Chapter 16

입출력 스트림 분리

스트림 분리의 이점

- Chapter 10의 스트림 분리: fork() 사용

  - 입력 루틴(코드)과 출력 루틴의 독립을 통한 구현의 편의성 증대
  - 입력에 상관없이 출력이 가능해서 속도 향상

  멀티프로세스 기반의 분리

- Chapter 15의 스트림 분리: 입출력용 파일 포인터 사용

  - FILE 포인터는 읽기 모드와 쓰기 모드를 구분해야 됨
  - 읽기 모드와 쓰기 모드의 구분을 통한 구현의 편의성 증대
  - 입력 버퍼와 출력 버퍼를 구분해서 버퍼링 기능 향상

  FILE 구조체 포인터 기반의 분리

스트림 분리 이후의 EOF에 대한 문제점 : FILE* 사용시 half-close()

- half-close 기능 write 중요 상에서 pad함.

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

int shutdown(int 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);
```

소켓 fd ⇒ FILE* fdopen

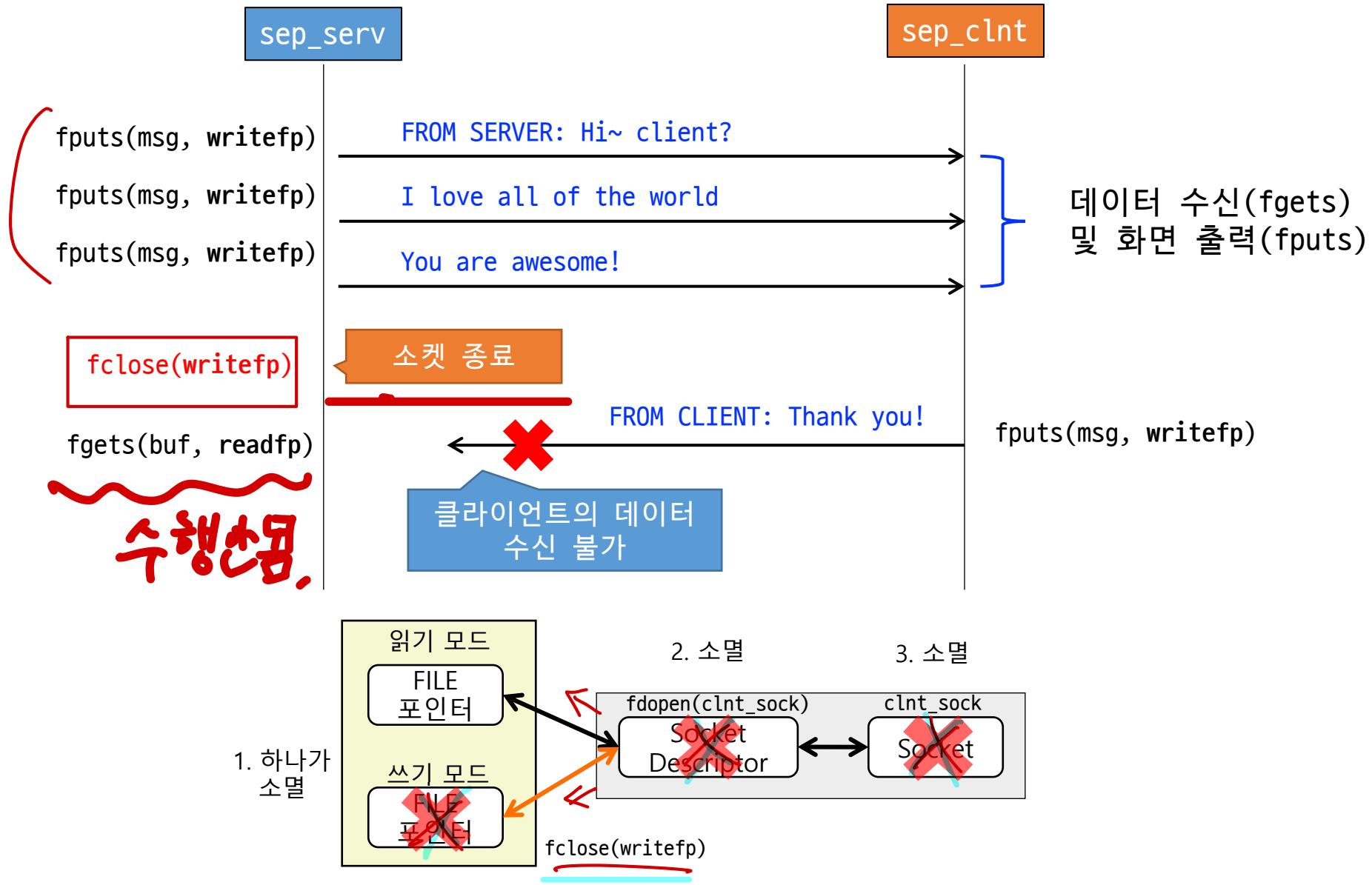
소켓의 완전 종료

✗ : 실행 안됨.

– 위의 코드에서 마지막 행의 fgets() 함수 호출은 성공하지 못함

하나의 소켓을 대상으로 입력용 그리고 출력용 FILE 구조체 포인터를 얻었다 해도, 이 중 하나를 대상으로 fclose() 함수를 호출하면, half-close가 아닌 완전 종료가 진행된다.

sep_serv.c 와 sep_clnt.c 실행 과정



sep_serv.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#define BUF_SIZE 1024

int main(int argc, char *argv[])
{
    int serv_sock, clnt_sock;
    FILE * readfp;
    FILE * writefp;

    struct sockaddr_in serv_adr, clnt_adr;
    socklen_t clnt_adr_sz;
    char buf[BUF_SIZE]={0,};

    serv_sock=socket(PF_INET, SOCK_STREAM, 0);
    memset(&serv_adr, 0, sizeof(serv_adr));
    serv_adr.sin_family=AF_INET;
    serv_adr.sin_addr.s_addr=htonl(INADDR_ANY);
    serv_adr.sin_port=htons(atoi(argv[1]));
    bind(serv_sock, (struct sockaddr*) &serv_adr, sizeof(serv_adr));
    listen(serv_sock, 5);
    clnt_adr_sz=sizeof(clnt_adr);
    clnt_sock=accept(serv_sock, (struct sockaddr*)&clnt_adr,&clnt_adr_sz);
    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);

    fputs(buf, stdout);
    fclose(readfp);
    return 0;
}
```

