



ISL

• • •

Network Programming #3

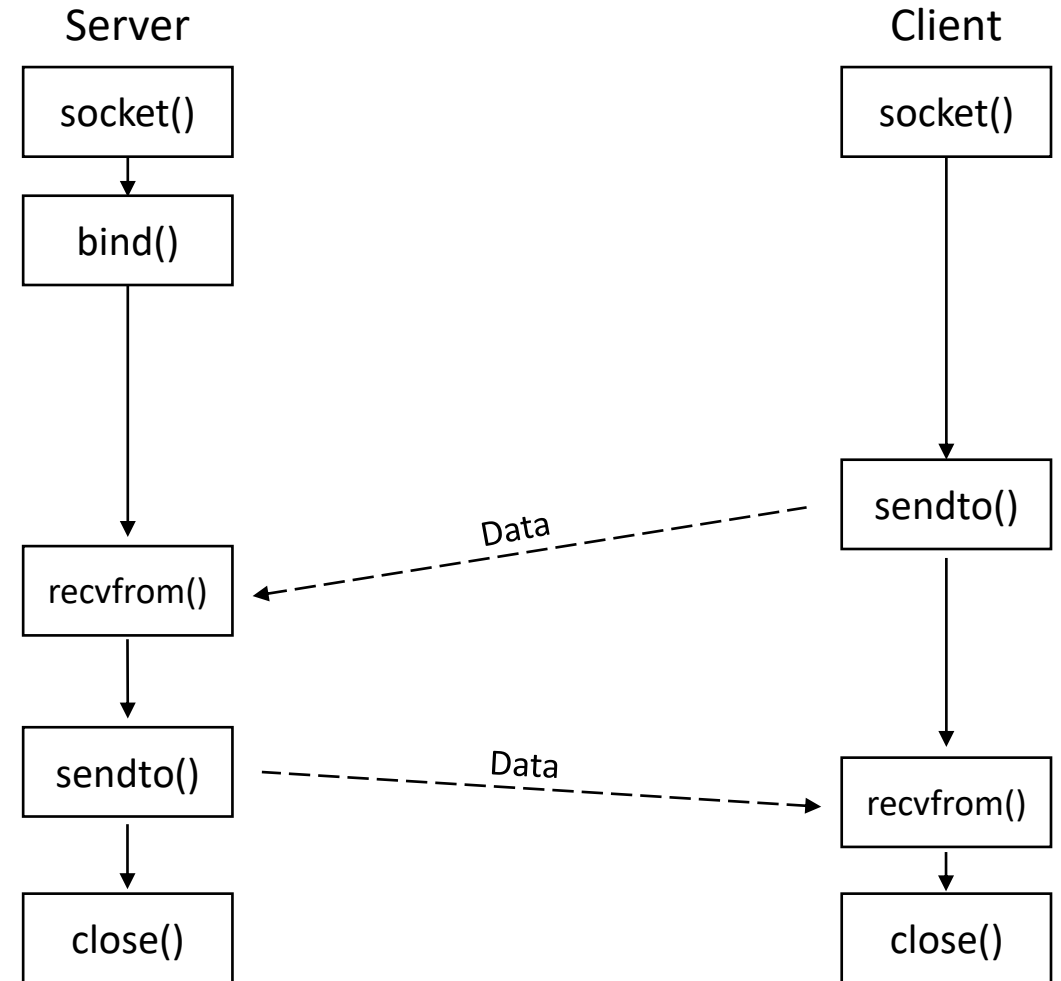
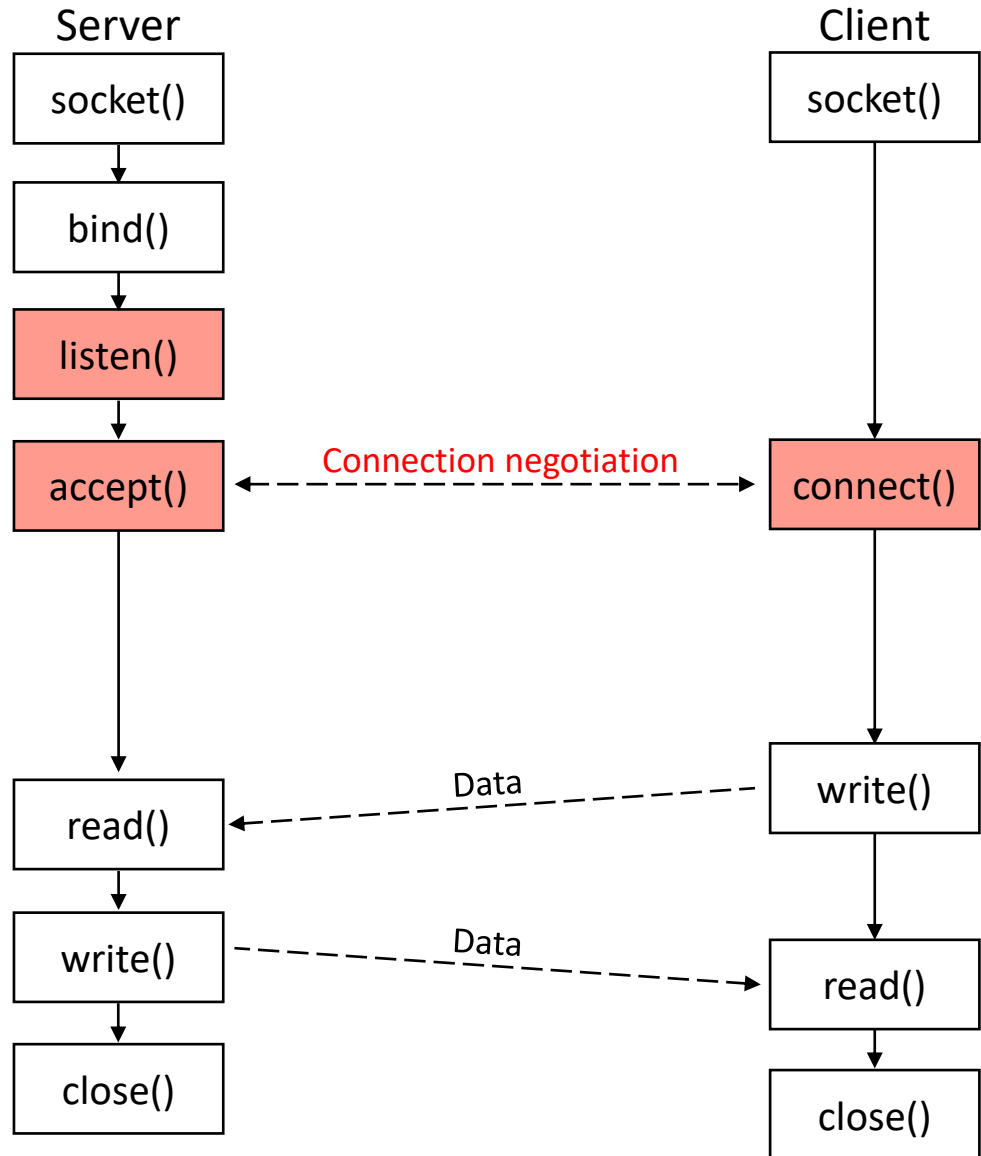
ISL (IoT Standard Lab)

