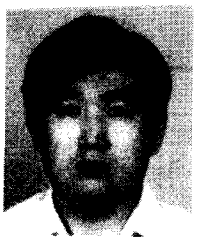
〈技術解說〉



실시간처리 시스템을 위한 Real Time Executive 설계관점 해석

A Design Perspective for Real Time Executive in Real Time System

金 正 鎬*
Kim, Joung-Ho

목 차

1. 서 론

컴퓨터를 이용한 적용 분야는 종래의 데이타 처리를 위한 시스템으로 부터 다양한 환경을 위한 시스템으로 그 응용 범위가 넓혀지고 있다. 특히, 컴퓨터 기술의 비약적인 발전과 하드웨어가격의 지속적인 하락 등에 기인하여 과거에는 다룰 수 없었던 실시간 처리를 효율적으로 처리할 수 있게 되었다. 이와같은 실시간 처리 시스템은 제한된 시간내에 정확한 동작을 수행해야 함은 물론 여러 공정의 처리, 외부신호나 예기치 못한 사건 등에 대한 시스템의 적절한 응답 등의 컴퓨터가 대응할 수 있게 관장하는 소프트웨어가 필요하며 이러한 소프트웨어는 결국 실시간 처리 시스템의 성패를 가름한다.

실제 실시간 처리를 위한 소프트웨어 구현은 단일(single monolithic)프로그램과 시스템 소프트웨어를 이용한 멀티태스킹(multi-tasking) 프로그램으로 구분된다. 어떤 공정에서 필요로 하는 모든 기능들을 사용자가 일률적으로 열거, 기술하는 단일 프로그램에서는 시간에 관련된 부분을 분리 동작시키기란 매우 어려우나 기존 시스템 소프트웨어에서 제공하는 실시간 기능을 이용하면 각 태스크(task)를 독립적으로 수행, 관리하기가 매우 용이하여 여러 복잡한 외부 신호 처리를 위한 시스템에서는 이와같은 멀티태스킹 프로그램 방식을 채택하고 있다. 따라서 시스템 자원의 효율적인 관리, 외부 인터럽트의 처리 등을 시스템 소프트웨어에 의하여 구현하며 이같은 시스템 소프트웨어를 실시간 운영체제(OS)라 칭하며 만일 실시간

* 電子計算組織應用技術士, 工業計測制御技術士, 韓國電子通信研究所先任研究員

환경이 비교적 단순하여 하나의 보드에 의하여 처리될 수 있는 경우 위에 언급한 실시간 OS를 사용하지 않고 실시간 OS의 커널(kernel) 부분에 해당하는 RTE(Real Time Executive)를 이용함으로써 실시간 OS에서 제공하는 대부분의 기능을 이용함은 물론 적은 규모의 경제적인 시스템을 개발할 수 있다.
따라서 이러한 RTE는
-다양한 실시간 기능 제공
-사용하기 쉬운 고급 언어에 의한 접근 기능
-이식성 및 손쉬운 구현
-멀티태스킹 기능
등과 같은 특성을 제공하여야 한다.

이와같은 기능을 가지는 RTE는 특정 마이크로프로세서의 제작회사에서 해당 프로세서에 맞는 RTE를 제공하거나 프로세서와 무관한 독립된 공급처에서 제공하는 소프트웨어가 있다. 물론 각 제품별로 고유 특성을 가지고 있지만, 사용자의 입장에서는 개발하려는 시스템에 적합하게 수정, 보완, 구현할 수 있어야 한다. 그러나 기존의 보드레벨 제품에 대해서 확실히 파악하지 않거나, 기술적으로 미흡할 경우는 기존 소프트웨어 패키지의 소스를 구입하여 다시 수정, 구현하여야 하는 어려움이 따르게 된다. 이러한 작업은 위험 부담이 크며 시간의 낭비, 개발비용의 상승등 매우 비합리적이므로 엔지니어가 새로운 하드웨어를 설계할때 표준 규격의 보드들을 선택하는 것과 마찬가지로 소프트웨어 역시 부품화된 실리콘 OS를 이용하기도 한다.