소켓의 완전 종료

fclose(writefp) 호출로 클라이언트가 전송하는 데이터는 수신하지 못함

sep_clnt.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#define BUF_SIZE 1024

int main(int argc, char *argv[])
{
    int sock;
    char buf[BUF_SIZE];
    struct sockaddr_in serv_addr;

    FILE * readfp;
    FILE * writefp;

    sock=socket(PF_INET, SOCK_STREAM, 0);
    memset(&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family=AF_INET;
    serv_addr.sin_addr.s_addr=inet_addr(argv[1]);
    serv_addr.sin_port=htons(atoi(argv[2]));

    connect(sock, (struct sockaddr*)&serv_addr, sizeof(serv_addr));
    readfp=fdopen(sock, "r");
    writefp=fdopen(sock, "w");

    while(1)
    {
        if(fgets(buf, sizeof(buf), readfp)==NULL)
            break;
        fputs(buf, stdout);
        fflush(stdout);
    }

    fputs("FROM CLIENT: Thank you! \n", writefp);

    fflush(writefp);
    fclose(writefp);
    fclose(readfp);
    return 0;
}
```

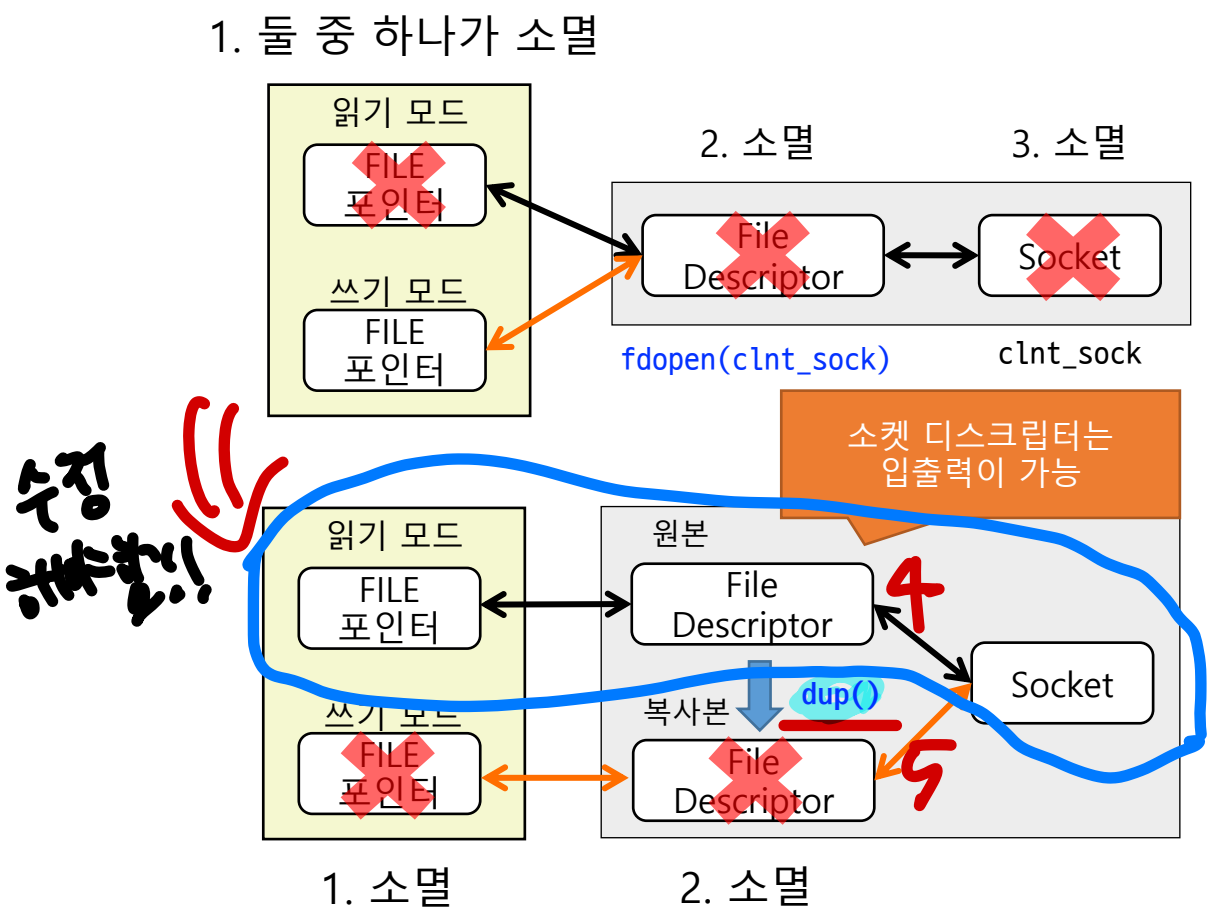
수신

서버로 전송

서버에서 수신 못함

```
$ ./sep_client 127.0.0.1 9190
FROM SERVER: Hi~ client?
I love all of the world
You are awesome!
```

스트림 종료 시 half-close가 진행되지 않는 이유



하나의 파일(소켓) 디스크립터를 대상으로 FILE 포인터가 생성됨

- FILE 포인터가 종료되면, 연결된 FILE 디스크립터도 종료됨

파일 디스크립터를 복사한 다음, 각각의 파일 디스크립터를 대상으로 FILE 포인터를 생성

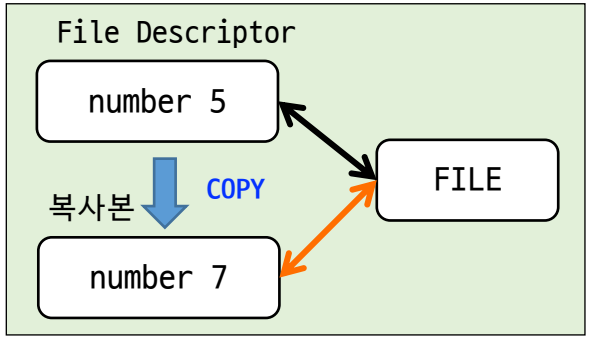
- 하나의 FILE 포인터 소멸 시 해당 파일 포인터에 연결된 파일 디스크립터만 소멸됨

하지만 위의 경우에도 half-close는 진행되지 않는다.
왜? 여전히 하나의 파일 디스크립터(소켓 디스크립터)가 남아있고, 이를 이용해서 socket 입출력이 가능함

파일 디스크립터 복사

- dup(), dup2() 함수