Index

1. Echo UDP server & client
2. TCP vs UDP
3. Connected UDP
4. TCP Half Close
5. Network assignment #3

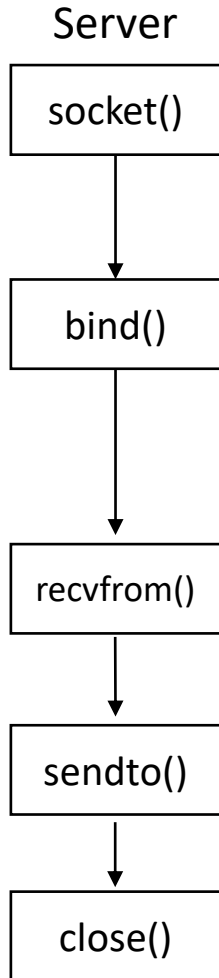
01 Echo UDP Server & Client

Echo UDP Server & Client 흐름



01 Echo UDP Server & Client

UDP 서버



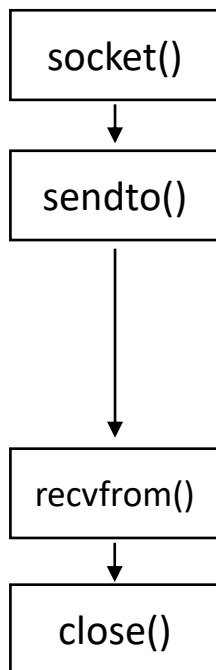
```
if((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {  
    perror("socket creation failed");  
    return -1;  
}  
  
servaddr.sin_family = AF_INET;  
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);  
servaddr.sin_port = htons(atoi(argv[1]));  
  
if(bind(sockfd, (const struct sockaddr *)&servaddr, sizeof(servaddr)) < 0) {  
    perror("bind failed");  
    return -1;  
}  
  
while(1) {  
    len = sizeof(cliaddr);  
    str_len = recvfrom(sockfd, buf, sizeof(buf), 0, (struct sockaddr*)&cliaddr, &len);  
    if(buf[0] == 'q' || buf[0] == 'Q') {  
        break;  
    }  
    sendto(sockfd, buf, str_len, 0, (struct sockaddr*)&cliaddr, len);  
}  
  
close(sockfd);  
return 0;
```

1. socket type을 SOCK_DGRAM으로 설정

- 1. 0 : flags
- 2. &cliaddr : socket address structure
- 3. &len : socket address structure size

UDP 클라이언트

Client



```
if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
    perror("socket creation failed");
    return -1;
}

servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = inet_addr(argv[1]);
servaddr.sin_port = htons(atoi(argv[2]));

while(1) {
    fputs("Input message(Q to quit): ", stdout);
    fgets(buf, BUF_SIZE, stdin);
    sendto(sockfd, buf, strlen(buf), 0, (struct sockaddr*)&servaddr, sizeof(servaddr));
    if(!strcmp(buf, "q\n") || !strcmp(buf, "Q\n")) {
        break;
    }

    len = sizeof(servaddr);
    str_len = recvfrom(sockfd, buf, sizeof(buf), 0, (struct sockaddr*)&servaddr, &len);
    buf[str_len-1] = 0;

    printf("Message from server: %s\n", buf);
}
close(sockfd);
return 0;
```

| 실행 결과

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket3/echo$ ./server 8080
smalldragon@SD-DESKTOP:~/Workspace/socket/socket3/echo$
smalldragon@SD-DESKTOP:~/Workspace/socket/socket3/echo$
smalldragon@SD-DESKTOP:~/Workspace/socket/socket3/echo$
```

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket3/echo$ ./client 127.0.0.1 8080
Input message(Q to quit): Hello
Message from server: Hello
Input message(Q to quit): How are you
Message from server: How are you
Input message(Q to quit): Bye
Message from server: Bye
Input message(Q to quit): q
smalldragon@SD-DESKTOP:~/Workspace/socket/socket3/echo$
smalldragon@SD-DESKTOP:~/Workspace/socket/socket3/echo$
smalldragon@SD-DESKTOP:~/Workspace/socket/socket3/echo$
```

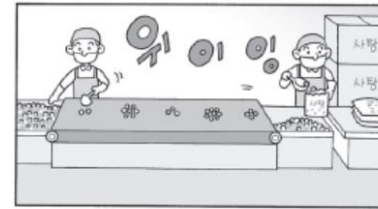
연결지향형 소켓(SOCK_STREAM) vs 비연결지향형 소켓(SOCK_DGRAM)

두 타입의 소켓



TCP 소켓

- ▶ 연결지향형 소켓(SOCK_STREAM)의 데이터 전송특성
 - ▶ 중간에 데이터 소멸되지 않는다.
 - ▶ 전송 순서대로 데이터가 수신된다.
 - ▶ 데이터의 경계가 존재하지 않는다.
 - ▶ 소켓 대 소켓의 연결은 반드시 1대 1의 구조.



TCP 데이터 전송특성

UDP 소켓

- ▶ 비 연결지향형 소켓(SOCK_DGRAM)의 데이터 전송특성
 - ▶ 전송순서 상관없이 빠른 속도의 전송을 지향
 - ▶ 데이터 손실 및 파손의 우려 있다.
 - ▶ 데이터의 경계가 존재한다.
 - ▶ 한번에 전송할 수 있는 데이터의 크기가 제한된다.



UDP 데이터 전송특성



연결지향형 소켓(SOCK_STREAM) vs 비연결지향형 소켓(SOCK_DGRAM)

- Client

```
char sndbuf[MAX];
char rcvbuf[RMAX];
```

```
#define MAX 1000
#define RMAX 30000
```

```
if(connect(sfd, (const struct sockaddr*)&servaddr, sizeof(servaddr)) < 0) {
    perror("connect error");
    return 1;
}
```

```
for(int i=0; i<MAX; i++) {
    sndbuf[i] = 'M';
}
```

1000bytes 데이터를 넣고 write()를 3번 호출

```
for(int i=0; i<3; i++) {
    write(sfd, sndbuf, sizeof(sndbuf));
    printf("send %d\n", i);
}
```

```
sleep(4);
```

```
memset(rcvbuf, 0, sizeof(rcvbuf));
int rcvlen = read(sfd, rcvbuf, sizeof(rcvbuf));
printf("%s %d\n", rcvbuf, rcvlen);
```

최대 30000bytes를 한번에
읽을 수 있도록 read()를 한 번
호출

TCP Client

```
for(int i=0; i<MAX; i++) {
    sndbuf[i] = 'M';
}
```

1000bytes 데이터를 넣고 sendto()를 3번 호출

```
for(int i=0; i<3; i++) {
    sendto(sockfd, sndbuf, strlen(sndbuf), 0, (struct sockaddr*)&serv_addr, sizeof(serv_addr));
    printf("send %d\n", i);
}
```

```
sleep(4);
```

```
memset(rcvbuf, 0, sizeof(rcvbuf));
int rcvlen = recvfrom(sockfd, rcvbuf, sizeof(rcvbuf), 0, (struct sockaddr*)&serv_addr, &len);
printf("%s %d\n", rcvbuf, rcvlen);
```

최대 30000bytes를 한번에
읽을 수 있도록 recvfrom()를 한
번 호출

UDP Client

연결지향형 소켓(SOCK_STREAM) vs 비연결지향형 소켓(SOCK_DGRAM)

- Server

```
#define MAX 30000
```

```
char buf[MAX];
```

```
int rcv_len, total_len = 0;
for(int i=0; i<3; i++) {
    memset(buf, 0, sizeof(buf));
    rcv_len = read(cfd, buf, sizeof(buf));
    total_len += rcv_len;
    printf("rcv_len: %d total_len: %d\n", rcv_len, total_len);

    write(cfd, buf, rcv_len);

    puts("sleep 1s...");
    sleep(1);
}

close(cfd);
close(sfd);
return 0;
```

TCP Server

```
int rcv_len, total_len = 0;
for(int i=0; i<3; i++) {
    memset(buf, 0, sizeof(buf));
    len = sizeof(cli_addr);
    rcv_len = recvfrom(sockfd, buf, sizeof(buf), 0, (struct sockaddr*)&cli_addr, &len);
    total_len += rcv_len;
    printf("rcv_len: %d total_len: %d\n", rcv_len, total_len);

    sendto(sockfd, buf, strlen(buf), 0, (const struct sockaddr*)&cli_addr, len);

    puts("sleep 1s...");
    sleep(1);
}

close(sockfd);
return 0;
```

UDP Server

연결지향형 소켓(SOCK_STREAM) vs 비연결지향형 소켓(SOCK_DGRAM)

