Part 03. 리눅스 소켓 프로그래밍

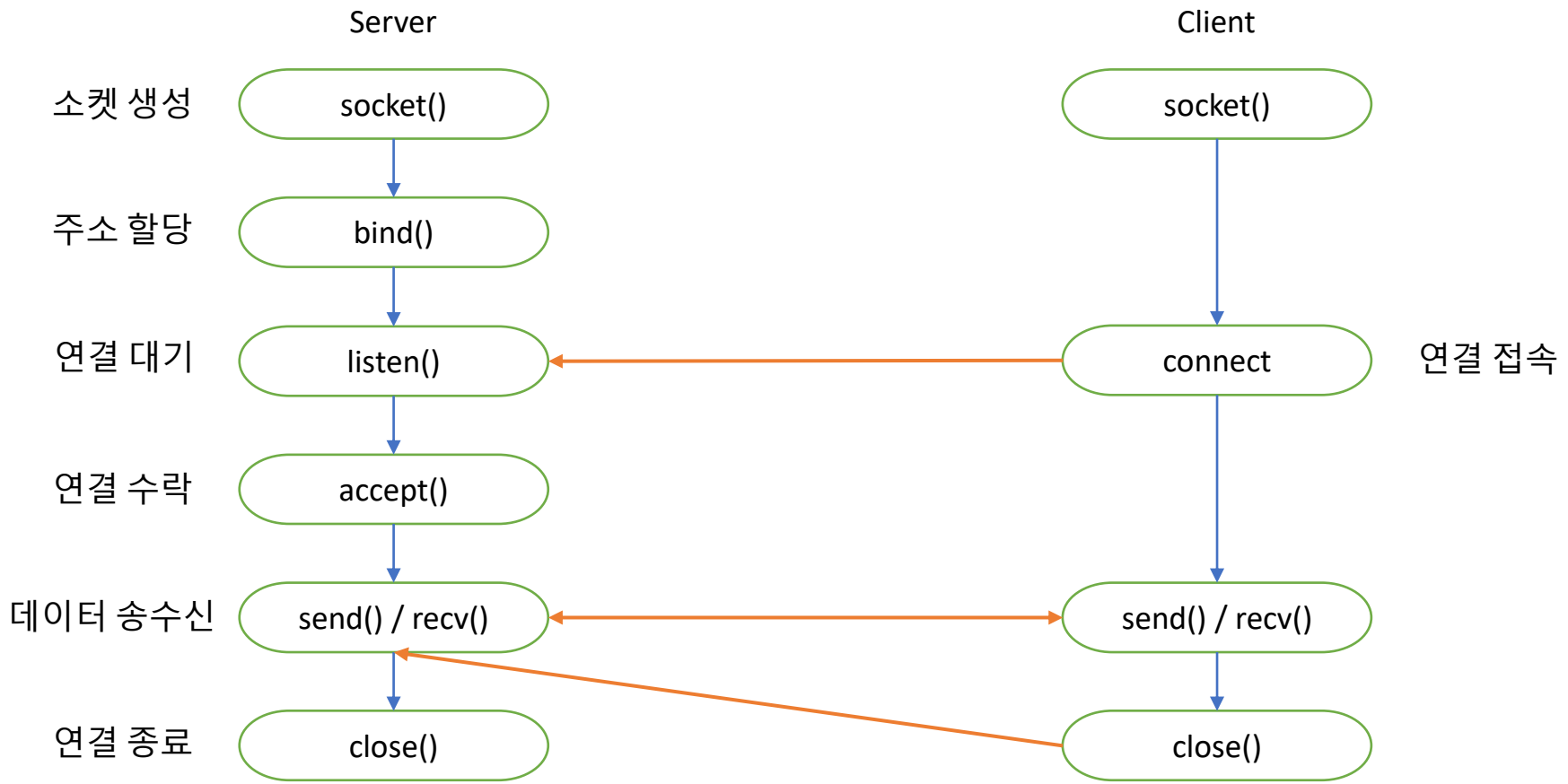
Chapter 02. TCP 소켓 통신

진행 순서

Chapter 02_01	TCP 소켓 통신 개요
Chapter 02_02	TCP 소켓 통신 함수
Chapter 02_03	TCP 소켓 통신 실습

Chapter 02_01 TCP 소켓 통신 개요

TCP 소켓 통신 함수 호출 흐름

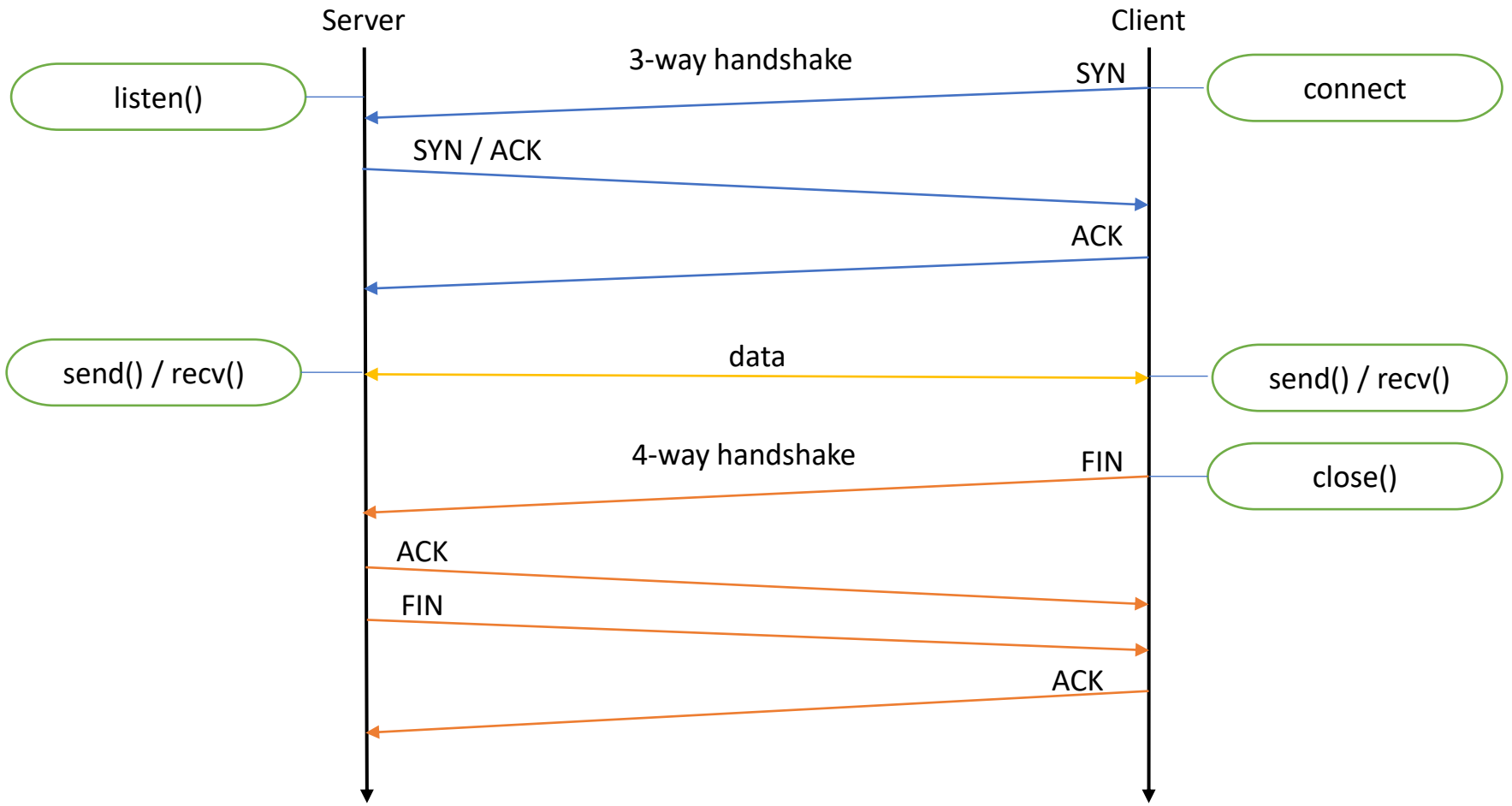


Chapter 02_01 TCP 소켓 통신 개요

TCP 소켓 통신 패킷 흐름

TCP segment header

Offsets	Octet	0								1								2								3							
Octet	Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset				Reserved 0 0 0				NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size														
16	128	Checksum																								Urgent pointer (if URG set)							
20	160	Options (if *data offset* > 5. Padded at the end with "0" bytes if necessary.)																															
...	...	...																															



Chapter 02_02 TCP 소켓 통신 함수

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

int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

socket()을 사용해 소켓을 만들면 이름 공간(주소 패밀리)에 존재하지만 할당 된 주소가 없습니다.
bind ()는 addr에 의해 지정된 주소를 파일 디스크립터 sockfd가 참조하는 소켓에 할당합니다.
addrlen은 addr이 가리키는 주소 구조의 크기를 바이트 단위로 지정합니다.
addr 인수에 전달 된 실제 구조는 주소 패밀리에 따라 다릅니다. sockaddr 구조는 다음과 같이 정의됩니다.

```
struct sockaddr {
        sa_family_t sa_family;
        char        sa_data[14];
}
```

