

실습 15

실습 12의 연장: cmdc_thread

Jaehong Shim

Dept. of Computer Engineering



cmdc_thread.c 파일 만들기

- ~/up/ IPC 디렉토리에서 기존 파일을 복사한다.

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

```
$ cp cmdc2.c cmdc_thread.c
```

```
$ ls
```

```
cmdc1.c cmdc2.c cmdc3.c cmdc.c cmdc_thread.c ...  
Makefile
```

- Makefile에 추가

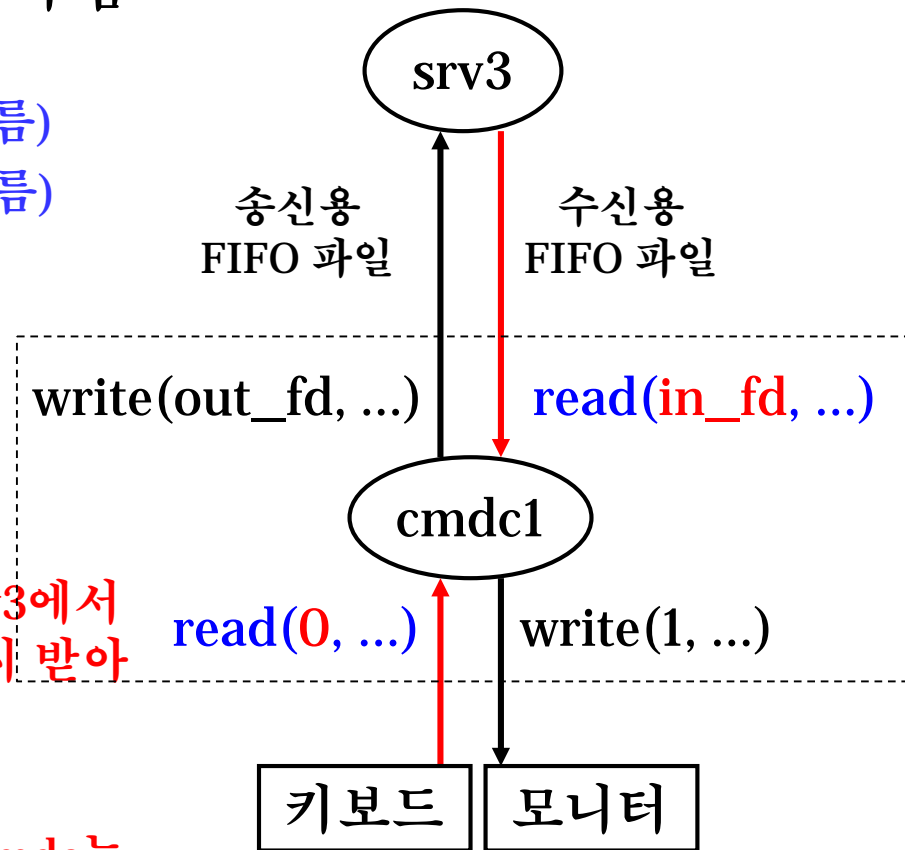
- CFLAGS에 `pthread`를 추가, 링크할 때 스레드 라이브러리 포함시킴
- TARGETS에 `cmdc_thread`를 추가

```
CFLAGS = -Wall -pthread
```

```
TARGETS = ... 기존과 동일 cmds cmdsd cmdc_thread
```

cmdc1 클라이언트 프로그램의 문제점

- ❑ cmdc1는 두 곳에는 들어오는 입력을 받아야 함
(어디에서 먼저 데이터가 들어 올까?)
 - 키보드 입력 데이터 (언제 입력 될지 모름)
 - Srv3가 전송한 데이터 (언제 수신 될지 모름)
- ❑ read(0, ...) 함수의 한계
 - 키보드에 도착한 데이터가 있으면 데이터를 가지고 바로 리턴
 - 도착한 데이터가 없으면 데이터가 도착할 때까지 **블록킹 상태에서 계속 대기**
 1. 문제는 블록킹 상태에서 대기하는 중에 srv3에서 보낸 데이터가 도착하면, in_fd에서 이를 즉시 받아 처리할 수가 없음
 - 반대 상황도 마찬가지임
 2. 만약 서버가 여러 번 메시지를 보낼 경우 cmdc는 몇 번 읽어야 할까? (문제는 모른다는 것)



cmdc1 문제의 해결 방안

□ 프로세스 기반 해결방안

- fork() 함수를 이용 자식 프로세스를 생성
- 부모 프로세스: 키보드에서 읽어 -> 서버(cmd)로 전송
- 자식 프로세스: 서버에서 보낸 데이터 읽어 -> 화면에 출력

