이중 실시간 커널의 설계를 위한 스케줄링 알고리즘

정회원 인 치 호*

A New Scheduling Algorithm for Dual Real-Time Kernel Design

Chi-ho Lin* *Regular Member*

요 약

본 논문은 실시간 커널과 비실시간 커널의 특성을 모두 가질 수 있는 이중 실시간 커널의 설계를 위한 이중 큐 스케줄링 알고리즘을 제안한다.

본 논문에서 제안한 실시간 커널은 실시간 제약들을 고려한 실시간 커널과 비실시간 커널의 특성을 가지도록 설계한다. 그리고 실시간 태스크 안에는 인터럽트처리를 수행하고, 비실시간 태스크는 일반작인 작업을 수행한다. 실시간 커널에는 인터럽트 처리와 실시간 태스크를 처리하도록 하였다. 비실시간 커널은 일반적인 태스크를 처리하도록 한다. 또한 실시간 커널과 비실시간 커널을 이중 큐를 통하여 태스크의 처리를 수행한다. 그리고 실시간 태스크는 고정 우선 순위기반 스케줄인 RMS을 사용하여 실시간 태스크 스케줄링을 한다.

제안된 실시간 시스템은 RT-Linux, QNX 와 인터럽트 지연, 스케줄링 정확성, 메시지 전달시간 등을 비교 분석하여 효율성을 입증한다.

ABSTRACT

This paper proposes a new dual queue scheduling algorithm for dual real-time kernel design that can have all real-time kernel and special quality of non real-time kernel.

In this paper, the proposed real-time kernel have planned to have real-time kernel and special quality of rain real-time kernel that consider real time restrictions. In real-time tasks, interrupt processing and real time task should be run. In general kernel, non-real time tasks or general tasks are run, and then real-time kernel and non real-time kernel process task through dual queue. Also, real-time task uses RMS(Rate Monotonic Scheduling) that is fixing priority order base scheduling and do real-time task scheduling.

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.

Ⅰ. 서 론

컴퓨터의 사용범위가 확대되고 응용분야가 다양해짐에 따라서 그에 대한 요구사항도 다양해지게 되었다. 이들 응용분야 중에서 외부에서 발생한 요구에 대한 처리가 제한된 빠른 시간 내에 이루어져야 하는 경우에 사용되는 시스템을 일반적으로 실시간 시스템이라 일컬으며 여기에 대한 많은 연구가 진행되어 왔다. 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[1-2].

실시간 시스템은 기존의 컴퓨터 시스템과 달리

* 세명대학교 컴퓨터과학과(ich410@venus.semyung.ac.kr)
논문번호 : K01155-0709, 접수일자 : 2001년 7월 9일

