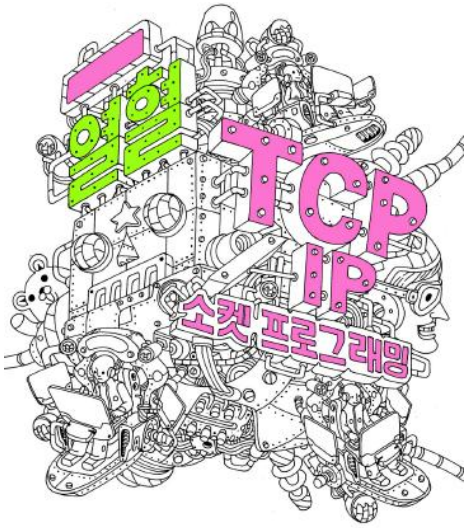
윤성우의 열혈 TCP/IP 소켓 프로그래밍

윤성우 저 열혈강의 TCP/IP 소켓 프로그래밍 개정판

Chapter 16. 입출력 스트림의 분리에 대한 나머지 이야기



Chapter 16-1. 입력 스트림과 출력 스트림의 분리

윤성우 저 열혈강의 TCP/IP 소켓 프로그래밍 개정판

스트림 분리의 이점

Chapter 10에서 설명한 스트림 분리의 목적

- 입력루틴(코드)과 출력루틴의 독립을 통한 구현의 편의성 증대
- 입력에 상관없이 출력이 가능하게 함으로 인해서 속도의 향상 기대

Chapter 10에서 설명한 스트림의 분리는 멀티 프로세스 기반의 분리였다.

Chapter 15에서 보인 스트림 분리의 이점

- FILE 포인터는 읽기모드와 쓰기모드를 구분해야 하므로,
- 읽기모드와 쓰기모드의 구분을 통한 구현의 편의성 증대
- 입력버퍼와 출력버퍼를 구분함으로 인한 버퍼링 기능의 향상

Chapter 15에서 보인 스트림의 분리는 FILE 구조체 포인터 기반의 분리였다.

Chapter 10에서 보인 스트림의 분리와 Chapter 16에서 보인 스트림의 분리는 그 방법에 있어서 차이를 보이고, 이로 인해서 기대하는 장점에도 차이가 있다.

스트림 분리 이후의 EOF에 대한 문제점

Chapter 07의 복습

· half-close shutdown(sock, SHUT_WR)

· 출력 스트림에 대해서 half-close 진행 시 EOF 전달

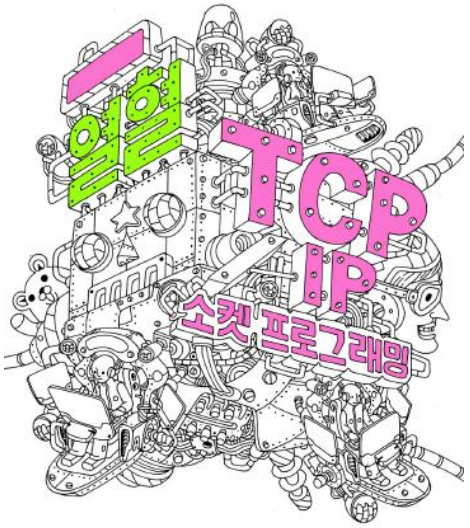
writefp를 대상으로 fclose 함수를 호출하면 half-close가 진행될까?

```
readfp=fdopen(clnt_sock, "r");
writefp=fdopen(clnt_sock, "w");

fputs("FROM SERVER: Hi~ client? \n", writefp);
fputs("I love all of the world \n", writefp);
fputs("You are awesome! \n", writefp);
fflush(writefp);

fclose(writefp);
fgets(buf, sizeof(buf), readfp);
```