- TCP Result - Localhost

```

● smalldragon@smalldragon-desktop:~/Workspace/network_programming_2023/week2$ ./tcp_server3 8080
rcv_len: 1000 total_len: 1000
sleep 1s...
rcv_len: 2000 total_len: 3000
sleep 1s...
rcv_len: 0 total_len: 3000
sleep 1s...
● smalldragon@smalldragon-desktop:~/Workspace/network_programming_2023/week2$ ./tcp_server3 8080
rcv_len: 1000 total_len: 1000
sleep 1s...
rcv_len: 2000 total_len: 3000
sleep 1s...
rcv_len: 0 total_len: 3000
sleep 1s...
● smalldragon@smalldragon-desktop:~/Workspace/network_programming_2023/week2$ ./tcp_server3 8080
rcv_len: 1000 total_len: 1000
sleep 1s...
rcv_len: 2000 total_len: 3000
sleep 1s...
rcv_len: 0 total_len: 3000
sleep 1s...
○ smalldragon@smalldragon-desktop:~/Workspace/network_programming_2023/week2$ █

```

TCP Server

[illegible]

TCP Client

= 1000Bytes 씩 write()를 호출해도, 정확하게 1000Bytes 씩 보내지 않음

연결지향형 소켓(SOCK_STREAM) vs 비연결지향형 소켓(SOCK_DGRAM)

- TCP Result – Remote host

```
● smalldragon@isl-k8s-worker-2:~/Workspace/test_udp$ ./tcp_server3 8080
rcv_len: 2448 total_len: 2448
sleep 1s...
rcv_len: 552 total_len: 3000
sleep 1s...
rcv_len: 0 total_len: 3000
sleep 1s...
○ smalldragon@isl-k8s-worker-2:~/Workspace/test_udp$
```

1	Client IP → Server IP	0.000000	TCP	74 60162 → 8080 [SYN] Seq=0
2	Server IP → Client IP	0.002705	TCP	74 8080 → 60162 [SYN, ACK] S
3	Client IP → Server IP	0.002745	TCP	66 60162 → 8080 [ACK] Seq=1
4	Client IP → Server IP	0.002841	TCP	1066 60162 → 8080 [PSH, ACK] S
5	Client IP → Server IP	0.003080	TCP	1514 60162 → 8080 [ACK] Seq=10
6	Server IP → Client IP	0.005878	TCP	66 8080 → 60162 [ACK] Seq=1
7	Server IP → Client IP	0.005879	TCP	66 8080 → 60162 [ACK] Seq=1
8	Server IP → Client IP	0.005880	TCP	1514 8080 → 60162 [ACK] Seq=1
9	Client IP → Server IP	0.005915	TCP	618 60162 → 8080 [PSH, ACK] S
10	Client IP → Server IP	0.005947	TCP	66 60162 → 8080 [ACK] Seq=30

4

Transmission Control Protocol, Src Port: 60162, Dst Port: 8080, Seq: 1, Ack: 1, Len: 1000

Source Port: 60162
Destination Port: 8080
[Stream index: 0]
[Conversation completeness: Complete, WITH_DATA (31)]
[TCP Segment Len: 1000]
Sequence Number: 1 (relative sequence number)
Sequence Number (raw): 2608003899
[Next Sequence Number: 1001 (relative sequence number)]
Acknowledgment Number: 1 (relative ack number)
Acknowledgment number (raw): 4081004643
1000 = Header Length: 32 bytes (8)
Flags: 0x018 (PSH, ACK)
Window: 502
[Calculated window size: 64256]
[Window size scaling factor: 128]
Checksum: 0xafdf [unverified]
[Checksum Status: Unverified]
Urgent Pointer: 0
Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
[Timestamps]
[SEQ/ACK analysis]
TCP payload (1000 bytes)

5

Transmission Control Protocol, Src Port: 60162, Dst Port: 8080, Seq: 1001, Ack: 1, Len: 1448

Source Port: 60162
Destination Port: 8080
[Stream index: 0]
[Conversation completeness: Complete, WITH_DATA (31)]
[TCP Segment Len: 1448]
Sequence Number: 1001 (relative sequence number)
Sequence Number (raw): 2608004899
[Next Sequence Number: 2449 (relative sequence number)]
Acknowledgment Number: 1 (relative ack number)
Acknowledgment number (raw): 4081004643
1000 = Header Length: 32 bytes (8)
Flags: 0x010 (ACK)
Window: 502
[Calculated window size: 64256]
[Window size scaling factor: 128]
Checksum: 0x069b [unverified]
[Checksum Status: Unverified]
Urgent Pointer: 0
Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
[Timestamps]
[SEQ/ACK analysis]
TCP payload (1448 bytes)

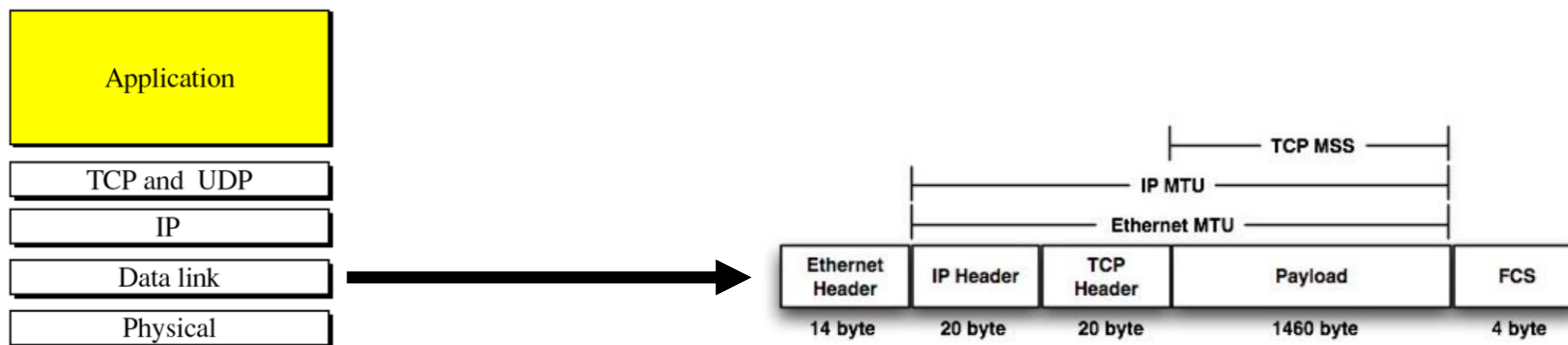
6

Transmission Control Protocol, Src Port: 60162, Dst Port: 8080, Seq: 2449, Ack: 1, Len: 552

Source Port: 60162
Destination Port: 8080
[Stream index: 0]
[Conversation completeness: Complete, WITH_DATA (31)]
[TCP Segment Len: 552]
Sequence Number: 2449 (relative sequence number)
Sequence Number (raw): 2608006347
[Next Sequence Number: 3001 (relative sequence number)]
Acknowledgment Number: 1 (relative ack number)
Acknowledgment number (raw): 4081004643
1000 = Header Length: 32 bytes (8)
Flags: 0x018 (PSH, ACK)
Window: 502
[Calculated window size: 64256]
[Window size scaling factor: 128]
Checksum: 0x4bad [unverified]
[Checksum Status: Unverified]
Urgent Pointer: 0
Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
[Timestamps]
[SEQ/ACK analysis]
TCP payload (552 bytes)

연결지향형 소켓(SOCK_STREAM) vs 비연결지향형 소켓(SOCK_DGRAM)

- TCP Result - MTU



```

Transmission Control Protocol, Src Port: 60162, Dst Port: 8080, Seq: 1001, Ack: 1, Len: 1448
Source Port: 60162
Destination Port: 8080
[Stream index: 0]
[Conversation completeness: Complete, WITH_DATA (31)]
[TCP Segment Len: 1448]
Sequence Number: 1001 (relative sequence number)
Sequence Number (raw): 2608004899
[Next Sequence Number: 2449 (relative sequence number)]
Acknowledgment Number: 1 (relative ack number)
Acknowledgment number (raw): 4081004643
1000 .... = Header Length: 32 bytes (8)
> Flags: 0x010 (ACK)
Window: 502
[Calculated window size: 64256]
[Window size scaling factor: 128]
Checksum: 0x069b [unverified]
[Checksum Status: Unverified]
Urgent Pointer: 0
> Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
> [Timestamps]
> [SEQ/ACK analysis]
TCP payload (1448 bytes)
    
```

