

실습 11

IPC: Interprocess Communication servers

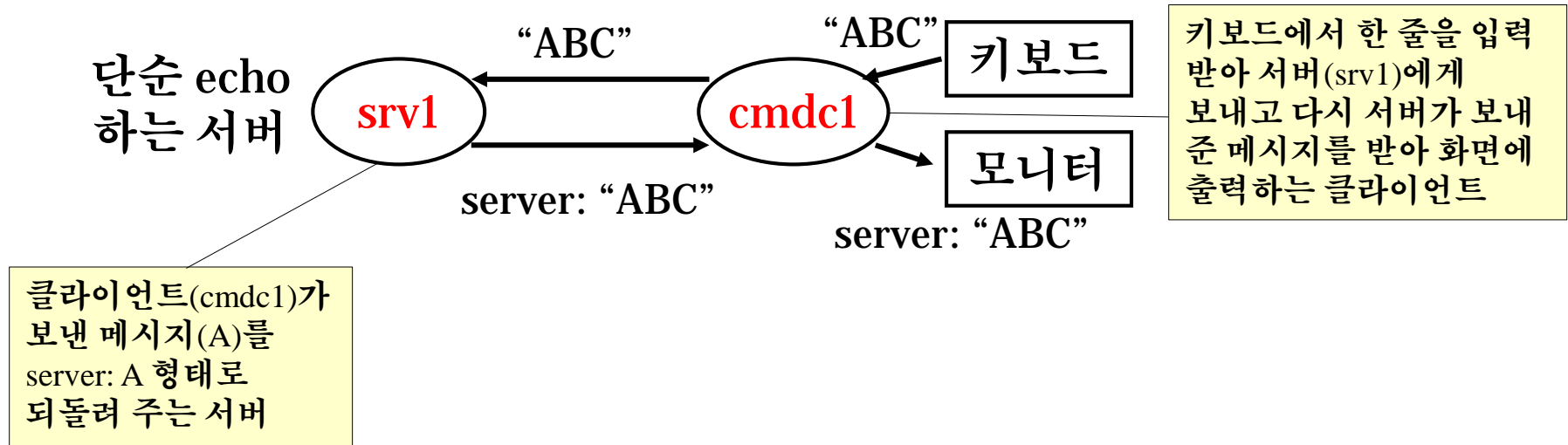
Jaehong Shim

Dept. of Computer Engineering



실습 11-1

cmdc1 과 srv1

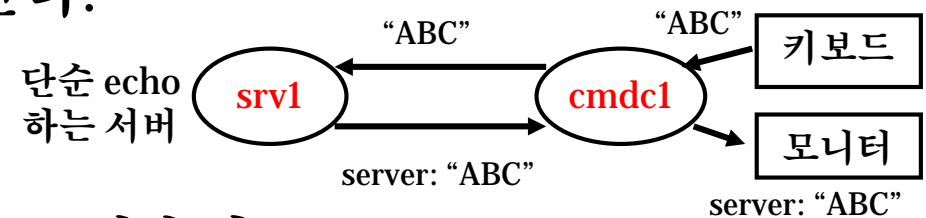


프로그램들 간의 통신

□ 서버(srv1)와 클라이언트(cmdc1): 두 개의 프로그램들간 통신

□ srv1: 에코 서버 프로그램

- 클라이언트가 보낸 명령어를 수신하고 해당 메시지를 서버 화면에 출력한 후 “server: 클라이언트-메시지” 에코 문자열을 만들고
- 이를 다시 클라이언트에게 전송한다.



□ cmdc1: 클라이언트 프로그램

- 키보드 입력(명령어)을 읽어 서버로 전송하고,
- 서버로부터 다시 에코 메시지를 수신하여 화면에 출력한다.
- srv1를 먼저 실행시킨 후, 새로운 터미널 창 또는 Putty을 띄워 cmdc1를 실행해야 함

새로 시작하기

- ~/up 디렉토리 밑에 IPC 디렉토리를 만들고 기존 파일을 복사한다.

```
$ cd
$ cd up
$ mkdir IPC
$ cd IPC
$ cp /home/jhshim/up/IPC/* .
$ ls
cmdc1.c  srv1.c  Makefile
```

서버와 클라이언트의 실행

- ❑ make 실행한 후 터미널을 세 개 실행한다.
- ❑ 한쪽 터미널에서 서버 **srv1**을 먼저 실행
 - cmdc1가 보내준 메시지(A)를 “server: A”로 에코 해 준다. 아무 출력도 없음
- ❑ 다른 쪽 터미널에서 **cmdc1**을 실행한 후 키보드에서 메시지를 한 줄씩 입력 후 엔터를 친다. 이를 계속 반복한다.
 - 자신이 입력한 내용과 서버(srv1)에서 되돌려준(echo) 메시지를 볼 수 있음

```
jhshim1@esl: ~/up/IPC
jhshim1: [~/up/IPC] $ srv1
This is 연습 이 다 .
오 우 좋 은 데
직 인 다 .
[ ]

jhshim1@esl: ~/up/IPC
jhshim1: [~/up/IPC] $ cmdc1
This is 연습 이 다 .
server: This is 연습 이 다 .
오 우 좋 은 데
server: 오 우 좋 은 데
직 인 다 .
server: 직 인 다 .
[ ]
```

```
jhshim1@esl: ~
jhshim1: [~] $ ps -u jhshim1
  PID TTY          TIME CMD
166174 ?            00:00:00 systemd
166177 ?            00:00:00 (sd-pam)
166248 ?            00:00:00 sshd
166249 pts/8        00:00:00 bash
166297 ?            00:00:00 sshd
166298 pts/9        00:00:00 bash
166347 ?            00:00:00 sshd
166348 pts/10       00:00:00 bash
166634 pts/9        00:00:00 cmdc1
166646 pts/8        00:00:00 srv1
166690 pts/10       00:00:00 ps
jhshim1: [~] $ [ ]
```

자신의 계정이름으로 대체할 것

프로세스간 통신용 FIFO 파일 만들기

- 아래처럼 두개의 통신용 FIFO 파일을 생성한다.

```
// IPC 디렉토리에서 아래 명령어를 실행하라.
```

```
$ mkfifo fifo_c_to_s fifo_s_to_c
```

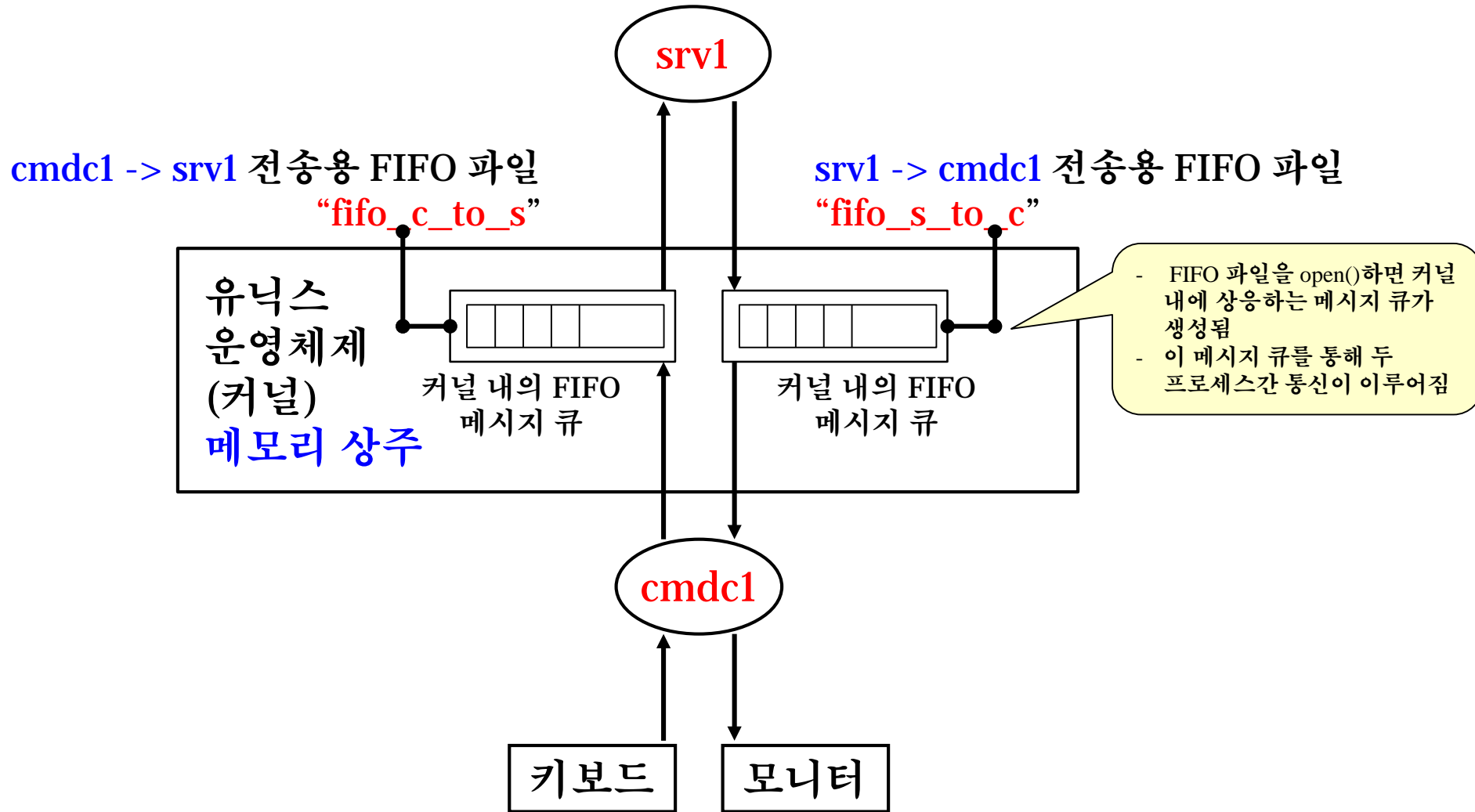
```
$ ls // 아래 두개 파일 존재해야 함
fifo_c_to_s  fifo_s_to_c
```

```
$ ls -F // 파일 이름 뒤에 | 는 fifo 또는 pipe 파일임을 의미
fifo_c_to_s|  fifo_s_to_c|
```

