41865 Unix Systems

실습 14
데몬 프로세스 (Deamon process)
cmdsd

Jaehong Shim

Dept. of Computer Engineering

조선대학교
CHOSUN UNIVERSITY

목적

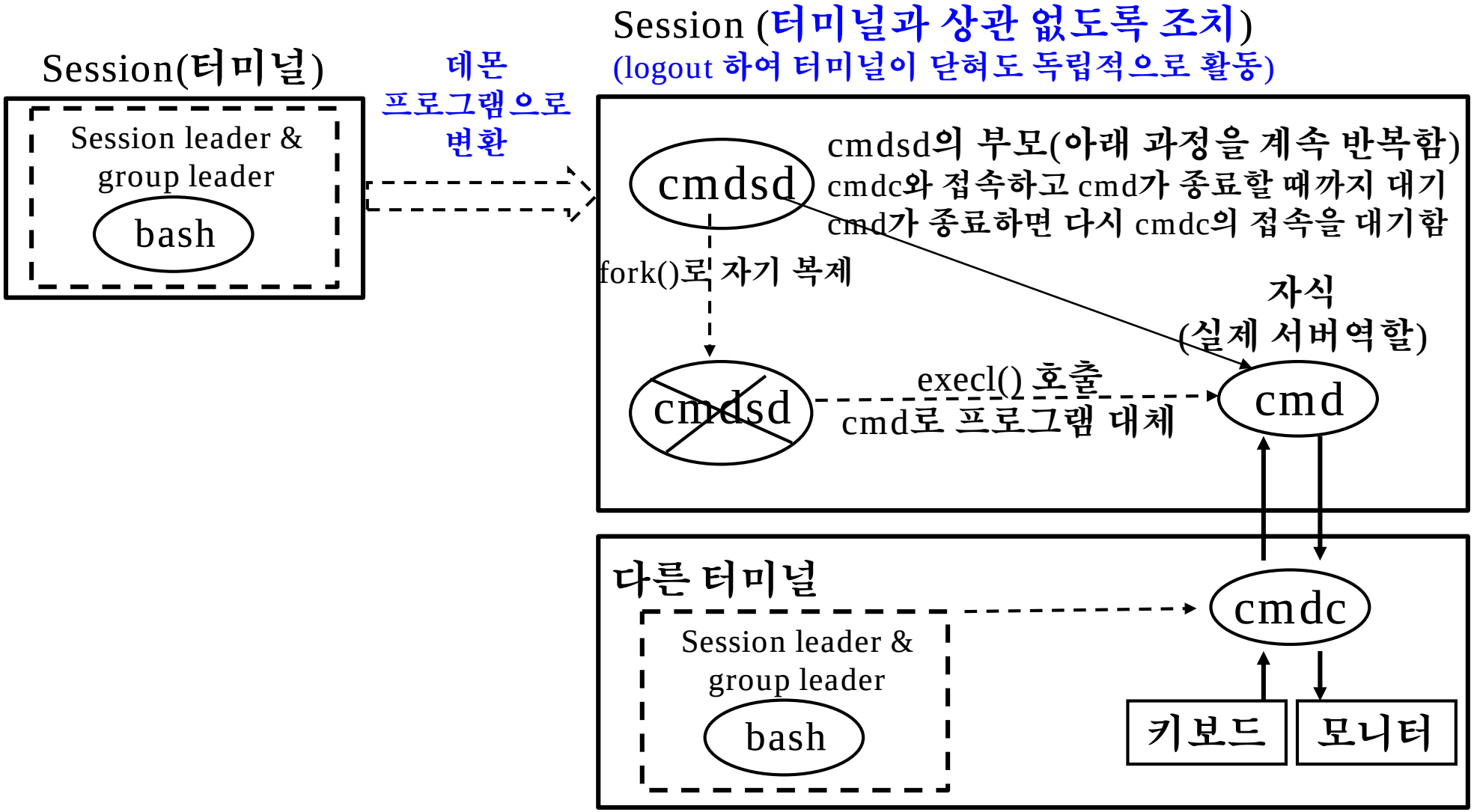
- 기존의 서버인 cmds를 데몬 프로세스로 만든다.
- 데몬(daemon) 프로세스란?
  - 사용자가 ssh 창에서 logout하여 완전히 시스템을 빠져 나가도 계속 백그라운드로 실행되면서 서비스를 해 주는 서버 프로그램
  - 일반적인 대부분의 서버 프로그램(Web 및 DB 서버)은 데몬 프로세스임
  - 데몬 프로세스를 처음 실행시키면 프로세스가 데몬으로 변하면서 바로 다음 명령 프롬프트가 나오고 계속해서 다른 명령어를 입력할 수 있다.

```
$ cmdsd
$
```

