과제

문제

```
과제: LED 디바이스 드라이버에서 User 시그널을 발생시켜 이 시그널을
      LED와 연동되고 있는 응용계층의 프로세스의 ID를 파악한 후,
      시그널을 전달하면 미리 등록되어 있던
      시그널 핸들러를 응용계층에서
      호출하는 프로그램을 작성하시오.

*** 기능정리 ***
*응용영역

1) 프로세스 ID를 확인하기 위한 변수선언 ??
2) 프로세스 ID 확인 (무슨 함수???)
3) 응용영역에서 프로세스 ID를 디바이스 드라이버로 전달하는 기능
   write(파일디스크립터, &id, 4) 함수 사용
4) usersignal 등록 및 핸들러 등록
5) 무한루프 or 쓰레드....

*드라이버영역
1) 커널타이머 핸들러에서 주기적으로 시그널 발생
    (커널 영역에서 --> 응용계층으로)
2) 시그널발생시 프로세스 ID를 응용영역에서 전달받아 해당 프로세스에
   User 시그널 발생
   (ex my_kill_proc)
   my_kill_proc --> 이 함수를 쓰세요..!!!
```

문제해설

이런 식으로 동작하는 거다.

그리고 테스크.? 오버? 지연처리를 할 수 있다. 그런 것들을 커널에 넣으면 지연처리를 할 수 있다. 함수 자체가 커널에 등록되서, 커널 쓰레드에 등록한 거다? 쓰레드 처럼 돈다고 보면 된다. 커널 쓰레드처럼 나중에 지연처리 메카니즘 만들어 놓은거다.

> 메인과제 설명

커널 프로그램 영역과 어플리케이션 영역은 유기적으로 코웍으로 돌아야 하는 것, 그걸 이해하기 위한 프로젝트다. 그게 이해가 되면 (SP 시스템 프로그래밍)때 배운 것들을 다 응용할 수 있다. 쓰레드 만들어서 디바이스 드라이버랑 DD와 통신할 수도 있는 거고 커널타이머로 계속 돌면서 어떤 일을 처리할 수도 있는 거고, 커널타이머에서 밑에 층에서 계속 데이터를 올려줄 수도 있는 거고, 현업에선 다 이렇게 쓴다.

*응용영역 / *드라이버영역 나눠있다. 확인해라.

- 위에가 응용 / 아래가 드라이버.

```
1) 프로세스 ID를 확인하기 위한 변수선언 ??
//pid_t pid, int pid도 가능
2) 프로세스 ID 확인 (무슨 함수???)
//getpid
3) 응용영역에서 프로세스 ID를 디바이스 드라이버로 전달하는 기능
   write(파일디스크립터, &id, 4) 함수 사용
//get_user
```

```
3) 응용영역에서 프로세스 ID를 디바이스 드라이버로 전달하는 기능
   write(파일디스크립터, &id, 4) 함수 사용
```

이걸 구현하려면 `open`

```
fd = open(dev/sk,.....)
//이것에 대한 DD를 mknod 에서 특수장치파일 만들고 fd파일디스크립터로 리턴
write(fd,.....)
//write fd 받으면
//여기까지 응용단에서 하면
```

밑에 단에서 xxx_write 함수가 호출되면서 연동될테고

- 밑에 단으로 id를 내려보내게 되고
- `get_user` 를 써서 받으면
  - 유저계층에서 데이터를 가져오라는 것.
  - 이 id를 받으면 이 프로세스 id가 구해졌을 테고
- 이 id에다가 신호를 쏘자. 시그날을 쓰자.
- sp시간에 배운 `kill`은 응용단에서 쓰는 것,
  - 킬 밑에 단에서는 컴파일 안되는 것.

그래서 킬 쓰기 위해 유저함수로 짜 놓은것. 그대로 쓰면 된다.

`int my_kill_proc(pid_t pid, int sig) {...}`

- 첫번째 매게변수 `pid_t pid`
  - 프로세스 id
  - 위에서 받은 id값 넘겨주고
- 두번째 매게변수 `int sig`
  - 시그날은 유저정의 시그날 넣으면 된다.
  - SIGUSR1, SIGUSR2...
- ***이렇게 써주면 신호가 커널에서 응용으로 올라가게 된다.***
  - 프로세서 많이 떠 있는데 정확히 id 딱 찝어서 쏴줄 수 있다.
  - signal 함수 초기화해서 기다리다가
  - 받을 시그날 sigusr 등록하고 여기다 시그널 핸들러 만들면
  - 신호 도착하면 시그널 핸들러 호출될 것이다.
  - 신호 도착하면 실행

```
//int my_kill_proc(pid_t pid, SIGUSR1) 이렇게 써라!
//그럼 이게 윗단으로 정확히 갈테니까!
//신호를 받는 순간 시그널 호출!
int my_kill_proc(pid_t pid, int sig) {
    int error = -ESRCH;           /* default return value */
    struct task_struct* p;
    struct task_struct* t = NULL;
    struct pid* pspid;
    rcu_read_lock();
    p = &init_task;               /* start at init */
    do {
        if (p->pid == pid) {      /* does the pid (not tgid) match? */
            t = p;
            break;
        }
        p = next_task(p);         /* "this isn't the task you're looking for" */
    } while (p != &init_task);    /* stop when we get back to init */
    if (t != NULL) {
        pspid = t->pids[PIDTYPE_PID].pid;
        if (pspid != NULL) error = kill_pid(pspid,sig,1);
    }
    rcu_read_unlock();
    return error;
}
```