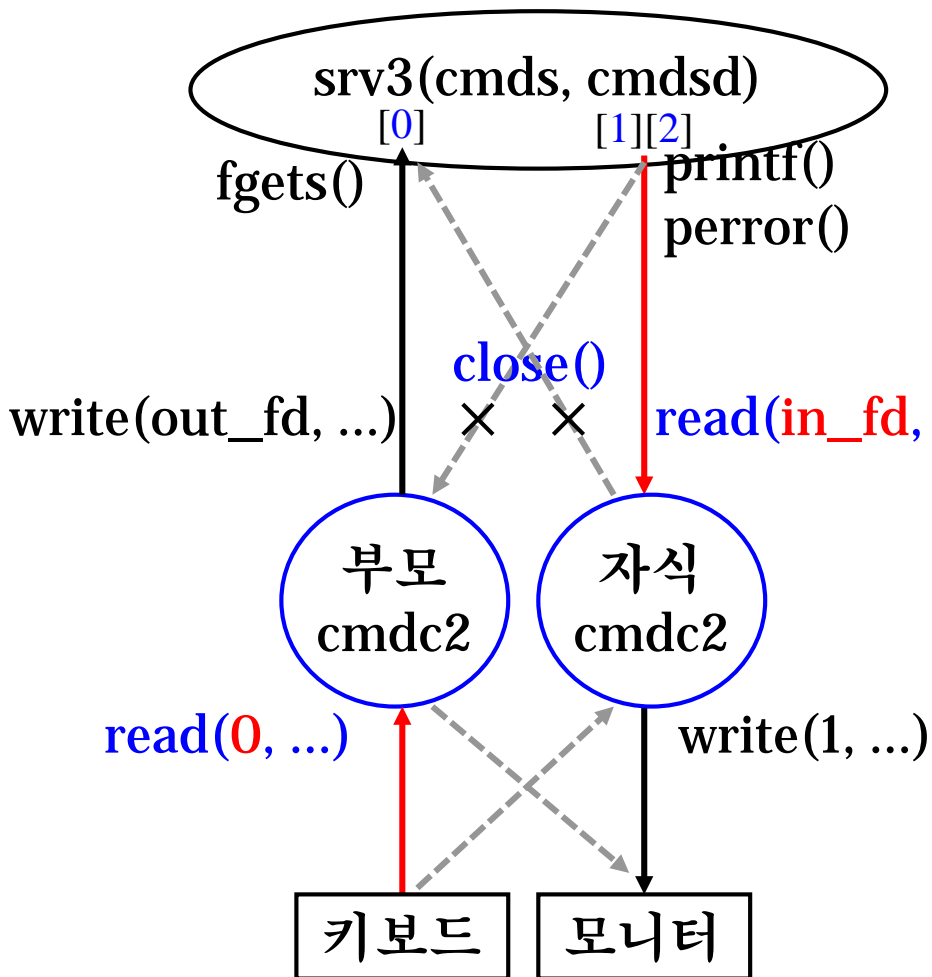
□ 쓰레드 기반 해결방안

- main 쓰레드에서 두 개의 쓰레드를 새로 생성
- InputSendThread: 키보드에서 읽어 -> 서버(cmd)로 전송 : 반복
- RecvOutputThread: 서버에서 보낸 데이터 읽어 -> 화면에 출력 : 반복

