Chapter 13

다양한 입출력 함수들

Linux에서 send & recv

window에서 send recv 쓰고 Linux에서 사용하는 법

- send() 함수

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

ssize_t send(int sockfd, const void *buf, size_t nbtyes, int flags);
-> 성공 시 전송된 바이트 수, 실패시 -1 반환
```

  - sockfd: 데이터 전송 대상의 소켓 파일 디스크립터
  - buf:전송할 데이터를 저장하고 있는 버퍼의 주소
  - nbytes: 전송할 바이트 수
  - flags: 데이터 전송 시 적용할 다양한 옵션 정보

- recv() 함수

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

ssize_t recv(int sockfd, void *buf, size_t nbtyes, int flags);
-> 성공 시 전송된 바이트 수, 실패시 -1 반환
```

  - sockfd: 데이터 수신 대상의 소켓 파일 디스크립터
  - buf: 수신된 데이터를 저장할 버퍼의 주소
  - nbytes: 수신할 수 있는 최대 바이트 수
  - flags: 데이터 수신 시 적용할 다양한 옵션 정보

* 리눅스 버전에 따라서 이것도 제대로 안될 수 있음!

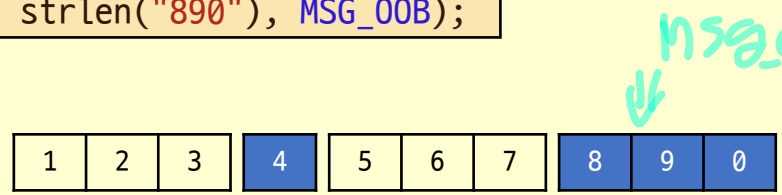
send & recv 함수의 옵션과 그 의미

옵션(Option)	의미	send	recv
MSG_OOB	▪ 긴급 데이터(Out-of-band data)의 전송을 위한 옵션	●	●
MSG_PEEK	▪ 입력 버퍼에 수신된 데이터의 존재 유무 확인을 위한 옵션		●
MSG_DONTROUTE	▪ 데이터 전송과정에서 라우팅(Routing) 테이블을 참조하지 않을 것을 요구하는 옵션 ▪ 로컬(Local) 네트워트상에서 목적지를 찾을 때 사용되는 옵션	●	
MSG_DONTWAIT	▪ 입출력 함수 호출 과정에서 블로킹 되지 않을 것을 요구하기 위한 옵션. ▪ Non-blocking IO의 요구에 사용되는 옵션	●	●
MSG_WAITALL	▪ 요청한 바이트 수에 해당하는 데이터가 전부 수신될 때까지 호출된 함수가 반환되는 것을 막기 위한 옵션		●

- 데이터 전송에 사용되는 옵션 정보는 (or) 연산자를 이용해서 둘 이상을 동시에 지정 가능
- 옵션의 종류와 지원 여부는 운영체제에 따라 차이가 있음