5

```

Transmission Control Protocol, Src Port: 60162, Dst Port: 8080, Seq: 2449, Ack: 1, Len: 552
Source Port: 60162
Destination Port: 8080
[Stream index: 0]
[Conversation completeness: Complete, WITH_DATA (31)]
[TCP Segment Len: 552]
Sequence Number: 2449 (relative sequence number)
Sequence Number (raw): 2608006347
[Next Sequence Number: 3001 (relative sequence number)]
Acknowledgment Number: 1 (relative ack number)
Acknowledgment number (raw): 4081004643
1000 .... = Header Length: 32 bytes (8)
> Flags: 0x018 (PSH, ACK)
Window: 502
[Calculated window size: 64256]
[Window size scaling factor: 128]
Checksum: 0x4bad [unverified]
[Checksum Status: Unverified]
Urgent Pointer: 0
> Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
> [Timestamps]
> [SEQ/ACK analysis]
TCP payload (552 bytes)
    
```

6

= TCP는 보낼 데이터를 바이트 단위로 보고 보낼 수 있는 최대한으로 보낸다는 점을 알 수 있음

연결지향형 소켓(SOCK_STREAM) vs 비연결지향형 소켓(SOCK_DGRAM)

- UDP Result

```
● smalldragon@smalldragon-desktop:~/Workspace/network_programming_2023/week4$ ./udp_server3 8080
rcv_len: 1000 total_len: 1000
sleep 1s...
rcv_len: 1000 total_len: 2000
sleep 1s...
rcv_len: 1000 total_len: 3000
sleep 1s...
● smalldragon@smalldragon-desktop:~/Workspace/network_programming_2023/week4$
```

UDP Server

[illegible]

UDP Client

= 한번의 recvfrom()으로는 하나의 데이터밖에 받지 못하는 걸 알 수 있음

연결지향형 소켓(SOCK_STREAM) vs 비연결지향형 소켓(SOCK_DGRAM)

- UDP Result

```
for(int i=0; i<3; i++) {
    memset(rcvbuf, 0, sizeof(rcvbuf));
    int rcvlen = recvfrom(sockfd, rcvbuf, sizeof(rcvbuf), 0, (struct sockaddr*)&serv_addr, &len);
    printf("%s %d\n", rcvbuf, rcvlen);
}
```

```
● smalldragon@smalldragon-desktop:~/Workspace/network_programming_2023/week4$ ./udp_server3 8080
rcv_len: 1000 total_len: 1000
sleep 1s...
rcv_len: 1000 total_len: 2000
sleep 1s...
rcv_len: 1000 total_len: 3000
sleep 1s...
● smalldragon@smalldragon-desktop:~/Workspace/network_programming_2023/week4$
```

[illegible]

UDP Server

UDP Client

= sendto()를 호출한 만큼 recvfrom()으로 처리되는 것을 볼 수 있음

연결지향형 소켓(SOCK_STREAM) vs 비연결지향형 소켓(SOCK_DGRAM)

- UDP Result

```
#define MAX 1000
#define RMAX 30
```

```

PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL
● smalldragon@smalldragon-desktop:~/Workspace/network_programming_2023/week4$ ./udp_server3 8080
rcv_len: 1000 total_len: 1000
sleep 1s...
rcv_len: 1000 total_len: 2000
sleep 1s...
rcv_len: 1000 total_len: 3000
sleep 1s...
○ smalldragon@smalldragon-desktop:~/Workspace/network_programming_2023/week4$ █

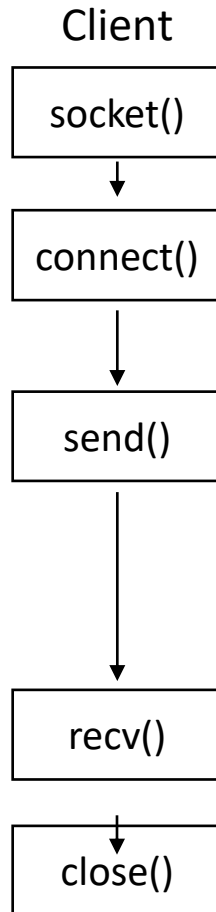
```

UDP Server

```
● smalldragon@smalldragon-desktop: ~/Workspace/network_programming_2023/week4$ ./udp_client3 127.0.0.1 8080
send 0
send 1
send 2
MMMMMMMMMMMMMMMMMMMMMMMMMMMMMM 30
MMMMMMMMMMMMMMMMMMMMMMMMMMMMMM 30
MMMMMMMMMMMMMMMMMMMMMMMMMMMMMM 30
○ smalldragon@smalldragon-desktop: ~/Workspace/network_programming_2023/week4$
```

UDP Client

Connected UDP Client



```

if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
    perror("socket creation failed");
    return -1;
}

servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = inet_addr(argv[1]);
servaddr.sin_port = htons(atoi(argv[2]));

if (connect(sockfd, (const struct sockaddr*)&servaddr, sizeof(servaddr)) < 0) {
    perror("connect error");
    return -1;
}

while(1) {
    fputs("Input message(Q to quit): ", stdout);
    fgets(buf, BUF_SIZE, stdin);

    send(sockfd, buf, strlen(buf), 0);
    //write(sockfd, buf, strlen(buf));

    if(!strcmp(buf, "q\n") || !strcmp(buf, "Q\n")) {
        break;
    }

    str_len = recv(sockfd, buf, sizeof(buf), 0);
    //str_len = read(sockfd, buf, sizeof(buf));
    buf[str_len-1] = 0;

    printf("Message from server: %s\n", buf);
}
close(sockfd);
return 0;
  
```

1. connect()는 TCP일 때 3way handshake를 진행함 (SYN 전송)
2. UDP 소켓일 경우, 단순히 socket에 서버 주소 정보만 기록함

**sys/socket.h*

```

/* Open a connection on socket FD to peer at ADDR (which LEN bytes Long).
   For connectionless socket types, just set the default address to send to
   and the only address from which to accept transmissions.
   Return 0 on success, -1 for errors.

   This function is a cancellation point and therefore not marked with
   __THROW. */
extern int connect (int __fd, __CONST_SOCKADDR_ARG __addr, socklen_t __len);
  
```

```

/* Give the socket FD the local address ADDR (which is LEN bytes Long). */
extern int bind (int __fd, __CONST_SOCKADDR_ARG __addr, socklen_t __len)
    __THROW;
  
