KU_PIR 보고서

과목	임베디드 소프트웨어1
학과	컴퓨터 공학과
학번	201311299
이름	이원오

<Submission>

- 인체 감지 센서(pir 센서)를 사용하여 공간에 사람이 있는 시간을 모니터링하기 위한 디바이스 및 라이브러리 제작.
- 작동 시간은 Linked List를 사용해서 Kernel 내부에 저장.
- 인체 감지 -> IRQF_TRIGGER_RISING 인체를 놓침 -> IRQF_TRIGGER_FALLING
- 적절한 동기화 방식을 사용해서 데이터 보호. -> spin_lock , rcu_read_lock

<Basic Design>

1. ku_pir.h

```
#define KU_PIR_START_NUM 0x80
#define KU_PIR_NUM1 KU_PIR_START_NUM+1
#define KU_PIR_NUM2 KU_PIR_START_NUM+2
#define KU_PIR_NUM3 KU_PIR_START_NUM+3
#define KU_PIR_NUM4 KU_PIR_START_NUM+4
#define KU_PIR_NUM5 KU_PIR_START_NUM+5

#define KU_PIR_NUM 'z'
#define KU_PIR_INSERTDATA       _IOWR(KU_PIR_NUM, KU_PIR_NUM1, unsigned long *)
#define KU_PIR_READ             _IOWR(KU_PIR_NUM, KU_PIR_NUM2, unsigned long *)
#define KU_PIR_CLOSE            _IOWR(KU_PIR_NUM, KU_PIR_NUM3, unsigned long *)
#define KU_PIR_INIT             _IOWR(KU_PIR_NUM, KU_PIR_NUM4, unsigned long *)
#define KU_PIR_FLUSH            _IOWR(KU_PIR_NUM, KU_PIR_NUM5, unsigned long *)

#define KUPIR_MAX_MSG 15
#define KUPIR_SENSOR 17

#define DEV_NAME "ku_pir_dev"

struct ku_pir_data{
        long unsigned int timestamp;
        char rf_flag;
};
```

- ioctl함수에 필요한 command 선언 및 ku_pir_data 구조체 선언
- KUPIR_SENSOR는 17번 gpio로 설정.
- 라이브러리에 있는 함수에 필요한 ioctl명령어(INSERTDATA, READ, CLOSE, INIT, FLUSH)

2. ku_pir.c

```c
#include "ku_pir.h"
#include <linux/module.h>
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/gpio.h>
#include <linux/interrupt.h>
#include <linux/spinlock.h>
#include <linux/wait.h>
#include <linux/cdev.h>
#include <linux/list.h>
#include <linux/slab.h>
#include <linux/uaccess.h>
#include <linux/rculist.h>
#include <linux/sched.h>
MODULE_LICENSE("GPL");

struct queue{
        struct list_head list;
        struct ku_pir_data data;
};

struct queue_list{
        struct list_head list;
        struct queue q;
        pid_t pid;
};

spinlock_t my_lock;
wait_queue_head_t my_wq;
static int irq_num;
static dev_t dev_num;
static struct cdev *cd_cdev;
struct queue_list my_queues;

pid_t init_queue(void);
struct queue *get_queue(pid_t pid);
int is_not_exist(pid_t pid);
int get_num(pid_t pid);
int is_full(pid_t pid);
int remove_queue(void);
int remove_queue_list(void);
void remove_old_queue(pid_t pid);
void insert_from_isr(long unsigned int timestamp, char rf_flag);
void ku_pir_read(struct ku_pir_data *kpd);
int insert_from_user(struct ku_pir_data * kpd);
```

- queue 구조체 -> 실질적인 ku_pir_data를 갖고 있는 linked list / pir 센서의 데이터가 보관되는 구조체.
- queue_list 구조체 -> 현 프로세스의 id인 pid와 queue구조체를 갖고 있는 linked list / 전체의 큰 자료구조
- spinlock_t my_lock -> 동기화에 필요한 spinlock 함수를 위한 전역 변수.
- wait_queue_head_t my_wq -> wait_queue 함수를 위한 전역 변수.
- irq_num : 인터럽트 핸들러 등록에 필요한 interrupt number 전역 변수.
- my_queues : 전체의 queue_list 구조체를 위한 전역 변수.
- 사용할 함수의 원형 선언 (detail은 Description for important functions 참고)

<Description for important functions>

● ku_pir_lib.c

1. int ku_pir_open()

int ku_pir_open()	Functionality	호출한 프로세스에서 사용할 자료구조를 초기화. -> ioctl KU_PIR_INIT 명령 실행
	Parameters	-
	Return Value	성공했을 시 해당 디바이스 파일에 할당된 실제 fd값 반환. 실패 시 -1 반환.