이 함수는 커널에 없기 때문에 이 부분을 긁어다가 DD 잡아서 테스트 할 때 .c 끝부분에 넣어라. 그러면 이거 쓸 수 있다.

시그널 핸들러란. : 신호를 받으면 거기에 등록되어 있는 함수가 실행되는것.

리눅스에서 `kill -l` 치면 리눅스에서 사용되는 신호목록 확인 가능

여기서 보면 10번하고 12번이 유저정의시그널이다.

- `SIGUSR1`, `SIGUSR2`

> 예제코드 확인 `sigcatch1.c`

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

void mySigHdlr(int signo)
{
  if (signo == SIGINT) {
    printf("received SIGINT\n");
  }
}

int main(void)
{
```

```
  if (signal(SIGINT, mySigHdlr) == SIG_ERR) {
    printf("\ncan't catch SIGINT\n");
  }
  while(1) {
    sleep(1);
  }
  return 0;
}
```

`kill -l` 해보면 `SIGINT`는 2번신호

컴파일 하고 실행해보면

```
# gcc sigcatch1.c
# ./a.out
^Creceived SIGINT
^Creceived SIGINT
^Creceived SIGINT
^Creceived SIGINT
```

- 컨트롤+c 가 `SIGINT` kill -l 에 2번
- 컨트롤+z 가 `SIGKILL` kill -l 에 9번

main문에 조건문을 보면 `(signal(SIGINT, mySigHdlr) == SIG_ERR) {...}`

- signal 시그널함수 호출
- SIGINT 이건신호

SIGINT라는 신호가 들어오면 mySigHdlr 라는 함수 실행하라는 말

아까 타이머핸들러는 정해진 집스에 HZ 더해서 그 시간에 도달하면 펑션에 등록되어 있어서 자동실행

시그널은 프로세스가 이 신호를 받으면 이 함수를 실행하라. 이게 시그널 핸들러

상단에 등록되어 있는 `mySigHdlr` 함수를 보면 SIGINT 들어오면 이 함수 신호

`int signo`에 SIGINT 2번이 넘어오게 된다.

```
void mySigHdlr(int signo)
{
  if (signo == SIGINT) {
    printf("received SIGINT\n");
  }
  //넘어온 신호가 SIGINT가 같으면 실행하라
}
```

리눅스에서 `ps - ef`

```
# ps - ef
root     26558  2614  0 17:09 pts/0    00:00:00 ./a.out
root     26572  2614  0 17:10 pts/0    00:00:00 ./a.out
root     26573  2614  0 17:10 pts/0    00:00:00 ./a.out
root     26647  2614  0 17:19 pts/0    00:00:00 ./a.out
root     26652  2614  0 17:19 pts/0    00:00:00 ps -ef
```

2번째 프로세스 아이디 3번째깨 부모id

지금 우리가 하고 있는 과제에서 신호를 등록하려면

- signal 함수를 디바이스 드라이버 모듈 or app?
  - app에서 사용해야 한다.
- 핸들러는?
  - app 에다가!!
- 그러면 보내는 신호는 무슨 신호 써야하냐?
  - 10. SIGUSR1
  - 12. SIGUSR2
  - 테스트용으로 만든거 쓰고

자 다시. 모듈에선

`my_kill_proc` 이 함수를 어떻게 써야하냐?

- int my_kill_proc(pid_t pid, SIGUSR1) 이렇게 써라!
- 그럼 이게 윗단으로 정확히 갈테니까!
- 신호를 받는 순간 시그널 호출!

이게 기본이 되면 별것 다 해볼 수 있다.

그 시그널 핸들러로 포크, 쓰레드, 또 그 쓰레드와 모듈하고 카피튜유저 통신할 수도 있는 거고 아이디어, 현업에선 이렇게 쓴다.!

- HINT1

```
#include <linux/sched.h>
```

- HINT2
  - `get_user` 쓸 때 끝에는 4 쓰면 도니다.
- HINT3

```
static int sk_write(struct file *filp, c...)
{
    char data[11];

    get_user(id, (int *)buf);
    printk("\n [kernel] id = %d", id);
```

```
    my_kill_proc(id, SIGUSR1);

    return count;
}
```

- HINT4
  - 시그널 받으면 프로세스 죽으면 안된다. 하고 기다리고 있어야 한다.
  - pause()
  - while()