실시간 운영체제의 태스크 사용시간 측정 방법 구현*

양회권[0], 박윤미, 류현수, 이철훈
충남대학교 컴퓨터공학과
{hkyang[0], ympark, hsryu, chlee}@ce.cnu.ac.kr

The Implementation of Method to Estimate Usage of Task for Real Time Operating System

Hui-Kwon Yang[0], Yoon-Mi Park, Hyun-Soo Ryu, Cheol-Hoon Lee
Dept. of Computer Engineering, Chungnam National Univ.

요 약

실시간 운영체제는 시간 결정성이 가장 중요시 되는 운영체제이다. 다시 말해, 예측가능성을 제공함으로써 시스템의 성능을 예측할 수 있게 하여 최적화된 시스템의 설계 및 구현을 가능하게 한다. 그러나 실시간 운영체제상의 응용프로그램은 여러가지 요인으로 인해 그 수행시간을 예측하기가 쉽지 않다. 본 논문은 시스템에 탑재된 타이머를 이용하여 태스크의 사용시간을 측정할 수 있는 방법을 설계 및 구현하였다.

1. 서 론

주로 임베디드 시스템에 탑재되는 실시간 운영체제는 커널의 수행시간을 일정하게 유지할 수 있기 때문에 시간 결정성(Time Determinism)을 제공한다. 이 시간 결정성은 시스템의 성능을 예측하게 할 뿐 아니라, 시스템의 전반적인 안정성에도 좋은 영향을 준다. 그러나 실시간 운영체제가 탑재된 시스템에서 수행되는 응용프로그램은 그 수행시간이 예측 가능할 뿐, 정확한 사용시간을 알기 어렵다. 시스템에서 동작하는 각 태스크의 사용시간(Usage of Task)을 보다 정확히 측정할 수 있다면, 시스템의 성능 측정 및 최적화에 많은 도움을 줄 수 있을 것이다. 본 논문은 실시간 운영체제인 *i*RTOS™에 시스템의 타이머를 이용하여 태스크의 사용시간을 측정하기위한 방법에 대한 설계 및 구현을 기술하였다[1].

본 논문의 구성은 2 장에서 기반연구로서 *i*RTOS™의 스케줄링에 대해 소개하고, 연구장비인 S3C2800 에 탑재된 Timer에 대해 기술한다. 3 장에서는 S3C2800 의 Timer를 이용하여 설계 구현한 태스크 사용시간 측정 방법, 4 장에서는 태스트 환경 및 결과를 보이고, 5 장에서는 결론 및 향후 연구 과제를 기술한다.

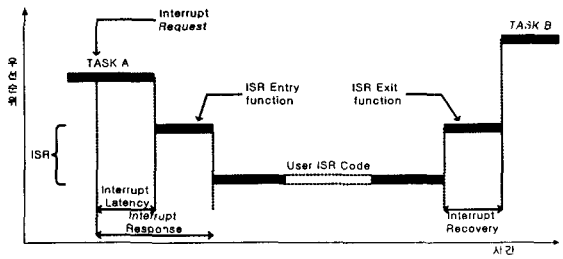
2. 관련 연구

2.1 *i*RTOS™

2.1.1 스케줄링

*i*RTOS™는 우선순위 기반 멀티 태스킹을 지원하는 실시간 운영체제로서, 256 단계의 우선순위를 제공한다. 서로 다른 우선순위의 태스크들 사이에는 선점형 스케줄링 정책을, 동일한 우선순위의 태스크들 사이에는 라운드-로빈(Round-Robin) 또는 FIFO(First in First Out) 스케줄링 정책을 적용한다. 선점형 스케줄링이란 프로세스의 우선순위에 기반하여 새로 생성되거나 시스템의 자원을 획득하여 실행 준비상태(Ready State)로 된 태스크가 가장 높은 우선순위이면 즉시, 실행 중이던 태스크의 수행을 멈추고 새로운 태스크에게 CPU 사용권을 부여하는 것으로서 응답성(Responsiveness)이 중요시되는 시스템에 적합하다. 대응 개념으로 비선점형(Non-Preemptive) 스케줄링이 있다[1][4][5][6].

2.1.2 인터럽트 처리 및 태스크의 동작



[그림 1] ISR 및 태스크의 수행

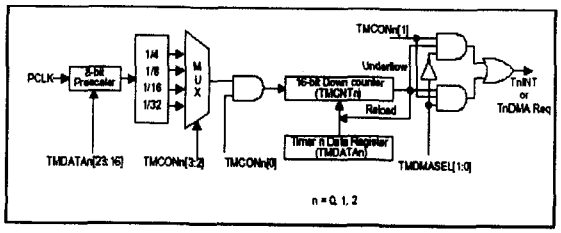
일반적으로 실시간 운영체제는 인터럽트가 발생하면 태스크의 수행을 멈추고 인터럽트 서비스 루틴(ISR)을 수행한 후 모든 인터럽트에 대한 처리가 끝나면 태스

* 본 논문은 산업자원부의 중기거점과제 연구비 지원에 의한 것임.