시스템 동작의 정확성이 논리적 정확성뿐만 아니라 시간적 정확성에도 좌우되는 시스템을 말한다. 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[3-4]. 예를 들어 비실시간 시스템인 기존 리눅스의 프로세스 관리에 보면 프로세스를 일반 프로세스와 실시간 프로세스로 분류하여 실시간 프로세스에게 더 높은 우선 순위를 주고 이들에게 별도의 스케줄링 정책을 사용하는 것을 볼 수 있다. 그러나 이는 단지 더 높은 권한을 부여하여 스케줄링을 할 때 실시간 프로세스가 다른 프로세스보다 먼저 수행되게 하는 것이지, 일반적인 실시간 스케줄링 요건을 충족하지 못한다. 리눅스는 커널 모드에서 선점할 수 없다. 어떤 프로세스가 시스템 콜을 불러 커널 모드에 진입한 경우 이 프로세스를 중지시킬 수 없다. 또한 커널은 필요한 경우 인터럽트가 발생하는 것을 막아버리기도 한다. 이 경우 인터럽트가 발생하더라도 그 수행이 지연될 수밖에 없다. 그리고 실시간 시스템의 예로서 QNX 커널[4]은 IPC, 스케줄러, 인터럽트 전달부, 네트워크 인터페이스만으로 이루어져 있다. 그리고 스케줄링 단위는 프로세스이다. 이와 같은 마이크로 커널의 경우 IPC에 의한 성능저하가 가장 큰 문제점이다. 이를 위해 QNX는 메시지 전달에서의 오버헤드를 최소화하기 위해 동기적인 메시지 전달 방식을 사용한다. 한편, QNX는 인터럽트 처리를 위해 프록시라는 개념을 사용한다. 위의 예에서와 같이 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. 실시간 시스템에 존재하는 시간제약 조건은 종료시한(deadline)으로 주어진다. 종료시한은 그것의 엄격성에 의해 두 가지로 분류될 수 있다. 첫째로, 경성 실시간(hard Real-Time)은 시스템이 주어진 종료시한을 만족시키지 못한 경우에 막대한 재산적 손실이나 인명의 피해를 주는 경우를 말한다. 둘째로 연성 실시간(Soft Real-Time)은 시간 제약 조건을 만족시키지 못하더라도 치명적이지 않고 종료시한을 넘겨 수행을 마쳐도 계산의 결과가 의미가 있는 경우를 말한다[5-6]. 이러한 실시간 커널과 비실시간 커널을 혼합한 RT Linux[11]의 구조를 살펴보면 무척 단순하기 때문에 기존 리눅스 프로그래머들이 쉽게 접근할 수 있다는 장점이 있다. 실시간 기능이 필요한 부분만 RT 태스크로 구현하고 나머지는 이미 어느 정도 검증되어 있는 안정된 리눅스 상에서 동작하므로 안정적이고, 디버깅이 용이하다는 장점이 있다. RT Linux는 기존의 리눅스가 가진 기능들을 실시간으로 바꾼 것은 아니다. 단지 RT 태스크 기능을 추가하였을 뿐, 대부분의 서비스는 기존 리눅스에서 그대로 담당하게 된다. 리눅스 코드의 대부분을 차지하는 디바이스 드라이버도 그대로 사용되고 있으며, 이들에게 실시간 기능이 필요한 경우 이들 디바이스 드라이버를 다시 설계해서 만들어야 하는 문제가 있다. 또한 RT Linux는 기존의 리눅스의 기능에 추가를 한 것이기 때문에 여전히 덩치가 크다는 부담이 있다.

본 논문에서는 고정 우선 순위기반 스케줄링인 RMS(Rate Monotonic Scheduling)[9]를 이용하여 실시간 태스크 스케줄링을 하였다. 실시간 스케줄링과 비 실시간 스케줄링을 따로 수행한 후 이중 큐를 이용하여 스케줄링된 실시간 태스크와 비 실시간 태스크를 우선 순위를 부여하여 라운드 로빈 방식에 이중 큐의 개념을 적용시켜 태스크의 누적으로 인한 오버헤드를 최소화하였다. 만약 종료 시간을 지키지 못할 경우 하드웨어 인터럽트에 의해 종료되고 다음으로 가장 우선 순위가 높은 태스크가 실행하게 하였다. 또한 큐를 이중으로 사용하므로 한 슬라이드에 하나의 작업 신호를 보내는 것을 지향하므로 유휴시간을 최소화할 수 있고, 이러한 개선 요소의 장점을 살려서 신속한 응답성, 스케줄링의 정확성을 목표로 스케줄링 방식을 설계 및 구현하였다.

본 논문의 구성은 실시간 커널에 주안점을 두어 1장에서는 서론을 기술하고, 2장에서는 실시간 커널의 요구사항에 따라 구성요소를 설계 및 구현하였고 3장에서는 다른 실시간 커널과 비교 분석함으로

써 성능을 평가함으로써 경성실시간 제약조건을 만족함을 보였다. 마지막으로 4장에서는 결론을 기술한다.

Ⅱ. 실시간 커널의 설계 및 구현

2.1 구성

실시간 커널에서는 사용자가 하드웨어를 직접적으로 이용하는 부담을 가지지 않고, 실시간 작업을 수행할 수 있도록 지원하기 위하여 다음과 같은 기능들을 제공한다.

아래 그림 1의 실시간 커널의 요구조건 (1)에서 문맥교환이 이루어지는 동안은 실제 응용 프로그램에서 원하는 작업은 이루어지지 않기 때문에 오버헤드라고 볼 수 있다. 따라서 문맥교환은 짧으면 짧을수록 좀 더 효율적인 실시간 시스템이 될 수 있다. 또한 예전의 실시간 시스템과 비교할 때는 이 시간이 얼마나 짧은가가 매우 중요한 척도가 된다. 그리고 (2)는 비실시간 커널과 실시간 커널이 공존함에 따라 실시간 기능이 가능한 부분만 실시간 태스크로 구현하고 나머지는 비실시간 커널에서 동작하므로 최소의 기능만을 보유한 작은 크기의 커널 구현이 가능하다.