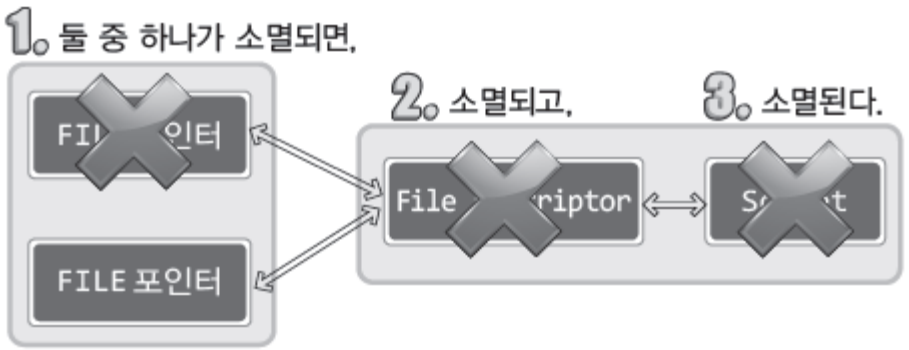
하나의 소켓을 대상으로 입력용 그리고 출력용 FILE 구조체 포인터를 얻었다 해도, 이 중 하나를 대상으로 fclose 함수를 호출하면, half-close가 아닌, 완전종료가 진행된다. 따라서, 위의 코드에서 마지막 행의 fgets 함수 호출은 성공하지 못한다.



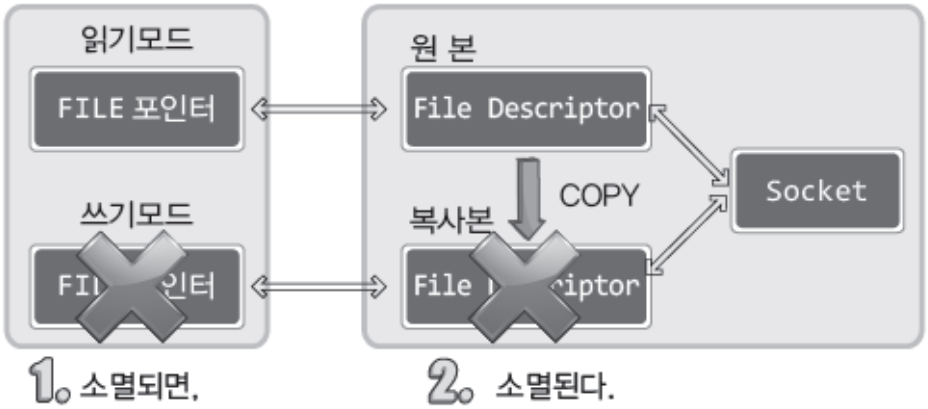
Chapter 16-2. 파일 디스크립터의 복사와 half-close

윤성우 저 열혈강의 TCP/IP 소켓 프로그래밍 개정판

스트림 종료 시 half-close가 진행되지 않는 이유



왼쪽 그림과 같이 하나의 파일 디스크립터를 대상으로 FILE 포인터가 생성되었으니, FILE 포인터가 종료되면, 연결된 FILE 디스크립터도 종료된다.



왼쪽 그림과 같이 파일 디스크립터를 복사한 다음에 각각의 파일 디스크립터를 대상으로 FILE 포인터를 만들면, FILE 포인터 소멸 시 해당 파일 포인터에 연결된 파일 디스크립터만 소멸된다.

하지만 위의 경우에도 half-close는 진행되지 않는다. 왜? 여전히 하나의 FILE 디스크립터가 남아있고, 이를 이용해서 입출력이 가능해야 하기 때문에!

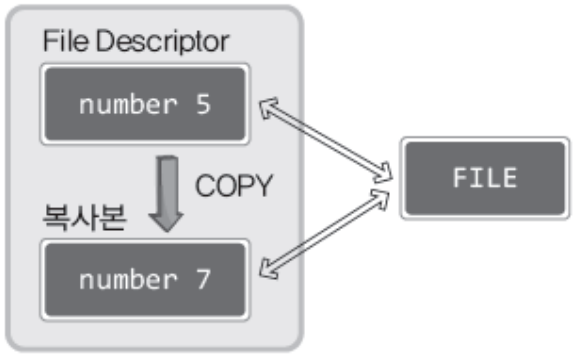
파일 디스크립터의 복사

```
#include <unistd.h>

int dup(int fildes);
int dup2(int fildes, int fildes2);
```

