기술

# 실시간 제어시스템용 RTE의 설계관점

한 동 원
채 영 도/전자기기개발부 디지틀시스템 개발실
이 직 열/전자기술연구단 사업개발실
조 삼 현/전자기기개발부

목 차



## Ⅰ. 서 론

컴퓨터를 이용한 적용분야는 종래의 데이터 처리를 위한 시스템으로부터 다양한 환경을 위한 시스템으로 그 응용범위가 넓혀지고 있다. 특히, 컴퓨터기술의 비약적인 발전과 H/W 값의 지속적인 하락 등에 기인하여 과거에는 다룰 수 없었던 여러가지 공정제어를 효율적으로 처리할 수 있게 되었다. 이와 같은 공정제어용 컴퓨터 시스템은 제한된 시간내에 정확한 동작을 수행해야함은 물론 여러 공정의 처리, 외부 신호나 예기치 못한 사건등에 대한 시스템의 적절한 응답등 컴퓨터가 대응할 수 있게 관장하는 S/W가 필요하며 이러한 S/W는 결국 제어 시스템의 성패를 가름한다.

실제 산업응용을 위한 S/W 구현은 단일(Single Monolithic) 프로그램과 시스템 S/W를 이용한 Multitasking 프로그램으로 구분된다. 공정에서 필요로 하는 모든 기능들을 사용자가 일률적으로 열거, 기술하는 단일 프로그램에서는 시간에 관련된 부분을 분리 동작시키기란 매우 어려우나 기존 시스템 S/W에서 제공하는 실시간 기능을 이용하면 각 task를 독립적으로 수행, 관리하기가 매우 용이하여 여러 복잡한 외부 신호처리

를 위한 시스팀에서는 이와 같은 Multitasking 프로그램 방식을 채택하고 있다.

따라서 시스팀 자원의 효율적인 관리, 외부 Interrupt의 처리등 실시간 기능 등을 시스팀 S/W에 의해 구현할 때 이와 같은 시스팀 S/W를 실시간 O.S.라 칭하며, 만일 실시간 환경이 비교적 단순하여 하나의 Board에 의하여 처리될 수 있는 경우, 위에 언급한 실시간 O.S를 이용치 않고 실시간 O.S의 Kernel 부분에 해당하는 RTE(Real-Time Executive)를 이용함으로써 실시간 O.S에서 제공하는 대부분의 기능을 이용함은 물론 적은 규모의 경제적인 시스팀을 개발할 수 있다.

RET는

- 다양한 실시간 기능 제공
- 사용하기 쉬운 고급언어에 의한 접근 기능
- 이식성 및 손쉬운 구현
- Mulitasking 기능

등과 같은 특성을 제공해야한다.

이와 같은 기능을 갖는 RTE는 특정 마이크로프로세서의 제작회사에서 해당 프로세서에 맞는 RTE를 제공하거나 프로세서와 무관한 독립된 공급처에서 제공하는 소위 Third-Party S/W 가 있다. 물론 각 제품별로 고유 특성을 갖고 있지만, 사용자의 입장에서는 개발하려는 시스팀에 적합하게 수정, 보완, 구현시킬 수 있어야 한다.

그러나 기존의 Board-Level 제품에 대해서 확실히 파악하지 않거나, 기술적으로 미흡할 경우는 기존 S/W 패키지의 source를 구입하여 다시 수정, 구현시켜야 되는 어려움이 따르게 된다. 이러한 작업은 위험부담이 크며, 시간의 낭비, 개발비용의 상승등 매우 비합리적이므로 엔지니어가 새로운 H/W를 설계할 때 표준 ICs와 boards를 선택하는 것과 마찬가지로 S/W 역시 부품화된 실리콘 S/W를 이용하기도 한다.

## Ⅱ. 실시간 시스팀의 요구사항

S/W 개발에 따른 시스팀의 요구사항은 적용환경에 따라 다르다. 특히 실시간 환경에 요구되는 시스팀의 기능은 비동기적으로 발생되는 다양한 외부사건에 대한 감시 및 제어를 들 수 있다. 특정 분야에 대한 S/W 개발에서 반복수행 기능, Low-Level Interface작업, 효율적인 자원의 관리, 용이한 S/W 구현등과 같은 일련의 기능은 응용 프로그램내에서 제공할 필요가 없이 RTE에 의존함으로써 시스팀을 효율적으로 개발할 수 있다.

