# 후진 경고 시스템

2024-10-31
김동민

# 과제 개요

Telechips

- 프로젝트 목표

후진 시 충돌을 방지하기 위하여 경고 시스템을 구현한다.

- 프로젝트 내용

거리가 가까워지면 LED와 부저로 경고해준다.

- 결과

1. 메인 기능은 구현 성공
2. 구현 실패
   a. TOPST, Raspberry pi간 통신 구현 실패(Rapsberry pi내에서 2개의 프로세스간 통신으로 대체)
   b. 버튼으로 서비스 ON/OFF 구현 실패(버튼을 눌렀을 때 메시지 출력 + 채터링 문제 해결까지만 구현)

# 세부 기능 목표

Telechips

- Sensor process

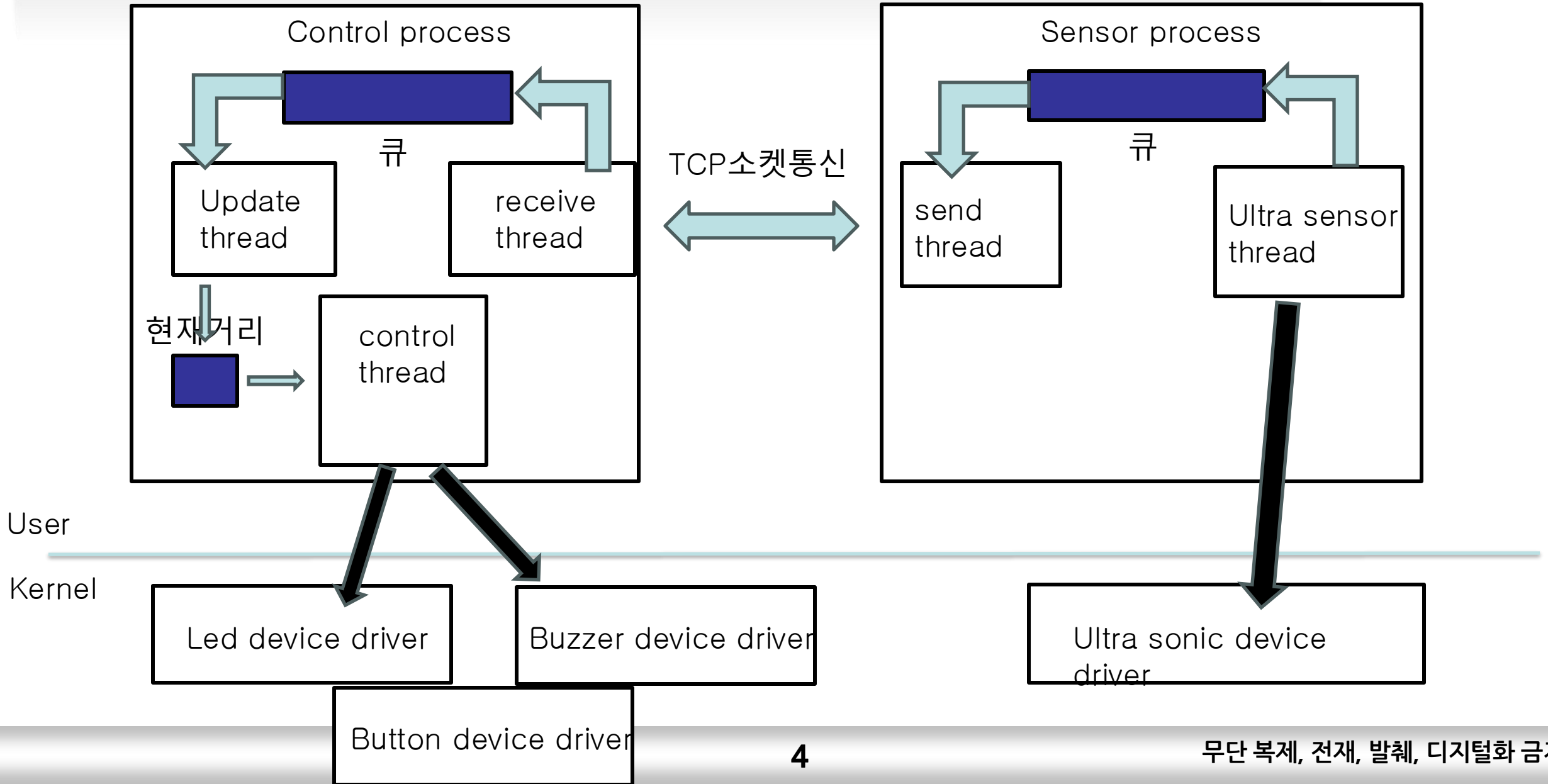
1. 주기적으로 차량 후미와 장애물 사이의 거리 측정
2. 측정 데이터를 네트워크를 통해 Control process로 전달

