論文 6-4-8

실시간 제약 커널 환경하에서의 이중 실시간 스케쥴링 설계

印 致 虎

A Dual Real-Time Scheduling Design under Real-Time Constraints Kernel Environments

Chi-ho Lin

요 약

본 논문은 실시간 제약 커널 환경 하에서의 이중 실시간 스케쥴링을 설계한다. 본 논문에서 제안한 이중 실시간 스케쥴링 설계는 실시간 제약 조건인 인터럽트 지연 시간, 스케줄링의 정확성, 메시지 전달시간을 만족하기 위하여 실시간 커널에서는 실시간 태스크 처리와 인터럽트 처리, 타이밍을 처리하도록 하였고 비 실시간 커널은 일반적인 태스크를 처리하도록 한다. 또한 태스크들의 충돌 시 혼합 우선 순위를 고려한 최적의 스케쥴링을 수행한다. 즉, 비 실시간 커널은 정적 우선 순위 스케쥴링을 수행하고, 실시간 커널은 동적 우선 순위 변형 스케쥴링인 최소 여유시간 우선 기반의 최소 선점을 갖는 스케줄링 알고리즘을 수행한다. 그리고 기존의 실시간 커널인 RT-Linux 0.5a, QNX 4.23A 와 제안한 실시간 커널이 인터럽트 지연, 스케줄링 정확성, 메시지 전달시간 등을 비교 분석함으로써 실시간 제약조건을 만족함을 보인다.

ABSTRACT

This paper proposes a dual real-time scheduling design under real-time constraints kernel environments. In this paper, we have designed both the real-time kernel and the general kernel that have their different properties to satisfy these properties, that is, interrupt latency, scheduling precision, and message passing. In real-time tasks, interrupt processing should be run. In general kernel, non real-time tasks or general tasks are run. Also, when tasks conflict, it executed the mixed priority scheduling that non real-time kernel executed static scheduling and real-time kernel executed dynamic priority transformation scheduling, that is, least-laxity-first/minimization preemption scheduling. We have compared the results of this study for performance of the proposal real-time kernel with both RT Linux 0.5a and QNX 4.23A, that is, of interrupt latency scheduling precision and message passing.

Keywords: Real-time, Non Real-time, Scheduling, Kernel, Dynamic Real-time Queue

1. 서 론

컴퓨터가 산업 전반에 폭넓게 이용됨에 따라 수송 시스템, 생산설비, 고속 통신 시스템 등 다양한 분야에서 고도의 제어 기능 및 편리한 사용자 환경이 컴퓨터를 활용하여 구현되고 있다. 기존의 시분할 시스템은 프로그램의 논리적 정확성과 처리량만을 위주로 설계되어 왔다. 이로 인하여 점차 필요성이 인식되고 있는 시간적 제약조건을 만족시키기에 많은 어려움을 겪고 있다. 이러한 문제점을 해결하기 위하여 실시간 시스템은 논리적이고 기능적인 특성뿐만 아니라 시간적인 제약까지도 중요한 요소로 고려하여 설계하고 있다. 즉 어떤 이벤트가 발생하였을 때 정해진 시간이내에 처리되는 것을 보장하는 시스템이다[1].

실제 사용되고 있는 대부분의 실시간 커널은 제어 시스템에서 동작하는 실행체제의 형태이다. 이러한 특

성을 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[2-4].

실시간 커널이 비 실시간 커널과 구별되는 특징은 스케줄링 방식에 있다[2,5]. 기존의 시분할 시스템에서는 전체 시스템을 모든 프로세서에게 공평하게 분배하면서 처리량의 증대 목적으로 스케줄링하고 있다. 이러한 상황에서 응용프로그램은 중앙처리장치와 같은 시스템 자원의 할당에 관여할 수 없었다. 그러나 실시간 커널에서는 프로그램의 시간제약 조건을 만족시키기 위해서 사용자가 각 태스크에 대해서 우선 순위를 직접 부여함으로써 시스템 자원의 할당에 관여를 하게 된다[6,8].