```


02

Connected UDP

실행 결과

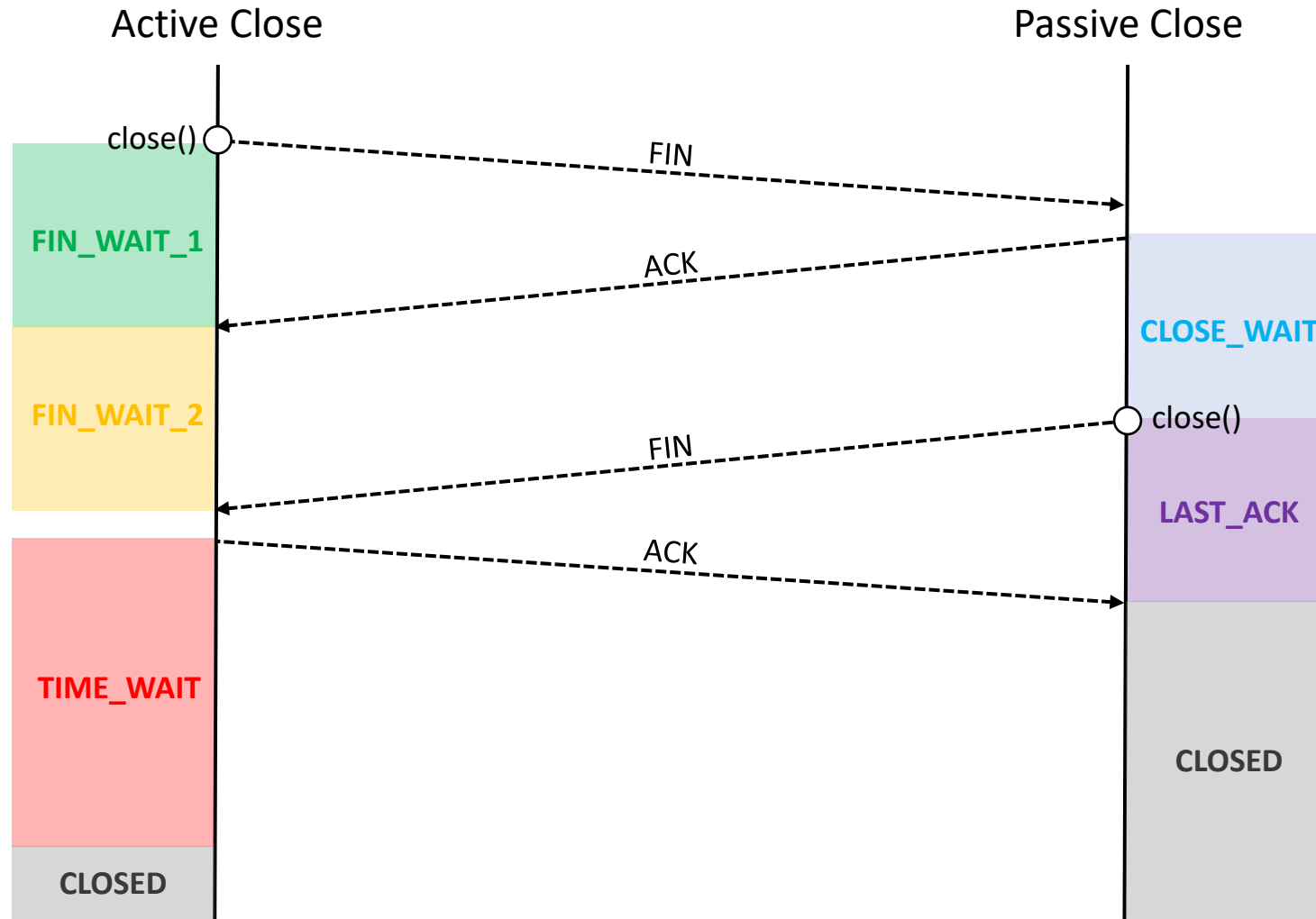
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket3/connected$ ./server 8080
smalldragon@SD-DESKTOP:~/Workspace/s
smalldragon@SD-DESKTOP:~/Workspace/socket/socket3/connected$ █
```

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket3/connected$ ./client 127.0.0.1 8080
Input message(Q to quit): Hello
Message from server: Hello
Input message(Q to quit): Connected Socket
Message from server: Connected Socket
Input message(Q to quit): q
smalldragon@SD-DESKTOP:~/Workspace/s
smalldragon@SD-DESKTOP:~/Workspace/socket/socket3/connected$ █
```