  - 이때 데몬 프로세스는 백그라운드로 시스템 내에서 계속 실행되고 있다.

```
$ ps -f -u jhshim(자신의 계정이름)
  // 자신이 실행한 모든 프로세스 리스트를 보여줌
UID     PID   PPID   C STIME    TTY       TIME CMD
jhshim 24695 11174  0 15:17:58 pts/27   0:00 ps -f -u jhshim
jhshim 16277 16275  0 11:13:47 pts/7    0:00 bash
jhshim 24653     1  0 15:11:55 ?        0:00 cmdsd
```

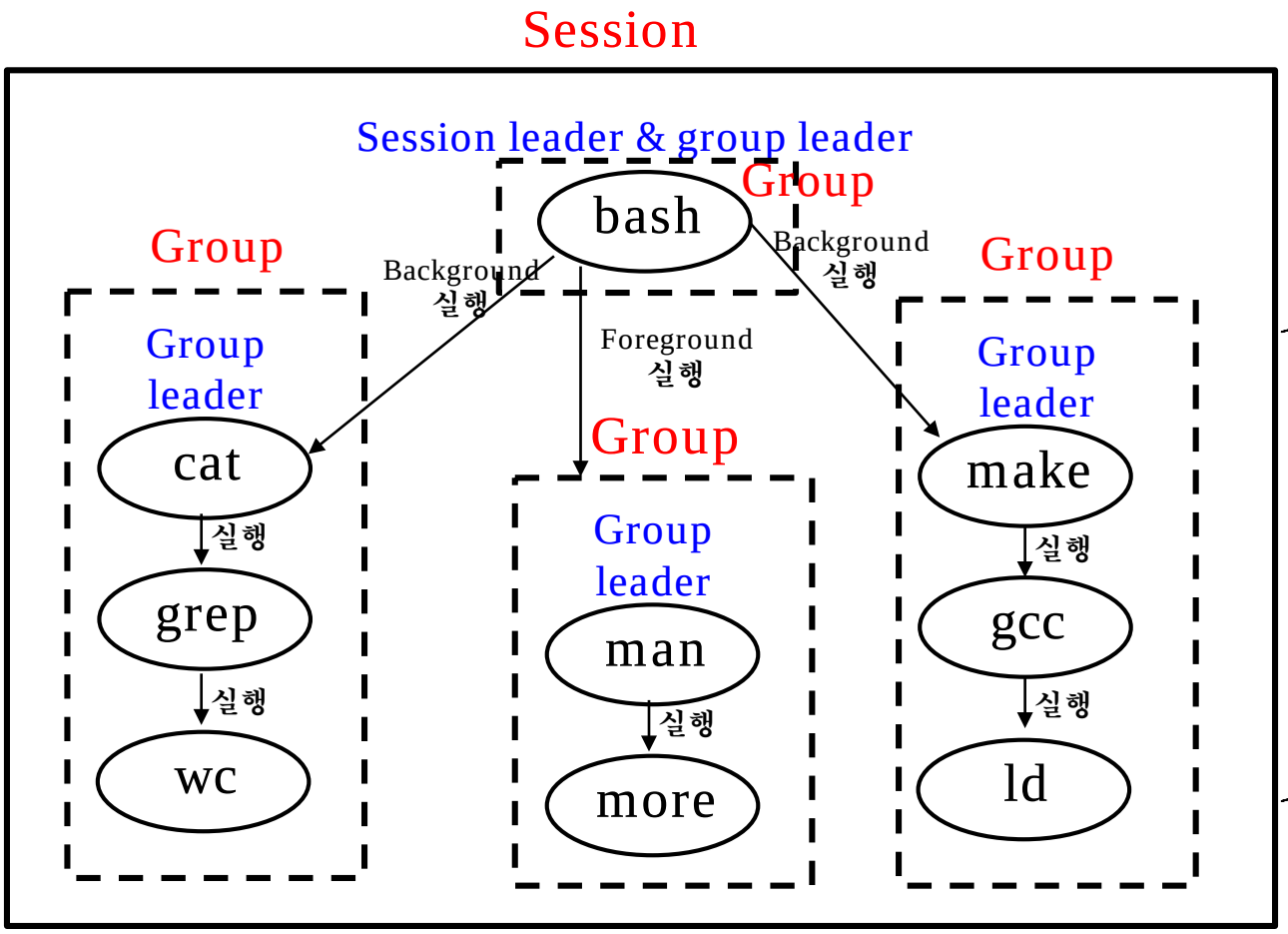
cmdsd의 전반적인 실행 과정



Session and Group

이렇게 &로 실행하는 것은 백그라운드로 실행하는 것임

```
$ cat f1 | grep “abc” | wc &
$ make &
$ man
```