□ Select() 기반 해결방안

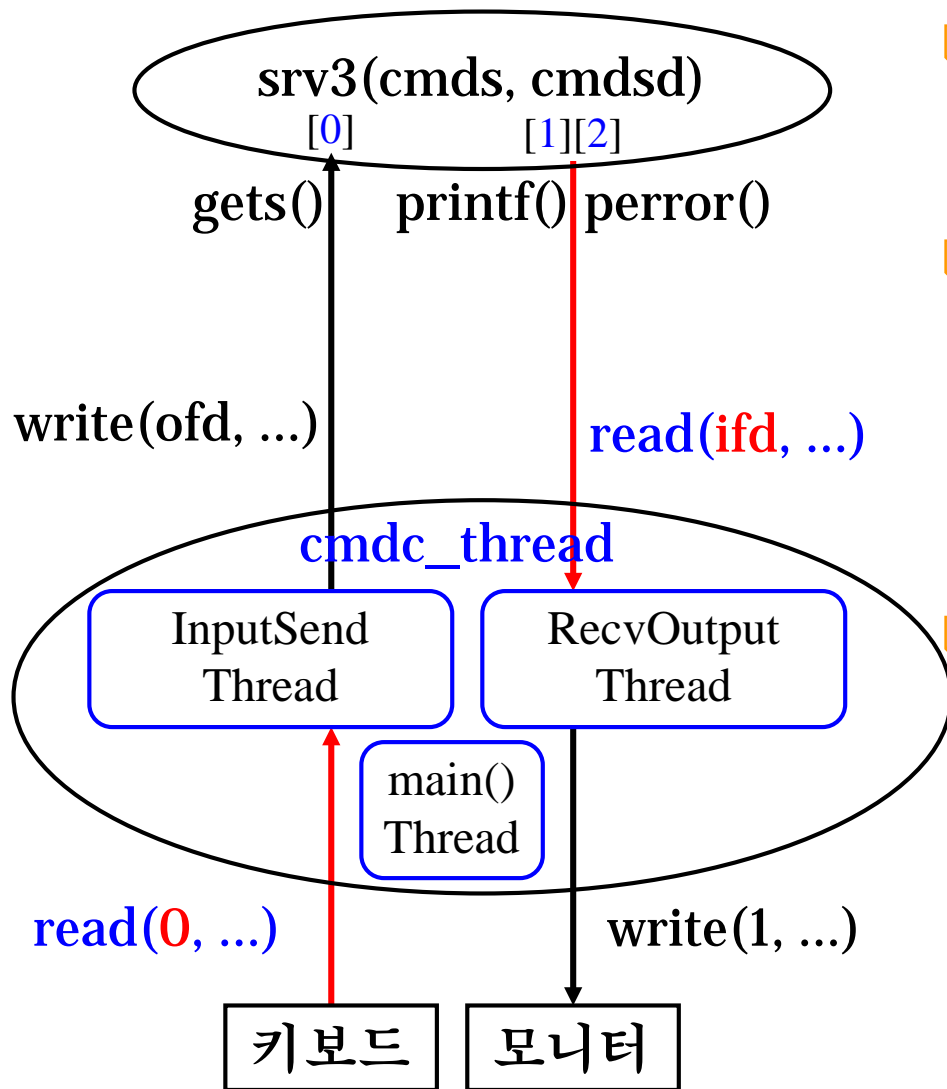
- 입력을 받을 두 개의 파일 기술자 (0, in_fd)를 미리 등록하고
- select() 함수를 호출하여 대기함
- select()는 둘 중 하나에서 데이터가 도착하면 리턴함
- 어느 쪽에서 데이터가 들어 왔는지 확인한 후 read()를 이용하여 읽음

프로세스 기반 해결방안 (프로세스 복제 cmdc2)



- 부모 cmdc2가 자기 복제 (`fork()`) 하여 자식 cmdc2를 생성
- 부모
 - 키보드에서 **대기하고 있다**가 사용자가 데이터를 입력하면 이를 읽어 서버로 전송
- 자식
 - `in_fd`에서 서버가 전송한 데이터를 **대기하고 있다**가 데이터가 도착하면 이를 수신하여 화면에 출력함
 - 서버가 계속 데이터를 보내면 이를 연속으로 받아 화면에 정상 출력하는 것이 가능함
- 부모와 자식 프로세스에서 사용하지 않는 파일 핸들은 `close()` 함

두 쓰레드 기반 해결방안 (cmdc_thread)



□ 메인 쓰레드

- RecvOutputThread가 종료되길 기다림
- 메인 쓰레드가 종료되면 전체 종료됨

□ 종료절차 1)