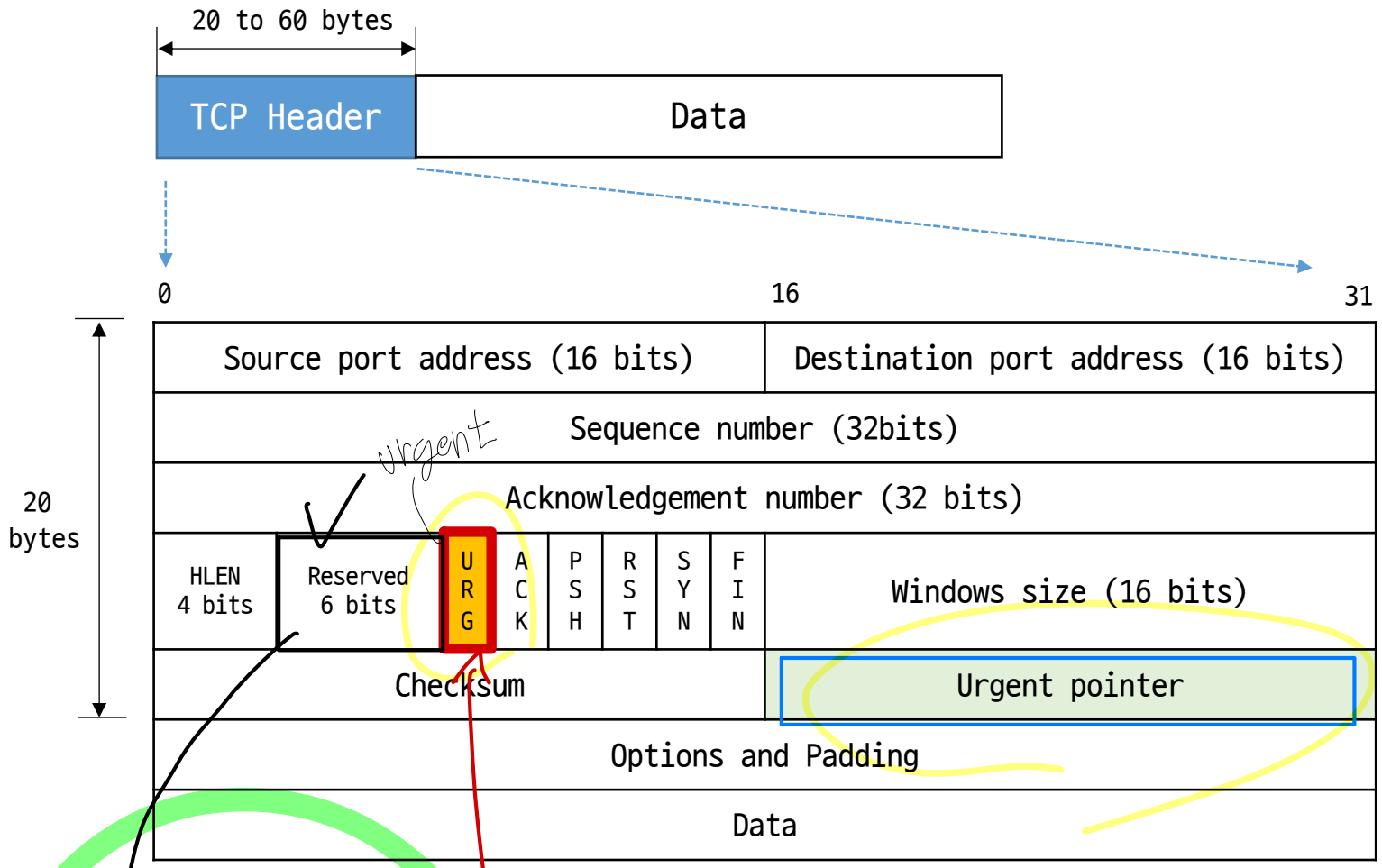
oob_send.c #1

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

#define BUF_SIZE 30
void error_handling(char *message);

int main(int argc, char *argv[])
{
    int sock;
    struct sockaddr_in recv_adr;
    if(argc != 3) {
        printf("Usage: %s <IP> <port>\n", argv[0]);
    }

    sock = socket(PF_INET, SOCK_STREAM, 0);
    memset(&recv_adr, 0, sizeof(recv_adr));
    recv_adr.sin_family = AF_INET;
    recv_adr.sin_addr.s_addr = inet_addr(argv[1]);
    recv_adr.sin_port = htons(atoi(argv[2]));
    if(connect(sock, (struct sockaddr*)&recv_adr, sizeof(recv_adr)) == -1)
        error_handling("connect() error!");

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

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

    close(sock);
    return 0;
}

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



"4", "890"을
긴급으로 전송

TCP Header Format



- URG(Urgent)
  - ✓ Urgent pointer 필드에 값이 채워져 있음을 알림
  - ✓ 긴급 데이터의 마지막 바이트+1 위치를 Urgent pointer가 가리킴
  - ✓ 긴급 메시지는 메시지 처리를 재촉하는데 의미가 있음
  - ✓ 긴급 전송의 의미는 아님