본 논문에서는 비 실시간 커널과 실시간 커널이 공유하도록 스케줄링을 설계 및 구현하였다. 비 실시간 커널은 공개되어 있는 소스를 이용해 구현하여 라운드 로빈(round robin)기법을 사용하였고, 실시간 커널은 실시간(real-time) 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 서론에 이어 2장은 실시간 제약 환경 하에서의 이중 실시간 스케줄링 설계에 대해 기술하고, 3장에서는 상용화된 다른 실시간 커널을 기반으로 스케줄링과 비교 분석함으로써 성능을 평가함으로써 실시간 제약조건을 만족하는 함을 보인다. 마지막으로 4장에서는 결론을 기술한다.

2. 실시간 제약 커널 환경하에서의 이중 실시간 스케줄링 설계

2.1 전반적인 실시간 요구 조건

본 논문에서 제안한 실시간 제약 커널 환경 하에서의 이중 실시간 태스크를 처리하고 실시간 제약을 가지는 부분과 일반적인 기능을 처리하는 부분인 비실시간 커널으로 사용자가 분리하여 프로그램을 작성할 수 있도록 그림 1과 같이 실시간 커널과 비실시간 커널으로 구분하여 설계하였다.

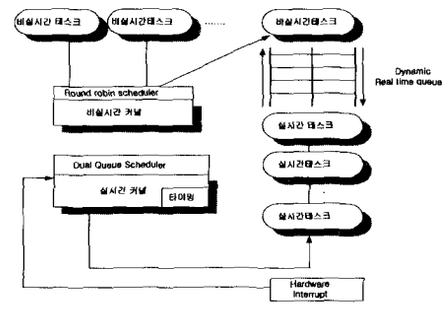

그림 1 이중 실시간 커널의 전체 구조
Fig. 1 The overall structure of dual real-time kernel

그림 1 에서처럼, 실시간 커널에서는 실시간 태스크와 인터럽트 처리가 이루어져야 하고 실시간 태스크는 시스템 콜의 오버헤드를 줄이고 빠른 문맥교환을 가능하게 하기 위해서 커널 모드에서 수행한다. 실시간 커널의 스케줄러는 우선 순위에 기반 한 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 비실시간 영역에서 태스크와 관련된 인터럽트 서비스를 위해 예약된다.

비실시간 커널에서 태스크는 유저 모드에서 수행되고 스케줄러는 비선점형 방식인 시분할 방식을 사용한다. 실시간 커널과는 달리 같은 우선 순위가 존재한다. 하나의 우선 순위에 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 하였

다. 두 커널 간의 통신을 위하여 실시간 태스크에서 생성하고 비 실시간 태스크에 의해 연결되는 동적 실시간 큐(Dynamic 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.

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 커널은 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 줄인다.

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 예측가능성을 높이게 한다.

실시간 태스크에서 태스크는 주기적 태스크와 비 주기적인 태스크로 구분될 수 있다. 프로세스 제어 시스템 등과 같은 경우에는 일정한 주기마다 입력 값을 읽고 그에 따라 적당한 제어 동작을 하는 주기적인 태스크들이 주로 사용된다. 이에 반해 자동 조립 시스템 등과 같은 경우에는 해당 부품의 도착 등과 같은 외부의 이벤트에 따라 적당한 작업을 수행하는 비 주기적인 태스크들이 주로 사용된다.

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 실행되는데 이러한 태스크는 비동기적 태스크이다. 본 논문에서는 다음 그림 2처럼 실시간 응용에 필요한 태스크에 대한 정보를 가진다.

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

그림 2 실시간 태스크의 제어 구조체
Fig. 2 The structure of dual real-time kernel

이러한 자료구조를 가지고 4 가지의 태스크들로 구분하여 처리하도록 하였다.