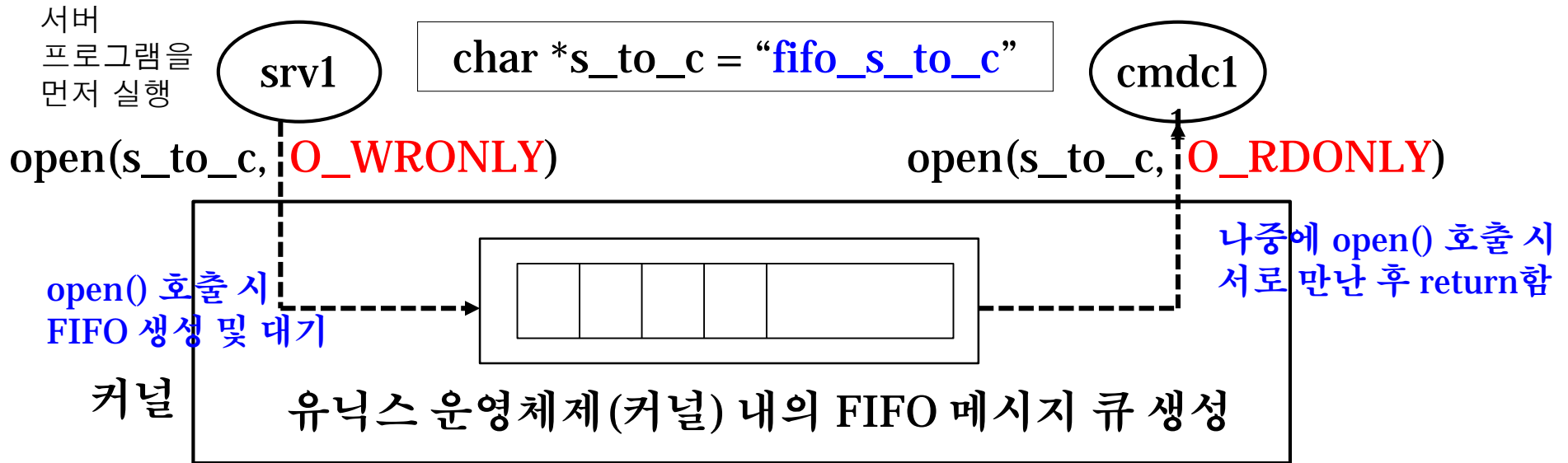
```
$ ls -lF // 파일맨 앞의 p는 fifo/pipe 파일임을 의미, 파일크기 0
prw-rw-r--  1 jhshim1 jhshim1      0  5월 27 19:40 fifo_c_to_s|
prw-rw-r--  1 jhshim1 jhshim1      0  5월 27 19:40 fifo_s_to_c|
```

- 위 두 파일은 두 프로세스간 통신을 위해 필요한 파일임
- 교재 15장, p.603 참조

한 방향 통신용 FIFO 파일과 상응하는 메시지 큐

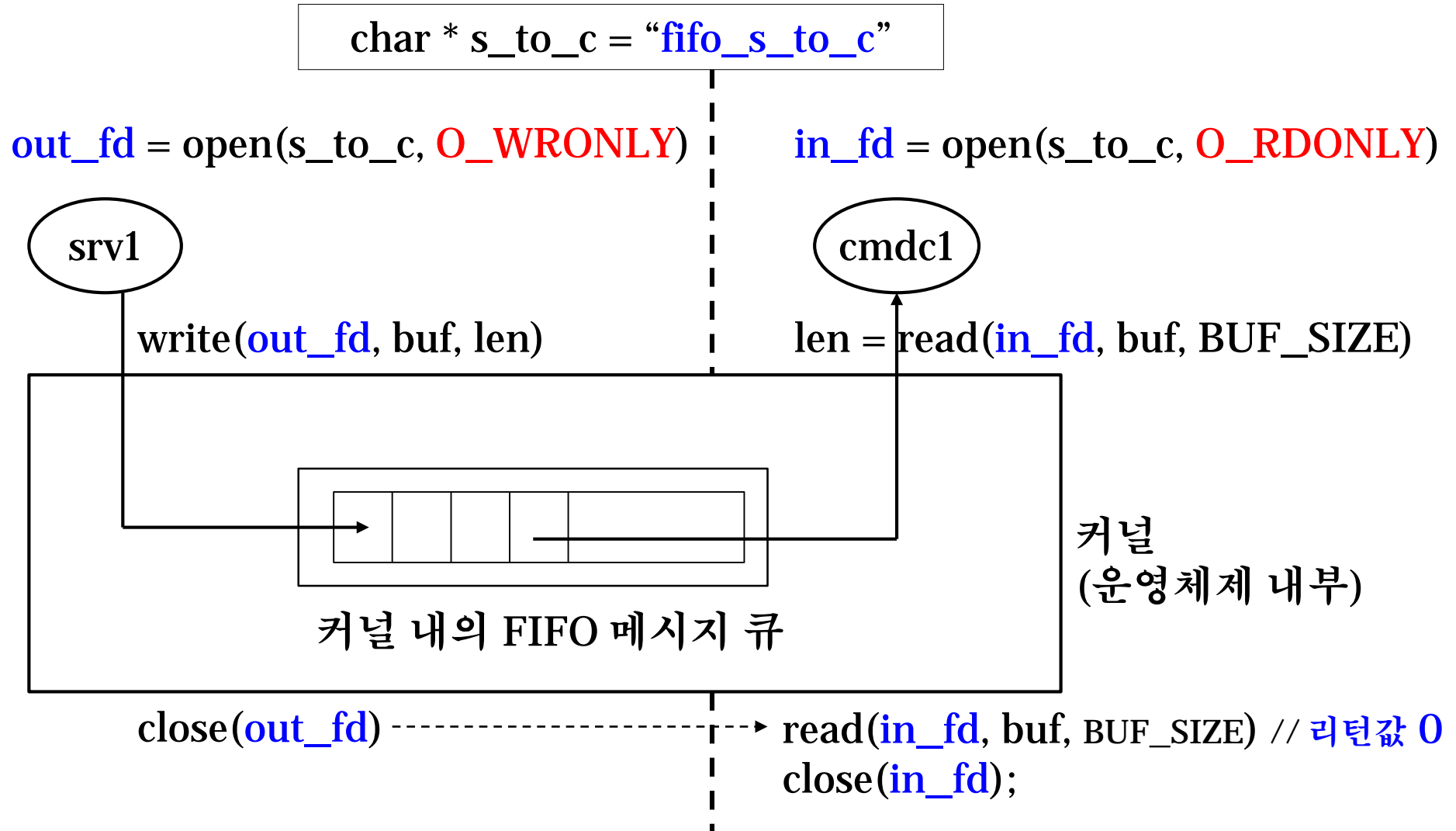


통신 연결: FIFO 파일의 열기(메시지 큐 생성)

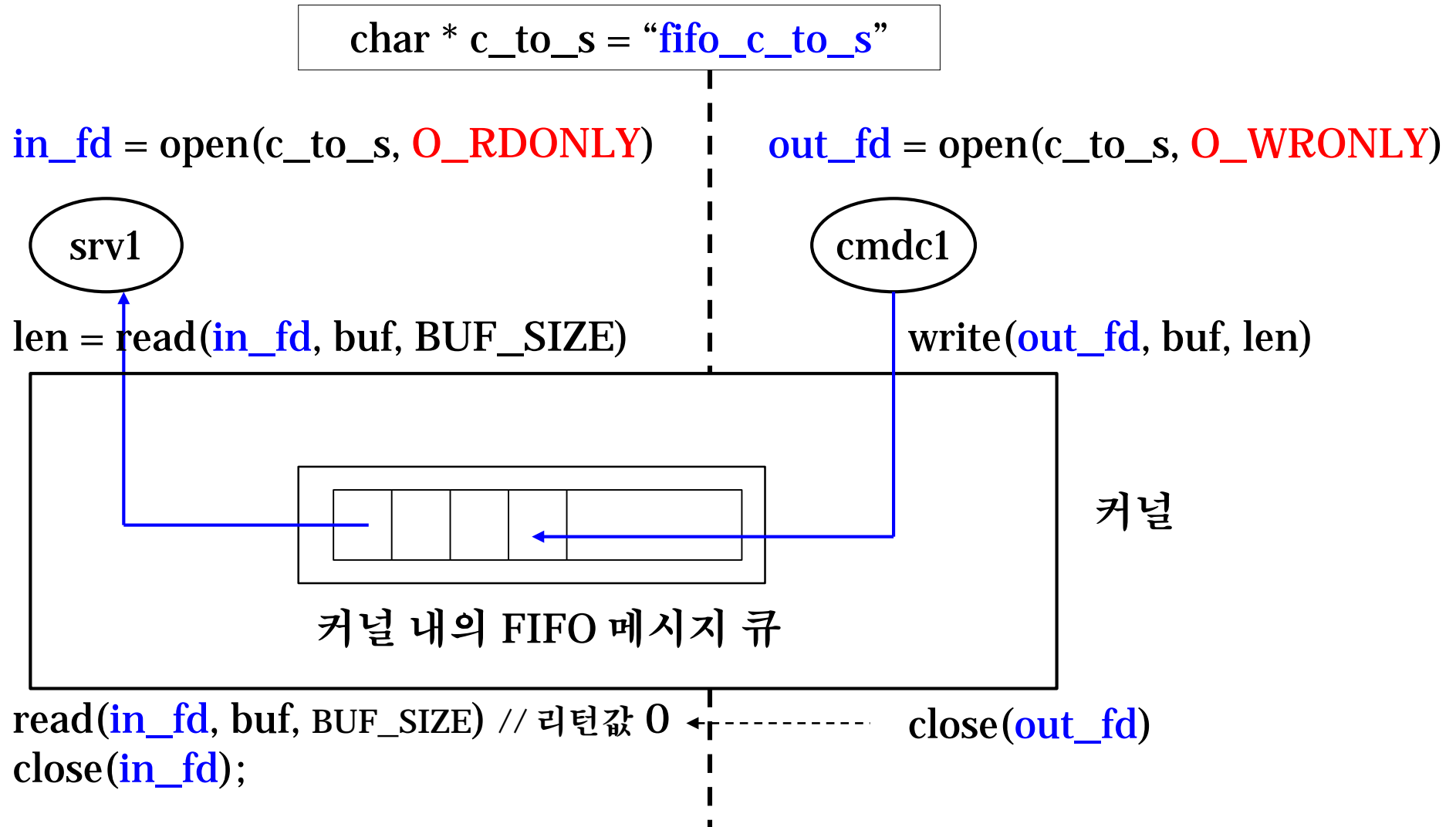


- 먼저 open()을 호출한 프로세스가 커널 내에 메시지 큐를 생성하고, 상대방이 open()을 호출할 때까지 대기함
- 나중에 open()을 호출한 프로세스에 의해 먼저 호출한 프로세스는 깨어남
- FIFO 파일 이름은 두 프로세스가 만나기 위한 장소를 제공하는 역할임
- FIFO 파일을 open하면 디스크에 있는 파일을 open하는 것이 아니라, 운영체제 내의 메모리에 메시지 큐를 만든다.
- 이후에 이 큐에 read(), write()하면 이 메시지 큐에 데이터를 읽고 쓴다.

