특집 10

ROSEK: OSEK 기반 자동차용 운영체제

목 차

서영빈 · 김상철 · 마평수 · 최태영
(한국전자통신연구원 · 오토에버시스템즈(주))

1. 서 론

최근 몇 년간 자동차 업계의 기술 경쟁이 치열하게 진행되고 있다. 이는 고유가, 환경규제, 운전자 및 보행자의 안전 및 편의성이 강조되면서 자동차 수요가 급변하고 있기 때문이다. 지금까지 자동차에서의 기술 개발이 공기 저항과 내구성을 고려한 차체 디자인이나 엔진의 성능과 같은 기계적 측면에서 중점적으로 이루어졌다면 최근에는 전자제어장치(ECU, Electronic Control Unit)를 활용하여 안전, 편의성, 비용 등을 개선하는 전자적 측면의 기술 개발이 활발하게 진행되고 있다. 최근 한 대의 자동차에 탑재되는 전자제어장치의 수를 살펴보면 제네시스의 경우 40여개, BMW나 LEXUS의 최신 기종의 경우는 100여개나 된다[1-2]. 이와 같은 추세에 따라 전장부품이 차량 제조비용에서 차지하는 비중이 〈표 1〉에서 보는 것과 같이 2002년에는 20%에 불과했지만 2015년에는 40%까지 늘어날 전망이다[3].

한편, 자동차에 대한 소비자들의 선호가 빠른 속도로 변함에 따라 모델의 수가 다양해지고 첨단 서비스 개발로 인해 요구되는 소프트웨어의 복잡도는 크게 증가되었다. 이와 같은 변화로 인해 자동차 제조비용에서 소프트웨어 개발비용이 상당한 비중을 차지하게 되었고 자동차 업계는 이에 대응하기 위하여 응용 소프트웨어를 모듈별로 재사용할 수 있고 다른 전자제어장치에도 쉽게 이식할 수 있도록 자동차용 임베디드 시스템의 표준인 OSEK/VDX[4]를 제정하였다.

〈표 1〉 차량 제조비용 중 전장부품이 차지하는 비중

	2002년	2015년
인테리어	13%	40%
파워트레인	4%	9%
새시	1%	5%
바디	2%	2%
총계	20%	40%

본고의 구성은 다음과 같다. 2장에서는 자동차용 표준으로 제정된 OSEK 표준의 내용과 특징, OSEK 기반의 운영체제 구현 사례들에 대해서 간략히 살펴보고 3장에서는 현재 한국전자통신연구원과 오토에버시스템즈(주)가 공동으로 개발하고 있는 자동차용 임베디드 운영체제인 ROSEK(Reliable OSEK)에 관해 상세히 서술하

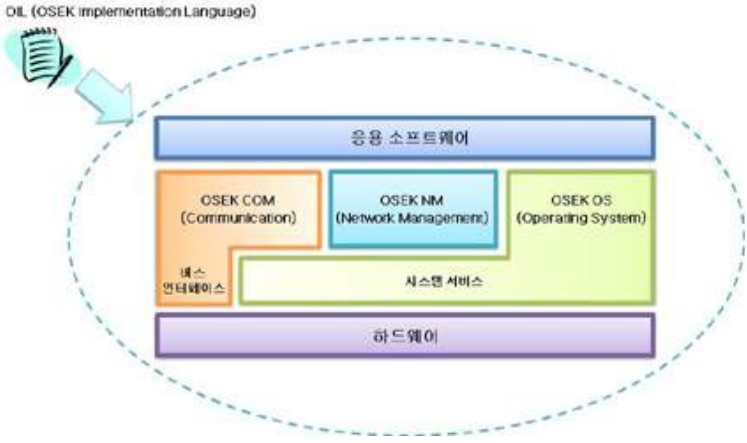
였으며 4장에서는 ROSEK에서 동작하는 간단한 응용 예제를 소개한다. 마지막으로 5장에서는 ROSEK의 개발 현황과 앞으로의 방향에 대하여 기술하며 결론을 맺는다.

