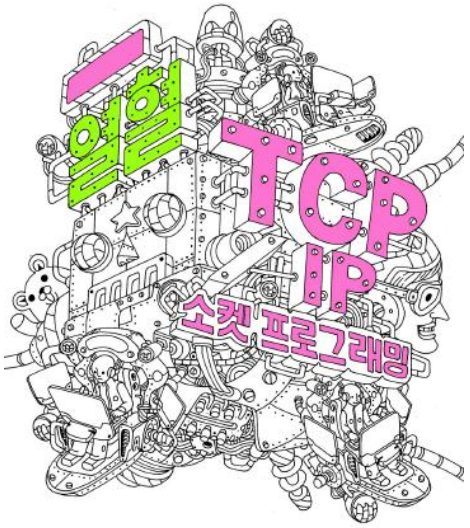
윤성우의 열혈 TCP/IP 소켓 프로그래밍

윤성우 저 열혈강의 TCP/IP 소켓 프로그래밍 개정판

Chapter 15. 소켓과 표준입출력



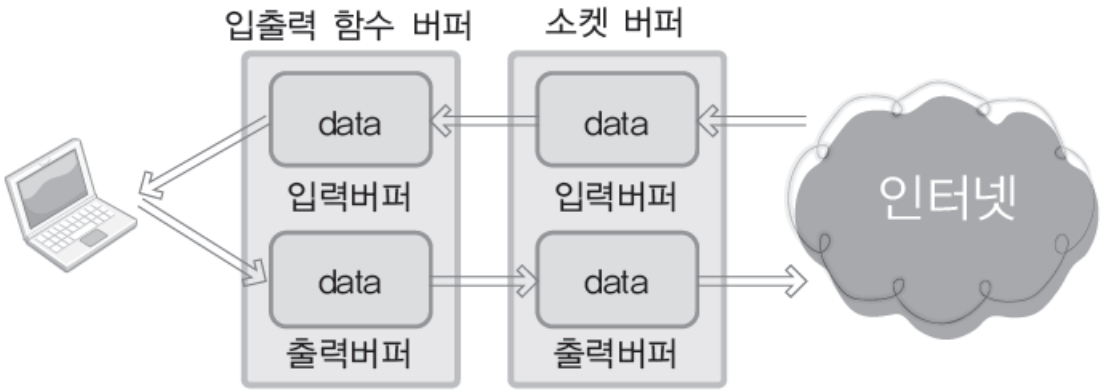
Chapter 15-1. 표준 입출력 함수의 장점

윤성우 저 열혈강의 TCP/IP 소켓 프로그래밍 개정판

표준 입출력 함수의 두 가지 장점

- 표준 입출력 함수는 이식성(Portability)이 좋다.
- 표준 입출력 함수는 버퍼링을 통한 성능의 향상에 도움이 된다.

ANSI C 기반의 표준 입출력 함수는 모든 컴파일러에서 지원을 하기 때문에 이식성이 좋아진다.



표준 입출력 함수를 이용해서 데이터를 전송할 경우 왼쪽의 그림과 같이 소켓의 입출력 버퍼 이외의 버퍼를 통해서 버퍼링이 된다.

표준 입출력 함수와 시스템 함수의 성능비교

예제 syscpy.c

```
int main(int argc, char *argv[])
{
    int fd1, fd2;     // fd1, fd2에 저장되는 것은 파일 디스크립터!
    int len;
    char buf[BUF_SIZE];

    fd1=open("news.txt", O_RDONLY);
    fd2=open("cpy.txt", O_WRONLY|O_CREAT|O_TRUNC);

    while((len=read(fd1, buf, sizeof(buf)))>0)
        write(fd2, buf, len);

    close(fd1);
    close(fd2);
    return 0;
}
```

왼쪽의 경우 시스템 함수를 이용해서 버퍼링 없는 파일 복사를 진행하고 있다.

아래의 경우 표준 입출력 함수를 이용해서 버퍼링 기반의 파일 복사를 진행하고 있다.

