논문 01-02-06

경성 내장형 실시간 시스템의 설계 및 구현

Design and Implementation of Hard Embedded Real-Time System

印 致 虎*

(Chi-Ho Lin*)

요 약

본 논문은 독립적으로 움직이면서 시간 제약을 만족시키는 새로운 내장형 실시간 시스템을 설계하며 구현하였다.

본 논문에서의 실시간 시스템 커널은 시간적인 요소를 가장 핵심으로 고려하여 설계되었다. 따라서 실시간 커널은 작은 용량을 가지며 빠르게 예측하며, 실시간 스케쥴링에 요구되는 많은 변화들을 통해서 실시간 커널에 융통성을 부여한다. 제안한 실시간 커널은 경성 실시간 제약 조건인 인터럽트 지연 시간, 스케줄링의 정확성, 메시지 전달시간을 만족하기 위하여 실시간 커널에는 실시간 태스크 처리와 인터럽트 처리, 타이밍을 처리하도록 하였고 비 실시간 커널은 일반적인 태스크를 처리하도록 하였다.

제안된 실시간 시스템은 RT-Linux, QNX와 인터럽트 지연, 스케줄링 정확성, 메시지 전달시간 등을 비교 분석하여 효율성을 입증하였다.

Abstract

In this paper, we have designed and implemented a new hard embedded real-time system to satisfy hard real-time constraints in moving independently.

Real-time kernel should be small size, fast and predictable. Because of the great variety of demands on real time scheduling, a real time kernel should also include a flexible and re-programmable task scheduling discipline. In this paper, we present that real-time applications should be split into small and simple parts with hard real-time constraints. To satisfy these properties, we designed real-time kernel and general kernel, that have their different properties. In real-time tasks, interrupt processing should be run. In general kernel, non real time tasks or general tasks are run.

The efficiency of the proposed hard embedded real-time system is shown by comparison results for performance of the proposal real time kernel with both RT-Linux and QNX.

Key words : Kernel, Embedded, Real-time, Scheduling, Constraints

I. 서 론

실시간 운영체제(Real-Time Operating System, RTOS)는 지정된 시간 제한 내에 확실한 출력을 보장하는 운영체제이다. 예를 들어, 어떤 객체가 조립라인

* 世明大學校 컴퓨터 科學科

(Dept. of Computer Science, Semyung University)

接受日: 2001年 8月13日, 修正完了日: 2001年11月15日

상의 로봇에 이용될 수 있게 보장하도록 운영체제를 설계할 수 있다. 실시간 시스템은 경성 실시간(hard real-time)과 연성 실시간(soft real-time)으로 보통 구분된다. 이벤트를 가끔 놓쳐도 크게 문제가 되지 않는 시스템을 연성 실시간이라고 한다. 반면 경성 실시간은 이벤트에 대한 반응성이 빠르고, 중요한 이벤트가 덜 중요한 이벤트보다 먼저 수행되며, 이벤트를 놓치는 일이 결코 일어나서는 안 되는 시스템을 말한다. 이를 보장하기 위한 중요한 요소로 인터럽트가 발생했을 때 빠른 시간에 인터럽트 핸들러를 불러주는 것과 중요한 태스크가 있으면 다른 태스크를 제치고 이를 먼저 처리하는 것이다.[1-2] [12]

실제 사용되고 있는 대부분의 실시간 커널은 제어 시스템에서 동작하는 실행체제의 형태이다. 이러한 특성을 지닌 실시간 시스템은 일반적으로 하나의 응용에 전용되어 사용된다. 따라서 제어시스템에서 동작하는 실시간 커널은 작은 크기의 운영체제로써 파일 시스템과 같은 구성을 제외한 태스크 관리, 태스크간의 통신 태스크간의 동기화, 인터럽트 처리와 같은 기본적인 운영체제의 기능만을 지니고 있는 소형 운영체제로써, 시스템의 작업을 수행할 뿐만 아니라 외부의 요구도 처리할 수 있어야 한다. 기존의 시분할 시스템은 시스템의 설계 시 시스템의 전체 성능 향상과 빠른 평균 응답시간, 자원의 공정한 분배를 목적으로 하고 있지만 실시간 시스템에서는 태스크가 종료시한을 만족하여 수행할 수 있는 지 여부가 가장 중요한 설계 요소가 된다. 시분할 시스템과는 달리 실시간 시스템에서는 시스템의 빠른 응답시간 보다는 시간의 예측성을 높인, 즉 최악의 수행시간이 어느 범위 이상을 넘지 않는 다는 것을 보장하는 데 더 관심을 가져야 하고 자원의 공정한 분배보다는 자원의 안정된 분배를 더 중요하게 여겨야 한다.[3-8]