2. OSEK 표준

OSEK[4]은 유럽의 자동차 회사들이 주축이 되어 시작되었으며 자동차 임베디드 시스템의 운영체제, 통신, 네트워크 관리 등의 표준 인터페이스를 위해 제정된 표준 명세이다. OSEK 표준 명세는 제조사마다 다른 인터페이스와 프로토콜 때문에 전자제어장치 간에 호환이 되지 않고 소프트웨어 개발 및 관리에 반복적이고 높은 비용이 드는 문제를 해결하기 위해 작성되었다. 모든 업계가 이 표준 명세를 지키도록 함으로써 응용 소프트웨어의 모듈별 재사용이 가능하고 전자제어장치의 제조사가 다르더라도 별도의 수정 없이 사용이 가능하기 때문에 소프트웨어의 개발 비용을 절약할 수 있다.

2.1 기존 구현 사례

OSEK 표준은 1995년에 버전 1.0이 나온 이후로 수차례에 걸쳐 개정이 이루어져 2005년에 발표된 버전 2.2.3을 마지막으로 어느 정도 안정화 단계에 들어섰으며 지금까지 이 OSEK 표준에 맞추어 다양한 운영체제들이 개발되었다. EB OSEK[5]은 현재 자동차 임베디드 솔루션 쪽에 가장 활발히 활동하고 있는 기업 중의 하나인 Elecktrobit에서 개발한 상용 운영체제이다. 응용의 특성에 맞게 커널을 생성할 수 있으며 자동차 업계에서 사용되는 거의 모든 마이크로컨트롤러를 지원하고 있을 정도로 지원이 잘 되어 있다. Vector Group의 osCAN[6]은 CAN(Controller Area Network) 통신을 위한 소프트웨어인 CANbedded를 함께 제공하여 자동차 임베디드 시스템 개발을 용이하게 한다. OSEKturbo[7]는 Freescale의 자회사가 된 Metrowerks에서 개발한 실시간 운영체제로 메모리를 적게 점유하고 문맥 전환 시간이 빠르며 응용의 재사용성을 높여준다. RTA-OSEK[8]은 ETAS 그룹의 LiveDevices에서 만든 고정 우선순위, 선점형 실시간 운영체제이다. 최악의 경우에도 커널의 CPU 부하 범위가 낮으며 실행 시간의 변동 폭이 좁아 예측 가능성이 높다는 것이 특징이다. 이외에도 독일의 Hochschule Esslingen 대학에서 uC/OS II를 기반으로 개발한 HSE Free OSEK[9], 프랑스의 연구기관인 IRCCyN에서 개발한 Trampoline[10], 일본의 uITRON을 기



(그림 1) OSEK 구성도

반으로 개발한 TOPPERS-OSEK[11], LEGO Mindstorms NXT에서 동작하는 자바 기반의 nxtOSEK[12] 등이 있다.

2.2 OSEK 표준의 구성

(그림 1)은 OSEK 표준의 구성을 나타내는 것이다. OSEK 표준 명세는 운영체제에 관련된 부분인 OSEK OS(Operating System), 통신에 관련된 부분인 OSEK COM(Communication), 네트워크 감시 및 관리에 관련된 부분인 OSEK NM(Network Management), 확정성을 위한 언어에 관련된 부분인 OIL(OSEK Implementation Language) 등으로 나누어져 있으며 이중에서 일부분은 ISO 17356에 표준화되어 있다.

3. ROSEK 운영체제

ROSEK(Reliable OSEK)은 한국전자통신연구원과 오토에버시스템즈(주)가 공동 협력하여 기존의 NanoQplus 기술을 기반으로 만든 자동차 전장장치용 운영체제이다. NanoQplus[13]는 한국전자통신연구원에서 개발한 센서네트워크 응용을 위한 운영체제이며 건설, 국방, 홈네트워크 등 다양한 응용 분야에 사용되고 있다. NanoQplus는 멀티 쓰레드를 지원하고 완전 선점형 스케줄러를 제공하여 빠른 응답성을 제공하며 사용자 응용에 최적화된 커널 설정 기능을 통해 4KB이하의 메모리를 갖는 초소형 하드웨어에서도 잘 동작도록 설계되어 있다. 이식성이 뛰어나 현재까지 MSP430, ATmega128 외에도 CC2430, PXA270, S12X 등 다양한 플랫폼에 포팅이 되어 있으며 소스 코드 공개를 통하여 무수한 검증과 피드백을 거쳐 안정성이 높다[14].