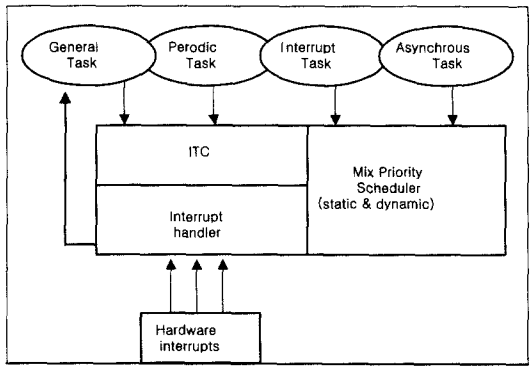

그림 3 태스크 4 가지 종류
Fig. 3 A class of four task

그림 3 에서 일반적인 태스크(General Task)는 비 실시간 커널에서 처리하는 태스크를 말한다. 인터럽트 태스크(Interrupt Task)는 모든 인터럽트가 발생했을 때와 지역변수가 재 초기화될 때 생성하게 되는데, InterruptTask.c 파일에 구성되어 있다. 주기적 태스크(Perodic Task)는 주기적으로 장치들로부터 서비스를 서로 제공받도록 설계되었고 Perio-dicTask.c 파일에

구성되어 있다. 비 동기적 태스크(Asynchrous 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 ASync-Task.c 파일에 구성되어 있다.

2.2 제안한 이중 큐 스케줄링 알고리즘

실시간 커널에서는 태스크의 종료시한을 만족시키기 위해서 스케줄링 알고리즘[10]에 대한 많은 연구가 진행되어 왔다. 스케줄링 알고리즘은 크게 정적인 알고리즘과 동적인 알고리즘으로 나눌 수 있다. 정적인 스케줄링 알고리즘은 태스크의 수행 이전에 태스크의 특성에 대해 모두 알고 있다는 가정 하에 오프라인 시에 스케줄링을 하는 방식으로 실제 실행시의 시스템의 부하가 적은 장점을 가지고 있다. 그러나 시스템의 환경 변화에 대처할 수 있는 없는 것이 단점이다. 이러한 정적인 스케줄링 알고리즘은 고정된 수의 센서와 구동기를 가지고 있으면서 시스템 환경 및 처리 요구가 잘 정의된 시스템에서 주로 이용된다. 반면 동적인 스케줄링 알고리즘은 모든 태스크의 집합에 대해서 그 특성을 알지 못하지만 현재 활성화된 태스크에 대해서는 모든 정보를 가지고 있다. 그러나 새로운 태스크가 미래에 도착할 수 있기 때문에 시간에 따라 스케줄링이 변하게 된다. 동적인 스케줄링 알고리즘은 수행 중에 스케줄링을 하기 때문에 외부 환경의 변화에 대처 할 수 있는 적응력이 뛰어나다는 장점이 있다. 그러나 실행시의 오버헤드가 크다는 단점이 있다. 동적인 방식에서는 수행 중에 스케쥴링이 이루어지므로 스케쥴링의 부담을 최소화하는 것이 매우 중요하다.

본 논문에서 제안한 이중 큐 스케쥴링은 우선 순위에 기반으로 한 정적 및 동적 알고리즘 개념을 적용하여 비 실시간 태스크는 라운드 로빈 방식에 이중 큐의 개념을 적용시키면 태스크 누적으로 인한 오버헤드를 최소화 할 수 있다. 반면 실시간 태스크는 동적 스케쥴링을 적용하여 즉각적인 처리가 이루어지도록 하면서 또한 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.