실시간 커널이 비 실시간 커널과 구별되는 특징은 스케줄링 방식에 있다. 기존의 시분할 시스템에서는 전체 시스템을 모든 프로세서에게 공평하게 분배하면서 처리량의 증대 목적으로 스케줄링하고 있다. 이러한 상황에서 응용프로그램은 중앙처리장치와 같은 시스템 자원의 할당에 관여 할 수 없었다. 그러나 실시간 커널에서는 프로그램의 시간제약 조건을 만족시키기 위해서 사용자가 각 태스크에 대해서 우선 순위를 직접 부여함으로써 시스템 자원의 할당에 관여를 하게 된다.[2]

본 논문에서는 비 실시간 커널과 공존하도록 실시간 커널을 설계 및 구현하였다. 비 실시간 커널은 공

(1) Fast Context Exchange
(2) Small Size and Basic Function
(3) Scheduling based priority
(4) Communication between tasks
(5) Considering Timer & Interrupt effectivity
(6) Maximum Time of Interrupt tolerance time

그림 1. 실시간 커널의 요구 조건

개되어 있는 소스를 이용해 구현하였고, 실시간 커널은 경성 실시간 시스템의 필요 조건을 만족하기 위해 높은 시간 정확성과 낮은 인터럽트 지연 시간과 적은 오버헤드를 가지는 실시간 태스크를 수행하도록 하였다. 비 실시간 커널과 실시간 커널을 구분하여 구현함으로써 사용자가 최소의 비용으로 하드웨어 처리 능력을 극대화할 수 있도록 설계하였다.

본 논문의 구성은 실시간 커널에 주안점을 두어 1장에서는 서론을 기술하였고, 2장에서는 실시간 커널의 요구사항에 따라 구성요소를 설계하고 3장에서는 설계한 구성 요소를 구현하였다. 4장에서는 다른 실시간 커널과 비교 분석함으로써 성능을 평가함으로써 경성실시간 제약조건의 만족함을 보였다. 마지막으로 5장에서는 결론을 기술하였다.

II. 실시간 커널의 설계

실시간 커널에서는 사용자가 하드웨어를 직접적으로 이용하는 부담을 가지지 않고, 실시간 작업을 수행할 수 있도록 지원하기 위하여 다음과 같은 기능들을 제공한다. 그림1은 실시간 커널의 설계 요구 조건을 나타낸다.

따라서 본 논문에서 제안한 실시간 커널은 실시간 태스크를 처리하고 경성 실시간 제약을 가지는 부분과 일반적인 기능을 처리하는 부분인 비 실시간 커널로 사용자가 분리하여 프로그램을 작성할 수 있도록 그림2와 같이 실시간 커널과 비 실시간 커널로 구분하여 설계하였다

위의 그림2에서 처럼, 실시간 커널에서는 경성 실시간 태스크와 인터럽트 처리가 이루어져야 하고 실시간 태스크는 시스템 콜의 오버헤드를 줄이고 빠른

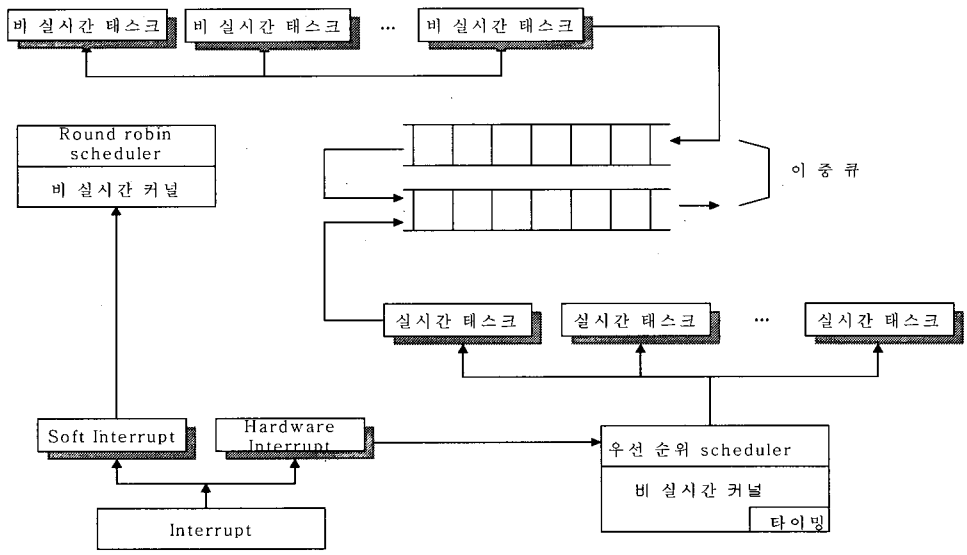

그림 2. 이중실시간 커널의 전체 구조

문맥교환을 가능하게 하기 위해서 커널 모드에서 수행한다. 실시간 커널의 스케줄러는 우선 순위에 기반한 선점형 이중 큐 스케줄러로 구현하였다. 그리고 같은 우선 순위의 태스크는 존재하지 않는다. 따라서 결과적으로 같은 시간에 최대 256 개의 태스크를 수행할 수 있다. 우선 순위 0, 1은 예약되어 사용된다. 우선 순위 0은 수행할 태스크가 없을 때 실시간 idle 태스크가 사용하고 우선 순위 1은 비 실시간 영역에서 태스크와 관련된 인터럽트 서비스를 위해 예약된다.