예제 stdcpy.c

```
int main(int argc, char *argv[])
{
    FILE * fp1; // fp1에 저장되는 것은 FILE 구조체의 포인터
    FILE * fp2; // fp2에 저장되는 것도 FILE 구조체의 포인터

    char buf[BUF_SIZE];

    fp1=fopen("news.txt", "r");
    fp2=fopen("cpy.txt", "w");

    while(fgets(buf, BUF_SIZE, fp1)!=NULL)
        fputs(buf, fp2);

    fclose(fp1);
    fclose(fp2);
    return 0;
}
```

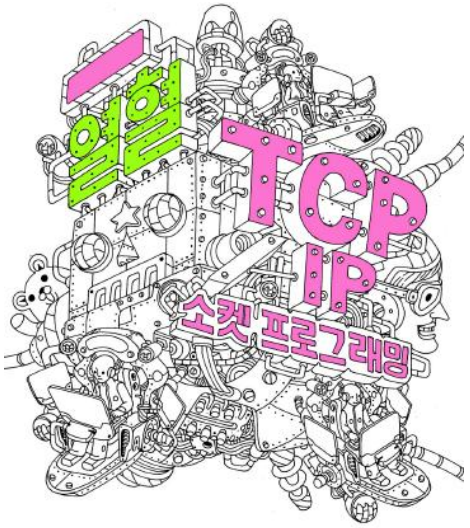
300메가 바이트 이상의 파일을 대상으로 테스트 시 속도의 차가 매우 극명하게 드러난다.

표준 입출력 함수의 사용에 있어서 불편사항

- 양방향 통신이 쉽지 않다.
- 상황에 따라서 fflush 함수의 호출이 빈번히 등장할 수 있다.
- 파일 디스크립터를 FILE 구조체의 포인터로 변환해야 한다.

fopen 함수 호출 시 반환되는 File 구조체의 포인터를 대상으로 입출력을 진행할 경우, 입력과 출력이 동시에 진행되게 하는 것은 간단하지 않다. 데이터가 버퍼링 되기 때문이다!

소켓 생성시 반환되는 것은 파일 디스크립터이다. 그런데 표준 C 함수에서 요구하는 것은 FILE 구조체의 포인터이다. 따라서 파일 디스크립터를 FILE 구조체의 포인터로 변환해야 한다.



Chapter 15-2. 표준 입출력 함수 사용하기

윤성우 저 열혈강의 TCP/IP 소켓 프로그래밍 개정판

fdopen 함수를 이용한 FILE 구조체 포인터로의 변환


```
#include <stdio.h>

FILE * fdopen(int fildes, const char * mode);
```

➜ 성공 시 변환된 FILE 구조체 포인터, 실패 시 NULL 반환

- fildes — 변환할 파일 디스크립터를 인자로 전달.
- mode — 생성할 FILE 구조체 포인터의 모드(mode)정보 전달.

예제 desto.c

```
int main(void)
{
    FILE *fp;
    int fd=open("data.dat", O_WRONLY|O_CREAT|O_TRUNC);
    if(fd==-1)
    {
        fputs("file open error", stdout);
        return -1;
    }

    fp=fdopen(fd, "w");
    fputs("Network C programming \n", fp);
    fclose(fp);
    return 0;
}
```

```
root@my_linux:/tcpip# gcc desto.c -o desto
root@my_linux:/tcpip# ./desto
root@my_linux:/tcpip# cat data.dat
Network C programming
```

실행결과

왼쪽 예제에서 주목해서 볼 것은 fdopen 함수 호출을 통해서 쓰기모드의 FILE 구조체 포인터가 반환되었다는 점이다.

fileno 함수를 이용한 파일 디스크립터로의 변환

```
#include <stdio.h>

int fileno(FILE * stream);
```