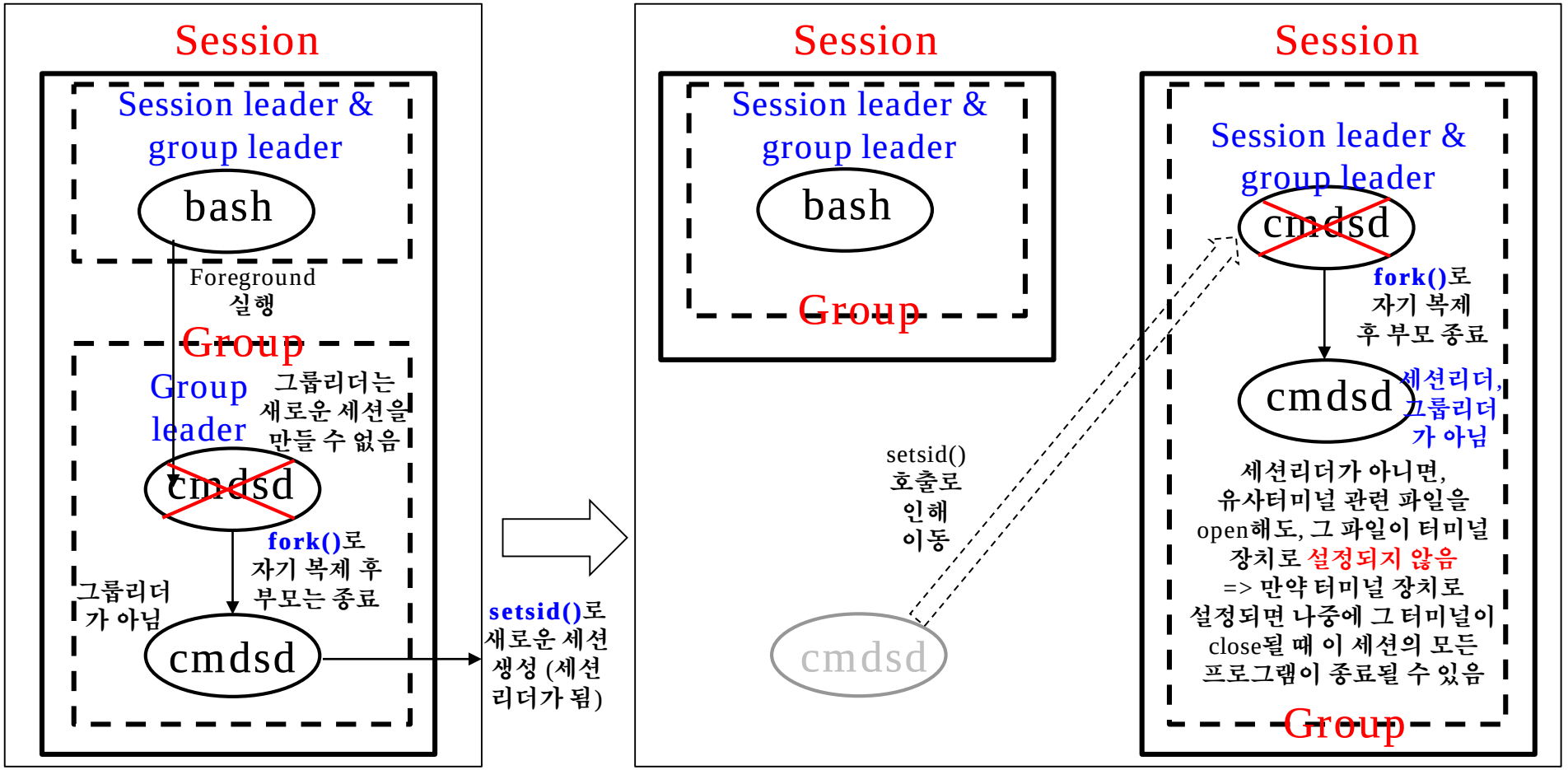


session: 동일한 단말기(표준입출력 장치)를 공유하는 프로세스들의 집합

데몬으로 실행되는 것이 아니므로 logout하면 모든 프로세스가 종료됨

cmdsd의 데몬화 과정

```
$ cmdsd
$
```



cmdsd.c 파일 작성

- ~/up/IPC 밑에서 기존 cmds.c 파일을 cmdsd.c 파일 복사한다.

```
$ cd  ~/up/IPC
$ cp cmds.c  cmdsd.c
$ ls
  cmds.c  cmdsd.c  Makefile   기타 기존 파일들
```

- Makefile의 TARGETS에 cmdsd를 추가

```
TARGETS = … srv2 srv3 cmdc cmds cmdsd
```

cmdsd.c의 main() 수정

// 기존 main()을 아래와 같이 수정함 (파란색과 빨간색 부분 수정)

```
int main(int argc, char *argv[])
{
   printf("cmdsd: Daemon is Ready.\n");

   daemonize(); // 이 프로세스를 데몬 프로세스로 만듦

   while (1) {
        … // 기존 코드
        else if (pid == 0) { // 자식 프로세스
            printf("client connected: cmd(pid %d) exec\n", getpid());
            duplicate_IO();
            execl("../cmd/cmd", "cmd", NULL);
        }
         else { // pid > 0 경우, 즉 부모 프로세스
             dis_connect();
             waitpid(pid, NULL, 0); // 자식(pid)이 종료될 때까지 대기
             printf("disconnected: cmd(pid %d) terminated\n", pid);
             sleep(2); // 클라이언트 cmdc가 종료되길 잠시 기다림
         }
}  }
```