서버(srv1) -> 클라이언트(cmdc1) 한 방향 통신



클라이언트(cmdc1) -> 서버(srv1) 한 방향 통신



cmdc1.c와 srv1.c에서 공용으로 사용되는 함수, 변수

```
#define SZ_STR_BUF    256

// 열린 FIFO 파일에 대한 파일 핸들
int  in_fd, out_fd;

// 메시지 큐용으로 사용될 FIFO 파일 이름
char *s_to_c = "fifo_s_to_c";
char *c_to_s = "fifo_c_to_s";

// 에러 원인을 화면에 출력하고 프로그램 강제 종료
void print_err_exit(char *msg)
{
    perror(msg);      // 에러 메시지 출력, 예) msg: 에러 원인
    exit(1);          // 프로그램 종료
}
```

srv1.c: main()

```
int main(int argc, char *argv[])
{
    int len; // 받은 또는 보낼 메시지 길이
    char cmd_line[SZ_STR_BUF]; // 받은 메시지 저장용 버퍼
    char ret_buf[SZ_STR_BUF]; // 보낼 메시지 저장용 버퍼

    connect_to_client(); // 클라이언트 프로그램 cmdc1와 연결한다.

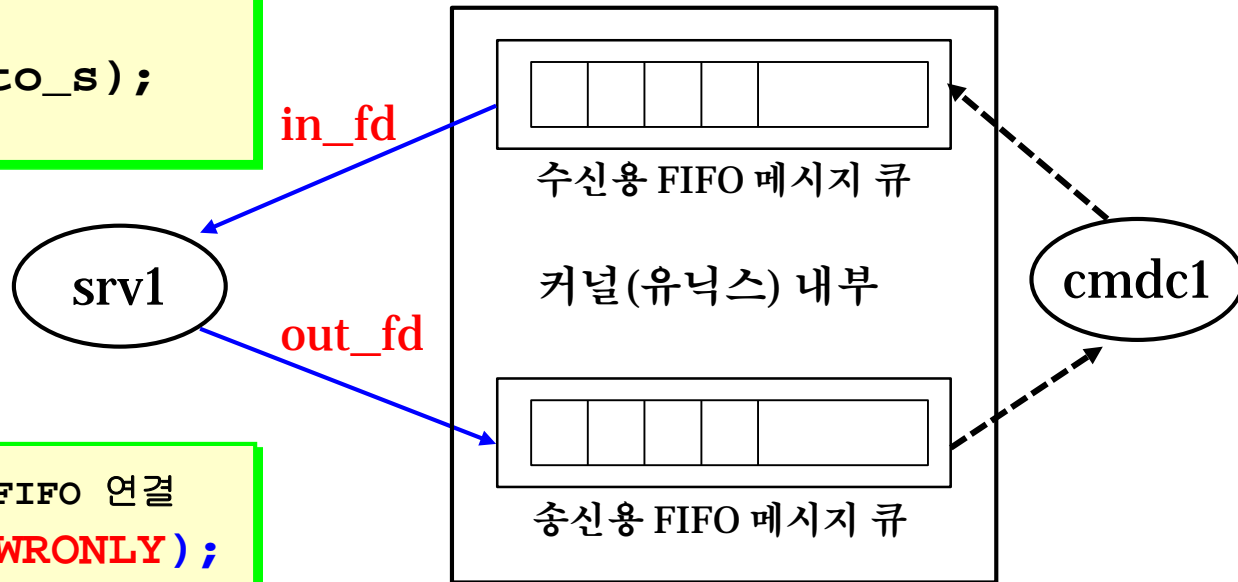
    while (1) {
        /* 클라이언트로부터 메시지 수신하는 문장을 여기에 삽입 */
        // cmdc1로부터 "exit"를 전달 받으면 서버 프로그램은 여기서 종료
        if (strncmp(cmd_line, "exit", 4) == 0)
            break; // "exit"를 받았다면 서버는 여기서 종료
        // 클라이언트에게 보낼 문자열을 작성: 받은 메시지를 다시 에코 해 줌
        sprintf(ret_buf, "server: %s", cmd_line);
        len = strlen(ret_buf); // len: 송신할 메시지 길이
        /* 클라이언트로 메시지 송신하는 문장을 여기에 삽입 */
    }
    dis_connect(); // 클라이언트 프로그램과 연결을 해제한다.
}
```

srv1.c: connect_to_client() 함수에 아래 코드 삽입

□ 서버(srv1) : 클라이언트(cmdc1)와 통신 연결

```
// 클라이언트 -> 서버: 방향 통신용 FIFO 연결  
in_fd = open(c_to_s, O_RDONLY);  
if (in_fd < 0)  
    print_err_exit(c_to_s);
```

char * c_to_s = "fifo_c_to_s"



```
// 서버 -> 클라이언트: 방향 통신용 FIFO 연결  
out_fd = open(s_to_c, O_WRONLY);  
if (out_fd < 0)  
    print_err_exit(s_to_c);
```

char * s_to_c = "fifo_s_to_c"

srv1.c, cmdc1.c: dis_connect() 함수에 아래 코드 삽입

- 상대방과 연결된 두 개 FIFO 큐의 연결 끊기
- 이 함수는 서버와 클라이언트가 모두 동일함 (아래처럼 수정할 것)

```
// 이미 연결되어 있는 두 개의 FIFO 큐의 핸들
// int in_fd, out_fd;

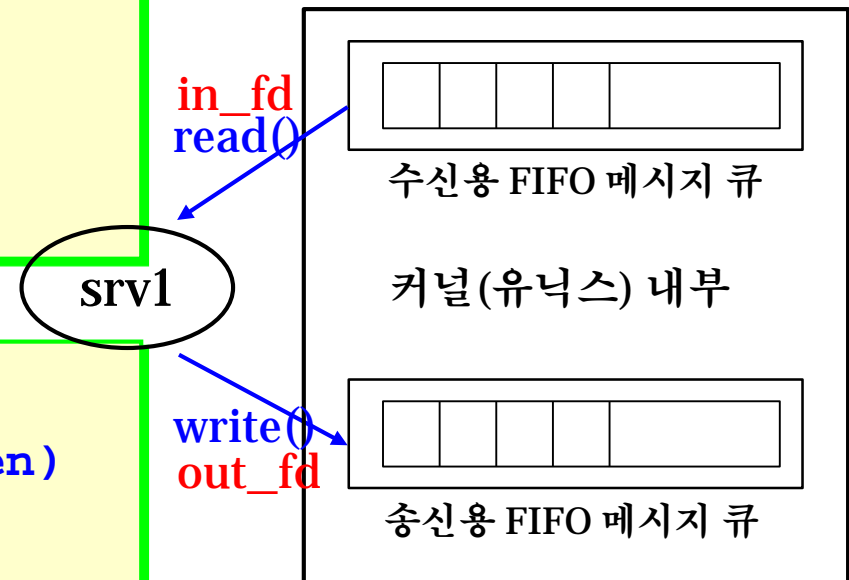
close(in_fd);
if (in_fd != out_fd)
    close(out_fd);
```

srv1.c: main() 함수에 메시지 수신 및 송신 코드 삽입

□ srv1.c의 main() 함수 내에서

```
/* 클라이언트로부터 메시지 수신하는 문장 */  
len = read(in_fd, cmd_line, SZ_STR_BUF);  
// 클라이언트가 먼저 종료되면(Ctrl+C)  
// 여기서 len은 0  
if (len <= 0) // len: 수신된 메시지 길이  
    break;  
// 수신된 메시지를 문자열로 만들어 줌  
cmd_line[len] = '\0';
```

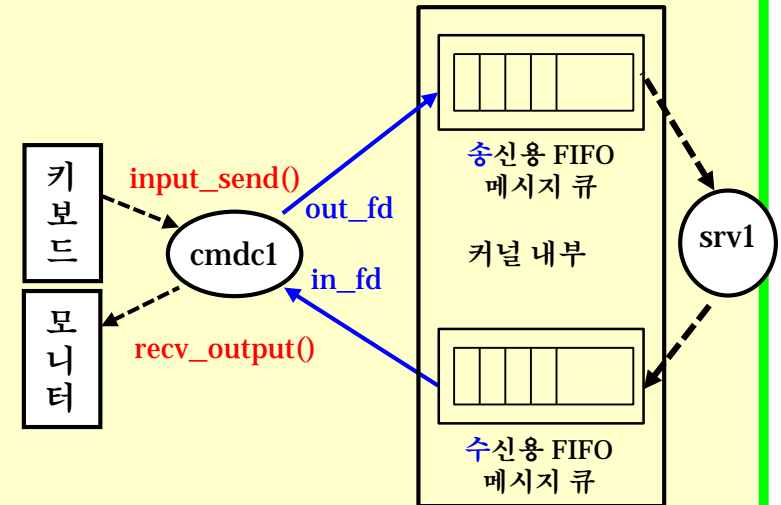
```
/* 클라이언트로 메시지 송신하는 문장 */  
if (write(out_fd, ret_buf, len) != len)  
    break;
```



cmdc1.c: main()

```
void single_process(void) // 하나의 프로세스에서 반복적으로 아래를 수행
{
    // 키보드 입력, 서버로 전송, 서버에서 수신, 화면에 출력을 순서적 처리함
    while (1) {
        // 키보드에서 읽어 서버로 보냄
        if (input_send() <= 0) break;
        // 서버에서 메시지 받아 화면에 출력
        if (recv_output() <= 0) break;
    }
}
```

```
int main(int argc, char *argv[])
{
    connect_to_server(); // 서버 프로그램 srv와 연결한다.
    single_process();    // 반복 수행: 입력, 서버로 전송, 서버에서 수신, 출력
    dis_connect();       // 서버 프로그램과 연결을 해제한다.
}
```



cmdc1.c: connect_to_server() 함수에 아래 코드 삽입

□ 서버(srv1) : 클라이언트(cmdc1)와 통신 연결

```
// 클라이언트 -> 서버: 방향 통신용 FIFO 연결  
out_fd = open(c_to_s, O_WRONLY);  
if (out_fd < 0)  
    print_err_exit(c_to_s);
```

```
char * c_to_s = "fifo_c_to_s"
```

키보드

모니터

cmdc1

out_fd

in_fd

송신용 FIFO 메시지 큐

커널(유닉스) 내부

수신용 FIFO 메시지 큐

srv1

```
// 서버 -> 클라이언트: 방향 통신용 FIFO 연결  
in_fd = open(s_to_c, O_RDONLY);  
if (in_fd < 0)  
    print_err_exit(s_to_c);
```

```
char * s_to_c = "fifo_s_to_c"
```

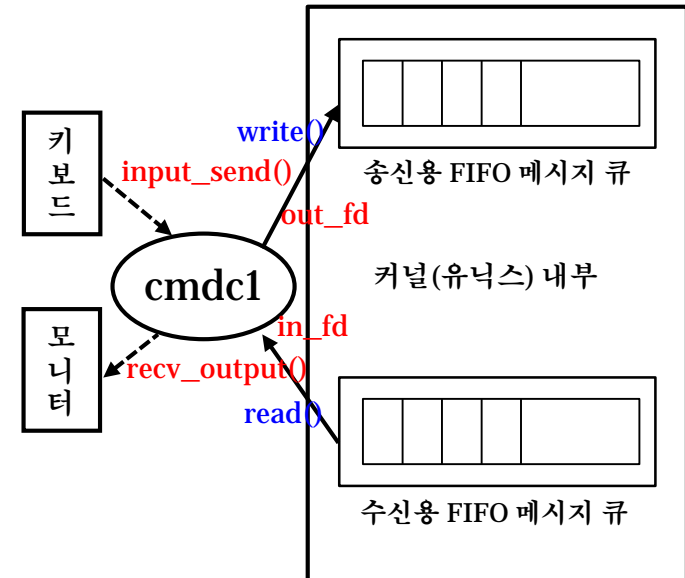
cmdc1.c: 메시지 송신 및 수신 코드 삽입

input_send(): 키보드에서 읽어 서버로 보냄

```
int len;
char cmd_line[SZ_STR_BUF];
// 0: STDIN_FILENO, 키보드에서 읽고
len = read(0, cmd_line, SZ_STR_BUF);
if (len <= 0) return len;
/* 서버로 전송하는 코드 */
if (write(out_fd, cmd_line, len) != len)
    return -1; // 서버가 종료한 경우
return len;
```

recv_output(): 서버에서 메시지 받아 화면에 출력

```
int len;
char cmd_line[SZ_STR_BUF];
/* 서버로부터 수신하는 코드 */
len = read(in_fd, cmd_line, SZ_STR_BUF);
if (len <= 0) return len; // 서버가 종료한 경우
// 화면에 출력, 1: STDOUT_FILENO
if (write(1, cmd_line, len) != len)
    return -1;
return len;
```



