# Real-Time Kernel을 이용한 보호계전기용 보조 소프트웨어의 구현방법의 개선에 관한 연구

윤영길 박인권 윤남선 안복신
(주)하우 LG산전 전력 연구소

# Implementing Auxiliary Software for Protective Relay Using Real-Time Kerı

Young-Kil Yoon In-Kwon Park Nam-Seon Yoon Bok-Shin Ahn
HOW Co.Ltd LG Industrial Systems

**Abstract** - The need to accommodate much complex algorithm, high communication functionality and easier user interface lays heavy burden on the software developer of the protection relay these days. Using lightweight real-time kernel like uC/OS, the software development process can have much structural and easier feature. And total cost needed to development and maintenance of the software also can be reduced by development based on these real-time kernels.

## 1. 서 론

최근 이루어진 디지털 기술의 발전으로 말미암아 광범위한 디지털 기술의 응용 제품이 시장에 쏟아지게 되었으며 전력계통도 이에 예외는 아니어서 기존의 EM타입 계전기 및 정지형 계전기를 급속히 디지털형 계전기가 대체하고 있는 추세이다. 또한 초기에 구현되었던 단순한 알고리즘에서 발전하여 근래에는 새로이 개발된 고도의 복잡성을 지니는 계전 알고리즘이 속속 도입되고 있으며 이러한 알고리즘은 필연적으로 보호계전기 내부의 하드웨어 구현 및 소프트웨어 구성의 복잡화를 수반한다. 또한 이러한 보호 계전기의 고기능화에 수반하여 기존의 분리되었던 계측기능 및 보호기능은 디지털 보호계전기 하드웨어로 통합되는 추세에 있으며 또한 이러한 디지털 계전기가 설치된 전력 계통 시스템 전반에 걸친 시스템 통합 요구는 필연적으로 고 수준의 통신작업 및 인간/기계 인터페이스등 많은 부가적 기능을 요구하게 된다. 그 동안 진행된 32Bit 마이크로 프로세서의 비약적인 발전과 반도체 가격의 대폭적인 하락은 그 동안 보호계전기 개발과정에 있어서 존재하였던 여러 가격 및 성능상의 제약조건을 대부분 제거하였다. 이러한 주변여건으로 인해 보호계전기를 개발하는데 있어서 하드웨어의 비중에 비하여 소프트웨어의 비중이 점증하고 있으며 이는 기존의 순차적인 처리로 일관하였던 계전기용 소프트웨어 개발방법으로는 처리할 수 없는 고도의 복잡성의 적절한 처리 및 작업간의 수행 조절기능을 요구하고 있다. 고도로 숙련된 개발자의 경우 이러한 상황을 능숙하게 해결해 나갈 수 있으나 극히 경험적인 측면이 강하여 그러한 작업을 정규화 하기 어려운 측면이 있다.

## 2. 본 론

### 2.1 Real Time System(실시간 시스템)

전력계통의 감시 및 제어를 위한 많은 디지털 장비들은 실시간 시스템(Realtime System)의 특성을 지닌다. 실시간 시스템이란 주어진 입력에 따른 시스템의 결과값의 정확성과 더불어 그 결과 값이 산출되기까지의 시간도 미리 정해진 한도 내에 존재하여야 하는 시스템으로 정의 할 수 있을 것이다. 본 논문에서 대상으로 선정한 보호 계전기의 경우 입력된 전기량에 의한 동작/부동작의 판단이 정확해야 함에 더불어 그 결과를 산출하기까지의 시간도 미리 정해진 제한 시간을 만족해야 하는 시스템으로 대표적인 실시간 시스템이다. 실시간 시스템은 그 응답이 산출되는 시간을 기준으로 하여 다음과 같이 분류될 수 있다.

경성 실시간 (Hard Real Time) 시스템 : 어떤 작업의 결과 출력이 미리 정해진 한도를 지키지 못하는 경우 즉시 동작 실패로 판정되는 시스템을 일컫는다. 즉 항공기 항법 시스템이나 군용 미사일 제어 프로그램, 기차 철로 제어 시스템, 핵발전소 제어 시스템 등은 정해진 시간 범위 내에 결과를 내지 못한다면 경우에 따라서 엄청난 결과를 낳을 수 있으므로 경성 실시간 시스템의 범주에 속한다. 보호 계전기도 예정된 결과가 미리 정해진 시간한도 내에 산출되지 못할 경우 동작실패로 판정되므로 경성 실시간 시스템에 속한다고 할 수 있을 것이다.