ROSEK의 기본적인 틀은 NanoQplus의 것을 가져오기는 했지만 〈표 2〉에서 보는 것과 같이 센서 네트워크 응용과 자동차 응용의 특성은 많은 차이가 있기 때문에 신뢰성과 실시간성을 높이기 위하여 많은 부분을 수정해야만 했다. 현재 구현된 ROSEK은 태스크를 최대 64개, 우선순위도 최대 64개까지 지원한다. 스케줄러는 OSEK 표준에서 제시하는 혼합 선점형 스케줄링 정책을 사용하여 운영체제 내 태스크간의 선점 능력을 극대화 하였다.

〈표 2〉 NanoQplus와 ROSEK의 비교

	NanoQplus	ROSEK
용도	센서 네트워크 응용	자동차 응용
표준성	TTA 표준	OSEK/VDX
신뢰성	보통	매우 중요
실시간성	보통	매우 높음
저전력	매우 중요	보통
스케줄러	선점형 스케줄링	혼합 선점형 스케줄링 (선점형 태스크 + 비선점형 태스크)
지원 우선순위 레벨 수	6개	최소 16개, 최대 64개
지원 태스크 수	최대 15개	최대 64개

ROSEK은 OSEK OS 표준 명세서 버전 2.2.3과 OSEK OIL 표준 2.5를 기반으로 작성되었다. (그림 2)는 ROSEK의 전체적인 구조를 나타낸다. 마이크로컨트롤러는 자동차에 사용될 수 있는 각종 전자제어장치의 하드웨어에 해당되는 부분이며, 하드웨어 추상화 계층(HAL, Hardware Abstraction Layer)은 운영체제가 하드웨어에 상관없이 일관된 방식으로 동작할 수 있도록 해주어 이식성을 높일 수 있도록 해주는 부분이다. 또한 응용에게 OSEK OS 표준에 부합하는 응용 프로그래밍 인터페이스(API, Application Programming Interface)를 제공함으로써 하드웨어나 운영체제에 독립적인 응용 개발을 보장하여 준다.

ROSEK은 크게 태스크 관리, 메모리 관리, 자원 관리, 이벤트 제어, 알람 관리, 인터럽트 처리 및 에러 처리 등의 기능들을 제공하며 〈표 3〉은 이러한 기능들과 관련하여 현재 ROSEK에서 제공하는 시스템 서비스 목록을 보여주고 있다. 이번 장의 나머지에서는 ROSEK에서 구현된 중요한 기능들을 중심으로 요약하여 서술하였다.