그림 4에서 보는 것처럼 비 실시간 태스크들이 많은 경우에 시스템이 할당해 준 시간 내에 작업 종료가 불가능하므로 실시간 커널은 짧은 대기 시간을 할당하도록, 양쪽으로 태스크를 입력받도록 한다. 즉, 실제 실행 상태에서 작업이 타임아웃 되면 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 우선 순위를 올려 주게 된다.

```
While (수행시킬 태스크가 나타날 때까지)
   for (이중 큐에 있는 비 실시간 태스크에
        대해 메모리에 적재되어 있는 것들 중
        가장 높은 우선 순위를 가진 태스크를
        생성)
   if (수행 가능한 태스크가 없다면)
      Non_RT_idle( );
이중 큐로부터 선택된 태스크 제거;
Dispatch(선택된 프로세스);
```

그림 4 비실시간 스케줄러 루틴
Fig. 4 The non-real-time scheduler routine

실시간 태스크들 또한 정적 우선 순위 스케쥴링과 동일한 방법이나 동적 우선 순위 방법 중 하나로수행 중에 여유시간에 가장 가까운 태스크가 우선 순위를 갖는 방법으로 최소 여유시간 우선(Least-Laxity-First) 스케쥴링은 최소 여유 시간을 갖는 태스크가 가장 높은 우선 순위를 갖는다. 여기서 여유 시간이란 태스크가 선점 당하지 않고 수행을 마쳤을 때 마감까지의 여분시간이다. 즉, 최소 여유시간을 갖는 가진 태스크가 여러 개 존재하여 여유시간 충돌이 발생한 경우 이 태스크들 간에 빈번한 문맥교환이 발생하게 되는 문제점이 있어 실용적이지 못하다. 그러므로 최소 여유시간 우선 기반의 최소 선점을 갖는 스케줄링 알고리즘(Least-Laxity-First/ Minimization Preemption scheduling)[8]은 여유시간이 충돌했을 때에 문맥교환을 최소화함으로써 최소 여유시간 우선 스케줄링 알고리즘의 단점을 해결한다. 그림 5는 최소 여유시간 우선 기반의 최소 선점 스케쥴링으로 다음과 같은 가정 하에서 동작한다.

```
가. n 개의 독립적인 태스크가 단일 프로세스 상
    에서 스케쥴링 된다.
나. 모든 태스크는 선점 가능하다.
```

태스크 Ti의 주기와 최악 수행시간, 그리고 임의의 시간 t에서 태스크의 상태는 태스크의 마감시간(deadline) Di(t)와 태스크가 수행 종료까지의 남은 잔여 수행시간 Ei(t), 이 태스크의 여유시간 Li(t)는 다음 식 (1)과 같다.

$$Li(t) = Di(t) - Ei(t) \quad (1)$$

```
/* 재스케쥴링 시점 조건 */
/* 1. Ta : 최소 여유시간을 갖는 태스크 중에서
        가장 작은 잔여 수행시간을 갖는 태스크 */
/* 2. 현재 태스크 Ta 에게 남은 잔여 프로세서
     할당시간 */
/* 3. 새요구된 태스크 Tnew 는 Dnew(t) < Da(t) 와
     Lnew(t) < La(t)를 만족 */
/* 4. 새요구된 태스크가 Tnew 는 Dnew(t) < Da(t) 와
     Lnew(t) ≥ La(t)를 만족하면 현재 수행되는 태스크
     Ta 에게 남은 잔여 프로세서 할당시간
     Π=min(Π,Dnew(t) - La(t))를 만족 */
Algorithm( 최소 여유시간 우선 기반의 최소
           선점 스케쥴링)
begin
 V1 = {Ti | Li(t) ≤ Lj(t), Ti, Tj ∈ T}와
 Ta= {Ti | Ei(t) ≤ Ej(t), Ti, Tj ∈ V1} 를
     만족하는 Ta 찾음 ;
 Tmin = {Ti | Di(t) ≤ Dj(t) 와 Li(t), Lj(t) >
  La(t), Ti, Tj ∈ T}를 만족하는 Tmin 를 찾음;
 어떤 재스케쥴링 시점 조건을 만족할 때까지 Ta 수행 ;
end
```

그림 5 최소 여유시간 우선 기반의 최소 선점 스케쥴링 알고리즘

Fig. 5 The minimum occupancy scheduling algorithm a base minimum idle time priority

