개방형 구조 실시간 운영체제 커널 설계*

박희상[0] 정명조 조희남 이철훈
충남대학교 컴퓨터공학과
{hspark, mjjung, hncho, chlee}@ce.cnu.ac.kr

Design of Open Architecture Real-Time OS Kernel

Hee-Sang Park[0], Myoung-Jo Jung, Hui-Nam Cho, and Cheol-Hoon Lee
Dept. of Computer Engineering, Chungnam National Univ.

요 약

실시간 운영 체제(Real-Time OS)는 특정 태스크가 정해진 시간 안에 수행될 수 있도록 시간 결정성(Determinism)을 보장하는 운영 체제이다. 실시간 운영체제는 멀티태스킹(Multitasking) 및 ITC(InterTask Communication 혹은 IPC, InterProcess Communication)을 제공한다는 점에서는 일반 운영체제인 Unix™, Linux™, Windows™ 등과 같지만, 시간 결정성을 보장한다는 점에서 일반 운영체제와 다르다. 본 논문에서는 전부 혹은 일부의 소스 공개를 고려한 개방형 구조를 기반으로 하여, 응용프로그램 개발자에게 보다 나은 융통성과 편의를 제공할 수 있도록 실시간 운영체제 커널을 설계한 내용을 설명하고 있다.

1. 서론

실시간 운영체제는 임베디드 시스템에 사용되는 대표적인 운영체제이다. 임베디드 시스템은 자원(메모리, 전원, 프로세서 처리 속도 등)이 제한된 환경에서 미리 정해진 특정 기능(function)을 수행하도록 설계된 시스템을 의미한다[1][2][5]. 이러한 특성은 워드프로세서, 스프레드시트, 게임 등의 응용프로그램을 폭 넓게 수행할 수 있는 데스크탑 컴퓨터와 비교된다. 실시간 운영체제(이하 RTOS)는 시간 결정성(Determinism)과 작은 크기의 실행이미지(Image)를 특징으로 하며, 초기에는 군사용 제어, 산업 기기 제어, 원자력 발전 기기 시스템 분야와 같은 특수한 목적으로 사용되어 왔으나 1990 년대 중반 이후에는 네트워크 장비, 정보 가전 등 여러 분야에서 폭 넓게 사용되고 있다. RTOS 도 Unix™, Linux™, Windows™ 등의 범용 운영체제와 같이 멀티태스킹, IPC(Inter-Process Communications), 인터럽트 서비스 루틴(Interrupt Service Routine), 메모리 할당(Storage Allocation) 등의 기능을 제공한다. 그러나, RTOS 는 범용 운영체제와 달리 시간 결정성을 제공할 수 있도록 내부적으로 특화된 자료구조 및 수행 루틴을 포함하게 된다[1].

이전까지는 RTOS 시장은 여러 제품들이 각개 약진하는 형태로 유지되어 왔는데, 최근에 WindRiver 사의 VxWorks™를 중심으로 재편되면서 상대적으로 시장점유율이 낮은 제품 및 후발 제품들은 마케팅의 한 전략으로 소스 공개를 통해 제품의 성능 및 안정성을 공인 받고자 노력하고 있다. 이와 함께 소스 공개는 부수적으로 응용프로그램 개발자에게 더 나은 융통성과 편의를 제공하게 되었다.

본 논문은 시스템 API 함수 원형은 물론 RTOS 커널의 자료구조 및 소스 공개를 고려하여 시스템 자원을 좀 더 효율적으로 사용할 수 있도록 하고 응용프로그램 개발자에게 보다 나은 환경을 제공할 수 있도록 RTOS 커널을 설계하고 구현한 내용을 설명하고 있다. 본 논문은 2 장에서 관련 연구를, 3 장에서 RTOS 커널 설계를, 4 장에서 테스트 환경 및 결과를, 5 장에서 결론 및 향후 연구 과제에 대해 기술하고 있다.