2. 실시간 처리 시스템의 요구사항

소프트웨어 개발에 따른 시스템의 요구사항은 적용환경에 따라 다르다. 특히 실시간 환경에 요구되는 시스템의 기능은 비동기적으로 발생되는 다양한 외부 사건에 대한 감시 및 제어를 들 수 있다. 특정 분야에 대한 소프트웨어 개발에서 반복수행기능, 저레벨 인터페이스 작업, 효율적인 자원의 관리, 용이한 소프트웨어 구현 등과 같은 일련의 기능은 응용 프로그램 내에서 제공할 필요가 없이 RTE에 의존함으로서 시스템을 효율적으로 개발할 수 있다. RTE의 주요 기능으로는 태스크 우선순위에 따른 시스템 자원의 관리, 태스크 간의 데이타 교환, 인터럽트에 의한 주변 장치 제어, 실시간 클럭제어, 인터럽트 핸들링 등을 들 수 있다. 결국 실시간 응용을 위한 소프트웨어 설계에서는 콘커런시(concurrency), 우선순위, 동기화, 태스크간의 데이타 교환 등에 부합되는 기능을 가지고 있어야 된다. 초기의 RTE는 개발 단계에 있어서 특정 호스트 시스템을 필요로 하였으며 한정된 목적 시스템에서만 수행가능 하였다. 따라서 응용 분야에 따라 수정, 보완해야 되는 부분들은 기능별로 독립된 표준 인터페이스를 이용하여 해당 인터페이스 부분만을 변경시킴으로써 목적 시스템에 적용시킬 수 있다.

표준 인터페이스는 OS의 외부 동작을 나타내는 다음의 네가지 인터페이스로 분류할 수 있다.

-Application Interface
OS에서 제공하는 system call sets에 의하여 정의되며 태스크 제어용 서비스 그룹과 입출력 그룹으로 나눌 수 있다.

-User Interface
사용자 터미날에서 입력되는 명령어들과 명령어 해석 프로그램에 의하여 정의된다.

-Media Interface
데이타/프로그램을 교환하기 위하여 시스템간에 이송되는 매체의 형태(format)에 따라 정의되지만 실시간 OS에서는 해당되지 않는다.

-Device Interface
OS에서 입출력 동작을 수행하기 위하여 입출력 장치나 각 장치에 특정된 루틴들을 억세스하는 명령어 집합에 따라 정의되며 주문형 장치 혹은 비표준장치들의 인터페이스에

따라 달라질 수 있다.

가. 응용 프로그램의 요구 사항

RTE에 의하여 동작 시키게 되는 응용프로그램이 가져야 하는 특성은 다음과 같다.

-Tasking Method

태스크는 제어동작을 두 가지 이상의 한정된 제어 부분으로 나누는 방법을 말한다. 이러한 각 부분은 태스크라 불리우며 각 태스크는 특정 제어 기능을 구현하기 위하여 설계된 제어 프로그램으로 구성된다. 실시간 시스템에서 태스크는 기본적이며 매우 중요한 개념이다. 태스크는 '논리적으로 완벽한 프로그램 세그먼트' 혹은 '시스템 공유자원(입출력 장치, CPU, 메모리…)을 할당받을 수 있는 최소한의 프로그램' '동작 상태에 있는 프로그램의 집합' 등과 같이 정의된다.

-Task Environment

태스크는 단일태스크(single-task)와 멀티태스크(multi-task)로 나누어지며 START에서 END로 끝나는 단일태스크에서는 START-END 사이에 몇몇 루틴들이 존재하게 되어 동작될때는 어느 특정경로를 통하여 동작된다. 이와는 대조적으로 그림 1과 같이 멀티 태스킹 환경에서는 독립된 기능을 가지는 태스크들이 동시에 존재할 수 있으며, 각 태스크는 서로 독립적이며 논리적으로 완성된 기능을 가진다. 또한 여러 태스크가 갖는 목표는 첫번째의 특정 부분을 수행하는 것과 프로세서를 독점하여 첫번째의 목표를 진행시키는 데 있다.