2.3 ITC (Inter-Task Communication)

ITC 는 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 Reliability, Content, Speed, Portability가 있는 데 중요도에 따라 사용할 ITC를 결정해야 한다. ITC는 특성상 반드시 두 개 이상의 태스크들과 연관되어 있다. 서로 상대방 태스크와의 연관관계의 정도에 따라서 크게 tightly coupled ITC 와 loosely coupled ITC로 분류할 수 있다.

tightly coupled ITC는 매우 빠른 속도로 공유메모리를 통하여 통신을 하는 것으로 오버헤드가 적고, 성능은 향상되지만, portability를 어렵게 한다. 반면에 loosely coupled ITC는 공유메모리를 사용하지 않고 두 태스크간의 정보 전송을 위한 통신규약에 따라 수행된다. 이 경우에 portability는 좋지만 상당한 오버헤드가 발생한다. 또한 실시간 태스크 처리에 있어서 가장 중요성이 부각되는 우선 순위 역전에 관한 문제는 낮은 우선 순위에 있는 태스크가 높은 우선 순위의 태스크보다 높은 우선 순위를 부여받는 상황을 일컫는 것으로 이의 해결 방법으로는 이런 상황의 해결 방법은 높은 우선 순위를 가진 태스크가 잠금(lack)을 가짐으로써 충돌 시 우선 순위를 비교한다. 다른 방법으로서는 자원의 낭비를 줄이기 위해 우선 순위 상속을 확장하고 수행 완료가 가까운 것을 우선 수행하게 한다. 하위 우선 순위 태스크의 수행이 완료되는 경우 상위 우선 순위를 상속한다. 반면, 하위 우선 순위의 태스크를 재 시도하는 경우 상위 우선 순위 태스크의 대기시간을 줄일 수 있게 하였다.

2.4 타이머(Timer)

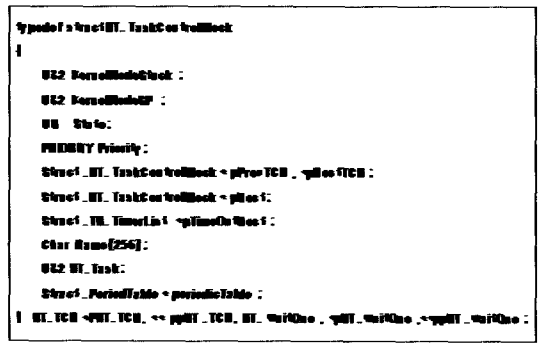

그림 6 타이머를 위한 자료구조

Fig. 6 The structure for the timer

시간 관리는 실시간 커널의 기본이다. 또한 정확한 타이밍은 올바른 스케줄러의 동작을 위해 필요하다. 스케줄러는 정해진 시간에 태스크의 스위칭이 필요하다. 시간의 부정확성은 태스크 릴리즈 지터(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 On-chip에서 발생하는 것이어야 한다. 시간 발생기는 일정한 시간 간격으로 인터럽트 제공하여 한다.

일정한 시간 주기는 시스템 초기화할 때 정해지며, 커널의 동작 동안에는 변화하지 않는다. ticks은 고정

된 주파수로 발생하기 때문에 실제의 시간은 tick 간격에 의해 시간을 곱함으로써 계산된 tick수로 표현된다. 따라서 One-shot을 사용하여 시간 관리를 하게 된다. 또한 본 논문에서 멀티플 타이머는 타이머 이벤트의 pending의 순서 리스트를 사용하여 연속적으로 관리한다. 타이머 이벤트의 수에 관계없이 모든 활성화된 타이머를 서비스하기 위한 시간은 결정적이다. 다음 그림 6은 타이머 서비스를 하기 위한 자료구조를 나타내고 있다.

