리눅스 상에서 실시간성을 제공하기 위한 타이머 핸들러 모듈 설계 및 구현

The Design and Implementation of Timer Handler Module in Linux for Real-Time

이 승 훈, 송 창 인, 이 철 훈
충남대학교 컴퓨터공학과

Lee seung-hoon, Song chang-in, Lee cheol-hoon
Chungnam Univ.
Dept. of Computer Engineering,

요약

의료 기기, 항공 제어, 군 장비 등에서 쓰이는 임베디드 시스템은 특정한 목적만을 수행하기 위해 따로 설계된 시스템을 말한다. 이러한 임베디드 시스템에서는 정확한 시간을 예측하여 정해진 시간 안에 처리하기 위한 실시간성이 제공되어야 한다. 리눅스는 다수의 사용자가 이용하는 안정되고 검증된 운영체제로서 임베디드 시스템에서 많이 사용되며, 실시간성을 제공하기 위해 RTAI, RT-Linux를 이용한다. 하지만 RTAI는 Hard Real-time을 제공하지 못하는 문제점을 가지고 있고, RT-Linux의 경우 Hard Real-time은 제공하지만 많은 어셈 코드를 수정해야 하기 때문에 개발의 어려움이 있다. 또한 리눅스 커널에 실시간적 요소를 직접 추가하는 방법도 있지만 커널 코드를 수정할 때 마다 커널 컴파일을 해줘야 하는 문제점이 있다. 이에 따라 리눅스 상에서 실시간성을 제공해 주면서, 개발의 편의성을 제공 할 수 있는 방법이 필요하다. 본 논문에서는 실시간성을 제공하기 위한 타이머 핸들러 부분을 커널 코드로부터 분리하여 실시간성을 제공 해주며, 개발의 편의성을 제공할 수 있는 방법을 설계 및 구현하였다.

Ⅰ. 서론

최근 디지털 컨버전스(Digital Convergence) 시대가 가속화 되면서 다양한 기기와 서비스가 통합되었다. 이로 인해 임베디드 시스템은 과거에 하나의 기능만을 수행했던 것과는 달리 다양한 기능을 수행하게 되었다. 또한 산업, 가전, 사무, 군사 분야 등 여러 분야 활용이 되고 있는 임베디드 시스템은 실시간성 제공에 대한 요구가 늘어가고 있다. 리눅스는 다수의 사용자가 이용하는 안정되고 검증된 운영체제이기 때문에 임베디드 시스템에서 많이 사용되며 실시간성을 제공하기 위해 RTAI(Real Time Application Interface)와 RT-Linux(Real Time - Linux)를 이용한다. 하지만 RTAI의 경우 Hard Real-time을 제공하지 못하는 문제점이 있고, RT-Linux는 Hard Real-time은 제공하지만 많은 어셈블리 코드를 수정해야 하기 때문에 개발의 어려움이 있다. 또한 리눅스 커널에 실시간적 요소를 직접 추가하는 방법도 있지만 커널 코드를 수정할 때 마다 커널 컴파일을 해줘야 하는 문제점이 있다. 이에 따라 리눅스에서 실시간성을 제공해 주면서 개발의 편의성을 제공 할 수 있는 방법이 필요하다. 본 논문에서는 실시간성을 제공하기 위해 커널에 등록되어 있는 타이머 핸들러를 모듈화하여 실시간성을 제공해 주며, 타이머 핸들러를 수정할 시 다시 커널 컴파일을 하는 불편함을 개선하기 위한 타이머 핸들러 모듈을 설계 및 구현하였다.