(1) Fast context exchange
(2) Kernel of small size that possess minimum function
(3) Priority order base scheduling
(4) Communication between task
(5) Minimization of interrupt latency time

그림 1. 실시간 커널의 요구 조건

위 그림 1에서 (3)의 우선 순위 기반 스케줄링은 태스크 집합이 유동적인 시스템에 적합한 사건 구동형 실시간 시스템에 사용된다. 따라서 우선 순위 기반 스케줄링을 사용하므로 유연성을 증가시킬 수 있다. (4)태스크간의 통신하는 방법으로는 global data를 쓰는 방법과 message passing 방법의 2가지가 있다. Global variable을 쓰는 경우에 exclusive access를 해야 하는데 ISR(Interrupt Service Routine)이 포함된 인터럽트를 disable해야 하며 태스크 사이에서는 인터럽트를 disable하는 방법 외에 semaphore를 사용할 수도 있다. Message passing의 방법으로는 mailbox, queue, pipe 등이 있다. 이들 3개의 방법 중에 어느 것을 쓸 것인가는 보낼 message의 성격에 따라 결정해야 한다. (5)의 인터럽트 지연시간은 실시간 시스템과 application이 실행 중에 인터럽트가 발생하면 해당 ISR의 첫 코드가 실행될 때까지 걸리는 지연 시간을 나타낸다. 인터럽트 지연시간은 외부 사건 발생에 대응한 태스크의 응답속도를 결정하는 중요한 요인이 된다.

본 논문에서 제안한 실시간 커널은 실시간 태스크를 처리하고 경성 실시간 제약을 만족하는 부분과 일반적인 기능을 처리하는 부분인 비실시간 커널로 사용자가 분리하여 프로그램을 작성할 수 있도록 그림 2와 같이 실시간 커널과 비실시간 커널로 구분하여 설계하였다.

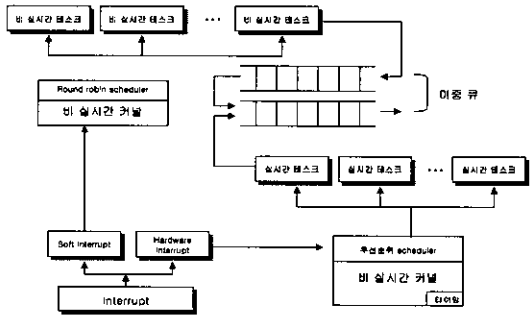


그림 2. 이중 실시간 커널의 전체 구조

위 그림 2에서 처럼, 실시간 커널에서는 경성 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 기반한 선점형 이중 큐 스케줄러로 구현하였다. 그리고 같은 우선 순위의 태스크는 존재하지 않는 것으로 가정한다. 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 비실시간 영역 idle 태스크를 수행하도록 하였다.

두 커널간의 통신을 위하여 실시간 태스크에서 생성하고 비실시간 태스크에 의해 연결되는 이중

큐라는 연결 통로를 구현하였다. 비실시간 커널은 선점(preempt)할 수 없다. 어떤 프로세스가 시스템 콜을 불러 커널모드에 진입한 경우 이 프로세스를 중지시킬 수 없다. 또한 비실시간 커널은 필요한 경우 인터럽트가 발생하는 것을 막아버리기도 한다. 따라서 실시간 커널은 비실시간 커널을 실시간 커널상에서 돌아가는 하나의 태스크로 생각한다. 그리고 이중 큐를 통하여 비실시간 태스크를 하위 큐의 끝으로 태스크를 진입시키고, 비실시간 커널에는 가장 낮은 우선 순위를 부여하여 실행할 수 있는 실시간 태스크가 하나도 없을 때 실행한다.

2.2 설계 및 구현

실시간 시스템에서는 기존의 범용 시분할 시스템과는 달리 시간 제약 조건의 만족을 위해 각각의 태스크가 종료시한과 우선 순위, 주기 등을 가지게 된다. 본 논문에서는 유연성이 높은 사건 구동형 실시간 시스템구조를 사용하여 자원에 대한 요구와 가용성에 대한 검사를 수행한다. 즉 태스크의 종료시한을 만족시키는 것이 실시간 시스템에서 주요한 설계 목표로 이를 만족하기 위해서 실시간 커널의 설계 시 다음과 같은 점에 주안점을 두어 다음과 같이 설계를 한다.

