iRTOS상에서의 타이머 관리를 위한 타이밍 휠의 설계 및 구현

The Design and Implementation of Timing Wheel for Timer Management in iRTOS

박세영, 정현태*, 이철훈
충남대학교, 한국전자통신연구원*

Park se-young, Jeong hyun-tae*, Lee cheol-hoon
Chungnam National Univ.,
Electronics and Telecommunications
Research Institute*

요약

실시간 운영체제 iRTOS는 타이머의 관리 기법으로 시간결정성을 위한 델타 프로세싱을 사용하고 있다. 델타 프로세싱은 타이머들 사이의 시간차로써 타이머를 관리하기 때문에 타이머의 삽입 시 해당 타이머가 삽입 될 위치를 찾는 데 있어 오버헤드가 발생한다. 이 오버헤드를 줄이기 위한 방법으로 타이머들 간의 상대적인 시간이 아닌 각 타이머의 절대적인 시간으로써 타이머들을 관리하는 방법이 있다. 본 논문에서는 절대적인 시간을 이용하여 타이머들을 관리하는 기법인 타이밍 휠을 설계 및 구현하였다.

I. 서론

임베디드 시스템에 탑재되는 실시간 운영체제는 커널의 수행시간을 예측할 수 있기 때문에 시간결정성을 제공한다. 시간결정성은 시스템의 성능을 예측하게 할 뿐 아니라, 시스템의 전반적인 안정성에도 좋은 영향을 준다. 실시간 운영체제 *i*RTOS는 이러한 시간결정성을 보장하기 위해 타이머 관리 기법으로 델타 프로세싱을 사용한다.

델타 프로세싱은 타이머 각각의 타이머 만료 시간을 타이머들의 상대적인 시간으로 계산하여 타이머 만료 시간이 가장 적은 타이머의 값만을 줄여 줌으로써 전체 타이머의 타이머 만료 시간 값이 줄어드는 효과를 가져와 시간결정성을 보장한다. 하지만 타이머 생성 시 기존의 타이머 리스트에서의 삽입 위치를 찾는 시간이 증가하게 되어 시간결정성이 저해되는 요인이 될 수 있다.

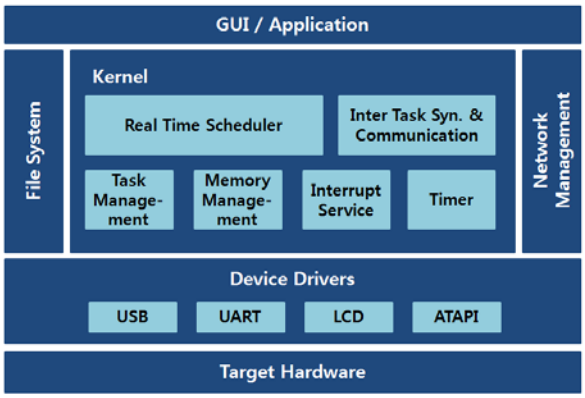
본 논문에서는 델타 프로세싱의 사용으로 인해 타이머 삽입 시 시간적인 오버헤드가 발생하는 것을 줄이는 방법이 될 수 있는 타이머 관리 기법인 타이밍 휠의 설계 및 구현에 대해 기술한다[1].

Ⅱ. 관련 연구

1. iRTOS

실시간 운영체제 *i*RTOS는 멀티태스킹 환경을 지원하며, 태스크 생성에 필요한 메모리가 존재하면 생성 가능한 태스크의 수에는 제한이 없다. 그리고 모든 태스크에 0부터 255까지 256단계의 우선순위를 부여하여 우선순위 기반의 선점형 스케줄러를 제공한다. 또한 동일한 우선순위에 대한 스케줄링 정책은 라운드-로빈 스케줄링을 사용한다. 임계 영역(Critical Region)이나 공유자원 사용을 위한 태스크간 동기화를 위해 세마포, 이벤트 플래그를 제공하며 메시지 큐, 메시지 포트, 메

시지 메일박스 등의 태스크간 통신 기법을 지원한다. 또한 힙 메모리를 동적으로 할당하기 위해서 힙 스토리지 매니저와 메모리 풀을 지원한다[2].



