1. 외부 명령어 실행할 수 있도록 cmd.c를 수정하라.
   1) 0-6-3-실습10-강의노트를 보고 구현하라.
   2) fork(), execvp(), waitpid() 함수의 헤드 파일을 찾아 include시킨다. 추가할 파일이 있음. 반드시 추가할 것.
   3) make하라. 그런 후 cmd를 실행시키고 명령어로 다음을 테스트하라. "**>** "는 cmd의 명령 프롬프트다.

```
> cal
> cal 2016
> ps
> who
> clear
> whereis gcc
> cd ../pr4
> make clean
> make
> cd ../cmd
> man cd
> more cmd.c
> vi cmd.c          // 확인하고 바로 나올 것
> touch cmd.c
> make // make하면 현재 실행 중인 cmd가 새로 만들어지므로 현재 실행 중인 cmd가 죽을 수도 있음
```

2. sleep 명령어를 구현하라.
   1) 먼저 Sleep() 함수를 정의하자.

```
--------------------------------------
// 프로그램을 원하는 시간 동안(초 단위) 일시적으로 중지하는 명령어
// 사용법: sleep 초단위시간
void Sleep(void) {} // 기존 함수 형태로 할 것
// 함수 이름이 대문자로 시작함에 유의할 것
--------------------------------------
```

   2) 교재 p.459를 참조하여 Sleep() 함수를 구현하자.
      i) 사용자가 명령어 첫 인자로 준 초 단위의 시간은 argv[0]가 포인터하는 문자열이다. (예, "5"초). 그런데 나중에 우리는 API 함수 sleep()를 호출해야 하고, 이 함수의 인자로 초 단위의 정수 값을 주어야 한다. 따라서 argv[0]의 문자열로 된 숫자를 정수형 값으로 변환해야 한다. 아래 코드를 삽입하라.

```
--------------------------------------
int sec;
sscanf(argv[0], "%d", &sec);
--------------------------------------
```

      이 함수는 argv[0]가 포인터하는 문자열("5")을 읽어 이를 정수형 값으로 변환하여 sec에 저장한다. (키보드가 아닌 argv[0]가 포인터하는 문자열에서 읽어 들임)
      ii) sleep() API 함수를 호출하여 프로그램의 실행을 sec 초 동안 일시 정지하게 한다.
   3) cmd_tbl[] 배열에 Sleep 관련 배열 원소를 추가하라. 위 1) 번의 주석문 중 사용법을 참조하기 바란다.
   4) sleep() 함수의 헤드 파일을 찾아 include시킨다. man 명령어 입력시 –s3 옵션을 주어야 한다.
   5) make하라. cmd를 실행시키고 명령어로 다음을 테스트하라. "**>** "는 cmd의 명령 프롬프트다.

```
> sleep 1
> sleep 5
> sleep 10
```

3. 지금까지의 구현에서 에러가 발생했을 때는 항상 PRINT_ERR_RET()를 호출하여 에러 원인을 출력하고 바로 리턴 했다. 이제 이것을 setjmp()와 longjmp()을 이용하여 교체해 보자.
   1) 먼저 기존의 cmd.c를 **cmdjmp.c**로 복사하라.
      i) 그런 후 Makefile에 맨 앞쪽에 다음을 추가하라.

```
--------------------------------------
TARGETS = cmd cmdjmp

all : $(TARGETS)

cmdjmp: cmdjmp.c
[탭] gcc -o cmdjmp cmdjmp.c

cmd: cmd.c // 기존 코드 그대로 유지할 것
 // 그리고 기존 clean 룰에서 cmd를 삭제하고
 // 그 자리에 $(TARGETS)을 삽입할 것
--------------------------------------
```

   2) 강의노트 7장 pp.22~27, 교재 pp.265 ~272를 참조하라.
      i) 강의노트를 참조하되 함수 사용하는 방식만 참조하고 변수이름이나 코드는 따라 입력하지 마라. 지금부터는 **cmdjmp.c**를 수정한다.
      ii) 먼저 cmdjmp.c의 전역변수로 다음을 추가하라.

```
--------------------------------------
char cur_work_dir[SZ_STR_BUF]; // 이 변수 다음에 입력할 것
jmp_buf jump;
int  cmd_idx;
--------------------------------------
```

      iii) 다음은 main()에서 다음의 지역변수를 선언하라.

```
int jmpret;
```

```
sleep(5);
int s = 10;
sleep(s);
```

