우선순위 기반의 µC/OS-II 커널에서 확장된 R/R 스케줄링 연구

김태호[†] · 김창수[††]

요 약

최근의 내장형(embedded) 실시간 운영체제들은 응용 분야에 따라 다양하게 개발되고 있다. 이러한 운영체제들의 중요한 차이점은 시간적인 제한 사항을 제어할 수 있는 타스크 스케줄링 기능과 다양한 환경에서 사용할 수 있는 장치들의 개발에 있다.

본 연구에서는 우선순위 기반의 선점(preemption) 메커니즘을 지원하는 µC/OS-II 버전 2.03에서 동일 우선순위를 가진 다중 태스크들을 관리하기 위한 시분할 방식의 Round/Robin(R/R) 타스크 스케줄링 기능을 추가하였다. 이를 위해 본 연구에서는 µC/OS-II의 커널 구조에서 가장 중요한 이베트 대기 리스트 구조를 제안하고, 제안된 구조에 대해 동일 우선순위를 가지는 다중 타스크의 실행 결과를 제시한다.

A Study on the Expanded R/R Scheduling in Priority-based µC/OS-II Kernel

Tae-Ho Kim[†] · Chang-Soo Kim[††]

ABSTRACT

Recently, the existing embedded real-time operating systems(RTOSs) are being developed in terms of various modified versions in every application fields. Major characteristics and difference of these OSs lie in their distinct development of mechanisms which can be used in various environment and task-scheduling function which can control time-limited contingencies.

In this paper, we design and implement round/robin scheduling algorithm based on time-sharing with equal-priority for multiple tasks which are provided preemptive and priority task allocation function in µC/OS-II version 2.03. We propose the most important event-ready list structure in µC/OS-II kernel, and provide the running result for multiple tasks with equal priority for the proposed structure.

Key words: µC/OS-II, Round-Robin Scheduling, Real-Time Operating System, Event-Ready List

1. 서 론

가정용 가전제품과 휴대폰, PDA 제품 등이 폭발적으로 증가하면서 내장형(embedded) 실시간 운영체제는 국내·외적으로 다수의 제품들이 개발되어 사용되고 있다. 내장형 실시간 운영체제는 일반적인 상용 운영체제와는 달리 특수 목적에 맞게 제한된 기능을 가지고 정해진 시간 제약조건을 만족시킬 수 있도록 설계되어 있으며, 범용 운영체제에서의 Windows 제품군처럼 한 운영체제가 독점하는 형태가 아니라 다양한 분야에 많은 운영체제들이 활용되고 있다. 현재 판매되고 있는 대부분의 상용 RTOS들은 선점(preemption) 방식의 우선순위 스케줄링을 기본으로 지원하고 있으며, 다양한 하드웨어 플랫폼에 포팅이 가능하고 GUI 및 네트워크 지원 등 다양한 서비스를 제공한다[2].

본 연구에서는 RTEMS, ETOS, µC/OS-II와 같은 공개용 실시간 운영체제를 분석하여 내장형 RTOS

[†] 준회원, (주)하우리 기술연구소 연구원

[††] 종신회원, 부경대학교 전자컴퓨터정보통신공학부 부교수

가 갖추어야 할 기본적인 기능들을 살펴보고, 현재 가전 제품이나 휴대폰 등에서 사용하고 있는 선점 방식 및 우선순위 기반의 µC/OS-II에 대해 동일 우선순위를 가지면서 시분할 방식의 Round-Robin(R/R) 스케줄링 기능이 제공될 수 있는 구조를 제시하고, 이러한 구조에 따라 기존 µC/OS-II의 커널 수정 및 수정된 커널에서 실행되는 예제 프로그램을 제시한다.

2. 관련 연구

2.1 실시간 시스템

실시간 시스템은 제출된 작업의 수행 결과가 정확해야 할 뿐만 아니라 도출되는 시간이 주어진 제약조건을 만족시켜야 하는 시스템이라 할 수 있다. 이러한 실시간 시스템에는 크게 경성 실시간(hard real-time)과 연성(soft real-time)으로 구분할 수 있다[6].