이 구조의 유일한 목적은 컴파일러 경고를 피하기 위해 addr에 전달 된 구조 포인터를 캐스트하는 것입니다.

```
AF_INET      struct sockaddr_in
AF_INET6     struct sockaddr_in6
AF_UNIX      struct sockaddr_un
```

성공하면 0이 리턴됩니다. 오류 발생 시 -1이 리턴되고 errno가 적절하게 설정됩니다.

Chapter 02_02 TCP 소켓 통신 함수

```
struct sockaddr_in {
   sa_family_t    sin_family; /* address family: AF_INET */
   in_port_t      sin_port;   /* port in network byte order */
   struct in_addr sin_addr;   /* internet address */
};

/* Internet address. */
struct in_addr {
   uint32_t       s_addr;     /* address in network byte order */
};
```

```
struct sockaddr_un {
   sa_family_t sun_family;            /* AF_UNIX */
   char        sun_path[108];         /* Pathname */
};
```

```
struct sockaddr_in6 {
   sa_family_t     sin6_family;   /* AF_INET6 */
   in_port_t       sin6_port;     /* port number */
   uint32_t        sin6_flowinfo; /* IPv6 flow information */
   struct in6_addr sin6_addr;     /* IPv6 address */
   uint32_t        sin6_scope_id; /* Scope ID (new in 2.4) */
};

struct in6_addr {
   unsigned char   s6_addr[16];   /* IPv6 address */
};
```

Chapter 02_02 TCP 소켓 통신 함수

The following example shows how to bind a stream socket in the UNIX (AF_UNIX) domain, and accept connections:

```
#include <sys/socket.h>
