실시간 운영체제를 위한 2단계 인터럽트 서비스 루틴의 설계 및 구현*

이재규°, 조문행, 정명조, 이철훈
충남대학교 컴퓨터공학과
{jklee°, mhcho, mjjung, chlee}@ce.cnu.ac.kr

Design and Implementation of two phase Interrupt Service Routine for Real-Time Operating Systems

Jae-Gyu Lee°, Moon-Haeng Cho, Myoung-Jo Jung, Cheol-Hoon Lee
Dept. of Computer Engineering, Chungnam National Univ.

요 약

실시간 운영 체제(Real-Time Operating Systems)는 시스템 동작이 논리적 정확성뿐만 아니라 시간적 정확성에도 좌우되는 운영 체제이다. 시간 결정성(Time Determinism)을 지키기 위해서는 시스템에서 발생하는 모든 사건에 대해서 예측이 가능해야 한다. 시스템에서 발생하는 사건은 인터럽트에 의해 처리되는데, 이것은 비동기적인 사건의 발생을 CPU에게 알리는 하드웨어 메커니즘으로서 인터럽트 서비스 루틴(Interrupt Service Routine : ISR)을 통해서 인터럽트에서 처리해야 할 부분을 수행한다. 본 논문에서는 인터럽트의 처리를 인터럽트의 인지와 처리로 분리할 수 있도록 LISR과 HISR을 설계 및 구현하였다.

1. 서 론

인터럽트는 외부 또는 내부적인 사건에 대한 즉각적인 반응을 제공하는 메커니즘이다. 인터럽트가 발생하면 프로세서는 현재 하던 일을 멈추고 적당한 ISR로 제어를 넘긴다. 이러한 인터럽트는 꼭 필요한 것이기는 하지만 실시간 운영 체제에서 여러 가지 문제점들을 야기시킬 수 있다. 가장 큰 문제점은 ISR에서 커널의 서비스들을 접근해야 할 필요성이 있다는 점이다. 즉, ISR에 의한 커널의 서비스들이 접근하는 데이터 구조들에 대한 보호가 필요하다는 것이다. 이 문제의 가장 간단한 해결책은 ISR을 처리하는 동안 인터럽트를 disable 시키는 것이다. 하지만 이러한 방법은 인터럽트에 대해서 최대한 빠른 응답 시간을 요구하는 실시간 운영 체제의 기본 방침에 위배 된다. 따라서 내부 데이터 구조들을 보호하기 위해서 인터럽트를 disable 하는 것은 바람직하지 않다. 또 다른 문제점으로 생각해 볼 수 있는 것은 ISR의 길이가 길어지면 인터럽트 응답 시간이 증가해서 다른 프로세스들의 작업에 방해요인이 된다는 것이다.

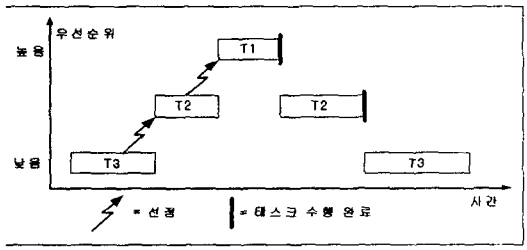
위에서 언급한 두 가지 문제점을 해결하기 위한 방안으로 본 연구팀에서는 선점형 스케줄링 방식을 지원하는 실시간 운영체제 iRTOS™에서의 ISR을 LISR(Low-level Interrupt Service Routine)과 HISR(High-Level Interrupt Service Routine)으로 구분해서 설계하고 구현하였다.

본 논문은 2장에서 관련연구로서 실시간 운영체제의 선점형 스케줄링과 인터럽트에 대하여 설명하고, 3장에서는 LISR과 HISR의 설계 및 구현을 설명하였으며, 4장에서는 테스트 환경 및 결과를 보이고, 5장에서는 결론 및 향후 연구 과제를 기술한다.

2. 관련연구

2.1 선점형 스케줄링 (Preemptive Scheduling)

실시간 운영 체제는 선점형과 비선점형 또는 복합된 형태의 스케줄링을 제공하는데, 본 연구는 선점형 스케줄링을 제공하는 실시간 운영체제를 기반으로 진행하였다. 선점형 스케줄링은 프로세스의 우선순위에 기반 하여 가장 높은 우선순위를 가지는 실행 준비상태의 프로세스에게 CPU 사용권을 부여하는 방법으로 응답성이 중요시되는 실시간 시스템에 적합하다. [그림 1] 에서와 같이 현재 CPU를 점유하고 있는 프로세스가 실행준비 상태가 되면 CPU 사용권은 해당 프로세스에게 넘어간다.

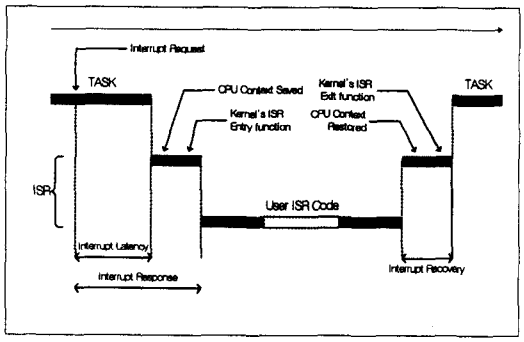


[그림 1] 선점형 스케줄링

*본 연구는 산업자원부의 지역전략산업 석, 박사 연구인력 양성사업의 지원으로 수행된 것임.

