IoT 환경에서 센싱 데이터 고속 처리를 위한 커널 수준 직접 경로

박준형*, 최종무, 전광일†
단국대학교 소프트웨어과, 한국산업기술대학교 컴퓨터공학과†
{google*, choijm}@dankook.ac.kr, gijeon@kpu.ac.kr†

Kernel-level Direct Path for Fast Processing of Sensing Data in IoT

Junhyung Park*, Jongmoo Choi, Gwangil Jeon†
Dankook university, Korea Polytechnic University†

요 약

IoT 환경에서 빠른 센싱 처리를 위한 커널 수준 직접 경로를 제안한다. 종래의 디바이스 입출력 제어는 유저 영역에서 처리하기 때문 유저 영역과 커널 영역 사이에서의 모드 스위치를 통해 System Call과 VFS를 거쳐서 디바이스 드라이버에 접근한다. 이 과정에서 발생하는 모드 스위치 과정에서 지연시간이 크게 발생하였다. 본 연구에서는 커널 수준의 직접 경로를 제안하여 IoT환경에서의 고속 센싱 처리 성능을 향상시킨다.

1. 서 론

임베디드 보드의 발전으로 다양한 전자제품이 연구, 개발되고 있다. 특히 스마트폰 혁신을 지나 사물과 통신하는 IoT(사물인터넷)를 주목하고 있다. IoT는 가트너 2016년 하이프 사이클 최고 정점에 있는 기술적 트랜드로 자동차, 집, 냉장고에 이르기까지 많은 사물에 인터넷 기능을 더한 IoT 제품이 개발되고 생산되고 있다.

본 연구는 이러한 IoT 환경에서의 센싱과 처리, 알림(Notify)의 고속 처리가 필요한 상황에서의 지연시간 절감을 목적으로 한다. 쿼드 콥터 드론에서의 균형유지, 화재감지, 가스누출 인식 등 센서 정보를 빠르게 읽고 처리해야 하는 경우에 용이하게 사용될 수 있다[1][2].

IoT 환경에서 커널 수준의 직접 경로를 형성하여 기존의 유저 영역에서의 처리에 비해 더 빠른 디바이스 제어를 가능하게 하는 *IoT 환경에서 빠른 센싱 처리를 위한 커널 수준 직접 경로*를 제안하고 실험한다.

기존의 디바이스 처리 방식은 GPIO, VFS, System Call과 같은 리눅스 인터페이스와 유저 API를 거쳐서 유저 영역에서 처리되기 때문에 고속으로 처리해야 하는 시스템에서 많은 지연시간이 발생했다.

그러나 커널 수준의 직접 경로를 이용하면 이러한 지연시간을 줄여 IoT 환경에 적절한 고속 센싱 처리가 가능하다. 본 논문에서는 커널 수준의 직접 경로를 활용한 고속 센싱 처리에 대해서 다룬다.

1장에서는 논문에서 다룰 내용과 그 기술적 배경 그리고 연구 전반에 대하여 설명하고 각 장을 설명한다.

2장에서는 기존의 IoT 임베디드 환경에서의 센싱 처리 방법과 그 과정 그리고 상세한 구현에 대하여 설명한다.

3장에서는 논문이 제안하는 센싱 처리를 위한 커널 수준의 직접 경로의 설계에 대해 각 부분별로 구체적으로 기술한다.

4장과 5장에서는 각각 실험 환경과 실험 결과에 대해 설명하고 6장에서 결론을 내리는 구성이다.

2. 디바이스 드라이버 처리

2.1 기존 디바이스 제어

일반적인 디바이스 제어는 크게 입력과정, 연산과정, 출력과정으로 구분할 수 있다. 입력과정은 임베디드 보드에 연결된 입력장치(센서)로부터 신호를 수신하는 과정이다. 연산과정은 입력받은 신호를 논리적 연산에 의해 처리하는 과정이다. 센서의 이원적 디지털 신호를 구분하거나 수신받은 아날로그 값을 임의의 임계값을 기준으로 처리 여부를 결정하거나 일련의 수학적 수식을 이용해 수신된 신호를 처리하는 과정이다. 출력과정은 연산된 결과에 의해 출력장치에 신호를 출력하는 과정이다[3].