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. 수행 시간을 줄이기 위해서 먼저 인터럽트가 발생했을 때 빠른 시간에 인터럽트 핸들러를 불러주어 인터럽트 지연 시간을 줄여준다. 또한 프로세스에 대해서 우선순위를 부여하여 가장 우선순위가 높은 프로세스가 현재 프로세스가 되는 것을 보장한다. 그리고 현재 프로세스보다 높은 우선순위를 갖는 프로세스가 나타나면, 현재 하는 작업을 멈추고 이를 수행하며, 절대로 더 낮은 우선순위를 갖는 프로세스에 선점되지 않게 한다.

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.

라. 경성 실시간 제약조건을 만족할 수 있는 신뢰성을 제공해야 한다. 실시간 시스템에서의 신뢰성은 장기간동안 시스템이 오류 없이 계속 동작할 수 있는 것을 의미한다. 신뢰성은 또한 필요한 모든 자원을 미리 할당함으로써 경성 실시간 프로세스가 수행될 수 있도록 보장하는 것을 의미한다.

2.2.1 태스크 설계 및 통신

실시간 태스크에서 태스크는 주기적 태스크와 비주기적인 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 실행되는데 이러한 태스크는 비 동기적 태스크이다. 본 논문에서는 다음 그림 3 처럼 실시간 응용에 필요한 태스크에 대한 정보를 가진다.