연성 실시간(Soft Real Time) 시스템 : 어떤 작업의 결과 출력이 데드라인을 지키지 못하더라도 심각한 시스템 에러는 발생하지 않는 시스템을 의미한다. 예를 들어 PDA(Personal Digital Assistant)와 그 응용 프로그램들은 정해진 범위를 넘은 시간 지연이 발생하더라도 그것이 시스템 에러의 발생을 의미하지는 않는다.

### 2.2 Real Time Kernel(실시간 커널)

이러한 실시간 시스템을 구현하기 위하여 기존에는 구현되는 CPU의 인터럽트를 이용하여 각 수행 기능에 우선 순위를 부여하는 방법으로 소프트웨어를 구현하는 것이 일반적이었다. 그러나 구현될 장치에 요구되는 기능이 점차 다양해지고 복잡해짐에 따라 여러 작업 사이의 (필요하다면 동적인)우선 순위의 조정이 필요하게 되었을 뿐 아니라 그 동안 고급 응용 소프트웨어에서 요구되었던 IPC(Interprocess Communication), 상호배제(Mutual Exclusion) 등의 기능을 구현해야할 필요성이 대두되게 되었다. 또한 커널 자체의 수행시간이 예측가능하여 전체적인 시간제한을 설계자가 직접 계산할 수 있게 할 수 있는 특성이 요구되었다. 기존의 범용 운영체제(유닉스, 윈도우즈 NT)등은 이러한 예측가능성을 결여하고 있으며 또한 실제 시스템 위에 적재되어야 하는 운영체제 자체의 거대한 사이즈는 극히 제한 적인 자원이 허용된 상황하에서 동작하는 것이 일반적인 실시간 시스템 응용에 적합치 못한 측면을 가지고 있다. 그리하여 이러한 운영체제의 기능중 수행될 작업간의 우선 순위를 관리하는 기능만을 집약한 실시간 커널(Realtime Kernel)이 여러 실시간 시스템에서 필요에 맞게 개발되어 사용되고 있다. 이러한 실시간 커널을 이용함으로서 개발자는 인터럽트등 하드웨어의 의존하는 우선 순위의 배정을 이용하지 않고 자신이 직접 작업을 생성하여 그 작업의 우선 순위를 훨씬 풍부한 선택범위를 가지고 배정할 수 있으며 수행되어야 할 작업중 외부의 이벤트에 신속히 반응하여야 하는 작업에 높은 우선 순위를 배정

함으로서 전체 응용 소프트웨어의 융통성을 높일 수 있을 뿐 아니라 이러한 우선 순위의 선정작업을 수행중 내재된 알고리즘에 의해 동적으로 수행할 수 있으므로 보다 더 유연한 응용 소프트웨어의 개발이 가능하게 되었다. 또한 수행되어야 할 작업을 더욱 작은 작업단위로 분해하는 것이 가능해 짐으로서 요구조건에 알맞은 응용 소프트웨어의 개발이 더욱 용이해졌을 뿐아니라 디바이스 드라이버 레벨의 기반 소프트웨어의 재사용의 가능성이 커지게 되었다. 또한 기존 실시간 커널을 이용하지 않은 응용 소프트웨어의 경우에는 일종의 완결성을 지니게 되어 이후 요구사항의 추가에 따른 확장이 매우 곤란하였으나 실시간 커널을 이용함으로서 이러한 기능의 확장이 단순한 작업(태스크)의 추가로 귀착됨으로서 응용 소프트웨어의 유지보수가 더욱 용이하여졌다. 또한 추가적인 작업의 삽입은 응용소프트웨어 개발단계에서의 디버깅용 작업의 삽입 및 전체 수행을 관찰하는 작업등의 진행을 가능하게 하여 응용 소프트웨어 개발 시에 필연적으로 발생하는 버그를 수정하는 것도 보다 용이하게 하여 줄 수 있다. 이러한 실시간 커널의 이용은 보다 구조적인 응용 소프트웨어의 개발을 가능하게 함으로서 향후 유지보수를 위한 문서화 작업도 보다 용이하게 이루어질 수 있다. 또한 선정된 실시간 커널에서 제공하는 커널 서비스 , 예로 들면 IPC(Interprocess Communication), 동기화 객체(Synchronization Object) 등, 를 이용하여 보다 용이하게 요구조건에 맞는 응용소프트웨어를 개발할 수 있게 되었다.

