실시간성 향상을 위한 리눅스에서의 타이머 운용방안 분석

o
김 재 언, 김 영 호
부산대학교 전자계산학과
jekim@liso.cs.pusan.ac.kr, yhkim@hyowon.pusan.ac.kr

Analysis of the Timer Management Methods for Real-Time property Enhancement on Linux

Jae-eon Kim, Youngho Kim
Dept. of Computer Science, Pusan National University

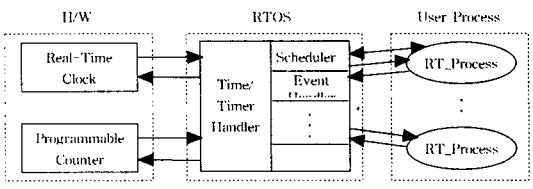
요 약

본 논문은 실시간 시스템에서 중요한 요소인 타이머 운용에 관련된 방법중 Linux를 위한 접근 방법을 분석한다. 현재 표준 PC는 ISA 버스와 연결된 8254 프로그래머블 카운터를 통해 시스템의 시각부분을 처리하고 있다. 이 경우 ISA 버스의 특성상 10us (in PentiumPro 200MHz) 정도 소요시간을 가지게 된다. 또한 여러 프로세스가 ISA 버스로의 접근을 위해 경쟁을 하게 될 경우 지연 시간은 더욱 커지며 예측이 불가능하게 된다. 보다 높은 microsec 수준의 실시간성을 요구하는 경우 이러한 근원적 시각처리 방안을 개선함은 필수적인 요소이다. 이를 위해 리눅스에 적용 가능한 타이머 운용방안 들로서 표준 8254 타이머, 펜티엄 프로세서의 TSC 이용방법 및 Intel SMP 보드의 APIC 타이머에 의한 방법들을 비교 분석하였다.

1. 서론

실시간 응용프로그램은 논리적 정확성뿐만 아니라 한정된 시간 제약조건에 대한 정확성을 가져야 한다. 시간적 제약조건을 만족하기 위하여 운영체제는 특정 시간에 응용프로그램이 요구하는 시스템 자원을 알 수 있어야 하며, 다른 프로세스로부터 실행권을 선점할 수도 있다.

타이머는 실시간 운영체제에서 중요한 요소이다. 대부분의 실시간 태스크들은 주어진 주기에 따라 반복수행을 하거나, 특정 시간 이후에 수행을 시작한다. 이 때 실시간 운영체제는 모든 태스크들이 각각의 시간 제약조건에 수렴할 수 있도록 시스템을 스케쥴링한다. 매 호출시에 스케쥴러는 특정시간에 수행되어야 하는 태스크가 있는지를 검사하기 위하여 시스템의 시각정보를 읽어들이게 된다. 일반적인 실시간 운영체제에서의 시각정보 접근은 [그림 1]과 같이 표현된다.



[그림 1] 실시간 운영체제에서의 시각정보 접근

실시간 운영체제에서 사용 가능한 타이머는 크게 두 가지 형태, 즉 변경되지 않는 고정된 주기로 발생하는 주기적 타이머 (periodic timer)와 일회성 타이머(oneshot timer)로 나눌 수 있다 [6]. 주기적 타이머는 시스템에 장착되어 있는 프로그래머블 카운터를 특정 주기로 분주시키고 그 값을 이용하여 주기를 생성하도록 되어있으며, 일회성 타이머는 보통 카운터 내 일회(oneshot) 모드를 이용하여 생성한다. 대부분의 실시간 프로세스는 주기적 타이머를 이용해 실행되며, 이 범주에 속하지 않는 경우 일회성 타이머를 이용한다. 실시간 운영체제에서 타이머 처리를 위해서는 시스템의 시각정보로의 접근이 반드시 필요한데, 이 접근시간을 줄임으로써 전체 시스템의 성능향상을 기대할 수 있다. 이것은 매 주기별로 시각정보에 접근하는 주기적 타이머는 물론, 주기가 동적으로 할당 가능한 일회성 타이머 모두에 성능향상을 보인다.