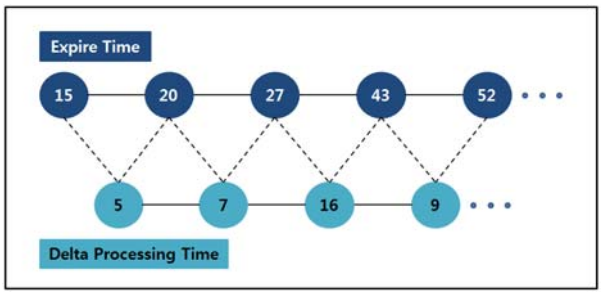
▶▶ 그림 1. /RTOS의 전체구성

2. 타이머 인터럽트

타이머 인터럽트는 주기적으로 발생하는 하드웨어 클럭에 의해 생기는 인터럽트이다. 타이머 인터럽트가 발생하면 운영체제는 하던 작업을 잠시 멈추고 인터럽트 서비스 루틴(Interrupt Service Routine : ISR)으로 점프하게 된다. 인터럽트 서비스 루틴은 발생한 인터럽트의 종류를 확인한 후 각 인터럽트에 맞는 실제 서비스 루틴을 수행하게 되는데 타이머 인터럽트는 해당 시간에 수행해야 할 소프트 타이머를 실행한다. 인터럽트 서비스 루틴은 종료시 레디 상태에 있는 태스크들 중에 가장 높은 우선순위의 태스크를 실행하게 하는데 이는 시간결정성에 도움이 되지만 인터럽트 서비스에 있어 그 시간이 길어진다면 시간결정성을 저해하는 요인이 되기도 한다[3].

3. 델타 프로세싱(Delta Processing)

델타 프로세싱은 활성화된 타이머 리스트의 타이머 만료 시간을 각각의 차이만큼으로 계산하는 방법이다. 만약 델타 프로세싱을 하지 않는다면, 타이머 인터럽트 서비스루틴 실행 시 활성화된 타이머 리스트의 모든 타이머의 타이머 만료 시간을 각각 하나씩 감소해야 한다. 활성화된 타이머가 몇 개 없을 때에는 문제가 없을지라도 활성화된 타이머가 많아질수록 타이머 만료 시간을 감소하는데 점점 더 많은 시간이 소요되게 된다. 결국 인터럽트 서비스 루틴에서 많은 시간을 소비하므로 인터럽트 응답시간이 길어질 수 있다. 그러나 델타 프로세싱을 하면 두 번째 타이머의 타이머 만료 시간은 첫 번째 타이머의 차이만큼을 유지하고 나머지 타이머도 같은 식으로 앞의 타이머와 타이머 만료 시간의 차이만큼을 유지하고 있다. 결국 타이머 리스트의 헤더의 타이머 만료 시간을 하나 감소시킴으로써 리스트 전체의 타이머 만료 시간을 하나 감소시킨 효과를 볼 수 있다[2].



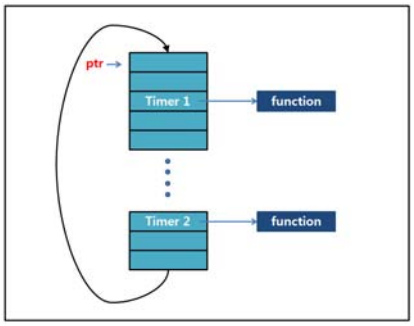
▶▶ 그림 2. 델타 프로세싱

Ⅲ. 타이밍 휠의 설계 및 구현

1. 타이밍 휠

타이밍 휠은 타이머에서 사용하는 최소 시간을 의미하는 엔트리로 이루어진 고정된 크기의 배열 구조체다. 타이밍 휠을 사용하면 정렬된 소프트 타이머의 타이머 갱신 성능을 발휘하면서도 효율적인 타이머의 설치와 취소가 가능하다. 소프트 타이머는 하드웨어 타이머를 이용해서 하나의 주기 타이머를 설치한다. 하나의 하드웨어에 기반을 둔 타이머가 나머지 모든 타이머의 작동을 가능하게 한다. 따라서 클럭 틱의 크기가 전체 소프트 타이머의 정밀도를 결정한다. 예를 들어 1클럭 틱이 50ms라면 타이밍 휠에서 하나의 슬롯은 50ms를 의미하며 이것은 시스템에서 사용할 수 있는 가장 작은 타이머 타임아웃 값이다. 타이머가 생성되면 배열의 해당 인덱스에 타이머를 설치한 후 하드웨어 타이머 한 주기마다 인덱스를 가리키는 포인터가 옮겨가며 해당 인덱

스에 타이머가 존재하면 그것을 서비스해주는 기법이다. 이는 델타 프로세싱의 단점인 타이머의 삽입 시에 오버헤드를 감소시켜 준다[4].



▶▶ 그림 3. 타이밍 휠

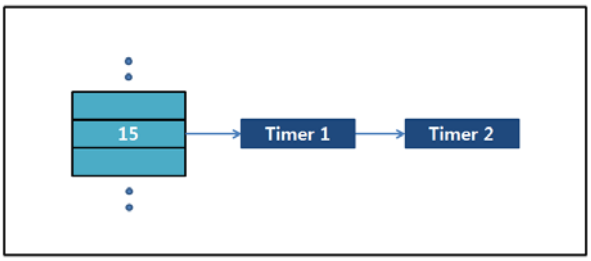
2. 타이밍 휠의 구현

2.1 타이밍 휠의 크기

타이밍 휠은 한정된 수의 타이머 엔트리를 가지고 있다. 엔트리의 개수가 많으면 넓은 범위의 타이머의 주기를 제공할 수 있지만 그에 따른 메모리 낭비를 감수해야 한다. 반대로 엔트리의 개수가 적으면 상대적으로 메모리의 낭비는 줄어들지만 좁은 범위의 타이머 주기를 제공할 수 밖에 없다. 타이밍 휠의 크기는 시스템 소프트웨어 설계자에 의해 최적으로 설계 되어져야 한다. 본 논문에서는 타이밍 휠 배열의 크기를 100으로 설정하였고 배열의 인덱스가 99가 되면 다시 0번째 인덱스로 돌아가 배열을 검사하도록 하였다.