MSG_OOB: 긴급 메시지의 수신

- fcntl() 함수
  - 이미 열려 있는 파일의 특성을 제어하기 위해 사용

```
#include <unistd.h>
#include <fcntl.h>

int fcntl(int fd, int cmd, ...);
-> 성공 시 cmd 인자에 따라 다름, 오류 시 -1 반환
```

MSG_OOB ⇒ SIGURG

- fcntl(recv_sock, F_SETOWN, getpid())
  - F_SETOWN
    - 비동기 입출력과 관련
    - recv_sock이 가리키는 소켓의 소유자(F_SETOWN)를 getpid() 함수가 반환하는 프로세스 ID로 변경 시킴
    - SIGIO, SIGURG 시그널을 수신하는 프로세스 아이디를 설정하기 위해 사용
      - 현재의 프로세스가 SIGURG 시그널을 수신하도록 설정
      - 원래 소켓의 소유는 운영체제가 담당하고 있기 때문

소유권 변경

MSG_OOB: 긴급 메시지의 수신 (oob_recv.c)

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 프로세스를 변경

← SIG URG 등록

= flag 값

```
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);
}
```

- MSG_OOB 메시지를 수신하면 운영체제는 SIGURG 시그널을 발생시킴
  - ✓ SIGURG 시그널을 처리를 위해 시그널 핸들링이 필요
- fcntl() 함수 호출을 통해 해당 소켓의 소유자를 현재 실행 중인 프로세스로 변경
  - ✓ 소켓 소유자 변경: 운영체제 -> 현재 프로세스

```
$ ./oob_recv 9190

Urgent message: 4
Urgent message: 0
123
56789
```

↙ 이전 버퍼에서 4개를 뜸 : "890"

```
$ ./oob_send 127.0.0.1 9190
```

- 실행결과를 보면, 긴급으로 메시지가 전달된 흔적이 보이지 않음
- MSG_OOB는 우리가 생각하는 긴급의 형태와 다름

oob_recv.c #1

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <fcntl.h>

#define BUF_SIZE 30
void error_handling(char *message);
void urg_handler(int signo);

int acpt_sock;
int recv_sock;

int main(int argc, char *argv[])
{
    struct sockaddr_in recv_adr, serv_adr;
    int str_len, state;
    socklen_t serv_adr_sz;
    struct sigaction act;
    char buf[BUF_SIZE];

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

```c
    act.sa_handler=urg_handler;
    sigemptyset(&act.sa_mask);
    act.sa_flags=0;
    acpt_sock=socket(PF_INET, SOCK_STREAM, 0);
    memset(&recv_adr, 0, sizeof(recv_adr));
    recv_adr.sin_family=AF_INET;
    recv_adr.sin_addr.s_addr=htonl(INADDR_ANY);
    recv_adr.sin_port=htons(atoi(argv[1]));

    if(bind(acpt_sock, (struct sockaddr*)&recv_adr, sizeof(recv_adr))==-1)
        error_handling("bind() error");
    listen(acpt_sock, 5);

    serv_adr_sz=sizeof(serv_adr);
    recv_sock=accept(acpt_sock, (struct sockaddr*)&serv_adr, &serv_adr_sz);

    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);
    }
    close(recv_sock);
    close(acpt_sock);
    return 0;
}
```

SIGURG 시그널 핸들러 등록

oob_recv.c #2

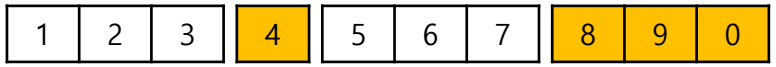
```c
void urg_handler(int signo)
{
    int str_len;
    char buf[BUF_SIZE];

    str_len=recv(recv_sock, buf, sizeof(buf)-1, MSG_OOB);
    buf[str_len]=0;
    printf("Urgent message: %s \n", buf);
}

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

OOB 메시지 처리

실행 결과

실행 #1 (Ubuntu 20.04)

```
$ ./oob_recv 9190
Urgent message: 0
123
56789
```




```
$ ./oob_send 127.0.0.1 9190
```

실행 #2 (Ubuntu 22.04)