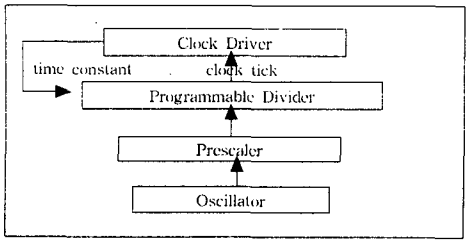
본 논문은 주기적, 일회성 타이머 들의 실시간성을 향상시킬 수 있는 보다 적은 접근시간을 가지는 타이머 운용 방법들을 비교 분석한다. 논문의 구성은 2장에서 일반 PC에서의 시각장치 구조와 리눅스에서의 시각운용 방법을 살펴본 후 실시간성 향상을 위한 타이머 운용 기법을 분석한다. 3장에서는 2장에서 분석한 기법을 실제 적용한 사례로 RT-Linux의 예를 살펴볼 것이며, 4장에서 결론 및 향후계획을 기술한다.

2. PC 에서의 타이머 운용

2.1 PC 시각장치구조

현재 운용되는 PC의 시각장치는 [그림 2]와 같이 네 개의 요소

로 구성된다 [8].



[그림 2] 일반적 시각 장치 모델

Oscillator는 시각장치의 시각 유지를 위한 기본 주파수를 발생시키며, 기본 시각 단위를 생성하기 위해 분주된다. Prescaler에서 분주된 주파수는 프로그래머블 Divider에 의해 다시 한번 분주된다. 프로그래머블 Divider는 Clock 카운터와 시정수를 가진 Register로 구성되며, Clock Driver에서 사용하기 위한 기본 시간 단위인 Clock Tick을 발생시킨다. 현재 사용되는 표준 PC는 8254를 프로그래머블 Clock Divider로 사용하며 AT 표준인 ISA 버스를 통해 연결되어 있다. 8254는 3개의 독립적인 16-bit 카운터로 구성되어 있으며 최대 10MHz, 즉 최대 resolution이 1microsecond인 입력들에 대해 동작한다.

Prescaler에 의해 분주된 주파수는 Clock 카운터에 입력되어 역계수(Down Counting)되며, 계수값이 0이 되는 순간 인터럽트가 발생한다. Clock Tick은 이 인터럽트에 의해 전달되며, 시스템 버스를 통해 Clock Driver로 전달된다. 그 후, 계수값은 시정수로 다시 자동 설정되며, 역계수가 계속된다. Clock Driver는 전달받은 Clock Tick의 주기를 시스템의 기본 시간 단위로 하며, 이것으로 시스템의 시간을 유지한다.

어떤 응용프로그램이 시스템에 시간 정보를 요구하게 되면, Clock Driver가 유지하고 있는 Clock Tick 계수를 시간으로 변환하여 제공한다. 이렇게 제공받는 시간은 시스템 시작으로부터 상대적인 시간이며, 실제로 시스템이 시작한 시간을 더하여 제공한다. 시스템 시작 시간은, 시스템의 동작이 멈춘 상태 즉, 외부 전원이 차단된 상태에서 독립적으로 시간을 유지하는 Real-Time Clock(RTC 혹은 BIOS 타이머)에 의해 얻을 수 있다. 이 시각 장치는 자체 전원을 통해 시간을 유지하며, 시스템이 시작한 이후에는 사용되지 않는다.

2.2 리눅스에서의 시각운용

리눅스는 시스템이 시작되면 시스템의 시간값과 프로그래머블 카운터인 8254를 초기화 한 뒤 RTC로부터 시스템 시작 시간을 읽어온다. 리눅스에서 8254는 ISA 버스에 연결되어 있지만 처리속도의 문제 때문에 Character device로 접근하지 않고 직접 접근하게 된다. 이후 시스템에서 유지되는 값은 8254가 초기화된 시점에서의 값에 대한 time tick의 상대적인 값이며 RTC로부터 읽어 들인 시스템 시작 시간과 더해져 하나의 시간을 표현한다. 현재 리눅스에서 타이머는 8254와 다음 장에서 설명하게 될 TSC를 이용한 방법을 프로세서 검출 코드를 통해 선택적으로 사용한다.

2.3 실시간성 향상을 위한 타이머 운용 기법

시간정보에 대한 접근시간을 줄이는 것은 운영체제의 실시간성을 향상시키는 중요한 요인이다. 아래에서 리눅스에서의 시간정보에 대한 접근시간을 줄일 수 있는 기법들을 비교 분석한다.