(그림 2) ROSEK의 구조

〈표 3〉 ROSEK 제공 서비스

	시스템 서비스	기능
태스크 관리	ActivateTask	태스크 활성화
	TerminateTAsk	태스크 종료
	ChainTask	다음 활성화될 태스크를 지정
	Schedule	태스크 스케줄링
	GetTaskID	현재 수행중인 태스크의 ID를 얻는다
	GetTaskState	호출될 당시의 태스크의 상태를 얻는다
인터럽트 처리	DisableAllInterrupts	모든 인터럽트 금지
	EnableAllInterrupts	모든 인터럽트 허용
	SuspendAllInterrupts	중첩을 허용하는 모든 인터럽트 금지
	ResumeAllInterrupts	중첩을 허용하는 모든 인터럽트 허용
	SuspendOSInterrupts	category 2의 중첩을 허용하는 모든 인터럽트 금지
	ResumeOSInterrupts	category 2의 중첩을 허용하는 모든 인터럽트 허용
자원 관리	GetResource	자원 획득
	ReleaseResource	자원 해제
이벤트 처리	SetEvent	이벤트 설정
	ClearEvent	이벤트 해제
	GetEvent	태스크의 이벤트 비트의 현재 상태를 얻는다
	WaitEvent	특정 이벤트를 기다리며 대기 상태가 된다
알람	GetAlarmBase	알람 베이스 특성을 얻는다
	GetAlarm	알람이 만료되기 전의 상대적인 틱값을 얻는다
	SetRelAlarm	상대적 알람 설정
	SetAbsAlarm	절대적 알람 설정
	CancelAlarm	알람 취소
응용 모드	GetActiveApplicationMode	현재의 응용 모드를 얻는다
시작 및 종료	StartOS	운영체제를 시작
	ShutdownOS	운영체제를 종료

3.1 시스템 시작 및 종료

(그림 3)은 ROSEK에서 시스템이 시작할 때의 동작 과정을 보여준다. CPU가 리셋이 되면 가장 먼저 하드웨어 초기화 코드가 수행되며 이후 StartOS 함수를 실행하여 운영체제를 시작한다. StartOS 함수는 설정된 응용 모드에 따라 운영체제의 동작 모드를 변경하도록 구현하였으며, ROSEK은 현재 기본 모드만 구현되어 있으며 개발자가 필요에 따라 응용 모드를 추가할 수 있도록 하였다. 커널이 실행되기 전에 사용자가 작성한 초기화 코드를 직접 수행할 수 있도록 StartupHook 함수를 제공한다.

(그림 3) 시스템 시작

ROSEK은 운영체제가 동작 중 치명적인 오류가 발생했을 경우를 대비해 ShutdownOS 함수를 제공한다. 이 함수는 사용자가 운영체제를 중지시키고자 하거나 운영체제가 내부적으로 정의되지 않은 상태에 빠졌을 경우에 호출될 수 있다. 시스템 시작 시와 마찬가지로 Shutdown Hook함수를 제공하여 시스템이 종료되기 전에 사용자가 작성한 종료 코드의 수행이 가능하도록 하였다.

3.2 적합성 등급(CC, Conformance Classes)

OSEK OS 표준에서는 응용 소프트웨어에 최적화된 운영체제를 선택하여 사용할 수 있도록 태스크의 특성에 따라 적합성 등급을 나누고 있다. 크게 기본 태스크만 존재하는 시스템(BCC, Basic Conformance Class)과 기본 태스크와 확장 태스크가 공존하는 시스템(ECC, Extended Conformance Class)으로 나누고 여러 태스크가

같은 우선순위를 가질 수 있는지 여부, 태스크의 중복 활성화 가능 여부에 따라 1과 2로 나누었다.

ROSEK은 이 표준에 따라 적합성 등급을 구현하였으며 응용 작성자에게 특정 등급을 선택하도록 하여 응용의 특성에 맞는 최적의 운영체제를 선택할 수 있도록 하였다.

3.3 태스크 관리

OSEK 운영체제는 자동차용 운영체제이므로 태스크의 실시간 특성이 매우 중요하다. 이를 위해 표준에서는 다음과 같이 3가지 스케줄링 방식을 소개하고 있다. 완전 선점형 스케줄링(Full preemptive scheduling)은 모든 태스크가 재 스케줄링 시점에서 선점이 가능한 방식이고, 비선점형 스케줄링(Non preemptive scheduling)은 모든 태스크가 자발적으로 CPU를 양도하기 전까지는 스케줄링이 일어나지 않는 방식이며, 이 두 가지를 합친 것이 혼합 선점형 스케줄링(Mixed preemptive scheduling)이다.

ROSEK은 태스크를 크게 선점 가능한 태스크와 선점 불가능한 태스크로 나누고, 선점 가능한 태스크에 대해서는 완전 선점형 스케줄링 방식을 적용하고, 선점 불가능한 태스크에 대해서는 비선점형 스케줄링 방식을 적용하는 혼합 선점형 스케줄링 방식을 지원하고 있다.