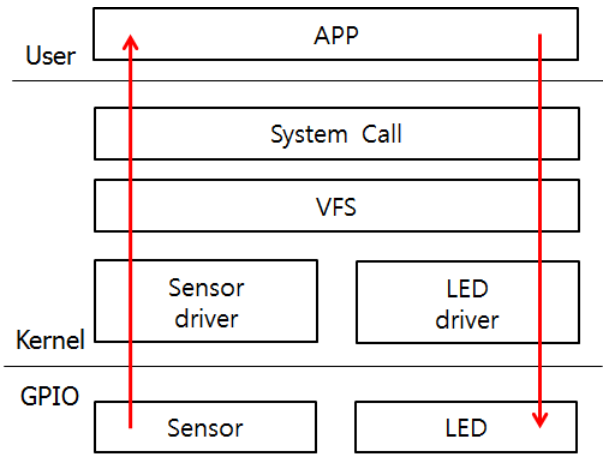
임베디드 환경에서의 일반적인 신호 처리 과정은 유저 영역에서 일어난다. 디바이스 제어의 일련의 과정들은 실재로 많은 단계를 거쳐 처리된다. 임베디드 보드가 하드웨어 신호를 수신하고 커널이 하드웨어 통신 인터페이스에 의해 신호를 읽고 각 디바이스 드라이버, VFS, System Call을 거쳐 유저 영역에 있는 유저 프로세스까지 도달하게 된다(그림1).

하드웨어 신호 입력, 인터럽트, 커널 영역의 디바이 드라이버, VFS, System Call 과정을 거쳐 Context Switch를 거쳐 유저 영역에 도달 하는데 많은 과정을 거치게 되어 많은 응답시간의 지연이 발생한다.

2.2 상세 구현 내용

유저 영역에서 작동하는 기존의 디바이스 제어 처리는 일반적으로 API를 이용한다. 이는 다시 리눅스 시스템이 제공하는 인터페이스인 /dev/gpio 또는 /sys/class/gpio 인터페이스를 이용한다.

API나 인터페이스를 통해서 디바이스에 해당하는 디바이스 파일에 대해 open(), read(), write() System Call을 호출하는데, 이 과정에서 커널 영역과 유저 영역간의 모드 스위치가 이루어진다. 여러 채널(센서)에서 신호를 읽고 처리하는 과정이 반복되는 시스템에서 이러한 문맥교환이 만들어낸 지연시간이 누적된다면 큰 비용적 손해가 발생할 수 있다.



(그림 1) 기존 디바이스 제어 계층

3. IoT 환경에서 빠른 센싱 처리를 위한 커널 수준 직접 경로

IoT 환경에서 빠른 센싱 처리를 위한 커널 수준 직접 경로는 본 논문에서 제안하는 방법으로 기존의 유저 영역에서의 디바이스 드라이버 접근과 제어에 수반되는 문맥교환 등의 절차적 과정을 간소화하기 위해 커널 영역에 디바이스를 제어 할 수 있는 커널 수준의 직접 경로이다(그림 1).

고속 센싱을 위한 커널 수준의 직접 경로의 바람직한 구현의 예는 직접경로 연결, 인터럽트 처리, 커널 스레드 동작 3단계로 구분할 수 있으며 상세한 내용은 다음과 같다.

3.1 직접경로 연결

고속 센싱을 위한 커널 수준의 직접 경로 설정을 위해 유저 영역에서 센서 디바이스 드라이버(입력)와 Notify 디바이스 드라이버(출력)를 연결하고 경우에 따라 선택적으로 콜백 함수를 등록하는 과정이 있다. 콜백 함수는 유저 영역에 있는 함수로(그림 2) 입력과 출력은 연결하기에 앞서 호출되어 입력정보를 기반으로 단순 비교 및 산술 연산 또는 수식을 이용한 필터 등으로 사용될 수 있다. 직접경로의 연결 정보는 커널 내부에 테이블 또는 리스트로 관리된다.