#include <sys/un.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#define MY_SOCK_PATH "/somepath"
#define LISTEN_BACKLOG 50

#define handle_error(msg) \
   do { perror(msg); exit(EXIT_FAILURE); } while (0)

int main(int argc, char *argv[])
{
   int sfd, cfd;
   struct sockaddr_un my_addr, peer_addr;
   socklen_t peer_addr_size;

   sfd = socket(AF_UNIX, SOCK_STREAM, 0);
   if (sfd == -1)
      handle_error("socket");
```

```
   memset(&my_addr, 0, sizeof(struct sockaddr_un)); /* Clear structure */
   my_addr.sun_family = AF_UNIX;
   strncpy(my_addr.sun_path, MY_SOCK_PATH,
   sizeof(my_addr.sun_path) - 1);

   if (bind(sfd, (struct sockaddr *) &my_addr, sizeof(struct sockaddr_un)) == -1)
      handle_error("bind");

   if (listen(sfd, LISTEN_BACKLOG) == -1)
      handle_error("listen");

   /* Now we can accept incoming connections one at a time using accept(2) */
   peer_addr_size = sizeof(struct sockaddr_un);
   cfd = accept(sfd, (struct sockaddr *) &peer_addr, &peer_addr_size);
   if (cfd == -1)
      handle_error("accept");

   /* Code to deal with incoming connection(s)... */

   /* When no longer required, the socket pathname, MY_SOCK_PATH
      should be deleted using unlink(2) or remove(3) */
}
```