비 실시간 커널에서 태스크는 유저 모드에서 수행되고 스케줄러는 비선점형 방식인 시분할 방식을 사용한다. 실시간 커널과는 달리 같은 우선 순위가 존재한다. 하나의 우선순위에 여러 개의 태스크가 존재할 수 있다. 우선 순위 0은 idle 태스크를 위해 예약되어 있다. 위 두 속성을 고려하여 사용자는 두 커널을 구분하여 프로그램을 작성하고 두 커널에서 수행할 태스크가 없다면 비 실시간 영역 idle 태스크를 수행하도록 하였다. 두 커널간의 통신을 위하여 실시간 태스크에서 생성하고 비 실시간 태스크에 의해 연결되는 실시간 큐(Real-time Queue)라는 연결 통로를 구현하였다. 실시간 큐는 데이터가 실시간 커널에서 비 실시간 커널으로 이동하거나 또는 반대 방향으로 이동한다. 또한 한 방향으로만 생성할 수 있다.

2.1 실시간 커널의 설계원칙

실시간 시스템에서는 기존의 범용 시분할 시스템과는 달리 시간 제약 조건의 만족을 위해 각각의 태스크가 종료시한과 우선 순위, 주기 등을 가지게 된다. 특히 태스크의 종료시한을 만족시키는 것이 실시간 시스템에서 주요한 설계 목표로 이를 만족하기 위해서 실시간 커널의 설계 시 다음과 같은 점에 주안점을 두었다.

가. 실시간 커널 자체에서 수행되는 시간을 줄여야 한다. 커널 자체에서 수행되는 시간이 길어지면 태스크의 종료시한 내에 끝내기가 힘든 경우가 발생하므로 타이머 인터럽트와 스케줄러와 같이 자주 수행되는 부분의 수행시간을 줄였다.

나. 실시간 커널에서는 태스크간의 동기화를 이루어야 하고, 자원의 무한정 대기는 허용하지 않아야 한다. 태스크의 수행 시 자원을 기다리는 시간을 제한하지 않으면 태스크의 수행시간의 예측이 힘들어져 실시간 예측성을 보장할 수 없다.

다. 실시간 커널에서는 태스크간의 통신을 지원해야 하고, 또한 메시지를 무한하게 기다리지 못하게 제한을 두어야 한다. 이렇게 함으로써 태스크의 예측 가능성을 높이게 한다.

2.2 실시간 태스크 관리

실시간 태스크에서 태스크는 주기적 태스크와 비 주기적인 태스크로 구분될 수 있다. 주기적 태스크는 일정한 주기로 지속적으로 실행되는 것으로 초당 30번의 동영상을 출력한다면 초당 30번을 주기적으로 반복하는 태스크가 필요하게 되는 것이다. 즉 하나의 영상을 출력하는 태스크를 주기적 태스크로 지정하여 초당 30번을 반복하는 경우와 동일하다. 비 주기적인 태스크는 주기가 없이 시스템이 어떤 정해진 상태에 있을 때 실행되는 태스크를 말한다. 정해진 상태는 시스템에서 이벤트로 구현된다. 이벤트는 발생장소에 따라서 외부 이벤트와 내부 이벤트로 구분할 수 있는데 외부 이벤트는 시스템의 외부에서 발생하는 이벤트로 주로 인터럽트로 구현된다. 외부 인터럽트에 의해 실행되는 태스크는 인터럽트 태스크이다. 내부 인터럽트는 시스템의 내부에서 발생하는 이벤트로 커널이 제공하는 모든 ITC, 동기화와 상호 배제에 해당하는 이벤트이다. 대부분의 태스크는 내부 이벤트에 의해서 실행되는 데 이러한 태스크는 비동기적 태스크이다. 본 논문에서는 아래 그림3처럼 실시간 응용에 필요한 태스크에 대한 정보를 가진다.

이러한 자료구조를 가지고 4 가지의 태스크들로 구분하여 처리하도록 하였다.