나. 실리콘 소프트웨어 설계시 고려 사항

일단 소프트웨어를 실리콘에 내장시키면 그 내용을 변경할 수 없게 되며 이를 수정보완하여 원하는 시스템에 이용하기 위해서는 새로운 하드웨어나 응용분야에 맞게 변경시킬 필요가 있다. 따라서 실리콘 소프트웨어 설계시 고려

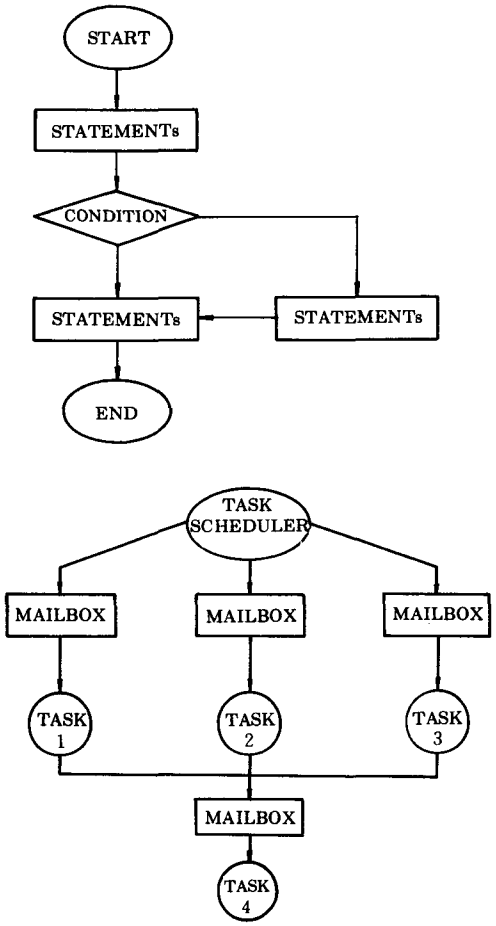


그림 1. 단일태스크와 멀티태스크의 비교
(Single-task와 Multi-task의 비교)

할 점은 다음과 같다. 실리콘 소프트웨어는 두 종류의 코드로 나눌 수 있는데 on chip code와 off chip code가 그것이다. 특별한 경우에 있어서는 off chip code의 일부가 on chip code와 매우 밀접한 관계를 가질 수 있으며 실리콘 소프트웨어 패키지의 한 부분으로 개발되기도 한다. 이와같은 기능을 갖는 off chip code에는 initialization, interface, system version update code 등이 있으며 실리콘 소프트웨어를 한 시스템 이상에서 사용하거나 변경에 무관한 기능을 부여하기 위해서는 position independ-

ence, configuration independence, stepping independence와 같은 특성을 지녀야 한다.

-Position Independence

최근의 마이크로프로세서는 최소한 1 메가 바이트의 메모리를 가리킬 수 있기 때문에 실리콘에 내장된 시스템 소프트웨어는 메모리 내의 위치에 무관하게 동작되어야 한다. 따라서 read only, on chip code/data의 절대번지(absolute address)는 시스템 구성에 제한을 준다. On chip code는 offset번지만 인식하므로 절대번지는 처리될 수 없게된다. 그러므로 on chip code가 position independent하게 되면 on chip code에서 지정하는 절대번지는 프로세서의 레지스터를 통하여 얻을 수 있다. 컴파일러나 어셈블러에서는 position dependent code를 만들게 되지만 linking, locationg 단계를 거치게 되면 쉽게 position independent code를 만들 수 있다.

-Configuration Independence

두번째 요구사항으로서 on chip code가 시스템의 기존 하드웨어와 소프트웨어 구성에 무관하여야 된다. 결국 on chip code는 간접적으로 다른 코드나 데이타를 억세스하는 대신 시스템 구성을 유추하기 위해서는 수행시간 데이타를 검지해야 한다. 실리콘 OS의 read only 성질로 말미암아 on chip code 내에 상수가 존재할 경우는 문제가 될 수 있다. 즉, 시스템의 어느 곳이나 코드내에 상수가 존재할 경우는 문제가 될 수 있다. 즉 시스템의 어느 곳이나 위치할 수 있는 하드웨어 장치의 값 혹은 유사한 기능을 갖고 있지만 서로 다른 프로그래밍 인터페이스를 수행하는 장치의 값들은 on chip code에 내재해서는 안된다. 결국 시스템 구성에 영향을 줄 수 있는 모든 값들은 간접적으로 억세스되어야 한다.

-Stepping Independence

이것은 configuration independence의 확장으로서 소프트웨어가 실리콘에 내장되기 위한 중요한 요구사항이다. 'Step'이란 on chip code와 off chip code는 각 코드들이 서로 변경되더라도 호환적이어야 한다. 즉, 모든 on chip code version이 off chip code version과 일치하게 동작할때 stepping independence가 있게 된다.