본 논문에서는 경성 실시간 시스템의 종료시한을 만족시키기 위하여 주기적인 태스크를 우선 순위 기반 스케줄링 방법에 의해 스케줄링을 한다. 우선순위에 기반한 스케줄링 정책을 사용하는 경우, 우선 순위 역전(priority inversion)현상이 발생할 수 있다. 우선 순위 역전은 높은 우선 순위를 갖는 태스크가 낮은 우선 순위를 갖는 태스크에 의해 수행이 지연되는 현상을 말한다. 이러한 우선순위 역전 시간의 상한이 존재하지 않는 경우는 태스크의 종료시한 만족을 보장 할 수 없다. 따라서 이와 같은 상황을 막기 위해서는 같은 semaphore를 쓰는 태스

```
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 3. 실시간 태스크의 제어를 위한 구조체

크는 같은 우선 순위를 주거나, 우선 순위 계승을 이용한다. 또는 PCP(priority ceiling protocol)나 SRP(stack resource policy)와 같은 우선 순위 역전 시간을 한정할 수 있는 자원 관리 정책을 사용함으로써 예측가능성을 높일 수 있다[7-8].

본 논문에서는 아래 그림 4와 같이 4 가지의 태스크들로 구분하여 처리하도록 하였다.

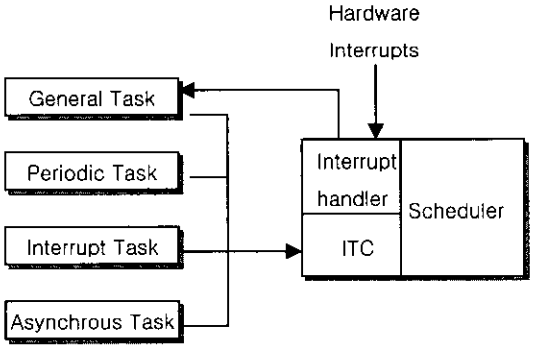


그림 4. 태스크 4가지 종류

그림 4 에서 일반적인 태스크(general task)는 비 실시간 커널에서 처리하는 태스크를 말한다[10]. 인터럽트 태스크(interrupt task)는 모든 인터럽트가 발생했을 때와 지역변수가 재 초기화될 때 생성하게 된다. 주기적 태스크(periodic task)는 주기적으로 장치들로부터 서비스를 서로 제공받도록 설계되었다. 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 사용한다.

ITC(Inter-Task Communication)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 신뢰도(reliability), 내용(Ccontent), 속도(speed), 호환성(portability)이 있는데 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, 호환을 어렵게 한다. 반면에 loosely coupled ITC는 공유메모리를 사용하지 않고 두 태스크간의 정보 전송을 위한 통신규약에 따라 수행된다. 이 경우에 호환성은 좋지만 상당한 오버헤드가 발생한다. 다른 분류방법으로는 정보에 데이터를 포함하고 있는 가에 따라서 Content ITC와 Non-Content ITC로 분류할 수 있다. Content ITC는 받는 태스크가 이벤트를 처리하는 데 부가적인 정보를 요구하는 것을 의미하고 Non-Content ITC는 받는 태스크가 이벤트를 처리하는 데 더 이상의 정보를 요구하지 않는 것을 의미한다.

실시간 커널에서는 태스크간의 동기화를 위해서 세마포어를 이용한다. 세마포어는 실시간 커널의 가장 기본적인 Non-Content ITC 방법이다. 세마포어를 구현하기 위해서는 atomic access를 가능하게 하는 커널 자료 구조체를 가지고 있어야 한다. atomic access 는 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.

2.2.2 실시간 태스크 스케줄링

본 논문은 우선 순위기반 스케줄링인 RMS를 사용하여 실시간 태스크 스케줄링을 하였다. 실시간 스케줄링과 비 실시간 스케줄링을 따로 수행한 후 이중 큐를 이용하여 스케줄링된 실시간 태스크와 비 실시간 태스크를 우선 순위를 부여하여 라운드 로빈 방식에 이중 큐의 개념을 적용시켰으며, 태스크의 누적으로 인한 오버헤드를 최소화하였다. 또한 이중 큐에서는 스케줄링 된 실시간 태스크들은 실행 큐인 상위 큐로 입력되고, 비 실시간 태스크들은