2.5 인터럽트 핸들링(Interrupt handling)

```
Interrupt( )
{
  disable interrupt;
            // 인터럽트 차단
  register each interrupt;
            // 사용자 정의 인터럽트
  save register & state in stack;
            // 인터럽트 정보를 지정
  interrupt processing;
            // 인터럽트 처리
  return to interrupt process;
            // 인터럽트 복귀
}
```

그림 7 인터럽트 처리 함수
Fig. 7 Interrupt processing function

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

실시간 특징인 예측성, 인터럽트가 발생에서 인터럽트 핸들러가 불리기까지의 시간인 인터럽트 지연시간을 최소화하도록 하였다. 실시간 인터럽트는 서비스를 요청한 H/W가 시간에 민감하여 즉시 서비스 해주어야 하는 것을 의미하고, 비 실시간 인터럽트는 실시간성이 필요 없는 H/W가 발생한 것을 의미한다. 인터럽트를 이벤트로 간주하기 때문에 실시간 인터럽트는 연결된 태스크에 인터럽트 이벤트를 보냄으로써 태스크를 wake-up하게 된다. 이때 인터럽트 이벤트를 받는 태스크를 인터럽트 태스크로 분류되어 스케줄러를 통해서 처리하도록 되어 있다. 또한 그림 7은 인터럽트 처리 함수를 나타낸다.

3. 성능 및 평가

본 논문에서 제안한 이중 실시간 커널은 AVIION WS SYSTEM에서 C++로 구현하였고, 또한 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 실시간 제약 조건인 인터럽트 지연시간, 스케줄링의 정확성, 메시지 전달시간을 시스템 안정성을 배제하고 실시간 제약 조건들의 수치적으로 제한하여 측정하였다.

3.1 인터럽트 지연 시간

인터럽트 지연시간의 최대 값을 비교하기 위해서 인터럽트 요청 신호를 보내고 난 후에 인터럽트가 응답하는 시간을 측정하였다. 그림 8처럼 RT Linux는 34.0 μs, QNX 는 31.2μs, 본 논문에서 제안한 실시간 커널은 29.7μs 측정됨으로써 다른 실시간 커널보다 인터럽트 지연시간이 짧음을 볼 수 있다.

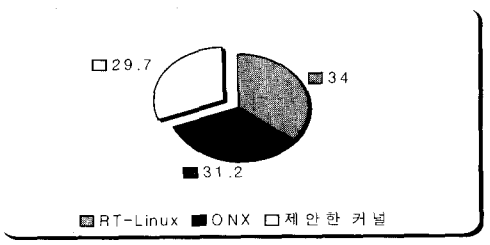

그림 8 인터럽트 지연시간 측정
Fig. 8 The measurement of interrupt delay time

3.2 스케줄링 정확성

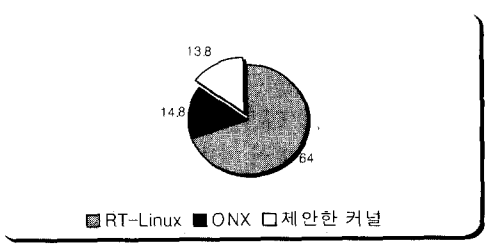

그림 9 스케줄링 정확성 측정
Fig. 9 The measurement of scheduling correlation

주기적인 실시간 태스크 수행의 스케줄링의 정확성을 측정하기 위해 각 태스크가 Wake-up할 때마다 시

간을 측정하고 평가하였다. 측정한 결과 그림 9에서 처럼 RT Linux는 64.0μs, QNX는 14.8μs, 본 논문에서 제안한 실시간 커널은 13.8μs로써 스케줄링 정확성이 다른 실시간 커널 보다 빠름을 볼 수 있다.

3.3 메시지 전달시간

생산자 프로세스가 Send 함수를 호출한 후 소비자 프로세스가 메시지를 읽어 receive 함수를 빠져 나올 때 까지의 시간을 측정하였다. 그림 10에서 처럼 RT Linux는 20.5μs, QNX 는 11.1μs, 본 논문에서 제안한 실시간 커널은 9.7μs로써 메시지 전달시간이 다른 실시간 커널 보다 빠름을 볼 수 있다.

