Embedded System Software 과제 3

(과제 수행 결과 보고서)

과목명: **[CSE4116] 임베디드시스템소프트웨어**

담당교수: **서강대학교 컴퓨터공학과 박성용**

학번 및 이름: **20161566, 권형준**

개발 기간: **2021.05.14~2021.05.30**

1. 개발 목표

Module programming, 디바이스 드라이버 구현, interrupt 등 배운 내용을 활용하여 간단한 stopwatch프로그램을 작성한다. interrupt부분에서는 수업시간에 배운 top half와 bottom half를 사용하여 다양한 종류의 interrupt를 학습한다.

2. 개발 범위 및 내용

Stop Watch의 start, stop, reset 등 기능을 구현하기 위해 그리고 FND에서 현재 timer를 보여주기 위해 timer interrupt, 직접 구현한 interrupt를 사용한다. 제출물은 크게 2개로 나뉜다. Stopwatch를 실행시키는 app.c와 실제 timer가 동작하는 모듈인 interrupt.c가 있다.

- App.c

  디바이스 파일(/dev/stopwatch)를 open하고 해당 stopwatch를 실행한다. 실행하기 위한 command를 write에 구현하였다. 마지막으로 디바이스 파일을 close하고 프로그램을 종료한다. 이때 stop watch가 실행되는 동안에는 해당 process는 sleep이 되며 이는 뒤에 interrupt.c의 write file operations에서 설명한다.

- Interrupt.c

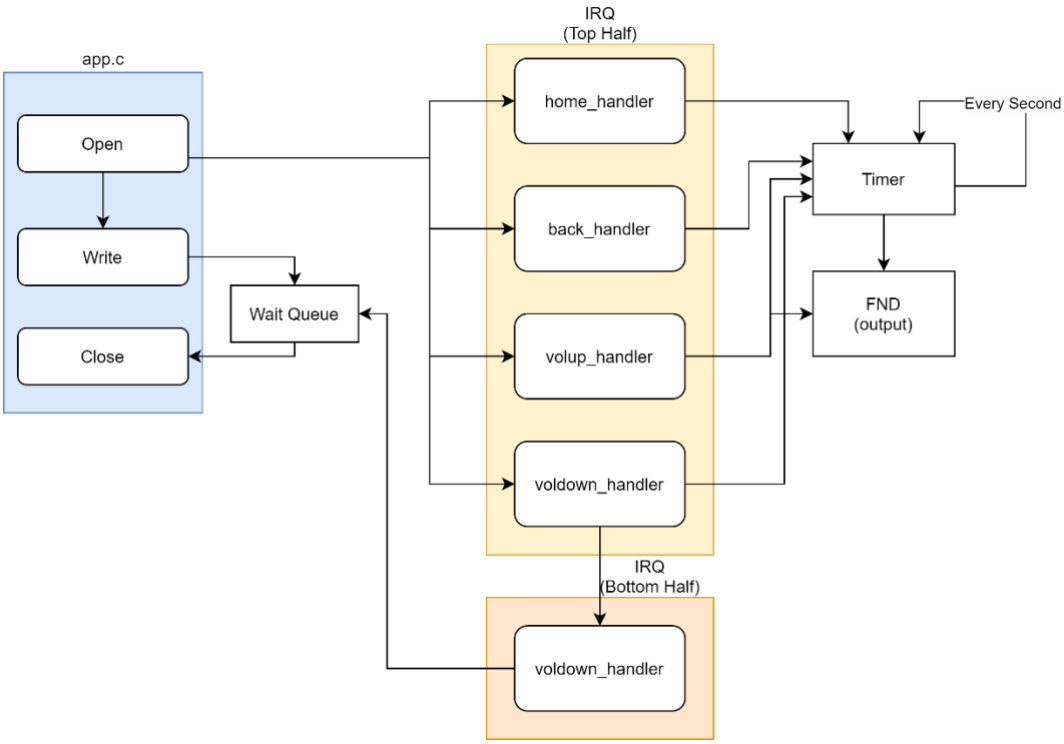
  Module을 작성하여 device driver를 구현한다. 기본적인 file operations들과 hardware button(home, back, vol+, vol-)에 대한 interrupt를 irq_desc에 등록한다. 이를 위하여 기존에 kernel에서 기존의 interrupt에 대한 코드를 수정(주석처리)하여 recompile한다. (이는 실습에서 미리 진행하였다) 총 4개의 interrupt handler(top half)를 작성하고 하나의 tasklet 수행 함수(bottom half)를 사용하였다. Stop watch의 가장 중요한 목적은 정확한 시간 측정이기 때문에 timer count의 start, stop, rest 그리고 실제로 보여지는 FND또한 즉각적인 반응이 필요하다고 판단하였다. 사실 보여지는 단위는 1초단위이기 때문에 kernel상에서는 매우 긴 시간으로 FND는 bottom half에서 진행할 수 있었지만, 조금 더 정확하게 하기 위하여 top half에 이를 추가하였다. Bottom half에서는 stop watch와는 직접적인 연관이 없는 process wakeup만을 구현하였다.