Chapter 02_02 TCP 소켓 통신 함수

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

int listen(int sockfd, int backlog);
```

listen()은 sockfd가 참조하는 소켓을 패시브 소켓, 즉 accept()를 사용하여 들어오는 연결 요청을 승인하는 데 사용될 소켓으로 표시합니다.
sockfd 인수는 SOCK_STREAM 또는 SOCK_SEQPACKET 유형의 소켓을 참조하는 파일 디스크립터입니다.
backlog 인수는 sockfd에 대해 보류 중인 연결 큐가 커질 수 있는 최대 길이를 정의합니다.
큐가 가득 찼을 때 연결 요청이 도착하면 클라이언트는 ECONNREFUSED 표시로 오류를 수신하거나 기본 프로토콜이 재전송을 지원하는 경우 나중에 연결 시도가 성공하도록 요청이 무시 될 수 있습니다.

백로그 값은 완전한 연결 즉 established 상태를 의미하며, 불완전한 연결, 즉 accept 대기중인 길이는 /proc/sys/net/ipv4/tcp_max_syn_backlog 값으로 설정할 수 있습니다.
백 로그 인수가 /proc/sys/net/core/somaxconn의 값보다 큰 경우 해당 값으로 자동으로 잘립니다. 이 파일의 기본 값은 128입니다.

성공하면 0이 반환됩니다. 오류가 발생하면 -1이 반환되고 errno가 적절하게 설정됩니다.

Chapter 02_02 TCP 소켓 통신 함수

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

int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

accept() 시스템 호출은 연결 기반 소켓 유형 (SOCK_STREAM)과 함께 사용됩니다. 수신 소켓 sockfd에 대해 보류중인 연결 큐에서 첫 번째 연결 요청을 추출하여 연결된 새 소켓을 생성하고 해당 소켓을 참조하는 새 파일 디스크립터를 리턴합니다.

인수 addr은 sockaddr 구조에 대한 포인터입니다. 이 구조는 통신 계층에 알려진 피어 소켓의 주소로 채워집니다. 반환 된 주소 addr의 정확한 형식은 소켓의 주소 패밀리에 의해 결정됩니다. addr이 NULL이면 아무것도 채워지지 않습니다. 이 경우 addrlen이 사용되지 않으며 NULL이어야 합니다.

addrlen 인수는 value-result 인수입니다. 호출자는 addr이 가리키는 구조의 크기(바이트)를 포함하도록 초기화 해야합니다. 반환되면 피어 주소의 실제 크기가 포함됩니다.

제공된 버퍼가 너무 작으면 리턴 된 주소가 잘립니다. 이 경우 addrlen은 호출에 제공된 것보다 큰 값을 반환합니다.

큐에 보류중인 연결이 없고 소켓이 넌블로킹으로 표시되지 않은 경우 accept()는 연결이 있을 때까지 호출자를 차단합니다. 소켓이 넌블로킹으로 표시되고 대기중인 연결이 큐에 없으면 EAGAIN 또는 EWOULDBLOCK 오류와 함께 accept()가 실패합니다.

성공하면 허용 된 소켓의 파일 디스크립터인 음이 아닌 정수를 리턴합니다.
오류가 발생하면 -1이 반환되고 errno가 적절하게 설정됩니다.

Chapter 02_02 TCP 소켓 통신 함수

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

int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

connect() 시스템 호출은 파일 디스크립터 sockfd가 참조하는 소켓을 addr이 지정한 주소에 연결합니다.
addrlen 인수는 addr의 크기를 지정합니다. addr의 주소 형식은 소켓 sockfd의 주소 공간에 의해 결정됩니다.

연결이 성공하면 0이 리턴됩니다. 오류가 발생하면 -1이 반환되고 errno가 적절하게 설정됩니다.

