StrongARM SA-1100 기반의 RTOS 커널 설계 및 구현

한성용° 박희상 이철훈
충남대학교 컴퓨터공학과
(syhan°,hspark)@pplab.ce.cun.ac.kr, chlee@ce.cnu.ac.kr

Design and Implementation of a RTOS Kernel for the StrongARM SA-1100

Seong-Yong Han°, Hi-Sang Park,Cheol-Hoon Lee
Dept. of Computer Engineering, Chungnam National Univ.

요 약 문

본 논문은 RISC CPU 인 Intel StrongARM SA-1100 을 기반으로 하는 실시간(Real-Time) 운영체제를 설계한 내용을 설명하고 있다. 본 논문에서 구현된 운영체제는 태스크들이 우선순위 기반으로 처리되는 선점형 스케쥴링 방식을 채택함으로써 실시간 운영체제의 주요 특징인 시간 결정성(determinism)을 보장하도록 하였다. Intel StrongARM SA-1100 은 고성능(High Performance), 저전력(Low Power)의 장점 때문에 모바일(Mobil) 환경에서 많이 사용되고 있다. 본 논문은 Intel StrongARM SA-1100 CPU 를 타겟으로 시간 결정성이 보장되도록 멀티 태스킹(Multitasking)과 ITC(InterTasking Communication)를 설계하고 구현한 내용에 대해 설명하고 있다.

1. 서론

실시간 시스템은 시간적인 정확성과 논리적인 정확성을 동시에 만족하는 시스템을 의미한다. 이러한 시스템은 일반적으로 연성 실시간(Soft Real-Time) 시스템과 경성 실시간(Hard Real-Time) 시스템의 두 가지 종류로 구분할 수 있다. 연성 실시간 시스템은 어떤 작업의 결과 출력이 주어진 시간 내에 이루어지지 않더라도 심각한 에러를 발생하지 않는 시스템을 말한다. 반면에 경성 실시간 시스템은 어떤 작업의 결과 출력이 데드라인을 지키지 못했을 경우 치명적인 에러를 발생시키는 시스템을 말한다. 예를 들어, 군사용 미사일 제어 프로그램, 기차 철로 제어 시스템, 핵 발전소 제어 시스템 같은 경우 정해진 시간 내에 결과를 내지 못한다면 엄청난 피해를 낼 수 있다. 대부분의 실시간 시스템들은 연성 실시간 시스템과 경성 실시간 개념을 혼용하여 사용한다. 본 RTOS 는 연성 실시간 시스템을 목표로 설계되었고, 이러한 실시간 시스템은 거의 대부분 임베디드(embedded) 시스템에 사용된다.

인터넷의 대중화와 더불어 웹과 연동되어 사용되는 정보가전 제품과 휴대용 무선통신 제품들을 제어하기 위해서는 RTOS 가 필수적으로 사용되는 소프트웨어라 할 수 있으며 시장 규모에 있어서 기존 PC 관련 업체보다 더 다양하고 거대하기 때문에 안정된 국산 RTOS 제품의 개발이 시급한 실정이다. 물론, 현재 사용되는 상용 RTOS(VxWorks,pSOS,QNX)들이 있지만 비교적 코드 크기가 크고, 실행코드(Binary Code)로 제공되기 때문에 각종 임베디드 시스템에 맞게 필요한 기능만을 최적화하여 포팅 할 수 없는 문제점들을 가지고 있으며, 기술지원을 받기 위해 외국 기업에 문의해야 하는 어려움을 안고있다.

위와 같은 여러 요인을 살펴볼 때, 비교적 메모리 크기가 작고, 저성능의 CPU 를 사용하는 small 또는 medium 규모의 임베디드 시스템에 외국의 상용 RTOS 를 구입하는 것은 오버헤드가 크므로 적당한 국산 RTOS 가 필요할 것으로 판단된다.

본 논문은 Intel 사의 StrongARM SA-1100 을 기반으로 한 RTOS 를 설계하고 구현한 내용을 설명한다. 논문의 구성은 다음과 같다. 2 장에서 실시간 운영체제의 구현 내용을, 3 장에서는 구현된 운영체제의 실험결과를 기술하고, 마지막으로 4 장에서는 결론 및 향후과제를 언급한다.