3.4 메모리 관리

자동차용 임베디드 운영체제는 응용 소프트웨어에게 실시간성을 보장해야 주어야만 하기 때문에 메모리를 할당하고 해제하는 시간이 예측 가능해야 한다. ROSEK에서는 이를 위하여 시스템 생성 시에 필요한 모든 메모리를 정적으로 할당하며 메모리의 할당 크기를 단계별로 나누어 더욱 더 정확하게 실행시간을 예측할 수 있도록 하였다.

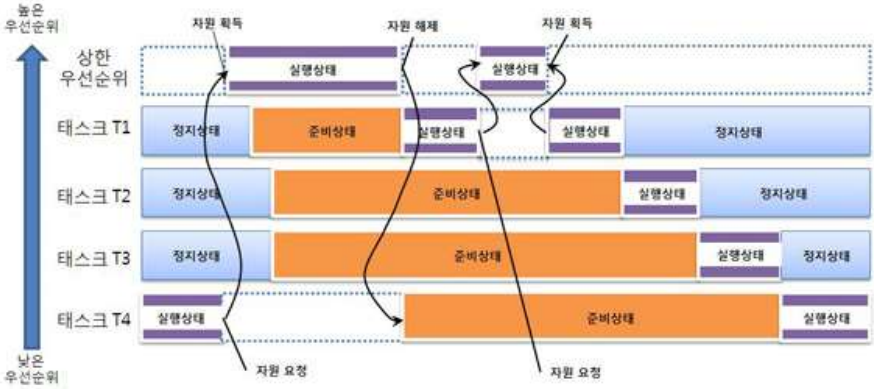
한편 태스크는 생성될 때에 고정된 크기의 스택을 할당 받아 사용하는 데 함수 호출의 깊이가 증가하거나 인터럽트 발생 횟수가 제어되지 않을 경우에는 스택이 넘치는 현상(overflow)이 발생할 수 있다. 이 경우 넘친 데이터가 다른 태스크의 영역이나 커널의 영역을 침범하여 전체 시스템의 안정성을 크게 해칠 수 있다.

ROSEK에서는 이러한 문제를 해결하기 위하여 스택이 넘치면 할당된 스택의 크기를 기록해 두고 문맥 교환이 일어날 때 이를 검사하여 스택이 넘쳤음을 알려준다. 이렇게 응용 개발자에게 일반적으로 쉽게 찾기 힘든 스택이 넘치는 시점을 알려줌으로써 안정성을 확보하도록 해준다.

3.5 자원 관리

OSEK 운영체제는 서로 다른 우선순위를 가진 태스크들이 공유 자원에 동시에 접근할 때를 위해 자원 관리 방법을 제공한다. 태스크나 인터럽트 서비스 루틴이 같은 자원에 접근하는 경우 우선순위 역전 현상이나 교착상태와 같은 동기화 문제가 발생하는 데, 이 문제 해결을 위해 OSEK 운영체제는 우선순위 상한 프로토콜(PCP, Priority Ceiling Protocol)[15]을 사용하고 있으며 ROSEK에서도 이를 따르고 있다.

(그림 4)는 ROSEK에서 선점 가능한 태스크들 사이에 우선순위 상한 프로토콜이 동작하는 메커니즘 예를 보여준다. T1과 T4가 같은 자원에 접근하려고 한다고 가정하면, T1과 T4가 사용하려는 자원의 상한 우선순위는 T1과 T4 중 높은 우선순위를 가진 T1의 우선순위보다 높은 우선순위로 할당된다. 그림에서처럼, 만일 T4가 실행되다가 공유 자원을 요청하여 획득하면 곧바로 T4의 우선순위는 자원의 상한 우선순위로 상승하게 되고 자원을 획득하고 있는 동안은 이 우선순위를 유지한다. 이후 T1이 준비상태가 되었지만 T4의 우선순위는 자원의 상한 우선순위로 높아졌으므로 선점되지 않고 계속 실행이 가능하다. 이후 T4가 획득한 자원을 해제하는 순간 T4의 우선순위는 자원을 획득하기 이전의 우