2.1.1 경성 실시간 시스템

경성 실시간 시스템은 외부의 이벤트에 대해 명시된 시간 내에 응답을 하지 못했을 경우 완전한 실패로 판정되는 시스템이다. 예를 들면 공항 관제시스템이나 인공위성 발사 제어시스템과 같이 시간 제약조건을 한번이라도 만족하지 못한다면 심각한 피해를 야기하기 때문에 수용할 수 없는 시스템을 의미한다[4].

2.1.2 연성 실시간 시스템

연성 실시간 시스템은 외부 이벤트에 대해 응답하는 시간의 제약이 중요하지만 한계 시간을 초과하더라도 중대한 문제가 발생하지 않는 시스템이다. 예를 들면 콘베이어 벨트를 통해 자동적으로 조립되는 공장에서 로봇 팔의 동작은 시간 제약 조건을 만족하지 못하더라도 전체적인 효율만이 떨어질 뿐 조립 라인의 동작은 계속 될 수 있기 때문에 경성 실시간 시스템의 범주에 속한다[6].

2.2 실시간 운영체제의 특징

실시간 운영체제(RTOS)는 Unix/Linux 계열 운영체제나 Windows 계열의 범용 운영체제와는 다른 특성을 가지고 있기 때문에 가전제품이나 공장 제어용 시스템과 같은 다른 특수한 목적을 가진 시스템에 사용된다. 실시간 운영체제가 가지는 주요 특성들은 다음과 같다 [7].

① 예측 가능한 시간을 지원하기 위해 다중 쓰레드와 선점(preemption) 기능이 제공되어야 한다.

② 시간 제한적인 스레드(thread)들을 위해 스레드 간의 우선순위가 보장되어야 한다.

③ 스레들간의 자원공유와 통신을 위한 동기화 기능이 필요하다.

④ 실시간 운영체제의 수행은 인터럽트 지연시간, 시스템 호출 처리 시간, 운영체제와 디바이스 드라이버간의 인터럽트 마스크 시간 등이 포함된다 할지라도 결정적인 수행결과가 요구된다.

⑤ 내장형 운영체제 구성은 응용 목적에 필요 없는 기능들은 모듈화하여 커널 자체 크기를 신축성 있게 구성하는 것이 요구된다.

2.3 운영체제 스케줄링 및 동기화

대부분의 운영체제는 멀티태스킹을 수행하기 위해서 작업의 수행 순서를 결정하는 스케줄링과 작업간의 상호 작용을 지원하는 동기화 및 통신 방법이 지원되어야 한다[2].

2.3.1 스케줄링

현재 알려진 내장형 실시간 운영체제들은 우선순위 기반의 선점형 스케줄링 기법을 많이 사용한다. 이유는 스케줄링 기능을 단순화시키면서 빠른 응답을 제공하는 것이 목적이다. 그러나 작업의 우선순위 결정은 정적으로 결정되기 때문에 다중타스크 운영에서는 제한성이 존재한다. 이를 위해 내장형 운영체제에서도 모듈화된 지원 기능으로 동일 우선순위의 타스크들에 대해서는 시분할 방식의 스케줄링 기법이 필요하다[5]. 그림 1은 Round-Robin(R/R) 스케줄링 기법으로 동일한 우선 순위를 갖는 작업이 하나 이상 존재할 경우 사용하는 방식으로 시분할(time

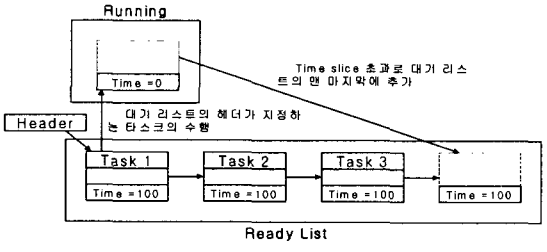

그림 1. 일반적인 Round-Robin 스케줄링

slice)라는 고정된 시간만큼 각 작업을 수행하고 동일한 우선 순위를 가지는 다른 작업으로 제어권을 넘기는 방식이다.

그림 1과 같이 대기 상태에 있는 작업의 문맥(context)은 TCB(task control block)의 연결 리스트로 구현된 대기 리스트에 보관되고, 할당된 시간이 만료된 경우 스케줄러는 현재 수행중인 작업을 대기 리스트의 마지막에 위치시키고 대기 리스트의 헤더 부분에서 새로운 작업을 선택하게 된다[2].