크의 수행을 재개한다. [그림 1]은 실시간 운영체제에서 인터럽트 서비스 루틴과 태스크의 수행을 도식화 한 것이다[4][5].

2.2. S3C2800 의 타이머

S3C2800 은 ARM9 RISC가 탑재된 보드로서 세개의 DMA기반 16 비트 타이머를 가지고 있다. [그림 2]는 S3C2800 에 탑재된 16 비트 타이머의 블록 다이어그램이다.



[그림 2] 16비트 타이머의 블록다이어그램

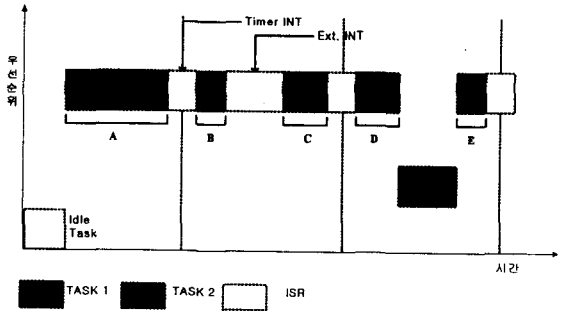
이와 같은 타이머는 메모리 주소로서 접근 가능한 레지스터들에 의해 제어된다. [표 1]은 0 번 타이머 제어에 사용되는 주요 레지스터를 나열한 것이다. 16 비트 타이머는 [그림 2]에서 볼 수 있듯이, PCLK 및 타이머 설정에 의해 생성된 타이머 클럭을 다운 카운트(Down Count)하여 TMDATA 레지스터에 지정된 값만큼 발생할 때 타이머 인터럽트를 발생시킨다. 따라서 타이머의 TMCNT 레지스터의 값을 이용하면 정확한 시간의 계산이 가능하다[2][3].

[표 1] 0번 타이머의 레지스터

Register	Address	기능
TMCONn	0x10130000	타이머 제어
TMDATAn	0x10130004	Prescaler 및 타이머데이터 설정
TMCNTn	0x10130008	타이머 카운터

3. 태스크 사용 시간 측정 방법의 구현

3.1 설계



[그림 3] 태스크의 수행시간

[그림 3]은 실시간 운영체제에서 태스크의 수행시간을 도식화 한 것이다. [그림 3]에서와 같이 태스크 1 의 수행중 주기적으로 타이머 인터럽트와 외부 인터럽트가 발생하고, 태스크 2 와 스케줄링이 일어나는 상황에서 태스크 1 의 실제 수행시간은 A, B, C, D, E를 모두 합한 시간이다. 그러나 일반적으로 실시간 운영체제에서는 타이머 인터럽트를 카운트하여 Tick단위로 태스크의 수행 시간을 결정하기 때문에 [그림 3]에서 세번째 타이머 인터럽트가 발생하기전까지 태스크 1 과 태스크 2 의 수행시간은 각각 2Tick과 0Tick이 된다.

본 연구팀은 타이머의 TMCNT 레지스터 값을 이용한 태스크의 사용시간 측정을 위하여 세가지 상황을 고려하였다. 첫째, 태스크의 시작이다. 태스크 시작시점에서의 TMCNT 값이 태스크가 시작된 시간값이 된다. 둘째, 태스크 수행 중 인터럽트의 발생이다. 실시간 운영체제에서 인터럽트는 중첩(Nesting)이 허용되기 때문에 태스크에서 인터럽트 서비스 루틴으로 수행이 이행된 시점에 태스크의 수행시간이 계산되어야 한다. 셋째, 모든 인터럽트의 처리가 끝나고 인터럽트의 중첩 단계(Interrupt Nesting Level)이 0 이 되어 태스크로 수행이 옮겨질 때 태스크가 시작된다. 넷째, 스케줄링을 통해 문맥교환(Context Switching)이 일어날 때, 이전 태스크는 현재 TMCNT 값을 기준으로 수행시간을 계산하고 새로운 태스크는 문맥교환이 끝나는 순간, 수행이 시작된다. 다섯째, 인터럽트가 비활성화(Interrupt Disable)되는 경우, TMCNT 값의 측정 시점이 모호함으로 이 구간을 2Tick 이하로 가정한다.

3.2 구현

[그림 4]는 태스크 사용 시간을 측정하기 위해 개발한 *i*RTOS™ 커널 소스에 추가로 정의한 사항들이다.

