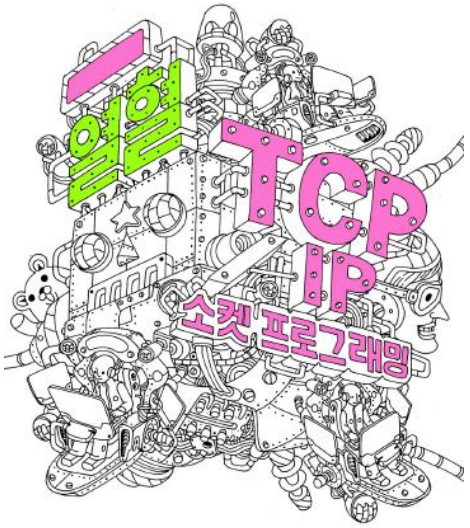
윤성우의 열혈 TCP/IP 소켓 프로그래밍

윤성우 저 열혈강의 TCP/IP 소켓 프로그래밍 개정판

Chapter 13. 다양한 입출력 함수들



Chapter 13-1. send & recv 입출력 함수

윤성우 저 열혈강의 TCP/IP 소켓 프로그래밍 개정판

리눅스에서의 send & recv

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

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

➡ 성공 시 전송된 바이트 수, 실패 시 -1 반환

- sockfd 데이터 전송 대상과의 연결을 의미하는 소켓의 파일 디스크립터 전달.
- buf 전송할 데이터를 저장하고 있는 버퍼의 주소 값 전달.
- nbytes 전송할 바이트 수 전달.
- flags 데이터 전송 시 적용할 다양한 옵션 정보 전달.

윈도우 기반 예제에서 사용해 온 send & recv 함수와 동일하다.

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

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

➡ 성공 시 수신한 바이트 수(단 EOF 전송 시 0), 실패 시 -1 반환

- sockfd 데이터 수신 대상과의 연결을 의미하는 소켓의 파일 디스크립터 전달.
- buf 수신된 데이터를 저장할 버퍼의 주소 값 전달.
- nbytes 수신할 수 있는 최대 바이트 수 전달.
- flags 데이터 수신 시 적용할 다양한 옵션 정보 전달.

send & recv 함수의 옵션과 그 의미

옵션(Option)	의 미	send	recv
MSG_OOB	긴급 데이터(Out-of-band data)의 전송을 위한 옵션.	●	●
MSG_PEEK	입력버퍼에 수신된 데이터의 존재유무 확인을 위한 옵션.		●
MSG_DONTROUTE	데이터 전송과정에서 라우팅(Routing) 테이블을 참조하지 않을 것을 요구하는 옵션, 따라서 로컬(Local) 네트워크상에서 목적지를 찾을 때 사용되는 옵션.	●	
MSG_DONTWAIT	입출력 함수 호출과정에서 블로킹 되지 않을 것을 요구하기 위한 옵션, 즉, 넌-블로킹(Non-blocking) IO의 요구에 사용되는 옵션.	●	●
MSG_WAITALL	요청한 바이트 수에 해당하는 데이터가 전부 수신될 때까지, 호출된 함수가 반환되는 것을 막기 위한 옵션		●

데이터의 전송에 사용되는 옵션정보이다. 옵션정보는 | 연산자를 이용해서 둘 이상을 동시에 지정 가능하다. 그러나 옵션의 종류와 지원여부는 운영체제에 따라서 차이가 있다.

MSG_OOB: 긴급 메시지의 전송

MSB_OOB 메시지를 받으면 SIGURG 시그널이 발생한다. 따라서 이에 대한 처리를 위해서 시그널 핸들링이 필요하다. 단, fcntl 함수의 호출을 통해 해당 소켓의 소유자를 현재 실행중인 프로세스로 변경해야 한다.