서버와 클라이언트의 종료

- 위 두 프로그램이 실행 중에 세 번째 터미널에서 “ps -u 자신의계정이름” 실행
 - 사용자가 실행시킨 모든 프로세스들의 리스트를 볼 수 있음; cmdc1과 srv1이 있어야 함
- 두 프로세스를 한번에 종료시키는 세가지 방법
 1. cmdc1에서 Ctrl+C를 누름
 - cmdc1이 종료되고 이어 srv1이 종료됨 (당분간 이 방법 추천)
 2. cmdc1에서 “exit[enter]”입력 또는 srv1에서 Ctrl+C 누름
 - 서버와 클라이언트 모두 종료하지만
 - srv1에서 Ctrl+C 누른 경우 클라이언트(cmdc1)가 바로 종료하지 않음
 - 다시 한번 입력 후 엔터를 입력해야 그제서야 종료함 (일종의 버그임 -> 추후에 수정할 예정임)

```
jhshim1@esl: ~/up/IPC
jhshim1:[~/up/IPC] $ srv1
```

Ctrl+C 입력

```
jhshim1@esl: ~/up/IPC
jhshim1:[~/up/IPC] $ cmdc1
This is 연습 이다 .
server: This is 연습 이다 .
오우 좋은데
server: 오우 좋은데
직 인 다 .
server: 직 인 다 .
```

Ctrl+C 또는
“exit[enter]”입력

```
jhshim1@esl: ~
jhshim1:[~] $ ps -u jhshim1
```

PID	TTY	TIME	CMD
166174	?	00:00:00	systemd
166177	?	00:00:00	(sd-pam)
166248	?	00:00:00	sshd
166249	pts/8	00:00:00	bash
166297	?	00:00:00	sshd
166298	pts/9	00:00:00	bash
166347	?	00:00:00	sshd
166348	pts/10	00:00:00	bash
166634	pts/9	00:00:00	cmdc1
166646	pts/8	00:00:00	srv1
166690	pts/10	00:00:00	ps

```
jhshim1:[~] $
```

프로그램을 강제로 종료하는 방법

- ❑ srv1나 cmdc1가 정상적으로 종료되지 않고 둘 중 하나라도 살아 있으면 두 프로그램을 다시 실행시킬 경우 앞 전에 실행시킨 프로그램으로 인해 정상 실행되지 않음

앞 전 프로그램이 FIFO 파일을 open하여 메시지 큐에 접속되어 있는 상태이기 때문에 뒤에 실행시킨 프로그램이 접속되지 않음

- ❑ 앞 전에 비 정상적으로 프로그램이 종료되었을 경우 재실행 전에 ps 명령어로 확인할 것

`$ ps -u jhshim(자신의계정이름)`

srv1 또는 cmdc1이 하나라도 살아 있는지 확인

- ❑ Kill 명령어를 사용하여 srv1 또는 cmdc1 을 강제로 종료시킬 수 있음

`$ kill -9 166634 166646`

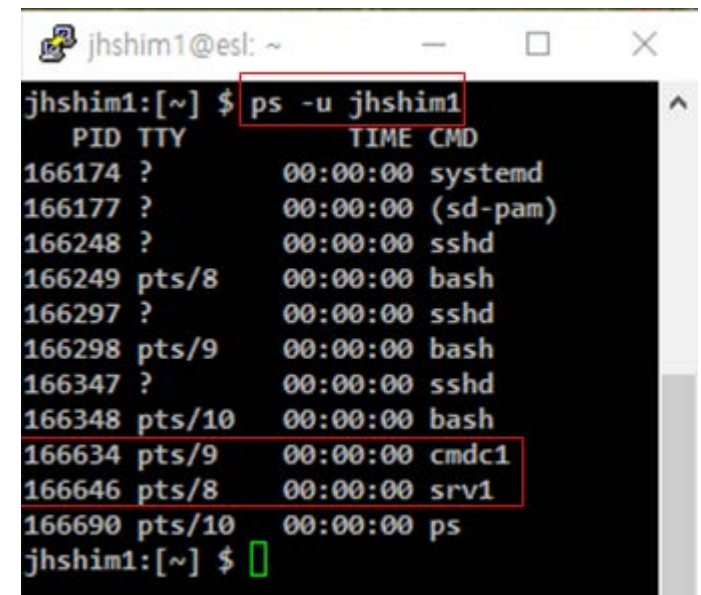
// 위에서 166634 166646는 종료시키고자 하는 proces들의 ID들, 즉 PID 값

// -9는 kill 명령의 강제로 죽이는 옵션 값임

`$ ps -u jhshim(자신의 계정이름)`

// srv1나 cmdc1가 없어야 정상임

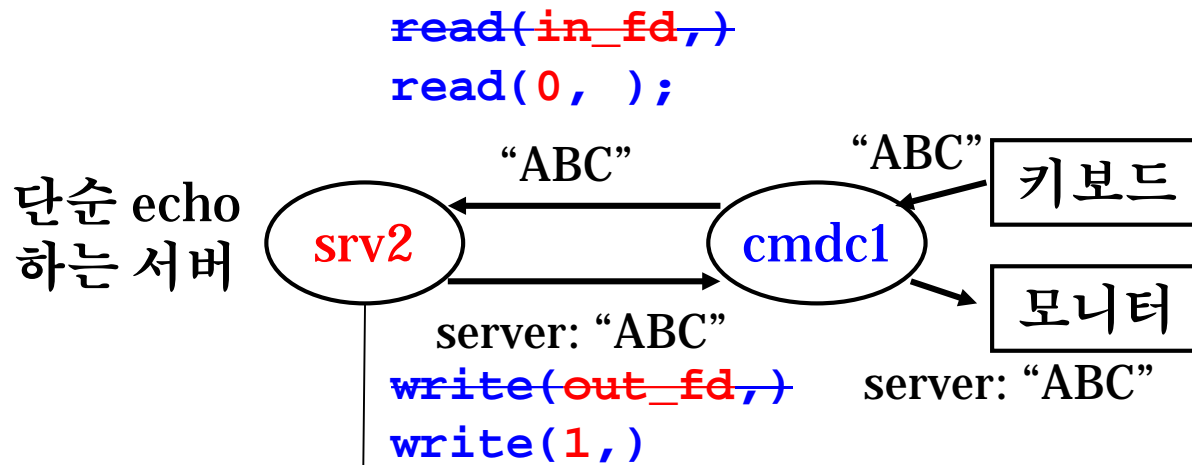
- 추후 필요한 경우 이런 방식으로 종료해야 함



```
jhshim1@esl: ~  
jhshim1:[~] $ ps -u jhshim1  
  PID TTY          TIME CMD  
166174 ?            00:00:00 systemd  
166177 ?            00:00:00 (sd-pam)  
166248 ?            00:00:00 sshd  
166249 pts/8        00:00:00 bash  
166297 ?            00:00:00 sshd  
166298 pts/9        00:00:00 bash  
166347 ?            00:00:00 sshd  
166348 pts/10       00:00:00 bash  
166634 pts/9        00:00:00 cmdc1  
166646 pts/8        00:00:00 srv1  
166690 pts/10       00:00:00 ps  
jhshim1:[~] $
```

실습 11-2

cmdc1 과 srv2



srv1과 기능은 동일함

차이점:

srv1은 in_fd, out_fd FIFO 파일 핸들을 이용해 클라이언트와 통신하였으나
srv2는 표준입출력 파일 핸들 0, 1을 통해 클라이언트와 통신함

srv2.c 파일 만들기

- ~/up/IPC 디렉토리에서 기존 파일을 복사한다.

```
$ cd ~/up/IPC
```

```
$ cp srv1.c srv2.c
```

```
$ ls
```

```
cmdc1.c  srv1.c  srv2.c  Makefile
```

- Makefile의 TARGETS에 srv2를 추가

```
TARGETS = cmdc1 srv1 srv2
```

- 이후 수정 사항들은 **srv2.c**에서 수정할 것

srv2.c: main() 수정: 표준 입출력으로부터 읽고 쓰기

```
while (1) {  
    /* 클라이언트로부터 메시지 수신하는 문장 */  
    len = read(in_fd, cmd_line, SZ_STR_BUF);  
    ...  
    /* 클라이언트로 메시지 송신하는 문장 */  
    if (write(out_fd, ret_buf, len) != len)  
        break;  
}
```

위 내용을 아래 내용으로 수정할 것

```
while (1) {    // 0: STDIN_FILENO 키보드에서 읽기  
    /* 키보드로부터 메시지 입력하는 문장 */  
    len = read(0, cmd_line, SZ_STR_BUF);  
    ...  
    /* 화면에 메시지 출력하는 문장 */  
    if (write(1, ret_buf, len) != len)  
        break; // 1: STDOUT_FILENO 화면에 출력  
}
```

in_fd
read()

srv2

write()
out_fd

송신용 FIFO 메시지 큐

커널(유닉스) 내부

수신용 FIFO 메시지 큐

0
read()

키보드

srv2

write()
1

모니터

srv2.c: main()

□ Main()에서 클라이언트와 접속 및 차단을 주석처리

```
int main(int argc, char *argv[]) {  
    // connect_to_client(); // 이 문장을 주석 처리 하라  
    while (1) {  
        ... // 앞 페이지 수정된 read(), write()  
    }  
    // dis_connect(); // 이 문장을 주석 처리 하라  
}
```

□ make한 후 srv2만 단독 실행

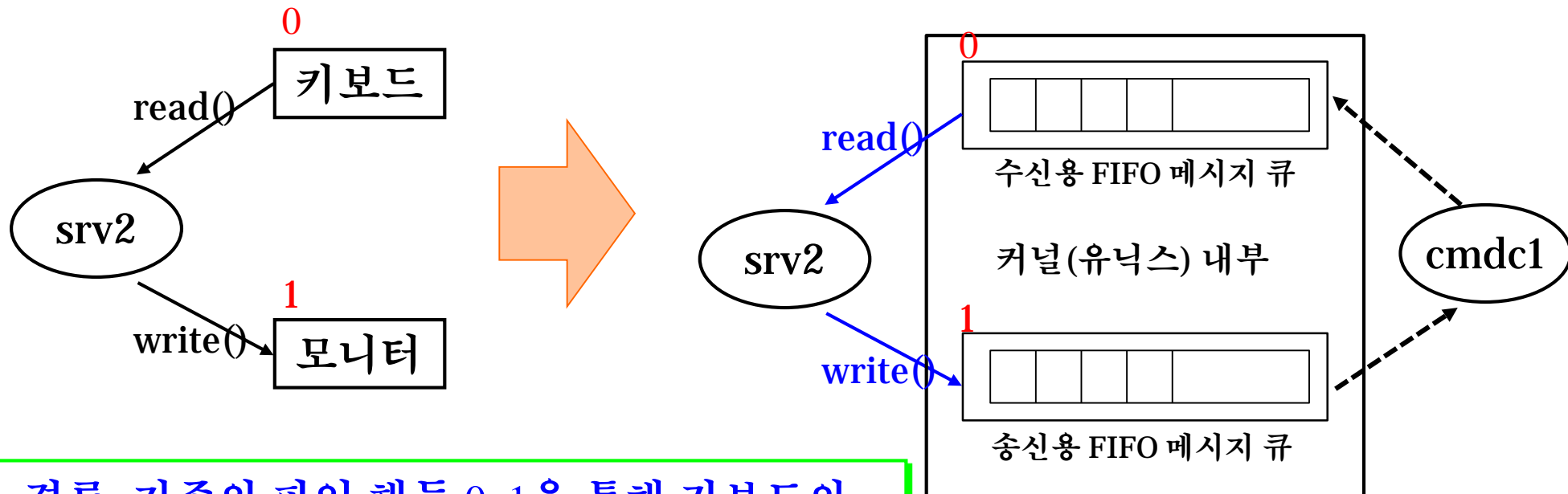
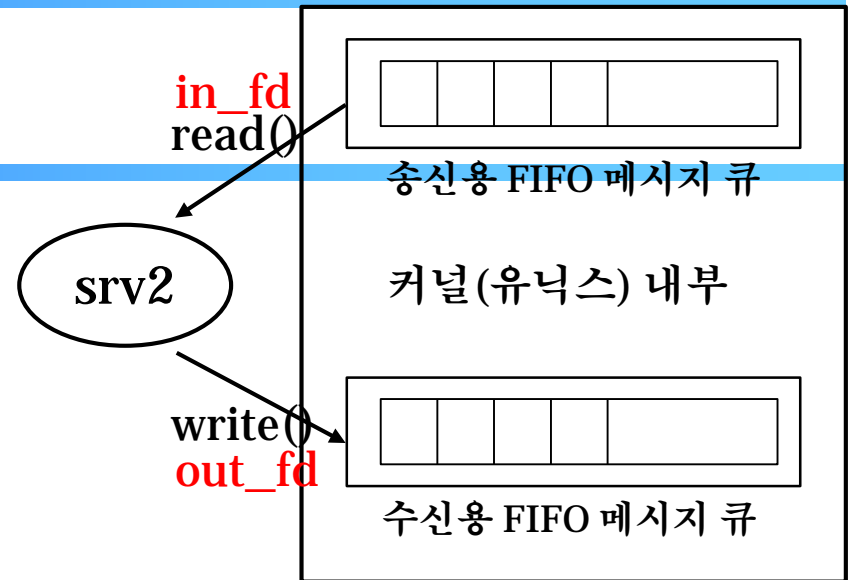
키보드에서 입력 받아 화면에 출력한다.