하위 큐로 입력되어 공백의 상위 큐로 입력되어 실행하게 한다. 만약 종료 시간을 지키지 못할 경우 하드웨어 인터럽트에 의해 종료되고 다음으로 가장 우선 순위가 높은 태스크가 실행하게 하였다. 그리고 RMS는 주기가 작은 태스크에 높은 우선순위를 할당하는 방법을 취한다. RMS는 먼저 다음과 같은 제약 조건을 가진다. 주기적 태스크를 Pi, Release Time를 Oi, Deadline를 Di, 주기를 Ti라 가정한다. 제약조건으로는 첫째 Pi는 서로 독립적이어야 한다. 둘째 Oi = 0 이고 Di = Ti 이다. 그리고 셋째로 Pi는 항상 선점 가능하다는 제약 조건을 가진다. 이러한 제약 조건 하에 RMS는 각 태스크 주기의 역수인 빈도율(rate)이 높을수록, 즉 주기가 짧을수록 더 높은 우선 순위를 부여하는 방식이다. 따라서 각 태스크의 주기가 주어지면 태스크들간의 우선 순위는 정적으로 결정될 수 있으며 이 우선 순위는 시스템이 수행되는 동안 고정된다. RMS 기법을 위하여 다음과 같은 스케줄링 분석 조건을 제시하였는데, 전체 프로세서 이용률(total utilization) U를 이용한다.

$$U = \sum_{i=1}^{n} \frac{e_i}{T_i} \le n(2^{1/n} - 1) \qquad \text{(식1)}$$

이를 이용하여 각 Pi가 Deadline을 만족시키는가를 수행 이전에 판단할 수 있다. 실시간 시스템 스케줄링은 두 단계를 수행한다. 먼저 우선 순위 할당을 수행한 후 다음으로 스케줄링 가능성 검사를 한다. 우선 순위 할당은 RM할당을 시행한다. 그리고 스케줄링 가능성 검사를 위해서 프로세서 이용률을 이용한다. 예를 들어 n개의 독립적인 태스크의 프로세서 이용률의 합이 U(n)보다 같거나 작으면 이 태스크 집합은 스케줄링 가능하다. 어느 특정 태스크의 스케줄링 가능성을 검사하기 위해서는 그 태스크와 함께 우선 순위가 높은 모든 태스크들의 이용률 값들을 더하여 검사한다.

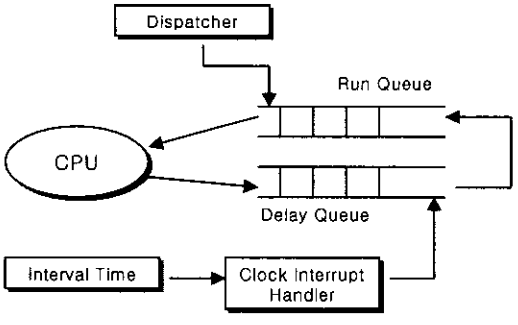


그림 5. 이중 큐를 이용한 정적 우선 순위 스케줄링 시스템 구성

2.2.3 이중 큐 스케줄링 알고리즘

우선 순위에 기반으로 한 정적 알고리즘 개념을 적용하여 라운드 로빈 방식에 이중 큐의 개념을 적용시키면 태스크 누적으로 인한 오버헤드를 최소화할 수 있다. 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.

이중 큐를 상위 큐와 하위 큐로 나누어 구현한다. 상위 큐를 실행 큐 라하고 하위 큐를 대기 큐라 할 때, 그림 2에서와 같이 하위 큐에는 스케줄링 된 비실시간 태스크가 대기하게 된다. 그리고 상위 큐에는 우선 순위 스케줄러에 의해 스케줄 된 실시간 태스크가 실행된다. 실시간 태스크가 더 이상 상위 큐에 존재하지 않을 때 비실시간 태스크가 상위 큐로 들어와 실행하게 된다. 비실시간 태스크가 종료 시간을 지키지 못할 경우 인터럽트 제어권을 가진 실시간 커널에서 인터럽트를 수행하게 된다. 그러나 기존의 비실시간 커널은 인터럽트에 의해 중단되고 싶지 않은 경우, 하드웨어 인터럽트를 막을 수 있다. 이렇게 인터럽트를 막아버리면 그 사이 발생한 인터럽트는 인터럽트가 허용될 때까지 기다리고 있어야 하고, 이는 인터럽트 반응 속도를 느리게 하는 요소가 된다. 따라서 실시간 커널은 비실시간 커널이 인터럽트를 막는 것을 소프트웨어적으로 에뮬레