2.3.2 동기화

멀티태스킹을 지원하는 운영체제에서 제출된 작업은 직접 혹은 간접적으로 상호작용을 하게되는 경우가 발생한다. 작업간의 상호작용에서 공유된 자원의 경쟁을 막기 위해서는 동기화(synchronization) 기능이 반드시 필요하다. 내장형 실시간 운영체제에서도 상용 운영체제와 유사하게 동기화 기능을 제공하기 위한 방법으로 세마포어, 이벤트, 그리고 mutex 등을 사용한다[5]. 본 연구에서 제시하고 있는 μC/OS-II에서도 앞의 기능들을 지원한다.

2.4 기존 실시간 운영체제

현재 세계 시장에서 개발 중이거나 사용되고 있는 상용 실시간 운영체제는 매우 다양하다. 대표적인 상용 실시간 운영체제는 QNX와 RTEMS 등이 있으며, 본 논문에서 제시한 μC/OS-II도 최근에는 가전제품 등에 사용하는 내장형 RTOS로 상용화가 되고 있다.

2.4.1 QNX

QNX는 대형 실시간 운영체제로 확장 가능(scalable)하며 우선순위 기반의 선점 기능과 시분할 스케줄링 기능을 지원한다. 또한 POSIX와 호환 가능하며, Intel 계열 프로세서에서도 동작 가능하며, X-Windows, Motif 등의 사용자 인터페이스 기능과 다양한 네트워킹 기능까지 지원하는 규모가 큰 실시간 운영체제에 적합한 방법이다[3].

2.4.2 RTEMS

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

2.4.3 μC/OS-II

μC/OS-II는 가정용 가전제품과 휴대폰 등과 같은 규모가 적은 실시간 시스템에 활용되고 있으며, 내장형 실시간 운영체제가 갖추어야 할 초보적인 기능만을 가지고 있다. 스케줄링 방식은 우선순위 기반의 선점 방식만을 지원하고 있으며, 동일한 우선 순위에 하나의 작업만을 할당 할 수 있으므로 Round Robin 방식은 지원하지 않는다. 그리고 세마포어를 통해 작업간의 동기화를 지원하고 있으며, message mailbox와 message queue를 통해 작업간의 통신을 지원한다[1].

3. 기존 μC/OS-II의 스케줄링

3.1 실행 대기 리스트의 상태 변환

μC/OS-II는 그림 2와 같이 두 개의 변수 OSRdyGrp(1byte)과 OSRdyTbl[](8byte)를 이용하여 작업의 실행 대기 리스트를 관리한다.

μC/OS-II는 우선순위가 높은 0부터 우선순위가 낮은 63까지의 64개 등급을 지원하고 있다. 그림 2에서 OSRdyGrp의 각 비트는 8개로 이루어진 우선순위 그룹을 의미하고 각 비트의 설정은 해당 그룹 내의 작업에 대한 대기 상태를 나타낸다. OSRdyTbl[]의 각 비트는 해당 우선순위 작업의 대기 상태를 나타낸다. 수행 작업을 결정하기 위해 스케줄러는 OSRdyTbl[]의 비트 중에서 우선순위 값이 가장 낮은 작업을 선택하게 된다