3.2 인터럽트 처리

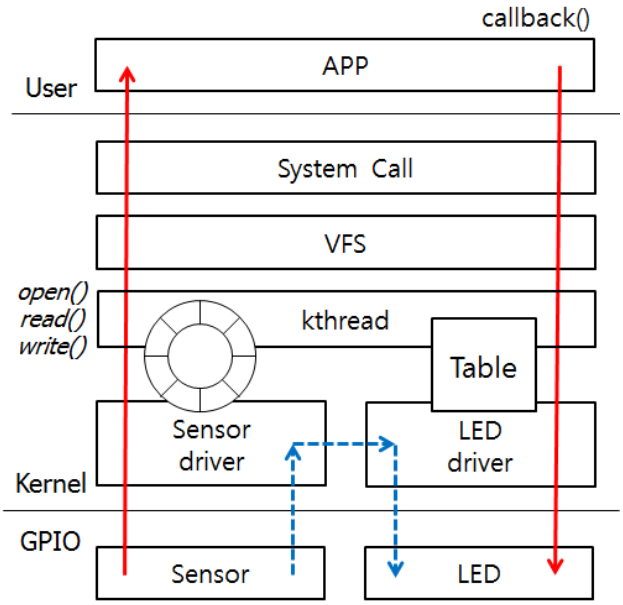
임베디드 보드의 GPIO로 입력되는 하드웨어 인터럽트를 신호 수신시 즉시 실행되는 인터럽트 핸들러에서 인터럽트 값을 링 버퍼에 쓴다. 링 버퍼는 센서(입력)정보를 저장하는 목적으로 사용된다.

인터럽트 핸들러의 명령이 많고 복잡하면 시스템 지연시간이 길어지고 전체 성능의 부정적 요인이 될 수 있으므로 링 버퍼를 이용한 짧은 쓰기만 수행하고 후반부 처리는 커널 쓰레드에 의해서 작동한다.

3.3 커널 쓰레드 동작

커널 쓰레드는 인터럽트 핸들러에서 처리하지 못하는 후반부 처리를 수행한다. 커널 쓰레드는 인터럽트 처리과정에서 링버퍼에 쓰여진 인터럽트 정보를 읽고 사전에 연결된 직접경로 정보를 기반으로 유저 영역의 콜백 함수를 커널 영역에서 수행하고 그 결과에 따라 출력(Notify)을 연결한다. 경우에 따라서 직접경로 연결 정보에 따라 하나의 센싱 입력에 따른 다수개의 출력(Notify)이 발생 할 수도 있다.

커널 쓰레드는 높은 우선순위로 스케줄링되고 커널에서 동작하기 때문에 모드 스위치 없이 빠르게 작동할 수 있다.



(그림 2) 제안된 디바이스 제어 계층

4. 실험

임베디드 보드를 이용해 입출력 회로를 구성하고 이를 읽고 처리하고 쓰는 작업에 대한 수행 속도를 실험한다. 일반 유저 영역에서의 유저 프로세스와 커널영역에 있는 커널 수준 직접 경로 이용한 두 경우를 측정하고 비교한다.

실재 실험은 3장에서 설명하는 커널수준의 직접경로 디자인의 전체에 대한 실험이 아니라 단편적으로 유저 영역의 App과 커널 영역의 제어 두 가지를 비교 한다.

4.1 실험환경

실험은 임베디드 보드로 널리 사용되는 라즈베리파이(Raspberry Pi)2 보드를 사용한다. Cortex7 900 MHz ARMv7 Quad Core를 사용하고, 1GB DDR2 450MHz 메모리를 갖는다. 40개의 GPIO를 사용하고 있어 확장성의 뛰어나고 많은 센서와 출력장치를 지원하고 있다.

실험에 사용된 OS는 리눅스 커널을 사용하는 경량화 OS인 OpenWrt를 사용했다.

실험에는 GPIO 인터페이스를 통한 입력, 출력 장치를 하나씩 두고 입력과 출력 그리고 그 사이에 입력 값을 비교하는 과정을 수행하는데 걸리는 시간을 비교한다.

User	clock_gettime() fd = open("/sys/class/gpio/gpioN...") read(fd, val, 2) if (val==0); wirte(fd, val, 2) close(fd) clock_gettime()
User Optimization	fd = open("/sys/class/gpio/gpioN...") clock_gettime() read(fd, val, 2) if (val==0); wirte(fd, val, 2) clock_gettime() close(fd)

Kernel	do_gettimeofday()
	
	val = gpio_get_value(N)
	If (val==0);
	gpio_set_value(M, val);
	
	do_gettimeofday()

(그림 3) 실험 방법

4.2 실험

각 실험의 정밀한 실행시간 측정을 위해 리눅스가 제공하는 API를 사용하였다.

유저 영역의 경우에는 <time.h> 라이브러리가 제공하는 구조체인 timespec를 이용해 clock_gettime()함수로 시간을 측정한다. 이 경우 나노(10^-9)초까지 시간 측정이 가능하다.

