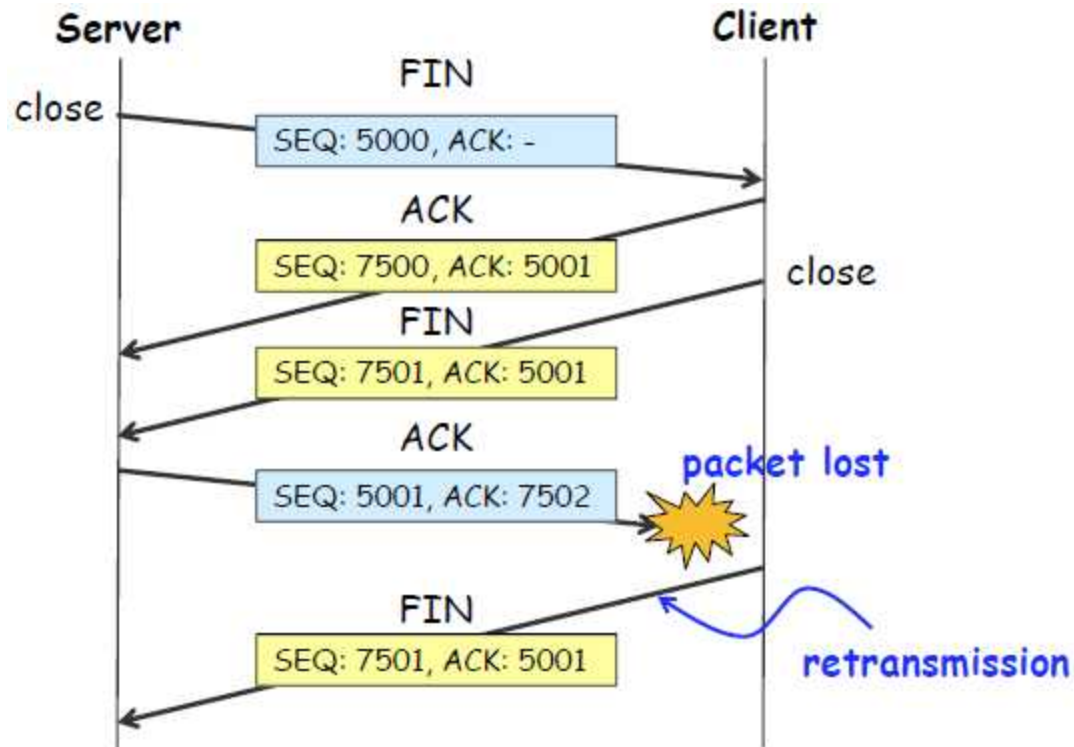
- "Address already in use"
- EADDRINUSE

에러, 주소가 사용되고 있다
error address in use