➜ 성공 시 복사된 파일 디스크립터, 실패 시 -1 반환

- fildes — 복사할 파일 디스크립터 전달.
- fildes2 — 명시적으로 지정할 파일 디스크립터의 정수 값 전달.



dup.c의 일부

```
cfd1=dup(1);
cfd2=dup2(cfd1, 7);

printf("fd1=%d, fd2=%d \n", cfd1, cfd2);
write(cfd1, str1, sizeof(str1));
write(cfd2, str2, sizeof(str2));

close(cfd1);
close(cfd2);
write(1, str1, sizeof(str1));
close(1);
write(1, str2, sizeof(str2));
```

왼쪽 코드에서 총 두 개의 파일 디스크립터를 복사한다. 그리고 복사된 파일 디스크립터까지 모두 종료를 한다. 때문에 마지막 행에 존재하는 write 함수의 호출은 성공하지 못한다.

실행 결과

```
root@my_linux:/tcpip# gcc dup.c -o dup
root@my_linux:/tcpip# ./dup
fd1=3, fd2=7
Hi~
It's nice day~
Hi~
```

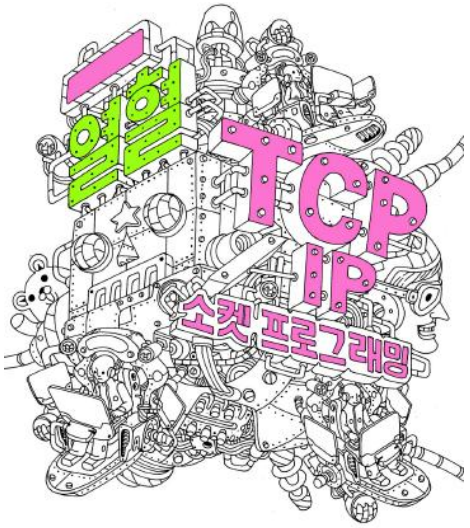
파일 디스크립터의 복사 후 스트림의 분리

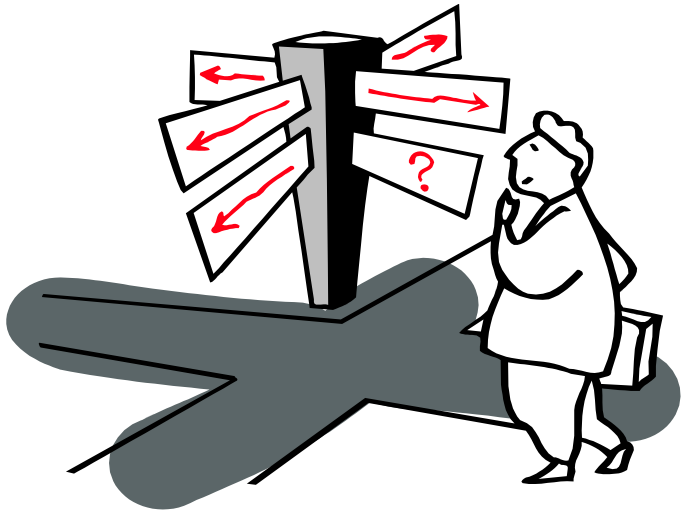
```
readfp=fdopen(clnt_sock, "r");
writefp=fdopen(dup(clnt_sock), "w");
fputs("FROM SERVER: Hi~ client? \n", writefp);
fputs("I love all of the world \n", writefp);
fputs("You are awesome! \n", writefp);
fflush(writefp);
shutdown(fileno(writefp), SHUT_WR);
fclose(writefp);
fgets(buf, sizeof(buf), readfp); fputs(buf, stdout);
fclose(readfp);
return 0;
```

dup 함수호출을 통해서 복사된 파일 디스크립터를 대상으로 FILE 구조체 포인터를 생성하고 있다.

파일 디스크립터로 변환해서 shutdown 함수를 호출한다. 따라서 이제 half-close가 진행되고, 이로 인해서 상대방 영역으로 EOF가 전달된다.

EOF 전달을 목적으로 하는 half-close는 파일 디스크립터를 이용해서 진행해야 한다.





Chapter 16이 끝났습니다. 질문 있으신지요?