μ C/OS-II에서의 확장된 시분할 스케줄링 설계

김태호*, 박철동**, 장용호**, 김창수*
부경대학교 전자계산학과* 부경대학교 전산정보학과**

The Design of Expanded Time Slice Task Scheduling in μC/OS-II

Tae-Ho Kim*, Chul-Dong Park**, Young-Ho Jang**, Chang-Soo Kim*
Dept. of Computer Science, PuKyong National University*
Dept. of Computer and Information, PuKyong National University**

요 약

다수의 RTOS(Real-Time Operating Systems)가 개발되면서 다양한 분야에서 적용되고 있다. 현재 RTOS는 가전제품과 같은 특정 목적의 장치들을 지원하기위해 설계되어 있기 때문에 목적에 따라 설계 방향이 다양하게 개발되어 있다.

본 논문에서는 μ C/OS-II 버전 2.03 환경에서 동일한 우선 순위를 가진 여러 개의 태스크들을 관리하는 방법과 우선 순위에 기반한 선점 메커니즘을 지원하는 μ C/OS-II에서 동일 우선 순위를 가진 태스크들을 동작하도록 하는 기능을 추가하였다. 이를 위해 본 연구에서는 μ C/OS-II의 커널 구조를 변경하여 시뮬레이션을 수행하였다.

1. 서론

내장형 실시간 운영체제는 현재 국내외적으로 다수의 제품들이 개발되고 있으며, 많은 가전 제품들은 이러한 내장형 운영체제를 사용하고 있다. 내장형 실시간 운영체제는 일반적인 범용 운영체제와는 달리 특수 목적에 맞게 제한된 기능을 가지고 정해진 시간 제약조건을 만족시킬 수 있도록 설계되어 있으며 범용 운영체제에서의 Windows 제품군 처럼 한 운영체제가 독점하는 형태가 아니라 다양한 분야에 많은 운영체제들이 사용되고 있다[2]. 현재 사용되고 있는 대부분의 사용 실시간 운영체제들은 우선 순위에 따른 선점(Preemption) 방식을 기본으로 하여 동일 우선 순위 작업간에는 Round Robin 방식을 지원하고 있다.

본 논문에서는 RTEMS, ETOS, μC/OS-II와 같은 공개용 실시간 운영체제의 분석을 통하여 실시간 운영체제가 갖추어야 할 기능들을 살펴보고, 현재 가전제품이나 의료 제품 등의 다양한 분야에 사용되고 있지만 우선 순위에 따른 선점 방식만을 지원하는 μ C/OS-II에 대해 동일 우선 순위에 다중 작업을 할당하고 관리 할 수 있는 방법을 설계하고 다양한 어플리케이션의 개발을 통해 설계 내용을 검증하였다.

2. 관련 연구

2.1 실시간 시스템

실시간 시스템은 시스템의 수행 결과가 기능적으로 정확해야 할 뿐만 아니라, 결과가 도출되는 시간이 주어진 제약 조건을 만족시켜야 하는 시스템이라고 할 수 있다. 이러한 실시간 시스템에는 Hard Real-Time과 Soft Real-Time 시스템으로 나뉠 수 있다[6].

Hard Real-Time 시스템은 외부의 이벤트에 대해 명시된 시간 내에 응답을 하지 못했을 경우 완전한 실패나 인명 피해를 야기할 수 있는 시스템이다[4]. 예를 들자면 공항 관제 시스템이나 인공위성 발사 제어 시스템과 같은 시스템은 시간 제약 조건을 한번이라도 만족하지 못한다면 심각한 피해를 야기할 수 있다. Soft Real-Time 시스템은 역시 외부의 이벤트에 대해 응답하는 시간의 제약이 중요하긴 하지만 한계 시간을 초과하더라도 중대한 문제가 발생하지 않는 시스템이다. 예를 들어 콘베이어 벨트를 통해 자동적으로 조립되는 공장에서 로봇팔의 동작이 시간 제약 조건을 만족하지 못한다 하더라도 전체적인 효율만이 떨어질 뿐 조립 라인의 동작은 계속 될 수 있기 때문에 Soft Real-Time 시스템의 범주에 속할 수 있다.

2.2 실시간 운영체제(RTOS)의 특징

실시간 운영체제는 Unix/Linux 계열 운영체제나 Windows 계열의 범용 운영체제와는 다른 특수한 목적을 가지고 있어 다른 특성을 가지고 있다. 실시간 운영체제가 가지는 몇 가지 특징은 다음과 같다[7].

① 다중 쓰레드를 지원하고, 선점(Preemption)이 가능해야 한다.
② 쓰레드간의 우선순위를 보장하여야 한다. 이는 작업의 중요성을 판단하는 기준이 된다.
③ 쓰레드간의 동기화을 지원해야한다.
④ 운영체제의 행동이 결정적이어야 한다.
⑤ 커널 자체 크기의 신축성이 필요하다. 목적에 필요 없는 기능들은 모듈화를 통해 과감히 배제 할 수 있어야 한다.