RTE의 주요 기능으로는 Task 우선순위에 따른 시스팀 자원의 관리, Task간의 데이터 교환, Interrupt에 의한 주변장치 제어, 실시간 Clock 제어, Interrupt Handling등을 들 수 있다. 결국 실시간 응용을 위한 S/W 설계에서는 Concurrency, 우선순위, 동기화, Task 간의 데이터 교환 등에 부합되는 기능을 갖고 있어야 된다. 초기의 Executives는 개발 단계에 있어서 특정 Host System을 필요로 했으며, 한정된 목적 시스팀에서만 수행 가능했다.

따라서, 응용 분야에 따라 수정 보완해야되는 부분들은 기능별로 독립된 표준 Interface를 이용하여 해당 Interface 부분만을 변경시킴으로써 목적 시스팀에 적용 시킬 수 있다. 표준 Interface는 O. S. 의 외부동작을 나타내는 다음의 네가지 Interfaces로 분류할 수 있다.

- Application Interface

O. S. 에서 제공하는 System Calls Set에 의하여 정의 되며, Task-제어용 서비스 그룹과 입출력 서비스 그룹으로 나눌 수 있다.

- User Interface

사용자 터미널에서 입력되는 명령어들과 명령어 해석 프로그램에 의해 정의된다.

- Media Interface

데이터 프로그램을 교환하기 위해 시스팀간에 이송되는 매체의 Format에 따라 정의 되지만 실시간 O. S. 에서는 해당되지 않는다.

- Device Interface

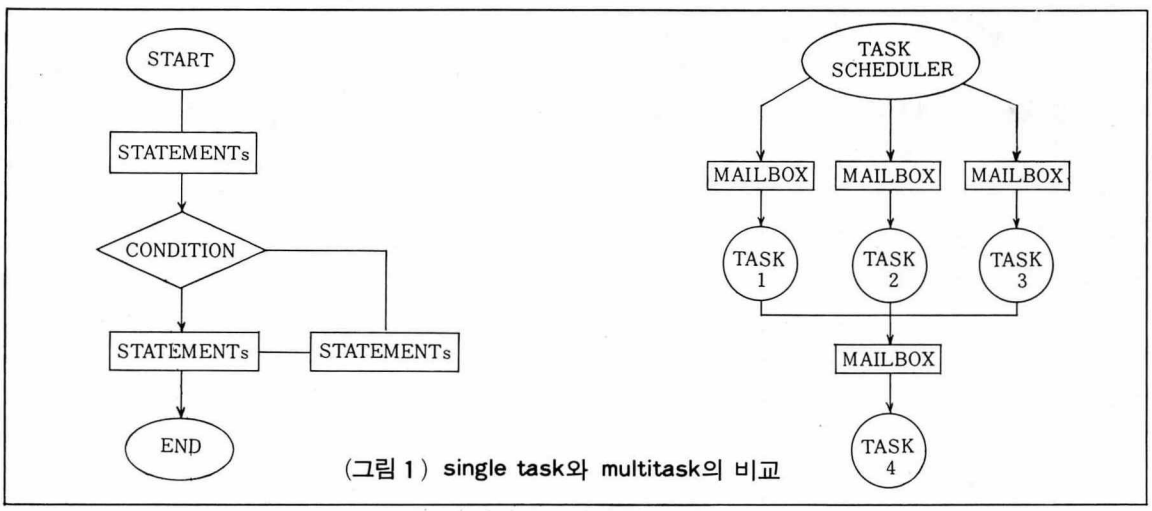
O. S. 에서 I/O 동작을 수행하기 위해 입출력 장치나 각 장치에 특정된 루틴들을 액세스하는 명령어 집합에 따라 정의되며, Custom Devices 및 비표준 소자의 Interface에 따라 달라질 수 있다.

## 1. 응용 프로그램의 요구사항

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

- Tasking Method : Task는 제어동작을 두 가지 이상의 한정된 제어부분으로 나누는 방법을 일컫는다. 이러한 각 부분은 Task라 불리우며, 각 Task는 특정 제어기능을 구현하기 위해 설계된 제어 프로그램으로 구성된다. 실시간 시스팀에서 Task는 기본적이며 매우 중요한 개념이다. Task는 "논리적으로 완벽한 프로그램 세그먼트" 혹은 "시스팀 공유자원(입출력 장치, CPU, 메모리, ··)을 할당받을 수 있는 최소한의 프로그램", "동작 상태에 있는 프로그램의 집합" 등과 같이 정의된다.