oob_send.c의 일부

```
write(sock, "123", strlen("123"));
send(sock, "4", strlen("4"), MSG_OOB);
write(sock, "567", strlen("567"));
send(sock, "890", strlen("890"), MSG_OOB);
```

oob_recv.c의 일부

```
act.sa_handler=urg_handler;
sigemptyset(&act.sa_mask);
act.sa_flags=0;
. . . . .
fcntl(recv_sock, F_SETOWN, getpid());
state=sigaction(SIGURG, &act, 0);

while((str_len=recv(recv_sock, buf, sizeof(buf), 0))!= 0)
{
    if(str_len==-1)
        continue;
    buf[str_len]=0;
    puts(buf);
}
```

recv_sock에서 발생하는 SIGURG 시그널을 처리하는 프로세스의 변경

```
void urg_handler(int signo)
{
    int str_len;
    char buf[BUF_SIZE];
    str_len=recv(recv_sock, buf, sizeof(buf)-1, MSG_OOB);
    buf[str_len]=0;
    printf("Urgent message: %s \n", buf);
}
```

```
root@my_linux:/tcpip# gcc oob_recv.c -o recv
root@my_linux:/tcpip# ./recv 9190
123
Urgent message: 4
567
Urgent message: 0
89
```

oob_recv.c의 실행결과

실행결과를 보면, 긴급으로 메시지가 전달된 흔적이 보이지 않는다. 이렇듯 MSG_OOB는 우리가 생각하는 긴급의 형태와 다르다.

oob_recv.c의 실행결과 관찰

oob_send.c의 일부

```
write(sock, "123", strlen("123"));
send(sock, "4", strlen("4"), MSG_OOB);
write(sock, "567", strlen("567"));
send(sock, "890", strlen("890"), MSG_OOB);
```

1 2 3 4 5 6 7 8 9 0

데이터의 전송순서

```
root@my_linux:/tcpip# gcc oob_recv.c -o recv
root@my_linux:/tcpip# ./recv 9190
123
Urgent message: 4
567
Urgent message: 0
89
```

oob_recv.c의 실행결과

긴급! 상황 시 다음 두가지 조건이 만족되어야 한다.

"더 빨리 전송을 해서 응급조치를 취한다."

그런데 소켓은 더 빨리 전송하지 않는다. 다만, Urgent-mode를 이용해서 긴급 상황의 발생을 알려서 우리가 응급조치를 취하도록 도울 뿐이다.

실행결과의 판단

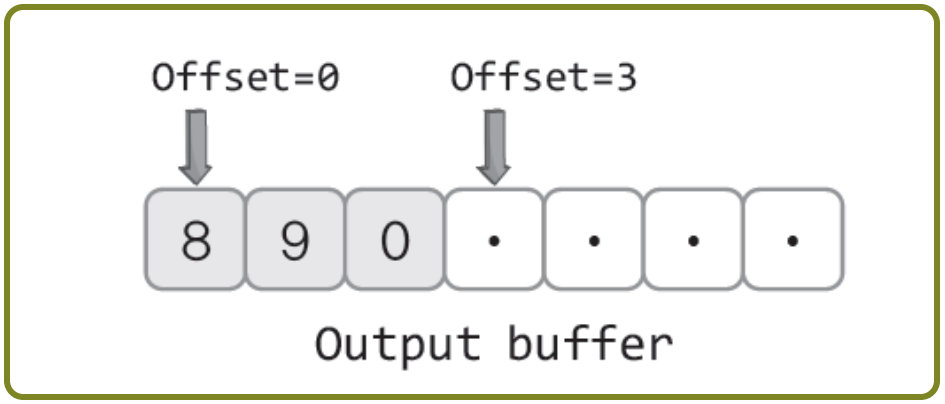
√ MSG_OOB 메시지라고 해서 더 빨리 전송되지 않는다.

√ 긴급으로 보낸 메시지의 양에 상관 없이 1바이트만 반환이 된다.