(그림 4) 선점 가능한 태스크들 간의 우선순위 상한 프로토콜 동작

선순위로 돌아가게 되고 그에 따라 CPU를 T1에게 넘겨주게 된다. 이와 같이 자원을 선점한 태스크가 준비상태에 빠지지 않도록 함으로써 교착상태를 피할 수 있고 태스크가 자원을 점유하는 기간 동안 자원의 상한 우선순위로 높여줌으로써 우선순위 역전 현상을 막을 수 있다.

3.6 이벤트 제어

태스크나 인터럽트 서비스 루틴간의 동기화를 위해 운영체제는 이벤트 메커니즘을 제공할 필요가 있다. 이벤트 메커니즘은 확장형 태스크에만 적용이 되기 때문에 기본형 태스크만으로 구성되는 BCC1과 BCC2에서는 이벤트 제어 부분을 제거하여 운영체제의 사이즈를 줄이고 실행속도를 향상시킬 수 있다. 확장형 태스크는 WaitEvent 함수를 호출하여 이벤트가 발생하기를 기다리는 대기 상태에 들어갈 수 있으며 대기상태의 태스크는 해당 이벤트가 발생하면 준비상태로 바뀌어 스케줄 되기를 기다린다. 이와 같이 이벤트 메커니즘을 제공함으로써 특정 사건에 대한 응답성을 높일 수 있다.

3.7 알람 관리

OSEK 운영체제는 시간에 따라 태스크의 활성화나 이벤트 설정을 알람 기능을 제공한다. 알람은 시스템 클록이나 카운터에 할당될 수 있으며 시스템 생성 시에 정적으로 정의된다. ROSEK에서는 카운터 값을 상대적인 값으로 지정하는 상대적 알람과 절대적인 값으로 지정하는 절대적 알람을 모두 지원하며 한번만 알리도록 하거나 반복하여 알리도록 설정할 수 있게 하였다.

3.8 인터럽트 처리

인터럽트는 외부로부터의 입력이나 타이머나 시리얼 포트와 같은 내부의 주변장치로부터의 입력을 처리하기 위한 것으로 전체 시스템의 응답성에 중요한 영향을 미친다. 인터럽트가 발생하면 인터럽트를 처리하는 인터럽트 서비스 루틴이 수행되며 OSEK 운영체제는 이 인터럽트 서비스 루틴을 운영체제 서비스를 사용할 수 없는 ISR category 1과 사용할 수 있는 ISR category 2로 나누어 처리한다. ISR category 1을 사용하면 운영체제 서비스를 사용하지 않기 때문에 최소한의 부하로 수행되어 시스템의 응답성을 높일 수 있다.

ROSEK에서는 성능에 지대한 영향을 미치는 인터럽트 서비스 루틴은 ISR category 1로 분류하고 나머지는 ISR category 2로 분류함으로써 성능과 편의성을 적절히 조율하였다. 또한 인터럽트 서비스 루틴에서는 중첩 인터럽트를 지원하여

최대 32번의 인터럽트 중첩까지 허용하고 있다.

3.9 에러 처리 및 디버깅

임베디드 시스템 개발에 있어서 에러 처리와 디버깅은 상당히 중요하다. ROSEK은 운영체제가 동작 중에 발생하는 에러를 효율적으로 처리하기 위하여 사용자에게 중앙화된 에러 처리와 분산화된 에러 처리 중에 하나를 선택할 수 있도록 해주며 디버깅을 위하여 여러 Hook 함수를 제공한다.