아래처럼 실행하되 종료 시 Ctrl+C 또는 exit 입력



표준 입출력 장치 접속 교체

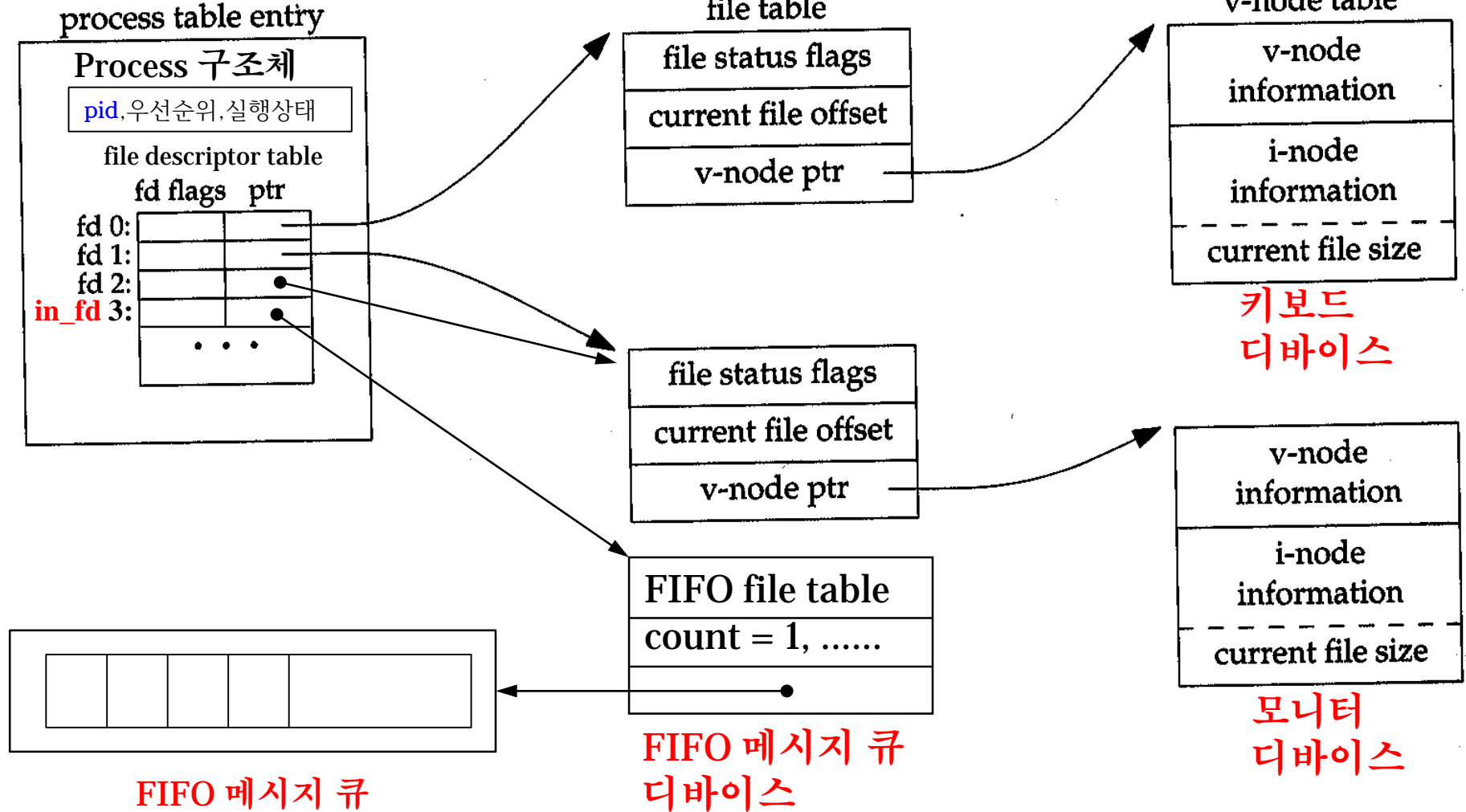
- 파일 핸들 **0**과 접속된 기존의 키보드 장치를 FIFO용 메시지 큐로 접속 변환
- 파일 핸들 **1**과 접속된 기존의 모니터 장치를 FIFO용 메시지 큐로 접속 변환



결론: 기존의 파일 핸들 0, 1을 통해 키보드와 모니터 대신 클라이언트와 통신 가능

Tables for Open Files: UNIX 운영체제 내부의 모습

srv2 프로세스 정보 테이블

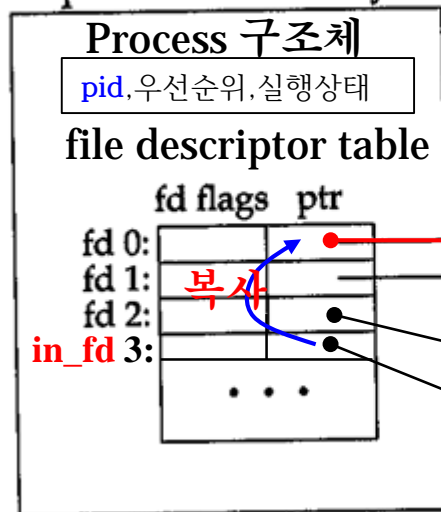


File Descriptor Table Entry의 복사: dup2(), **교재** p.99

❑ `int dup2(int fd1, int fd2);` 예) `dup2(3, 0)`

- 만약 fd1과 fd2가 같은 값이면, 바로 리턴함
- fd2:0가 기존에 열려진 파일이라면, fd2:0를 먼저 **close(fd2:0)** 한다.
- fd1:3이 포인트하는 포인터를 fd2:0에 **복사**한다. => 디바이스 중복(공유)

srv2 프로세스 정보 테이블
process table entry



close(0)