```
$ ./oob_recv 9190
Urgent message: 4
Urgent message: 0
123
56789
```

```
$ ./oob_send 127.0.0.1 9190
```

Linux Raspberrypi 4.4.50

```
$ ./oob_recv 9190
123
Urgent message: 4
567
Urgent message: 0
89
```

"890"

```
$ ./oob_send 127.0.0.1 9190
```

- MSG_OOB 사용은 긴급 메시지의 내용(“890”)이 중요한 것이 아님
- 긴급 상황의 발생을 알리는 역할

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);
```

긴급! 상황 시 다음 두가지 조건이 만족되어야 한다.

"**더 빨리 전송**을 해서 **응급조치**를 취한다."

그런데 소켓은 더 빨리 전송하지 않는다.

다만, **Urgent-mode**를 이용해서 긴급 상황의 발생을 알려서 우리가 응급조치를 취하도록 도울 뿐이다.

oob_recv.c의 실행 결과 (Ubuntu 22.04)

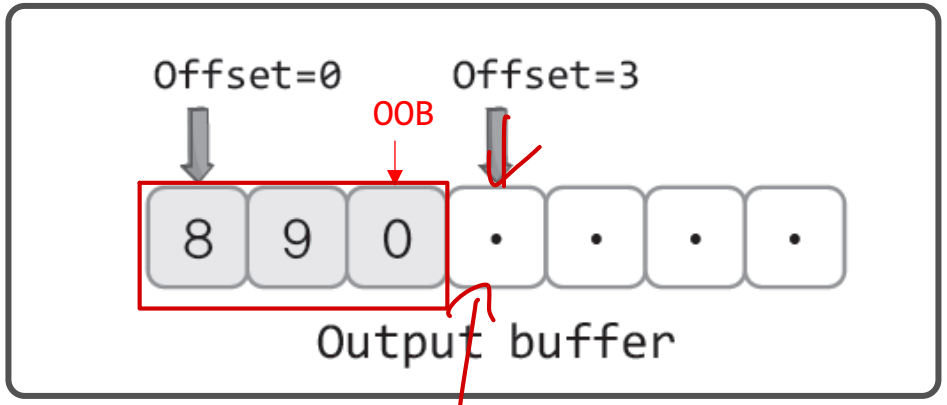
```
$ ./oob_recv 9190
Urgent message: 4
Urgent message: 0
123
56789
```

- 실행 결과의 판단
  - ✓ MSG_OOB 메시지라고 해서 더 빨리 전송되지는 않음
  - ✓ 긴급으로 보낸 메시지의 양에 상관없이 1바이트만 반환됨

Urgent mode의 동작원리

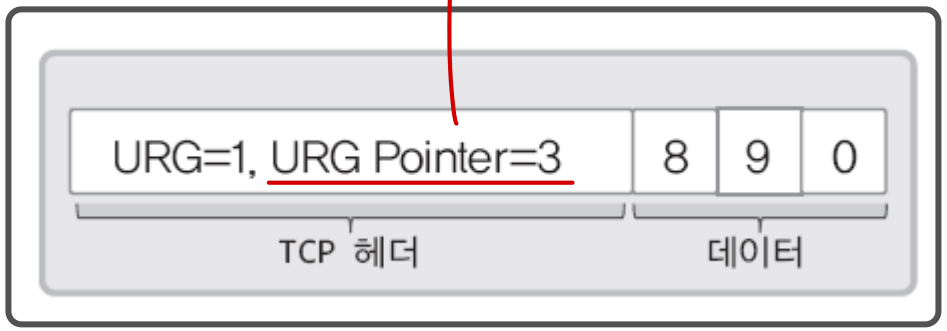
```
send(sock, "890", strlen("890"), MSG_OOB);
```