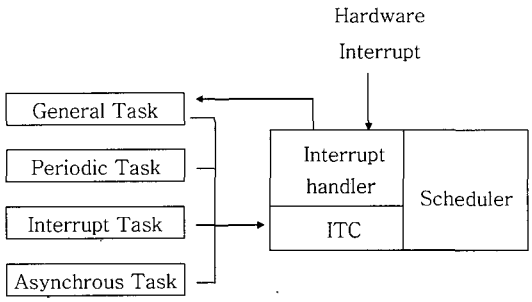

그림 4. 태스크 4가지 종류

위의 그림4에서 일반적인 태스크(General Task)는 비 실시간 커널에서 처리하는 태스크를 말한다. 인터럽트 태스크(Interrupt Task)는 모든 인터럽트가 발생했을 때 와 지역변수가 재 초기화될 때 생성하게 되는데, InterruptTask.c 파일에 구성되어 있다. 주기적 태스크(Periodic Task)는 주기적으로 장치들로부터 서비스를 서로 제공받도록 설계되었고 PeriodicTask.c 파일에 구성되어 있다. 비동기적 태스크(Asynchronous Task)는 명령이나 데이터 처리를 위해서 사용된다. 즉 명령에 의해서 장치를 제어하거나 또는 데이터를 처리하고 결과를 어느 실시간 태스크나 비 실시간 태스크에 전송하고자 할 때 비동기 태스크를 사용되고

```
Typedef struct RT_TaskControlBlock
{
        U32 KenelModeStack ;
        U32 KenelModeSP ;
        U8 State ;
        PRIORITY    Priority ;
        Struct _RT_TaskControlBlock* pPrevTCB, *pNextTCB ;
        Struct _RT_TaskControlBlock* pNext ;
        Struct _RT_TimerList *p TimeOutNext ;
        Char Name[256] ;
        U32 RT_Task ;
        Struct _PeriodTable* periodicTable ;

} RT_TCB*PRT_TCB,**ppRT_TCB,RT_WaitQue,*pRT_WaitQue,**ppRT_WaitQue ;
```

그림3. 실시간 태스크의 제어를 위한 구조체

ASyncTask.c 파일에 구성되어 있다.

III. 실시간 커널의 구현

실시간 커널에서 태스크의 종료시한을 만족시키기 위해서 스케줄링 알고리즘에 대한 많은 연구가 진행되어 왔다.[11] 본 논문의 스케줄링 알고리즘은 수행중에 스케줄링을 하기 때문에 외부 환경의 변화에 대처 할 수 있는 적응력이 뛰어나다는 장점이 있으며, 실시간 커널의 태스크간의 동기화를 위해 세마포어를 이용한다. 또한 커널이 동기화의 수단으로 인터럽트 불능을 사용하여 구현하였다.

3.1 실시간 스케줄링 알고리즘

본 논문에서는 우선 순위에 기반으로 한 정적 알고리즘 개념을 적용하여 라운드 로빈 방식에 이중 큐의 개념을 적용시키면 태스크 누적으로 인한 오버헤드를 최소화한다. 큐를 이중으로 사용하므로 한 슬라이스에 하나의 작업 신호를 보내는 것을 지향하므로 유휴시간을 최소화할 수 있다. 이러한 개선 요소의 장점을 살려서 신속한 응답성, 스케줄링의 정확성을 목표로 스케줄링 방식을 구현하였다.

위의 그림5에서 보는 것처럼 태스크들은 시스템이 할당해준 시간 내에 작업 종료가 불가능하므로 실시간 커널은 짧은 대기 시간을 할당하도록, 양쪽으로 태스크를 입력받도록 한다. 즉, 실제 실행 상태에서 작업이 타임아웃 되면 준비(ready) 상태에서는 타임아웃을 기준으로 실행 상태 점유를 많이 하지 않아도 되는 작업을 지연 가능한 태스크로 판단하고 큐의 하위 큐의 끝으로 작업을 재진입 시키고, 그렇지 않으면 작업횟수가 많이 남았으며 또한 오랫동안 대기했던 태스크에 대하여 상위 대기 큐에 재진입 시키게 된다. 상위 큐의 대기 열에 빈 공간이 생기면 하위 큐에서 상위 큐의 가장 끝으로 진입시킴으로써 우선순위를 올려 주게 된다.

3.2 ITC(Inter-Task Communication)

ITC는 태스크들간의 데이터 전송이나 이벤트를 전달해주는 방법으로 구현하는 방법은 다양하다. 이러한 방법들 중에 어떤 것을 선택하는가는 작성해야 할 응용과 동작하는 환경에 따라 결정해야 한다. 필요한 ITC를 선택하는 기준으로는 신뢰도(reliability), 내용(content), 속도(speed), 호환성(portability)가 있는 데 중요도에 따라 사용할 ITC를 결정해야 한다. ITC는 특성상 반드시 두 개 이상의 태스크들과 연관되어 있다. 서로 상대방 태스크와의 연관관계의 정도에 따라서 크게 강 결합(tightly coupled) ITC와 약 결합(loosely coupled) ITC로 분류할 수 있다. 강 결합 ITC는 매우 빠른 속도로 공유메모리를 통하여 통신을 하는 것으로 오버헤드가 적고, 성능은 향상되지만, 상호 호환을 어렵게 한다. 반면에 약 결합 ITC는 공유메모리를 사용하지 않고 두 태스크간의 정보 전송을 위한 통신규약에 따라 수행된다. 이 경우에 호환성은 좋지만 상당한 오버헤드가 발생한다.

