실시간 운영체제에서 인터럽트 서비스 루틴을 위한 Pending Lock/Unlock 설계 및 구현*

안희중[0], 성영락[†], 이철훈
충남대학교 컴퓨터공학과, [†] 국민대학교 전자정보통신공학부
{hjahn[0], chlee}@ce.cnu.ac.kr, [†] yeong@mail.kookmin.ac.kr

Design and Implementation of Pending Lock/Unlock for ISR in Real-Time Operating Systems

Hee-Joong Ahn[0], Yeong Rak Seong[†], and Cheol-Hoon Lee
Dept. of Computer Engineering, Chungnam National University
[†] School of Electrical Engineering, Kookmin Univ.

요 약

실시간 운영체제는 태스크 수행의 논리적인 정확성뿐만 아니라 시간적인 정확성도 만족하는 스케줄링을 하는 운영체제이다. 태스크가 공유자원을 사용할 때 가용한 자원이 없으면 그 자원을 얻기 위해 기다리는 Pending 상태가 된다. 이러한 상태가 일반적인 상황에서 발생하면 스케줄러에 의해서 적절하게 관리되지만 인터럽트 상태나 시그널 상태와 같은 특수한 경우에 발생하면 데이터의 충돌이나 심지어 실시간 운영체제가 붕괴되는 상황이 발생할 수 있다. 이를 해결하기 위해 상용 RTOS 에서는 개발자가 ISR 에서 이러한 상황이 발생하지 않도록 주의 깊게 사용할 것을 경고하며, 커널 내부적으로는 복잡한 처리과정을 거쳐 해결할 수 있다. 본 논문에서는 이러한 상황이 발생할 가능성이 있을 때 PENDING LOCK/UNLOCK 함수를 사용하여 개발자에게 개발상의 편의를 제공하고, 실시간 운영체제의 안정성 및 신뢰성을 보장할 수 있도록 설계 및 구현한 내용을 기술한다.

1. 서 론

실시간 운영체제는 예상치 못한 특정 이벤트가 발생하는 악조건 속에서도 태스크(Task) 수행이 데드라인을 초과하지 않도록 시간 결정성을 보장하는 스케줄링 기능을 제공하는 운영체제이다. 실시간 운영체제는 Unix™, Linux™, Windows™ 등의 범용 운영체제와 같이 멀티태스킹, ITC(Inter-Task Communication), 메모리 할당, 예외처리(Exception Handling)를 제공하지만 시간 결정성 보장을 위해 우선순위 기반의 선점형 스케줄링과 특수한 내부적인 수행 루틴을 포함하고 있다[1][2]. 멀티태스킹 환경을 제공하는 실시간 운영체제에서 태스크들간의 통신을 위해 세마포(Semaphore), 메시지 큐(Message Queue), 메시지 메일박스(Message Mailbox)를 위한 API(Application Program Interface) 함수를 제공하고 있다. 일반적인 상황에서 위와 같은 함수의 사용은 스케줄러에 의해서 적절하게 관리되지만 인터럽트 상태에서 이러한 함수를 잘못 사용하면 운영체제가 붕괴되는 상황이 발생할 수 있다. 그리고 인터럽트와 비슷한 시그널 상태에서도 이러한 함수의 적절치 못한 사용은 데이터의 충돌이나 운영체제의 오동작을 유발할 수 있다. 이러한 경우 상용 RTOS 에서는 개발자가 주의 깊게 사용할 것을 경고하고 커널 내부적으로 복잡한 처리를 함으로써 해결한다[3] [4].

본 논문은 ISR 에서 태스크를 PENDING 상태로 변경시킬수 있는 함수의 잘못된 사용으로 인해 시스템 붕괴가 일어날 소지가 있는 부분에서 PENDING LOCK/UNLOCK 함수를 사용하여 적절한 에러처리를 함으로써 개발자에게 개발의 편의성을 제공하고 실시간 운영체제에는 안정성과 신뢰성을 제공하도록 하였다. 본 논문의 주요구성은 2 장에서 실시간 운영체제의 전체적인 구성을 설명하고, 3 장 에서는 PENDING LOCK/UNLOCK 함수의 설계 및 구현 내용을 설명하고, 4 장에서는 구현된 함수를 적용한 실시간 운영체제의 테스트 환경 및 결과를 설명하고, 마지막으로 5 장에서는 결론 및 향후 연구 과제에 대해서 기술하였다.

2. 관 련 연 구

2.1 실시간 운영체제

실시간 운영체제에서 가장 중요한 부분은 태스크를 관리하고 CPU 를 할당하는 스케줄러이다. 실시간 운영체제의 스케줄러는 태스크의 중요도에 따라 우선순위를 할당하고 우선순위 기반의 선점형 스케줄링 방법으로 태스크의 논리적 정확성과 시간적 정확성을 보장한다. 실시간 운영체제는 태스크 관리와 태스크간 통신 및 동기화를 위한 세마포, 메시지 큐, 메시지

*이 논문은 산업자원부 중기거점과제 연구비 지원에 의한 것임

메일박스와 타이머, 시그널 등을 제공한다.