```
DUqueue_Scheduler( )
{
 loop
   if( Task == Real_time_task)
        push  Run_queue;
        Execute();
   else
        push  Delay_queue;
   if ( Run_queue == Empty )
        Run_queue = Delay_queue;
        Execute();
   Deadline();
   if (Execute_time ≤ Deadline() )
     Hardware_interrupt();
 end loop
```

그림 6. 스케줄링 알고리즘

이션하고 실제로 인터럽트를 막을 수 없도록 한다.

그림 6에서 보는 것처럼 우선 순위 스케줄링에 의해 스케줄링 된 실시간 태스크들은 실행 큐인 상위 큐로 입력되고, 비실시간 태스크들은 하위 큐로 입력되어 상위 큐가 비었는지를 검사하여 비었으면 상위 큐로 입력되어 실행하게 된다. 만약 종료 시간을 지키지 못할 경우 하드웨어 인터럽트에 의해 종료되고 다음으로 가장우선 순위가 높은 태스크가 실행된다.

Ⅲ. 성능 및 평가

본 논문에서 제안한 실시간 커널에 대한 성능평가를 위해 다른 상용 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.

본 논문에서의 실험은 실시간 시스템 전체에 대한 실험이 아니라 이중 큐를 이용한 스케줄링에 대한 부분만을 대상으로 간단한 커널로서 실험을 하였다. 따라서 실험환경을 최저 환경 하에서 수행하였다. 본 논문과 가장 유사한 실시간 태스크와 비실시간 태스크를 함께 처리하는 RT-Linux 와 성능평가에서 보다 나은 성능 향상을 보였으며, 기존의 실시간 시스템과 비교하여 성능 향상의 이유는 실시간 태스크와 비실시간 태스크가 각각 보다 빠른 고정 우선 순위 기반 알고리즘을 사용하였으며, 스케줄링된 실시간 태스크와 비실시간 태스크를 이중 큐를 이용하여 우선 순위를 부여하여 스케줄링 하므로 그 수행속도가 빠르다는 장점이 있다.

3.1 인터럽트 지연 시간

그림 7과 같은 인터럽트 지연시간은 실시간 시스템과 응용프로그램이 실행 중에 인터럽트가 발생하면 해당 인터럽트서비스 루틴의 첫 코드가 실행될 때까지 걸리는 지연 시간을 말한다. 이 지연시간은 주로 실시간 시스템 내부 코드 실행 시에 행하는 인터럽트 Disable과 Enable설정 때문에 생기는 것으로 그 중 가장 긴 인터럽트 Disable Time을 사용한다. 인터럽트 지연시간은 외부 사건 발생에 대응한 태스크의 응답 속도를 결정하는 중요한 요인이 된다.

실시간 태스크에서 인터럽트의 제어권은 실시간

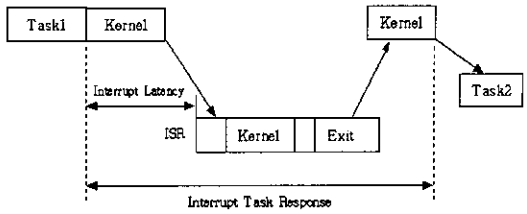


그림 7. 인터럽트 지연시간

커널에 있다. 기존 비실시간 커널은 인터럽트에 의해 중단되고 싶지 않은 경우, 하드웨어 인터럽트를 막을 수 있다. 이렇게 인터럽트를 막아버리면 그 사이 발생한 인터럽트는 인터럽트가 허용될 때까지 기다리고 있어야 하고, 이는 인터럽트 반응 속도를 느리게 하는 요소가 된다. 이에 본 논문에서는 실시간 태스크는 비실시간 커널이 인터럽트를 막는 것을 소프트웨어적으로 에뮬레이션을 하고, 실제로 인터럽트를 막을 수 없도록 한다. 비실시간 커널이 인터럽트를 막으면 별도의 플래그에 이를 표시해 두었다가, 실제로 인터럽트가 발생하면 이 플래그를 참조하여 인터럽트 가능 상태이면 인터럽트를 전달하고, 인터럽트 금지 상태이면 인터럽트가 발생했다는 표시를 하고, 인터럽트를 허용할 때까지 전달하지 않는 것이다. 이렇게 하여 비실시간 커널의 입장에서는 자신이 인터럽트 되지 않는 효과를 주고, 실제로 실시간 인터럽트를 바로 처리할 수 있도록 한다. 이렇게 함으로써 인터럽트 지연시간을 줄일 수 있다. 본 논문에서는 실행 중에 인터럽트가 발생하면 해당 인터럽트서비스 루틴의 첫 코드가 실행될 때까지 걸리는 지연 시간인 인터럽트 지연시간을 측정하여 비교하였다. 인터럽트 지연시간의 최대 값을 비교하기 위해서 인터럽트 요청 신호를 보내고 난 후에 인터럽트가 응답하는 시간을 측정하였다.

그림 8 처럼 RT-Linux는 34.0μs, QNX 는 31.2 μs, 본 논문에서 제안한 실시간 커널은 29,8μs 측정

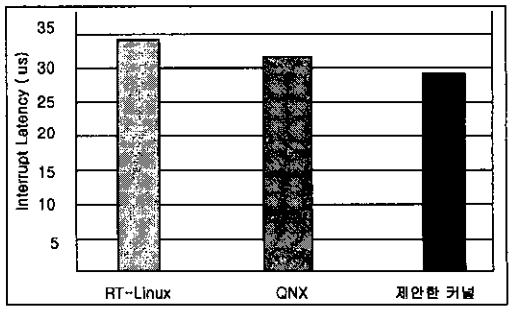


그림 8. 인터럽트 지연시간 측정

됨으로써 다른 실시간 커널보다 인터럽트 지연시간이 짧음을 볼 수 있다.

3.2 스케줄링 정확성

주기적인 실시간 태스크 수행의 스케줄링의 정확성을 측정하기 위해 각 태스크가 Wake-up 할 때마다 시간을 측정하고 평가하였다. 측정한 결과 그림 9 에서 처럼 RT-Linux는 64.0㎲, QNX는 14.8㎲, 본 논문에서 제안한 실시간 커널은 13.9㎲로써 스케줄링 정확성이 다른 실시간 커널보다 빠름을 볼 수 있다.

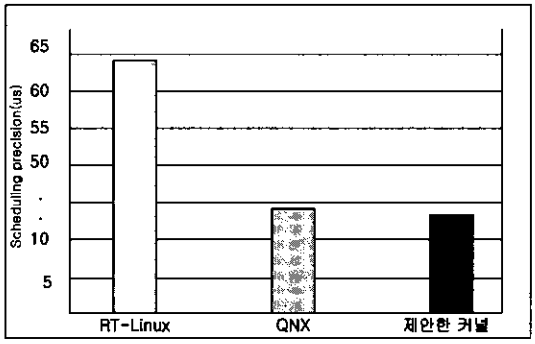


그림 9. 스케줄링 정확성 측정