- Task Environments : Task는 Single-Task와 Multi-Task로 나누어지며 START에서 END로 끝나는 Single-Task에서는 START-END 사이에 몇몇 루틴들이 존재하게 되어 동작될 때는 어느 특정 경로를 통해 동작된다. 이와는 대조적으로 (그림 1)과 같이 Multi-Task Environment에서는 독립된 기능을 갖는 Task 들이 동시에 존재할 수 있으며, 각 Task는 서로 독립적이며 논리적으로 완성된 기능을 갖는다. 또한 여러 Task들이 필요로 하는 자원들에 의하여 Job이 구성되며 이러한 Task가 갖는 목표는 첫번째로 작업의 특정부분을 수행하는 것과 프로세서를 독점하여 첫번째의 목표를 진행시키는데 있다.




(그림 1) single task와 multitask의 비교

## 2. 실리콘 S/W 설계시 고려사항

일단 S/W를 실리콘에 내장시키면 그 내용을 변경할 수 없게 되며 이를 수정, 보완하여 원하는 시스템에 이용하기 위해서는 새로운 H/W나 응용분야에 맞게 변경시킬 필요가 있다. 따라서 실리콘 S/W 설계시 고려할 점은 다음과 같다. 실리콘 S/W는 두 종류의 코드로 나눌 수 있는데 On-Chip 코드와 Off-Chip 코드가 그것이다. 특별한 경우에 있어서는 Off-Chip 코드의 일부가 On-Chip 코드와 매우 밀접한 관계를 가질 수 있으며, 실리콘 S/W 패키지의 한 부분으로 개발되기도 한다.

이와 같은 기능을 갖는 Off-Chip 코드에는 Initialization, Interface, System, Version Update 코드 등이 있으며 실리콘 S/W를 한 시스템 이상에서 사용하거나 변경에 무관한 기능을 부여하기 위해서는 Position Independence, Configuration Independence, Stepping Independence와 같은 특성을 지녀야 한다.

- Position Independence :

최근의 마이크로프로세서는 최소한 1메가 바이트의 메모리를 가리킬 수 있기 때문에 실리콘에 내장된 시스템 S/W는 메모리내의 위치에 무관하게 동작되어야 한다. 따라서 Read-Only, On-Chip 코드/데이터의 절대번지(Absolute Address)는 시스템 구성에 제한을 준다. On-Chip 코드는 Offset 번지만 인식하므로 절대번지는 처리될 수 없게 된다. 그러므로 On-Chip 코드가 Position Independent하게 되면, On-Chip 코드에서 지정하는 절대번지는 프로세서의 레지스터를 통해 얻을 수 있다.

컴파일러나 Relocatable 어셈블러에서는 보통 Position - Dependent 코드를 만들게 되지만 Linking, Locating 단계를 거치게 되면 쉽게 Position-Independent 코드를 만들 수 있다.

- Configuration Independence :

두번째 요구사항으로서 On-Chip 코드가 시스템의 기존 H/W와 S/W 구성에 무관해야 한다. 결국 On-Chip 코드는 간접적으로 다른 코드나 데이터를 액세스하는 대신 시스템 구성을 유추하기 위해서는 Run-Time 데이터를 검지해야 된다. 실리콘 S/W의 Read-Only 성질로 말미암아 On-Chip 코드내에 상수가 존재할 경우는 문제가 될 수 있다. 즉, 시스템의 어느 곳이나 위치할 수 있는 H/W 장치의 값, 혹은 유사한 기능을 갖고 있지만 서로 다른 Programming Interfaces를 행하는 장치의 값들은 On-Chip 코드에 내재해서는 안된다. 결국 시스템 구성에 영향을 줄 수 있는 모든 값들은 간접적으로 액세스되야 한다.