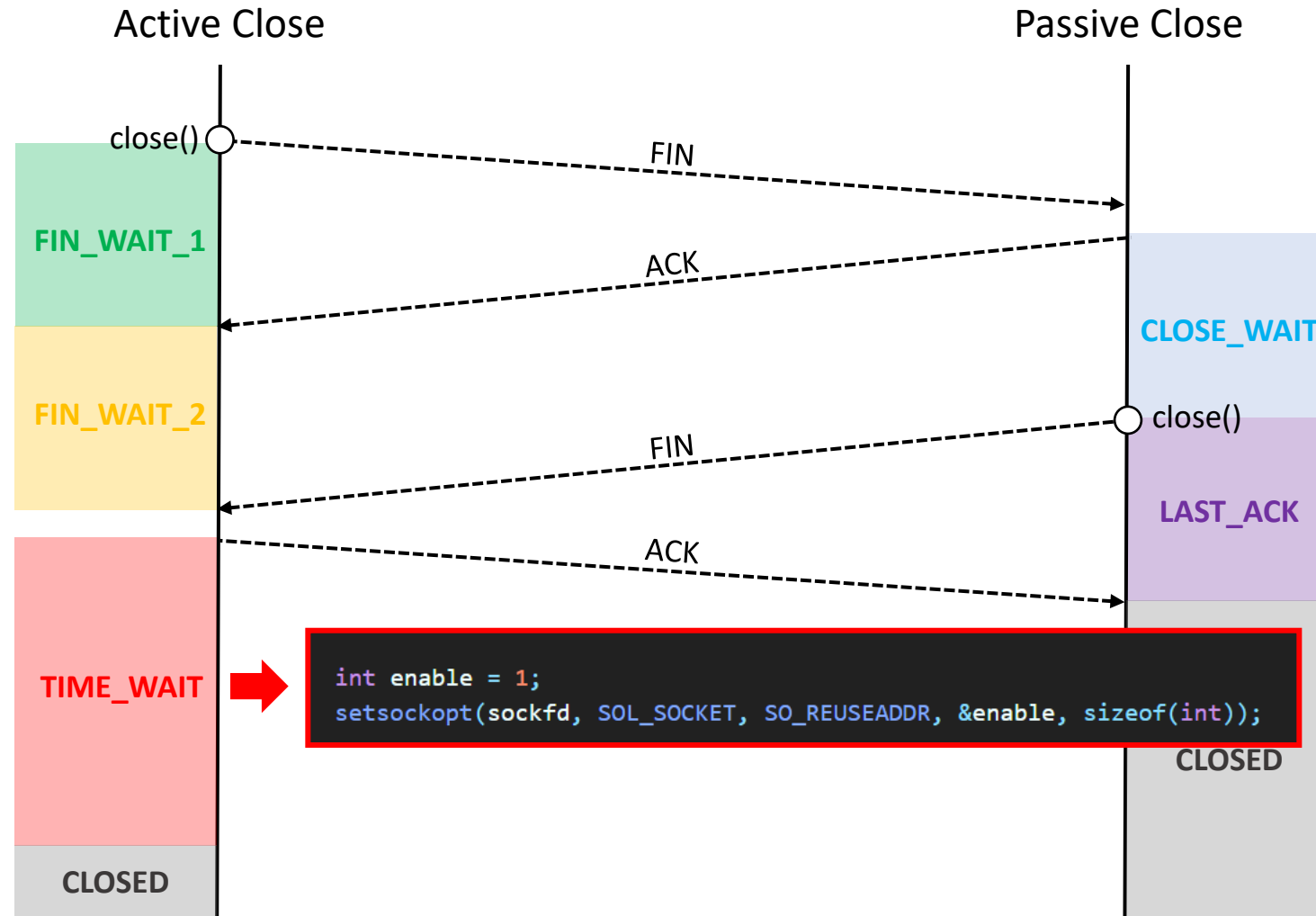
03 TCP Half Close

4way Handshake



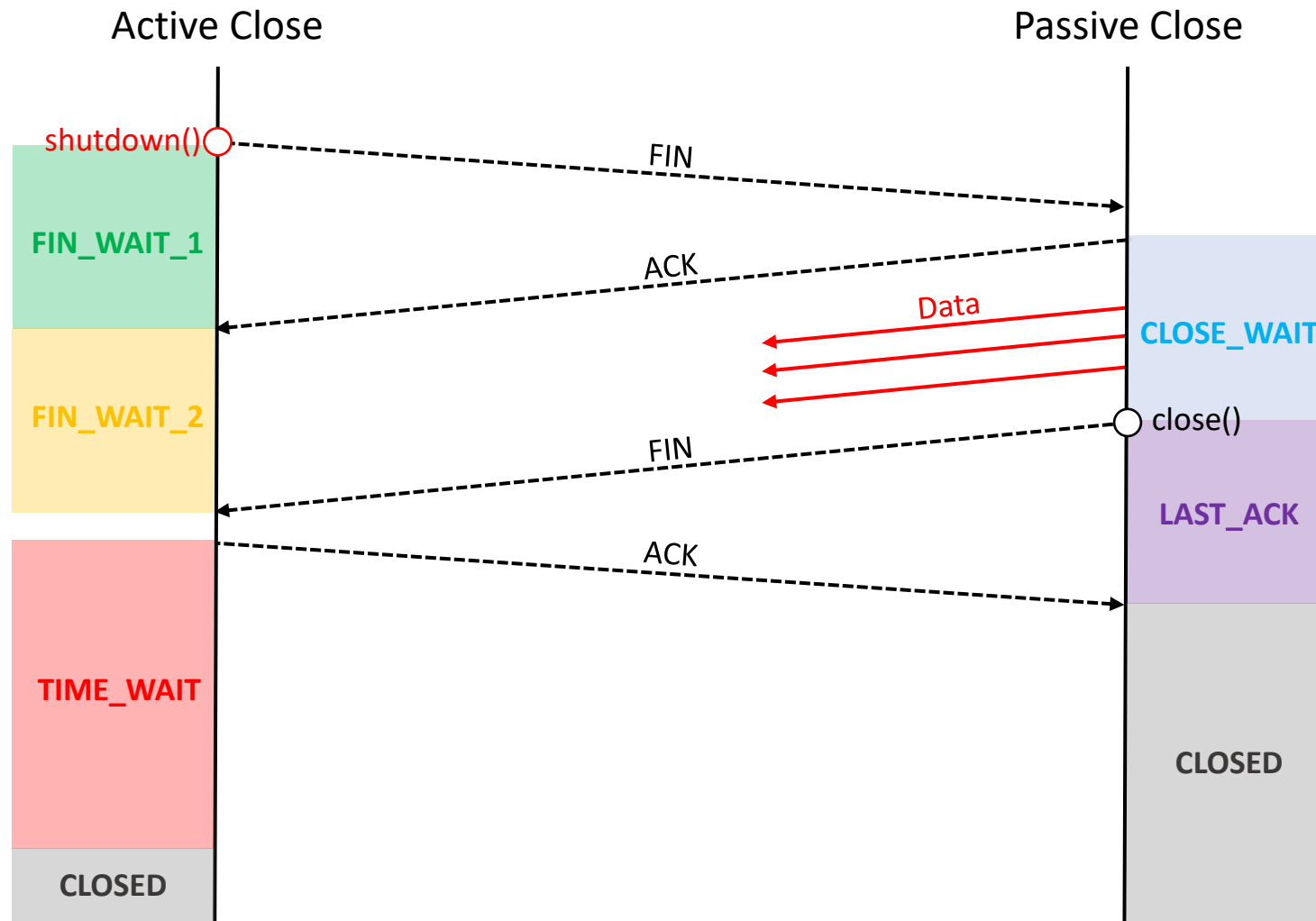
03 TCP Half Close

4way Handshake



03 TCP Half Close

Half close



Half-close 기반 파일 전송 프로그램



file_server_win.c 의 일부

```
while(1)
{
    read_cnt=fread((void*)buf, 1, BUF_SIZE, fp);
    if(read_cnt<BUF_SIZE)
    {
        write(clnt_sd, buf, read_cnt);
        break;
    }
    write(clnt_sd, buf, BUF_SIZE);
}

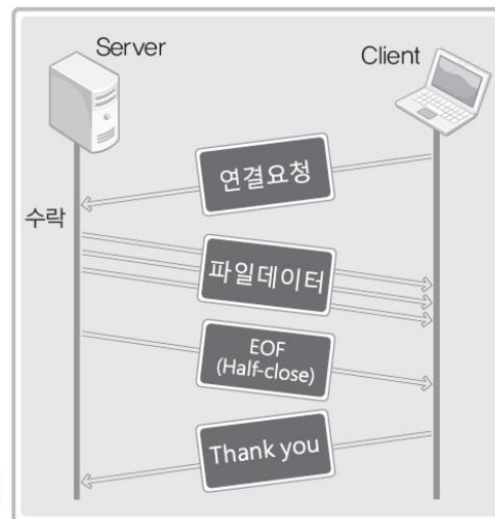
shutdown(clnt_sd, SHUT_WR);
read(clnt_sd, buf, BUF_SIZE);
printf("Message from client: %s \n", buf);

fclose(fp);
close(clnt_sd); close(serv_sd);
```

```
while((read_cnt=read(sd, buf, BUF_SIZE ))!=0)
    fwrite((void*)buf, 1, read_cnt, fp);

puts("Received file data");
write(sd, "Thank you", 10);
fclose(fp);
close(sd);
```

file_client_win.c 의 일부



Half-close가 필요한 상황의 연출

Half close Server

```
FILE* fp = fopen("text.txt", "r");  
  
int read_cnt;  
while(1) {  
    read_cnt = fread((void*)buf, 1, BUF_SIZE, fp);  
    if(read_cnt < BUF_SIZE) {  
        write(cSockfd, buf, read_cnt);  
        break;  
    }  
    write(cSockfd, buf, BUF_SIZE);  
}  
  
shutdown(cSockfd, SHUT_WR);  
read(cSockfd, buf, BUF_SIZE);  
printf("Message from Client: %s \n", buf);  
  
fclose(fp);  
close(cSockfd);  
close(sockfd);  
return 0;
```

1. `fopen()` : 첫 매개변수의 이름으로 된 파일을 열어서
파일포인터 반환

2. 두 번째 매개변수로 파일을 읽을지 쓸지를 결정함 (r, w, ...)

1. `fread()` : 파일을 1바이트로 `BUF_SIZE`만큼 읽어서 `buf`에 저장

1. `fclose()` : 파일닫기

Half close Server

```
FILE* fp = fopen("text.txt", "r");

int read_cnt;
while(1) {
    read_cnt = fread((void*)buf, 1, BUF_SIZE, fp);
    if(read_cnt < BUF_SIZE) {
        write(cSockfd, buf, read_cnt);
        break;
    }
    write(cSockfd, buf, BUF_SIZE);
}
```

```
shutdown(cSockfd, SHUT_WR);
read(cSockfd, buf, BUF_SIZE);
printf("Message from Client: %s \n", buf);
```

- 1. shutdown() : Write Stream을 닫고 종료 (Half close)
- 2. 소켓을 통해 read만 가능

```
fclose(fp);
close(cSockfd);
close(sockfd);
return 0;
```

- 1. 여기서 close()는 OS가 잡아 둔 소켓 자원을 해제하는 용도

Half close Client

1. fopen() : 파일을 쓰기 형태로 열고 파일포인터 반환
2. 이 때 첫 번째 매개변수의 이름을 가진 파일이 없다면 생성

```
FILE* fp = fopen("copy.txt", "w");
```

```
if (connect(sockfd, (const struct sockaddr*)&servaddr, sizeof(servaddr)) < 0) {  
    perror("connect error");  
    return -1;  
}
```

```
int read_cnt;  
while((read_cnt = read(sockfd, buf, BUF_SIZE)) != 0)  
    fwrite((void*)buf, 1, read_cnt, fp);
```

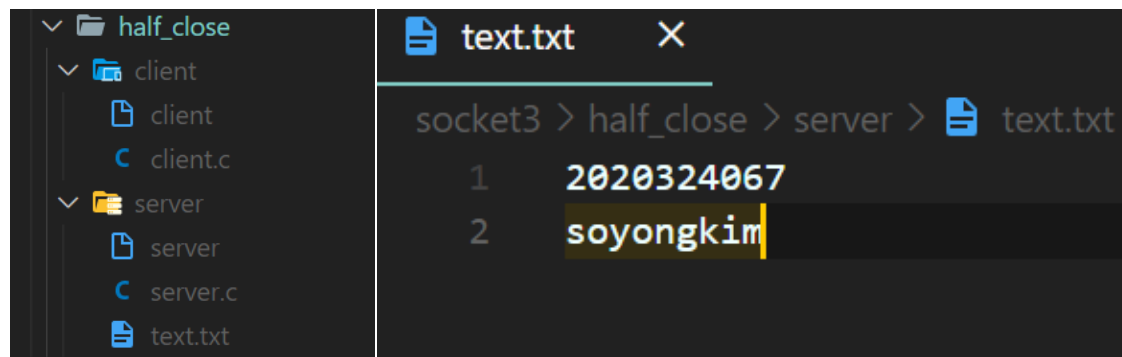
1. fwrite() : 파일에 read()한 데이터를 씀

```
puts("Received file data");  
write(sockfd, "Thank you", 10);
```