- “exit” 입력 -> 1-1) 서버 종료 -> 서버측 송수신 FIFO close됨 -> RecvOutputThread의 `read()` 함수 리턴 0 -> RecvOutputThread 종료 -> main 쓰레드 종료

□ 종료절차 2)

- 키보드에서 Ctrl+D(EOF) -> InputSendThread의 `read()` 함수 리턴 0 -> `close(out_fd)` 한 후 종료 -> 서버의 `gets()`가 NULL 리턴 -> 서버 종료 -> 이후는 위 1-1)과 동일

스레드 and 프로세스

- ❑ Dispatching is referred to as a *thread*
 - 스케줄링 대상, **processor** 할당 대상
 - 동시에 실행될 수 있는 함수와 관련된 스케줄링 대상
 - 모든 프로세스에는 항상 **main** 함수에서 시작하는 하나의 **main thread**가 존재

- ❑ Resource ownership is referred to as a *process*
 - 기존의 프로세스는 자원의 소유주 역할을 담당
 - 열린 파일, 소켓, 또는 할당된 메모리의 주인은 누구?

- ❑ 과거 **thread**가 제공되지 않았을 때는 **process**가 위 두 가지 역할을 동시에 수행

스레드의 생성

```
#include <pthread.h>
int pthread_create(pthread_t *tidp, const pthread_attr_t *attr,
                  void *(*start_rtn)(void *), void *arg);
```

Returns: 0 if OK, error number on error

- ❑ *tidp*: 스레드 id 변수의 포인터
- ❑ *arg*: 스레드 시작 함수에 넘겨 줄 매개변수
- ❑ *void* (*start_rtn)(void* targ)*: 스레드의 시작 함수
 - *targ*: 스레드 생성시 넘겨 주는 매개변수 값(일반적으로 타입을 변형해서 사용함)
 - 보통은 NULL을 반환; 필요시 값을 넘겨 줄 수 있음

스레드의 생성 예제

```
int pthread_create(pthread_t *tidp, const pthread_attr_t *attr,  
                  void *(*start_rtn)(void *), void *arg);
```

```
void* ThreadName(void *arg) {  
    // 함수 이름은 사용자가 임의로 작명해도 되지만  
    // 매개변수는 반드시 void * 여야 한다.  
    /* 여기서 스레드가 수행해야 할 일을 반복 수행한다. */  
    return NULL;  
}  
  
int main() {  
    pthread_t tid; int ret;  
    ret = pthread_create(&tid, NULL, ThreadName, NULL);  
    if (ret != 0)  
        에러 발생;  
    // 지금부터 ThreadName 스레드와 main 스레드가 동시에 수행된다.  
    // tid 스레드가 종료할 때까지 여기서 대기  
    pthread_join(tid, NULL);  
}
```

스레드의 종료

```
#include <pthread.h>
int pthread_exit(void *rval_ptr);
int pthread_join(pthread_t thread_id, void **rval_ptr);
```

- ❑ *pthread_exit()*: 스레드 실행 도중 함수 중간에서 종료하고자 할 경우 사용
 - 스레드에서 넘겨 줄 값이 없을 경우 `pthread_exit(NULL);`
- ❑ *pthread_join()*: *tid* 스레드가 종료될 때까지 이 함수를 호출한 스레드는 대기함(실행 정지함)
 - *tid* 스레드로부터 넘겨 받을 값이 없을 경우 두번째 인자를 NULL로 지정
 - *main()* 스레드가 종료하면 프로그램 전체가 종료하므로 *main()* 은 항상 종료 직전에 이 함수를 사용하여 다른 스레드가 모두 종료할 때까지 대기해야 함
 - 종료하는 각 스레드마다 한번씩 호출해야 하며 필요 없는 경우 바로 종료해도 됨