2.3 실시간 운영체제의 멀티태스킹

실시간 운영체제에서 멀티태스킹을 수행하기 위해서는 작업의 수행 순서를 결정하는 스케줄링과 작업간의 상호 작용을 지원하는 동기화와 통신 방법이 지원되어야 한다[2]. 일반적으로 실시간 운영체제에서 스케줄링은 우선순위를 기반으로 한 선점 알고리즘을 사용하고 동일한 우선 순위 작업이 있을 경우만 Round-Robin 방식을 제공한다[5].

선점의 경우 보다 높은 우선 순위의 작업이 대기하게 되면 즉시 현재 수행되는 작업의 제어권을 높은 우선 순위의 작업에게 전달하는 형태이다. 따라서 우선 순위가 높은 작업의 응답성을 최적화 되고 결정적으로 이루어지게 된다[5].

Round-Robin 스케줄링은 그림1과 같이 동일한 우선 순위를 갖는 작업이 하나 이상 존재 할 경우 사용하는 방식으로 time slice라는 고정된 시간만큼 각 작업을 수행하고 동일한 우선 순위를 가지는 다른 작업으로 제어권을 넘기는 방식이다.

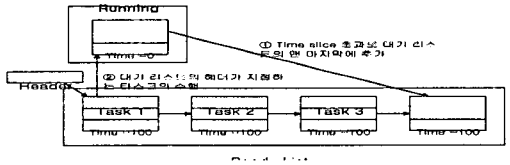

그림1. Round-Robin 스케줄링

2.4 기존 실시간 운영체제

현재 전세계에서 사용되고 있는 상용 실시간 운영체제와 본 논문에서 사용된 μC/OS-II를 살펴보겠다.

2.4.1 QNX

QNX[3]는 대형 실시간 운영체제로 우선 순위를 기반으로 선점을 지원하고 동일 우선 순위 작업간에 time slice를 이용한 Round Robin을 지원한다. 또한 POSIX와 호환 가능하며 X-Windows, motif등의 다양한 GUI와 네트워킹 기능 까지 지원 가능하다.

2.4.2 RTEMS

RTEMS[6]는 미 국방성의 군사 시스템을 관리하기 위한 목적으로 개발된 실시간 내장형 시스템으로 POSIX를 지원하고 우선 순위에 따른 선점 방식과 동일 우선 순위에 대해 Round Robin 방식을 지원하고 크로스 컴파일 환경과 기본적인 네트워킹 기능을 지원하는 공개용 실시간 운영체제이다.

2.4.3 μC/OS-II

μC/OS-II[1]는 가전제품과 같은 작은 실시간 시스템에 사용되고 있으며 기본적인 기능을 갖는 실시간 운영체제이다. 스케줄링 방식은 우선 순위에 기반한 선점 방식만을 지원하고 있으며 동일한 우선 순위에 하나의 작업만을 할당 할 수 있으므로 Round Robin 방식은 지원하지 않는다.

3. 기존 μC/OS-II의 스케줄링

3.1 실행 대기 리스트의 상태 변환 처리

μC/OS-II는 그림2와 같이 두 개의 변수 OSRGrp과 OSRdyTb[]를 이용하여 작업의 실행 대기 리스트를 관리한다.

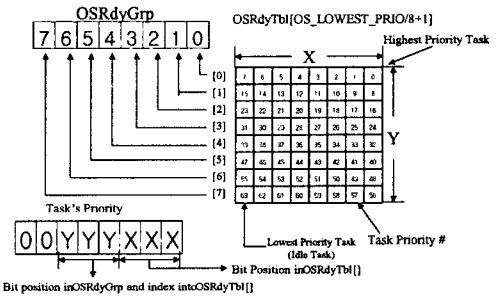

그림2. μC/OS-II의 실행 대기 리스트 관리

μC/OS-II는 가장 우선 순위가 높은 0부터 가장 우선 순위가 낮은 63까지의 우선순위를 지원한다. 그림2에서 OSRdyGrp의 각 비트는 8개로 이루어진 우선 순위 그룹을 의미하고 각 비트의 설정은 그 그룹 내의 작업에 대한 대기 상태를 나타낸다. OSRdyTbl[]의 각 비트는 해당 우선 순위 작업의 대기 상태를 나타낸다. 수행 작업을 결정하기 위해 스케줄러는 OSRdyTbl[]의 비트 중에서 우선 순위 값이 가장 낮은 작업을 선택하게 된다

3.2 이벤트 대기 리스트의 상태 변환 처리

작업은 수행 중 다른 작업으로부터의 메시지가 필요하거나 자원 획득을 위해 세마포어를 대기하는 경우 해당 이벤트를 기다리며 대기하게 된다. 이러한 이벤트를 기다리는 작업을 관리하고 이벤트가 발생하였을 경우 어떤 작업을 실행 대기 리스트에 추가 할 것인지를 결정하는 방법이 역시 필요하며. μC/OS-II에서는 세마포어, message mailbox, message queue와 같은 이벤트들의 관리를 위해 사용되는 OSEventGrp과 OSEventTbl[] 변수를 사용한다. 최고 우선 순위 작업을 선택하고 이벤트 대기 리스트에 추가 및 제거하는 방법은 실행 대기 리스트에서 사용하는 방법과 동일하다.