- Stepping Independence :

이것은 Configuration Independence의 확장으로서, S/W가 실리콘에 내장 되기위한 중요한 요구사항이다. "Step"이란 On-Chip 코드의 Updated Version을 나타내며 On-Chip 코드와 Off-Chip 코드는 각 코드들이 서로 변경되더라도 호완적이어야 한다. 즉, 모든 On-Chip 코드 Version이 Off-Chip 코드 Version과 서로 일치되게 동작할 때 Stepping Independece가 있게된다.

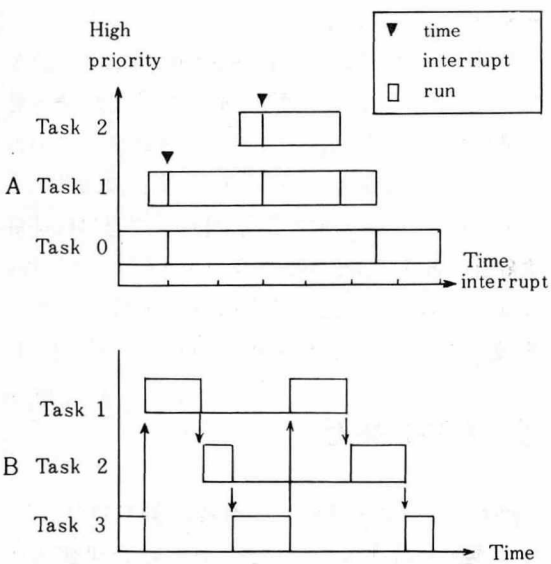
## Ⅲ. RTE 특성

마이크로프로세서용으로 설계된 대부분의 RTE는 프로세서가 갖는 속도와 기능을 최대한 활용하도록 Realtime Multitasking System에 적용된다.

Executives는 두 가지의 파라미터로서 규정 지을 수 있는데 Tasks간의 수행전환을 위한 Trigger 메카니즘과 Tasks 수행선택을 위한 Scheduling이 그것이다. Trigger 메카니즘에는 Interrupt Driven과 Task Driven 방식이 있으며, Interrupt Driven 방식에서는 한 Task에서 다른 Task로의 수행 변환(Context Switching)이 S/W나 H/W에서 야기되는 Interrupt에 따라 결정된다.

Task Driven Executive에서의 수행 변환은 한 Task의 수행이 끝났을때만 다음 Task로의 전환이 일어난다. Tasks 수행을 위한 Scheduling 방법에도 Round-Robin과 우선순위 방식이 이용되며 Round-Robin 방식에서는 한정된 시간 간격동안 Task가 수행되며 각 Task들은 순환방식으로 Scheduling 된다. 이와는 반대로 우선순위에 의한 Scheduling 방식에서는 각 Task들이 우선순위를 부여받게 되며, 한 Task에서 다음 Task로 변환하기 전에 Scheduler가 Task의 우선순위를 확인한다. 따라서, CPU를 사용하는 가장 높은 우선순위의 Task는 다음 Task의 우선순위가 자기 자신보다 낮을 경우 CPU를 계속해서 사용하게 된다.

(그림 2)에서 보면, Interrupt 구동에 의한 시스팀(A)은 Time Interrupt가 발생할 경우, 우선순위가 높은 Task가 먼저 수행되며, Task 구동에 의한 시스팀(B)의 각 Task는 순차적으로 수행되며, 진행중인 Task가 끝났을 경우, 다음 차례의 Task가 Scheduler에 의해 수행된다.

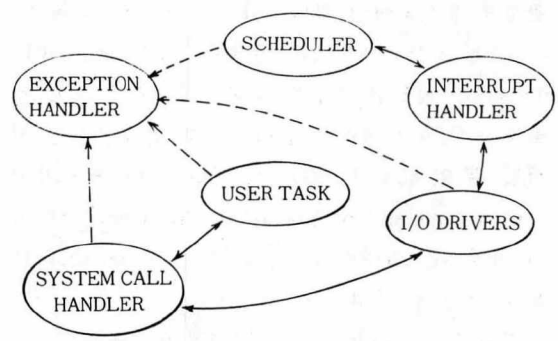



(그림 2) 우선순위 방식 (A)과 순환 방식 (B)의 scheduling

## 1. RTE 구조

Executive의 개별적인 구성을 언급하기에 앞서 기본적으로 갖추어야 할 주변 장치들에 대한 입출력 동작 처리를 기술한다.