```
int sock;
struct sockaddr_in server;

sock = socket(AF_INET, SOCK_STREAM, 0);
if (sock == -1) {
                /* error */
}

server.sin_addr.s_addr = inet_addr(”100.100.100.1");
server.sin_family = AF_INET;
server.sin_port = htons(80);

if (connect(sock, (struct sockaddr *)&server , sizeof(server)) < 0) {
                /* error */
}
```

Chapter 02_02 TCP 소켓 통신 함수

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

ssize_t send(int sockfd, const void *buf, size_t len, int flags);
```

send() 시스템 호출은 다른 소켓으로 메시지를 전송하는 데 사용됩니다.
메시지는 buf에 있으며 길이는 len입니다.
성공 시 전송 된 바이트 수를 리턴합니다. 오류 발생 시 -1이 리턴되고 errno가 적절하게 설정됩니다.
len 보다 작은 값이 리턴된 경우 처리되지 않은 나머지 바이트를 계산하여 재전송해야 합니다.
flags는 전송 시 사용할 작동 플래그를 지정합니다.

flags

MSG_DONTROUTE 게이트웨이를 사용하여 패킷을 보내지 말고 직접 연결된 네트워크에서만
호스트에게 보내십시오. 이것은 일반적으로 진단 또는 라우팅 프로그램에 의해서만 사용됩니다.
이것은 라우팅하는 프로토콜 제품군에 대해서만 정의됩니다. 패킷 소켓은 그렇지 않습니다.

MSG_DONTWAIT 넌블록킹 작업을 가능하게 합니다.
작업이 차단되면 EAGAIN 또는 EWOULDBLOCK이 반환됩니다.

MSG_MORE 발신자에게 더 많은 데이터를 보낼 수 있습니다.

MSG_NOSIGNAL 스트림 지향 소켓의 피어가 연결을 닫은 경우 SIGPIPE 신호를 생성하지 않습니다.
EPIPE 오류는 계속 반환됩니다.

MSG_OOB Out-Of-Band 데이터를 전송합니다.