커널 영역의 경우 <linux/time.h>라이브러리가 제공하는 구초체인 timeval의 do_gettimeofday() 함수를 이용해서 시간을 측정하며 마이크로초(10^-6)초까지 시간 측정이 가능하다.

두 시간 측정 함수 API와 단위가 상이하고 최소 측정 단위가 제한됨에 따라 정확한 정량적 수치의 측정은 어려움이 있을 수 있으나, 반복적 실험과 개략적 추이와 비교 분석에 크게 여하한 차질 없이 실험을 진행 할 수 있었다.

그림 3과 같은 방법으로 센싱 처리를 1000회 반복하고 1회당 센싱 시간을 추정한다. 이는 커널이 측정할 수 있는 최소 단위를 고려해 신뢰할 수 있는 시간을 추정하기 위함이다.

실험은 센싱 처리에 대한 3가지 환경에 대하여 진행되었으며, 유저 영역에서의 API사용하는 경우, 유저 영역에서의 최적화된 API 사용하는 경우, 커널에서 직접 디바이스를 제어하는 경우에 대하여 실험하였다.

4.2.1 유저영역 : API

유저 영역에서 API를 통한 일반적인 GPIO 제어의 경우 [4] API가 호출 될 때마다 GPIO를 open(), read(), write() 과정을 모두 거치게 된다.

경우에 따라 더 많은 과정을 거칠 수 있지만 최소 3번 이상의 System Call을 거쳐 VFS을 거쳐서 디바이스를 제어하게 된다. (그림1의 System Call 과정)

실험에 사용된 OpenWrt는 /sys/class/gpio/ 인터페이스를 사용한다. 파일시스템 노드를 생성하고 입출력을 정의하는 초기 단계는 사전 작업으로 최초 1회만 발생하기 때문에 시간 측정에 생략하고, API호출에 의해 파일시스템 노드를 open(), read(), write(), close()하는 과정에 대한 시간을 측정한다.

4.2.2 유저영역 : 최적화

디바이스를 유저 영역에서 제어하는 경우, 상황에 따라서 단순한 API 함수의 경우 내부에서 open(), read(), write(), close()를 모두 수행할 수 있다. 그러나 연속적인 센싱 프로세스에서 불필요한 open()과 close()는 지연시간을 발생시킬 수 있다.

따라서 시스템 지연 시간을 발생시킬 수 있는 open()과 close()함수의 사용을 제외하고 실질적인 입출력 제어가 이루어지는 read(), write()를 반복적으로 수행하도록 하여 System Call과 모드 스위치를 최대한 줄인 경우에 대하여 소요되는 시간을 측정하는 실험을 한다.

4.2.3 커널영역

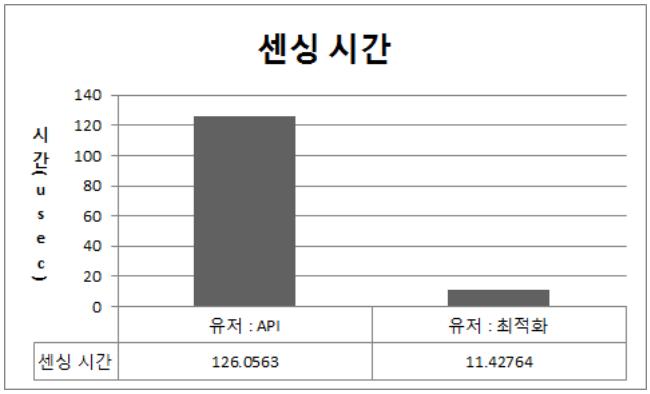
커널 영역에서는 커널 수준 직접 경로를 이용해 처리되는 시간을 측정한다. GPIO의 인터럽트를 등록한다. 인터럽트가 발생하고 인터럽트 핸들러가 작동하면 워크큐에 커널 쓰레드를 실행시키고 인터럽트 핸들러가 종료된다. 인터럽트 핸들러에 의해 워크큐에서 작동하는 커널 쓰레드에서 실질적인 GPIO 입출력 제어가 발생한다.

커널에서의 GPIO 제어는 커널에서 제공하는 GPIO 함수인 gpio_get_value()와 gpio_set_value() 인터페이스를 사용한다.