실시간 커널에서는 태스크간의 동기화를 위해서 세마포어를 이용한다. 세마포어는 실시간 커널의 가장 기본적인 비 내용(non-content) ITC 방법이다. 세마

```
While (수행시킬 태스크가 발견될 때까지)
      for (이중 큐에 있는 모든 태스크에 대해 메모리에 적재되어
          있는 것들 중 가장 높은 우선순위를 가진 태스크를 생성)
      if(수행 가능한 태스크가 없다면)
        RT_Idle();
이중 큐로부터 선택된 태스크 제거;
Dispatch(선택한 프로세스);
```

그림 5. 스케줄러 루틴

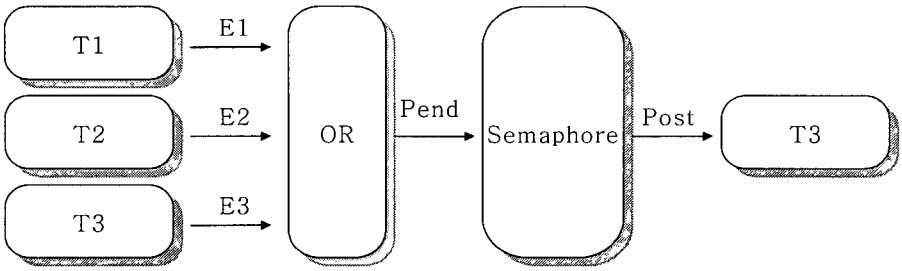

그림 6. Multi-Semaphore 의 구조

포어를 구현하기 위해서는 기초적인 접근(atomic access)을 가능하게 하는 커널 구조체를 가지고 있어야 한다. 기초적인 접근은 두 개 이상의 태스크가 세마포어를 동시에 접근하려고 할 때 발생하는 레이스 조건(race condition)을 방지해 준다. 세마포어는 제공자와 소비자로 나누어져 구현되어 있다. 제공자는 세마포어를 제공하여 소비자가 사용할 수 있도록 한다. 만일 소비자가 세마포어를 사용할 때 비어 있다면 사용을 하지 못하고 다른 행동을 하게 된다. 아래 그림 6은 일반적인 세마포어와는 달리 구현된 다중 세마포어(multi-semaphore)를 설명하고 있다.

세 개의 태스크(T1, T2, T3)와 각 태스크의 Event(E1, E2, E3)를 가정한다면, T3가 기상(wake-up)하기 위해서는 T1과 T2가 모두 E1, E2를 T3에게 보내야만 하거나 T1,T2 중 하나만 이벤트를 보내도 되는 경우가 있는 데 이런 모든 경우를 지원하도록 한 것이다.

Mutex는 두 개 이상의 태스크들이 동일한 리소스를 접근할 수 있다. 선점형 스케줄링을 수행하는 시스템에서는 보다 높은 우선 순위를 가진 태스크가 다른 태스크를 선점할 수 있기 때문에 동시에 여러 태스크가 접근하지 못하도록 Mutex를 지원한다.

Unix 계열의 신호(signal)는 연성 인터럽트(soft interrupt) 또는 비동기 인터럽트(asynchronous interrupt)라고 불려진다. 보통 인터럽트는 하드웨어가 발생시키는데, 신호는 태스크가 발생시키고 하드웨어 인터럽트는 발생 즉시 실행하게 되는 반면에 신호를 받은 태스크가 다시 프로세서를 획득하여 수행될 때 해당 신호에 대한 함수를 수행한다. 물론 수행을 마친 후에는 원래의 루틴으로 돌아와서 계속 자신의 루틴을 수행하게 된다. 본 논문에서 구현된 신호는 두 가지 모드를 지원하는 데 하나는 신호의 특성을 그대로 간직하여 해당하는 함수를 수행 후 원래의 루틴으로 되돌아오는 경우이고 또 하나는 전혀 새로운 루틴으로 태스크를 이동시키는 경우이다.

시간 관리는 실시간 커널의 기본이다. 또한 정확한 타이밍은 올바른 스케줄러의 동작을 위해 필요하다. 스케줄러는 정해진 시간에 태스크의 스위칭이 필요하다. 시간의 부정확성은 태스크 해제 흔들림(task release jitter)이라 불리는 계획된 스케줄링으로부터 벗어나게 한다. 따라서 이것을 최소화하는 것은 중요하다. 이러한 문제점을 보완하기 위해서 타이머는 시스템 시간을 기반으로 하나의 태스크를 주기적인 관점으로 비주기적인 인터럽트를 받는 것으로 관리한다. 인터럽트를 발생시키는 장치는 외부 신호 또는 원 칩(one-chip)에서 발생하는 것이어야 한다. 시간 발생기

```
#define TICK_HZ(100)
#define ONESHOT_HZ(1000)
#define TICK_PER_SWITCH (TICK_HZ / 10)
#define PULSE_PER_TICK (TIMER_FREQ / TICK_HZ)
#define PULSE_PER_ONESHOT (TIMER_FREQ / ONESHOT_HZ)
#define TICK_INTERVAL (1000/TICK_HZ)
#define ONESHOT_INTERVAL (1000 / ONESHOT_HZ)
```