주소할당 에러의 원인 time-wait

13

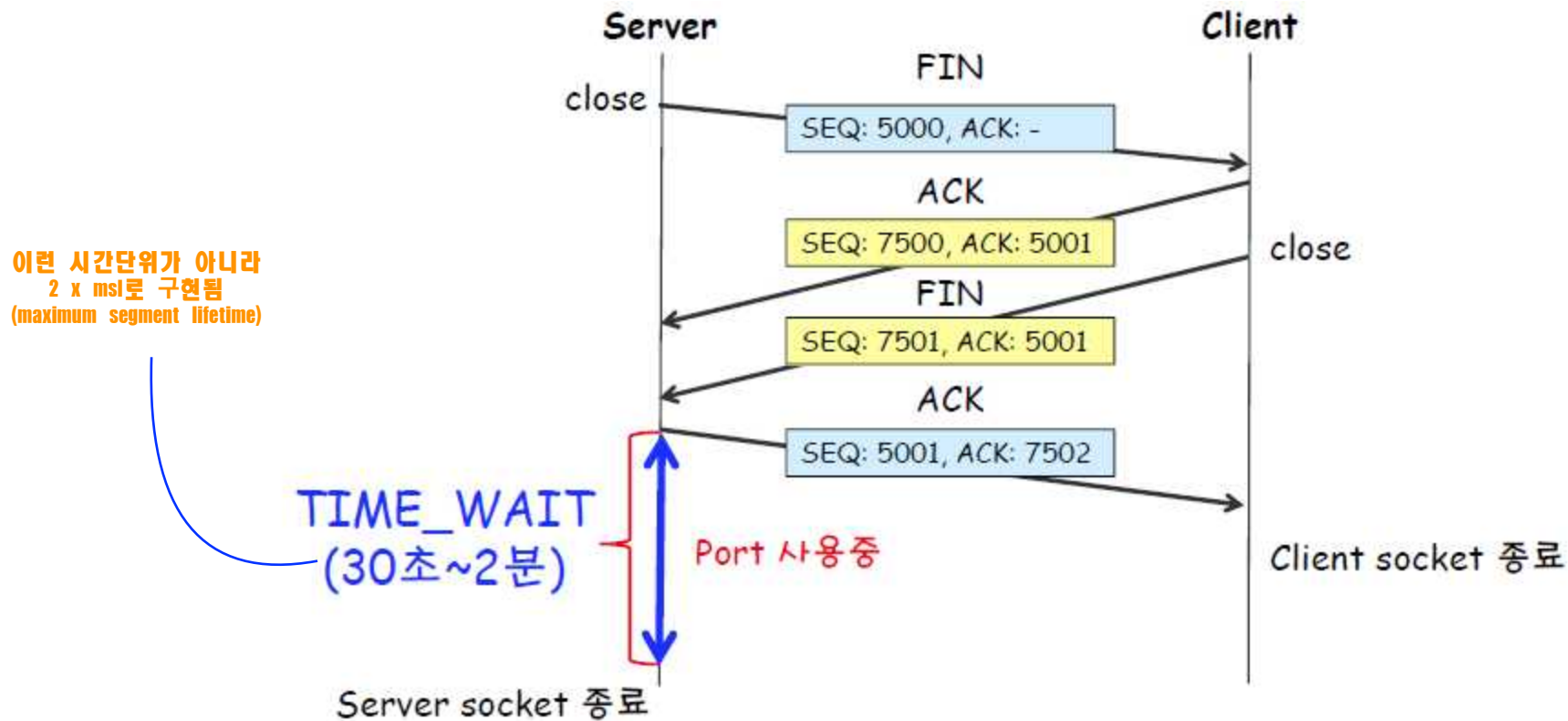
- TCP소켓에서 클라이언트가 서버의 마지막 ACK를 수신하지 못 할 경우 FIN을 재전송 한다.