3. 추진 일정 및 개발 방법

가) 추진 일정

5/16	App.c, interrupt.c 큰 틀 작성
5/16~5/20	Timer interrupt를 사용한 stopwatch timer증가 구현
5/20~5/25	Custom interrupt handler (top half)구현
5/25~5/30	Interrupt handler(bottom half), 보고서 작성

나) 개발 방법



우선 실행 process(app.c)에서 device file을 열고 write를 호출한다. 이때 인자는 의미가 없다. Device Driver에서 open이 호출되면 board의 button(home, back, vo+, vol-)에 대한 interrupt를 등록한다. Write이 호출되면 기존에 timer가 존재했다면 이를 삭제하고 process를 sleep시킨다. 추후에 interrupt로 sleep 중인 process를 깨우기 위해 interruptible_sleep_on함수를 사용하여 wait queue에 sleep시킨다. 이제 board는 interrupt를 받기 전까지 아무런 동작도 수행하지 않는다. 받을 수 있는 interrupt의 경우의 수는 4가지로 각 button이 이에 해당한다. Home을 누르면 새로운 timer를 등록하여 1초마다 interrupt를 발생시키고 FND를 update하게 된다. Back을 누르면 현재 돌고 있는 timer가 있다면, 해당 timer의 남은 interval (expire에서 현재 jiffies값을 뺀 값)을 저장하고 timer를 멈춘다. 이후에 다시 Home을 누르면 처음 expire시간은 기존에 저장한 남은 interval로 설정하고 다시 timer를 1초마다 interrupt걸게 한다. Vol+를 누르면 현재 timer가 실행 중, 정지 중, 종료 후 상관없이 timer의 시간을 0으로 초기화한다. 기존에 실행 중이었다면 초기화한 이후에 바로 수행이 되도록 하였다. 마지막으로 vol-의 경우 IRQF_TRIGGER_FALLING뿐만 아니라 IRQF_TRIGGER_RISING의 경우에도 interrupt가 걸리게 하여(버튼을 누를 때, 버튼을 땔 때) 3초 이상 누르고 있는지를 판독하도록 하였고, 3초 이상 누르고 있는 것을 확인하기 위하여 end_timer라는 또 하나의 timer를 사용하였다. 즉, 처음 눌렀을 때 expire time을 3초로 하는 timer를 추가하였고 그 사이에 button release가 되면 timer를 삭제하였다. Timer가 무사히 3초

간 살아있었다면 시간을 초기화하고 FND를 reset하는 hardware에 민감한 작업은 interrupt handler에서 수행하고, bottom half에서 수행해도 되는 process wakeup은 tasklet을 사용하여 구현하였다.

4. 연구 결과

가) Process Sleep

App.c로 인해 실행되는 process는 device file을 열고 write을 호출한 뒤에 sleep이 된다. 이는 interrupt.c의 write file operations에서 interruptible_sleep_on함수에 의해 구현하였다. <linux/wait.h>에 구현이 되어 있는 wait queue head를 사용하여 wait queue에 sleep 중인 process를 추가하였다. 이후에 bottom half에서 process wake up을 해야 될 때에도 미리 정의된 __wake_up 함수를 사용하였다.

나) FND, Timer 관련

FND에 현재 timer의 상태를 출력해야 되기 때문에 insmod시에 FND device와 mapping을 하였다. 1초마다 증가하는 stop watch를 구현하기 위하여 전역 변수로 현재 시간을 저장하였고, timer를 두어 expire interval을 1HZ로 하였다. 해당 timer는 kernel_timer_blink를 호출하게 되는데, 이 함수에서 또 재귀적으로 kernel_timer_blink를 1HZ이후에 expire되면 실행하게 되는 timer를 add하게 된다. Kernel_timer_blink내에서 stopwatch_time을 증가시키는 동시에 fnd_update를 호출하여 현재 stopwatch_time를 FND에 출력하였다. 이외에 fnd_reset함수는 프로그램이 종료되거나 vol+가 눌려 시간을 초기화 하는 경우에 사용하였는데, 이는 fnd에 0000을 출력하게 된다.