2.1.1 태스크 스케줄링 정책

태스크가 생성되면 자신의 로컬 스택과 0 부터 255 까지 256 단계의 우선순위를 할당받고 별도의 자료구조로 관리된다. 실행준비 상태의 태스크 중에서 우선순위가 가장 높은 태스크가 가장 먼저 실행하도록 스케줄링하여 시간적 정확성(Determinism)을 제공하며 같은 우선순위의 태스크에 대해서는 FIFO 또는 Round-Robin 방식으로 스케줄링한다[5].

2.1.2 태스크간 통신 및 동기화

□ 세마포

한 자원에 대한 상호배제(Mutual Exclusion)와 안전

한 공유자원 관리와 태스크간 동기화에 사용한다.

□ 메시지 큐

특정 태스크나 ISR 에서 다른 태스크로 여러 개의 메시지를 전달할 때 사용한다.

□ 메시지 메일박스

메시지 큐의 특수한 경우로 하나의 메시지를 빠르게 전달할 때 사용한다.

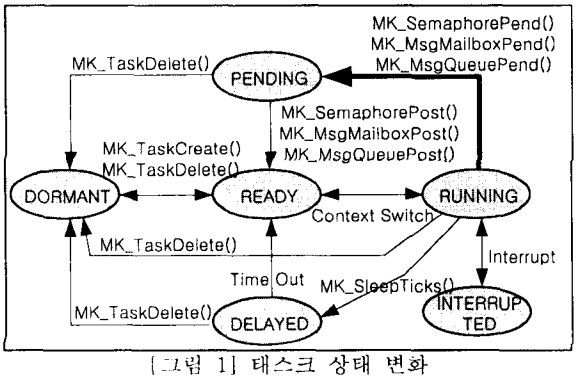
2.1.3. 시그널 처리

특정 태스크에 비동기적인 신호를 보내 특정 이벤트를 처리할 때 사용된다. 현재 실행하는 태스크에 비동기적인 신호를 보내는 인터럽트와는 달리 시그널은 실행 하고 있는 태스크 또는 실행하고 있지 않는 태스크에 비동기적인 신호를 보낼 수 있다. 시그널을 받은 태스크는 시그널에 관련된 Context 를 스택에 저장하고 현재 데이터를 유지한 상태에서 실행준비상태가 되어 스케줄링된다.

3. PENDING LOCK/UNLOCK 함수의 설계 및 구현

3.1 태스크 상태 변화

각 태스크는 [그림 1]의 상태 중 하나의 상태를 유지하며 시스템 콜에 의해서 다른 상태로 변화된다.



[그림 1] 태스크 상태 변화

DORMANT 상태는 태스크가 생성되어 메모리에 상주되었지만 스케줄링 대상에서 제외된 상태이고, READY 상태는 실행 준비가 되어 스케줄링 되기를 기다리는 상태이며, RUNNING 상태는 CPU 를 점유하여 실행하고 있는 상태이고, PENDING 상태는 어떠한 이벤트나 공유자원 반납을 기다리는 상태이고, DELAYED 상태는 실행중인 태스크가 스스로 CPU 점유권을 놓고 일정시간 기다리는 상태이고, INTERRUPTED 상태는 인터럽트를 처리중인 상태이다.

실시간 운영체제는 공유자원의 효율적인 관리를 위해서 세마포를 사용한다. RUNNING 상태의 태스크가 세마포를 이용하여 공유자원을 요구했을 경우, 가용한 공유자원이 있으면 그 태스크에 공유자원을 할당하고 가용한 공유자원이 없으면 PENDING 상태로 태스크의 상태가 변화된다. 실시간 운영체제는 태스크간 통신을 위해서 메시지 큐와 메시지 메일박스를 사용한다. RUNNING 상태의 태스크가 메시지 큐나 메시지 메일박스에 메시지를 요청했을 경우, 가용한 메시지가 존재하면 그 메시지를 전달하면 되지만 가용한 메시지가 존재하지 않으면 그 태스크는 메시지를 받기 위해서 PENDING 상태가 된다. 그리고 RUNNING 상태의 태스크가 메시지를 메시지 큐나 메시지 메일박스에 저장하려 할 경우 가용한 메시지 저장 공간이 존재하지 않을 경우에도 PENDING 상태가 된다.

3.2 ISR 에서 PENDING 함수 사용의 문제점

PENDING 함수는 태스크의 상태를 RUNNING 에서 PENDING 으로 변화시키는 함수를 의미한다. 본 논문에서 구현한 실시간 운영체제의 PENDING 함수는 공유자원이나 메시지 큐와 메시지 메일박스에서 메시지를 요구하는 MK_SemaphorePend(), MK_MsgQueue Pend(), MK_MsgMailboxPend()가 있으며, 메시지를 메시지 큐나 메시지 메일박스에 저장하는 MK_MsgQueuePost(), MK_MsgMailboxPost() 함수를 사용할 때 버퍼에 메시지가 가득 차 있으면 태스크는 PENDING 상태가 된다. 그 밖에 태스크를 생성하는 MK_TaskCreate()와 메모리 할당에 관련된 MK_Get Memory(), MK_Free Memory()가 있다.