주소할당 에러의 원인 time-wait

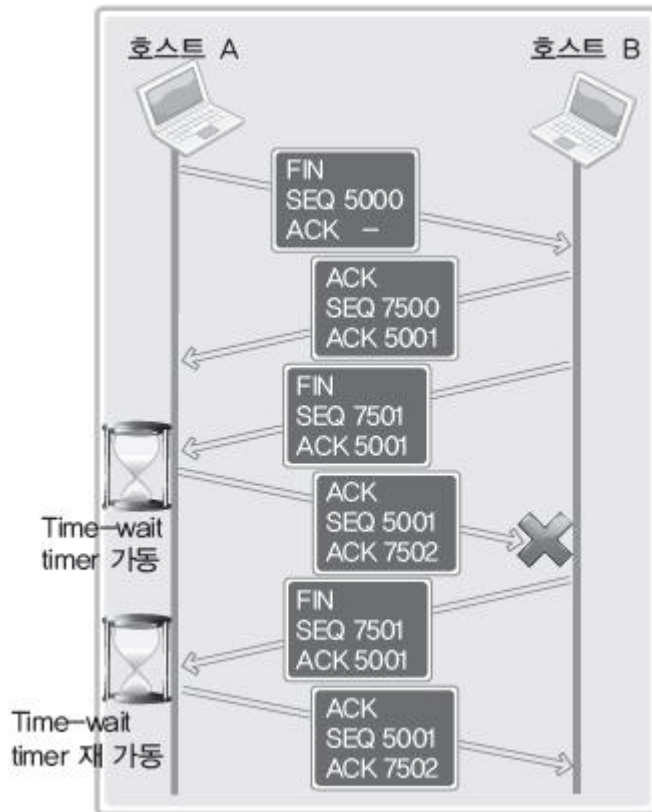
14

- 클라이언트에서 재전송되는 FIN의 수신을 위해 서버는 일정기간 TIME_WAIT 상태에 접어든다. 이때, port 번호가 여전히 lock되어 있다.



주소의 재할당

15



Time-wait은 길어질 수 있다

언제?

bind하다 에러가 났으니까
bind전에만 호출하면 된다

Port 할당이 가능하도록 옵션의 변경

reuseaddr_eserver.c의 일부

```
optlen=sizeof(option);  
option=TRUE;  
setsockopt(serv_sock, SOL_SOCKET, SO_REUSEADDR, (void*)&option, optlen);
```

16

TCP_NODELAY

tcp level에서의 option

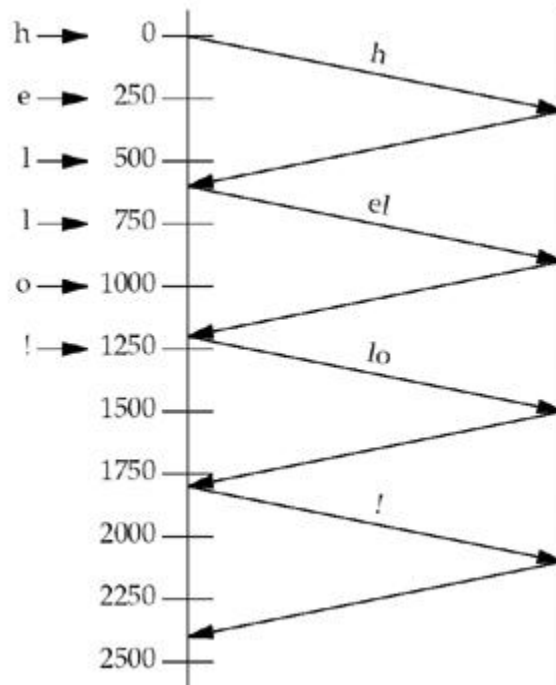
Nagle 알고리즘

요점은 최대한 모은 후 보낸다

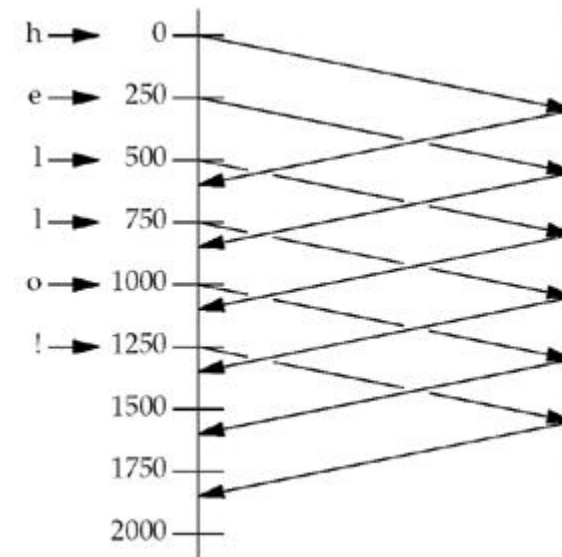
목적, 최대한 큰 segment를 만들어서 보냄
> 네트워크 혼잡도를 줄임

17

현재 tcp 구현부의 기본 동작
> Nagle이 켜져있는 상태



Nagle 알고리즘 ON



Nagle 알고리즘 OFF

"Nagle 알고리즘은 앞서 전송한 데이터에 대한 ACK 메시지를 받아야만,
다음 데이터를 전송하는 알고리즘이다!"