스레드의 종료 예제

```
int pthread_exit(void *rval_ptr);  
int pthread_join(pthread_t thread_id, void **rval_ptr);
```

```
void* ThreadStart(void *arg) {  
    int value = (int *)arg;  
    /* do something */  
    /* pthread_exit((void*)value); 중간에 스레드가 종료하고자 할 때 */  
    ++value;  
    return (void*)value; /* return NULL; 리턴할 값이 없을 경우 */  
}  
  
int main() {  
    int ret, value; pthread_t tid; void *vp;  
    ret = pthread_create(&tid, NULL, ThreadStart, (void*)value);  
    // tid 스레드가 종료할 때까지 여기서 대기  
    pthread_join(tid, &vp); // tid 스레드로부터 넘겨 받을 값이 있을 경우  
    /* pthread_join(tid, NULL); tid 스레드로부터 넘겨 받을 값이 없을 경우 */  
    value = (int)vp;  
}
```

cmdc_thread.c

- 아래 주석 처리된 헤더파일들은 코드에서 삭제할 것
 - 여기서는 필요 없는 헤더 파일임

```
//#include <sys/types.h>           // mkfifo()  
//#include <sys/stat.h>             // mkfifo()  
  
//#include <sys/wait.h>             // wait()  
//#include <signal.h>               // signal()
```

cmdc_thread.c: 아래 코드를 삽입할 것

```
void thread_err_exit(int err, char *msg)
{ printf("%s: %s\n", msg, strerror(err)); exit(1); }

void dual_threads(void)
{
    int ret;
    pthread_t tid1, tid2;

    if ((ret = pthread_create(&tid1, NULL, InputSendThread, NULL)) != 0)
        thread_err_exit(ret, "pthread_create");
    if ((ret = pthread_create(&tid2, NULL, RecvOutputThread, NULL)) != 0)
        thread_err_exit(ret, "pthread_create");
    pthread_join(tid2, NULL);
}

int main(int argc, char *argv[]) {
    connect_to_server();
    // dual_process(); // 두 개의 프로세스 기반 클라이언트
    dual_threads();    // 두 개의 스레드 기반 클라이언트
    dis_connect();
}
```

cmdc_thread.c

```
// 기존의 dual_process(void) 함수 전체를 삭제
void dual_process(){
```

// 위 삭제된 자리에 아래 코드를 삽입할 것

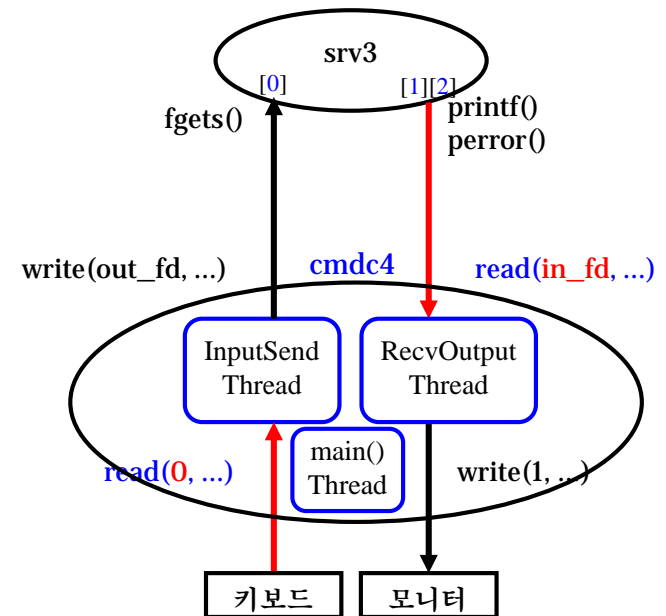
```
#include <pthread.h>
```

// 키보드 전담 스레드의 시작 함수

```
void *InputSendThread(void *arg)
{
    // 키보드에서 받아 -> 서버로 전송
    input_send_loop();
    return(NULL);
}
```

// 서버 전담 스레드의 시작 함수

```
void *RecvOutputThread(void *arg)
{
    // 서버에서 받아 -> 화면에 디스플레이
    recv_output_loop();
    return(NULL);
}
```



cmdc_thread 내에 세 개의 스레드가 동시에 실행되고 있는 상황 (전송할 데이터가 없을 경우 정지되어 있음)

아래 함수는 기존과 동일함

// 반복하여 키보드에서 입력 받아 서버로 전송

```
void input_send_loop(void)
```

```
{
    while (1) {
        if (input_send() <= 0) // 키보드 입력 후
                               // 서버로 전송
            break;
    }
}
```

