Chapter 09

소켓의 옵션과 입출력 버퍼의 크기

다양한 소켓의 옵션

Get: 참조
Set: 변경

Protocol Level	Option Name	Get	Set
SOL_SOCKET	SO_SNDBUF	O	O
	SO_RCVBUF	O	O
	SO_REUSEADDR	O	O
	SO_KEEPALIVE	O	O
	SO_BROADCAST	O	O
	SO_DONTROUTE	O	O
	SO_OOBINLINE	O	O
	SO_ERROR	O	X
	SO_TYPE	O	X
IPPROTO_IP	IP_TOS	O	O
	IP_TTL	O	O
	IP_MULTICAST_TTL	O	O
	IP_MULTICAST_LOOP	O	O
	IP_MULTICAST_IF	O	O
IPPROTO_TCP	TCP_KEEPALIVE	O	O
	TCP_NODELAY	O	O
	TCP_MAXSEG	O	O

< /usr/include/x86_64-linux-gnu/bits/socket-constants.h>

< /usr/include/x86_64-linux-gnu/bits/in.h>

</usr/include/netinet/tcp.h>

- 소켓의 옵션들
  - 소켓의 특성을 변경시킬 때 사용하는 옵션 정보
  - 소켓의 옵션은 계층별로 분류
- SOL_SOCKET 레벨 옵션
  - 소켓에 대한 가장 일반적인 옵션
- IPPROTO_IP 레벨 옵션
  - IP 프로토콜에 관련된 사항
- IPPROTO_TCP 레벨 옵션
  - TCP 프로토콜에 관련된 사항

소켓 옵션 내용 #1

Protocol Level	Option Name	자료형	설명
SOL_SOCKET	SO_SNDBUF	int	송신 버퍼의 크기 지정 및 확인
	SO_RCVBUF	int	수신 버퍼의 크기 지정 및 확인
	SO_REUSEADDR	BOOL(0, 1)	이미 사용된 주소를 재사용 여부
	SO_KEEPALIVE	BOOL	일정시간마다 keep alive 메시지 전송 여부
	SO_BROADCAST	BOOL	브로드캐스트 사용 가능 여부
	SO_DONTROUTE	BOOL	데이터 전송시 라우팅 테이블 참조 과정 생략 여부
	SO_OOBINLINE	BOOL	일반 데이터 스트림으로 out-of-band (긴급 데이터) 수신 여부: 13장
	SO_ERROR	int	에러 상태 반환 (에러 코드 리턴)
	SO_TYPE	int	소켓의 타입 (TCP, UDP)

- Protocol level과 option name들은 #define 매크로 상수로 정의되어 있음

소켓 옵션 내용 #2

Protocol Level	Option Name	자료형	설명
IPPROTO_IP	IP_TOS	int	Type of Service 설정
	IP_TTL	int	IP패킷의 TTL(time-to-live) 설정
	IP_MULTICAST_TTL	int	멀티캐스트 패킷의 TTL 설정
	IP_MULTICAST_LOOP	BOOL	자신이 보낸 멀티캐스트 패킷 수신 여부
	IP_MULTICAST_IF	in_addr	멀티캐스트 패킷을 보낼 인터페이스 설정
IPPROTO_TCP	TCP_KEEPALIVE	int	TCP keep alive 시간 간격 지정
	TCP_NODELAY	BOOL (int)	Nagle 알고리즘 사용 여부 지정
	TCP_MAXSEG	int	TCP 최대 세그먼트 지정

소켓 옵션 참조 함수:getsockopt()

- getsockopt()