작은 단위로 보내게 되면
>헤더만 40바이트이기 때문에
너무 낭비가 심함
>>네트워크가 혼잡해지는 상황을 초래

Small Packet Problem

이를 막기 위해 nagle 알고리즘 사용시
한번 보내고 다음 것을 보낼때
ack 를 받을 때 까지 기다린다
>>조금 더 나은 효율을 보여줌

18

- 클라이언트 프로그램이 서버에게 1 바이트 문자 단위로 입력하는 경우를 고려해 보자.
 - ▣ 실제 1 바이트 데이터 전송을 위해, TCP/IP 헤더(=40바이트)를 더하면 41 바이트가 소켓을 통해 전송된다.
 - ▣ 서버 소켓이 ACK 패킷(=40바이트)을 응답한다.
 - ▣ 수신자 응용 프로그램이 1바이트를 입력버퍼(Recv-Q)로 부터 읽으면, 다시 Window Update 패킷(=40바이트)를 응답한다.

+ 경우
1. 수신자측 어플리케이션에서 데이터를
빠르게 가져가지 않을때

MTU로 결정
MTU의 헤더와 트레일러를 제거시킴는 것
ip, tcp헤더를 포함한 메시지 세그먼트 사이즈
== MSS

Nagle 알고리즘

maximum segment size

19

전송할 데이터의 발생

만약 윈도우 크기가 MSS보다 크고 버퍼에 있는 데이터의 크기가 MSS보다 클 경우
MSS 크기의 세그먼트를 만들어서 전송

그렇지 않은 경우

전송 후 ACK를 받지 않은 데이터가 있을 경우

ACK를 받을 때까지 대기

그렇지 않은 경우

바로 전송

== 보낸것에 대한 ack는 다 받았다는 것

▪ 다음과 같은 경우에만 전송

- 송신 버퍼에 쌓여 있는 데이터의 양이 MSS보다 크고 수신 버퍼에 MSS만큼의 여유가 있을 때
- 송신 버퍼에서 대기 중 ACK를 수신할 때
- 전송 후 ACK를 받지 못한 데이터가 없을 때(즉 현재 들어온 데이터 이전에 전송한 데이터에 대해서 모두 ACK를 받은 상태일 때)

Nagle 알고리즘의 중단

20

□ reuseadr_eserver.c의 일부

디폴트가 켜져있는것이니, 끄는 방법부터 설명

Nagle 알고리즘의 중단을 명령하는 코드

```
int opt_val=1;
setsockopt(sock, IPPROTO_TCP, TCP_NODELAY, (void*)&opt_val, sizeof(opt_val));
```

Nagle 알고리즘의 설정상태 확인하는 코드

```
int opt_val;
socklen_t opt_len;
opt_len=sizeof(opt_val);
getsockopt(sock, IPPROTO_TCP, TCP_NODELAY, (void*)&opt_val, &opt_len);
```

꾸물거린다.

소켓레벨옵션

21

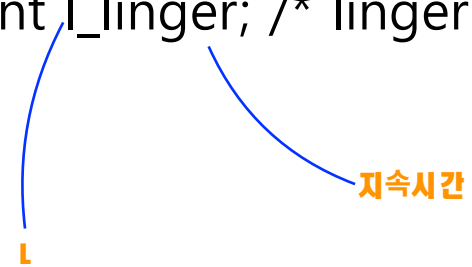
SO_LINGER

종료방식 조절

SO_LINGER를 이용한 연결 종료 방식 조정

22

```
struct linger {  
    int l_onoff; /* 0=off, nonzero=on */  
    int l_linger; /* linger time, POSIX specifies units as seconds */  
};
```