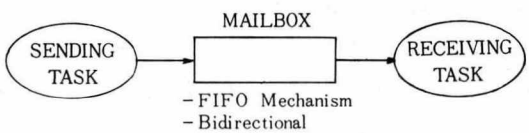
입출력 요청은 사용자 Task에서 발생되며 시스팀에서는 이러한 요청에 따라 입출력의 시작, 데이터 전송 등과 같은 기능을 처리한다. 만약, 특정 입출력 장치를 동시에 여러 Tasks에서 요청 할 경우 이 장치는 아무런 서비스를 제공할 수 없으므로 여러 Tasks에 순서를 할당하여 효율적으로 입출력 요청을 처리 시키기 위한 Scheduler가 필요하다. (그림 3 참조)




(그림 3) RTE의 전체 구조

단일 Task에 의한 수행에서는 하나의 Task 가 완전히 끝난 후에 다음 Task가 수행되므로 데이터 전송에 소요되는 시간적 낭비가 많으나 Multitasking, Interrupt-Driven 환경에서는 입출력 동작을 위해 기다리는 동안 다른 Task를 수행 시켜 줄 수 있으므로 효율을 높일 수 있다. 물론, 각 주변 장치들의 상태를 감지해야하는 부담이 수반 되지만, 크게 영향을 주는 요인이 되지 못한다.

Executive의 중요한 다른 기능으로서 Task 간의 데이터 교환 기능을 들 수 있다. Tasks는 서로 독립된 프로그램으로 존재하지만, (그림 4)와 같이 데이터를 공유하거나 교환할 수 있는 장치가 필요하게 되며 Tasks간의 데이터 교환은 "MESSAGEs"로서 이루어진다. Tasks간의 데이터 교환은 두 가지의 데이터 구조인 Dynamic Buffer와 Global Common Memory Variables로 구성된다.




(그림 4) 메시지 교환 (채널)

Dynamic Buffer 방식에서는 한 Task에서 다음 Task로 메시지를 보낼 때 RTE 루틴을 호출하며 RTE에서는 버퍼를 선택하여 수신할 Task에 버퍼의 위치만 알려준다. 이는 Task 간의 정보전달에 있어 효율적인 방식이된다.

Global 공통 메모리 방식에서는 Tasks 들로 부터 접근가능한 공통 메모리내에 버퍼를 만들게되며, 이러한 버퍼는 명령어, 상태, 데이터 영역으로 구분지을 수 있다. 각 버퍼는 지정된 Source Task와 Destination Task간 메시지 전송을 위한 것이므로 Tasks에 한정된 버퍼만 할당되어 Global 공통 메모리 방식은 Dynamic 버퍼 방식 보다 더 많은 메모리 영역이 필요하지만, Task에 대한 Queueing 메시지나 버퍼 할당을 위한 Executive의 특정 루틴이 없어도 된다. 따라서 Executive의 설계시 Task간의 통신을 위한 Global공통 메모리 방식이 쉽게 적용될 수 있다.

이상에서 언급한 입출력 처리와 Task간의 통신 이외에 Executive를 구성하고 있는 기본 요소로는 다음과 같다.

가. Real-Time Clock Handler

RTE에서는 우선 순위가 가장 높은 Task의 서비스를 위해 반복적으로 시스팀의 상태를 점검하며 이러한 동작은 단순하면서도 짧은 시간 동안 처리된다. 따라서 한 Task가 CPU를 점유하고 있는 동안은 Executive가 CPU를 제어할 수 없으므로 실시간 Clock이나 프로세서에서 지원하는 Interrupt 시스팀에 있어서 이와 같은 기능은 필수적이다.

실시간 Clock은 세분된 Time Interrupts를 계속 발생시켜야되며, 매 Interrupt후 S/W에서 리셋시켜야 되는 프로그래머블 타이머의 Interval 타이머와는 상이하다. 수행중인 Task에 있어서 진행 과정속에 임의의 Interrupt가 발생할 수 있다. 그렇지만 Task의 수행 결과는 Interrupt 발생과 무관하며 Executive에서는 Interrupt발생시 Task전환에 따른 모든 정보나 CPU의 상태 등을 저장시킨다.