Urgent mode의 동작원리

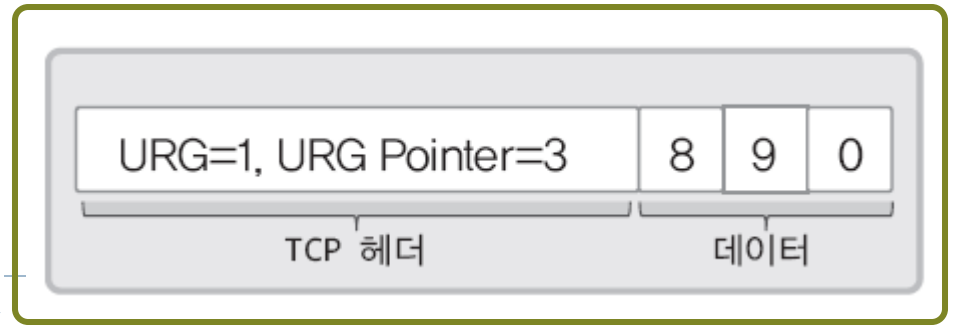
```
send(sock, "890", strlen("890"), MSG_OOB);
```

함수호출의 결과로 만들어진 출력버퍼의 상황



URG_OOB가 설정된 데이터가 전달되면서, 전달받은 운영체제는 SIGURG 시그널을 발생시켜서 **메시지의 긴급처리가 필요한 상황임을 프로세스에게 알린다.**

데이터 전송 시 패킷의 구성



URG=1은 긴급 메시지가 존재하는 패킷임을 알린다. URG Pointer=3은 긴급메시지가 설정된 위치 정보.

입력버퍼 검사하기

peek_send.c의 일부

```
if(connect(sock, (struct sockaddr*)&send_adr, sizeof(send_adr))==-1)
    error_handling("connect() error!");
```

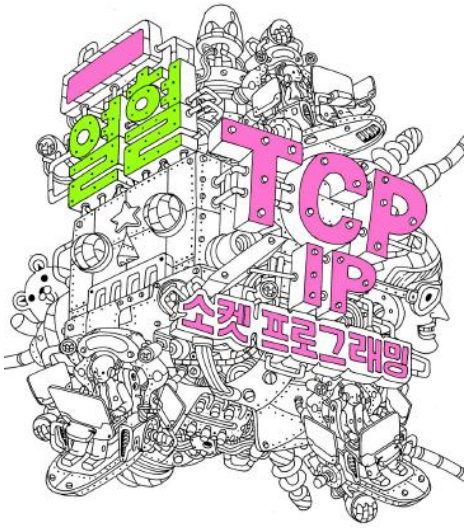
peek_recv.c의 일부

```
recv_sock=accept(acpt_sock, (struct sockaddr*)&recv_adr, &recv_adr_sz);

while(1)
{
    str_len=recv(recv_sock, buf, sizeof(buf)-1, MSG_PEEK|MSG_DONTWAIT);
    if(str_len>0)
        break;
}
buf[str_len]=0;
printf("Buffering %d bytes: %s \n", str_len, buf);

str_len=recv(recv_sock, buf, sizeof(buf)-1, 0);
buf[str_len]=0;
printf("Read again: %s \n", buf);
```

MSG_PEEK과 MSG_DONTWAIT의 옵션지정으로 인해서 블로킹 되지 않고 데이터의 존재 유무를 확인하게 된다.

버퍼에서 데이터를 읽으면 소멸된다. 그러나 MSG_PEEK 옵션이 지정되면 데이터를 읽어도 소멸되지 않는다.

```
root@my_linux:/tcpip# peek_recv.c -o recv
root@my_linux:/tcpip# ./recv 9190
Buffering 3 bytes: 123
Read again: 123
```