2. int ku_pir_close (int fd)

int ku_pir_close (int fd)	Functionality	호출한 프로세스가 사용했던 자료구조를 제거하는 함수 -> ioctl KU_PIR_CLOSE 명령 실행
	Parameters	file descriptor / 해당 프로세스에 할당된 fd값
	Return Value	성공시 0, 실패시 -1 리턴

3. void ku_pir_read (int fd, struct ku_pir_data *data)

void ku_pir_read (int fd, struct ku_pir_data* data)	Functionality	fd에 해당하는 자료구조에서 저장되어 있는 PIR 데이터를 읽어오는 함수. 커널에 blocking 방식으로 데이터를 요청. (데이터가 존재하지 않는 경우 데이터가 들어올 때까지 대기.) -> ioctl KU_PIR_READ 명령 실행
	Parameters	읽고 싶은 자료구조의 fd pir데이터를 읽어올 ku_pir_data 구조체
	Return Value	-

4. void ku_pir_flush (int fd)

void ku_pir_flush (int fd)	Functionality	fd에 해당하는 자료구조의 데이터를 모두 제거. -> ioctl KU_PIR_FLUSH 명령 실행
	Parameters	제거하고 싶은 자료구조의 fd (file descriptor)
	Return Value	-

5. int ku_pir_insertData (long unsigned int ts, char rf_flag)

int ku_pir_insertData (long unsigned int ts, char rf_flag)	Functionality	모든 커널 자료구조에 timestamp와 rf_flag값을 인위적으로 추가하는 함수 -> ioctl KU_PIR_INSERTDATA 명령 실행
	Parameters	추가하고 싶은 센서데이터 (timestamp, rf_flag)
	Return Value	성공 시 0 반환 실패 시 -1 반환

● ku_pir.c

1. pid_t init_queue(void)

pid_t init_queue (void)	Functionality	전체 리스트(my_queues)에 현재 프로세스의 pid값을 갖는 자료구조를 추가 및 초기화하는 함수 -> 사용자에서 ku_pir_open을 실행할 때 커널 내부에서 실질적으로 실행되는 함수
	Parameters	-
	Return Value	할당된 pid값 리턴

2. struct queue* get_queue(pid_t pid)

struct queue* get_queue (pid_t pid)	Functionality	전체 리스트에서 현 프로세스의 pid를 갖는 queue 구조체를 반환하는 함수
	Parameters	구하고 싶은 queue가 있는 pid값
	Return Value	성공 시 해당 pid의 queue 구조체 포인터 실패 시 NULL값 반환

3. int is_not_exist(pid_t pid)

int is_not_exist (pid_t pid)	Functionality	인자값 pid에 해당하는 queue 구조체가 있는지 없는지 확인해주는 함수
	Parameters	확인하고 싶은 pid값
	Return Value	만약 pid값을 갖는 queue 구조체가 없을 경우 1을 반환 존재하는 경우에는 0을 반환

4. int get_num(pid_t pid)

int get_num (pid_t pid)	Functionality	해당 pid를 갖는 queue구조체에 실질적인 ku_pir_data가 몇 개 들어있는지 확인 해주는 함수
	Parameters	데이터의 개수를 확인하고 싶은 해당 queue의 pid값
	Return Value	해당 pir 데이터의 개수 반환 pid에 해당하는 queue가 없을 경우 -1 반환

5. int is_full(pid_t pid)

int is_full (pid_t pid)	Functionality	해당 pid를 갖는 queue에 자료개수가 KUPIR_MAX_MSG보다 큰지 확인해주는 함수
	Parameters	데이터의 상태를 확인하고 싶은 queue의 pid값
	Return Value	데이터 개수가 KUPIR_MAX_MSG보다 클 경우 1 반환 // 작을 경우 0 반환