Ⅱ. 관련연구

2.1 리눅스 커널 모듈

리눅스 모듈은 시스템이 부팅된 후 언제라도 커널에 동적으로 링크될 수 있는 코드의 집합이며, 모듈이 더 이상 필요하지 않을 때는 커널과의 연결을 해제하고 제거할 수 있다. 리눅스 커널의 상당수는 디바이스 드라이버와, 네트워크 드라이버나 파일시스템 같은 유사 디바이스 드라이버(Pseudo device driver)이다. 사용자는 insmod나 rmmod같은 명령으로 리눅스 커널 모듈을 명확하게 로드 또는 언로드를 할 수 있고, 커널 자신이 필요로 할 때 커널 데몬(kerneld)에게 모듈을 로드/언로드 할 것을 요구할 수 있다. 필요로 할 때 코드를 동적으로 로드 하는 것은 커널 크기를 최소화할 수 있다. 또한 모듈은 새로운 커널 코드를 다시 컴파일하고 커널을 재부팅하지 않고 테스트를 해보고자 할 때 유용하다. 로드된 리눅스 모듈은 다른 보통 커널 코드처럼 커널의 한 부분이 된다. 모듈은 커널 코드와 똑같은 권한과 책임을 진다. 다르게 말하면, 리눅스 커널 모듈은 모든 커널 코드나 디바이스 드라이버처럼 커널을 망가뜨릴 수도 있다는 것이다[1][2].

Ⅲ. 타이머 핸들러 모듈 설계 및 구현

3.1 타이머 핸들러 모듈 구현

Kprobes는 중단점을 실행 커널에 삽입할 수 있도록 하는 리눅스의 메커니즘이며, 이는 리눅스 커널을 후킹

하여, 동적으로 중단점을 삽입하는 것을 의미한다. 커널을 후킹하기 위해서는 커널 모듈을 만들어야 하며, 커널에 있는 함수들에 대해 알아야 한다[3].

타이머 핸들러 모듈을 구현하기 위해 커널 모듈을 생성하며, 커널 모듈과 Kprobe를 이용하여 커널을 후킹하기 위해 [그림 1]와 같이 후킹하려는 Local APIC 타이머 핸들러의 위치를 지정하여 핸들러의 주소에 중단점 삽입한다. 또한 후킹한 Local APIC 타이머 핸들러를 생성한 커널 모듈에서 동작시키기 위해 Kprobe 핸들러를 등록한다.

```
static struct kprobe kp = {
        .symbol_name = "ck_ns_exchange_timer_interrupt",
};
```

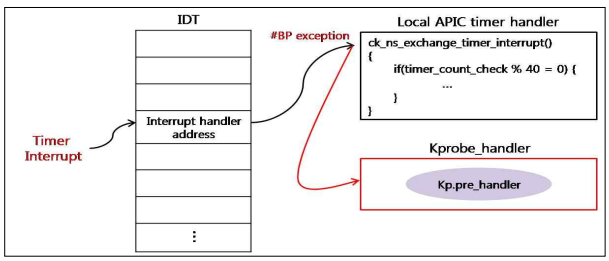
▶▶ 그림 1. 후킹하려는 핸들러 지정

타이머 인터럽트의 주기를 설정하고 동작시키기 위해 설정한 주기에 맞는 타이머 인터럽트 핸들러를 등록한다. 등록한 타이머 인터럽트 핸들러는 grep 명령어를 통하여 확인할 수 있으며, 설정한 주기로 동작시키기 위해 각 주기에 맞는 타이머 핸들러를 지정하여 설정한 주기로 동작하도록 한다.

Local APIC 타이머 핸들러를 후킹하여 커널 모듈에서 동작하는 타이머 핸들러는 사용자가 수정하여도 다시 커널 컴파일을 하지 않아도 된다. 이것은 타이머 핸들러가 커널 모듈에서 동작하기 때문에 커널 모듈만 컴파일하고, insmod 명령어를 사용하면 커널 모듈을 커널에 붙여서 동작 시킬 수 있다.