실시간 Interrupt를 위한 시간간격은 H/W에 따라 다르지만 간격이 너무 짧을 경우는 Executive가 대부분의 CPU시간을 차지하게 되어 시스팀의 효율이 떨어지며, 길 경우는 우선 순위가 높은 Tasks가 CPU의 제어를 받기가 힘들게 된다. 일반적으로 10ms이하의 시간간격이 채택되고 있다.

나. Scheduler

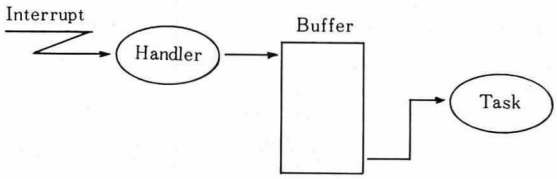
Scheduler는 RTE와 사용자의 응용 프로그램 사이의 긴밀한 동작을 처리시켜 주며, 이의 기본 기능으로는 다음과 같다. 먼저, 사용자나 시스템 Task에서 어떤 Task가 가장 높은 우선순위를 갖고 있는지를 결정하고 Task의 상태를 전환시킨다. 즉,

- CPU의 제어를 요청하는 우선 순위가 가장 높은 Task의 결정
- Tasks 상태 전환
- Task 데이터 영역의 저장 및 복구
- Task 레지스터와 Stack Pointer의 복구

Task Scheduler는 자주 수행되므로 'ZEROth' Task로 볼 수 있으며 Scheduler의 제한된 부분들은 Nonreentrant 영역으로 Interrupts을 Disable 시키면서 수행되기도 한다. Task Scheduler는 보통 Executive에서만 사용 가능한 하나의 Entry 포인터를 갖고있다. Task Scheduler가 Interrupt를 받게되면 이 Entry 포인터에서 시작되며, 레지스터의 내용을 저장하거나 Interrupt 포인터로 부터 Task Scheduler를 재 수행시킬 필요는 없다.

Task Scheduler의 설계는 일반적으로 두 가지 종류로 구분 짓는데 Preemptive와 Non-Preemptive로 나눈다.

Preemptive Scheduler 하에서 동작되는 Task는 Interrupt Handler에 의해 가장 높은 순위의 Task로 전환되기 위해 Suspend된다. 즉, Interrupt Handler에서 받은 데이터는 Handler에 전환된 Task가 즉시 처리해야 되며 이와 같은 방식의 Scheduler에 있어서 (그림 5)와 같이 Interrupt 처리를 위한 데이터 버퍼링이 Single Type인 경우, 데이터가 처리되기 전에 또 다른 데이터에 의해 Overwritten될 수 있으므로 Double- 혹은 Multiple- 버퍼링으로 수시로 발생할 수 있는 Interrupt에 대한 Task 처리를 효율적으로 다룰 수 있다.




- 버퍼가 채워지면 handler에서 task를 구동시킨다.
- handler와 task는 버퍼를 서로 공유함으로써 정보교환을 한다.

(그림 5) 버퍼링 개념

Non-Preemptive Scheduler 하에서 동작되는 Task는 우선 순위가 높은 Task가 생기더라도 Suspend 되지 않지만, Interrupt 서비스를 위해 동작하는 Task가 스스로 자신을 Suspend 할 때 까지는 계속 CPU를 점유할 수 있다. 따라서, Non-Preemptive Scheduler를 적용한 시스템에서는 Task가 임의로 중지되는 과정에서, 주어진 프로세서의 Bandwidth내에 Task 처리량을 제한함으로써 Task 들 사이의 상호 동작을 원만히 꾀할 수 있다.

결국, 시간적인 제한 요소가 없는 실시간 응용 분야에 있어서는 이와 같은 방식의 Scheduling이 보편적으로 이용되고 있으며, Scheduler의 설계를 간단히 할 수 있고 메모리, 수행시간에 따른 제한 요인도 줄일 수 있다. 특히, Non-Preemptive Scheduling의 큰 장점으로는 Task 전환에 따른 프로그램간에 전달될 수 있는 일치되지 않은 데이터의 발생을 줄일 수 있다.

Preemptive Scheduler에서 Interrupt Handler가 우선 순위가 높은 Task를 구동시키려 할 때, 메시지에 있는 데이터의 일부를 처리하게 되며 원래의 Task가 처리하던 메시지의 내용이 바뀔 수

도 있다. 결국 Scheduler 설계시 또 다른 서비스 메카니즘이 수반되어야 한다. 따라서, Nonpreemptive 방식의 Scheduler가 쉽게 설계될 수 있으며 보편적으로 사용된다.