3. RTE 수행과 구조

가. RTE의 수행

마이크로프로세서용으로 설계된 대부분의 RTE는 프로세서가 갖는 속도의 기능을 최대한 활용하도록 실시간 멀티태스킹을 수행한다. RTE는 두 가지의 파라미터로서 규정 지을 수 있는데 태스크간의 수행 전환을 위한 트리거 메카니즘(trigger mechanism)과 태스크들을 수행하기 위한 스케쥴링(scheduling)이 그것이다. 트리거 메카니즘에는 인터럽트 드라이브와 태스크 드라이브 방식이 있으며 인터럽트 드라이브 방식에서는 한 태스크에서 다른 태스크로의 수행 변환(context switching)이 소프트웨어나 하드웨어에서 야기되는 인터럽트에 따라 결정된다.

태스크 드라이브에서의 수행 변환은 한 태스크의 수행이 종료되었을 때만 다음 태스크로의 전환이 일어난다. 태스크 수행을 위한 스케쥴링 방법에도 라운드로빈(round robin) 방식(순환방식이라고도 함)과 우선 순위(priority) 방식이 이용되며 라운드로빈 방식에서는 한정된 시간 간격동안 태스크가 수행되며 각 태스크들은 순환방식으로 스케쥴링 된다. 이와는 반대로 우선순위에 의한 스케쥴링 방식에서는 각 태스크들이 우선순위를 부여받게 되며 한 태스크에서 다음 태스크로 변환하기 전에 스케쥴러가 태스크의 우선 순위를 확인한다.

따라서 CPU를 사용하는 가장 높은 우선 순위의 태스크는 다음 태스크의 우선 순위가 자기 자신 보다 낮을 경우 CPU를 계속해서 사용하게 된다. 그림 2 에서 보면 인터럽트 구동

에 의한 시스템 (A)은 인터럽트가 발생할 경우 우선 순위가 높은 태스크가 먼저 수행되며 태스크 구동에 의한 시스템 (B)의 각 태스크는 순차적으로 수행되며 진행중인 태스크가 끝났을 경우 다음 차례의 태스크가 스케쥴러에 의하여 수행된다.

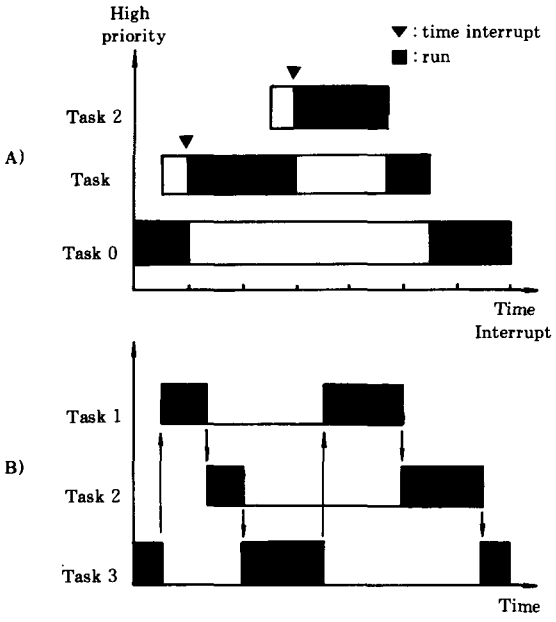


그림 2. 우선순위방식과 라운드로빈방식의 스케쥴링

나. RTE의 구조

RTE의 개별적인 구성을 언급하기에 앞서 기본적으로 갖추어야 할 주변 장치들에 대한 입출력 동작 처리를 기술 한다. 입출력 요청은 사용자 태스크에서 발생되며 시스템에서는 이러한 요청에 따라 입출력의 시작, 데이타 전송 등과 같은 기능을 처리한다. 만약, 입출력 장치를 동시에 여러 태스크들이 요청할 경우 이 장치는 아무런 서비스를 제공할 수 없으므로 여러 태스크들에 순서를 할당하여 효율적으로 입출력 요청을 처리 시키기 위한 스케쥴러가 필요하다. (그림 3 참조)