키보드 file table
count = 0,

키보드
디바이스

모니터 file table
count = 2,

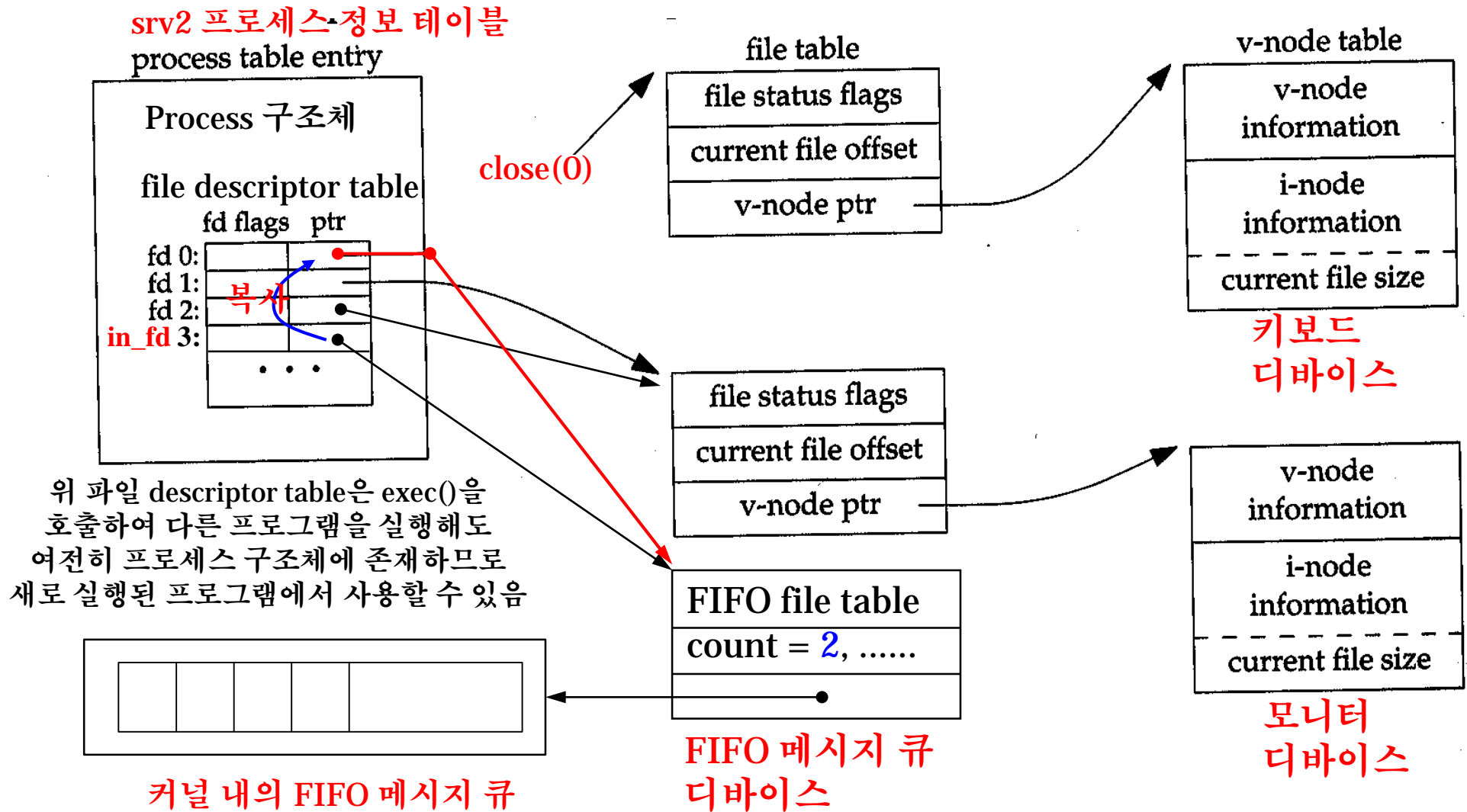
모니터
디바이스

FIFO file table
count = 2,

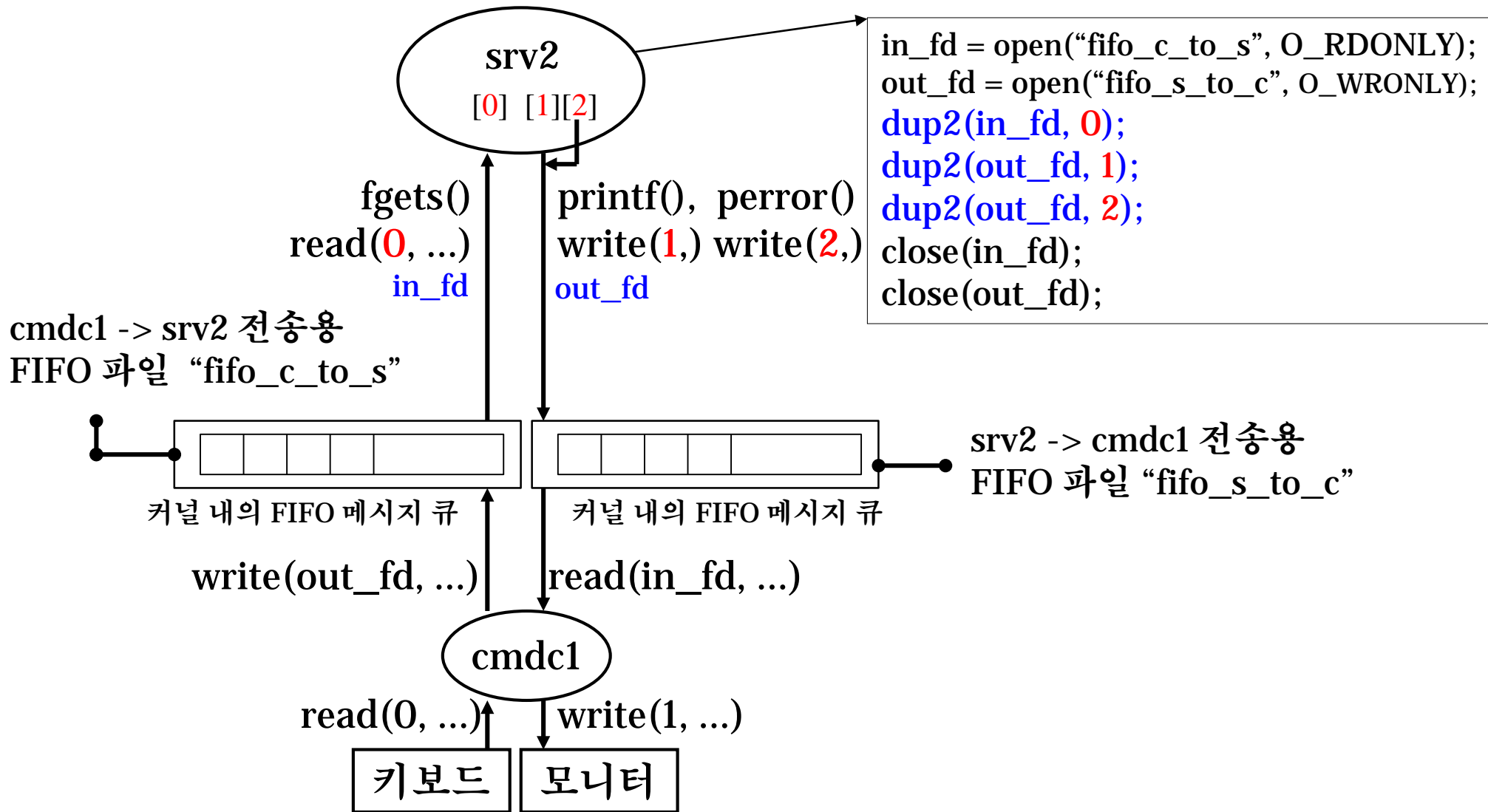
FIFO 메시지 큐
디바이스

in_fd

dup2(3, 0)을 실행한 후의 File Descriptor Table



srv2의 표준입출력, 에리출력의 장치 접속 교체 작업



srv2.c: main() 수정

- Main()에서 주석처리 했던 문장을 다시 원상 복구함
 - 클라이언트와 접속 및 연결 차단 문장 복구
- duplicate_IO() 문장 추가

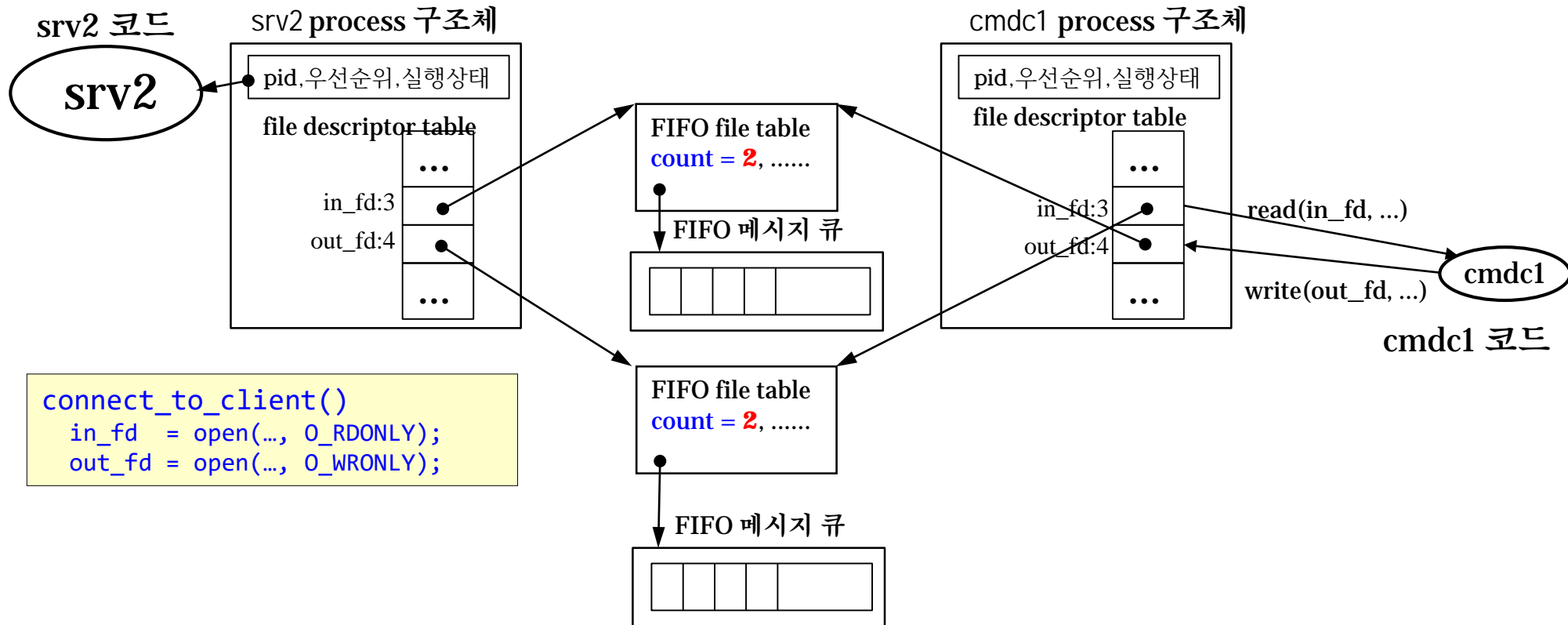
```
int main(int argc, char *argv[]) {  
    //connect_to_client(); // 주석을 제거 하라.  
    duplicate_IO();           // 표준 입출력, 에러 출력 장치 접속 교체 작업  
  
    while (1) {  
        read(0, ...);         // 이제 클라이언트로부터 입력 (FIFO 큐)  
        . . .  
        write(1, ...);        // 클라이언트로 출력 (FIFO 큐)  
    }  
    //dis_connect();       // 주석을 제거 하라.  
}
```

srv2.c: duplicate_IO() 추가

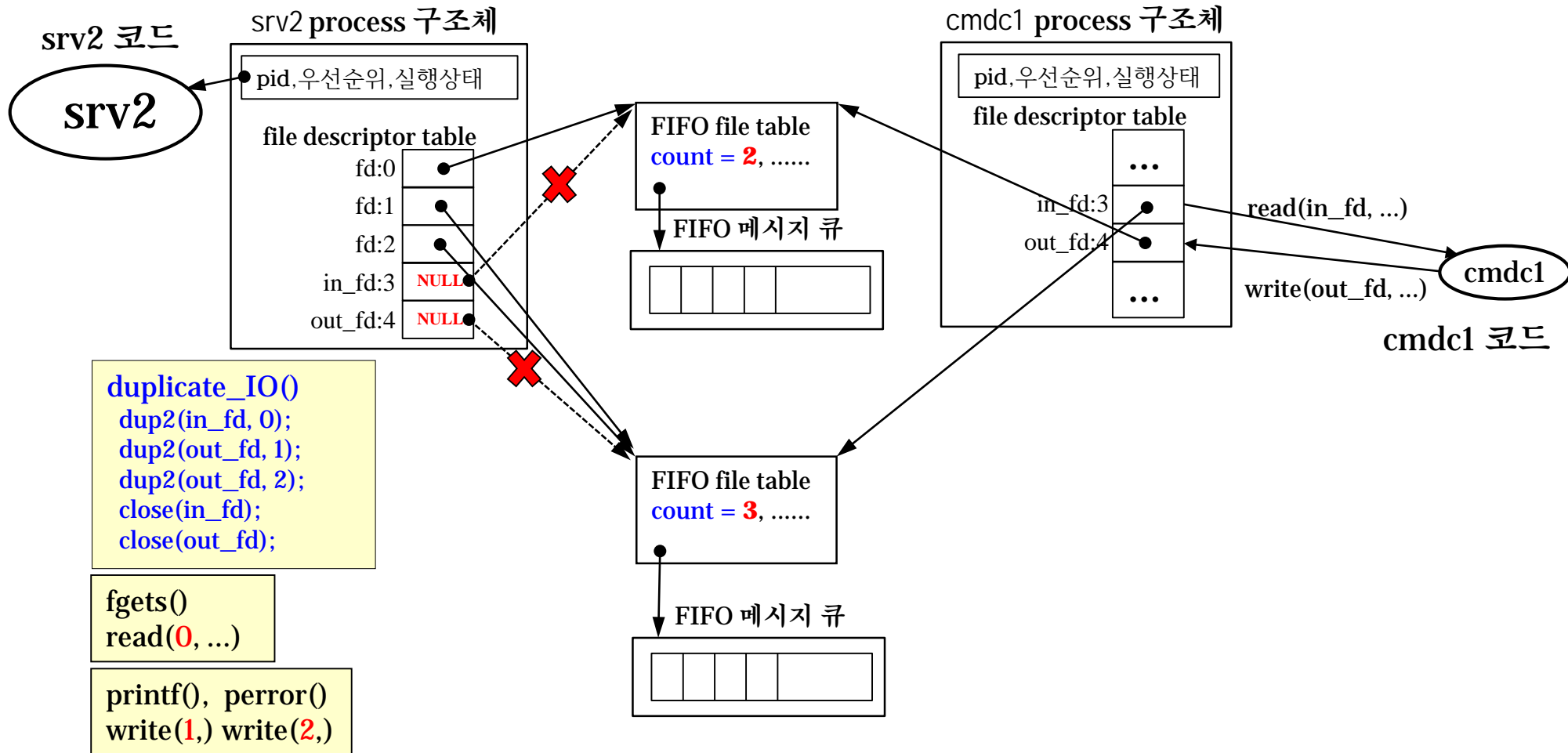
□ Main() 함수 바로 위에 아래 함수 추가

```
void duplicate_IO() {  
    // 서버의 표준 입력을 수신용 FIFO 파일 핸들로 변경하고  
    // 표준 출력, 에러출력을 송신용 FIFO 파일 핸들로 변경한다.  
    dup2(in_fd, 0); // 0: STDIN_FILENO   서버 <- 클라이언트  
    dup2(out_fd, 1); // 1: STDOUT_FILENO  서버 -> 클라이언트  
    dup2(out_fd, 2); // 2: STDERR_FILENO  서버 -> 클라이언트  
  
    // 표준 입출력 버퍼를 없앤다. 입출력이 중간에 버퍼링 되지 않고 바로 I/O됨  
    setbuf(stdin, NULL); // scanf(), fgets(..., stdin)에서 사용  
    setbuf(stdout, NULL); // printf()에서 사용  
    // stderr는 기본적으로 버퍼링 되지 않음  
  
    // 표준 입출력 0, 1, 2 핸들에 복제 되었으므로 이제 in_fd, out_fd는 필요 없음  
    dis_connect(); // close(in_fd); close(out_fd);  
    // 표준 I/O 핸들 0, 1, 2를 통해 클라이언트와 연결되어 있으므로  
    // close()해도 실제로 클라이언트와의 접속이 끊어지는 것은 아님  
}
```