### 2.3 uC/OS

uC/OS는 Jeans J. Labrosse에 의해 개발된 간단한 구조의 실시간 커널이다. 1992년 5월에 Embedded Systems Programming 지의 기사로 처음 소개되었으며 초기에는 모토롤러사의 CPU인 68HC11상에서 구현되었다. uC/OS 약 600라인의C코드 및 프로세서 의존적인 어셈블러 코드로 이루어져 있으며 버전 1.12의 경우 태스크 스케쥴링, 기본적인 커널 서비스 기능 및 기본적인 IPC 서비스가 구현되어 있으며 메모리 관리 등의 복잡한 운영체제 서비스는 구현되어 있지 않다. 최근 발표된 버전 2.0의 경우는 각 태스크에 대한 엔트리/액시트 훅 및 고정된 크기를 지니는 메모리 관리 기능이 첨가되었다. 이러한 구성은 대부분의 상용 실시간 운영체제의 경우 커널 및 메모리 관리 모듈로 이루어져 있는데 비해 매우 간단한 구성이라 할 수 있다. 따라서 이러한 단순성으로 인하여 80186/188의 경우 ROM화 될 수 있는 실행 모듈 크기 최소 3150바이트 정도가 요구되며 실행 시에 필요한 최소 RAM 요구사항도 6400바이트 정도가 필요한 아주 작은 크기의 실시간 커널이다. 이에 비해 다른 상용 실시간 커널의 경우 최소 200KB정도의 메모리가 필요하므로 보호 계전기 등의 실시간 응용 시스템에서의 이용에 걸림돌이 된다. 또한 내장된 각 모듈/함수별 수행 시간의 예측이 가능하다. 선점형(Preemptive) 멀티 태스킹을 지원하는 커널이므로 비선점형(Non-Preemptive) 멀티태스킹 커널에서 볼 수 있는 하위 우선 순위 태스크의 수행지연으로 인한 상위 우선 순위 태스크의 수행 방해현상은 없다. 또한 소스코드까지 공개된 커널이므로(uC/OS The Real-Time Kernel ,ISBN 0-13-031352-1 Prentice Hall)그 동안 모토롤라, 인텔, Z80계열 등 여러 마이크로 프로세서 환경에 이식되어 실제로 입수하여 이용할 수 있는 코드가 풍부할 뿐아니라 커널의 제작자가 의도적으로 C언어로 대부분의 코드를 작성하였고 시스템 의존적인 부분만을 어셈블러언어로 작성하여 기계 의존적인 어셈블러 코드부분을 최소화하였으므로 이외의 다른 마이크로 프로세서로의 이식도 용이한 편이다.

uC/OS내에서 하나의 작업은 태스크(Task)라 불리며 이는 다른 운영체제에서 CPU의 점유단위로 이용되는 스레드(Thread)와 동일한 의미를 갖는다. 태스크는 수행되는 도중, 즉 Running 상태에서는 자신이 CPU를 점유하고 있다고 생각하며 또한 uC/OS의 구현상 고유의 우선 순위 및 자신만의 컨텍스트, 스택영역을 가진다. uC/OS에서 각각의 태스크는 다음 그림과 같은 상태를 가진다.