2. 관련 연구

2.1 실시간 운영체제(Real-Time OS)

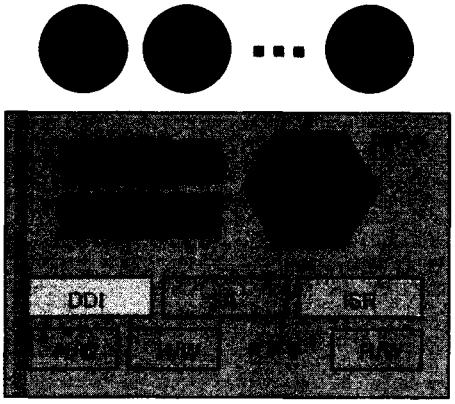
RTOS 에서 가장 중요한 부분은 스케줄러(Scheduler)로서 여러 가지 기능들을 관리하는 태스크들에게 CPU time 을 할당해주는 기능을 한다. 고전적인 스케줄러는 대드라인(Deadline) 이라는 시간 제한을 기준으로(time driven) 동작하는데 대드라인은 이벤트 발생 후부터 그 이벤트 처리를 시작할 때까지의 허용 가능한 시간 이다. 여러 가지 스케줄링(Scheduling) 알고리즘 들이 발표되었는데, 단일 프로세서(single processor) 시스템에서는 EDF(Earlist Dealine First) 알고리즘이 최적이라고 알려져 있다. EDF 는 여러가지 이벤트들이 발생했을 때 대드라인까지의 시간 여유가 가장 짧은, 즉 급한 이벤트부터 처리하는 알고리즘으로 다른 이벤트 처리 중이라도 급한 이벤트부터 처리해 준다. 그러나 대드라인을 기준으로 동작하는 RTOS 는 매우 특별한 시스템에서만 의미가 있고 대부분의 시스템은 대드라인이라는 기준이 그다지 중요한 사항은 아니다. 그래서 RTOS 를 2 가지 종류로 나눴는데, 대드라인 이라고 표현되는 고전적인 실시간 시스템을 경성 RTOS(Hard Real-Time system)이라고 부르고, 대드라인을 반드시 지키지 않아도 잘 수행되는 되는 시스템을 연성 RTOS(Soft Real-Time system)이라고 부른

* 이 논문은 BK21 대전, 충남 정보통신인력양성 사업단의 RA 연구비 지원에 의한 것임

다. 연성 RTOS 에서 사용되는 RTOS 는 대드라인이 아닌 Priority 를 기준으로 운영된다. 태스크(task)들 중에 Priority 가 가장 높은 이벤트가 무조건 먼저 실행된다. UNIX™, Linux™ 등의 범용 운영체제는 기본적인 스케줄러의 기능이 일반적인 RTOS 와 크게 다르지는 않지만 태스크 생성 시간, 태스크 전환 시간, 커널 모드에서 유지되는 시간 등이 다르다. RTOS 의 경우 커널 모드에서 소비되는 시간, critical section 의 최대 크기, interrupt 응답 시간 등을 최소화하고 worst-case 를 계산할 수 있도록 만들어진다는 점에서 범용 운영체제와 다르다.

3. 개방형 구조 RTOS 설계

3.1 RTOS 커널 전체 구성

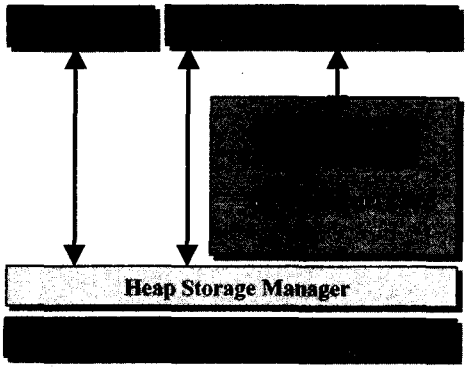


[그림 1] 실시간 운영체제 전체 구성

* DDI : Device Driver Interface
SA : Storage Allocation
ISR : Interrupt Service Routine

3.2 Storage Allocation

동적 메모리 관리는 범용 운영체제에서와 마찬가지로 임베디드 시스템(Embedded System) 및 RTOS 의 기본이 된다. 그런데, 동적 메모리 관리는 작은 크기의 메모리를 사용하는 RTOS 에서 더욱 주의하여 사용되어져야 한다[2][4][7].