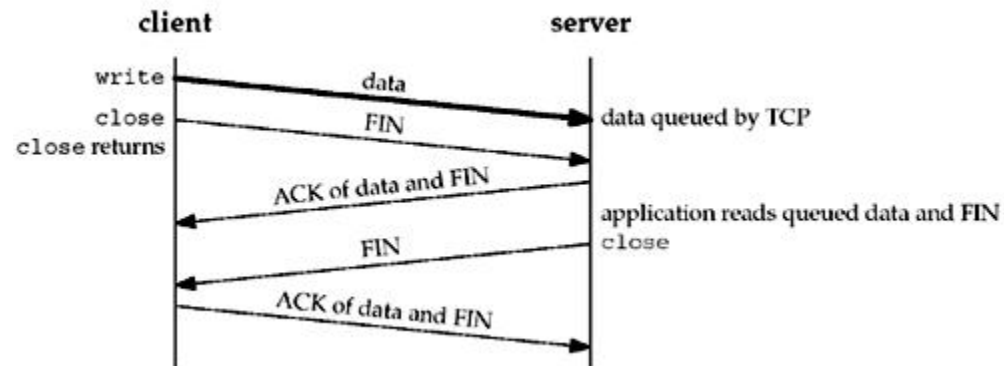
지속시간

L

SO_LINGER를 이용한 연결 종료 방식 조정

23

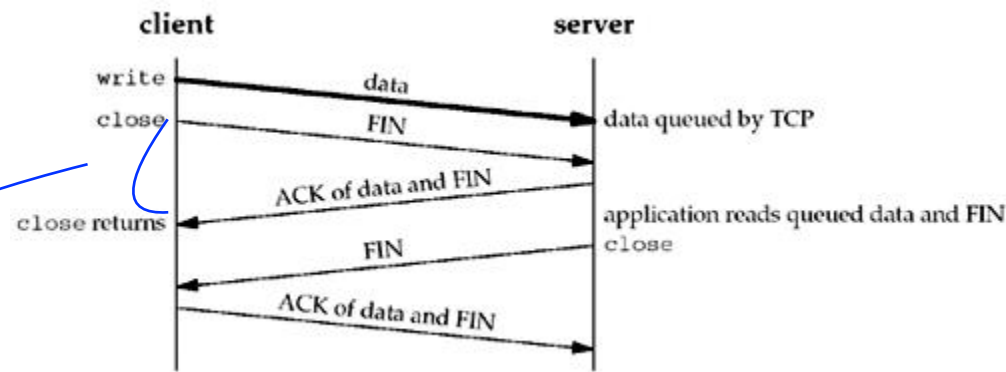
- close() (default)
 - ▣ l_onoff = 0



SO_LINGER를 이용한 연결 종료 방식 조정

24

- close()
 - ▣ l_onoff = 1
 - ▣ l_linger != 0



- ▣ close() could return ETIMEDOUT.

기다리다가 시간내에 ack가 안오면
timeout이 걸림

close호출시 원래 바로 return 해야 하지만
이렇게 세팅해 준 경우 FIN에 대한 ACK가 올때까지 기다리게된다
> 즉 내가 보낸 fin에 대한 ack를 확인 할 수 있게 된다

SO_LINGER를 이용한 연결 종료 방식 조정

25

□ close()

- l_onoff = 1
- l_linger = 0

질문: reset을 상대방이 못받은 경우에는 어떻게 되는가?
같은 방식으로 timewait 걸어서 ack 올때까지 기다리는 형식?
> timewait을 거는 것이 아니라 송신자가 바로 죽어버리는 것이기 때문에
수신자 쪽에서 처리하는 것이 아니라
송신자 포트에 새로운 프로세스가 바인딩되는 경우
기존의 수신자가 보낸 메시지에 RST를 보내는 방식으로 처리한다.

갑작스런 종료

□ abortive 종료

- RST sent to other end
- connection state set to CLOSED (no TIME_WAIT state)

fin 대신에 reset을 보냄 socket send buffer and socket receive buffer discarded.