Task 전환을 위해 Scheduler 에서는 몇 가지의 테이블을 사용하고 있으며, 이러한 테이블은 Task Control Block (TCB) 〈혹은 Task Data Block (TDB), Task Status Table (TST)〉라 불리우며, 특히 Linked-List 데이터 구조를 갖는 TCB에서는 Scheduler가 TCB를 액세스 하기 위한 Forward-Link 포인터와 Task가 시작되는 번지값, Stack Pointer, State Flag 등을 포함하여 Linked-List 데이터 구조를 이용하여 Task의 우선 순위를 결정하거나 Scheduling 동작을 손쉽게 처리한다.

Scheduler가 Task를 인식하는 것으로 다음의 세가지 방법이 있다.

첫째 : 시스템 호출로서 Inactive 상태에 있는 Task가 대기상태로 전환된다.

둘째 : 데이터의 입출력 처리를 위한 사용자 Task 요청 인식

셋째 : 시스템의 공유자원 요청에 따른 Interrupt 발생등으로 나눌 수 있다.

### 다. System Call Set

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

- 입출력 명령어
- Task Creation / Deletion 명령어
- Task 간의 데이터 교환용 명령어
- Clock 명령어
- Task Identification 명령어
- Task / 오퍼레이터 통신용 명령어

1) 입출력 명령어

: 사용자 Task에서 특정 장치로의 데이터 전송을 위해 요청하는 시스템 명령어

- 입출력 장치의 정보
- 전송 데이터의 수
- 전송 채널 수
- 데이터의 코딩
- 메모리에서 데이터의 Source / Destination
- 정보에 관계되는 입출력 동작

2) Task Creation / Deletion 명령어

: Task를 Scheduler에 인식시키며, Task의 상태를 전환시킨다.

- Task Identification Number
- Task의 우선 순위등과 같은 정보를 통해 Scheduler가 시스템의 공유자원을 Task에 할당한다.

3) Task의 데이터 교환용 명령어

; Task간의 메시지 전송을 위한 것으로 Task들 사이에 동기가 이루어질때 S/W에서 정의된 채널을 통해 이루어진다.

4) Clock 명령어

; 실시간 Clock을 위한 시간간격을 결정하며 타이머 interrupt을 발생한다.

5) Task identification 명령어

; Task creation/deletion 명령어와는 달리 Task의 ID만 바꿀 수 있다.

6) Task/오퍼레이터 Communication 명령어

; Task와 오퍼레이터간의 메세지 전송을 위한

것으로 실시간 공정-오퍼레이터간 통신에 유용하다.

### 라. Exception Handler

RTE 설계시 필히 다루어야 되는 것으로 동작중 에러 발생에 응답해야한다. 즉, RTE는 절대로 멈추거나 오동작을 일으켜서는 안되므로 존재치 않는 메모리의 위치를 지정하거나 시스템 응용에서 제한된 메모리의 위치를 사용할 경우, 이러한 에러를 검지해야 하며, 우선 순위나 Job지정이 타당한가에 대해서 필히 검지해야된다.

지연(latency) 문제에 있어서도 CPU 시간을 요구하는 Job이나 프로그램이 너무 많을 경우, 공정이나 사용자에 의해서 지정되는 제한시간 이내에 Critical Jobs들을 처리할 수 있어야하므로 지연 문제 역시 고려되어야 한다. 결국, 에러 처리는 실시간 컴퓨터 시스템의 중요 관점중의 하나로서 어떠한 Task라도 수행중 발생하게 되는 예기치 않은 조건에 대해 검지, 격리, 인식, 정정 시켜야하며, 시간적 제한요인이나 제어를 필히 지속시켜야 되는 다른 요소들과 함께 시스템에 있어서 매우 필수적인 부분으로 볼 수 있다.

어떠한 종류의 메모리 Map을 채택하고 있는 컴퓨터에 있어서 응용 프로그램은 시스템 테이블과 분리되어 있어야하며, 이와 같은 보호용 메카니즘은 한편으로는 응용 프로그램이 오동작을 일으키거나 예외적인 조건에 적절히 응답해야되는 경우에 있어서 큰난점이 있다.