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

int getsockopt(int sock, int level, int optname, void *optval, socklen_t *optlen);
-> 성공 시 0, 실패 시 -1 반환
```

- sock: 옵션 확인을 위한 소켓 파일 디스크립터
- level: 확인할 옵션의 프로토콜 레벨
- optname: 확인할 옵션의 이름
- optval: 해당 옵션 이름에 대한 결과값을 저장하기 위한 버퍼의 주소 전달
- optlen: optval의 크기를 담고 있는 변수의 주소 전달
  - 함수 호출이 완료되면, 반환된 옵션 정보의 크기가 바이트 단위로 저장됨

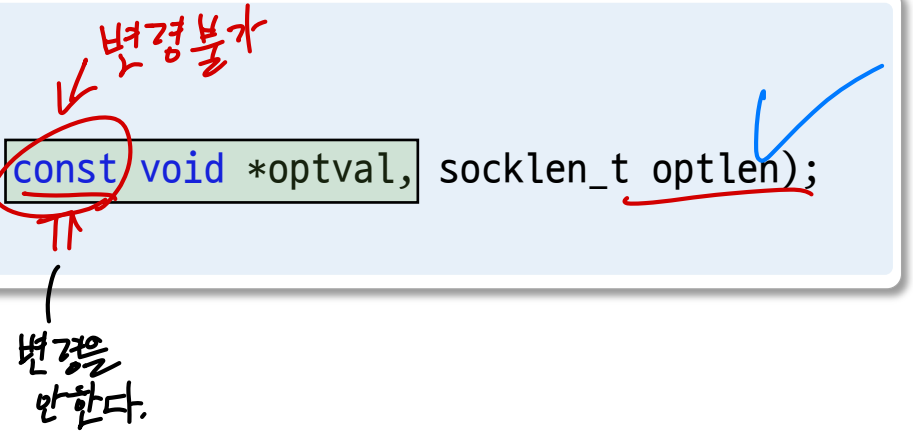
- 소켓 옵션 테이블에서 제시한 Protocol Level은 두 번째 인자(int level), Option Name은 세 번째 인자(int optname)로 전달되어 해당 옵션의 등록 정보를 가져옴

소켓 옵션 설정 함수: setsockopt()

- setsockopt()

```
#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: 옵션 변경을 위한 소켓 파일 디스크립터
- level: 변경할 옵션의 프로토콜 레벨
- optname: 변경할 옵션의 이름
- optval: 변경할 옵션정보를 저장한 버퍼의 주소 전달
- optlen: optval로 전달될 옵션정보의 크기가 바이트 단위로 저장됨

소켓의 타입정보(TCP or UDP)의 확인: sock_type.c

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

void error_handling(char *message)
{
    fputs(message, stderr);
    fputc('\n', stderr);
    exit(1);
}

int main(int argc, char *argv[])
{
    int tcp_sock, udp_sock;
    int sock_type;
    socklen_t optlen;
    int state;

    optlen=sizeof(sock_type);

    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);

    return 0;
}
```

```
$ gcc sock_type.c -o socktype
$ ./socktype
SOCK_STREAM: 1
SOCK_DGRAM: 2
Socket type one: 1
Socket type two: 2
```

소켓의 타입정보(SO_TYPE)는
변경이 불가능하고, 확인만 가능

소켓의 입출력 버퍼 크기 확인: get_buf.c

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

void error_handling(char *message)
{
    fputs(message, stderr);
    fputc('\n', stderr);
    exit(1);
}
```

```c
int main(int argc, char *argv[])
{
    int sock;
    int snd_buf, rcv_buf, state;
    socklen_t len;

    sock=socket(PF_INET, SOCK_STREAM, 0);  // 소켓 생성은 필수

    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("Receive buffer size: %d \n", rcv_buf);
    printf("Send buffer size: %d \n", snd_buf);

    return 0;
}
```

커널에서 설정한 입출력 버퍼 크기
(송수신 데이터 사이즈와 상관없음)

```
$ gcc get_buf.c -o get_buf
$ ./get_buf
Receive buffer size: 131072 더큰 10배 정도
Send buffer size: 16384
```

```
Receive buffer: 131072 (128 bytes)
Send buffer: 16384 (16 bytes)
```

소켓의 입출력 버퍼 크기 설정

- set_buf.c 일부

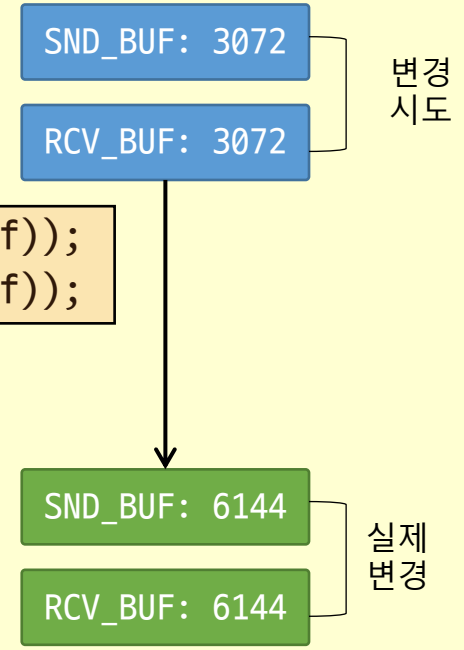
```c
int sock;
int snd_buf=1024*3;
int rcv_buf=1024*3;
int state;
socklen_t len;

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