cmdsd.c의 daemonize() 추가 - 1

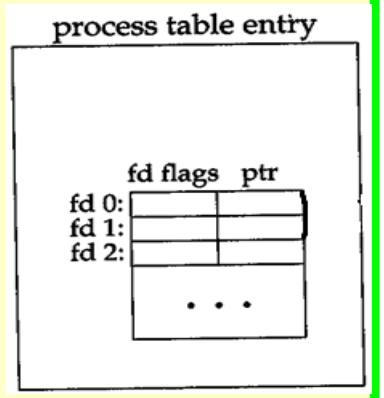
```
void daemonize(void)   // 이 함수를 main() 함수 앞에 새로 추가한다.
{
   int i;
   pid_t pid;
   struct rlimit rl;

   // 파일 접근 모드 매스크를 리셋함. 이 함수를 호출하지 않으면 향후 데몬이
   // 생성할 파일의 접근모드가 creat()에서 저정하는데로 안될 수 있다.
   umask(0);
   // 아래의 setsid()를 호출하는 프로세스는 그룹 리더가 아니어야 한다.
   // 부모가 혹시 그룹리더일 수 있으므로 부모는 죽고 자식이 계속 진행하게 함
   // 부모가 죽었으므로 자식은 절대 그룹리더가 아님
   if ((pid = fork()) < 0)
         print_err_exit("fork");
   else if (pid > 0)   // 부모는 여기서 종료, 자식은 계속 실행
        exit(0);
   // 1) 스스로 세션 리더가 된다.
   // 2) 터미널(키보드와 모니터) 장치와 연결을 해제하고,
   // 3) 세션 내의 그룹의 그룹리더가 됨
   // 지금부터는 키보드 입력과 화면 출력은 안됨
   setsid();
   // 다음 페이지에 계속 ...
```

cmdsd.c의 daemonize() 추가 - 2

```
// 터미널(전화선 모뎀)과 연결이 끊어질 때 전달되는 시그널을 무시함
// 사실 터미널이 없기 때문에 이 신호는 들어 오지 않지만,
// 그래도 혹시 들어오면 무시하게 함
signal(SIGHUP, SIG_IGN);

// 향후 본의 아니게 유사(pseudo)터미널 관련 파일(/dev/pts/*)을 open할 경우,
// 데몬이 세션리더인 경우, 그 파일이 데몬의 터미널로 설정됨
// 터미널이 닫히면 그 터미널의 모든 프로그램이 죽게 되므로(그러면 안됨)
// 따라서 세션리더인 부모는 죽고 자식이 계속 진행함
// 세션리더인 부모가 죽었기 때문에 자식은 절대 세션리더가 아님
if ((pid = fork()) < 0)
     print_err_exit("fork");
else if (pid > 0)   // 부모는 여기서 종료, 자식은 계속 실행
     exit(0);

// 지금의 현재 작업 디렉토리가 있는 하드 디스크가 관리자에 의해 시스템에서
// 동적으로 강제 제거 될 수도 있으므로(고장 등의 이유로) 현재 작업
// 디렉토리를 /로 변경함 (/는 시스템이 종료되기 전까지는 항상 존재하므로)

//   chdir("/"); // 우리는 현재 디렉토리에서 계속 작업하기로 함
```

cmdsd.c의 daemonize() 추가 - 3

```
    // 한 프로세스가 open할 수 있는 최대 파일의 수를 얻음
    if (getrlimit(RLIMIT_NOFILE, &rl) < 0)
          print_err_exit("getrlimit: can't get file limit");

    if (rl.rlim_max == RLIM_INFINITY)
         rl.rlim_max = 1024;

    // 기존에 open한 파일이 터미널 관련 파일이 있을 수 있으므로
    // 열려 있는 모든 파일을 닫는다.
    // 따라서 필요한 파일은 데몬으로 전환한 이후에 open하여야 한다.
    for (i = 0; i < rl.rlim_max; ++i)
         close(i);
}
```