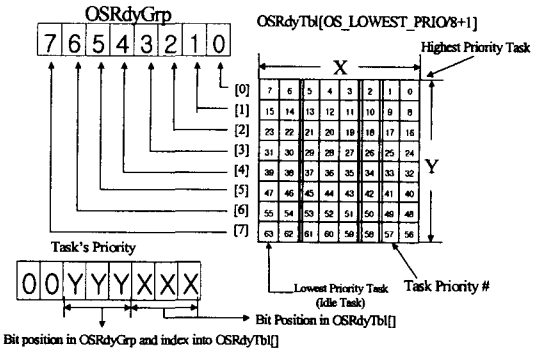

그림 2. μC/OS-II의 실행 대기 리스트 관리

3.1.1 우선순위 선정 알고리즘

대기중인 타스크 리스트들에 대해 최고 우선순위 타스크를 선택하기 위한 수행 방법은 그림 3과 같다. 우선순위 값을 계산하기 위해서는 먼저 "y" 영역에 해당하는 OSRdyGrp에서 최고 우선순위 그룹을 선택하고, 다음으로 "x" 영역에 해당하는 OSRdyTbl[] 의 해당 컬럼에서 가장 높은 우선순위의 작업을 선택하게 된다.

```
y = OSUnMapTbl[OSRdyGrp];
   /* OSRdyGrp의 비트 중 1로 설정된 가장 하위 비트 선택 */
x = OSUnMapTbl[OSRdyTbl[y]];
   /* 해당 비트 그룹 중 1로 설정된 가장 하위 비트 선택 */
prio = (y << 3) + x; /* 해당 우선 순위 계산 */
```

그림 3. 우선순위 선정 알고리즘

3.1.2 실행 대기 리스트의 작업 추가 및 제거 방법

새로운 작업을 수행하기 위해서는 실행 대기 리스트에 등록되어야 한다. 이를 위해서는 그림 4와 같이 우선순위가 속하는 그룹이 0이면 OSRdyGrp의 해당 비트를 1로 설정하고 OSRdyTbl[]의 해당 비트를 1로 설정한다.

```
OSRdyGrp |= OSMapTbl[prio >> 3];
   /* OSRdyGrp에서 해당 우선 순위 그룹을 1로 설정 */
OSRdyTbl[prio >> 3] |= OSMapTbl[prio & 0x07];
   /* OSRdyTbl에서 우선 순위에 해당하는 비트를 1로 설정 */
```

그림 4. 실행 대기리스트 작업 추가 알고리즘

실행 대기 리스트에 등록된 작업 중 수행이 종료된 작업은 그림 5와 같이 우선순위에 해당하는 OSRdyTbl의 항목을 0으로 설정하고, 만일 해당 작업의 제거로 인해 이 작업이 속한 그룹의 작업이 하나도 존재하지 않을 경우 OSRdyGrp의 해당 비트를 0으로 설정한다.

```
if((OSRdyTbl[prio >> 3] &= ~OSMapTbl[prio & 0x07]) == 0)
   /* OSRdyTbl에서 우선 순위에 해당하는 비트를 0으로 설정 */
OSRdyGrp &= ~OSMapTbl[prio >> 3];
   /* OSRdyGrp에서 해당 그룹 비트를 0으로 설정 */
```

그림 5. 실행 대기리스트 작업 제거 알고리즘

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 방법 역시 시스템의 스케줄링에서 필요하게 된다. µC/OS-II에서는 세마포어, message mailbox, message queue와 같은 이벤트들의 관리를 위해 사용되는 ECB(event control block)마다 실행 대기리스트에서 사용되는 OSRdyGrp과 OSRdyTbl[] 자료구조와 동일한 OSEventGrp과 OSEventTbl[] 변수를 사용한다. 최고 우선순위 작업을 선택하고 이벤트 대기리스트에 추가 및 제거하는 방법은 실행 대기 리스트에서 사용하는 방법과 동일하다.

4. 동일 우선순위 작업 할당 설계 및 구현

본 연구에서는 동일 우선순위의 멀티타스킹 스케줄링을 지원하기 위해 OS_Tcb_Control이라는 새로운 구조체를 추가하여 동일 우선순위 타스크들에 대해 Round-Robin(R/R) 스케줄링 기법을 추가하였다. 그림 6은 OS_Tcb_Control 구조체 정의를 위해 필요한 구조를 나타내고 있다.

```
typedef struct {
  OS_TCB *firs;t /* 첫 번째 TCB */
  OS_TCB *last; /* 마지막 TCB */
  INT32U count; /* 대기리스트에 대기하는 TCB 수 */
} OS_Tcb_Control;
```

그림 6. 제안된 OS_Tcb_Control 구조체

본 연구에서 제시한 동일 우선순위 R/R 스케줄링 기법은 다수의 작업이 존재할 때만 사용하도록 조건부 컴파일을 적용하여 구현함으로써 기존의 단일 우선순위만으로 멀티태스킹을 수행할 경우 µC/OS-II의 코드를 그대로 사용하고, 동일 우선순위 멀티타스킹 작업을 수행할 경우는 수정된 자료 구조를 사용하도록 구성하였다. 또한 동일 우선순위 작업을 리스트로 연결하여 관리하기 위한 자료구조로 OS_Tcb_Control type으로 선언된 OSReadyChain[]과 OSEventChain[]을 각각 실행 대기 리스트와 이벤트 대기

리스트에 추가하였다.