```
$ ./set_buf
Receive buffer size: 6144
Send buffer size: 6144
```

- 입출력 버퍼는 상당히 주의 깊게 다뤄져야 하는 영역임
- 실행결과와 같이 코드에서 요구하는 크기가 완벽히 반영되지는 않음

set_buf.c #1

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/socket.h>
void error_handling(char *message);

int main(int argc, char *argv[])
{
    int sock;
    int snd_buf=1024*3;
    int rcv_buf=1024*3;
    int state;
    socklen_t len;

    sock = socket(PF_INET, SOCK_STREAM, 0);
    state = setsockopt(sock, SOL_SOCKET, SO_RCVBUF, (void*)&rcv_buf, sizeof(rcv_buf));
    if(state)
        error_handling("setsockopt() error!");

    state = setsockopt(sock, SOL_SOCKET, SO_SNDBUF, (void*)&snd_buf, sizeof(snd_buf));
    if(state)
        error_handling("setsockopt() error!");
```

SO_RCVBUF와 SO_SNDBUF의 크기를 임의로 변경 (3072)

set_buf.c #2

```
    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("Receive buffer size: %d \n", rcv_buf);
    printf("Send buffer size: %d \n", snd_buf);
    return 0;
}

void error_handling(char *message)
{
    fputs(message, stderr);
    fputc('\n', stderr);
    exit(1);
}
```

변경 요청 크기와 다른 결과 리턴
(6144 = 1024 * 6)

```
$ ./set_buf
RCV buffer size: 6144
SND buffer size: 6144
```

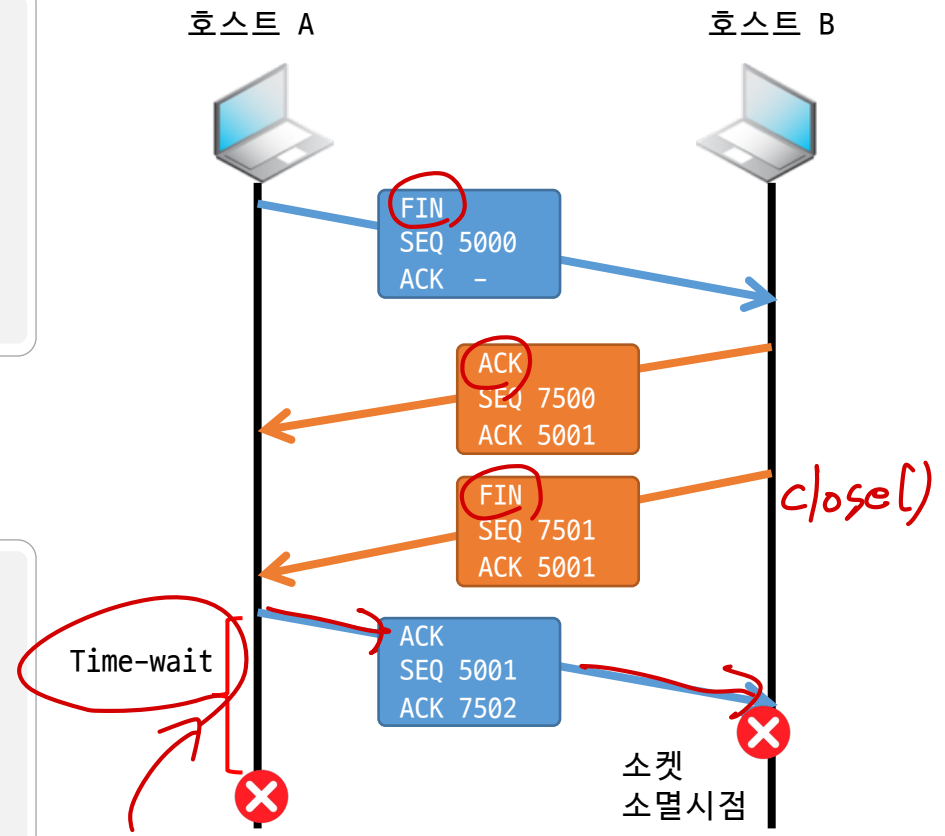
주소할당 에러의 원인 time-wait

Time-wait의 이해

- 서버, 클라이언트에 상관없이 TCP 소켓에서 연결 종료를 목적으로 Four-way handshaking의 첫 번째 메시지(FIN)를 전달하는 호스트의 소켓은 Time-wait 상태를 거침
- Time-wait 상태 동안에는 해당 소켓이 소멸되지 않아서 할당 받은 Port 를 다른 소켓이 할당할 수 없음 (Binding Error 발생)