TIMEDATA는 타이머 카운터의 초기값이고 t_Used Ticks는 TMCNT 값에 의해 계산된 실제 수행시간을 Tick단위로 저장될 태스크 제어 블록(TCB)의 필드이다. StartClocks은 태스크의 시작 시간을 나타낸다.

```
#define TIMEDATA 65536
typedef struct mk_task_struct {
    ...
    float   t_UsedTicks;
    ...
} MK_TASK

ULONG   StartClocks;
```

[그림 4] 추가로 정의된 사항

[그림 5]는 실제 *i*RTOS™ 커널의 수행중 태스크의 수행시간을 계산하도록 추가된 코드이다. 스케줄러(MK_Schedule())에서는 문맥교환이 이루어지기전에 태스크의 수행시간을 계산하고 문맥교환 후엔, 새 태스크의 수행시간 계산을 위해 StartClocks를 TMCNT 값으로 초기화 한다. 또한 인터럽트 처리 전후의 수행시간 계산을 위해 MK_IntEnter()에서 인터럽트 처리하기 직전에 수행시간을 계산하고, MK_IntExit()에서 인터럽

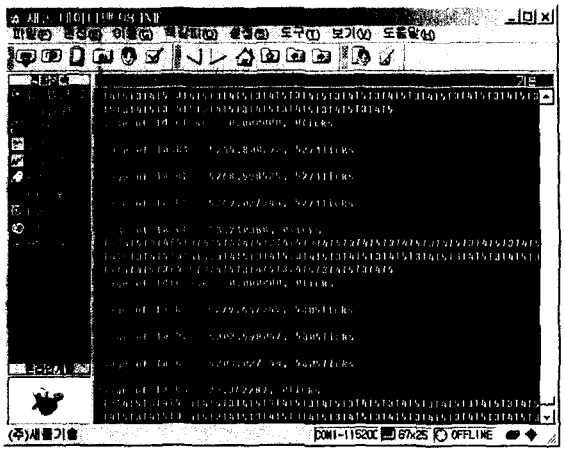
트 처리가 끝났을 때 StartClocks를 초기화 한다

```
MK_Schedule( ) {
    ...
  LastClocks = (ULONG)TMCNT0;
  if(StartClocks <= TMCNT0)
    TotalClocks = StartClocks + (TIMEDATA -
TMCNT0);
  else
    TotalClocks = StartClocks - TMCNT0;
  pTask->t_UsedTicks += TotalClocks / TIMEDATA;
  MK_ContextSwitch();

  StartClocks = TMCNT0;
    ...
}

MK_IntEnter() {

if(MK_CountNesting ==0) {
  if(StartClocks <= TMCNT0)
    TotalClocks = StartClocks + (TIMEDATA -
TMCNT0);
  else
    TotalClocks = StartClocks - TMCNT0;
  pTask->t_UsedTicks += TotalClocks / TIMEDATA;
}

MK_CountNesting++;
}

MK_IntExit() {
  if( (--MK_CountNesting) <=0 ) {
    StartClocks = (ULONG)TMCNT0;
    MK_Schedule();
  }
}
```

[그림 5] 태스크 수행시간 계산

4. 테스트 환경 및 결과

본 연구는 ARM-920T CPU가 탑재된 S3C2800 보드를 기반으로 SDT2.51 ARM용 통합개발환경을 사용하여 진행하였다. 커널은 본 연구팀에서 자체 개발한 *i*RTOS ™를 사용하였다. [그림 6]은 시스템에서 각 태스크의 이름을 출력하는 네개의 태스크를 수행한 결과이다. 각 태스크는 동일한 우선순위를 갖도록 하였고, 처음 세 태스크는 동일한 라운드-로빈 시간을 주어 일정한 주기로 동작하도록 하였고, 네번째 태스크는 시스템내 태스크들의 실제 수행시간과 타이머 인터럽트에 의한 수행시간을 Tick단위로 출력하도록 하였다. 그림에서 볼 수 있듯이 T1, T2, T3 의 수행주기와 동작이 같음에도 계산된 수행시간에 약간의 차이를 보임을 알 수 있다. 또한 T4 는 단위 수행시간이 짧기 때문에 실제 23Ticks 정도 수행했음에도 불구하고 0Tick으로 나타난다[2][3][6].



[그림 6] 테스트 결과

5. 결론 및 향후 과제

본 논문에서는 실시간 운영체제에서 태스크의 수행시간을 측정하기 위한 방법을 커널 수준에서 설계하고 구현하였다. 타이머 인터럽트에 의한 수행시간에 비해 정확한 측정을 위해 여러가지 상황을 고려하였고, 예제 프로그램의 수행을 통해 훨씬 향상된 수행시간 측정방법을 구현할 수 있었다.

향후, 본 연구의 성과를 토대로 다양한 응용 가능성을 모색하고 나아가 실시간 운영체제용 디버깅 툴 개발에 대한 연구가 진행되어야 한다.

6. 참고문헌

[1] http://www.inestech.com
[2] http://www.imcu.co.kr/
[3] S3C2800 32-BIT RISC Microprocessor User's Manual Revision 0
[4] Jean, J. Labrosse, "µC/OS The Real-Time Kernel", R&D Publications, 1995.
[5] C.M.Krishna, Kang G.Shin, "Real-Time Systems", McGraw-Hill, 1997
[6] *i*RTOS™ User's Guide