4.1 확장된 실행 대기 리스트

동일 우선순위의 다중 작업을 지원하기 위해 기존의 µC/OS-II가 가지고 있는 그림 2의 구조에서 실행 대기 리스트 구조가 추가된 그림 7과 같이 변경하였다. 그림 7에서 각 우선 순위마다 다수의 작업을 관리하기 위해서는 OS_Tcb_Control을 이용하여 동일 우선순위 작업을 TCB(task control block)에 추가된 next와 prev라는 포인터를 이용해 양방향 연결 리스트로 구성한다.

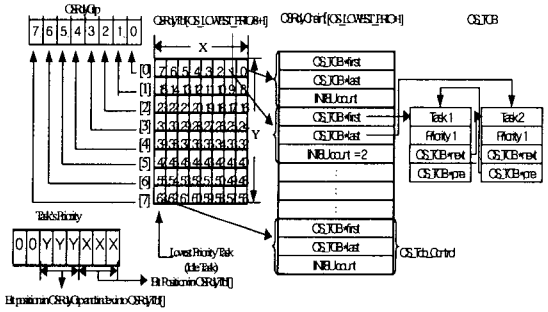

그림 7. 확장된 실행 대기 리스트 구성

그림 8은 실행 대기 리스트에 새로운 타스크를 추가하기 위한 과정을 나타내고 있는 것으로 점선은 새로운 타스크가 추가되었을 때 관리하기 위한 흐름을 표시한 것이다.

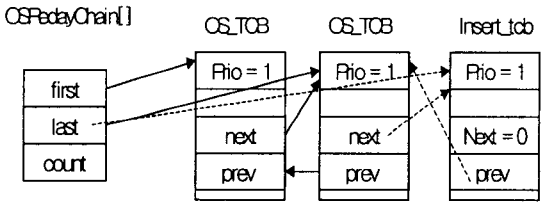

그림 8. 새로운 타스크의 실행 대기리스트 추가 구성

그림 9는 실행 대기 리스트에 새로운 작업을 추가하기 위해 필요한 코드를 나타낸 것이다. 실행 대기 리스트에서 삭제하려 할 때에는 아래 알고리즘과 같이 OSRdyGrp과 OSRdyTbl[]의 해당 비트를 0으로 설정하고, 선택된 대기 리스트의 OSReadyChain[]에 해당하는 TCB 링크를 삭제한다.

```
void Chain_Append( OS_Tcb_Control*the chain, OS_TCB*the_node)
{ if(the_chain->count == 0)
  { the_chain->first = the_node;
    the_chain->last = the_node;
    the_node->next = (OS_TCB *)0;
    the_node->previous = (OS_TCB *)0;
  }else
  { the_chain->last->next = the_node;
    the_node->next = 0;
    the_node->previous = the_chain->last;
    the_node->last = insert_tcb
  }  /* 대기 리스트의 맨 마지막에 추가 */
  the_chain->count++;
     /* 대기 리스트의 카운터 증가 */
}
```

그림 9. 실행 대기 리스트의 작업 추가 코드

4.2 확장된 이벤트 대기 리스트

기존의 µC/OS-II는 기본적으로 이벤트 대기 리스트의 상태 변환을 실행 대기 리스트와 동일한 형태로 처리한다. 그러나 단일 우선순위에 대해 다중 작업을 지원하도록 스케줄링을 변경하고 실행 대기 리스트의 상태 변환 처리를 이벤트 대기 리스트에 그대로 적용한다면 메시지 큐나 메시지 메일박스, 세마포어와 같은 각 이벤트마다 우선 순위만큼의 OSReady-Chain[]을 위한 메모리 공간을 필요로 하게 된다. 사용하는 메모리의 양을 조절하기 위해 각 이벤트마다 우선순위만큼의 OSReadyChain[] 공간을 할당하는 대신 OSRdyGrp과 OSRdyTbl[]은 그대로 사용하고 이벤트마다 최대 2개까지의 OSEventChain[]을 갖도록 설계하였다. 이와 같은 기법을 적용하여 그림 10과 같은 이벤트 대기 리스트를 재구성하였다.