그림 7. 시간 계산을 위한 상수 선언

```
typedef struct RT_TaskControlBlock
{
        U32 KenelModeStack ;
        U32 KenelModeSP ;
        U8 State ;
        PRIORITY    Priority ;
        Struct _RT_TaskControlBlock* pPrevTCB, *pNextTCB ;
        Struct _RT_TaskControlBlock* pNext ;
        Struct _RT_TimerList *pTimeOutNext ;
        Char Name[256] ;
        U32 RT_Task ;
        Struct _PeriodTable* periodicTable ;
} RT_TCB*PRT_TCB,**ppRT_TCB,RT_WaitQue,*pRT_WaitQue,**ppRT_WaitQue ;
```

그림 8. 타이머를 위한 자료구조

는 일정한 시간 간격으로 인터럽트 제공하여 한다. 다음 그림7은 실제 시간 계산을 위한 상수들을 선언한 것이다.

위의 그림7에서 타임 인터럽트는 ticks로 표시되며 인터럽트를 발생시키는 장치의 일정한 시간주기로 표현한다. 일정한 시간 주기는 시스템 초기화할 때 정해지며, 커널의 동작 동안에는 변화하지 않는다. ticks은 고정된 주파수로 발생하기 때문에 실제의 시간은 tick 간격에 의해 시간을 곱함으로써 계산된 tick수로 표현된다. 따라서 원 샷(one-shot)을 사용하여 시간 관리를 하게 된다. 또한 본 논문에서 멀티플 타이머는 타이머 이벤트의 보류(pending)의 순서 리스트를 사용하여 연속적으로 관리한다. 타이머 이벤트의 수에 관계없이 모든 활성화된 타이머를 서비스하기 위한 시간은 결정적이다. 다음 그림8은 타이머 서비스를 하기 위한 자료구조를 나타내고 있다.

3.3 인터럽트 핸들링(interrupt handling)

인터럽트는 시스템에서 어떤 기능을 수행하고 있을 때 그 수행을 중단시키는 사건으로 하드웨어에 의해 처리된다. 경성 실시간의 문제점 중의 하나가 커널이 동기화의 수단으로 인터럽트 불능을 사용한다는 것이다. 인터럽트 불능 그리고 인터럽트 가능의 무분별한 사용은 인터럽트 지명(dispatch)의 비 예측성에 영향을 준다.

경성 실시간 특징인 예측성, 인터럽트가 발생에서 인터럽트 핸들러가 불리기까지의 시간인 인터럽트 지연시간을 최소화하도록 그림9처럼 하드웨어에 따라서 인터럽트를 실시간 인터럽트와 비 실시간 인터럽트로 분류하여 하나의 이벤트로 간주하여 처리하도록 하였다. 실시간 인터럽트는 서비스를 요청한 하드웨어가 시간에 민감하여 즉시 서비스 해주어야 하는 것을 의

