가)표지

시스템프로그래밍(F049-1)

과제 3 레포트

LED 제어 버튼 프로그램 만들기

아주대학교 소프트웨어학과

202220775

박민정

제출일: 2025.05.29.

나)LED 제어 프로그램 소스코드 설명

이번 과제에선 Raspberry Pi환경에서 GPIO핀을 제어하여, 버튼 입력에 따라 LED의 상태를 제어한다. 해당 과제는 다음과 같은 절차로 구성되어 있다.

초기 설정

상태 변수 state, prev_state, LED 상태 변수 light, 버튼 누름 지속 시간 변수 presstime 등을 선언한다.

버튼 입력 핀(PIN=20), LED 출력 핀(POUT=17), 버튼 전원 공급 핀(POUT2=21)을 GPIOExport() 및 GPIODirection()을 통해 각각 export하고 방향을 설정한다.

LED를 끈 상태(LOW)로 초기화하고, 버튼이 동작하기 위해 21번 핀에 HIGH 전압을 공급한다.

'while(1)'을 통한 버튼 입력 감지 및 LED 제어

GPIORead(PIN)으로 버튼의 현재 상태를 읽는다.

GPIOWrite(POUT2, 1)로 버튼 입력을 위한 전압을 지속적으로 공급한다.

1)현재 버튼이 눌려있는 경우 (state==0)

1-a)버튼이 새롭게 눌린 경우 (prev_state==1)

LED상태의 값을 반전하여, 토글한다.

반전된 값을 GPIOWrite(POUT,light)를 통해 LED에 출력한다.

1-b)버튼이 계속 눌려있는 경우 (prev_state==0)

'presstime'을 100ms씩 증가시킨다.

'presstime'>=800인 경우, 버튼이 계속 눌려있는 동안 500ms 간격으로 LED상태를 반전하여, 점멸효과를 구현한다.

2)버튼이 눌려있지 않은 경우 (state==1)

'presstime'을 0으로 초기화한다.

3)공통 처리

'prev_state=state'를 통해 현재 상태를 이전 상태로 저장한다.

'usleep(100 * 1000)'을 통해 루프를 100ms간격으로 반복한다.

다)디바이스 드라이버 소스코드 구조 분석

본 프로그램에서 사용된 디바이스 드라이버 관련 함수들은 사용자 공간에서 GPIO를 제어할 수 있도록 커널과 통신하는 인터페이스를 제공한다. 각 함수는 특정 파일 시스템 경로를 통해 GPIO에 접근한다.

1. GPIOExport(int pin)

/sys/class/sysprog_gpio/export 경로에 핀 번호를 작성하여 export 한다.

open() 함수로 파일을 열고, snprintf()를 사용해 정수 핀 번호를 문자열로 변환한 후, write()로 해당 값을 작성한다. 해당 작업을 통해 커널은 GPIO 핀을 사용자 공간에서 접근 가능하도록 설정한다.

2. GPIODirection(int pin, int dir)

문자열 배열과 파일경로를 생성한다. 이후 "in" 또는 "out" 문자열을 선택하고, write() 함수로 전달한다. /dev/gpioX 파일을 열어, 선택한 in 또는 out 문자열을 작성하여 GPIO의 방향을 설정한다.

3. GPIOWrite(int pin, int value)

‘value’에 따라 "0"(LOW) 또는 "1"(HIGH)을 선택한다. /dev/gpioX 파일에 0 또는 1을 write()를 통해 문자열 형태로 작성하여 핀의 출력을 제어한다.

4. GPIORead(int pin)

read() 함수로 3바이트의 데이터를 받아오고, 버튼 또는 센서의 현재 상태를 읽는다.

이를 atoi()를 통해 정수로 변환하여 반환한다.

5. GPIOUnexport(int pin)

GPIO 핀을 더 이상 사용하지 않을 때 /sys/class/sysprog_gpio/unexport 경로에 핀 번호를 써서 커널로부터 제어권을 반납한다. GPIOExport()와 유사한 방식으로 동작한다.

위 함수들은 모두 open(), write(), read(), close()등의 시스템 콜을 사용해 GPIO를 제어한다.

라)프로그램이 GPIO를 제어하는 방식에 대한 이론적 설명

해당 과제는 Raspberry Pi 의 GPIO(General Purpose Input/Output) 핀을 제어하기 위해 File based I/O 방식을 사용한다. 이는 Memory Mapped I/O 와 달리, 사용자 공간에서 파일을 열고 읽고 쓰는 방식으로 하드웨어를 제어하는 방식이다.

임베디드 리눅스에서 프로그램이 GPIO 를 제어하는 방식은 리눅스 시스템에서 여러 계층과 인터페이스를 통해 구상되는데, 아래는 해당 과정을 설명한 내용이다.

1. File based I/O

리눅스에서는 장치를 파일처럼 다루며, 하드웨어 장치도 일반 파일처럼 read() 또는 write() 시스템 콜을 통해 제어한다.

예를 들어, /sys/class/gpio/gpioN/value 파일에 값을 쓰거나 읽는 방식으로 GPIO 핀의 상태를 변경하거나 확인할 수 있다.

2. Device Driver

디바이스 드라이버는 하드웨어(GPIO)와 커널 사이의 인터페이스를 구성하는 소프트웨어 계층이다. 드라이버는 내부적으로 하드웨어의 레지스터를 직접 조작하며, 사용자는 파일 인터페이스를 통해서만 접근가능하다.

3. Device File

커널은 각 디바이스를 /dev 디렉터리 아래의 디바이스 파일로 노출시킨다. 예를 들어, /dev/gpiochip0 같은 이름으로 존재하며, 사용자는 이 파일을 open(), read(), write() 과 같은 시스템 콜을 통해 접근하고 드라이버와 통신한다.

4. Sys class file (sysfs 인터페이스)

sysfs 는 커널 내부 정보를 사용자 공간에 노출시키는 가상 파일 시스템이며, GPIO 는 /sys/class/gpio 아래에서 관리된다. 예를들어 사용자는 echo 4 > /sys/class/gpio/export 명령을 통해 핀을 export 하고, 그에 따라 /sys/class/gpio/gpio4/ 디렉터리가 생성된다.

이후 direction 파일에 out 을, value 파일에 1 또는 0 을 쓰는 방식으로 방향 설정 및 출력 제어를 할 수 있다.

5. System Call

사용자 공간의 프로그램은 open(), read(), write(), close() 등의 시스템 콜을 통해 위 파일 인터페이스에 접근하고 요청을 커널로 전달한다. 커널은 요청을 수신하면 디바이스 드라이버를 호출하여 하드웨어를 제어한다. 시스템 콜은 사용자와 커널 사이의 안전한 인터페이스를 제공하며, 간접적으로 GPIO 를 제어할 수 있도록 구성된다.

이러한 과정을 통해 리눅스 시스템에서는 디바이스 드라이버와 파일 시스템, 시스템 콜을 조합하여 GPIO 를 안정적으로 제어할 수 있다.

참고자료

IT Trip. "C 언어로 임베디드 리눅스의 GPIO 제어하기: 초보자 가이드." IT Trip, https://ko.ittrip.xyz/c/embedded-linux-gpio-control. 접속일: 2025년 6월 1일.

The Linux Kernel. "GPIO Sysfs Interface for Userspace" https://www.kernel.org/doc/html/next/admin-guide/gpio/sysfs.html 접속일: 2025년 6월 1일.