그림 11은 이벤트 대기 리스트에 새로운 작업을 추가하기 위한 과정을 나타내고 있다. 이벤트 대기

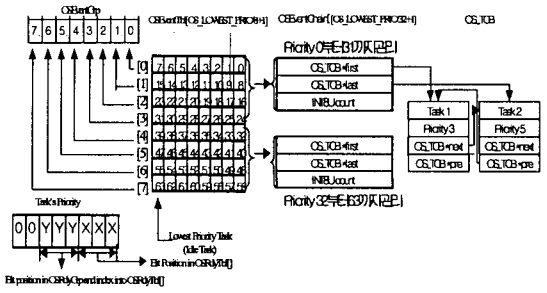

그림 10. 확장된 이벤트 대기 리스트 구성

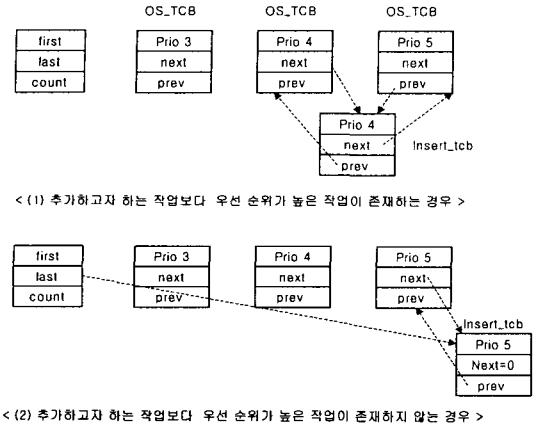

그림 11. 이벤트 대기 리스트에 작업 추가 과정

리스트에서의 삭제 경우는 실제 이벤트가 발생하여 이벤트 대기 리스트의 처음부터 차례로 제거하는 경우와 시간초과로 인해 임의의 작업을 제거하는 두 가지 경우로 구분된다. 임의의 작업을 제거하는 경우는 실행 대기 리스트에서 작업을 제거하는 경우와 동일하며 OS_EventChain[]의 첫번째를 입력 작업으로 변경해주는 추가 작업이 요구된다.

4.3 Time Tick 발생 시 시분할 변경

동일한 우선순위 작업들을 R/R 방식을 사용하여 처리하기 위해서 time tick이 발생할 때마다 수행하고 있는 작업의 시분할(time-slice) 값을 감소시키고 대기 리스트의 상태에 따라 추가적인 처리가 필요하다. 만약 작업에 할당된 time tick이 0이 될 경우 동일한 우선 순위의 대기 작업이 존재한다면 문맥 교환을 수행하고, 그렇지 않다면 time slice 만큼의 시간을 재설정하고 다시 수행하게 된다. 이러한 처리는 OSTimeSliceReduce() 함수를 통해서 이루어지고 시스템의 tick이 발생할 때마다 tick ISR(Interrupt Service Routine)에서 이 함수를 호출하도록 설계하였다.

5. 동일 우선 순위 작업 검증

본 연구는 기존의 µC/OS-II에서 제공하지 않는 동일 우선순위에 대한 다중 작업 스케줄링을 위해 시분할 방법에 의한 Round-Robin 스케줄링 알고리즘을 추가하여 구현하였다. R/R 스케줄링이 추가된 µC/OS-II의 커널이 올바로 실행되는지 확인하기 위하여 몇 가지 응용 프로그램을 작성하여 수행되는 과정을 검증하였다. 이러한 검증 작업을 위해 한 가지 방법은 이벤트 처리가 포함되지 않고 순수한 작업들로 구성된 어플리케이션과 다른 한가지 방법은 이벤트 처리를 포함하는 멀티타스킹 작업들로 구성하여 수행해 보았다.

5.1 이벤트 처리가 없는 동일 우선순위 작업

동일 우선순위를 가지는 다중 작업들에 대한 스케줄링이 올바르게 수행되는지 테스트하기 위해 아래와 같은 조건을 갖는 어플리케이션을 구성하였다.