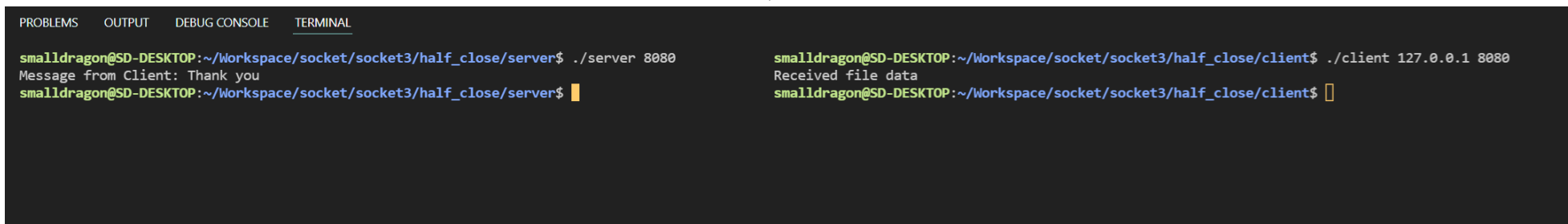
```
fclose(fp);  
close(sockfd);  
return 0;
```


03 TCP Half Close

실행 결과

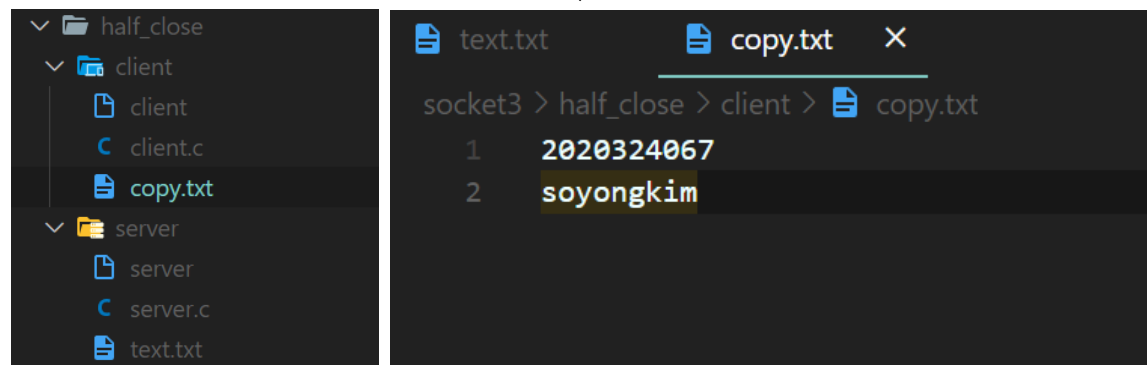


```
socket3 > half_close > server > text.txt
1 2020324067
2 soyongkim
```



```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket3/half_close/server$ ./server 8080
Message from Client: Thank you
smalldragon@SD-DESKTOP:~/Workspace/socket/socket3/half_close/server$

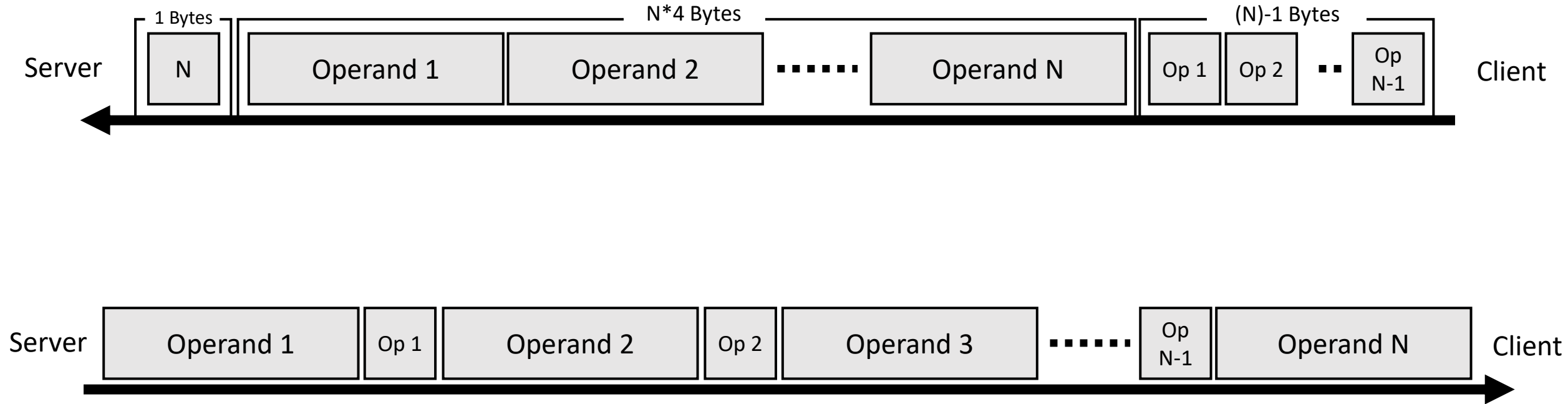
smalldragon@SD-DESKTOP:~/Workspace/socket/socket3/half_close/client$ ./client 127.0.0.1 8080
Received file data
smalldragon@SD-DESKTOP:~/Workspace/socket/socket3/half_close/client$
```



```
socket3 > half_close > client > copy.txt
1 2020324067
2 soyongkim
```

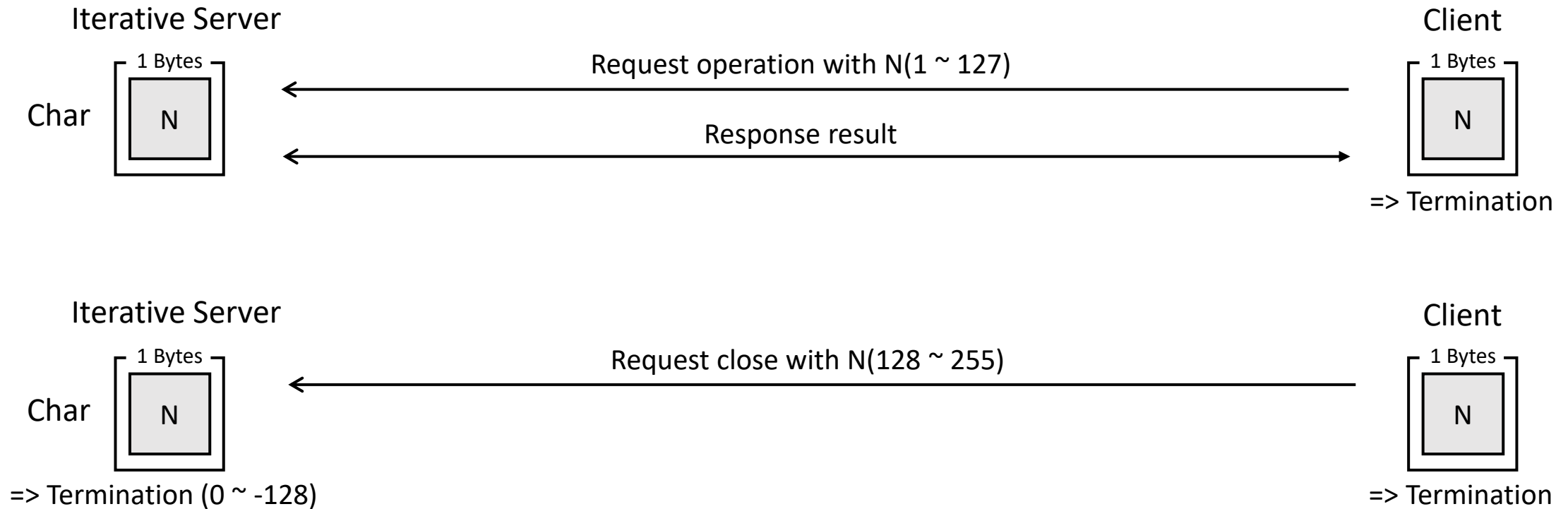
04 Network assignment #3

UDP Advanced Calculator Program



04 Network assignment #3

UDP Advanced Calculator Program



UDP Advanced Calculator Program

- client.c

1. 클라이언트 실행 시 main 함수의 매개변수로 포트번호와 서버의 IP주소를 받아서 실행

Ex) ./client 8080 127.0.0.1 (순서 확인)