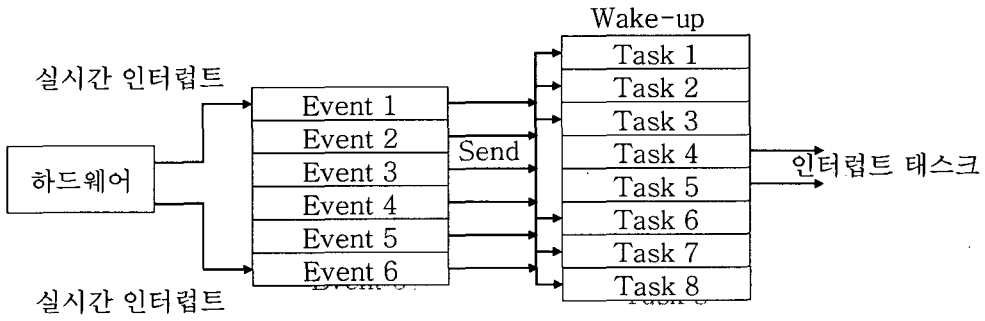

그림 9. 인터럽트 처리과정

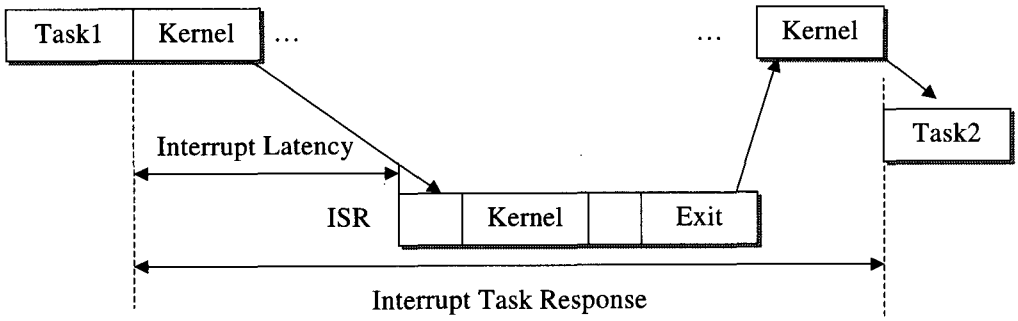

그림 10. 인터럽트 지연시간

미하고 비 실시간 인터럽트는 실시간 성질이 필요 없는 하드웨어가 발생한 것을 의미한다. 인터럽트를 이벤트로 간주하기 때문에 실시간 인터럽트는 연결된 태스크에 인터럽트 이벤트를 보냄으로써 태스크를 기상(wake-up)하게 된다. 이때 인터럽트 이벤트를 받는 태스크를 인터럽트 태스크로 분류되어 스케줄러를 통해서 처리하도록 되어 있다.

IV. 성능 및 평가

본 논문에서 제안한 실시간 커널에 대한 성능평가를 위해 다른 실시간 커널과 실험을 통해서 비교 분석하였다. 실험 환경은 Intel Pentium 166MHz, RAM 32MB 가지는 IBM PC 호환 컴퓨터에서 수행하는 Real-time Linux 0.5a 와 QNX 4.23A를 가지고 인터럽트 지연시간, 스케줄링의 정확성, 메시지 전달시간을 측정하였다.

인터럽트 지연시간은 실시간 시스템과 응용프로그램이 실행 중에 인터럽트가 발생하면 해당 ISR의 첫 코드가 실행될 때까지 걸리는 지연 시간을 말한다. 이 지연시간은 주로 실시간 시스템 내부 코드 실행 시에 행하는 인터럽트 불능(disable)과 가능(enable)설정 때문에 생기는 것으로 그 중 가장 긴 인터럽트 불능 시간(disable time)을 사용한다. 인터럽트 지연시간은 외부 사건 발생에 대응한 태스크의 응답 속도를 결정하는 중요한 요인이 된다. 본 논문에서는 인터럽트 지연시간의 최대 값을 비교하기 위해서 인터럽트 요청 신호를 보내고 난 후에 인터럽트가 응답하는 시

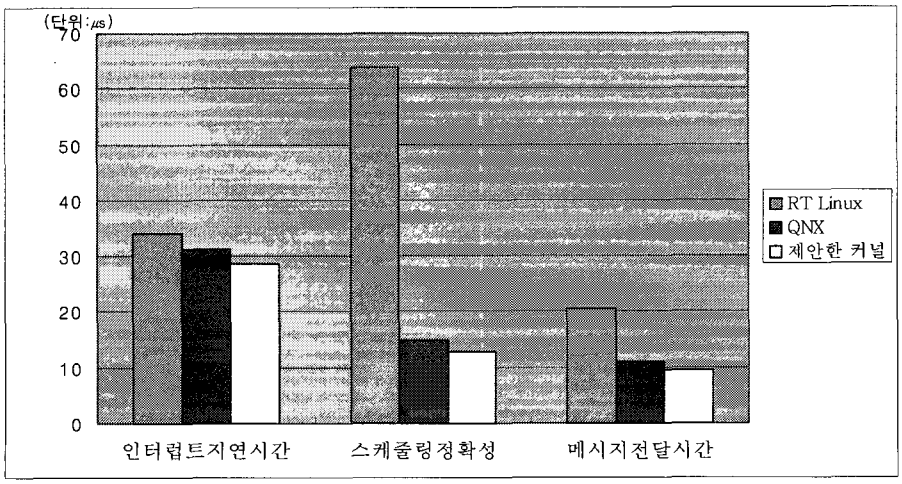

그림 11. 실험 결과 비교 분석

표 1. 전체 측정 결과표

(단위:μs)

	RT LINUX 0.5a	QNX 4.23A	제안한 커널
인터럽트 지연 시간	34.0	31.2	28.6
스케줄링 정확성	64.0	14.8	12.8
메시지 전달시간	20.5	11.1	9.6

간을 측정하였다.

그림11에서 RT Linux는 34.0μs, QNX 는 31.2μs, 본 논문에서 제안한 실시간 커널은 28.6μs 측정됨으로써 다른 실시간 커널보다 인터럽트 지연시간이 짧음을 볼 수 있다.

스케줄링의 정확성은 주기적인 실시간 태스크 수행의 스케줄링의 정확성을 측정하기 위해 각 태스크가 기상 할 때마다 시간을 측정하고 평가하였다. 측정한 결과 그림1에서 처럼 RT Linux는 64.0μs, QNX는 14.8μs, 본 논문에서 제안한 실시간 커널은 12.8μs로써 스케줄링 정확성이 다른 실시간 커널보다 빠름을 볼 수 있다. 메시지 전달시간은 생산자 프로세스가 송신(send) 함수를 호출한 후 소비자 프로세스가 메시지를 읽어 수신(receive) 함수를 빠져 나올 때 까지의 시간을 측정하였다. 아래 그림11에서 처럼 RT Linux는 20.5μs, QNX 는 11.1μs, 본 논문에서 제안한 실시간 커널은 9.6μs로써 메시지 전달시간이 다른 실시간 커널 보다 빠름을 볼 수 있다.