Chapter 13-2. readv & writev 입출력 함수

윤성우 저 열혈강의 TCP/IP 소켓 프로그래밍 개정판

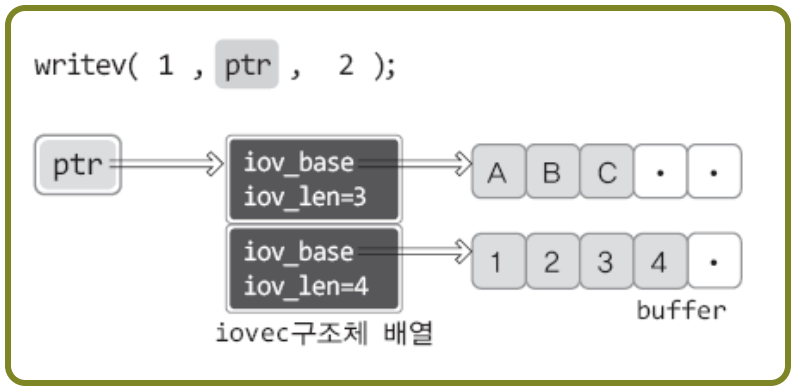
writev 함수의 사용

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

ssize_t writev(int filedes, const struct iovec * iov, int iovcnt);
```

➜ 성공 시 전송된 바이트 수, 실패 시 -1 반환

- filedes — 데이터 전송의 목적지를 나타내는 소켓의 파일 디스크립터 전달, 단 소켓에만 제한된 함수가 아니기 때문에, read 함수처럼 파일이나 콘솔 대상의 파일 디스크립터도 전달 가능하다.
- iov — 구조체 iovec 배열의 주소 값 전달, 구조체 iovec의 변수에는 전송할 데이터의 위치 및 크기 정보가 담긴다.
- iovcnt — 두 번째 인자로 전달된 주소 값이 가리키는 배열의 길이정보 전달.

둘 이상의 영역에 나뉘어 저장된 데이터를 묶어서 한번의 함수호출을 통해서 보낼 수 있다.




```
struct iovec
{
    void * iov_base;   // 버퍼의 주소 정보
    size_t iov_len;    // 버퍼의 크기 정보
}
```

writev 함수의 예

writev.c

```
int main(int argc, char *argv[])
{
    struct iovec vec[2];
    char buf1[]="ABCDEFG";
    char buf2[]="1234567";
    int str_len;

    vec[0].iov_base=buf1;
    vec[0].iov_len=3;
    vec[1].iov_base=buf2;
    vec[1].iov_len=4;

    str_len=writev(1, vec, 2);
    puts("");
    printf("Write bytes: %d \n", str_len);
    return 0;
}
```

실행 결과

```
root@my_linux:/tcpip# gcc writev.c -o wv
root@my_linux:/tcpip# ./wv
ABC1234
Write bytes: 7
```

readv 함수

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

ssize_t readv(int filedes, const struct iovec * iov, int iovcnt);
```

➜ 성공 시 수신된 바이트 수, 실패 시 -1 반환

- filedes — 데이터를 수신할 파일(혹은 소켓)의 파일 디스크립터를 인자로 전달.
- iov — 데이터를 저장할 위치와 크기 정보를 담고 있는 iovec 구조체 배열의 주소 값 전달.
- iovcnt — 두 번째 인자로 전달된 주소 값이 가리키는 배열의 길이정보 전달.

단 한번의 함수호출을 통해서 입력되는 데이터를 둘 이상의 영역에 나눠서 저장이 가능하다.

readv.c

```
struct iovec vec[2];
char buf1[BUF_SIZE]={0,};
char buf2[BUF_SIZE]={0,};
int str_len;

vec[0].iov_base=buf1;
vec[0].iov_len=5;
vec[1].iov_base=buf2;
vec[1].iov_len=BUF_SIZE;

str_len=readv(0, vec, 2);
printf("Read bytes: %d \n", str_len);
printf("First message: %s \n", buf1);
printf("Second message: %s \n", buf2);
```

실행결과

```
root@my_linux:/tcpip# gcc readv.c -o rv
root@my_linux:/tcpip# ./rv
I like TCP/IP socket programming~
Read bytes: 34
First message: I lik
Second message: e TCP/IP socket programming~
```

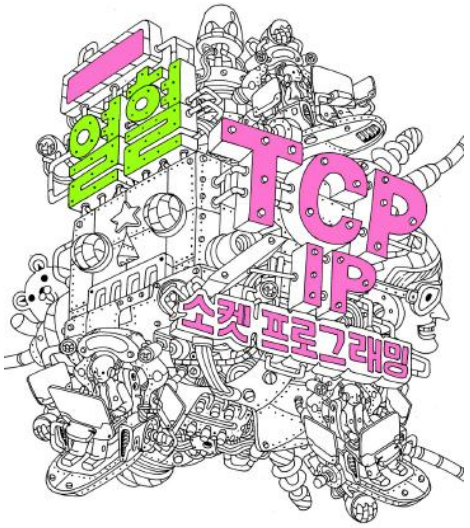
readv & writev 함수의 적절한 사용

여러 영역에 나뉘어 있는 데이터들을 하나의 배열에 순서대로 옮겨다 놓고 write 함수를 호출하는 것과 그 결과는 같다.