Multitasking environments의 사용자 프로그램에 있어서 사용자 Task가 시스템 Call을 요청할 때, 그 수행 결과가 원하는 동작을 취하지 않고 오동작을 일으킬 경우, 즉, Task가 존재치 않는 메모리를 요구하거나 잘못된 파라미터를 나타낼 경우는 요청 결과에 대한 가부 조건에 해당되는 코드를 제공해야한다. 이와 같은 가부사항(not complete success)을 나타내는 조건을 예외적 조건이라 칭하며, 이것은 프로그래머 에러와 환경적인 조건에 의한 것이 있다.

프로그래머 에러는 요청하는 Task에 의해 야기되는 것으로 미연에 방지될 수 있으나 환경적인 조건은 외부 여건에 따라 발생하는 Exceptional Condition으로 볼 수 있다. iRMX 86 O. S.에서는 Task에서 정의할 수 있는 exception handler의 시스템 Call을 제공하고 있다. 만약 Task에 이러한 기능을 포함시키지 않을 때는 Task가 속해있는 Job의 exception handler를 수행시키며, 이 때의 exception handler는 Job이 만들어 질때 제공 받게 되고, 그렇지 않으면 시스템 exception handler를 다루게 된다. 따라서 iRMS 86 O. S. 의 Nucleus는 항상 동작중인 Task에 대한 exception handler를 찾는다.

일반적으로 exception handler는 exceptional condition이 발생할 경우 제어를 받게되며, Task가 exceptional condition에 직면하더라도 Task의 exception handler에 제어를 넘기지 않는 경우도 있다.

## Ⅳ. 결 론

오늘날 많은 시스템 설계자들은 특수환경 및 응용분야에 적용하기위한 실시간 S/W의 개발 필요성을 절실히 느낀다. 생산라인이나 공정과정에서 발생되는 독특한 응용분야에 요구되는 실시간 S/W등은 이미 개발되어 있으므로 원하는 분야에 쉽게 적용할 수 있다. 그러나 기 개발된 S/W 들이 모든 조건을 만족시키지 못하므로, S/W나 H/W의 수정 및 보완과 같은 일련의 작업과정을 거쳐 실제로 필요로하는 분야에 적용되게된다. 또한 응

용분야에 따라서 실시간의 범위가 달라지므로 이에따른 적용분야도 세분화 되고있다. 즉 어떤 분야에서의 시스템 응답시간은 10초인 반면, 1 ms의 응답시간을 요구하는 분야도 있다.

이와 같이 다양하고 특수한 분야에서 쓰이는 실시간 S/W 중에서 시스템이 갖는 동작환경이 특수하거나 제한된 기능만 처리하는 시스템과 같은 경우 RTE를 이용한 S/W의 설계 및 개발은 효율적이며 시스템의 신뢰성은 매우높다. 또한 S/W를 ROM에 내장시킨 실리콘 S/W와 같은 형태로서 RTE를 일종의 H/W 부품과 같이 취급함으로써 편리하게 사용할 수 있다. RTE는 공정제어용 S/W나 통신용 S/W, 그리고 군사용 S/W등 특수목적을 위한 시스템에 적합하며, 현재 매우 활발히 연구되고있다.

## 참 고 문 헌


1. Mellichamp, Real-Time Computing, Van Nostrand Reinhold, New York, 1983.
2. Daniel A. Crowl, "A Real-Time Fortran Executive," IEEE MICRO, Aug. 1985, pp. 48-66.
3. W. S. Heath, "A System Executive for Real-Time Microcomputer Programs," IEEE MICRO, June. 1984, pp. 20-32.
4. K. Burgett, E. F. O Neil, "An Integral Real-Time Executive for Microcomputers," COMPUTER DESIGN, Jule. 1977, pp, 72-82.
5. Carlos J. Tavora, "A Basic Technique for Real-Time System Design," COMPUTER DESIGN, Oct. 1980, pp. 147-152.
6. Ron Slamp, Jim Person, "Software That resides in Silicon," VLSI DESIGN, Mar/Apr. 1983.
7. iRMX86 Nucleus Reference Manual, Intel Corp, Santa clara, Calif., 1981.
8. James Issak, "Designing system for Real-Time Applications," BYTE, April. 1984, pp. 127-132.