그림11은 제안한 실시간 커널과 다른 실시간 커널과 실험을 통한 비교 분석결과를 도표로 나타내었다.

그림 11에서 본 실험 항목들에 대한 측정 결과들을 아래와 같이 하나의 도표로 나타내었다. 위의 표1에서 보는 바와 같이 본 논문에서 제안한 실시간 커널이 다른 실시간 커널 보다 성능이 향상되었음을 볼 수 있다. 따라서 경성 실시간 제약조건을 만족하고 있음을 보여주고 있다.

V.. 결 론

본 논문에서는 시간적인 제약 조건을 만족하고 경성 실시간 시스템의 높은 예측성, 빠른 반응성, 짧은 인터럽트 지연시간, 스케줄러의 정확성 및 메시지 전달 시간 등을 만족하는 비실시간 커널의 기능들을 설계 및 구현하였다.

실시간 커널은 실시간 요소(인터럽트처리, 타이밍)를 가지는 기능들을 처리할 수 있도록 구현하였고, 비 실시간 커널에서는 일반적인 기능을 처리하도록 구현하였다. 그리고 두 영역간에 데이터 공유를 위하여 실시간 태스크를 생성하고, 비 실시간 태스크에 의해 연결되는 실시간 큐라는 연결 통로를 구현하였다. 또한 기존의 실시간 커널과의 비교 분석 및 실험을 통해, 인터럽트 지연시간에 있어서 QNX와 비교하여 본 논문에서 제안된 방법이 2.6μs 짧고, 스케줄링의 정확성은 QNX보다 2.0μs 빠름을 볼 수 있다. 메시지 전달시간은 RT- Linux와 비교하여 10.9μs 빠른 결과를 산출하였다.

향후 연구과제로 본 논문에서 제안한 최소한의 기능들만을 가지는 실시간 커널에 메모리 관리, 파일 시스템 등을 추가함으로써 완전한 운영체제로 갖추기 위한 연구가 필요하다.

참 고 문 헌

[1] A. Burns. "Scheduling hard real-time systems: A review", Software Engineering Journal, vol 6, no 3, pp. 116-128, June 1991.

[2] H. Tokuda and C. W. Mercer, "ARTS: A Distributed Real-Time Kernel", ACM Operating Systems Review(Special Issue), pp. 29-53, 1989.

[3] J. A. Stankovic and K. Ramamritham, "Scheduling algorithms and operating systems support for real-time Systems." Proceedings 15th IEEE Real-Time System symposium. vol 2, no 1, pp. 55-67, Jan. 1994.

[4] J. Reisinger, A. Damm, W. Schwabl, and H. Kopetz, "The Real-Time Operating System of MARS", ACM Operating Systems Review, vol 23, no 3,pp. 141-157, 1989.

[5] J. A. Stankovic and K. Ramamritham, "The Design of the Spring Kernel", Proceedings 8th IEEE

Real-Time Systems Symposium, San Jose, California, pp. 146-157, Dec. 1987.

[6] K. Schwan, A. Geith, and H. Zhou, "From Chaosbase to Chaosarc : A family of real-time kernels", Proceedings 11th IEEE Real-Time Systems Symposium, pp. 83-91, Dec. 1990.

[7] J. Labrosse, "A Portable Real-Time Kernel in C.", Embedded Systems Programmings, pp. 40-53, May 1992.

[8] O. Gudmundsson, D. Mosse, K-T. Ko, A. K. Agrawala, and S. Tripathi, "MARUTI · A platform for hard real-time applications", Proceedings of the Operating System for Mission-Critical Computing. Workshop. Oct. 1992.

[9] L. Sha, R. Rajkumar, and J. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization", IEEE Transactions on Computers, vol 39, no 9, pp. 1175-1185, Sep. 1990.

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

[11] M. Dertouzos and A. Mok, "Multiprocessor on-line scheduling of hard real-time tasks", IEEE Transactions on Software Engineering, vol 15, no 12, pp. 1497-1506, Dec. 1989.

[12] N. Audsley, Survey: Scheduling Hard Real-Time Systems, Department of Computer Science, University of York, 1990.

저 자 소 개

印 致 虎

1985년 한양대학교 전자공학과 공학사
1987년 한양대학교 대학원 공학석사
1996년 한양대학교 대학원 공학박사
1992년 ~ 현재 세명대학교 컴퓨터과학과 부교수
관심분야 : VLSI CAD, ASIC 설계, CAD 알고리듬, RTOS 및 내장형 시스템