2. UDP 소켓을 생성하고, 매개변수를 활용하여 서버에게 연결 요청
3. 표준입력을 통해 operand count와 이 수만큼 operand를 받고 (operand count)-1 의 수만큼 operator(1바이트)를 입력 받음

1. 이 때 operator는 +, -, *로 한정

2. Operand count를 입력 받을 시 표준출력으로 *Operand count:* 를 출력

3. Operand와 Operator 입력을 받을 시 표준출력으로 각 이름들과 함께 입력을 받는 순으로 0부터 번호를 같이 출력

1. Operand 0: ,Operand 1:

2. Operator 0: ,Operator 1:

4. 표준입력으로 받은 데이터를 char 배열로 받아 한번에 전송

1. 서버에게 operand count를 1바이트로 보내고 이 수만큼 operand를 4바이트로 전송하고, (operation count)-1 의 수만큼 operator(1바이트)를 한 번의 sendto()로 전송

Ex) (3 | 4, 5, 7 | +, -) 형태를 char 배열을 통해 **한번에 sendto()로 전송 – connected UDP X**

5. 결과를 받게 되면 이를 표준출력으로 Operation result: 와 함께 출력하고 소켓을 닫고 종료
6. operand count를 전송할 때 char 기준 0보다 작거나 같은 값을 전송하는 케이스의 경우, 이 값을 서버에게 전송하고 추가적인 표준입력없이 소켓을 닫고 종료

UDP Advanced Calculator Program

- server.c

1. 서버 실행 시 main 함수의 매개변수로 포트번호를 받아서 실행

Ex) ./server 8080

2. UDP 소켓을 생성하고, INADDR_ANY와 매개변수로 받은 포트번호로 소켓 바인드 진행
3. 서버는 recvfrom()을 통해 클라이언트의 연산데이터를 수신함
4. 서버는 Iterative Server 형태로 구현하고 종료 조건은 다음과 같음

1. 클라이언트가 1바이트로 전송한 operand count 정보를 char 형태로 저장

2. Operand count 정보가 char 기준으로 0보다 작거나 같은 숫자가 나온다면 서버는 반복을 빠져나와서 소켓을 닫고 프로그램 종료

1. 이 때 server close(operand count)형태로 표준 출력을 하고 종료

5. 서버는 클라이언트가 전송한 데이터를 기반으로 데이터를 아래와 같이 연산함

1. 클라이언트의 operand count 수만큼 operand를 받고 (operand count)-1 수만큼 연산자를 받음

2. 클라이언트 전송한 operand를 앞에서부터 차례대로 operator로 계산하며, 이 때 *의 연산순위를 우선적으로 두지 않고 앞에서부터 계산

Ex) [3 | 4, 5, 7 | +, -] > [4 + 5 - 7], [3 | 4, 5, 7 | +, *] > [(4 + 5) * 7]

5. 계산된 결과를 sendto()로 클라이언트에게 전송 – connected UDP X

1. 전송하기 전에 클라이언트에서 출력된 결과와 똑같이 표준 출력을 하고 전송

2. 프로그램 종료 조건에서는 똑같은 필요 없음

6. 반복된 클라이언트 요청에도 문제없이 결과가 전송되어야 함

04

Network assignment #3

UDP Advanced Calculator Program

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket3/udp_calculator$ ./server 8080
Operand count: 3
Operand 0: 6
Operand 1: 3
Operand 2: 2
Operator 0: +
Operator 1: -
Operation result: 7
Operand count: 4
Operand 0: 10
Operand 1: 20
Operand 2: 30
Operand 3: 40
Operator 0: +
Operator 1: +
Operator 2: -
Operation result: 20
Operand count: 2
Operand 0: 5
Operand 1: 2
Operator 0: *
Operation result: 10
Server close(-128)
smalldragon@SD-DESKTOP:~/Workspace/socket/socket3/udp_calculator$
```

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket3/udp_calculator$ ./server 8080
Server close(-127)
smalldragon@SD-DESKTOP:~/Workspace/socket/socket3/udp_calculator$ ./server 8080
Server close(-1)
smalldragon@SD-DESKTOP:~/Workspace/socket/socket3/udp_calculator$
```

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket3/udp_calculator$ ./client 8080 127.0.0.1
Operand count: 3
Operand 0: 6
Operand 1: 3
Operand 2: 2
Operator 0: +
Operator 1: -
Operation result: 7
smalldragon@SD-DESKTOP:~/Workspace/socket/socket3/udp_calculator$ ./client 8080 127.0.0.1
Operand count: 4
Operand 0: 10
Operand 1: 20
Operand 2: 30
Operand 3: 40
Operator 0: +
Operator 1: +
Operator 2: -
Operation result: 20
smalldragon@SD-DESKTOP:~/Workspace/socket/socket3/udp_calculator$ ./client 8080 127.0.0.1
Operand count: 2
Operand 0: 5
Operand 1: 2
Operator 0: *
Operation result: 10
smalldragon@SD-DESKTOP:~/Workspace/socket/socket3/udp_calculator$ ./client 8080 127.0.0.1
Operand count: 128
smalldragon@SD-DESKTOP:~/Workspace/socket/socket3/udp_calculator$
```

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket3/udp_calculator$ ./client 8080 127.0.0.1
Operand count: 129
smalldragon@SD-DESKTOP:~/Workspace/socket/socket3/udp_calculator$ ./client 8080 127.0.0.1
Operand count: 255
smalldragon@SD-DESKTOP:~/Workspace/socket/socket3/udp_calculator$
```

UDP Advanced Calculator Program

- 참고사항

1. 서버가 의도한 것 이외의 값을 받는 케이스를 예외처리 할 필요 없음
2. Operand count가 0~255까지만 정상 동작하면 됨
3. 총 계산 결과 값이 int 형을 넘어가서 overflow가 발생할 경우도 예외처리 할 필요 없음
4. UDP 소켓을 통해 구현할 것. TCP 구현물로 제출할 경우 점수 없음
5. 과제에서 의도한 대로 데이터를 주고받고 이를 출력하는 방식이 아닌, 겉으로 출력 결과만 똑같이 보인다면 점수 없음
6. 과제 관련 문의 : thdyd324@gmail.com

- 제출관련

1. 서버 프로그램은 server.c, 클라이언트 프로그램은 client.c로 명명하여 과제 진행
2. 빌드 시(gcc) Warning이 발생해서는 안됨
3. 제출 시 두 파일을 “자신의 학번.tar” 파일로 제출

Ex) 2020324067.tar

~/Workspace/socket1/(server.c, client.c)

```
smalldragon@DESKTOP-PMPPMH:~/Workspace$ tar cvf 2020324067.tar -C socket1 server.c client.c
server.c
client.c
```

압축파일명	폴더명	파일명	파일명
2020324067.tar	socket1	server.c	client.c

4. 과제는 10점 만점
5. 제출 기한: 2023.04.07(금) PM 11:59
6. 지각 제출 허용: 2022.04.11(화) PM 11:59 / 하루 늦을 때 마다 2점 씩 감점
지각제출 시 보낼 이메일: pjm9562@naver.com
7. 기한 안에 아예 제출을 하지 않았을 시 점수 없음