connect_to_client() 후의 운영체제 내부의 모습



duplicate_IO() 이후의 커널(운영체제) 내부의 모습



서버와 클라이언트의 실행

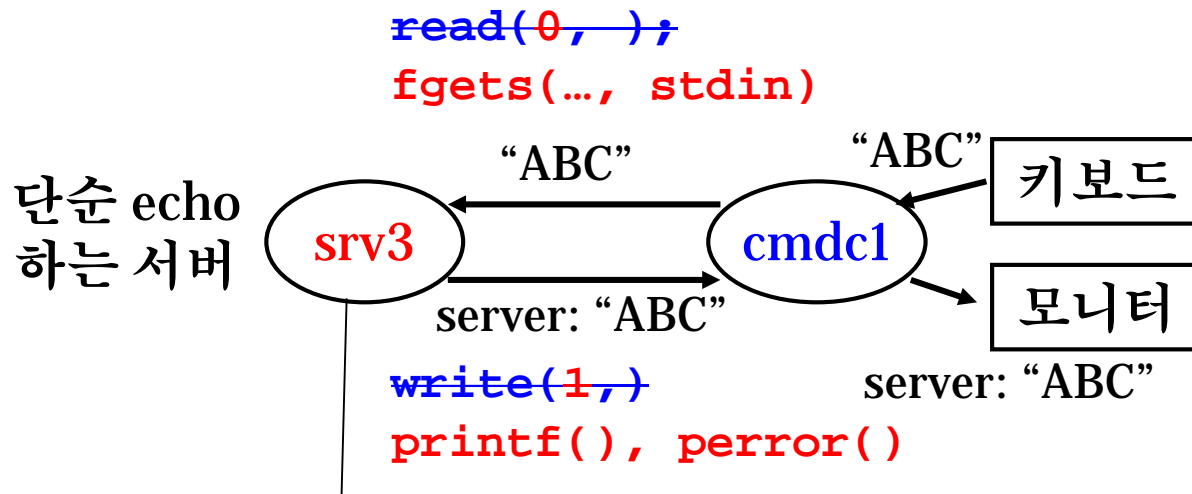
- ❑ make 실행한 후 터미널을 두 개 실행한다.
- ❑ 한쪽 터미널에서 서버 `srv2`을 먼저 실행
 - 클라이언트(cmdc1)가 보내준 메시지를 다시 cmdc1에게 에코해 줌
- ❑ 다른 쪽 터미널에서 `cmdc1`을 실행한 후 키보드에서 메시지를 한 줄씩 입력 후 엔터를 친다. 이를 계속 반복한다.



The image shows two terminal windows. The top window has the title bar 'jhshim1@esl: ~/up/IPC' and the prompt 'jhshim1: [~/up/IPC] \$'. The command 'srv2' is entered and highlighted with a red box. The bottom window also has the title bar 'jhshim1@esl: ~/up/IPC' and the prompt 'jhshim1: [~/up/IPC] \$'. The command 'cmdc1' is entered and highlighted with a red box. Below the prompt, the following text is displayed: 'This is 연습 이 다 .', 'server: This is 연습 이 다 .', '오 우 좋 은 데', 'server: 오 우 좋 은 데', '직 인 다 .', 'server: 직 인 다 .'. A green cursor is visible at the end of the last line.

실습 11 – 3

cmdc1 과 srv3



srv2과 기능은 동일함

차이점: srv2는 `read()`와 `write()` 함수를 통해 클라이언트와 통신하였으나, srv3는 표준입출력 함수 `fgets()`, `scanf()`, `printf()`, `perror()` 등을 이용하여 클라이언트와 통신함

srv3.c 파일 만들기

- ~/up/IPC 디렉토리에서 기존 파일을 복사한다.

```
$ cd ~/up/IPC
```

```
$ cp srv2.c srv3.c
```

```
$ ls
```

```
cmdc1.c  srv1.c  srv2.c  srv3.c  Makefile
```

- Makefile의 TARGETS에 srv3를 추가

```
TARGETS = cmdc1 srv1 srv2 srv3
```

- 이후 수정 사항들은 **srv3.c**에서 수정할 것

기존 FIFO 파일 삭제

- 아래처럼 기존에 생성했던 두개의 통신용 FIFO 파일을 삭제

```
$ rm fifo_c_to_s fifo_s_to_c
$ ls
// fifo_c_to_s    fifo_s_to_c 파일이 없어야 함
```

- 이제부터는 사용자가 직접 FIFO 파일을 생성하지 않고 서버 프로그램에서 FIFO 파일을 생성하게 할 예정임

프로그램 내에서 FIFO 파일 만들기: 교재 p.676

- `srv3.c`의 `connect_to_client()` 함수의 맨 앞에 아래 코드를 삽입
 - 서버 시작 시 두 개의 FIFO 파일 생성함
 - 없으면 새로 생성, 있으면 에러 (그냥 무시하면 됨)

```
// 서버와 클라이언트가 통신할 두 개의 FIFO 파일을 생성함.  
// char * c_to_s = "fifo_c_to_s"  
// char * s_to_c = "fifo_s_to_c"  
// 0600: 파일의 접근권한 rw-----  
mkfifo(c_to_s, 0600); // 클라이언트 -> 서버로 전송할 FIFO 파일  
mkfifo(s_to_c, 0600); // 서버 -> 클라이언트로 전송할 FIFO 파일
```

- 위 `mkfifo()` 함수를 `man -s3`로 검색하여 필요한 헤드 파일 삽입
- `make`한 후 `srv3`와 `cmdc1`을 실행한 후 정상 종료
- `ls -l`하여 두 개의 FIFO 파일이 정상적으로 생성되었는지 확인

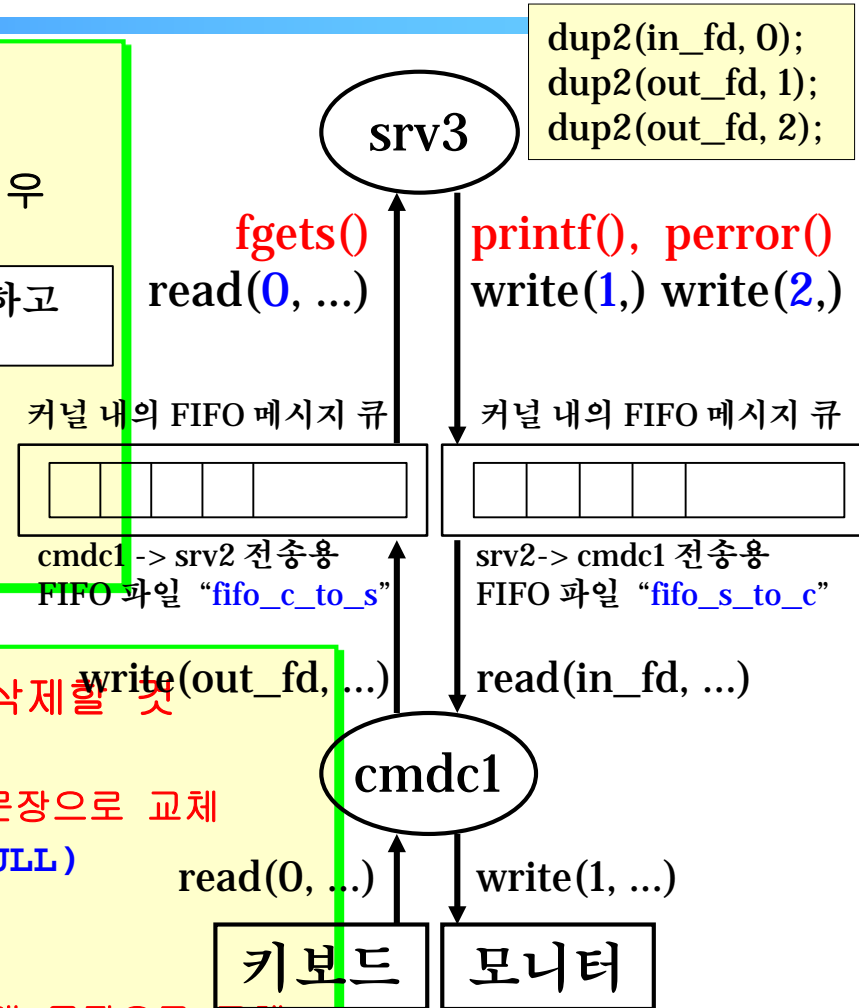
srv3.c: main(), 표준 입출력으로부터 읽고 쓰기

```
while (1) { // 클라이언트로부터 메시지 수신
    len = read(0, cmd_line, SZ_STR_BUF);
    if (len <= 0) break; // 클라이언트가 죽은 경우
    cmd_line[len] = '\0';
    ...
    sprintf(...);
    len = strlen(...); // 클라이언트로 메시지 송신
    if (write(1, ret_buf, len) != len)
        break; // 클라이언트가 죽은 경우
}
```