6. int remove_queue(void)

int remove_queue (void)	Functionality	현재 프로세스의 pid를 갖는 queue에 있는 pir 데이터들 (ku_pir_data)을 모두 삭제시키는 함수 -> ku_pir_flush할 때 실행되는 함수
	Parameters	-
	Return Value	pid를 갖는 queue값이 없을 경우 -1 반환 삭제 성공 시 0 반환

7. int remove_queue_list(void)

int remove_queue_list (void)	Functionality	해당 pid에 해당하는 queue_list를 삭제하는 함수 -> ku_pir_close할 때 실행되는 함수
	Parameters	-
	Return Value	pid값을 갖는 queue값이 없는 경우 -1 반환 삭제 성공시 0 반환

8. void remove_old_queue(pid_t pid)

void remove_old_queue (pid_t pid)	Functionality	pid에 해당되는 queue에 있는 센서 데이터중에서 가장 오래된 데이터를 제거하는 함수 -> 데이터가 KUPIR_MAX_MSG를 넘어가는 경우 가장 오래된 데이터를 제거하기 위해서 사용
	Parameters	찾고 싶은 queue의 pid값
	Return Value	-

9. void insert_from_isr (long unsigned int timestamp, char rf_flag)

void insert_from_isr (long unsigned int timestamp, char rf_flag)	Functionality	pir 센서에서 인터럽트가 발생할 시 해당 정보(ts, rf_flag)를 모든 queue에 추가해주는 함수
	Parameters	인터럽트가 발생해서 측정된 센서 데이터값
	Return Value	-

10. void ku_pir_read(struct ku_pir_data *kpd)

void ku_pir_read (struct data_pid *df)	Functionality	pid값에 해당하는 자료구조에 존재하는 값을 실질적으로 읽어서 사용자 영역으로 넘겨주는 함수 (copy_to_user함수 사용)
	Parameters	struct ku_pir_data 구조체 포인터 입력을 받을 user영역의 ku_pir_data구조체 포인터 값
	Return Value	-

11. insert_from_user(struct ku_pir_data *kpd)

insert_from_user (struct ku_pir_data *kpd)	Functionality	User영역에서 전체 queue리스트에 인위적으로 pir센서 데이터를 커널 영역으로 추가해주는 함수 -> 실질적인 ku_pir_insertData함수 (copy_from_user함수 사용)
	Parameters	유저가 입력하고 싶은 pir 센서의 데이터 값 (ku_pir_data 구조체 포인터)
	Return Value	성공 시 0 반환 실패 시 -1 반환

12. static irqreturn_t ku_pir_isr(int irq, void* dev_id)

static irqreturn_t ku_pir_isr (int irq, void *dev_id)	Functionality	인터럽트 발생시 실행될 함수.
	Parameters	해당 인터럽트의 번호 및 디바이스의 자료구조
	Return Value	IRQ_HANDLED (인터럽트 handle)

13. static long ku_pir_ioctl
(struct file *file, unsigned int cmd, unsigned long arg)

static long ku_pir_ioctl (struct file *file, unsigned int cmd, unsigned long arg)	Functionality	캐릭터 디바이스에 등록할 ioctl 함수 cmd(커맨드)에 들어갈 값에 따라서 각기 다른 기능 실행
	Parameters	file descriptor (open한 디바이스의 정보), ioctl에 쓰일 명령어, ioctl함수에 필요한 인자값
	Return Value	cmd(명령)에 따라서 다양한 리턴 값

14. static int __init ku_pir_init(void)

static int __init ku_ipc_init (void)	Functionality	커널 모듈을 시작하는 함수 캐릭터 디바이스 등록 / gpio 할당 및 gpio-irq 연결 설정 / irq request
	Parameters	-
	Return Value	성공 시 0 반환

15. static void __exit ku_pir_exit(void)

static void __exit ku_ipc_exit (void)	Functionality	커널 모듈을 종료하는 함수 캐릭터 디바이스 해제 및 커널 영역에 존재하는 queue리스트 삭제 irq 및 gpio restore 작업 진행
	Parameters	-
	Return Value	-