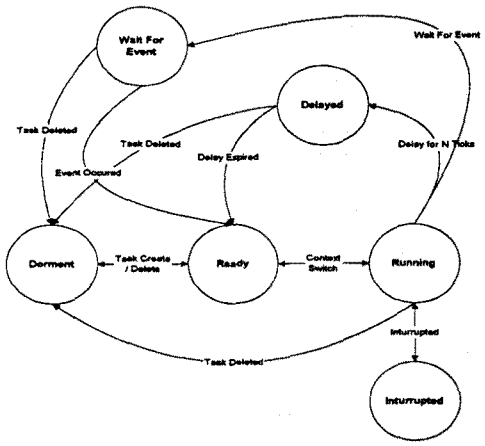


그림 1. uC/OS내 태스크의 상태도

uC/OS에서 구현되는 태스크의 우선 순위는 미리 정해진 63단계의 우선순위중 하나를 태스크에 할당하게 되며 서로 같은 우선 순위를 두 개 또는 그 이상의 태스크가 공유할 수 없으므로 Round Robin형식의 태스크 스케쥴링은 일어나지 않는다. uC/OS에서 제공하는 IPC 메커니즘은 세마포어, 메일박스 및 메시지 큐가 제공된다. 멀티 태스킹 시스템의 경우는 수행중 크리티컬 섹션(여러 태스크에 의해 공유되는 전역 코드영역)에 대한 접근 통제가 필요하다. uC/OS에서는 이러한 크리티컬 섹션에 대한 접근 통제문제를 세마포어로서 해결하고 있다. 세마포어의 초기화 시 카운트 수를 설정함으로서 바이너리/카운팅 세마포어가 모두 구현 가능하다. 메시지 큐 및 메일 박스는 그 구현이 매우 비슷하여 서로 혼용될 수 있다. 메시지/메일 박스를 기다리는, 즉 Pending하는 태스크는 위의 상태도에서 Wait for event 상태가 되며 다른 태스크/인터럽트 핸들러가 메시지/메일을 제공(Posting)하지 않는 한 계속 그 상태를 유지하며 CPU의 점유권을 가질 수 없다.

### 2.3 uC/OS를 이용한 응용 프로그램의 구축

uC/OS를 이용한 응용 프로그램의 작성은 1) uC/OS의 목표 하드웨어 환경 및 CPU로의 이식 2) 목표 하드웨어에 장치된 각종 디바이스를 응용 프로그램에서 이용하기 위한 디바이스 드라이버의 제작 3) 요구조건에 맞는 실제 응용프로그램의 작성 의 3단계로 이루어진다. 본 응용에서는 보호 계전시스템 내에서 통신 및 인터페이스를 담당하며 그 특성상 동작되는 시간범위가 밀리초 단위인 MMI(Man-Machine Interface)를 그 대상으로 하여 응용 프로그램을 작성하였다. 장착된 디바이스로는 통신을 위한 RS232C 시리얼 포트, 문자표시를 위한 LCD 패널, 사용자의 입력을 받아들이기 위한 키패드, 보호계전기의 현상태를 표시하기 위한 LED 등이 장착되어 있다.

#### 2.3.1 목표 하드웨어로의 이식

uC/OS는 기계 독립적인 코드부분과 기계 의존적인 코드부분의 두부분으로 나눌 수 있다. 기계 독립적인 코드부분은 C로 작성되어 있으며 기계 의존적인 부분은 목