2.2 인터럽트

일반적으로 CPU는 FIQ(Fast Interrupt Request)와 IRQ(Interrupt Request) 두 가지의 인터럽트 모드를 지원한다. IRQ는 일반적인 인터럽트로 중첩(Nesting)이 가능하며, FIQ는 즉시 처리되어야 하는 인터럽트로서 중첩되지 않도록 하며 중요한 몇 개의 인터럽트만 등록해서 사용한다. 인터럽트는 CPU의 특정 레지스터에 정해진 값을 쓰거나 상태 레지스터를 통해서 부분 또는 전체 인터럽트를 disable 또는 enable 할 수 있다. enable상태에서 인터럽트가 발생하면 CPU는 인터럽트 모드로 전환하게 된다. 이때 운영 체제는 현재 수행 중이던 프로세스의 컨텍스트를 저장하고 발생한 인터럽트에 해당하는 ISR을 실행한다. ISR의 실행이 종료되면 선점형 실시간 운영 체제는 스케줄링을 통해서 가장 높은 우선순위를 가지는 프로세스를 선택하여 컨텍스트를 복원해서 인터럽트의 처리를 마치게 된다. [그림 2]는 선점형 실시간 운영 체제에서의 Interrupt Latency, Response, Recovery Time을 도시한 것이다.



[그림 2] Interrupt Latency, Response, Recovery Time

3. LISR / HISR의 설계 및 구현

3.1 LISR (Low-level Interrupt Service Routine)

LISR은 현재 태스크의 스택을 사용하는 일반적인 ISR을 수행한다. iRTOS™는 LISR을 호출하기 전에 컨텍스트를 저장하고 LISR로부터 복귀한 후에 컨텍스트를 복원하도록 되어 있기 때문에 LISR은 C언어로 작성할 수 있고 다른 C 루틴도 호출 가능하다. LISR에서는 인터럽트가 disable된 상태가 되므로 가능한 빠른 응답을 요구되는 부분만을 작성하고 코드의 길이도 짧게 작성해야 한다. LISR에서는 단지 몇몇의 커널 서비스만이 가능하기 때문에 인터럽트의 발생을 알리고 HISR을 활성화 시켜서 커널의 전체적인 서비스를 받을 수 있도록 하였다.

3.2 HISR (High-level Interrupt Service Routine)

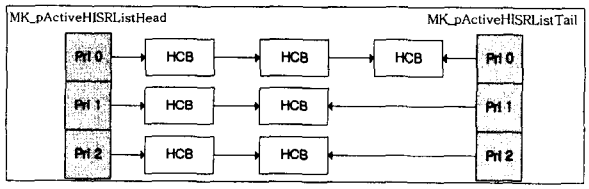
HISR은 동적으로 생성되고 삭제될 수 있다. 즉, 각 HISR은 자신의 스택을 가지며 태스크처럼 고유의 제어 블록을 가진다. 스케줄링이 가능하게 하기 위해서 태스크의 구조를 가지며 태스크 보다 높은 우선순위를 가지도록 하였다. HISR에 0부터 2의 우선순위를 부여해서 활성화된 HISR이 있으면 항상 태스크보다 먼저 CPU를 점유하게 된다. 동일한 우선순위의 HISR에 대해서는 라운드 로빈 방식의 스케줄링을 적용해서 기아상태가 되는 것을 방지 하였다. HISR은 각각 자신의 스택과 제어 블록을 가지기 때문에 이미 접근되고 있는 커널의 데이터 구조를 접근하려고 하면 일시적으로 block될 수 있다. 이러한 메커니즘으로 커널의 데이터 구조를 보호 할 수 있다.

3.2.1 HISR 리스트

생성된 HISR은 이중 순환 연결 리스트로 구성하였으며 새로 생성된 HISR은 리스트의 끝에 삽입되며 삭제 시에는 리스트에서 완전히 제거 된다.

3.2.2 Active HISR 리스트

활성화된 HISR은 각 우선순위 별로 생성된 단일 연결 리스트로 구성하여 동일한 우선순위의 HISR을 관리할 수 있게 하였다. [그림 3]



[그림 3] Active HISR 리스트

3.2.3 HISR 제어 블록

HISR의 제어 블록은 [표 1]과 같다.

[표 1] HISR 제어 블록

t_Magic	HISR을 구별하는 식별자
t_Name	HISR의 이름
t_Priority	HISR의 우선순위
t_CurrentOfStack	현재 모드의 스택 포인터
t_TopOfStack	스택의 Top 포인터
t_BottomOfStack	스택의 Bottom 포인터
t_PreviousCurrentOfStack	이전 모드의 스택 포인터
t_LengthOfStackArea	스택 영역의 길이
t_Function	HISR의 실행 함수
t_ActivateCount	HISR이 활성화된 횟수
t_pHISRNext	다음 HISR을 가리키는 포인터
t_pHISRPrev	이전 HISR을 가리키는 포인터
t_pHISRActiveNext	다음 활성화된 HISR을 가리키는 포인터

3.2.4 HISR 생성과 삭제

[표 2]는 HISR을 생성하고 삭제하는 함수를 보여주고 있다.