4. 동일 우선순위 작업 할당 설계 및 구현

본 논문에서는 동일 우선 순위에 대해 다수의 작업을 할당하기 위해 OS_Tcb_Control이라는 새로운 자료구조를 이용하고 동일 우선 순위 작업들을 관리 하기 위해 Round Robin 스케줄링을 추가하였다. Round Robin 스케줄링은 동일 우선 순위를 갖는 다수의 작업이 존재할 때만 사용하도록 조건부 컴파일을 이용해 구현함으로써 단일 우선 순위에 대해 다중의 작업을 사용하지 않는다면 μC/OS-II의 코드를 그대로 사용하고, 단일 우선 순위에 다중의 작업을 할당하기를 원하는 경우만 부가적인 처리를 하게된다. 또한 동일한 우선 순위의 작업을 리스트로 연결하여 관리하기 위한 자료구조로 OSReadyChain[]과 OSEventChain[]을 각각 실행 대기 리스트와 이벤트 대기 리스트에 사용하였다.

4.1 추가된 실행 대기 리스트

동일 우선 순위에 다중 작업을 지원하기 위해 그림2의 실행 대기 리스트에서 그림3과 같이 변경하였다.

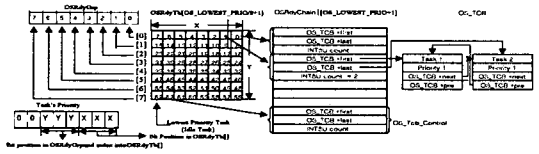

그림3. 실행 대기 리스트 관리의 설계

먼저 그림3에서는 각 우선 순위 마다 다수의 작업을 관리하기 위해서 OS_Tcb_Control이라는 자료구조를 새로 생성하고 동일 우선 순위의 작업을 TCB (Task Control Block)에 추가된 next와 prev라는 포인터를 이용해 양방향 연결 리스트로 구성한다. 그림4에서는 실행 대기 리스트에 새로운 작업을 추가하기 위한 과정을 보이고 있다.

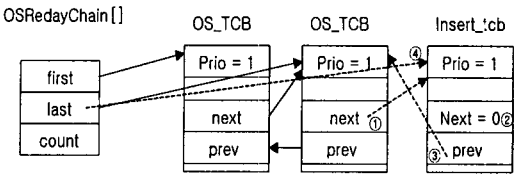

그림4. 실행 대기 리스트에 작업 추가 과정

실행 대기 리스트에서 삭제하려 할 때에는 OSRdyGrp과 OSRdyTbl[]의 해당 비트를 0으로 설정하고 선택된 대기 리스트의 OSReadyChain[]에 해당하는 TCB에 대한 링크를 삭제한다.

4.2 추가된 이벤트 대기 리스트

μC/OS-II는 이벤트 대기 리스트의 상태 변환을 실행 대기 리스트와 기본적으로 동일한 형태로 처리한다. 그러나 단일 우선 순위에 대해 다중 작업을 지원하도록 스케줄링을 변경하기 위해 그림5와 같이 이벤트 대기 리스트를 재설계 하였다.

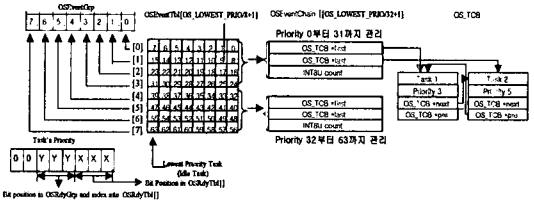

그림5. 이벤트 대기 리스트 관리의 설계

사용하는 메모리를 최소화 하기 위해 각 이벤트마다 우선 순위만큼의 OSReadyChain[] 공간을 할당 하는 대신 OSRdyGrp과 OSRdyTbl[]은 그대로 사용하고 이벤트마다 우선 순위만큼이 아닌 최대 2개까지의 OSEventChain[]을 갖도록 설계한다. 그림6에서는 이벤트 대기 리스트에 작업을 추가하기 위한 과정을 보이고 있다.