4. 구현 응용 실험

이 장에서는 현재 구현한 ROSEK의 스케줄링 동작을 확인하기 위한 간단한 실험 내용을 기술한다. 실험에 사용된 응용은 선점 가능한 3개의 태스크로 구성되었으며 각각 우선순위를 다르게 설정하였다. 이 실험은 자동차 전자제어장치에서 주로 사용되는 Freescale의 MC9S12XDP512 보드 상에서 이루어 졌으며 이 보드는 16비트인 S12X 마이크로프로세서를 장착하고 있고 3개의 LED와 4개의 푸시 버튼을 갖고 있다. 컴파일러는 CodeWarrior를 사용하였으며 P&E 에뮬레이터와 디버거를 사용하여 실험을 진행하였다.

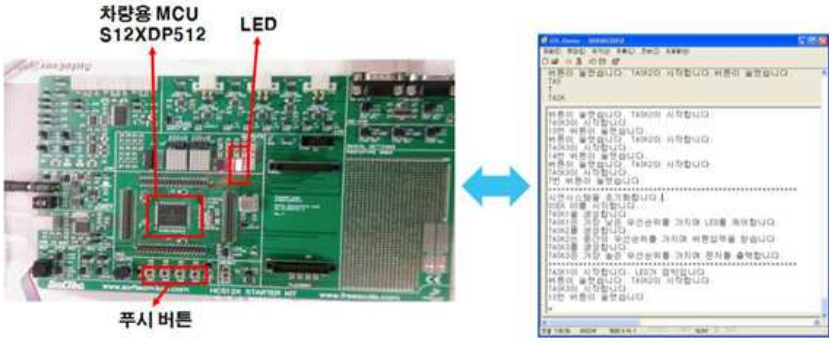
실험에서 사용된 3개의 태스크는 태스크 번호가 클수록 높은 우선순위를 가지며 선점이 이루어지는 과정을 살펴보기 위하여 처음에는 태스크 1만을 활성화시킨 채로 동작을 시킨다. 태스크들은 각각 다음과 같은 기능을 수행한다.

- 태스크 1: LED를 깜빡이는 동작을 수행한다.
- 태스크 2: 보드 상의 버튼 입력을 받으면 태스크 3을 활성화시킨다.
- 태스크 3: 화면으로 문자를 출력한다.

응용을 동작시키면 처음에는 태스크 1만이 활성화되어 있으므로 스케줄러는 태스크 1을 선택하여 실행을 하며 보드에서는 LED의 깜빡임을 확인할 수 있다. 이후 사용자가 보드 상의 버튼을 눌러 태스크 2를 활성화하면 태스크 2의 우선순위가 태스크 1보다 높기 때문에 선점이 일어난다. 태스크 2가 실행되면 태스크 3을 활성화시키고 태스크 3의 우선순위가 더 높으므로 태스크 3이 CPU를 선점하여 화면에 문자를 출력하는 것을 확인할 수 있었다. 이러한 실험을 통하여 ROSEK의 스케줄러에 의해 우선순위가 높은 태스크가 우선순위가 낮은 태스크를 선점하면서 수행되는 과정을 확인할 수 있다.

5. 결론 및 향후 방향

본고에서는 자동차 임베디드 시스템을 위한 개방형 업계 표준인 OSEK에 대해서 자세히 살펴보았고 한국전자통신연구원과 오토에버시스템즈(주)가 공동으로 개발하고 있는 자동차 전장장치용 운영체제인 ROSEK을 소개하였다.



(그림 5) MC9S12XDP512 보드와 실험 결과

ROSEK은 기존에 개발된 NanoQplus를 기술을 기반으로 하여 실시간성과 이식성이 뛰어나며 매우 안정적으로 동작하도록 설계되었다. 현재 ROSEK 운영체제는 OSEK 표준에 완전히 부합되도록 개발 완료된 상태이며 지속적인 기능 테스트를 통해 안정성을 검증하고 있다. OSEK Implementation Language(OIL)는 운영체제의 핵심과는 거리가 있어 본고에서 자세히 다루지는 않았지만 개발을 병행하여 거의 완성 시점에 와 있다. 앞으로 OIL의 개발이 완료되는 대로 한국전자통신연구원과 오토에버시스템즈(주)는 독일의 Mercedes-Benz technology에 OSEK 표준 인증을 의뢰하여 ROSEK의 표준성과 안정성을 인증 받을 예정이다. ROSEK의 인증을 마치게 되면 국내 최초의 자동차용 임베디드 운영체제로서 자리 매김할 수 있을 것으로 예상된다.