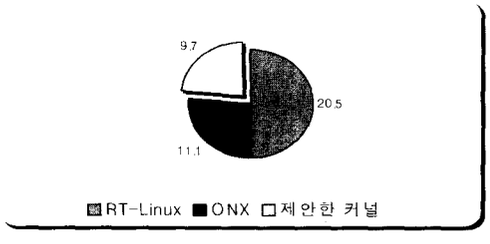

그림 10 메시지 전달 지연시간
Fig. 10 The message passing delay time

4. 결 론

본 논문은 실시간 제약 커널 환경 하에서의 이중 실시간 스케쥴링 설계를 하였다.

제안된 이중 실시간 스케쥴링은 실시간 커널은 실시간 요소를 가지는 기능들을 처리 및 동적 우선 순위 스케쥴링인 최소 여유시간 우선 기반의 최소 선점을 갖는 스케줄링 알고리즘을 수행하여 동적 스케쥴링의 단점인 문맥교환을 최소화하였고, 비 실시간 커널에서는 일반적인 기능을 처리와 정적 스케쥴링을 수행하도록 하였다. 그리고 두 영역간에 데이터 공유를 위하여 실시간 태스크에서 생성하고 비 실시간 태스크에 의해 연결되는 동적 실시간 큐라는 연결 통로를 구현하였다. 비교실험에서는 기존의 RT Linux, QNX 와 같은 실시간 커널과 실험을 통해 인터럽트 지연시간, 스케줄링의 정확성, 메시지 전달 시간을 비교 분석함으로써 실시간 제약을 만족함을 보였다.

향후 연구과제로 본 논문에서 제안한 알고리즘을 토대로 최소한의 기능들만을 가지는 실시간 커널에 메모리 관리, 파일 시스템 등을 추가함으로써 스케쥴링의 최적화로 인한 완전한 운영체제로 갖추기 위해서 연구되어야 할 것이다.

참 고 문 헌

[1] J. A. Stankovic, "Misconceptions about real-time computing." IEEE Comput. Vol. 21, No. 10, Oct. 1988.

[2] Krithi Ramamritham, John A.Stankovic, "Scheduling Algorithms and Operating Systems Supportfor Real-Time Systems." Proceedings of the IEEE. Vol. 82, No. 1, January pp. 55~67, 1994

[3] H. Kopetz, A. Demm, C. Koza and m. Mulozzani, "Distributed fault tolerant real-time systems : The MARS approach" IEEE Micro, pp. 25~40, 1989.

[4] http://redwood.snu.ac.kr 서울대학교 실시간운영체제연구실.

[5] J. A Stankovic and K. Ramamritham, "The Spring kernel : A New paradigm for hard real-time operating systems" IEEE Siftware, vol. 8, No. 3 pp. 62~72, May 1991.

[6] K. Schwan, A. Geith and H. Zhou, "From Chaosbase to Chaosarc : A family of real-time kernels", Proc. Real-Time System Symp, pp. 83~91, Dec. 1990.

[7] H. Tokuda and C. Mercer, "ARTS: Adistributed real-time kernel", ACM Operating System Rev, Vol. 23, No. 3, July 1989.

[8] O. Gudmundsson, D. Mose, K. Ko, A. Agrawala and S. Tripathi, "MARUTI An environment for hard real-time applications" Mission Critical Operating Systems. IOS Press 1992.

[9] Michael Barabanov, " A Linux-based Real-Time Operating System" , New Mexico Institute of Mining and Technology Socorro, New mexico, June 1, 1997.

[10] C. L Liu and J. W Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. Journal of the ACM,20(1) : 44~61, January 1973.

저 자 소 개

인치호(印致虎)

1985년 한양대 전자공학과 졸업. 1987년 동 대학원 졸업(석사). 1996년 동 대학원 졸업(박사). 1992년~현재 세명대학교 컴퓨터과학과 부교수.