2. RTOS Kernel 설계 및 구현

실시간 운영체제는 멀티 태스킹(Multitasking)과 ITC (InterTask Communication)를 근간으로 한다. Unix™, Linux™, Windows™ 등의 일반 운영체제도 멀티태스킹과 ITC 를 제공하지만, 실시간 운영체제는 이들 운영체제와 달리 시간결정성(Determinism)을 보장하도록 설계되어야 한다.

1. 멀티테스킹(Multitasking)

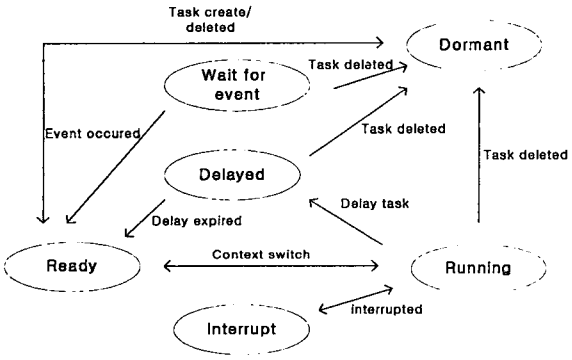
실시간 운영체제가 멀티태스킹(Multitasking)을 지원하기 위해서는 각각의 태스크마다 스택(Stack)이 존재하고, 각각의 컨텍스트(Context)가 저장되어야 한다. 또한, 각각의 태스크는 여러가지 상태가 존재하게 되고, 시간 결정성을 지원하기 위해서는 우선순위(Priority) 기반의 선점형(Preemption) 스케쥴러(Scheduler)를 포함해야 한다.

1.1 태스크의 상태

개발된 실시간 운영체제는 [그림 1]처럼 다음과 같은 상태들을 가진다.

- Dormant
  메모리에는 존재하나 스케쥴링의 대상이 아닌 TCB 만을 가지고 있는 상태이다.

- Ready
  태스크가 생성되거나 CPU 의 제어 권을 사용하기 위해 기다리고 있는 상태이다.
- Running
  태스크가 현재 CPU 를 점유 사용중인 상태이다.
- Delayed
  주어진 클럭틱(Clock Tick) 동안 자기 스스로를 스케줄 대상에서 제외시키며 지연(Delayed) 되는 상태이다.
- Wait
  다른 태스크나 인터럽트(interrupt)에 의한 이벤트의 생성을 기다리는 상태로 스케줄링 대상에서 제외된다.
- Interrupted
  운영체제에 인터럽트(Interrupt)가 요청될 경우 이를 처리하는 인터럽트 서비스 루틴 (ISR, Interrupt Service Routine)으로 CPU 사용권을 넘기고, ISR 수행을 완료한 후에는 Ready 상태에서 우선순위가 가장 높은 태스크에게 CPU 사용권을 넘기는 선점 방식이 사용된다.



[그림 1]태스크 상태 전이도

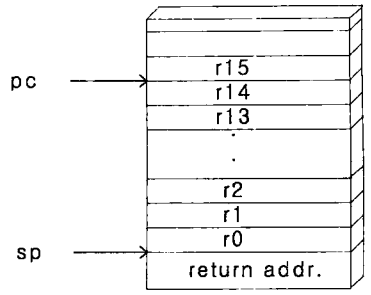
1.2 태스크의 생성 - MK_CreateTask

태스크를 생성시키는 함수로써, 해당 태스크의 스택과 TCB(Task Control Block)가 생성되고 관련 컨텍스트가 레지스터별로 분류되어 스택에 저장된다. [그림 2]는 태스크가 생성되었을 때, 초기 스택의 모습이다.

StrongARM 프로세서(Processor)는 30 개의 32bit 범용 레지스터(General Register)와 Program Counter Register, Current Program Status Register(CPSR), 다섯 개의 Saved Program Status Register(SPSRs) 를 가진다.

이 레지스터들은 빠른 컨텍스트 스위칭(context switching)을 지원하기 위해서 부분적으로 나뉘어 묶여진다. Current processor mode 에 따라 15 개의 r0-r14 까지의 범용 레지스터가 사용 가능하게 된다.