참고문헌

[1] 박미룡, 김재영, "자동차 전장 SW 플랫폼 규격(AUTOSAR) 표준화 동향," TTA 저널, 117호, pp.89-100, 2008년 6월.

[2] 정희식, "첨단 전장부품의 기술트렌드와 전망", IT SoC 매거진, 2008년 5월.

[3] McKinsey/PTW-HAWK-Survey, VDA 2003.

[4] OSEK Group, "OSEK/VDX Operating System, Communication, Binding Specification etc.", pp.1-86, Feb. 2005. http://www.osek-vdx.org

[5] Elektrobit, http://www.elektrobit.com/index.php?163

[6] Vector Informatik, "osCAN Data Sheet", May, 2007. https://www.vector-worldwide.com

[7] Freescale, "OSEKturbo OS/12 v.2.2.2 Technical Reference", http://www.freescale.com

[8] LiveDevices, "RTA-OSEK Freescale S12X with the Metrowerks Compiler", http://www.etas.com/en/products/rta__osek.php

[9] Hochschule Esslingen(University of Applied Science), "HSE Free OSEK V3 final", http://www2.hs-esslingen.de/~frruit00

[10] Bechennec J-L., Briday M., Faucou S., Trinquet Y., "Trampoline An Open Source Implementation of the OSEK/VDX RTOS Specification", IEEE Emerging Technologies and Factory Automation, pp.62-69, Sep. 2006. http://trampoline.rts-software.org

[11] TOPPERS, http://www.toppers.jp/en/index.html

[12] nxtOSEK, http://lejos-osek.sourceforge.net

[13] 송준근, 마평수, "IP-USN을 위한 센서 네트워크 운영체제 동향", 전자통신동향분석, 23권, 1호, pp.12-20, 2008년 2월.

[14] NanoQplus, http://qplus.or.kr

[15] Lui Sha, Ragunathan Rajkumar, John P. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization", IEEE Transactions on Computers, Vol. 39, No. 9, pp.1175 - 1185, Sep. 1990.

저자약력

서 영 빈

2004년 아주대학교 전자공학부 학사
2006년 아주대학교 전자공학부 석사
2006년~현재 한국전자통신연구원, 연구생, University of Science & Technology 박사
관심분야 : 임베디드 시스템, 운영체제, 컴퓨터 구조, 센서 네트워크
이 메 일: youngbin@etri.re.kr

마 평 수

1985년 서울대학교 식물병리학과 학사
1992년 City Univ of New York 전산학과 석사
1995년 Wright State Univ 전산학과 박사
1996년~현재 한국전자통신연구원, 책임연구원, 팀장
관심분야 : 센서 네트워크, 자동차용 운영체제, 임베디드 시스템
이 메 일 : pmah@etri.re.kr

김 상 철

1999년 경북대학교 전자전기공학부 학사
2001년 포항공과대학교 전자전기공학과 석사
2006년 포항공과대학교 전자전기공학과 박사
2006년~현재 한국전자통신연구원 융합소프트웨어연구본부 센서네트워크플랫폼연구팀 선임연구원
관심분야 : 센서 네트워크, 자동차용 운영체제, 임베디드 시스템
이 메 일 : sheart@etri.re.kr

최 태 영

1987년 연세대학교 물리학과 학사
1989년 연세대학교 레이저물리학 석사
1991년~1998년 삼성전자 SW센터 선임연구원
1999년~2003년 (주)애니큐 대표이사
2005년~현재 오토에버시스템즈(주) 임베디드센터 수석연구원, 팀장
관심분야 : 자동차 운영체제, SW 플랫폼, 모델기반개발, SW 테스트
이 메 일 : codezen@autoeversystems.com