// 반복하여 서버에서 받은 데이터를

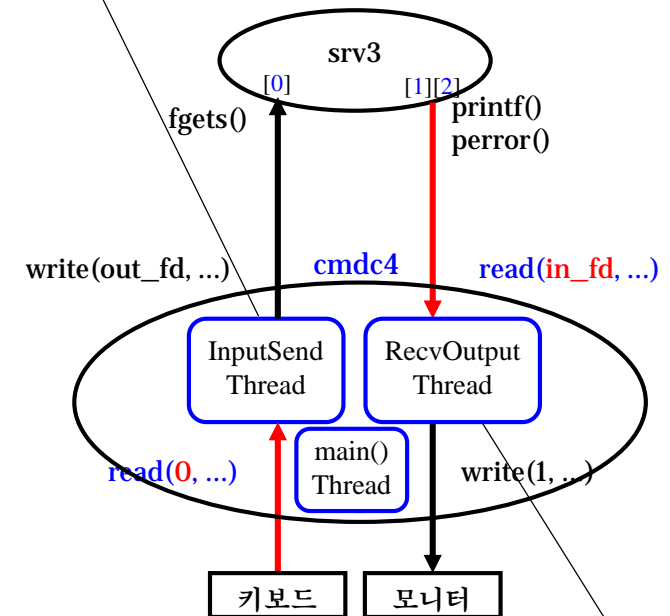
// 모니터(화면)로 출력

```
void recv_output_loop(void)
```

```
{
    while (1) { // 서버로부터 받고 화면에 출력
        if (recv_output() <= 0)
            break;
    }
}
```

기존 input_send() 함수

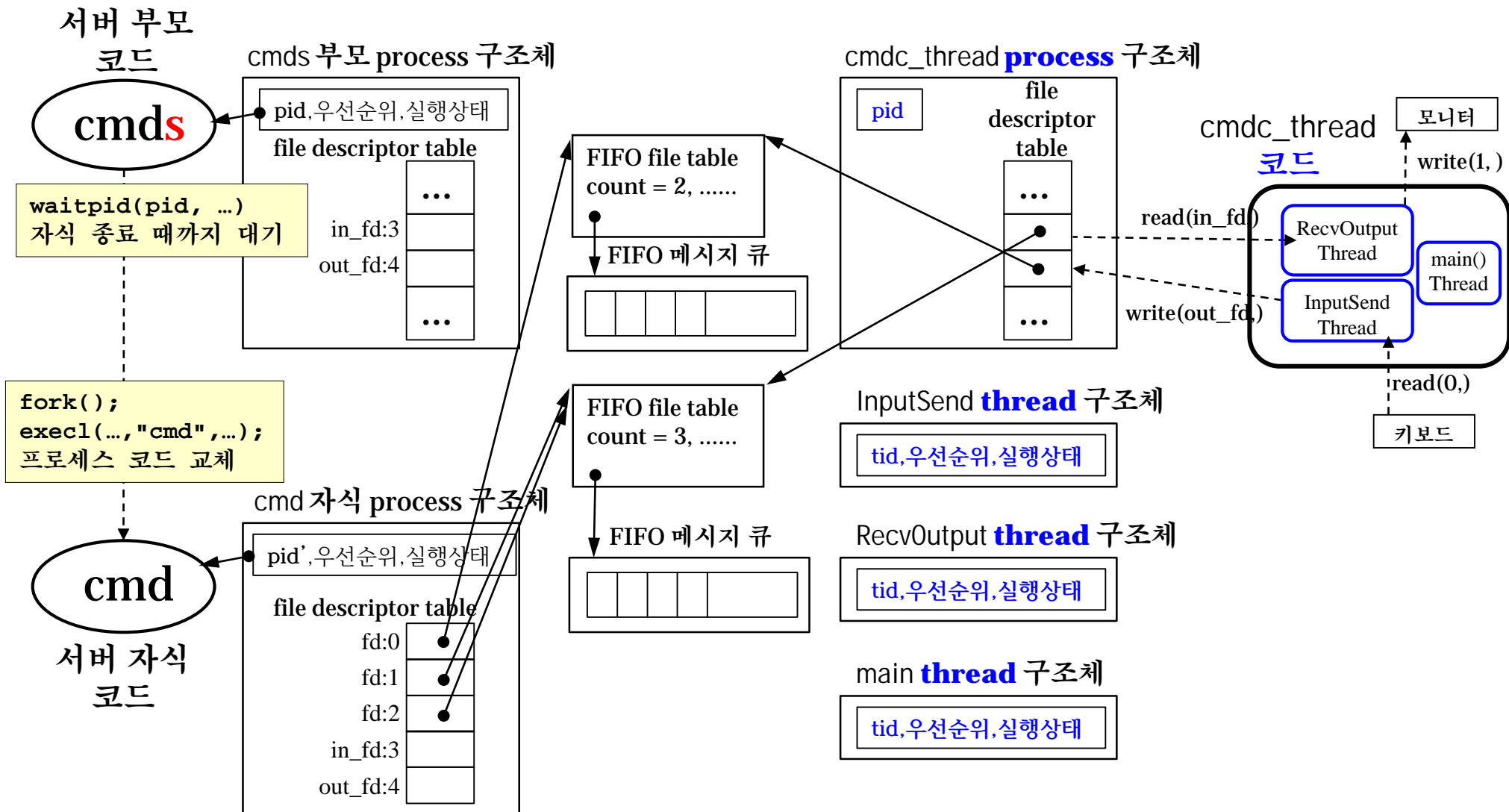
```
len = read(0, cmd_line, SZ_STR_BUF);
if (len <= 0) return len;
if (write(out_fd, cmd_line, len) != len)
    return -1;
return len;
```