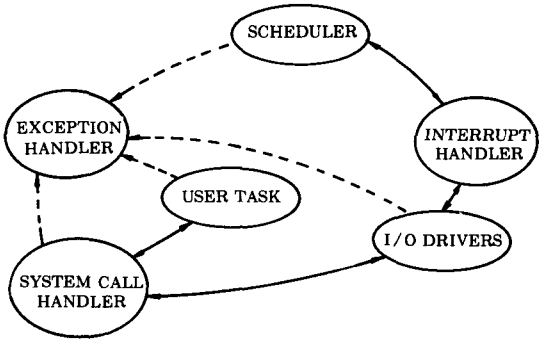


그림 3. RTE의 전체 구조

만일 태스크에 의한 수행에서는 하나의 태스크가 완전히 끝난후에 다음 태스크가 수행되므로 데이타 전송에 소요되는 시간적 낭비가 많으나 멀티태스킹, 인터럽트 드라이브되는 환경에서는 입출력 동작을 위하여 기다리는 동안에 다른 태스크를 수행 시켜줄 수 있으므로 효율을 높일 수 있다. 물론 각각 주변 장치들의 상태를 감지해야하는 부담이 수반되지만 크게 영향을 주는 요인이 되지 못한다.

RTE의 중요한 다른 기능으로서 태스크간의 데이타 교환 기능을 들 수 있다. 태스크들은 서로 독립된 프로그램으로 존재하지만 그림 4 와 같이 데이타를 공유하거나 교환할 수 있는 장치가 필요하게 되며 태스크들간의 데이타 교환은 'Messages'로서 이루어진다. 태스크들간의 데이타교환은 두 가지의 데이타구조인 다이나믹 버퍼(Dynamic Buffering)와 글로벌 공통 메모리 변수들(Global Common Memory Variables)로서 구성된다.

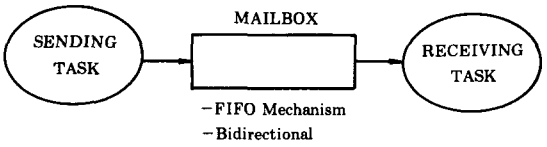


그림 4. 메세지 교환(채널)

다이나믹 버퍼방식에서는 한 태스크에서 다음 태스크로 메세지를 보낼때 RTE 루틴을 호출하며 RTE에서는 버퍼를 선택하여 수신할 태스크에 버퍼의 위치만 알려준다. 이는 태스크간의 정보전달에 있어 효율적인 방식이 된다. 글로벌 공통 메모리 방식에서는 태스크들로 부터 접근 가능한 공통 메모리내에 버퍼를 만들게되며 이러한 버퍼는 명령어, 상태, 데이타 영역으로 구분지을 수 있다. 각 버퍼는 지정된 소스 태스크와 데스티네이션 태스크간 메세지 전송을 위한 것이므로 태스크들간에 한정된 버퍼만 할당되어 글러벌 공통 메모리 방식은 다이나믹 버퍼 방식보다 더 많은 메모리 영역이 필요하지만 태스크에 대한 큐잉 메세지나 버퍼 할당을 위한 RTE의 특정 루틴이 없어도 된다. 따라서 RTE 설계시 태스크간의 통신을 위한 글로벌 공통 메모리 방식이 쉽게 적용될 수 있다. 이상에서 언급한 입출력 처리와 태스크간의 통신이외에 RTE를 구성하고 있는 기본 요소로는 다음과 같다.

1) Real Time Clock Handler

RTE에서는 우선 순위가 가장 높은 태스크의 서비스를 위하여 반복적으로 시스템의 상태를 점검하며 이러한 동작은 단순하면서도 짧은 시간 동안 처리된다. 따라서 한 태스크가 CPU를 점유하고 있는 동안은 RTE가 CPU를 제어할 수 없으므로 실시간 클럭이나 프로세서에서 지원하는 인터럽트 시스템에 있어서 이와같은 기능은 필수적이다. 실시간 클럭은 세분된 타임 인터럽트들을 계속 발생시켜야 되며 이때 인터럽트후 소프트웨어에서 리세트 시켜야 되는 프로그래므블 타이머의 인터벌타이머와는 다른다. 수행중인 태스크에 있어서 진행 과정속에 임의의 인터럽트가 발생할 수 있다. 그렇지만 태스크의 수행결과는 인터럽트 발생과 무관하며 RTE에서는 인터럽트 발생시 태스크 전환에 따른 모든 정보나 CPU의 상태 등을 저장시킨다. 실시간 인터럽트를 위한 시간 간격은 하드웨어에 따라 다르지만 간격이 너무 짧을 경우는 RTE가 대부분의. CPU 시간을 차지하게 되어 시스템의 효율이 떨어지며 길 경우는 우선 순위가 높은 태스크들이 CPU의 제어를 받기가 힘들게 된다 일반적으로 10ms 이하의 시간 간격이 채택되고 있다.