iii) main()의 for() 문장 바로 앞에 다음을 삽입한다.

```
-------------------------------------
setjmp() 함수를 호출하여 리턴 값을 jmpret에 저장;
즉, jmpret = setjmp(jump);
// 이때 함수 인자는 앞서 선언한 전역변수(jump)를 지정할 것.
만약 jmpret가 0이 아니면 { // 위 문장과 하나로 통합할 것
   // longjmp()에 의해 리턴된 경우임
   만약 jmpret가 -1이면
        // PRINT_ERR_RET()에서 longjmp()한 경우
       perror()을 호출함;
        // perror() 함수의 인자는 PRINT_ERR_RET() 참조
   else if jmpret가 -2이면
       // check_arg()와 check_opt()에서 longjmp()한 경우
       print_usage()를 호출함;
       // (proc_cmd()에 있는 이 함수호출을 복사하여 삽입
       // 하되 k 대신에 전역변수 cmd_idx를 사용할 것)
}
-------------------------------------
```

또한 **main()** 함수 내에서 **proc_cmd();**와 **cmd_count++;** 문장의 순서를 서로 바꾸어라. 즉, **cmd_count++;** 가 먼저 나오게 한다.

iv) 다음은 proc_cmd() 함수의 if (EQUAL(cmd, cmd_tbl[k].cmd) 문장의 { } 속을 다음과 같이 수정하라.

```
-------------------------------------
  cmd_idx = k;
  check_arg(cmd_tbl[k].argc); // 명령어 인자 체크
  check_opt(cmd_tbl[k].opt); // 명령어 옵션 체크
  cmd_tbl[k].func();          // 명령어 처리함수 호출
  return;
-------------------------------------
```

위 코드에서 **check_arg()**와 **check_opt()**는 에러가 발생한 경우 **longjmp()**로 **main()**으로 바로 jump하여 명령어 사용법을 출력하기 때문에 여기서는 에러 체크할 필요가 없다. 그리고 이 함수들이 정상적으로 리턴했다는 것은 에러가 발생하지 않았다는 것을 의미하기도 한다.

v) 다음은 **check_arg()** 함수로 이동하여 이 함수의 리턴 데이터 타입을 **int**에서 **void**로 수정하라.
return(0);를 전부 **return;**으로 변경하라.
마지막 **return(-1);**를 **longjmp(jump, -2);**로 대체하라. 즉, 인자 개수가 잘못되었을 경우 long jump한다. 그러면 갑자기 main()으로 되돌아가 setjmp() 함수에서 리턴하게 된다. setjmp() 함수의 리턴 값은 -2가 된다.

vi) 다음은 **check_opt()** 함수로 이동하여 이 함수의 리턴 데이터 타입을 **int**에서 **void**로 수정하라. 그리고 마지막 **return(err);**를 **if (err) longjmp(jump, -2);**로 대체하라. 이 역시 옵션 중 하나라도 잘못되었으면 long jump하여 main()으로 되돌아가게 한다.

vii) **PRINT_ERR_RET()** 매크로 함수 전체를 삭제하라. 즉, **#define PRINT_ERR_RET()** 문장의 마지막 **} while(0)**까지 삭제하라.

viii) cmdjmp.c에서 **PRINT_ERR_RET();**를 호출하는 모든 곳을 찾아 **longjmp(jump, -1);**로 변경하라. 그러면 갑자기 main()으로 되돌아가 setjmp() 함수에서 리턴하게 된다. setjmp() 함수의 리턴 값은 -1이 된다.

3) setjmp() 함수의 헤드 파일을 찾아 include시킨다. 추가할 파일이 있음. 반드시 추가할 것.

4) make하라. 그런 후 **cmdjmp**를 실행시키고 명령어로 다음을 테스트하라. "**>** "는 **cmdjmp**의 명령 프롬프트다.

```
> ls a b c
> ls -a -l
// 위 모든 명령어가 정상적으로 에러 출력을 하고
// 명령어 사용법도 출력해야 정상임
// 명령어 번호도 정상적으로 증가해야 함
> cp tttt  t1
> cat /home/jhshim/UP(대문자임)/cmd/cmd.c
// 위 모든 명령어가 정상적으로 에러 출력을 해야 함
// 명령어 번호도 정상적으로 증가해야 함
```

4. 명령 창에서 다음을 실행하여 정상임을 확인하라.

```
$ eshw 10
$ progtest 10
```