함수호출의 결과로 만들어진 출력버퍼의 상황



- MSG_OOB가 설정된 데이터가 전달되면, 운영체제는 SIGURG 시그널을 발생시킴
- 메시지의 긴급처리가 필요한 상황을 프로세스에게 알림

데이터 전송 시 패킷의 구성



- URG=1은 긴급 메시지가 존재하는 패킷임을 알림
- URG pointer=3은 긴급 메시지가 설정된 위치 정보를 나타냄
- offset 3의 바로 앞에 존재하는 것이 긴급 메시지

입력 버퍼 검사

- MSG_PEEK
  - 입력 버퍼에 수신 된 데이터가 존재하는지 확인 용도로 사용됨
  - recv() 함수에서 MSG_PEEK 옵션이 사용되면,
    - 입력 버퍼에 존재하는 데이터가 읽혀지더라도 입력 버퍼의 데이터는 지워지지 않음
- MSG_PEEK | MSG_DONTWAIT ← Non-blocking
  - 블로킹이 되지 않게 데이터의 존재 유무를 확인

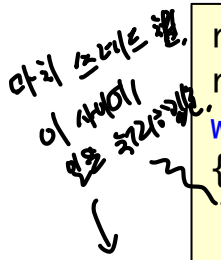
입력 버퍼 검사하기

peek_send.c의 일부

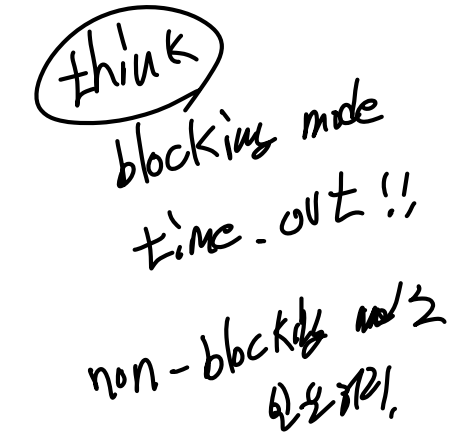
```
if(connect(sock, (struct sockaddr*)&send_adr, sizeof(send_adr))==-1)
    error_handling("connect() error!");

write(sock, "123", strlen("123"));
close(sock);
```

peek_recv.c의 일부

```
recv_adr_sz=sizeof(recv_adr);
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);
```



데이터 수신 확인



- 버퍼에서 데이터를 읽으면, 데이터는 소멸
- MSG_PEEK | MSG_DONTWAIT 옵션
  - ✓ MSG_PEEK: 데이터를 읽어도 소멸되지 않음
  - ✓ Non-blocking 모드에서 데이터 존재 유무를 확인할 수 있음

peek_send.c

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

int main(int argc, char *argv[])
{
    int sock;
    struct sockaddr_in send_adr;
    if(argc!=3) {
        printf("Usage : %s <IP> <port>\n", argv[0]);
        exit(1);
    }

    sock=socket(PF_INET, SOCK_STREAM, 0);

    memset(&send_adr, 0, sizeof(send_adr));
    send_adr.sin_family=AF_INET;
    send_adr.sin_addr.s_addr=inet_addr(argv[1]);
    send_adr.sin_port=htons(atoi(argv[2]));

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

    write(sock, "123", strlen("123"));
    close(sock);
    return 0;
}

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

peek_recv.c

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

#define BUF_SIZE 30
void error_handling(char *message);

int main(int argc, char *argv[])
{
    int acpt_sock, recv_sock;
    struct sockaddr_in acpt_adr, recv_adr;
    int str_len, state;
    socklen_t recv_adr_sz;
    char buf[BUF_SIZE];
    if(argc!=2) {
        printf("Usage : %s <port>\n", argv[0]);
        exit(1);
    }
    acpt_sock=socket(PF_INET, SOCK_STREAM, 0);
    memset(&acpt_adr, 0, sizeof(acpt_adr));
    acpt_adr.sin_family=AF_INET;
    acpt_adr.sin_addr.s_addr=htonl(INADDR_ANY);
    acpt_adr.sin_port=htons(atoi(argv[1]));
    if(bind(acpt_sock, (struct sockaddr*)&acpt_adr, sizeof(acpt_adr))==-1)
        error_handling("bind() error");
    listen(acpt_sock, 5);