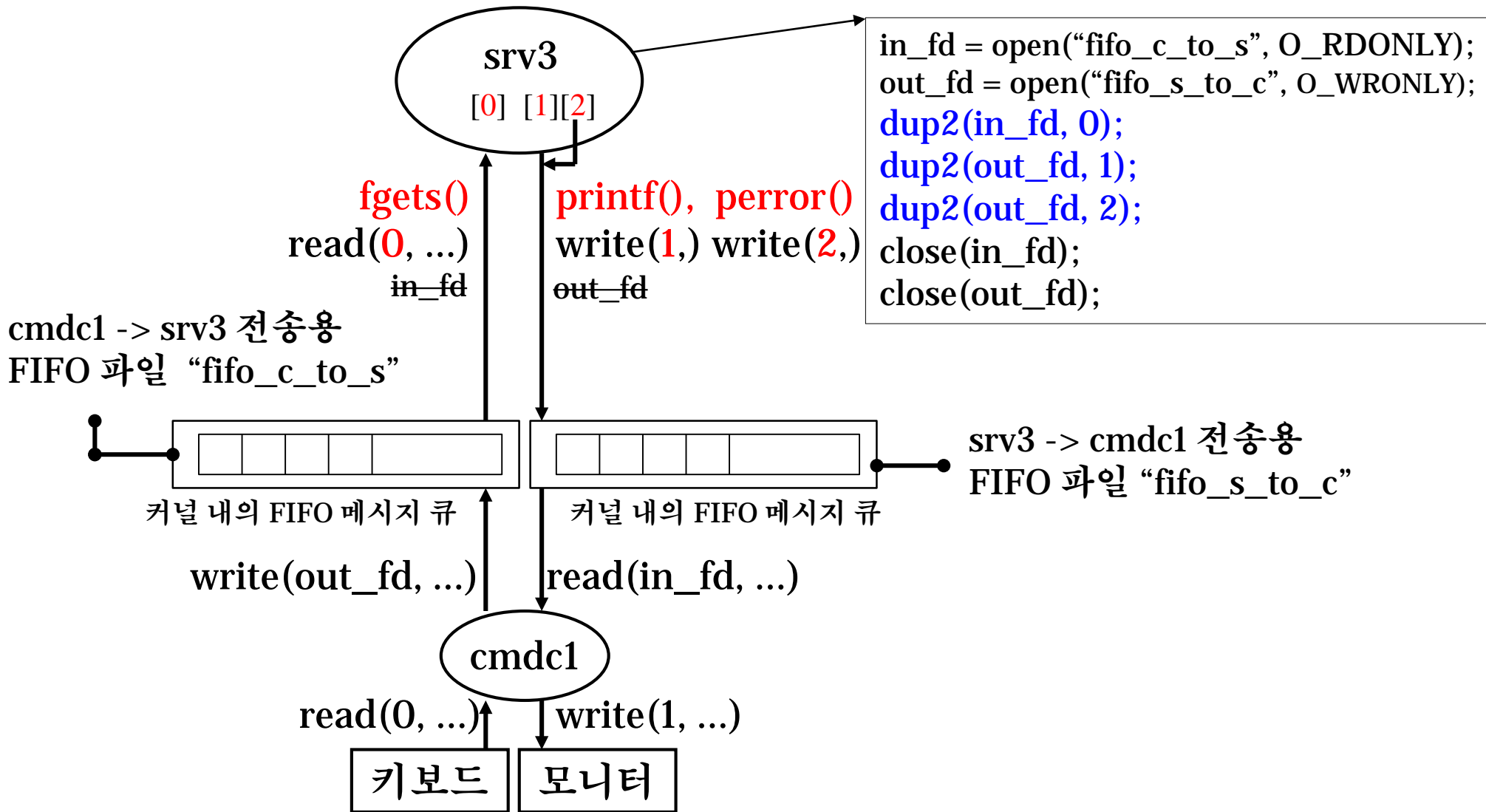
빨간색 으로 표시된 문장들 모두 삭제하고
아래 파란색 내용으로 수정할 것

위 내용을 아래 내용으로 수정할 것

```
int len; char ret_buf[SZ_STR_BUF]; 이 문장 삭제할 것
while (1) {
    // 위 read(),break,cmd_line[] 문장들을 삭제하고 아래 문장으로 교체
    if (fgets(cmd_line, SZ_STR_BUF, stdin) == NULL)
        break; // 클라이언트가 종료한 경우 NULL 리턴
    ...
    // 위 sprintf(),strlen(),write() 문장들을 삭제하고 아래 문장으로 교체
    printf("server: %s", cmd_line);
}
// 이렇게 하는 이유: cmd와 같은 방식으로 I/O 하기 위해
```

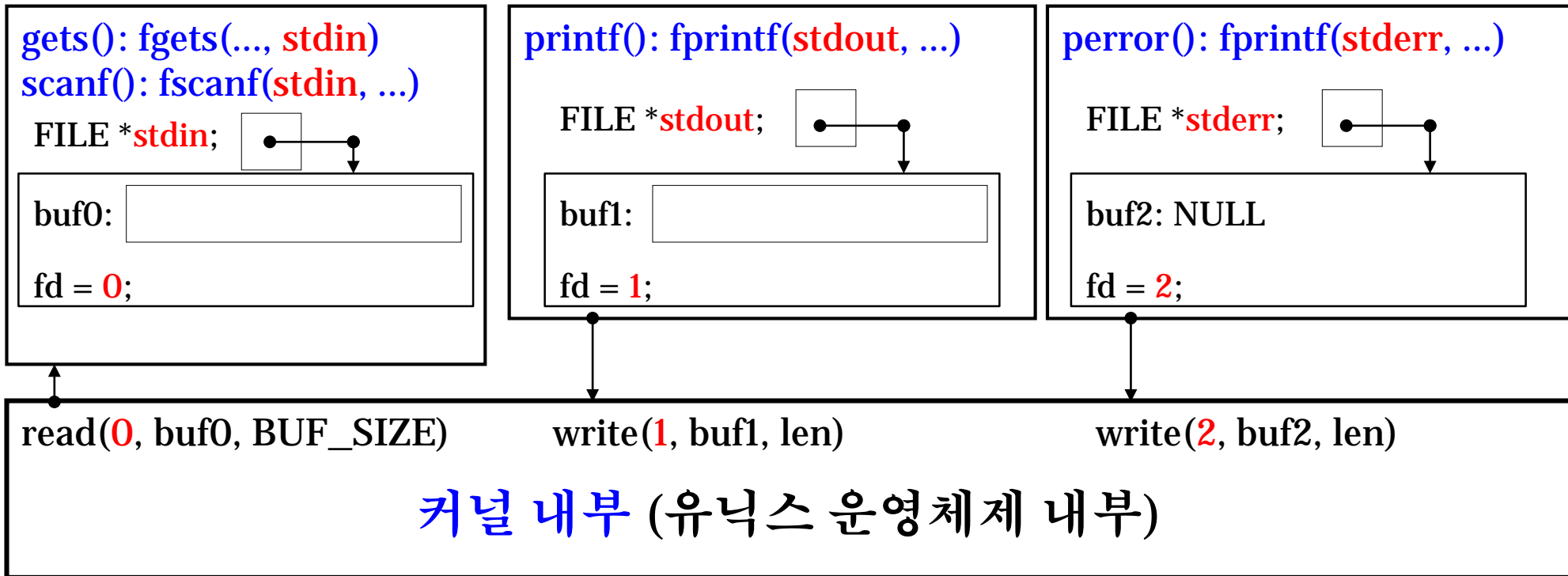


fgets(), printf(), perror()이 어떻게 FIFO 파일로 통신이 되나?



gets(), printf(), perror() 구조

프로그램에 포함된 라이브러리 함수들

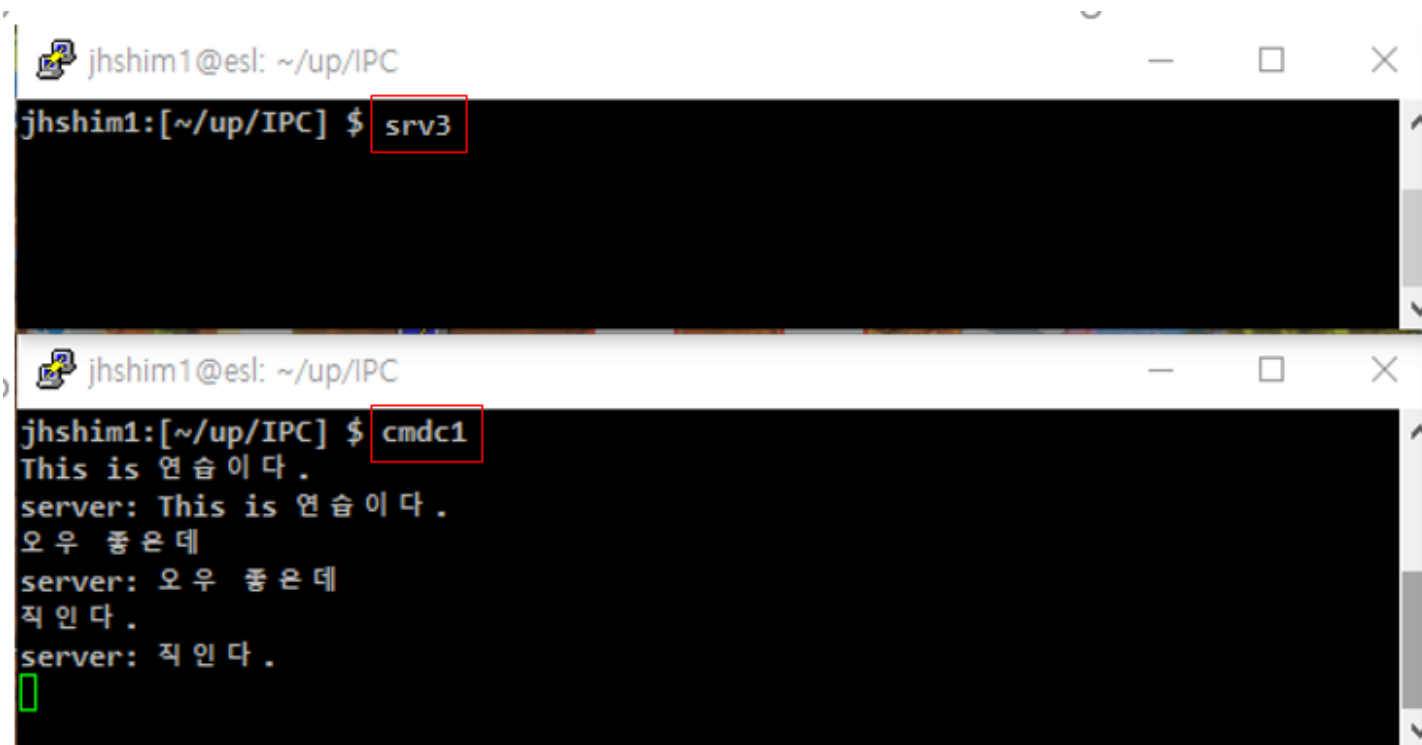


```
setbuf(stdin, NULL);  
setbuf(stdout, NULL);  
setbuf(stderr, NULL);
```

```
// 위의 stdin, stderr, stdout의 buf을 없애고  
// 버퍼링하지 말라는 의미  
// 퍼버링을 하지 않으면 바로 read() write() 함수 호출함
```

서버와 클라이언트의 실행

- ❑ make 실행한 후 터미널을 두 개 실행한다.
- ❑ 한쪽 터미널에서 서버 `srv3`을 먼저 실행
 - 클라이언트(cmdc1)가 보내준 메시지를 클라이언트에게 단순 에코 해 줌
- ❑ 다른 쪽 터미널에서 `cmdc1`을 실행한 후 키보드에서 메시지를 한 줄씩 입력 후 엔터를 친다. 이를 계속 반복한다.



The image shows two terminal windows. The top window has the title bar 'jhshim1@esl: ~/up/IPC' and the prompt 'jhshim1: [~/up/IPC] \$'. The command 'srv3' is entered and highlighted with a red box. The bottom window also has the title bar 'jhshim1@esl: ~/up/IPC' and the prompt 'jhshim1: [~/up/IPC] \$'. The command 'cmdc1' is entered and highlighted with a red box. Below the prompt, the following text is displayed: 'This is 연습 이 다 .', 'server: This is 연습 이 다 .', '오 우 좋 은 데', 'server: 오 우 좋 은 데', '직 인 다 .', 'server: 직 인 다 .'. A green cursor is visible at the end of the last line.