writev 함수호출이 유용한 이유

√ 단순하게 보면, 함수의 호출횟수를 줄일 수 있다

√ 잘게 나뉜 데이터들을 출력버퍼에 한번에 밀어 넣기 때문에 하나의 패킷으로 구성되어서 전송될 확률이 높아지고, 이는 전송속도의 향상으로도 이어질 수 있다.



Chapter 13-3. 윈도우 기반으로 구현하기

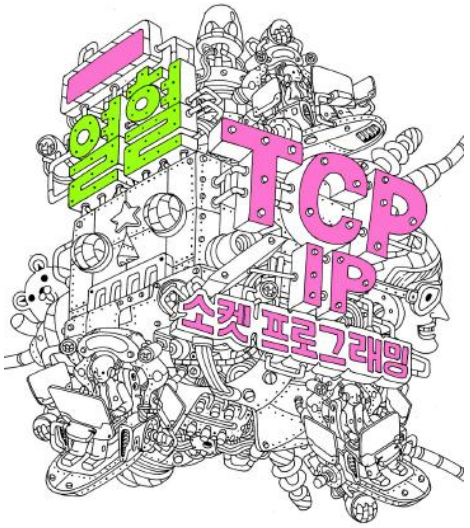
윤성우 저 열혈강의 TCP/IP 소켓 프로그래밍 개정판

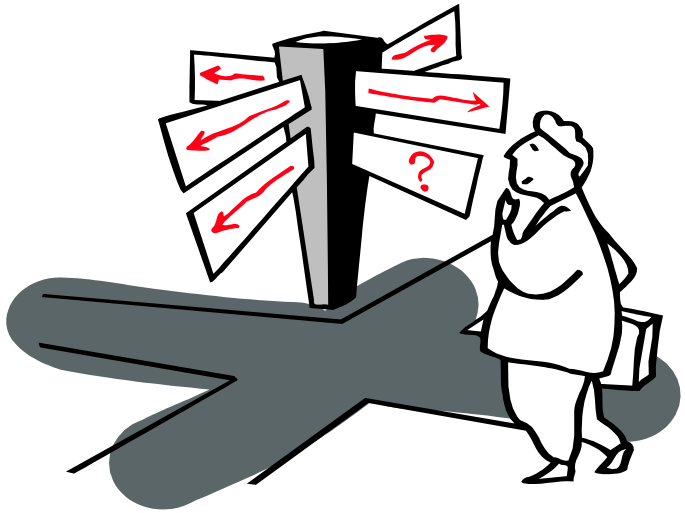
윈도우에서 시그널 핸들링?

윈도우에서의 MSG_OOB 관찰방법

윈도우에는 리눅스에서 보인 형태의 시그널 핸들링이 존재하지 않는다. 따라서 select 함수를 이용해서 MSG_OOB 메시지를 구분한다. MSG_OOB 메시지가 전달되면, select 함수의 3가지 관찰항목 중 예외상황이 발생한 소켓의 관찰항목에 등록이 된다.

```
while(1)
{
    readCopy=read;
    exceptCopy=except;
    timeout.tv_sec=5;
    timeout.tv_usec=0;

    result=select(0, &readCopy, 0, &exceptCopy, &timeout);

    if(result>0)
    {
        if(FD_ISSET(hRecvSock, &exceptCopy))
        {
            strLen=recv(hRecvSock, buf, BUF_SIZE-1, MSG_OOB);
            buf[strLen]=0;
            printf("Urgent message: %s \n", buf);
        }

        if(FD_ISSET(hRecvSock, &readCopy))
        {
            . . . . .
        }
    }
}
```





Chapter 13이 끝났습니다. 질문 있으신지요?