① 어플리케이션 작업은 총 7개로 구성된다. 시스템의 idle 타임을 계산하는 우선순위 12의 idletask와 우선순위 11의 프로세서 이용률을 계산하는 stattask, 그리고 응용 프로그램의 시작을 의미하는 우선순위 5의 starttask가 수행되도록 하였으며, 나머지 4개 작업은 우선순위 6을 할당하였다.

② 모든 작업은 무한 루프를 수행하도록 하고, 문맥 교환에 의해서만 작업이 교체되도록 하였다.

③ 시스템은 1초당 tick count를 200회로 하였으며, 작업의 time slice는 2 tick으로 설정하였다.

④ 우선순위 6을 갖는 4개의 작업은 2 tick 마다 동일한 동작을 수행하도록 하였다.

⑤ 1초당 문맥 교환 횟수를 판단하기 위해 starttask에서 문맥 교환 횟수를 출력하도록 하였다.

위의 조건을 만족하도록 수정된 µC/OS-II 커널에 7개의 어플리케이션 프로그램이 어떻게 동작하는지를 테스팅하였다. 어플리케이션 프로그램의 구현 환경은 Windows 98 환경에서 Borland C++ 3.1 컴파일러를 사용하였다. 그림 12는 수정된 커널에서 작성된 어플리케이션 프로그램의 동작 과정을 캡처한 것이다. 수행 결과를 분석해 보면 전체 작업의 수는 그림 12의 (b)에서 알 수 있듯이 7개이고, 문맥 교환은 그림 12의 (c)와 같이 1초에 101회 발생하였다. 101회 중 1회는 우선순위가 가장 높은 starttask에서 다음 우선 순위의 작업으로 넘어갈 때 발생하는 문맥교환이고, 1초에 200회의 tick이 발생하는데 time slice가 2 tick이기 때문에 우선순위 6을 갖는 4개의 작업이 Round-Robin 방식에 의해 100회 문맥 교환이 발생하게 된다. 그림 12의 (a)는 현재 수행 중인 태스크가 출력하는 메시지이다.

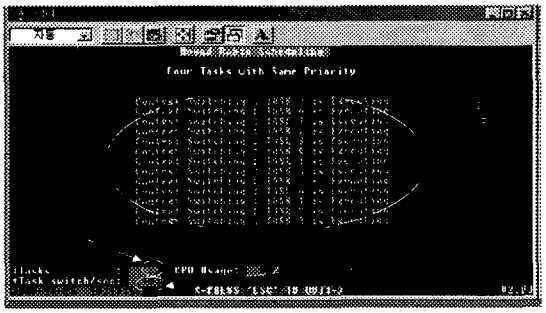

그림 12. 이벤트 처리가 없는 다중 작업 처리

5.2 이벤트 처리가 포함된 동일 우선순위 작업

동일 우선순위를 가진 다중 작업간에 단순한 스케줄링이 아닌 이벤트 처리를 포함하는 경우를 테스트하기 위해 아래와 같은 조건을 갖는 응용 프로그램을 구성하였으며, 수행 결과는 그림 13과 같다.

① 어플리케이션 작업은 총 7개로 구성된다. idletask와 stattask, starttask의 조건은 그림 12와 동일하지만, task1은 우선순위 7을 할당하며 task2부터 4까지는 우선순위 9를 할당하도록 하였다.

② 모든 작업은 무한 루프의 구조로 수행하게 하고 문맥 교환에 의해서만 작업이 교체된다.

③ 시스템은 1초당 tick count를 200으로 설정하고 작업의 time slice는 3 tick으로 설정하였다.

④ 우선순위 7을 갖는 task1은 4 tick 마다 한번씩 메시지 큐에 메시지를 Post하도록 하였다.

⑤ 우선순위 9를 갖는 task2~4까지의 작업은 2 tick마다 한번씩 메시지 큐를 Pend하도록 하였다.

⑥ 만약 메시지 큐가 비어 있으면 이벤트 리스트에 대기하게 되고, 그렇지 않다면 메시지를 전달받고 수행을 계속할 수 있게 된다.