    recv_adr_sz=sizeof(recv_adr);
    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);

    close(acpt_sock);
    close(recv_sock);
    return 0;
}
```

입력 버퍼에 수신된 데이터 존재 여부만 확인 (버퍼에서 삭제되지 않음)

실제 메시지 수신

실행 결과

peek_recv.c

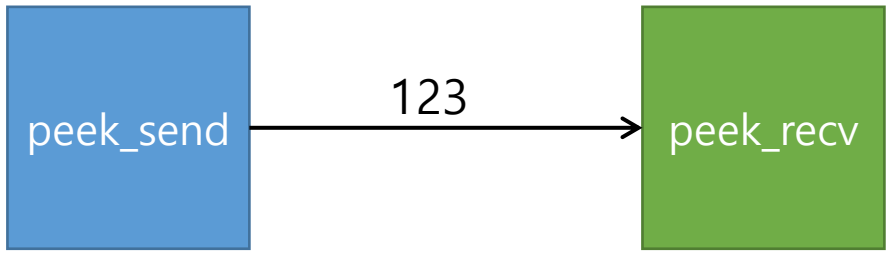
```
$ gcc peek_recv.c –o recv
$ ./ recv 9190
Buffering 3 bytes: 123
Read again: 123
```

⇐ 데이터 수신 확인

⇑ 수신된 데이터 읽어옴.

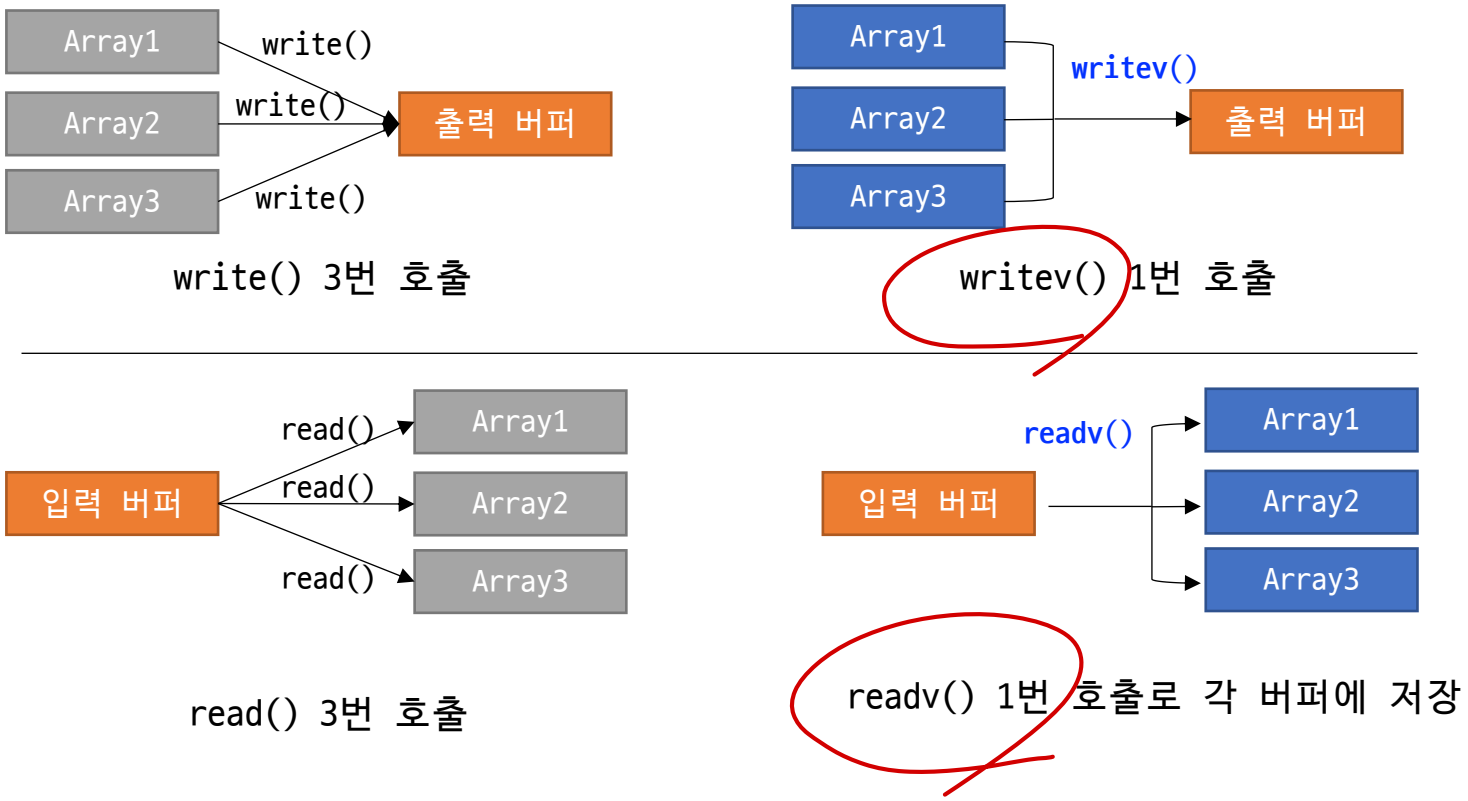
peek_send.c