2) Scheduler

Scheduler는 RTE와 사용자의 응용 프로그램 사이의 긴밀한 동작을 처리시켜 주며 이의 기본 기능으로는 다음과 같다. 먼저, 사용자나 시스템 태스크에서 어떤 태스크가 가장 높은 우선 순위를 갖고 있는지를 결정하고 태스크의 상태를 전환시킨다.

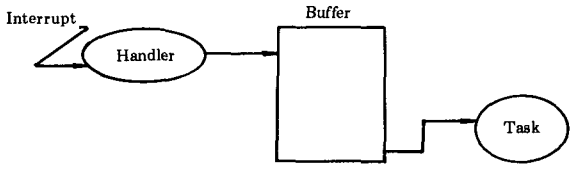
즉

- CPU의 제어를 요청하는 우선 순위가 높은 태스크의 결정
- 태스크들의 상태 전환
- 태스크 데이타 영역의 저장 및 복구
- 태스크 레지스터와 스택 포인터의 복구

태스크 스케쥴러는 자주 수행되므로 'ZERO-th' 태스크로 볼 수 있으며 스케쥴러의 제한된 부분들은 다시 수행되지 않는 영역으로 인터럽트들을 디스에이블시키면서 수행되기도 한다. 태스크 스케쥴러는 보통 RTE에서만 사용가능한 하나의 엔트리 포인터를 가지고 있다. 태스크 스케쥴러가 인터럽트를 받게되면 이 엔트리 포인터에서 시작되며 레지스터의 내용을 저장하거나 인터럽트 포인터로 부터 태스크 스케쥴러를 재 수행시킬 필요는 없다.

태스크 스케쥴러의 설계는 일반적으로 두 가지 종류로 구분 되는데 preemptive와 non preemptive로 나눈다. Preemptive 스케쥴러 하에서 동작되는 태스크는 인터럽트 핸들러에 의해 가장 높은 순위의 태스크로 전환되기 위하여 서스펜드 된다. 즉 인터럽트 핸들러에서 받은 데이타는 핸들러에 의하여 전환된 태스크가 즉시 처리해야 되며 이와같은 방식의 스케쥴러에 있어서 그림 5 와 같이 인터럽트 처리

를 위한 데이타 버퍼링이 싱글인 경우 데이타가 처리되기 전에 또 다른 데이타에 의하여 반복기록 될 수 있으므로 더블 혹은 멀티플 버퍼링으로 수시로 발생할 수 있는 인터럽트에 대한 태스크 처리를 효율적으로 다룰 수 있다.



-버퍼가 채워지면 핸들러에서 태스크를 구동시킨다.
-핸들러와 태스크는 버퍼를 서로 공유함으로서 정보 교환을 한다.

그림 5. 버퍼링 개념

Non preemptive 스케쥴러하에서 동작되는 태스크는 우선 순위가 높은 태스크가 생기더라도 서스펜드되지 않지만 인터럽트 서비스를 위해 동작하는 태스크가 스스로 자신을 서스펜드 할 때 까지는 계속 CPU를 점유할 수 있다. 따라서, non preemptive 스케쥴러를 적용한 시스템에서는 태스크가 임의로 중지되는 과정에서 주어진 프로세서의 대역폭내에 태스크 처리량을 제한함으로서 태스크들 사이의 상호 동작을 원만히 꾀할 수 있다. 결국 시간적인 제한요소가 없는 실시간 응용 분야에 있어서는 이와같은 방식의 스케쥴링이 보편적으로 이용되고 있으며 스케쥴러의 설계를 간단히 할 수 있고 메모리, 수행시간에 따른 제한 요인도 줄일 수 있다. 특히, non preemptive 스케쥴링의 큰 장점으로서는 태스크 전환에 따른 제한요인도 줄일 수 있다. 특히 non preemptive 스케쥴링의 큰 장점으로는 태스크 전환에 따른 프로그램간에 전달될 수 있는 일치되지 않은 데이타의 발생을 줄일 수 있다.