표 CPU의 어셈블러 언어로 작성되어 있다. 목표 CPU 상에서 uC/OS를 동작시키기 위해서는 기계 의존적인 코드부분을 목표 CPU에 맞게 변환하는 작업을 거쳐야 한다. CPU마다 그 레지스터 구조가 다르므로 스케쥴러 및 인터럽트 발생에 의한 컨텍스트 전환 작업시 사용되었던 레지스터의 저장 및 복구 작업을 위하여는 목표 CPU에 대한 정확한 이해가 요구된다. 본 응용에서는 모토롤러 사의 범용 CPU인 MC68340에 대해 이식작업을 수행하였다. 먼저 각 CPU 및 이용되는 컴파일러/어셈블러에 적용되는 데이터 타입을 정의하여야 한다. 프로세서마다 이용되는 데이터의 길이가 다를 수 있으므로 동일한 수행을 보장하기 위해서는 이의 일치가 필요하다. uC/OS 에서 사용되는 함수 및 제공되는 서비스 함수중 기계의존적인 코드로 작성된 함수는 다음과 같다. OSTaskCreate(), OSStartHighRdy(), OSCtxSw(), OSIntCtxSw(). 또한 인터럽트 서비스 루틴을 작성할 때 일반적으로 컴파일러에서 제공하는 키워드인 'Interrupt'를 사용하는 경우가 많으나 이는 인터럽트 서비스 루틴 내에서 사용되는 레지스터만을 대피하도록 지시하는 키워드로서 uC/OS의 인터럽트 서비스를 위해서는 사용할 수 없다. uC/OS의 경우 인터럽트 서비스 루틴의 수행을 끝낸 후 수행되는 OSIntExit()함수 내에서 다시 컨텍스트 스위칭을 시도하므로 인터럽트 서비스 루틴에 진입할 때의 CPU 컨텍스트와 인터럽트 루틴의 수행을 마치고 복구해야할 CPU컨텍스트가 상이한 경우가 발생할 수 있으므로 응용 프로그램 작성자가 명시적으로 모든 CPU레지스터의 내용(컨텍스트)을 대피/복구하여야 한다. 인터럽트 서비스 루틴을 마치고 컨텍스트 스위칭이 일어나기 위한 선결 조건은 인터럽트 Nesting이 Lock되어 있지 않은 상태(즉, 인터럽트 Nesting이 가능하게 되어있는 상태)에서 다른 인터럽트가 Nesting되어있지않는 경우에, 인터럽트 서비스 루틴이 실행 된 후 실행 대기 태스크 테이블을 참조하여 인터럽트 서비스 루틴 호출전과 수행후의 태스크가 상이할 때 일어난다.

OSIntCtxSw()루틴의 내부에서 컨텍스트 스위칭을 결정하는 부분에서 내부적으로 사용하는 함수호출 및 변수들로 인하여 스택포인터가(MC68340의 경우에는 A7 레지스터) 그 개수만큼 감소하게 된다. 따라서 OSIntCtxSw() 내부에서 컨텍스트를 인터럽트 서비스 루틴 호출전의 원래대로 복구한 후 대피시키기 위해서는 이 숫자만큼의 어드레스를 더하여야 한다. 이 부분이 uC/OS이식 작업의 핵심으로서 이 부분은 목표 CPU 및 사용되는 컴파일러에 따라서 달라지게 된다. 따라서 이 숫자를 결정하는 것은 개발자의 경험적인 측면이 강하며 해당 CPU의 MDS 장비로 스택 및 레지스터를 확인하여 결정하는 것이 일반적이다. MC68340의 경우는 마이크로텍사의 MCC68K 컴파일러를 이용하였을 때,

OSIntExit() 호출시 : Program Counter Push : 4byte
SR(status register)Push : 2byte
OSCtxSw() 호출시 : PC push 4byte
총 10byte가 스택포인터에서 감소하므로 이를 보상해 주어야 한다.

### 2.3.2 디바이스 드라이버의 작성

목표 하드웨어에 장치되어 있는 디바이스는 통신을 위한 RS232C 시리얼 포트, 문자표시를 위한 LCD 패널, 사용자의 입력을 받아들이기 위한 키패드, 보호계전기의 현상태를 표시하기 위한 LED 등이 장착되어 있다. 일반적으로 널리 이용되며 또한 계전기 구현시 필수적인 통신 장치에 대한 디바이스 드라이버는 수신 측은 CPU 내에서 제공되는 인터럽트를 이용한 인터럽트 서비스 루틴과 이 서비스 루틴으로부터 uC/OS 의 내장 서비스인 메시지 큐를 이용하여 수신 메시지를 받아 처리하는 수신처리 태스크로 구성되었다. 송신 측은 하나의 태스크를 할당하여 인터럽트 서비스가 아닌 일반 태스크 서비스 영역에서 구동되도록 하여 수신 서비스가 더 높은 우선 순위 가지고 동작할 수 있도록 하였다. 또한 인터럽트 서비스 중에는 중첩된 인터럽트의 발생 시를 제외하고는 사실상 컨텍스트 스위칭이 이루어지지 않으므로 응용 프로그램 작성자의 의도와는 다르게 응용 소프트웨어의 동작을 막는 현상이 발생할 수 있으므로 응용 프로그램 작성자는 되도록 인터럽트 서비스루틴의 크기를 작게 하여 CPU를 점유하는 시간을 최소화하여야 한다. 일반적으로 인터럽트 서비스 루틴은 ISR(Inturrupt Service Routine) 와 IST(Interrupt Service Thread/Task)로 나눌 수 있다. ISR 에서는 발생한 인터럽트를 인지하여 해당 인터럽트를 처리하는 IST에게 IPC를 통하여 인터럽트 발생사실을 알린다. 이러한 방법을 통하여 ISR의 크기를 줄일 수 있다.