시스템에서 현재 시간을 읽어오는 데 소요되는 시간은 다음과 같은 수식으로 표현될 수 있다.

$$T_a = T_d + T_r + T_t + T_c + T_{dr}$$

T_a : Total access time
T_d : Device access time
T_r : Read operation execution time
T_t : Transmission time
T_c : Offset calculation time
T_{dr} : Device response time

일반적으로 같은 시스템 상에서 서로 다른 방법으로 시간 정보에 접근하는 경우의 비교에 있어서, 이전의 값과 새로 구해진 두 값의 단순 차감 연산시간인 T_c 는 동일한 프로세서의 동일 연산을 이용하므로 거의 같다고 할 수 있다. 또한 T_{dr} 는 ns 수준의 값으로 대상 범주가 ms단위인 다른 값들에 비해 무시할 수 있을 만한 수치이므로 비교대상에서 제외된다. 그러므로 실제 비교대상의 요소로 T_d , T_r , T_t 만을 선택한다.

2.3.1 8254

8254는 현재 사용되는 표준 PC에 장착된 프로그래머블 카운터이며, AT의 표준인 ISA 버스에 연결되어 있다. 모든 PC에서 사용 가능하다는 장점이 있지만 ISA 버스자체의 한계로 인해 상대적으로 큰 T_d 와 T_t 값을 가지게 되며, 프로세스간 경쟁에 의해 예측 불가능한 지연이 발생할 수 있다. 실제 PentiumPro 200MHz의 시스템에서 전체 접근 시간은 10us으로 나타났다 [6]. 이러한 접근시간은 높은 실시간성을 요구하는 경우 적용하기에 충분치 않으며 개선을 위한 기본값으로 간주될 수 있다.

2.3.2 TSC(Time Stamp Counter)

TSC는 Intel 펜티엄 이상의 CPU에 포함되어 있는 64bit 카운터이다. Processor가 Power-on이나 초기화된 경우에 0으로 설정되며 매 CPU clock마다 1씩 값이 증가한다. CPU 내부에 있으므로, 별도의 디바이스 접근시간과 디바이스로부터의 전송시간이 없다. 따라서 TSC에서의 접근시간은

$$T_a \simeq T_r + T_c$$

로 간주할 수 있다. 앞에서 언급했듯이 T_c 가 동일하다고 볼 때, 결국 전체 접근시간은 T_r 과 같게 된다. 그러므로 TSC를 사용할 수 있는 펜티엄 이상의 시스템에서는 TSC를 이용하는 것이 가장 최적이라고 할 수 있다. 하지만 APM mode에서 disk의 suspend가 일어나게 되면 TSC가 다시 재설정되는 경우나 Intel 복제 프로세서(AMD, Cyrix)에서의 호환성 문제와 같은 구현상의 문제점에 대한 개선이 요구된다.

2.3.3 APIC 타이머

APIC(Advanced Programmable Interrupt Controller) 타이머는 Intel의 대칭형 멀티 프로세서(SMP : Symmetric Multi

Processor) 보드 상에 존재하는 타이머이며, 2.3.1에서 설명한 8254와 동일한 3개의 카운터로 구성되어 있으나, 시스템 버스 대신 SIO/SIO.A에 연결되어 있다. 접근 시간은 CPU내의 TSC 값을 읽는 것보다는 느리지만, 시스템 버스를 통한 접근 시간이 없기 때문에 ISA나 PCI를 통한 접근보다는 충분히 빠르다. 모든 SMP 보드에 적용 가능하며, 몇몇 특정 보드를 제외한 거의 대부분의 보드에서 하나의 CPU만 장착하더라도 동작이 가능하다는 보고가 있다 [7]. APIC 타이머를 이용할 경우 프로세서의 종류나 다른 코드와의 문제가 발생하여 TSC를 사용할 수 없는 경우에 TSC의 사용과 비슷한 수준의 성능을 제공해 줄 수 있다.

2.3.4 Specialized External Hardware

시스템의 타이머 운용을 위해 외부장치를 추가하는 경우 고정밀의 시각정보를 가질 수 있고, 프로세서의 종류나 보드의 종류에 상관없이 모든 경우에 적용이 가능하나 시스템 버스에 대한 의존성 때문에 상대적으로 큰 T_d , T_l 값을 가진다. 결국 특수목적의 외부 장치를 사용하더라도 연결된 시스템 버스의 성능에 따라 전체 시간이 결정된다.

리눅스에서의 실시간성 향상을 위한 타이머 운용 방안들을 정리해 보면 다음과 같다.