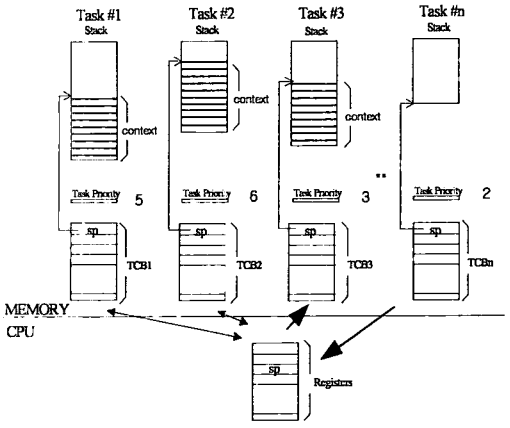
[그림 2]에서 보는 것과 같이 생성된 태스크의 스택은 초기에 태스크가 CPU 를 선점했을 때, 실행될 함수의 address 를 제외한 모든 값이 0 으로 set 되어진다. 가장 높은 우선순위를 가지는 태스크가 될 경우, 현재 CPU 상에서 실행되던 태스크의 정보는 CPU 레지스터에서 자신의 태스크로 Push 되고, CPU 를 선점한 그림상의 태스크는 CPU 레지스터에 매칭되는 정보를 CPU 레지스터에 로드 한 후, [return addr.]에 저장되어 있는 주소로 가서 함수를 실행하게 된다.



[그림 2] 태스크 스택구조

1.3 스케줄링 정책

RTOS 는 우선순위 테이블을 별도로 관리하여 정해진 시간 내에 가장 높은 우선순위의 태스크를 찾을 수 있다. 이를 기반으로 RTOS 는 서로 다른 우선 순위의 경우, 우선 순위가 높은 태스크를 선점(Preemption)하여 실행하고, 동일한 우선순위의 태스크들은 타임 슬라이스(Time Slice) 동안 차례로 수행되는 라운드 로빈 방식을 따른다.



[그림 3] Context Switching

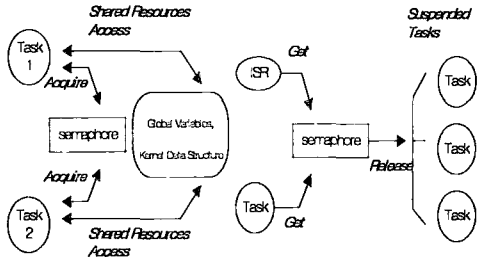
타이머 인터럽트 서비스 루틴(Timer Interrupt Service Routine)은 태스크가 소유한 틱 값을 감소시켜, 대기중인 태스크의 타임 슬라이스가 0 이 되면, ready 리스트에 포함시킨다. 본 논문에서 구현한 운영체제의 1 틱은 10ms 로 설정되었다.

1.4 ITC(InterTask Communication)

태스크간 통신은 멀티태스킹을 지원하는 운영체제에서 병행수행이 가능하게 하며 자원 공유, 계산 속도 증가, 모듈화 등의 이점을 제공한다. 본 논문에서 구현된 RTOS 는 동기화(Synchronization)와 태스크간의 통신을 위해서 세마포어, 메시지 큐, 메시지 메일 박스를 제공한다.

세마포어(Semaphore)는 공유자원을 배타적으로 사용하기 위해 제공되는 것으로서 공유자원 혹은 특정한 이벤트(event)를 기다리는 태스크들은 대기(Pend) 리스트로 구성된다. 기다리던 공유자원이 획득하게 되면, 메시지 메일박스, 메시지 큐를 통해서 메시지가 전달되고 대기 리스트 중에서 가장 높은 우선순위를 가진 태스크가 그 메시지를 받아 자원에 대한 세마포어를 얻게 된다. 세마포어의 값이 양수이면 사용

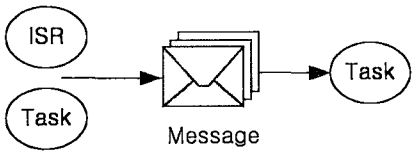
가능한 리소스의 수를 의미하고 음수이면 대기중인 태스크의 수를 의미한다. [그림 4]에서 태스크들이 공유자원에 대한 세마포어를 획득하는 과정을 보여주고 있다. 태스크들은 세마포어를 얻기위해 대기(pend) 리스트에서 대기하다가 세마포어가 해제되면, 가장 높은 우선순위의 태스크가 그 세마포어를 얻게된다.