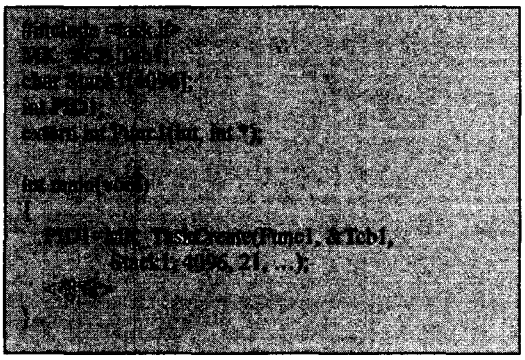


[그림 2] Storage Allocation 관리 체계

범용 운영체제에서 제공되는 동적 메모리 관리체계를 RTOS 에 그대로 적용하여 사용할 경우, 시간 결정성을 파괴하고 (외부) 단편화 문제를 일으킬 수 있다. 이와 같은 문제는 시스템의 안정성을 심각하게 훼손할 수 있다. 동적 메모리 관리 체계는 [그림 2]와 같이 2 단계로 구성된다. 즉, 시스템 부팅(Booting) 초기 임의의 크기 메모리를 할당 받을 수 있는 Heap Storage Manager 와 이를 통한 고정된 메모리 블록을 사용할 수 있도록 하는 Memory Pools 을 의미한다. 시스템 부팅 초기에는 시간 결정성 및 힙(Heap) 메모리 단편화 문제가 발생하지 않으므로, 내부적으로 항구적으로 필요한 메모리 및 Memory Pools 에서 필요로 하는 메모리를 미리 할당 받는다. 이후부터는 Memory Pools 을 통한 고정된 메모리를 사용하면 된다. 각각의 Memory Pool 은 고정된 크기의 블록을 가지며, 여러 종류의 메모리 블록을 가질 수 있다.

3.3 태스크 생성

태스크의 생성은 개방형 구조 RTOS 커널의 핵심이다. 개방형 구조 RTOS 는 소스 공개를 고려한 RTOS 를 의미한다. 각각의 태스크는 이를 관리하는 태스크 관리 블록(Task Control Block)과 1:1 사상(mapping)이다. 따라서 생성 가능한 태스크 관리 블록의 최대수는 시스템에서 생성 가능한 태스크의 최대수가 된다.

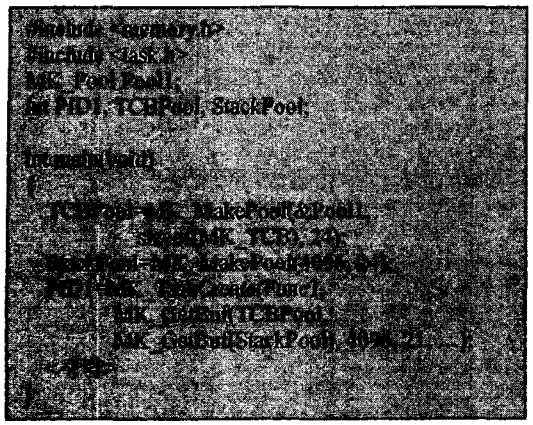


[그림 3] 전역 변수 사용 태스크 생성

[그림 3]와 같이 태스크 생성 함수 원형(MK_TaskCreate)를 제공하게 되면 MK_TCB 를 사용하여 선언된 배열의 수만큼 태스크 생성이 가능하다. Func1 은 태스크 함수이고, Tcb1 은 해당 태스크의 태스크 관리 블록이고, Stack1 은 해당 태스크의 스택의 끝이고, 4096 은 스택의 크기(스택의 시작은 Stack1-4096)이고, 21 은 해당 태스크의 우선 순위이다. PID1 은 Tcb1 의 시작주소로서 다른 태스크 관리 블록과 겹치지 않는다. 또한 PID1 을 캐스트 연산 (MK_TCB *)를 통해 태스크 관리 블록의 포인터(Pointer)로 역 변환이 가능하다. [그림 3]의 경우, 여러 가지 점에서 장점이 있다.

- 필요한 양의 메모리 획득 가능 : 응용 프로그램 개발자가 필요한 태스크의 수만큼 메모리 할당 가능
- 태스크 마다 스택의 크기 조절 용이 : 특정 태스크는 스택이 더 많이 필요하거나 혹은 더 적게 필요할 경우 용이
- PID 와의 변환 및 역 변환 용이 : 가상 ID(Virtual ID)와 달리 Free ID 획득이 필요 없고, 역 변환이 (MK_TCB *) 캐스트 연산만으로 충분
- 동적 메모리 사용 배제 가능 : 태스크의 수가 적거

나 동적 메모리가 필요 없는 시스템의 경우 동적메모리 관리 체계를 제외하고 사용 가능

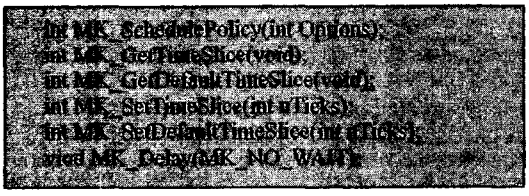


[그림 4] 동적 메모리 사용 태스크 생성

[그림 3]와 같이 전역 변수를 사용하여 태스크를 생성할 경우, 전역변수가 모두 실행 이미지가 커지는 문제가 있는데 이 경우 [그림 4]와 같이 동적 메모리를 사용하면 이 문제를 해결할 수 있다. 앞서 설명한 태스크 생성은 세마포어, 메시지 큐의 생성등에도 동일하게 적용할 수 있다.