2.2 동일한 인덱스 내 타이머의 처리

예를 들어, 현재 포인터가 10번째 인덱스를 가리키고 있고, 15번째 인덱스에 타이머 1이 존재하는 상황일 때, 타이머 2가 5클럭 틱 후에 활성화되게 생성되었다면 해당 인덱스에는 두 개의 타이머가 존재하게 된다. 두 타이머를 연결리스트를 이용하여 동일한 인덱스에 존재하게 처리하였으며 타이머의 처리 순서는 FIFO(First In First Out)이다.



▶▶ 그림 4. 동일 인덱스의 처리

2.3 타이머 생성

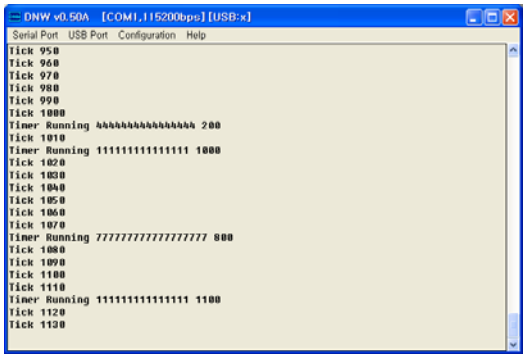
iRTOS의 타이머 생성 API인 MK_CreateTimer에서 제공하는 타이머 컨트롤 블록, 타이머 이름, 타이머 실행 함수를 그대로 적용할 수 있으며 특히 타이머의 초기 시간과 반복 시간을 적용할 수 있게 하여 기존 API를 그대로 사용할 수 있도록 하였다.

표 1. MK_CreateTimer()의 프로토 타입

```
MK_Status_t
MK_CreateTimer (
MK_TIMER *pTimer,
MK_S8_t *pName,
MK_TIMER_FUNC_T Function,
MK_S32_t Arg1,
MK_Void_t *Arg2,
MK_U32_t InitialTime,
MK_U32_t RepeateTime,
MK_Bool_t Enable)
```

Ⅳ. 테스트 환경 및 결과

본 논문에서 구현한 타이밍 휠은 MBA2440 Evaluation Board에 iRTOS를 탑재하여 구현하였으며, 컴파일러는 Metrowerks사의 Code Warrior를 사용하였다.



▶▶ 그림 5. 타이밍 휠 사용 시 결과 화면

위 그림 5는 클럭 틱이 약 1000번 가량 발생했을 때의 각 타이머의 실행 횟수를 나타낸 결과 화면이다. 타이머 1은 1000번, 타이머 4는 약 200번, 타이머 7은 약 800번이 실행 됐고, 그 중에 타이머 1의 주기가 가장 짧은 것을 확인할 수 있다.

Ⅴ. 결론 및 향후 연구 과제

본 논문에서는 임베디드 시스템에서 주로 사용되는 실시간 운영체제 iRTOS의 타이머 관리 기법 델타 프로세싱의 단점인 타이머의 삽입 시 높은 오버헤드를 줄이기 위해 타이밍 휠 기법을 설계하고 구현한 내용을 기술하였다.

임베디드 시스템은 시스템의 처리 속도도 중요하지만 일반 범용 시스템과는 달리 자원이 제한적이기 때문에 적은 자원을 효율적으로 사용하고 관리하는 기법이 필요하다. 타이밍 휠 기법은 기존의 델타 프로세싱보다 타이머 삽입 시에 속도적인 면에서 오버헤드를 줄여주지만 제한적인 타이머 시간 설정과 배열의 사용으로 인한 메모리 낭비가 단점이다. 향후 연구 과제로 델타 프로세싱을 사용했을 때와 타이밍 휠을 사용했을 때의 속도와 메모리 효율 측면에서 비교 시험하는 것에 대한 연구가 진행되어야 한다.

■ 참 고 문 헌 ■

[1] 양희권, 박윤미, 류현수, 이철훈 “실시간 운영체제의 태스크 사용시간 측정 방법 구현”, 한국정보과학회 학술발표논문집, 제31권, 제1호(A), pp. 166 ~ 168, 2004. 4

[2] iRTOS User's Guide

[3] 양희권, 조희남, 성영락, 이철훈 “Implementation of Interrupt Service Process for Efficient Interrupt Handling”, 한국정보과학회, Vol. 30. No. 2(1), pp.319~321, 2003.10

[4] Li, Qing and Yao, Caroline, Real-time concepts for embedded systems, pp. 176 ~ 181, 에이콘출판사, 의왕, 2004.

[5] MBA2440 board manual KOR v2.1, Aiji System, 2006