Time-wait의 존재 이유

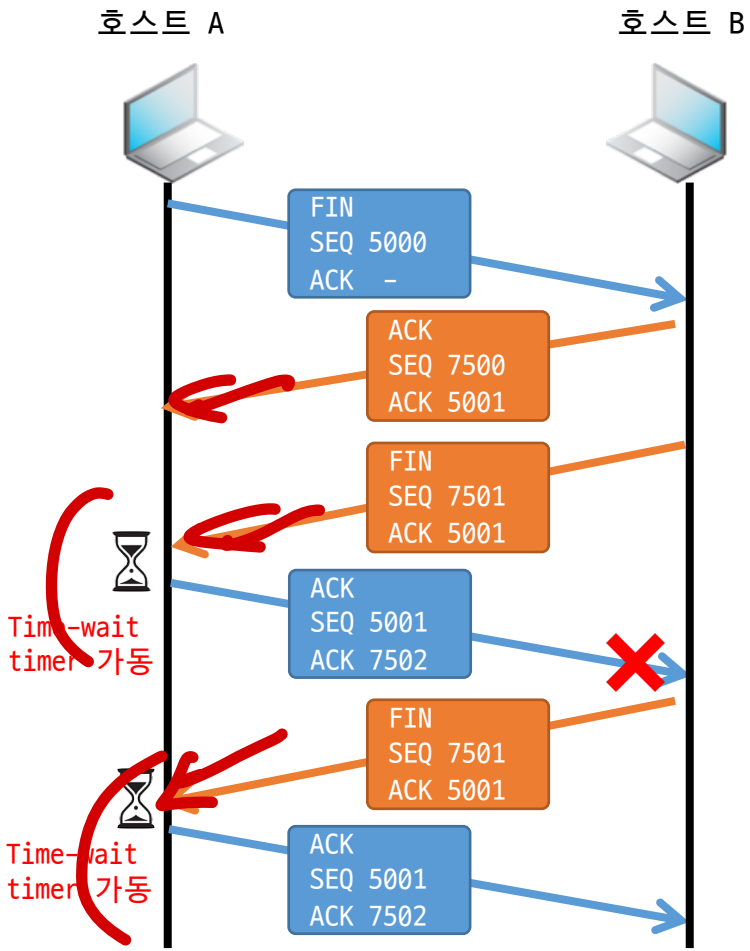
- 오른쪽 그림에서 호스트 A의 마지막 ACK가 소멸되는 상황을 대비해서 Time-wait 상태가 필요함
- 호스트 A의 마지막 ACK가 소멸되면, 호스트 B는 계속해서 FIN 메시지를 호스트 A에 전달하기 때문



주소의 재할당: SO_REUSEADDR 옵션

Time-wait은 길어질 수 있음

- Time-wait은 필요하지만, 실제 서비스중인 서버의 경우 Time-wait은 심각한 문제가 될 수 있음
- Time-wait 상태에 있는 포트 번호의 재할당이 가능하도록 코드 수정
- 클라이언트의 경우, 포트 번호가 자동 할당되기 때문에 서버 프로그램과 달리 큰 영향이 없음



포트 번호 할당이 가능하도록 옵션 변경 (SO_REUSEADDR)

```
optlen=sizeof(option);
option=TRUE;

setsockopt(serv_sock, SOL_SOCKET, SO_REUSEADDR, &option, optlen);
```

reuseadr_eserver.c #1

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

#define TRUE 1
#define FALSE 0
void error_handling(char *message);

int main(int argc, char *argv[])
{
    int serv_sock, clnt_sock;
    char message[30];
    int option, str_len;
    socklen_t optlen, clnt_adr_sz;
    struct sockaddr_in serv_adr, clnt_adr;

    if(argc!=2) {
        printf("Usage : %s <port>\n", argv[0]);
        exit(1);
    }
    serv_sock=socket(PF_INET, SOCK_STREAM, 0);
    if(serv_sock==-1)
        error_handling("socket() error");
```

4장의 echo_client.c 소스와 테스트

reuseadr_eserver.c #2

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

    memset(&serv_adr, 0, sizeof(serv_adr));
    serv_adr.sin_family=AF_INET;
    serv_adr.sin_addr.s_addr=htonl(INADDR_ANY);
    serv_adr.sin_port=htons(atoi(argv[1]));

    if(bind(serv_sock, (struct sockaddr*)&serv_adr, sizeof(serv_adr)))
        error_handling("bind() error ");

    if(listen(serv_sock, 5)==-1)
        error_handling("listen error");
    clnt_adr_sz=sizeof(clnt_adr);
    clnt_sock=accept(serv_sock, (struct sockaddr*)&clnt_adr,&clnt_adr_sz);

    while((str_len=read(clnt_sock,message, sizeof(message)))!= 0)
    {
        write(clnt_sock, message, str_len);
        write(1, message, str_len);
    }
    close(clnt_sock);
    close(serv_sock);
    return 0;
}
```