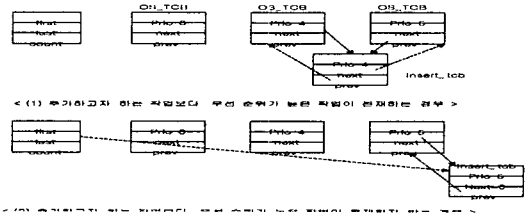

그림6. 이벤트 대기 리스트에 작업의 추가

이벤트 대기 리스트에서 삭제는 경우는 실제 이벤트

가 발생하여 이벤트 대기 리스트의 처음부터 차례로 제거하는 경우와 시간초과로 인해 임의의 작업을 제거하는 두 가지 경우로 나뉘며 임의의 작업을 제거하는 경우는 실행 대기 리스트에서 작업을 제거하는 경우와 동일하며 OS_EventChain[]의 first를 입력 작업으로 변경해주는 추가 작업이 요구된다.

4.3 Time Tick 발생 시 time slice 변경

동일한 우선 순위 작업들을 Round Robin 방식을 사용하여 처리할 때 Time Tick이 발생할 때마다 주어진 작업의 time slice를 감소시키고 0이 될 경우 다른 동일한 우선 순위의 작업으로 제어권을 넘기거나 동일 우선 순위 작업이 없을 경우 time slice를 다시 설정하는 추가적인 작업이 필요하다.

5. 결론

동일 우선 순위를 갖는 다중 작업간에 스케줄링이 올바르게 수행되는지 테스트하기 위해 아래와 같은 조건을 갖는 어플리케이션을 구성하였다.

① 시스템에서 사용하는 작업을 제외하고 나머지 4개의 작업에 동일한 우선 순위 6을 할당한다.
② 시스템 1초당 tick 횟수는 200번이며 time slice는 2tick이다.
③ 우선 순위 6인 4개의 작업은 2tick 마다 똑같은 동작이 수행되도록 구성한다.
④ 1초당 문맥 교환 횟수를 판단하기 위해 1초에 한 번씩 문맥 교환 횟수를 출력하도록 한다.

위의 조건을 만족하는 μC/OS-II의 어플리케이션 프로그램을 작성하였다. 어플리케이션 프로그램의 구현 환경은 Windows 98 환경에서 Borland C++ 3.1 컴파일러를 사용하였다. 그림7은 구현된 어플리케이션의 동작하는 모습이다.

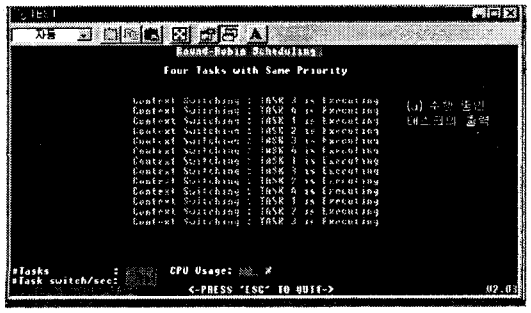

그림 7. 동일 우선 순위를 갖는 다중 작업 처리

수행 결과를 분석해 보면 전체 작업의 수는 7개이고 문맥 교환은 1초에 101번 발생했다. 101번 중 1번은 우선 순위가 가장 높은 starttask에서 다음 우선 순위의 작업으로 넘어갈 때 발생하는 문맥교환이고 1초에 200번의 tick이 발생하는데 time slice가 2 tick이기 때문에 우선 순위 5를 갖는 4개의 작업이 Round-Robin 방식에 의해 100번 문맥 교환이 발생하게 된다.

향후 연구과제로써 본 논문에서 사용한 Round Robin 스케줄링보다 효율성에서 우수한 알고리즘에 대한 연구가 필요하다. 그리고 각 스케줄링 알고리즘의 결정적인 행동에 관한 연구도 동반되어야 한다.

[참고문헌]

[1] Jean J. Labrosse, "MicroC/OS-II The Real-Time Kernel", R&D Books Lawrence, KS 66046, 1999
[2] R.Grehan, R.Moote, I.Cyliax, "Real-Time Programming", Addison Wesly, Feb. 2000
[3] Frank. Kolnick, "The QNX 4 Real-Time Operating System", Basis Computer Systems Inc. 2000
[4] J.Lala, R.Harper, L.Alger "A Design Approach for Ultrareliable Real-Time Systems", IEEE Computer, May 1991, 12-22
[5] Finkelstein. David, Hutchison. Norman C, "Real Time Threads Interface." TR-95-07, Department of Computer Science, University of British Columbia, March 1995.
[6] F.Panzieri, R.Davoli "Real Time Systems : Tutorial", Technical Report UBLCS-93-22 University of Bologna, October 1993
[7] N.Audsley, A.Burns, M.Richardson, A.Wellings, "Hard Real-Time Scheduling: The deadline-Monotonic Approach" IEEE Workshop, 1991
[8] http://www.rtems.com/RTEMS/rtems.html