3.2 타이머 핸들러 모듈 동작 과정

타이머 핸들러 모듈은 [그림 2]와 같은 동작 과정을 거친다. 설정한 주기로 타이머 인터럽트가 발생 한다면 IDT(Interrupt Descriptor Table)에 등록되어 있는 타이머 핸들러의 주소를 찾게 되고, 해당하는 타이머 핸들러를 호출하게 된다. 이때 Kprobe를 이용하여 해당 타이머 인터럽트 핸들러에 breakpoint exception을 발생시키는 instruction을 삽입하고, Kprobe 핸들러를 등록하여 지정한 타이머 핸들러가 실행되기 직전에 exception이 발생되도록 하고 이 exception handler에서 등록된 Kprobe의 핸들러를 호출하는 방식으로 동작된다.



▶▶ 그림 2. Kprobe 동작 과정

Ⅳ. 실험 환경 및 결과

리눅스에 실시간성을 제공하기 위한 타이머 핸들러 모듈의 실험을 위해 [표 1]과 같은 환경에서 타이머 핸들러 모듈을 동작 시켰으며, 주기적인 핸들러의 동작을 확인하였다.

표 1. 실험 환경

실험 환경	
CPU	Intel(R) Pentium(R) 4 CPU 3.00GHz
OS	Ubuntu 8.04
Kernel ver.	linux-2.6.24
GCC ver.	gcc-3.3

[표 2]는 0.1ms의 주기로 동작하는 타이머 핸들러 모듈의 주기를 워크로드의 개수를 늘려가며 측정한 결과이다. 측정 결과에서 확인 할 수 있듯이 타이머 핸들러 모듈은 리눅스 스케줄링에 영향을 받지 않고 설정된 주기를 지키며 동작하며, 리눅스에 실시간성을 제공할 수 있다는 것을 보여준다.

표 2. 타이머 핸들러 모듈의 0.1ms 주기 실험결과

주기 측정 결과		
워크로드 개수	최소(ms)	최대(ms)
0 개	0.097808	0.102193
1 개	0.091048	0.108034
5 개	0.094028	0.107532

Ⅴ. 결론

리눅스에서 실시간성을 제공하기 위해 리눅스 커널에 타이머를 설정하고 타이머 핸들러를 등록하여 실시간성을 제공하지만 사용자가 핸들러를 수정하거나 약간 변경을 할 경우 커널 컴파일을 다시 해야 하는 불편함이 있다.

본 논문에서는 실시간성을 제공하기 위한 타이머 핸들러를 모듈 화하여 사용자가 타이머 핸들러를 수정하여도 다시 커널 컴파일을 하지 않고 타이머 핸들러가 있는 모듈만 컴파일 하여 모듈을 올리는 것으로 실시간성을 제공하며, 불편함을 개선했다. 하지만 타이머 핸들러 모듈의 경우 커널을 후킹하기 위해 만든 커널 모듈이기 때문에 사용자 API를 사용하여 타이머 핸들러를 작성할 수 없다. 그렇기 때문에 타이머 핸들러를 작성하기 위해서는 커널 API를 사용하여 타이머 핸들러를 작성해야한다. 일반적으로 개발은 사용자 API를 사용하기 때문에 커널 API에 대한 전반적인 이해가 부족하다. 이와 같은 이유로 타이머 핸들러를 이용하여 주기적인 작업을 생성하기 위해 커널 API를 분석하여 적용해야 하는 문제점이 있다. 향후 연구과제로는 사용자에게 익숙한 API를 사용하여 실시간 작업을 생성할 수 있도록 개선하는 것이며, 사용자 영역에서 실시간 작업이 보장되도록 하는 것이다.

▌참 고 문 헌▐

[1] Matthias Kalle Dalheimer, Lar Kaufman, Matt Welsh, Running Linux, pp. 760, O'Reilly, 1999
[2] RUBINI, Alessandro, Linux Device Driver, pp. 586, O'Reilly, 2001
[3] http://www.ibm.com/developerworks/library/l-kprobes.html
[4] 이귀영, Linux Kernel Programming, 글로벌, 2001