- Control process

1. Sensor Process로부터 거리데이터를 수신받는다.
2. 거리정보에 따라 경고 시스템을 활성화/비활성화 한다.
   a. 거리가 일정 이하로 짧아지면 경고시작(LED : GREEN -> RED, 부저 ON)
   b. 거리가 좁아질수록 부저가 더 빈번하게 울린다.
   c. 거리가 일정 이상 회복되면 경고 해제(LED : RED -> GREEN, 부저 OFF)

# 프로젝트 내용(전체 구조)

Telechips



# 핵심 기술1(producer&consumer + 상호배제)

Telechips

## queue


P연산(큐가 비었다면 wait)

```
sem_wait(&consume);

pthread_mutex_lock(&mutex);
new_node = dequeue();
pthread_mutex_unlock(&mutex);
```

```
new_node = create_node(distance_data);

pthread_mutex_lock(&mutex);
enqueue(new_node);
pthread_mutex_unlock(&mutex);

sem_post(&consume);
```

V연산(큐에 삽입했다고 알림)

- 장점

1. 세마포어로 producer consumer 구현

   -> dequeue하는 쓰레드에서 큐가 비었는지 계속 체크할 필요X

   (CPU 자원 낭비 방지)

2. 뮤텍스로 상호배제

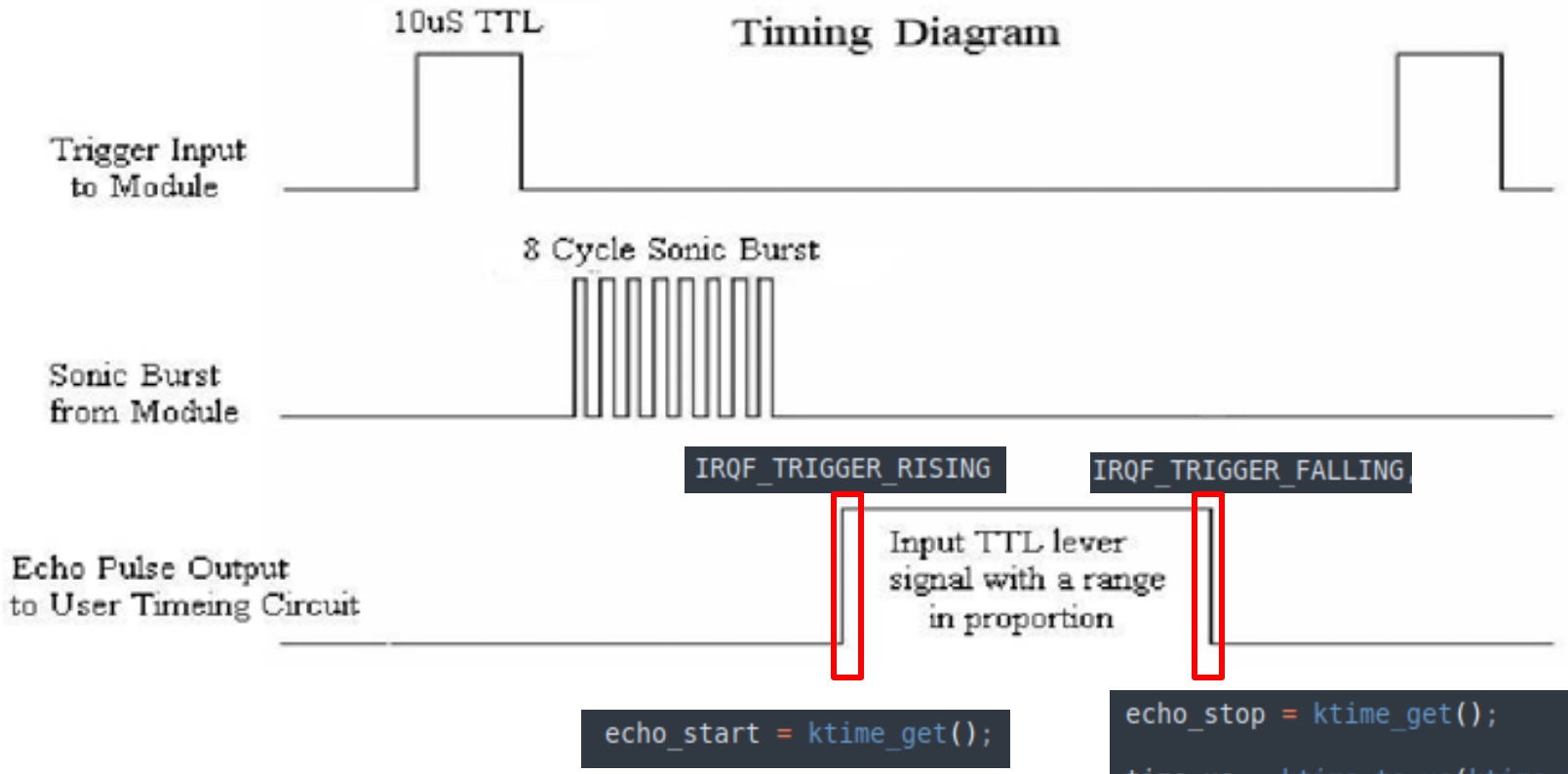
   -> 공유자원인 큐에 대해서 race condition 방지

# 핵심 기술2(초음파 측정)

Telechips

```
gpio_set_value(TRIG_PIN, 1);
udelay(10);
gpio_set_value(TRIG_PIN, 0);

wait_event_interruptible(wait_queue, ultra_state == 3);
```




```
echo_start = ktime_get();
```

```
echo_stop = ktime_get();

time_us = ktime_to_us(ktime_sub(echo_stop, echo_start));
cm = time_us/58;

wake_up_interruptible(&wait_queue);
```

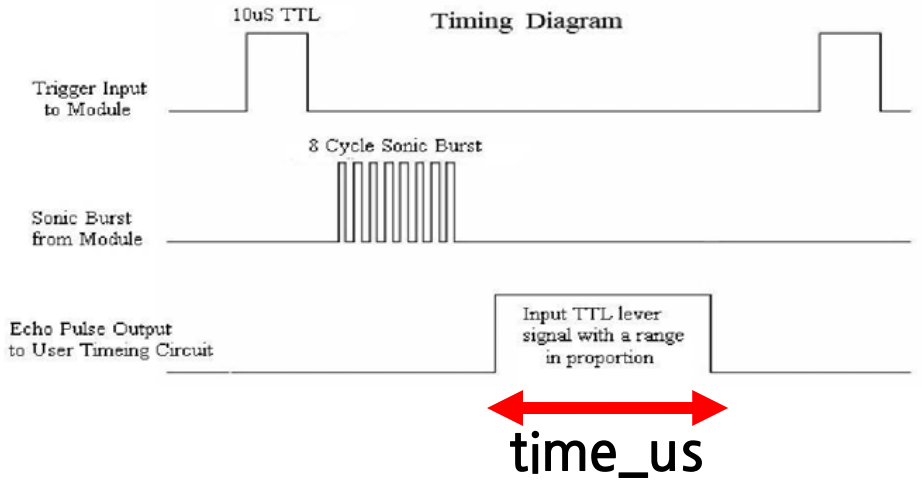
# 핵심 기술2(cm = time_us ÷ 58)

Telechips

```
echo_stop = ktime_get();

time_us = ktime_to_us(ktime_sub(echo_stop, echo_start));
cm = time_us/58;

wake_up_interruptible(&wait_queue);
```