[표 2] HISR 생성, 삭제 API

CreateHISR()	HISR을 생성하는 함수
DeleteHISR()	HISR을 삭제하는 함수

*HISR을 생성하기 위해서 CreateHISR()*함수를 사용하고, 함수의 인자로는 HISR의 제어 블록, HISR 이름, HISR의 실행 함수 포인터, HISR의 우선순위, 스택 포인터 및 스택 크기이다. 생성된 HISR은 전체 HISR 리스트에 삽입하고 스케줄링 되지는 않는다. 이 HISR은 LISR에서 Activate HISR() 함수를 호출해서 활성화 되고난 후 다음 스케줄링이 일어나게 되면 활성화된 횟수만큼 HISR 실행 함수를 수행하도록 하였다.

3.2.5 HISR 제어

[표 3]은 HISR을 제어하는 함수들을 보여준다.

[표 3] HISR 제어 API

ActivateHISR()	HISR을 활성화 시키는 함수
GetCurrentHISR()	현재 실행중인 HISR을 얻는 함수

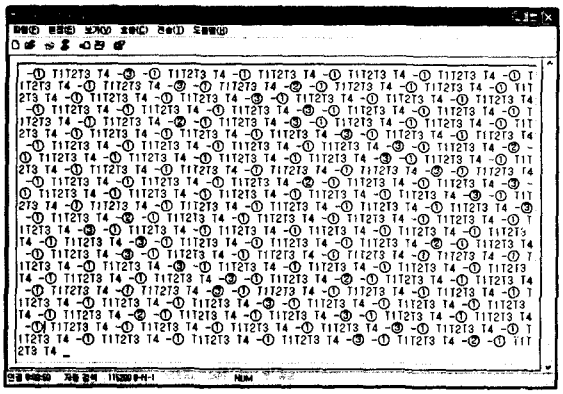
HISR을 활성화 시키기 위해서 ActivateHISR()함수를 사용하며, 함수의 인자는 HISR의 제어 블록이다. 활성화된 HISR의 우선순위는 다른 어떤 태스크 보다 우선순위가 높기 때문에 다음 스케줄링이 일어나게 되면 HISR중에서 가장 우선순위가 높은 HISR이 수행되게 된다.

3.3 HISR 스케줄링

스케줄 함수에서는 태스크를 스케줄 하기 이전에 0부터 2의 우선순위를 가지는 활성화된 HISR이 있는지 확인하고 해당하는 우선순위의 *Active HISR* 리스트에서 HISR을 가져와서 현재의 태스크 또는 HISR과 컨텍스트 스위치를 한다. 컨텍스트 스위치를 할때 활성화된 HISR이 있으면 항상 태스크보다 먼저 CPU를 점유하게 하고 동일한 우선순위의 HISR에 대해서는 라운드 로빈 방식의 스케줄링을 적용해서 기아상태가 되는 것을 방지 하였다.

4. iRTOS™에서의 테스트 및 결과

본 논문에서 구현한 ISR을 ARM-920T CPU가 탑재된 S3C2400 보드에서 동작하는 iRTOS™에 적용시켜서 테스트해 보았다. [그림 4]는 태스크와 HISR이 스케줄링 되는 결과이다. 3개의 타이머 HISR을 각각 10, 255, 30 클럭 주기로 활성화 하고 테스트 하였다. 화면에서 ①, ②, ③은 타이머 인터럽트를 나타내며 T1, T2, T3, T4는 시스템에서 동작중인 프로세스이다.



[그림 4] 테스트 프로그램 실행 결과

5. 결론 및 향후 연구 과제

본 논문에서는 실시간 운영체제에서 인터럽트를 처리하기 위한 ISR을 LISR과 HISR로 나누어서 HISR이 스케줄링이 가능하게 구현함으로써 인터럽트가 disable 되는 시간을 최소화 하였고, HISR 각각이 자신의 데이터 구조를 갖게 되어 커널의 데이터 구조에 대한 보호도 가능하게 하였다. 또한 ISR의 길이가 길어지게 되어 시스템의 반응성이 느려지게 되는 문제도 어느 정도 해결 할 수 있었다. 향후 테스트를 더 거쳐서 성능의 향상과 안정화를 이루어야 할 것이다.

참고문헌

[1] http://www.inestech.com

[2] Jean, J. Labrosse, "uC/OS-II The Real-Time Kernel", R&D BOOKS, 1999

[3] David Stepner 외 2명, "Embedded Application Design Using a Real-Time OS", IEEE 1999

[4] Craig Hollabaugh, "Embedded Linux: Hardware, Software, and Interfacing", Addison-Wesley, 2002

[5] http://www.acceleratedtechnology.com