3.4 스케줄러(Scheduler)

스케줄러는 범용 운영체제와 RTOS 를 구분하는 주요 요소이다. 대부분의 상용 RTOS 는 우선순위(Priority) 기반의 선점형(Pre-Emptive) 스케줄링(Scheduling)을 수행한다[3][4]. 이에 대한 내용은 잘 알려져 있다. 특정 이벤트의 발생이 후 바로 다음에 수행될 가장 높은 우선순위의 태스크를 찾기 위해 사상(mapping)테이블을 사용한다. 기본은 64 개의 우선순위이지만 약간의 변형으로 255 개 및 그 이상까지 확장이 가능하다[6]. 특정 태스크들은 동일한 우선순위에서 동작해야 하는 경우가 있는데, 이 경우 Round-Robin 정책뿐만 아니라 FIFO 가 가능하도록 하였다. 이를 위해 별도로 [그림 5]와 같이 스케줄링 정책을 위한 시스템 API 를 제공한다.

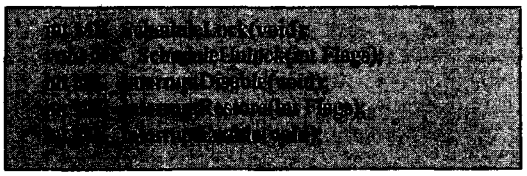


[그림 5] 스케줄링 관련 시스템 API

FIFO 스케줄링의 경우, 할당된 TimeSlice 를 전부 소모하지 않은 상태에서 필요로 하는 연산 수행을 모두 마친 경우, MK_Delay(MK_NO_WAIT)를 사용하여 해당 우선순위의 리스트 맨 뒤로 이동할 수 있다. 이 경우 동일한 우선 순위의 실행 대기 태스크가 먼저 수행되고, 태스크가 존재하지 않으면 바로 다시 수행하게 된다.

3.5 Interrupt Service Routine

ISR 은 Scheduler 와 마찬가지로 RTOS 의 주요 특징중의 하나이다. RTOS 의 ISR 은 Nested ISR 이 가능해야 한다. 즉, 특정 인터럽트가 수행되는 도중 다른 인터럽트가 중복해서 처리 가능해야 한다. 이를 위해 [그림 6]과 같이 시스템 API 를 제공한다.



[그림 6] ISR 관련 System API

4. 테스트 환경 및 결과

본 논문에서 기술하고 있는 RTOS 는 32-bit CPU 를 대상으로 설계되었으며, ARM 920T 를 기반으로 한 삼성 S3C2800™ 32-bit RISC MicroProcessor 에서 테스트 중이다. 컴파일러는 ARM SDT 2.51 을 사용하였다. 실행 이미지는 30K 정도이다.

5. 결론 및 향후 연구 과제

본 논문은 소스의 일부 혹은 전부를 공개함을 전제로 RTOS 커널을 설계할 경우, 응용 프로그램 개발자에게 더 많은 융통성과 자원(특히 메모리) 절약의 장점을 가짐을 설명하였다. 디버깅을 위한 Hooking 함수 및 통계 정보 함수들은 추가로 설계되고 구현되어야 하겠다.

6. 참고문헌

[1] http://www.inestech.com

[2] David Stepner 외 2 명, " Embedded Application Design Using a Real-Time OS" , IEEE, 1999

[3] Jean, J. Labrosse, " μ C/OS The Real-Time Kernel" , R&D Publications, 1992.

[4] David Lafreniere, " An Efficient Dynamic Storage Allocator" , Embedded Systems Programming, Sept. 1998.

[5] C.M.Krishna, Kang G.Shin, " Real-Time Systems" , The McGraw-Hill Companies, Inc.1997

[6] 오선진, 이철훈, " Deterministic Task Scheduling for Real-Time GPS Controllers" , GNSS Workshop, Vol. 8, 2001

[7] 박희상, 안희중, 김용희, 이철훈, " Design and Implementation of Memory Management Facility for Real-Time Operating System, iRTOS™" , 한국정보과학회, 2002