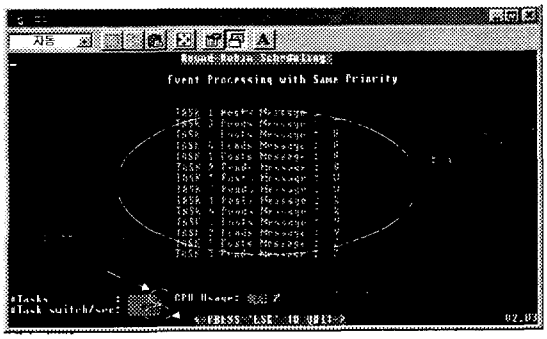

그림 13. 이벤트 처리가 포함된 다중 작업 처리

⑦ 1초당 문맥 교환 횟수를 판단하기 위해 starttask에서 문맥 교환 횟수를 출력하도록 하였다.

수행 결과를 분석하게 되면 전체 작업의 수는 그림 13의 (b)와 같이 7개임을 알 수 있고, (c)를 통해 문맥 교환은 1초에 153회 발생했음을 알 수 있다. 153번 중 50회는 우선순위가 가장 높은 starttask에서 4 tick 지연 이후에 다시 수행되는 횟수가 되고, 66번은 3개의 동일 우선순위 작업간 Round-Robin에 의한 문맥교환 횟수가 된다. 또한 37회는 메시지를 대기함으로서 발생하는 문맥 교환의 횟수가 된다. 그리고 그림 13의 (a)에는 수행 중인 작업을 출력하게 되는데 task1이 하나의 메시지를 메시지 큐에 Post하면 대기 중인 3개의 작업들이 하나씩 메시지를 Pend하게 된다.

6. 결 론

본 논문은 우선순위 기반의 선점(preemption)방식만을 지원하는 μC/OS-II에 대해 Round-Robin 스케줄링 기법을 추가하여 동일 우선순위의 다중 작업을 처리할 수 있는 확장된 μC/OS-II 커널을 설계 및 구현하였다. 이를 위해 실행 대기 리스트 관리를 위한 관리 테이블을 우선 순위마다 하나씩 할당함으로써 수행 시간의 결정성을 유지하도록 하였고, 이벤트 대기 리스트를 위한 관리 테이블을 이벤트마다 두개씩 할당함으로써 코드 크기를 최소화하도록 하였다. 또한 기존의 μC/OS-II 커널을 그대로 적용하기 위해 응용 분야가 동일 우선순위 다중의 작업을 요구하는지 여부에 따라 구분되어 적용할 수 있도록 설계하였다. 제안된 설계 원칙에 따라 본 연구에서 구현한 알고리즘의 정당성을 입증하기 위해 이벤트 처리가 없는 동일 우선순위 작업 수행과 이벤트 처리가 포함된 동일 우선순위 작업을 테스팅할 수 있는 응용 프로그램을 작성하여 실제 수행과정을 테스팅하였다. 수행된 결과는 제안된 방식으로 정확하게 동작됨을 보이고 있다.

향후 연구과제로는 본 논문에서 사용한 Round Robin 스케줄링 보다 내장형 RTOS에서는 효율성이 뛰어난 비율단조(rate monotonic) 스케줄링 기법을 추가하는 연구가 필요하며, 현재 제안된 알고리즘으로 실제 내장형 RTOS 제품을 구현하는 연구가 요구된다.

참 고 문 헌

[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. June. 2001.

[4] J.Lala, R.Harper, L.Alger "A Design Approach for Ultrareliable Real-Time Systems", IEEE Computer, May 1997, 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, 1995.

[8] http://www.rtems.com/RTEMS/rtems.html.

김 태 호

2000년 2월 부경대학교 전자계산학과 졸업 (이학사)

2002년 2월 부경대학교 전자계산학과 대학원 졸업 (이학석사)

2002년 1월~ 현재 (주)하우리 기술연구소 연구원

관심분야 : Real-Time OS Scheduling, Unix/Linux 운영체제 보안, 네트워크 보안,

김 창 수

1984년 울산공과대학 전자계산학과(공학사)

1996년 중앙대학교 전자계산학과 (이학석사)

1991년 중앙대학교 전자계산학과 (공학박사)

1992년~1996년 부산수산대학교 전임강사 및 조교수

1996년~현재 부경대학교 전자컴퓨터정보통신공학부 부교수

관심분야 : 내장형 실시간 운영체제, 네트워크 보안, PDA기반 GIS개발, 지문인식 시스템 등