Preemptive 스케쥴러에서 인터럽트 핸들러가 우선 순위가 높은 태스크를 구동시키려 할 때 메세지에 있는 데이타의 일부를 처리하게 되며 원래의 태스크가 처리하던 메세지의 내용이 바뀔 수도 있다. 결국 스케쥴러 설계시 또 다른 서비스 메카니즘이 수반되어야 한다. 따라서 non preemptive 방식의 스케쥴러가 쉽게 설계될 수 있으며 보편적으로 사용된다.

태스크 전환을 위해 스케쥴러에서는 몇 가지의 테이블을 사용하고 있으며 이러한 테이블은 task control block(TCB) (혹은 task data block(TDB), task status table(TST))라 불리우며 특히 링크연결된 데이타 구조를 갖는 TCB에서는 스케쥴러가 TCB를 억세스하기 위한 포워드 링크 포인터와 태스크가 시작되는 번지값, 스택 포인터, 상태 플래그 등을 포함하여 링크 연결된 데이타 구조를 이용하여 태스크의 우선순위를 결정하거나 스케쥴링 동작을 손쉽게 처리한다.

스케쥴러가 태스크를 인식하는 것으로 다음의 세가지 방법이 있다.

첫째, 시스템 호출로서 활성화되지 않는 상태에 있는 태스크가 대기상태로 전환된다.

둘째, 데이타의 입출력 처리를 위한 사용자 태스크 요청을 인식한다.

세째, 시스템의 공유자원 요청에 따른 인터럽트 발생 등으로 나눌 수 있다.

3) System Call Set

RTE에서 동작되는 모든 기능을 시스템 명령어 형식으로 나누어 사용자로 하여금 쉽게 시스템을 파악하고 이용할 수 있으며 각 시스템마다 고유의 기능을 위한 특정 명령어들을 제공한다. 이와같은 시스템 명령어들을 기능면에서 분류하면 다음과 같다.

-입출력 명령어
-태스크 생성/삭제 명령어
-태스크간의 데이타 교환용 명령어
-클럭 명령어
-태스크 인식 명령어
-태스크/오퍼레이터 통신용 명령어

(1) 입출력 명령어

사용자 태스크에서 특정 장치로의 데이타 전송을 위해 요청하는 시스템명령어로서

-입출력 장치의 정보
-전송 데이타의 수
-전송 채널 수
-데이타의 코딩
-메모리에서 데이타의 소스/데스티네이션
-정보에 관계되는 입출력 동작

(2) 태스크 생성/삭제 명령어
태스크를 스케쥴링에 인식시키며 태스크의 상태를 전환시키는 명령어로서
-태스크 인식 번호
-태스크의 우선 순위
등과 같은 정보를 통해 스케쥴러가 시스템의 고유자원을 태스크에 할당한다.

(3) 태스크의 데이타 교환용 명령어
태스크간의 메세지 전송을 위한 것로 태스크들 사이에 동기가 이루어질때 소프트웨어에서 정의된 커널을 통하여 이루어 진다

(4) 클럭명령어
실시간 클럭을 위한 시간 간격을 결정하며 타이머 인터럽트를 발생한다.

(5) 태스크 인식 명령어
태스크 생성/삭제 명령어와는 달리 태스크의 ID 만 바꿀 수 있다.

(6) 태스크 오퍼레이터 통신 명령어
태스크들과 오퍼레이터간의 메세지 전송을 위한 것으로 실시간 공정-오퍼레이터 간 통신에 유용하다.