```
#include <unistd.h>

int dup(int filedes);
int dup2(int filedes, int filedes2);
-> 성공 시 복사된 파일 디스크립터, 실패 시 -1 반환
```



동일한 파일에 접근,
파일 디스크립터는 다름

- filedes: 복사할 파일 디스크립터
- filedes2: 명시적으로 지정할 파일 디스크립터의 정수값 전달

dup.c 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));
```

출력 안됨

- 총 두 개의 파일 디스크립터를 복사하고, 복사된 파일 디스크립터까지 모두 종료
- 따라서 마지막 행에 존재하는 write 함수의 호출은 성공하지 못함

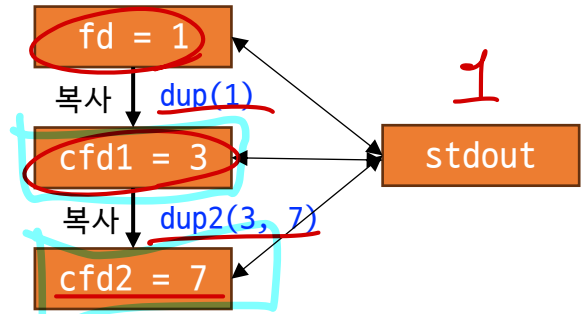
dup.c

```c
#include <stdio.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
    int cfd1, cfd2;
    char str1[]="Hi~ \n";
    char str2[]="It's nice day~ \n";

    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));
    return 0;
}
```

close(1) 호출로
str2는 출력 안됨

dup(1)을 수행했을 때 cfd1은 3이 됨
(0, 1, 2는 예약되어 있기 때문)

총 3개의 file descriptor
- 1, cfd1, cfd2



cfd1, cfd2가 close()
여전히 1은 남아있음

```
$ ./dup
fd1=3, fd2=7
Hi~
It's a nice day~
Hi~
```

파일 디스크립터의 복사 후 스트림의 분리

- sep_serv2.c 일부 코드

```
FILE * readfp;
FILE * writefp;
. . .
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;
```

추가 dup(clnt_sock)

복사

fileno(writefp)

writefp ⇒ FILE* ⇒ File 디스크립터 변경

↳ half-close

FILE*

- 소켓 디스크립터를 파일 디스크립터로 변환해서 shutdown 함수를 호출
- 이제 half-close가 진행되고, 이로 인해서 상대방에게 EOF가 전달됨
- EOF를 전달할 목적으로 half-close를 사용할 경우, 파일 디스크립터를 이용해서 진행

sep_serv2.c #1

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#define BUF_SIZE 1024

int main(int argc, char *argv[])
{
    int serv_sock, clnt_sock;
    FILE * readfp;
    FILE * writefp;
    struct sockaddr_in serv_adr, clnt_adr;
    socklen_t clnt_adr_sz;
    char buf[BUF_SIZE]={0,};

    serv_sock=socket(PF_INET, SOCK_STREAM, 0);
    memset(&serv_adr, 0, sizeof(serv_adr));
    serv_adr.sin_family=AF_INET;
    serv_adr.sin_addr.s_addr=htonl(INADDR_ANY);
    serv_adr.sin_port=htons(atoi(argv[1]));

    bind(serv_sock, (struct sockaddr*) &serv_adr, sizeof(serv_adr));
    listen(serv_sock, 5);
    clnt_adr_sz=sizeof(clnt_adr);
    clnt_sock=accept(serv_sock, (struct sockaddr*)&clnt_adr,&clnt_adr_sz);
    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;
}
```

소켓 디스크립터 복사

half-close
- 클라이언트로 EOF가 전달됨

클라이언트가 전송한 메시지 수신

서버 실행

```
$ ./sep_serv2 9190
FROM CLIENT: Thank you!
```

클라이언트 실행

```
$ ./sep_clnt 127.0.0.1 9190
FROM SERVER: Hi~ client?
I love all of the world
You are awesome!
```

sep_serv.c 와 sep_serv2.c 파일 비교

sep_serv.c

```
    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);

    fputs(buf, stdout);
    fclose(readfp);
    return 0;
}
```

sep_serv2.c

```
    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;
}
```

half-close

- dup(clnt_sock)으로 소켓 디스크립터 복사
- 쓰기용 디스크립터를 shutdown() 시킴

Questions?