➜ 성공 시 변환된 파일 디스크립터, 실패 시 -1 반환

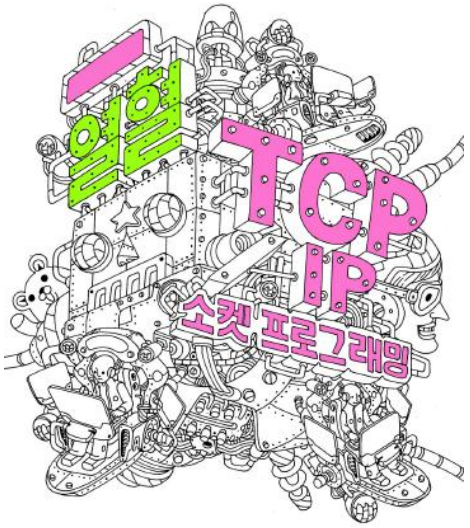
예제 todes.c

```
int main(void)
{
    FILE *fp;
    int fd=open("data.dat", O_WRONLY|O_CREAT|O_TRUNC);
    if(fd==-1)
    {
        fputs("file open error", stdout);
        return -1;
    }
    printf("First file descriptor: %d \n", fd);
    fp=fdopen(fd, "w");
    fputs("TCP/IP SOCKET PROGRAMMING \n", fp);
    printf("Second file descriptor: %d \n", fileno(fp));
    fclose(fp);
    return 0;
}
```

```
root@my_linux:/tcpip# gcc todes.c -o todes
root@my_linux:/tcpip# ./todes
First file descriptor: 3
Second file descriptor: 3
```

실행 결과

역으로 fileno 함수호출을 통해서 FILE 구조체 포인터를 파일 디스크립터로 변환하고 있다.



Chapter 15-3. 소켓 기반에서의 표준 입출력 함수 사용

윤성우 저 열혈강의 TCP/IP 소켓 프로그래밍 개정판

소켓 기반에서의 표준 C 입출력 함수의 호출 예

표준 C 입출력 함수의 호출 모델

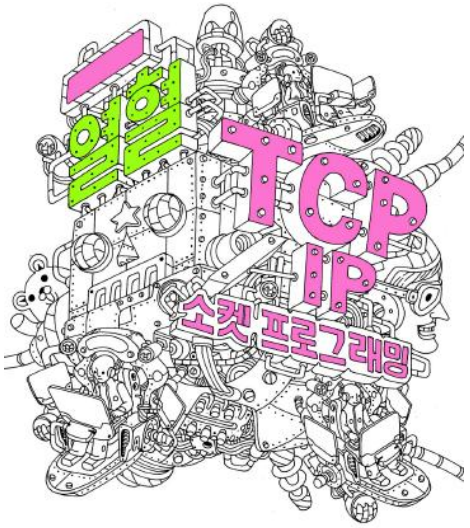
```
readfp=fdopen(sock, "r");
writefp=fdopen(sock, "w");
while(1)
{
    fputs("Input message(Q to quit): ", stdout);
    fgets(message, BUF_SIZE, stdin);
    if(!strcmp(message,"q\n") || !strcmp(message,"Q\n"))
        break;

    fputs(message, writefp);
    fflush(writefp);
    fgets(message, BUF_SIZE, readfp);
    printf("Message from server: %s", message);
}
```

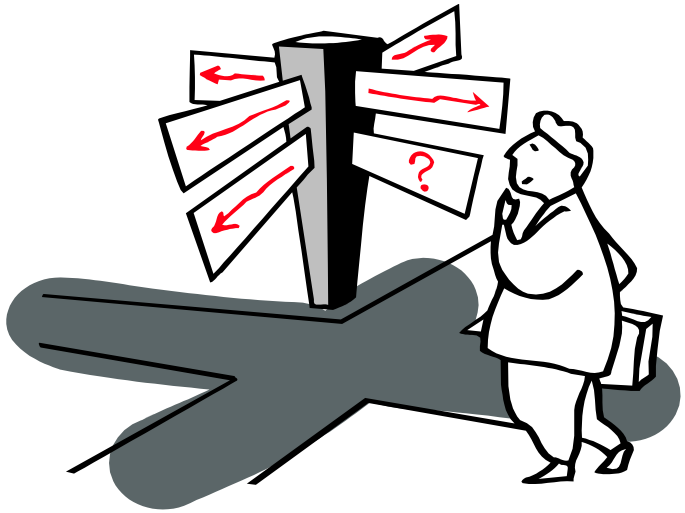
입력용, 출력용 FILE 구조체 포인터를 각각 생성해야 한다.

표준 C 입출력 함수를 사용할 경우 소켓의 버퍼 이외에 버퍼링이 되기 때문에 필요하다면, fflush 함수를 직접 호출해야 한다.

일반적인 순서

1. 파일 디스크립터를 FILE 구조체 포인터로 변환
2. 표준 입출력 함수의 호출
3. 함수 호출 후 fflush 함수호출을 통해서 버퍼 비움





Chapter 15가 끝났습니다. 질문 있으신지요?