4) Exception Handler

RTE 설계시 필히 다루어야 되는 것으로 동작 중 에러 발생에 응답하여야 한다. 즉 RTE는 절대로 멈추거나 오동작을 일으켜서는 안되므로 존재치 않는 메모리의 위치를 지정하거나 시스템 응용에서 제한된 메모리의 위치를 사용할 경우 이러한 에러를 검지하여야하며 우선순위나 잡(job) 지정이 타당한가에 대해서 필히 검지하여야 한다. 레이턴시(latency) 문제에 있어서도 CPU 시간을 요구하는 프로그램이 너무 많을 경우 공정이나 사용자에 의해서 지정되는 시간 제한이내에 critical jobs들을 처리할 수 있어야 하므로 지연 문제 역시 고려되어야 한다. 결국, 에러 처리는 실시간 컴퓨터 시스템의 중요관점중의 하나로서 어떠한 태스크라도 수행중 발생하게 되는 예기치 않은 조건에 대하여 검지, 격리, 인식, 정정시켜야 하며 시간적 제한 요인이나 제어를 필히 지속시켜야 되는 다른 요소들과 함께 시스템에 있어서 매우 필수적인 부분으로 볼 수 있다. 어떠한 종류의 메모리 맵을 채택하고 있는 컴퓨터에 있어서 응용 프로그램은 시스템 테이블과 분리되어야 하며 이와같은 보호용 메카니즘은 한편으로는 응용 프로그램이 오동작을 일으키거나 예외적인 조건에 적절히 응답하여야 되는 경우에 있어서 큰 난점이 있다.

멀티태스킹 환경이 사용자 프로그램에 있어서 사용자 태스크가 시스템 콜(call)을 요청할 때 그 수행 결과가 원하는 동작을 취하지 않고 오동작을 일으킬 경우 즉 태스크가 존재하지 않는 메모리를 요구하거나 잘못된 파라미터를 나타낼 경우는 요청 결과에 대한 가부 조건에 해당되는 코드를 제공하여야 한다. 이와같은 가부사항(not complete success)을 나타내는 조건을 exceptional condition이라 칭하며 exceptional condition은 프로그래머 에러와 환경적인 조건에 의한 것이 있다. 프로그래머 에러는 요청하는 태스크에 의하여 야기되는 것으로 미연에 방지될 수 있으나 환경적인 조건은 외부 여건에 따라 발생되는 exceptional condition으로 볼 수 있다. iRMX 86 OS에서는 태스크에서 정의할 수 있는 exception handler의 시스템 call을 제공하고 있다. 만약 태스크에 이러한 기능을 포함시키지 않을 때는 태스크가 속해있는 job의 exception handler를 수행시키며, 이때의 exception handler는 job이

만들어질때 제공 받게 되고 그렇지 않으면 시스템 exception handler를 다루게 된다. 따라서 iRMX 86 OS의 nucleus는 항상 동작중인 태스크에 대한 exception handler를 찾는다.

일반적으로 exception handler는 exceptional condition이 발생할 경우 제어를 받게 되며 태스크가 exceptional condition에 직면하더라도 태스크의 exception handler에 제어를 넘기지 않는 경우도 있다.

4. 결 론

오늘날 많은 시스템 설계자들은 특수 환경 및 응용 분야에 적용하기 위한 실시간 소프트웨어의 개발 필요성을 절실히 인식하고 있다. 생산라인이나 공정과정에서 발생되는 특수한 응용분야에 요구되는 실시간 소프트웨어 등은 이미 개발되어 있으므로 원하는 분야에 쉽게 적용할 수 있다. 그러나 기 개발된 소프트웨어들이 모든 조건을 만족시키지 못하므로 소프트웨어나 하드웨어의 수정 및 보완과 같은 일련의 작업 과정을 거쳐 실제로 필요로 하는 분야에 적용되게 된다. 또한 응용 분야에 따라서 실시간의 범위가 달라지므로 이에 따른 적용분야도 세분화되고 있다. 즉 어떤 분야에서의 시스템 응답 시간은 10초인 반면 1 ms의 응답시간을 요구하는 분야도 있다. 이와같이 다양하고 특수한 분야에서 사용되는 실시간 소프트웨어 중에서 시스템이 가지는 동작 환경이 특수하거나 제한된 기능만 처리하는 시스템과 같은 경우 RTE를 이용한 소프트웨어의 설계 및 개발은 효율적이며 시스템의 신뢰성은 매우 높다. 또한 소프트웨어를 ROM에 내장시킨 실리콘 소프트웨어와 같은 형태로서 RTE를 일종의 하드웨어 부품과 같이 취급함으로서 편리하게 사용할 수 있다. RTE는 통신용 소프트웨어, 공정제어용 소프트웨어 그리고 군사용 소프트웨어 등 특수 목적을 위한 시스템에 적합하며 현재 매우 활발히 적용되고 있다.