	TSC	APIC Timer	External H/W
적용 대상	Pentium이상	SMP 시스템	모든 경우에 가능
장점	가장 적은 접근시간	Linux 내부의 다른 코드와의 충돌이 없이 적용가능	고정밀의 시각정보 획득 가능
단점	APM 모드에 의한 재설정 문제	SMP 보드에서만 적용가능	버스 interface의 성능에 의해 접근시간이 결정
Access time	2-3 us	5-6 us	6-9 us (66MHz PCI의 경우)

[표 1] Timer Handling 운용방법 비교

3. RT-Linux

본 장에서는 앞서 논의한 리눅스의 타이머 운용방안들에 대한 RT-Linux에서의 구현 예를 살펴본다. RT-Linux는 New Mexico Institute of Mining and Technology의 Victor Yodaiken과 Michael Barabanov에 의해 개발되었으며 소프트 인터럽트 에뮬레이션, 정적 우선순위 기반 선점형 스케쥴러, 커널 모듈 모델, 소프트웨어 문맥전환 등의 특징을 가지는 리눅스의 Hard real-time 확장커널이다.

RT-Linux는 주기적 타이머에는 8254를 사용하고, 일회성 타이머에는 TSC를 사용하여 구현하였다. 실제 PentiumPro 200 MHz 시스템상에서의 TSC 접근은 3us의 시간이 소요됨을 보이고 있다 [6]. RT-Linux는 리눅스와 마찬가지로 프로세서 검출 코드를 이용하여 8254와 TSC를 선택적으로 사용하며, 현재 SMP 보드의 APIC 타이머를 사용하는 방법이 논의중이며 실험용 코드가 만들어지고 있다.

RT-Linux상에서 부가장치인 3C905b에 내장된 16bit 카운터를 320ns으로 작동시킨 실험용 코드도 있었다. 이 카운터는 packet의 buffering을 위한 것이었는데 PCI 버스를 통한 access와 자체 card와의 충돌로 만족할 만한 실험 결과를 얻지는 못했다.

4. 결론 및 향후계획

지금까지 Linux에서의 실시간성 향상을 위한 타이머 운용방안을 분석해 보았다. 이를 통해 Linux에서 실시간성 향상을 위해서는 시각정보에 대한 접근 시간을 줄여야 하며, 사용될 수 있는 카운터는 다음과 같은 특성을 지녀야 한다는 것을 알 수 있었다.

- 현실적인 한계치내에서 오버플로우 없이 지속적인 count가 가능해야한다.
- 타이머가 프로그램 가능한 특정 값에 도달했을 경우 인터럽트를 발생시킬 수 있어야 한다.
- 최소 ISA 버스 이상의 성능을 가지며 연결되어야 한다.

펜티엄 계열의 프로세서가 주종을 이루는 현재의 운용환경으로 볼 때 TSC를 이용한 타이머 운용방법은 부가적인 장치요소의 필요없이 응용프로그램이 요구하는 상당한 수준의 실시간성을 만족할 수 있을 것으로 보인다. 그리고 APIC 타이머를 이용한 방법을 선택적으로 사용 가능하게 하여 TSC 사용이 불가능한 시스템에서도 TSC의 사용과 비슷한 수준의 실시간성을 만족할 수 있을 것이다.

향후 본 논문에서 보여진 APIC-Timer를 이용한 code를 구현 및 검증할 것이며, 표준 리눅스에서 정의되지 않은 Timing constant (deadline, worst-case execution time 등)를 적용하여 Linux의 실시간성을 높일 수 있는 방안을 병행해 연구할 것이다.

참고문헌

[1] Michael Beck, Harald Bohme, et al. Linux Kernel Internals. Addison-Wesley, 1996

[2] David A Rusling, The Linux Kernel, http://metalab.unc.edu/pub/Linux/docs/LDP/linux-kernel/

[3] Yu-Chung Wang and Kwei-Jay Lin, Enhancing the Real-Time Capability of the Linux, University of California, 1998

[4] Balaji Srinivasan, Shyamalan Pather, et al. A Firm Real-Time System Implementation Using Commercial Off-The-Shelf Hardware and Free Software, University of Kansas, 1998

[5] Michael Barabanov, A Linux-based Real-Time Operating System. Master's thesis, New Mexico Institute of Mining and Technology, 1997

[6] Victor Yodaiken and Michael Barabonov, A Real-Time Linux, http://luz.cs.nmt.edu/~rtlinux/

[7] E. Bianchi, L. Dozio and P. Mantegazza, A Hard Real Time support for LINUX, http://server.aero.polimi.it/projects/rtai/

[8] 이성진, GPS 시각원과 FLL을 사용한 시각 정밀도 향상 기법, 부산대학교 전자계산학과, 1998