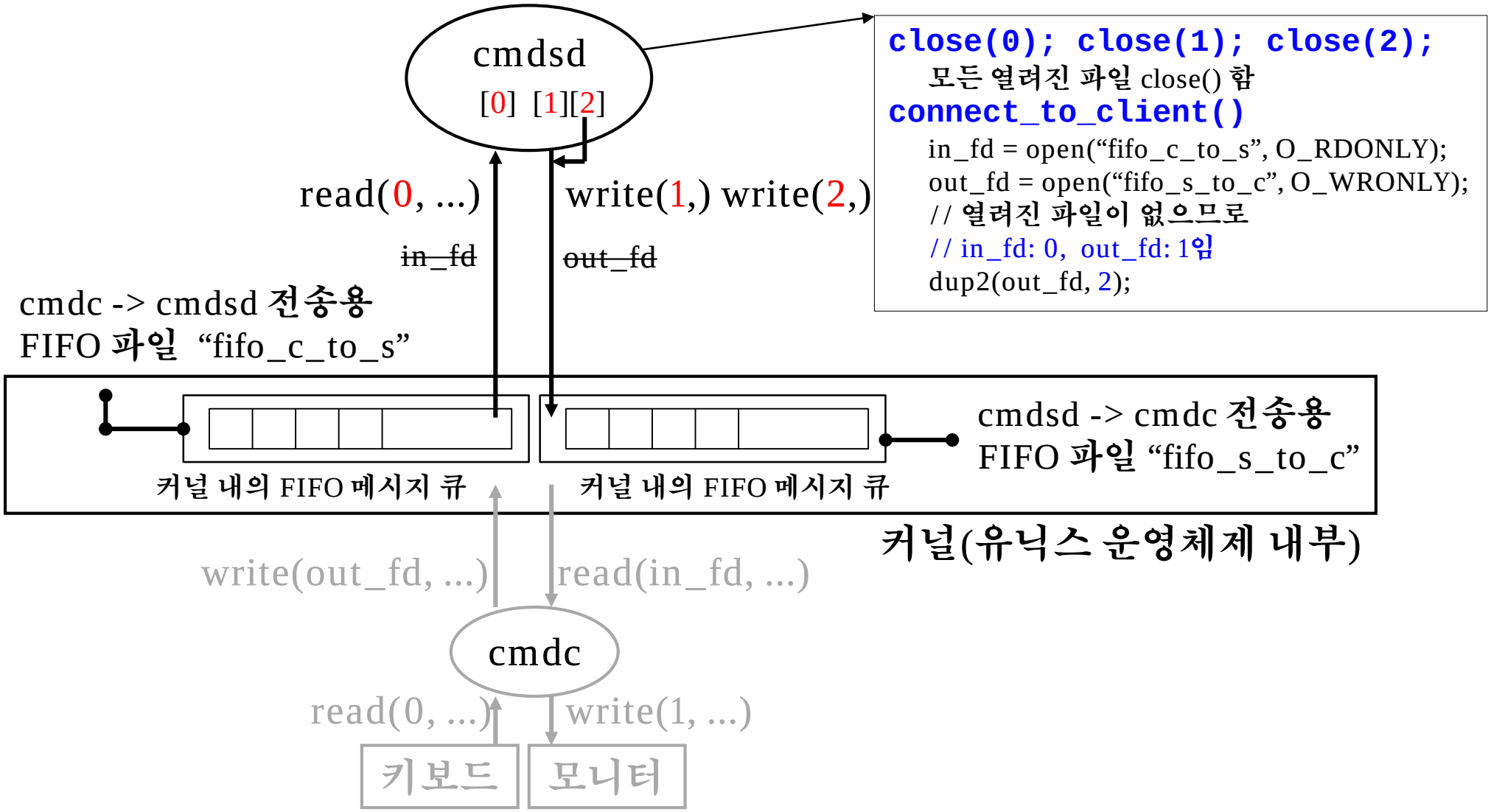
- man 명령어를 이용하여 signal(), umask(), setsid(), getrlimit() 함수들의 선언이 포함되어 있는 헤드 파일을 찾아서 include 한다.

duplicate_IO() 수정

- 아래 진한 글씨 부분을 모두 주석 처리하라.

```
void duplicate_IO()
{
   // daemonize()에 의해 모든 파일 핸들들이 close() 되었기 때문에
   // connect_to_client()할 때 in_fd는 0, out_fd는 1로 자동 설정된다.
   // 따라서 0, 1에 대해 dup2()를 호출할 필요가 없다.
   //dup2(in_fd, 0);   // 0: STDIN_FILENO,  클라이언트 -> 서버
   //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);
   setbuf(stdout, NULL);

   // in_fd, out_fd 자체가 각각 파일 핸들 0과 1 이기 때문에
   // 이들을 close()하면 안된다.
   //dis_connect();
}
```