거리 = 시간 × 속력

초음파 속력 = 약 340m/s

거리(cm) = 시간(us) ÷ 2 × 340m/s
(time_us는 왕복 시간이므로 ÷2)

= 시간 (us) × 170m/s
= 시간 (us) × 0.00017m/us
= 시간 (us) × 0.017cm/us
= 시간 ÷ 58.82... = 거리(cm)

# 핵심 기술3(채터링 문제 해결)

Telechips

채터링 문제 : 버튼을 한번 누르면, 인터럽트가 여러번 발생

해결방법 :

버튼 interrupt handler에서...

1. 버튼 interrupt를 비활성화(interrupt가 여러번 발생하지 않도록)
2. Kernel timer 설정(약 200ms 후에 동작하도록)

Timer interrupt handler에서...

3. 버튼 interrupt를 활성화(누르면 interrupt 발생하도록)

# 핵심 기술3(채터링 문제 해결)

Telechips

## Button interrupt handler

```
//button interrupt service routine
static irqreturn_t button_isr(int irq, void* dev_id){

    //disable_irq(button_irq);
    disable_irq_nosync(button_irq);

    button_timer.delay_jiffies = msecs_to_jiffies(200);
    button_timer.irq_num = button_irq;
    timer_setup(&button_timer.timer, button_timer_func, 0);
    button_timer.timer.expires = jiffies + button_timer.delay_jiffies;
    add_timer(&button_timer.timer);

    return IRQ_HANDLED;
}
```

## Timer interrupt handler

```
static void button_timer_func(struct timer_list* t){
    struct button_timer_info *info = from_timer(info, t, timer);

    enable_irq(info->irq_num);
}
```

# 핵심 기술3(커널 패닉 발생 및 원인 추측)

Telechips

## Button interrupt handler

```
//button interrupt service routine
static irqreturn_t button_isr(int irq, void* dev_id){

    //disable_irq(button_irq);
    disable_irq_nosync(button_irq);

    button_timer.delay_jiffies = msecs_to_jiffies(200);
    button_timer.irq_num = button_irq;
    timer_setup(&button_timer.timer, button_timer_func, 0);
    button_timer.timer.expires = jiffies + button_timer.delay_jiffies;
    add_timer(&button_timer.timer);

    return IRQ_HANDLED;
}
```

* In contrast to `disable_irq_nosync()`, `disable_irq(n)` waits until all interrupt handlers for IRQ*n* that are running on other CPUs have completed before returning.

(Understanding Linux Kernel 책에서 발췌)

*disable_irq* not only disables the given interrupt but also waits for a currently executing interrupt handler, if any, to complete.

(Linux Device Driver 책에서 발췌)

- Interrupt handler에서 현재 interrupt를 disable_irq()로 비활성화하면 **무조건 wait한다.**(현재 해당 interrupt handler를 실행 중이니까)
- **Interrupt handler는 다른 process에 기생하여 실행되기 때문에(process가 아니기 때문에) 스케줄링을 유발하는 동작을 해선 안된다.**
- 결과적으로 커널 패닉 발생
- 따라서 disable_irq()가 아닌 disable_irq_nosync()를 사용해야 한다.

# 결론

Telechips

- 이번 프로젝트에서 얻은 점
  1. 텔레칩스 교육에서 배운 내용(큐, 상호배제, 세마포어, wait queue등) 사용 경험
  2. disable_irq와 disable_irq_nosync의 차이 이해
  3. interrupt context에서 스케줄링 유발하는 함수를 호출하면 안된다는 사실을 상기
  4. 하나의 led device driver로 두개의 LED를 제어함으로써, device driver는 장치마다 존재하는 것이 아닌 장치 종류마다 하나씩 존재한다는 사실을 상기

- 아쉬운 점
  1. device driver에서 gpio레지스터에 직접 접근하는 것이 아닌, 라이브러리 함수를 사용
  2. TOPST D3를 활용하지 못함.
  3. button으로 서비스 ON/OFF 기능을 구현하지 못함.