3.3 메시지 전달시간

생산자 프로세스가 Send 함수를 호출한 후 소비자 프로세스가 메시지를 읽어 reseive 함수를 빠져나올 때까지의 시간을 측정하였다. 그림 10 에서처럼 RT Linux는 20.5㎲, QNX 는 11.1㎲, 본 논문에서 제안한 실시간 커널은 9.8㎲로써 메시지 전달 시간이 다른 실시간 커널 보다 빠름을 볼 수 있다.

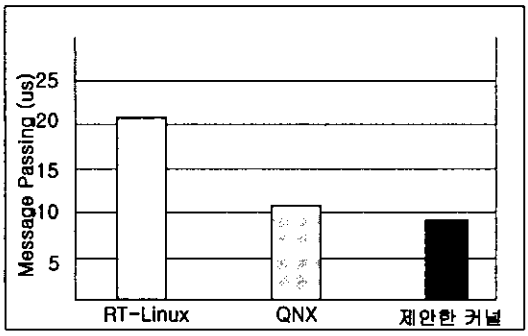


그림 10. 메시지 전달 지연시간

앞서 본 실험 항목들에 대한 측정 결과들을 아래와 같이 하나의 도표로 나타내었다.

표 1. 전체 측정 결과표

측정 대상 / 측정 항목	RT Linux	QNX	제안한 커널
인터럽트 지연 시간	34.0	31.2	29.8
스케줄링 정확성	64.0	14.8	13.9
메시지 전달시간	20.5	11.1	9.8

표 1에서 보는 바와 같이 본 논문에서 제안한 실시간 커널이 다른 실시간 커널 보다 성능이 향상되었음을 볼 수 있다.

Ⅳ. 결 론

기존의 시분할 시스템은 시간적 제약조건을 만족하기에 많은 어려움이 있었다. 이러한 문제점을 해결하기 위해 실시간 시스템은 논리적이고 기능적인 특성, 시간적인 제약까지도 중요한 요소로써 고려하여 설계하고 있다 따라서 본 논문에서는 시간적인 제약 조건을 만족하면서 경성 실시간 시스템의 필요 조건인 높은 예측성, 빠른 응답성, 짧은 인터럽트 지연시간, 스케줄러의 정확성, 메시지 전달 시간 등을 만족하도록 비실시간 커널과 공존하도록 실시간 커널에서 제공해야 하는 기능들을 설계 및 구현을 하였다. 실시간 커널은 실시간 요소(인터럽트처리, 타이밍)를 가지는 기능들을 처리할 수 있도록 하였고, 비실시간 커널에서는 일반적인 기능을 처리하도록 하였다. 그리고 두 영역간에 데이터 공유를 위하여 이중 큐라는 연결 통로를 구현하였다.

그리고 기존의 RT Linux, QNX지연시간, 스케줄링의 정확성, 메시지 전달 시간을 비교 분석함으로써 성능 향상을 보였다.

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

참 고 문 헌

[1] S. Evanczuk, "Real Time O.S", Electronics, pp 105-115, Mar 1983.

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

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

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

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

[6] Daniel Stodolsky, J. Bradley Chen, and Brian Bershad. Fast interrupt priority management in operating system kernels. In proceedings of the 2nd USENIX Symposium on Microkernels and Other kernel Architectures. USENIX, September 1993.

[7] T. Baker, "A Stack-based Resource Allocation Policy for Real-Time Processes", Proceedings of Real-Time System Symposium, pp. 191-200, 1990.

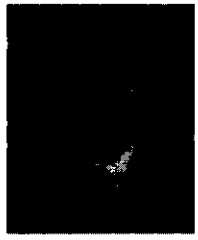
[8] L. Sba, R. Rajkumar, and J. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization", IEEE Trans. on Software Engineering, Vol. 39, No. 9, pp. 1175-1185, 1990.

[9] C. Lin and J. Layland, "Scheduling Algorithm for Multiprogramming in a Hard Real-Time Environment", The Journal of the ACM, Vol. 20, No. 1, pp. 46-61, 1973.

[10] Ori Pomerantz, "Linux Kernel Module Programming Guide", Version 1.1.0, 26 April 1999.

[11] Michael Barabanov, "A Linux-based Real-Time Operating System", New mexico, June 1, 1997.

인 치 호(Chi-ho Lin) 정회원



1985년 : 한양대학교 전자공학과 공학사
1987년 : 한양대학교 대학원 공학석사
1996년 : 한양대학교 대학원 공학박사
1992년~현재 : 세명대학교 컴퓨터과학과 부교수

<주관심 분야> VLSI CAD, ASIC 설계, CAD 알고리듬, RTOS 및 내장형 시스템