기존 recv_output() 함수

```
len = read(in_fd, cmd_line, SZ_STR_BUF);
if (len <= 0) return len;
if (write(1, cmd_line, len) != len)
    return -1;
return len;
```

cmdc_thread 실행 후의 커널 모습



cmdc_thread 클라이언트 프로그램 실행

□ make를 실행하여 cmdc_thread를 생성함

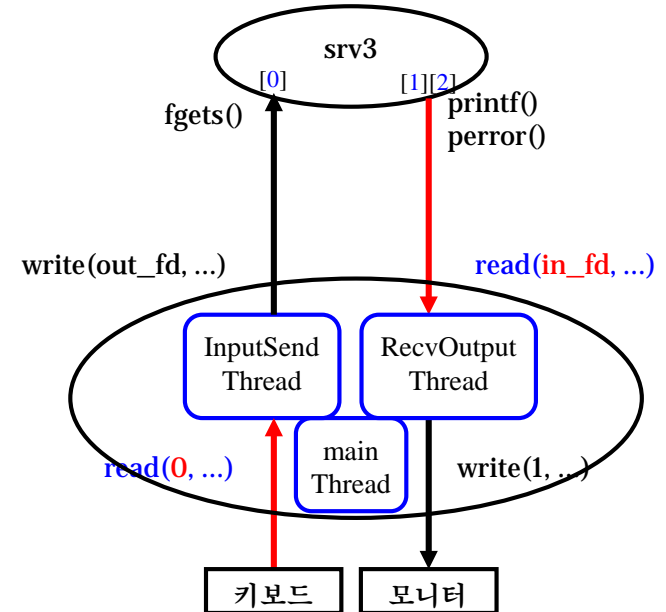
□ 서버 프로그램으로 srv3 (에코서버) 실행
\$ srv3

□ srv3 대신에

- cmds(명령어 해석기 서버, cmd의 서버 버전)를 사용해도 됨
- 또는 cmdsd (cmds의 데몬 버전) 사용 가능

□ 다른 터미널 창에서 cmdc_thread를 실행

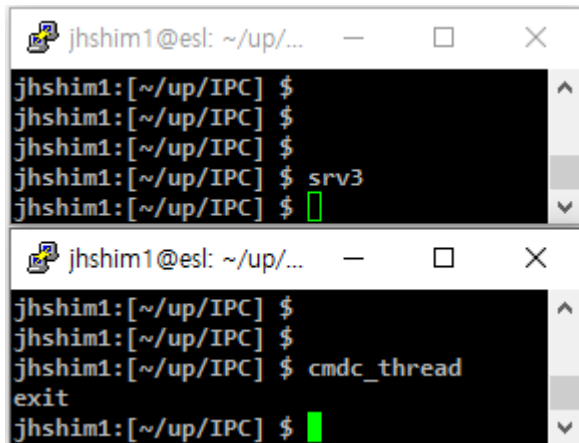
- 두 스레드는 각각 read() 함수에서 데이터가 들어오길 대기하고 있다가 데이터가 들어오면 이를 서버 또는 모니터에 write() 함
- 두 스레드는 각자 따로 실행됨