```
$ gcc peek_send.c –o send
$ ./send 127.0.0.1 9190
```



readv & writev 입출력 함수

- 데이터 송수신의 효율성 향상을 위한 목적
  - read or write data into multiple buffers
  - 데이터를 모아서 전송 (writev): 한 번 호출하여 한 번에 보냄
  - 데이터를 분산 수신 (readv): 한 번 호출하여 나누어진 버퍼에 저장

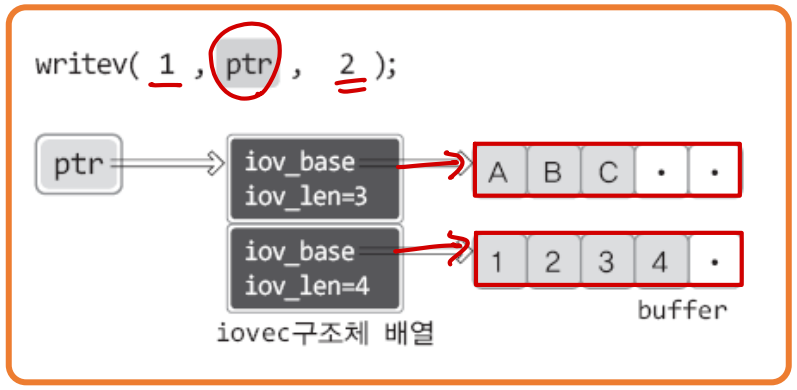


writev 함수의 사용

- writev() 함수

```
#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: 파일이나 소켓 디스크립터
- iov: 구조체 iovec 배열의 주소
- iovcnt: iovec 구조체 배열의 길이




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

- 둘 이상의 영역에 저장된 데이터를 묶어서 한 번의 함수 호출로 모든 데이터를 전송

writev() 함수 예제

writev.c

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

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 = strlen(buf1);
    vec[1].iov_base = buf2;
    vec[1].iov_len = strlen(buf2);

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

두 개의 데이터를 한 번의 writev() 함수 호출로 화면 출력

실행 결과

```
$ gcc writev.c -o writev
$ ./writev
ABCDEFG1234567
Write bytes: 14
```

readv 함수

- 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 또는 소켓의 파일 디스크립터
- iov: 데이터를 저장할 위치와 정보를 담고 있는 iovec 구조체 배열의 주소
- iovcnt: iovec 구조체 배열의 길이