### 2.3.3 응용 소프트웨어의 작성

운영체제의 이식이 성공적으로 이루어 지고 목표 하드웨어에 대한 디바이스 드라이버의 작성이 완료 되었을 때 요구조건에 맞는 응용 소프트웨어의 작성이 시작된다. 본 논문에서 대상으로 삼은 MMI 하드웨어는 다음과 같은 기능을 수용하여야 한다. 1) 사용자의 키 입력을 받아들인다. 2) 사용자의 키 입력에 따른 적절한 내용을 LCD를 통하여 제시한다. 3) 사용자의 키 입력에 따른 적절한 내용을 시리얼 포트를 통하여 메인 프로세서에 전달한다. 4) 메인 프로세서에 주기적으로 데이터를 요청한다. 5) 수신된 데이터의 내용에 따라 적절한 표시를 LCD 및 LED 를 통하여 사용자에게 제시한다. MMI의 특성상 이같은 내용은 메뉴 구조 안에서 이루어지게 되며 메뉴 구조는 하나의 상태기계(State Machine)로 생각할 수 있다. 요구조건에 맞는 메뉴 구조를 구현하기 위해서 본 구현에서는 목(Tree)구조를 변형한 자료구조를 이용하여 메뉴 자료구조를 정의한 후 각 메뉴 상태에 해당하는 자료 및 처리함수를 정의하였다. 또한 이러한 상태를 나타내는 전역변수들을 보호하기 위해 uC/OS 에서 제공하는 세마포어를 적절히 이용하여 임계역역을 구현하였다. 일반적으로 실시간 시스템 개발환경은 고급 운영체제 개발환경(유닉스, 윈도우즈 NT등)에 비해 매우 열악하여 대부분 C 컴파일러 정도가 구비되어 있는 것이 현실이며 이러한 개념을 쉽게 살릴 수 있는 객체지향형 C++컴파일러는 찾아보기 어려운 것이 현실이다. 그러나 자료구조 및 함수 포인터의 적절한 이용으로 객체 지향적인 구현이 가능하며 이는 더욱 구조적이며 체계적인 응용 소프트웨어의 개발을 가능케 한다.

## 3. 결 론

실시간 환경에서 응용 소프트웨어를 구현하기 위해서 기존에는 개발자의 경험에 의존하여 목표 CPU에서 제공하는 인터럽트를 적절히 이용하여 요구조건을 구현하는 것이 일반적이었다. 경량의 실시간 커널을 이용하므로서 좀더 체계적으로 응용 소프트웨어를 개발할 수 있게 되었으며 구현이 매우 까다로운 기능의 구현도 커널에서 제공하는 서비스를 이용하여 용이하게 구현 가능하게 되었다. 또한 개발 이후의 기 개발품의 유지보수도 기존의 구현방법에 비해 용이하게 할 수 있다. 또한 공개 실시간 커널을 이용함으로서 상용 실시간 운영체제를 이용할 때 투입되는 막대한 비용-상용 실시간 운영체제로 널리 이용되는 VxWorks의 경우 개발환경 구축에 수천 만원이 투자되어야 한다-을 회피할 수 있다.

### [참 고 문 헌]


[1] Jean J. Labrosse, "uC/OS, The Realtime Kernel".
[2] MC68340 Document Set.
[3] MCC68K, LNK68K Document Set.
[4] 김태한,'RTS프로그래밍', 월간 프로그래밍 세계, 1998