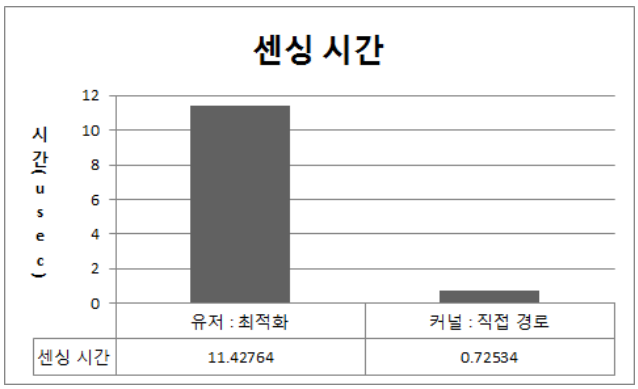
인터럽트 핸들러 등록과 커널 쓰레드 실행 워크큐 제어 등의 처리는 커널 모듈로 개발되어 리눅스 시스템에 로드하여 작동하였다. (그림 2의 점선)

상기 실험이 이루고자 하는 목적은 유저 영역에서 API를 사용하여 디바이스를 제어하는 경우, 유저 영역에서 최적화된 방법으로 API를 사용하는 경우, 커널 영역에서 직접 경로를 이용해 디바이스를 제어하는 경우에 3가지에 대하여 각각 소요되는 시간을 측정하고 이를 서로 비교하는 것이다.

5. 실험 결과



(a) 유저영역 비교



(b) 유저 커널 비교

(그림 4) 실험결과

그림 4는 각 실험 경우에 대한 측정 결과이다. 그래프에 표시된 값은 1회의 센싱 처리에 소요되는 시간이며 단위는 usec(10^-6 sec)이다.

그림 4의 a는 유저 영역에서 API를 사용하는 경우와, 유저 영역에서 불필요한 System Call을 줄여 최적화된 상태에서 실험한 시간이다.

유저 영역에서 최적화되지 않은 API를 사용하는 경우 센싱처리 1회에 약 126usec가 소요되고, System Call을 줄이는 방식으로 모드 스위치 발생을 줄인 최적화 한 경우 센서를 한 번 제어하는데 걸리는 시간 11usec로 그렇지 않은 경우에 비해 약 11빠른 센싱 처리가 가능했다.

그림 4의 b는 유저 영역에서 최적화된 API를 쓰는 경우와 커널 영역에서 직접 디바이스를 제어하는 경우를 비교한 그래프이다.

이 경우 커널 영역에서 직접 센싱처리 1회에 약 0.72usec가 소요되어 유저 영역에서 최적화된 API를 쓰는 경우보다 약 15배 빠른 센싱 처리가 가능했다.

6. 결론

IoT 환경에서의 고속화된 센싱 처리를 위해 커널 수준의 직접경로를 제안하고, 커널에서의 센싱처리 시간을 측정한 결과 최적화 여부에 따라서 크게는 약 150배, 적게는 15배의 성능 향상 효과를 볼 수 있었다.

3장에서 제안한 내용의 링 버퍼와 테이블 등 IoT 환경에서 고속 센싱 처리를 위한 커널 수준 직접 경로에 필요한 부가적인 기능을 모두 구현 했을 때 발생하는 오버헤드를 커널에서 저수준의 GPIO 제어로 지연시간을 줄이는 등의 추가적인 연구와 실험이 필요하다.

참 고 문 헌

[1] 박재열, 심재홍, 홍경택, “다중 센서 융합을 통한 화재 감시 임베디드 시스템”, 『한국정밀공학회 2013년도 춘계학술대회 논문집』 2013. pp.449-450

[2] 박진희, 박호준, 김성현, “이동형 재난관리를 위한 IoT 센서 플랫폼” 『한국정보과학회 2014 한국컴퓨터종합학술대회 논문집』 2014.6, pp.1228-1230

[3] 이태우, 조용환, 조지용, “커널 최적화를 이용한 임베디드 시스템의 성능 개선”, 『2011 춘계학술대회 논문집』 2011.5, pp.228-233

[4] http://wiringpi.com/

이 논문은 2015년 정부(미래창조과학부)의 재원으로 (재)스마트 IT융합 시스템 연구단(글로벌프런티어사업)의 지원을 받아 수행된 연구임((재)스마트 IT 융합시스템 연구단-2011-0031863)