[그림 4] Semaphore

- 메시지 큐(Message Queue)

메시지 큐는 하나 또는 그 이상의 메시지를 태스크들에게 전달함으로써 태스크간 통신과 동기화를 제공한다. 메시지 큐는 메시지들의 배열이며, 메시지는 FIFO 로 처리 된다. 메시지 큐의 메시지들은 태스크 뿐만 아니라 인터럽트 서비스 루틴에서도 보낼 수 있다..



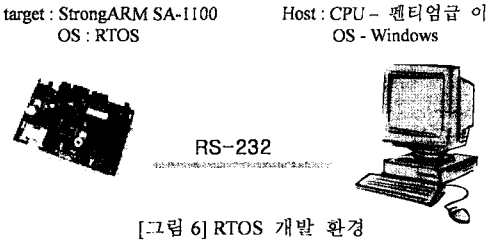
[그림 5] 메시지 큐

- 메시지 메일박스(Message Mailbox)

메시지 메일박스는 하나의 이진 세마포어(Binary Semaphore)와 유사하지만, 메시지를 수반한다는 점에서는 메시지 큐와 같다.

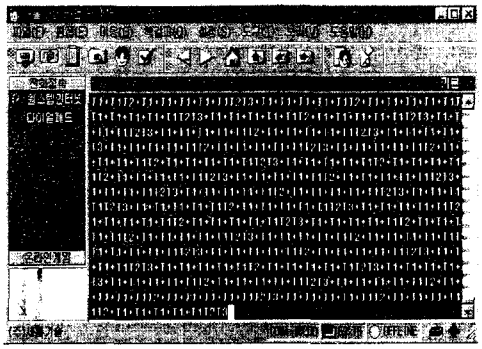
4. 실험결과

본 논문의 구현은 Intel StrongARM SA-1100 타겟보드(TargetBoard)를 기반으로 구현하였다. RTOS 개발환경은 [그림 6]와 같이 구성하였다.



[그림 6] RTOS 개발 환경

크로스 컴파일러(Cross-Compiler)는 ARM SDT V2.11 을 사용하였고, 만들어진 실행이미지(Image)는 롬 라이터를 이용해 롬에 다운로드(download)후 실험결과를 RS-232 로 연결된 호스트 컴퓨터에서 통신프로그램(새롬 데이터맨)을 이용하여 확인하였다. 간단한 메시지를 출력하는 태스크들을 우선순위와 지연시간(Timedelay)에 차별을 두어 생성하는 모듈을 미리 추가하였다. 태스크 생성시 정해놓은 우선순위와 지연시간에 맞게 정상적으로 멀티태스킹이 이루어 지는 것을 확인할 수 있었다.



[그림 7] 출력 결과

5. 결론 및 향후 과제

본 논문에서 구현된 RTOS 는 기본적으로 멀티태스킹을 위한 커널구조(kernel structure)와 태스크 관리 및 스케쥴링, 태스크간 통신 및 동기화 등의 서비스를 제공하고 있으며, 부가적인 기능으로 동적 메모리 관리, 디바이스 드라이버 인터페이스, 타이머, 향상된 인터럽트 처리 등을 모듈별로 지원함으로써 임베디드 응용프로그램의 요구에 맞게 필요한 모듈만을 선택하여 시스템에 적재함으로써 적은 메모리를 소모하면서도 최상의 성능을 가질 수 있도록 구현되어 있다.

또한, StrongARM 을 기반으로 구현되었으나 이식성과 성능을 고려, 하드웨어 의존적인 부분을 효율적으로 분리시키고 최소화 시킴으로써 타겟(target)이 변해도 이식(porting)이 쉽도록 설계되었으므로 적은 메모리의 저 성능 CPU 를 사용하는 소형 또는 중형 시스템에 사용하기 적합하다.

향후 연구할 부분으로는 실시간으로 전해지는 다량의 정보들을 정확히 처리하기위해 태스크 관리 및 스케쥴에 대한 더 많은 실험이 남아있고 POSIX(Portable Operating System Interface)를 지원할 수 있도록 표준화를 위한 노력이 계속 되어야 할 것이다.

6. 참고문헌

[1] **Advanced RISC Machine Ltd(ARM), "ARM Software development Toolkit User Guide", 1996.**

[2] **Jean, J. Labrosse, "uC/OS The Real-Time Kernel", R&D Publications, 1992.**

[3] **Intel, "StrongARM™ SA-1100 Microprocessor Technical reference Manual" ,Dec. 1998.**

[4] **Z16C30 USC User's Manual, Zilog Inc., 1997.**