Chapter 02_02 TCP 소켓 통신 함수

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

ssize_t recv(int sockfd, void *buf, size_t len, int flags);
```

recv() 호출은 소켓에서 메시지를 받는 데 사용됩니다.
성공적으로 완료되면 메시지 길이를 반환합니다. 메시지가 제공된 버퍼에 비해 너무 길면 메시지를 받는 소켓 유형에 따라 초과 바이트가 삭제 될 수 있습니다.
소켓에서 사용할 수 있는 메시지가 없으면 소켓이 넌블록킹 되지 않은 경우 메시지가 도착하기를 기다립니다.
넌블록킹의 경우 값 -1이 리턴되고 외부 변수 errno가 EAGAIN이나 EWOULDBLOCK으로 설정됩니다.

이 호출은 수신 된 바이트 수를 리턴하거나 오류가 발생한 경우 -1을 리턴합니다. (errno)
스트림 소켓 피어가 순서대로 종료를 수행하면 반환 값은 0 (EOF)이 됩니다.

flags	
MSG_OOB	Out-Of-Band 데이터를 수신합니다.
MSG_PEEK	수신 성공 이후에도 소켓 수신 버퍼 큐에서 데이터를 제거하지 않습니다.
MSG_TRUNK	버퍼보다 큰 데이터를 수신하는 경우 초과분은 모두 삭제합니다.
MSG_WAITALL	버퍼 크기가 다 채워질 때까지 대기합니다.

Chapter 02_03 TCP 소켓 통신 실습

server.c

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

#define handle_error(msg) \
        do { perror(msg); exit(EXIT_FAILURE); } while (0)

#define LISTEN_BACKLOG 50

int main(int argc, char *argv[])
{
        int sfd, cfd;
        int portnum;
        struct sockaddr_in saddr;
        struct sockaddr_in caddr;
        socklen_t caddr_len;
        int ret;
        char buf[1024] = {0,};

        if (argc < 2) {
                printf("Usage: %s <port>\n", argv[0]);
                exit(1);
        }

        portnum = atoi(argv[1]);

        sfd = socket(AF_INET, SOCK_STREAM, 0);
        if (sfd == -1)
                handle_error("socket");
}
```

```c
memset(&saddr, 0, sizeof(struct sockaddr_in));
saddr.sin_family = AF_INET;
saddr.sin_addr.s_addr = INADDR_ANY;
saddr.sin_port = htons(portnum);

if (bind(sfd, (struct sockaddr *)&saddr, sizeof(saddr)) == -1)
        handle_error("bind");

if (listen(sfd, LISTEN_BACKLOG) == -1)
        handle_error("listen");

caddr_len = sizeof(caddr);
cfd = accept(sfd, (struct sockaddr *)&caddr, &caddr_len);
if (cfd == -1)
        handle_error("accept");

printf("accepted host(IP: %s, Port: %d)\n",
        inet_ntoa(((struct sockaddr_in *)&caddr)->sin_addr),
        ntohs(((struct sockaddr_in *)&caddr)->sin_port));

while (1) {
        memset(buf, 0, sizeof(buf));
        ret = recv(cfd, buf, sizeof(buf), 0);
        if (ret == -1)
                handle_error("recv");
        else if (ret == 0)
                break;
        printf("recv data: %s\n", buf);
}

printf("finish.\n");
close(cfd);
return 0;
```

Chapter 02_03 TCP 소켓 통신 실습

client.c

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

#define handle_error(msg) \
        do { perror(msg); exit(EXIT_FAILURE); } while (0)


int main(int argc, char *argv[])
{
        int sockfd;
        int portnum;
        struct sockaddr_in saddr;
        socklen_t saddr_len;
        int ret;
        char buf[1024] = {0,};

        if (argc < 2) {
                printf("Usage: %s <ipaddr> <port>\n", argv[0]);
                exit(1);
        }

        portnum = atoi(argv[2]);

        sockfd = socket(AF_INET, SOCK_STREAM, 0);
        if (sockfd == -1)
                handle_error("socket");

        memset(&saddr, 0, sizeof(struct sockaddr_in));
        saddr.sin_family = AF_INET;
        saddr.sin_addr.s_addr = inet_addr(argv[1]);
        saddr.sin_port = htons(portnum);

        if (connect(sockfd, (struct sockaddr *)&saddr, sizeof(saddr)) == -1)
                handle_error("connect");

        while(1) {
                char buf[1024] = {0,};
                printf("Input message(.:quit): ");
                fgets(buf, sizeof(buf), stdin);
                if (buf[0] == '.')
                        break;
                if (send(sockfd, buf, strlen(buf), 0) == -1)
                        handle_error("send");
        }

        printf("finish.\n");
        close(sockfd);
        return 0;
}
```

Chapter 02_03 TCP 소켓 통신 실습

```
[root@localhost ch02]# gcc -g server.c -o server
[root@localhost ch02]# ./server
Usage: ./server <port>
[root@localhost ch02]# ./server 8888
accepted host(IP: 127.0.0.1, Port: 60612)
recv data: hello

recv data: this is test client

recv data: bye

finish.
[root@localhost ch02]#
```

```
[root@localhost ch02]# gcc -g client.c -o client
[root@localhost ch02]# ./client
Usage: ./client <ipaddr> <port>
[root@localhost ch02]# ./client 127.0.0.1 8888
Input message(.:quit): hello
Input message(.:quit): this is test client
Input message(.:quit): bye
Input message(.:quit): .
finish.
[root@localhost ch02]#
```