- 단 한번의 readv() 함수 호출을 통해 입력되는 데이터를 둘 이상의 영역에 나누어 저장하는 것이 가능함

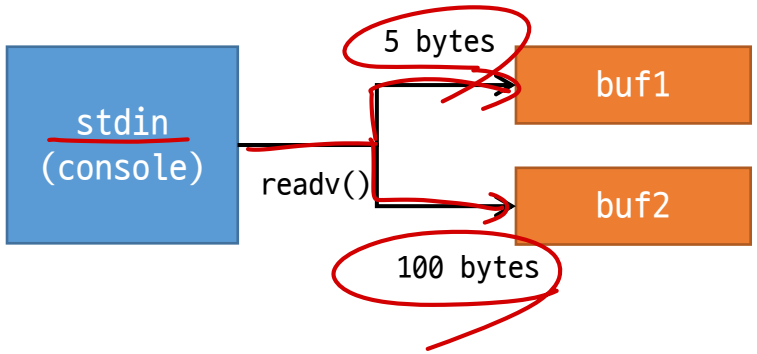
readv 함수 예제 (readv.c)

```c
#include <stdio.h>
#include <sys/uio.h>
#define BUF_SIZE 100

int main(int argc, char *argv[])
{
    struct iovec vec[2];
    char buf1[BUF_SIZE] = {0,};
    char buf2[BUF_SIZE] = {0,};
    int str_len = 0;

    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);

    return 0;
}
```

buf1에는 5 바이트 저장
buf2에는 100 바이트 저장

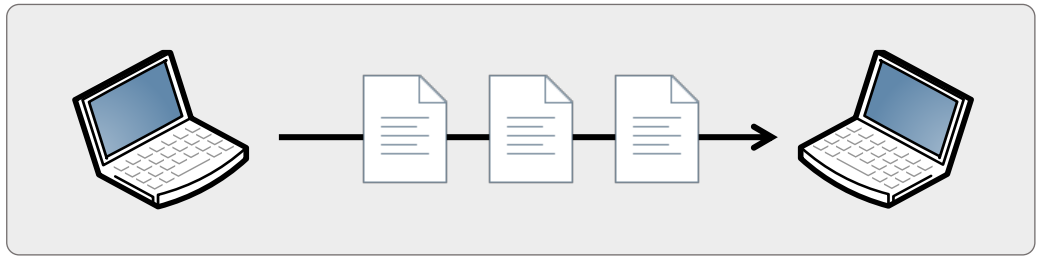
실행 결과

```
$ gcc readv.c -o readv
$ ./readv
I like TCP/IP socket programming.
Read bytes: 34
First message: I lik
Second message: e TCP/IP socket programming.
```

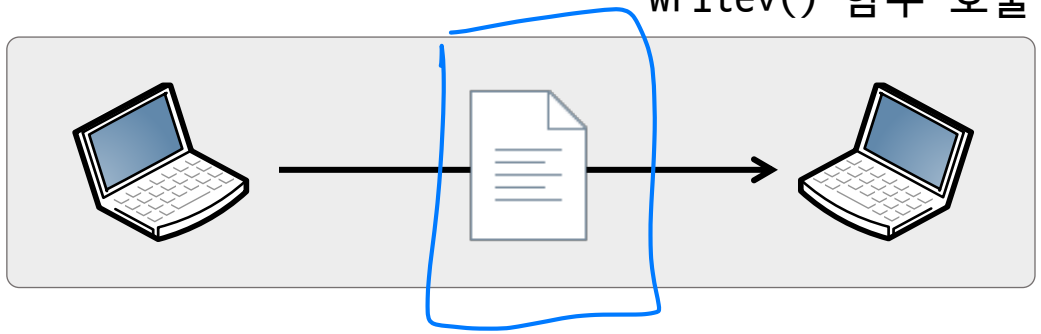


readv & writev 함수의 적절한 사용

write() 함수 호출



writev() 함수 호출



- 여러 영역에 나뉘어 있는 데이터들을 하나의 배열에 순서대로 옮겨놓고 write() 함수를 호출하는 것과 그 결과는 같음

- writev() 함수 호출이 유용한 이유
  - ✓ 단순하게, 함수 호출 횟수를 줄일 수 있음
  - ✓ 작게 나누어 진 데이터들을 출력 버퍼에 한 번에 전달하여 하나의 패킷으로 구성
  - ✓ 전송 속도 향상

Questions?