SO_REUSEADDR 사용

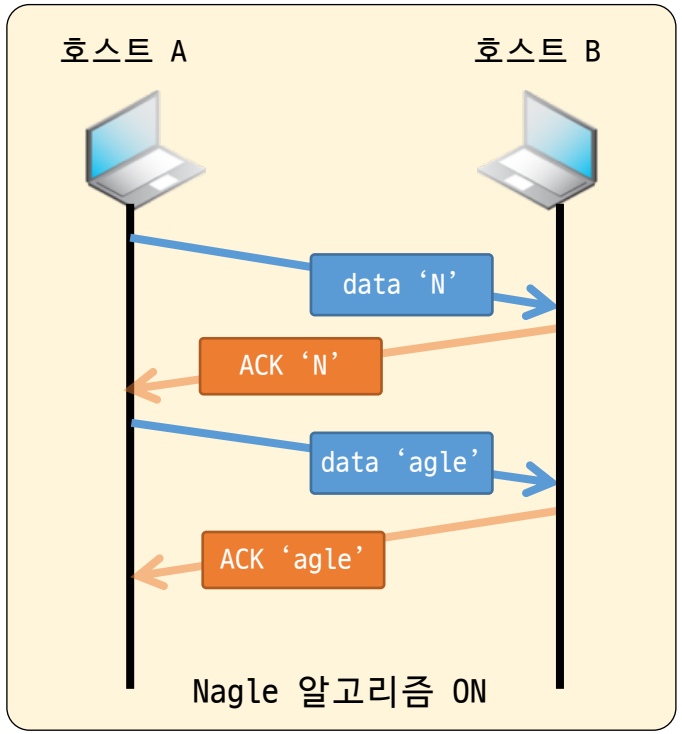
추가

write(1, message, str_len)
- 화면 출력

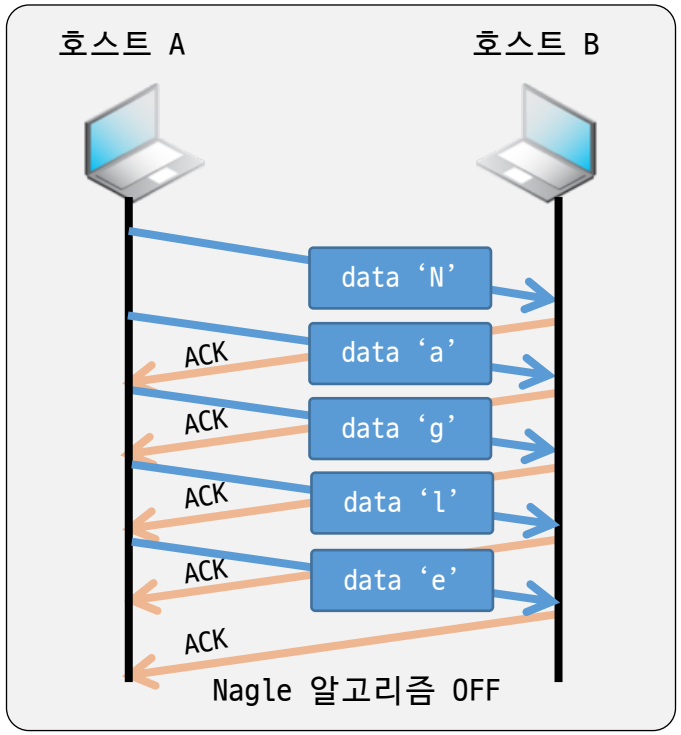
Nagle 알고리즘

기본 TCP 전송 방법

- Nagle 알고리즘
  - 앞서 전송한 데이터에 대한 ACK 메시지를 받은 후, 다음 데이터를 전송하는 알고리즘
    - 인터넷의 과도한 트래픽으로 인한 전송 속도의 저하를 막기 위해 디자인
  - TCP 소켓은 기본적으로 Nagle 알고리즘을 적용해서 데이터를 송수신함



default



Nagle 알고리즘의 중단: TCP_NODELAY

- TCP_NODELAY: Nagle 알고리즘 중단 옵션
  - 1: Nagle 알고리즘 멈춤
  - 0: Nagle 알고리즘 동작
- Nagle 알고리즘 중단

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

- Nagle 알고리즘 설정 상태 확인

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

Questions?