서버와 클라이언트의 정상 종료

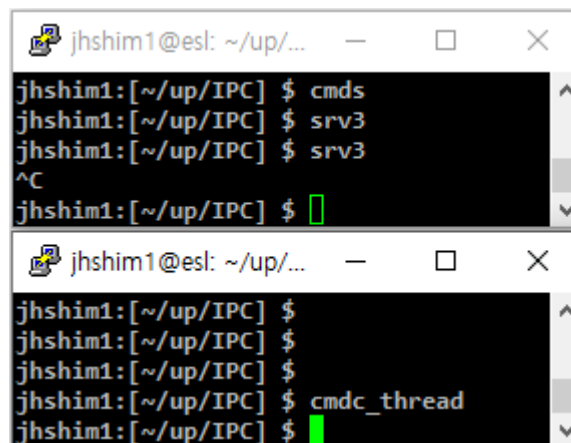
❑ 서버로 srv3가 실행되고 있다고 가정 (cmds, cmdsd 사용가능)

1. cmdc_thread에서
“exit[enter]” 입력
○ 모두 종료



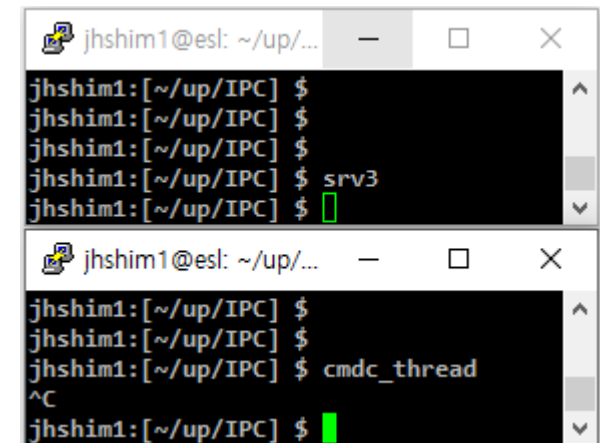
```
jhshim1@esl: ~/up/...  
jhshim1: [~/up/IPC] $  
jhshim1: [~/up/IPC] $  
jhshim1: [~/up/IPC] $  
jhshim1: [~/up/IPC] $ srv3  
jhshim1: [~/up/IPC] $  
  
jhshim1@esl: ~/up/...  
jhshim1: [~/up/IPC] $  
jhshim1: [~/up/IPC] $  
jhshim1: [~/up/IPC] $ cmdc_thread  
exit  
jhshim1: [~/up/IPC] $
```

2. srv3에서 Ctrl+C 입력
○ 모두 종료



```
jhshim1@esl: ~/up/...  
jhshim1: [~/up/IPC] $ cmds  
jhshim1: [~/up/IPC] $ srv3  
jhshim1: [~/up/IPC] $ srv3  
^C  
jhshim1: [~/up/IPC] $  
  
jhshim1@esl: ~/up/...  
jhshim1: [~/up/IPC] $  
jhshim1: [~/up/IPC] $  
jhshim1: [~/up/IPC] $  
jhshim1: [~/up/IPC] $ cmdc_thread  
jhshim1: [~/up/IPC] $
```

3. cmdc_thread에서
Ctrl+C 입력
○ 모두 종료



```
jhshim1@esl: ~/up/...  
jhshim1: [~/up/IPC] $  
jhshim1: [~/up/IPC] $  
jhshim1: [~/up/IPC] $  
jhshim1: [~/up/IPC] $ srv3  
jhshim1: [~/up/IPC] $  
  
jhshim1@esl: ~/up/...  
jhshim1: [~/up/IPC] $  
jhshim1: [~/up/IPC] $  
jhshim1: [~/up/IPC] $ cmdc_thread  
^C  
jhshim1: [~/up/IPC] $
```

❑ cmdsd를 서버로 사용하였다면 로그 아웃하기 전에 반드시 cmdsd를 종료해야 함 (\$ kill -9 pid)