cmdsd의 역할: cmdc와 접속 및 표준입출력을 메시지 큐로 교체




```
close(0); close(1); close(2);
    모든 열려진 파일 close() 함
connect_to_client()
    in_fd = open("fifo_c_to_s", O_RDONLY);
    out_fd = open("fifo_s_to_c", O_WRONLY);
    // 열려진 파일이 없으므로
    // in_fd: 0,  out_fd: 1임
    dup2(out_fd, 2);
```

cmdsd 데몬 프로그램 실행

- make를 실행하여 cmdsd를 생성함
- 한쪽 터미널 창에서 cmdsd를 먼저 실행시키면, cmdsd는 데몬 서버 프로세스가 되면서, 바로 다음 명령 프롬프트가 나온다.

```
$ cmdsd
$
```

- cmdsd가 살아서 실행되고 있는지 확인하는 방법

```
$ ps -f -u jhshim(자신의 계정이름)
  // 다른 터미널 창에서 실행한 것도 포함해서
  // 자신이 실행한 모든 프로세스 리스트를 보여줌
UID     PID   PPID   C STIME    TTY       TIME CMD
jhshim 24695 11174   0 15:17:58 pts/27    0:00 ps -f -u jhshim
jhshim 16277 16275   0 11:13:47 pts/7     0:00 bash
jhshim 24653     1   0 15:11:55 ?         0:00 cmdsd
```

  - 위처럼 cmdsd가 존재해야 정상임
  - 만약 cmdsd가 여러 개 있다면 앞 전에 실행시킨 cmdsd가 죽지 않은 것임

cmdc 클라이언트 프로그램 실행

- 다른 터미널 창 또는 동일한 터미널 창에서 cmdc를 실행
- 아래처럼 출력되어야 함

```
$ cmdc
</home/……/up/IPC> 1:
```

  - 이후 여기에서 오른쪽 명령어들을 입력하면
  - 결과가 여기에 출력됨
- exit 명령어 입력하면 cmdc가 종료하지만,
  cmdsd 데몬은 여전히 살아 있다.
  - cmdsd가 여전히 살아 있는지 확인하는 방법

```
$ ps -f -u jhshim(자신의 계정이름)
```

- 모든 터미널 창에서 logout하고,
  다시 로그인 해도 여전히 cmdsd가 살아 있는지 확인한다.
- 다시 cmdc를 실행시켜 본다.
  - 정상적으로 cmdsd와 접속해서 실행되어야 함

```
$ cat cmdc.c
$ cd ../cmd
$ ls -l
$ cd ../IPC
$ uname  -a
$ pwd
$ date
$ whoami
$ cp  cmdc.c  test.c
$ mv test.c tmp.c
$ ls
$ rm tmp.c
$ id
$ cal
$ cal 2016
$ ps
$ who
$ clear
$ whereis gcc
$ cd ../pr4
$ make clean
$ make
$ cd ../IPC
```

cmdsd 데몬 프로그램 종료

- cmdsd는 데몬 서버 프로그램이기 때문에 로그 아웃을 해도 살아 있게 됨
- 따라서 실습을 끝내고 로그 아웃하기 전에 반드시 cmdsd를 종료해야 함
- Kill 명령어를 사용하여 cmdsd를 강제 종료시킴

```
$ ps –f –u jhshim(자신의 계정이름)
  // 자신이 실행한 모든 프로세스 리스트를 보여줌
 UID    PID   PPID   C STIME    TTY       TIME CMD
 jhshim 24695 11174  0 15:17:58 pts/27   0:00 ps –e –u jhshim
 jhshim 16277 16275  0 11:13:47 pts/7    0:00 bash
 jhshim 24653     1  0 15:11:55 ?        0:00 cmdsd

$ kill -9 24653 (24653은 cmdsd의 process ID, 즉 PID 값)
$ ps –f –u jhshim(자신의 계정이름) // cmdsd가 없어야 정상임
```