Timer는 총 2개를 사용하였다. 하나는 stopwatch의 시간 증가를 하기 위한 timer와 종료 조건을 만족하는지 확인하기 위한 timer이다. 우선 첫번째 timer는 home이 눌려 stopwatch가 시작하면 expire time을 current jiffies + 1HZ로 하여 1초 후에 expire하게 설정을 하고 그때마다 stopwatch_time을 update하였다. back으로 인해 pause가 될 경우 남은 expire time을 다음 expire time으로 해주는 과정도 back handler에 구현하였다. 두번째 timer인 end_timer의 경우 vol-을 3초간 누르고 있었는지 확인하기 위해 사용하였다. 이 timer는 expire되면 end_stopwatch라는 함수가 호출이 되며 이 함수에서 tasklet을 사용하여 bottom half를 등록하여 추후에(system safe time) 실행이 되도록 하였다.

다) Interrupt Handler (Top Half)

- **Home handler**

  Home button이 눌렸을 때 실행되는 interrupt handler이다. 정지되어 있는 stopwatch를 실행시키는 역할을 한다. 기존에 resume_time이 0이 아니라면, 즉 pause 상태라면, 마지막 left expire time만큼 next expire time을 설정한다. 실제 stop watch에서 '시작'버튼을 누르면 바로 실행되는 것은 중요하기 때문에(눈에 보이는 시계 또한) 모든 과정을 top half에서 처리하였다.

- **Back handler**

  Back button이 눌렸을 때 실행되는 interrupt handler이다. 현재 stopwatch가 돌고 있다면 pause시키는 역할을 한다. 이 또한 누르는 시점이 stopwatch특성상 민감하기 때문에 모든 것을 top half에서 처리하였다. paus하기 전에 현재 timer의 expire time에서 현재 jiffies를 빼서 남은 expire time을 (소수점 첫째 자리까지) 저장해 둔다.

- **Volup handler**

  Vol+ button이 눌렸을 때 실행되는 interrupt handler이다. 현재 stopwatch의 시간을 0으로 초기화하는 역할을 한다. 만약 기존의 stopwatch가 돌고 있었다면 초기화한 이후 새로운 timer를 추가하여 실행시켜준다. Pause 중이어서 resume_time이 0이 아니었던 경우에도 timer를 reset시켰기 때문에 0으로 초기화해준다. 이 또한 hardware와 밀접한 동작을 많이 하기 때문에 모든 것을 top half에서 처리하였다.

- **Voldown handler**

  Vol- button이 눌리거나 release되었을 때 수행되는 interrupt handler이다. 우선 pressed된 경우 end_timer라는 expire time이 3초인 timer를 추가한다. 해당 timer의 expire time이 되기 전에 release가 되면 end_timer를 삭제한다. 이러한 구현으로 인해 3초간 누르고 있음을 보장해줄 수 있다. 앞에서 말한 동작은 time sensitive하기 때문에 top half에서 진행하였다. 또한 end_timer가 expire되는 시점에 수행되는 end_stopwatch에서 fnd_reset또한 FND에 즉각적인 수행이 필요함으로 top half에서 수행하였다. 하지만 마지막으로 sleep 중이던 process를 wakeup시키는 과정은 긴박하거나 hardware에 관련된 것이 아님으로 bottom half(tasklet)을 사용하였다.

라) Interrupt Handler (Bottom Half)

본 프로젝트에서 top half와 bottom half를 구분하기 위한 기준은 hardware와 연관이 있는지였다. Button을 눌렀을 때, 사용자에게 보여지는 FND와 내부에 저장된 시간 등

대부분의 영역이 이에 해당하였다. 구현하기 위해 Softirq를 사용하는 tasklet을 사용하였다. DECLARE_TASKLET Macro를 사용하여 short_do_tasklet이라는 함수를 tasklet으로 만들었으며, 해당 함수는 앞에서 sleep 중이던 wait queue의 process를 깨워주는 역할을 한다. 이 tasklet은 end_stopwatch에서 scheduling이 되어 system-determined safe time에 수행이 된다. Process를 wakeup하는 것은 hardware에 관련되어 있거나, stopwatch의 수행에 영향을 끼치지 않기 때문에 bottom half에 구현하였다.

5. 기타

앞에서 배운 module programming, device driver, timer interrupt를 모두 사용하면서 추가로 interrupt handler를 실제로 implementation할 수 있는 기회여서 모든 것을 복습하는 느낌이었다. 다만 bottom half와 top half를 나누기는 했지만, 실제로 체감이 되지 않아서 아쉬웠다.