실시간 운영체제에서 인터럽트는 외부의 이벤트에 대한 정보를 시스템에 전달하는 중요한 의미를 갖는다. 인터럽트의 빠른 응답시간을 위해서 태스크는 자신의 Context 외에 특별한 Context 를 필요로 하게 되며 이 때문에 인터럽트 서비스 루틴을 실행 중에는 스케줄링이 일어나 문맥교환(Context Switch)이 일어날 수 없도록 해야 한다. 그러나 인터럽트 서비스 루틴에서 PENDING 함수를 사용하여 태스크의 상태가 CPU 점유를 해제하고 PENDING 으로 변경되면 CPU 를 선점하여 실행하는 태스크도 존재하지 않고 그 후로도 스케줄링이 일어나지 않아 CPU 를 선점하는 태스크도 존재하지 않는 현상이 발생하여 시스템이 붕괴된다. 시그널 상태에서도 PENDING 함수를 사용하면 인터럽트 처리시와 같은 문제가 발생한다.

일반적으로 인터럽트나 시그널 처리 중에는 태스크의 상태를 PENDING 상태로 변화시키는 함수를 사용할 수 없으며, 상용 RTOS 는 매뉴얼을 통해서 이를 명시하고 개발자가 책임지고 적절하게 사용

하도록 경고하고 있다. 그리고 커널 내부적으로 위와 같은 상황이 발생되는 것을 감지하고 에러를 처리할 수 있는 복잡한 절차를 거쳐야 한다.

3.4 PENDING LOCK/UNLOCK 함수의 설계 및 구현

```
#define  MK_PLOCK    1
#define  MK_PUNLOCK  0
int MK_PENDING_LOCK(void);
int MK_PENDING_UNLOCK(void);
```

본 논문에서는 위와 같은 문제를 해결하기 위해서 인터럽트 서비스 루틴이나 시그널 처리 등에서 태스크의 상태가 PENDING 상태로 변경될 때 문제를 발생시킬 가능성이 있는 부분을 MK_PENDING_LOCK() 함수와 MK_PENDING_UNLOCK() 함수를 쌍으로 사용하여 태스크의 상태가 PENDING 으로 변경될 수 없도록 구현하였다. 이를 위해서 태스크의 Context 에는 t_PendingLock 필드가 추가적으로 필요하다.

```
typedef struct mk_task_struct {
    UINT    t_TopOfStack;          /* Top Pointer of
Stack */
    UINT    t_BottomOfStack;    /* Bottom Pointer of
Stack */
    int        t_Status;                /* Status of
Task */
    int        t_Priority               /* Priority
of Task */
   MK_TASK_FUNC_T t_Function;/* Function for Task */
     long      t_TimeSlice;          /*  Initialized
TimeSlice */
   long      t_DelayedDeltaTicks;
   UINT  t_PendingLock;      /* Pending Lock/Unlock */
   UINT    t_Signal;
   UINT    t_SigMask;
   void      (*t_pSigFunction)(UINT);
   struct    mk_task_struct *t_pTaskNext;
   struct    mk_task_struct *t_pTaskPrev;
   struct    mk_task_struct *t_pTaskReadyNext;
   struct    mk_task_struct *t_pTaskReadyPrev;
   struct    mk_task_struct *t_pDelayedNext;
   struct    mk_task_struct *t_pDelayedPrev;
   struct    mk_task_struct *t_pPendingNext;
   struct    mk_task_struct *t_pPendingPrev;
   char      t_pName[MK_NAME_MAX];    /* Task's Name */
   . . .
} MK_TASK;
```

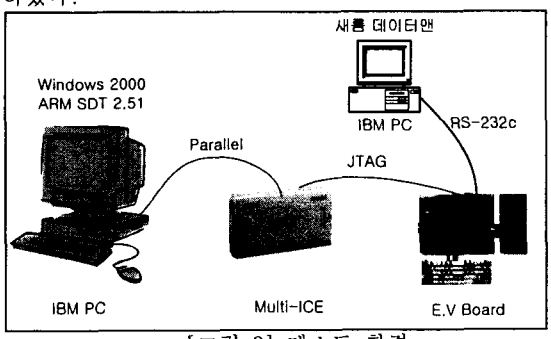
[그림 2] Task Control Block

MK_PENDING_LOCK() 함수는 t_PendingLock 필드를 태스크가 Pending 되지 않도록 MK_PLOCK 으로 변경하고 MK_PENDING_UNLOCK 함수는 t_PendingLock 필드를 태스크가 Pending 가능하도록 MK_PUNLOCK 으로 변경한다. 태스크가 Pending 하여 문제가 생길 수 있는 구간에 대해서 MK_PENDING_LOCK()/UNLOCK() 함수를 쌍으로 사용한다.

태스크가 인터럽트 처리나 시그널 처리에서 세마포나 메시지 큐를 사용하여 공유자원이나 메시지를 요구할 때 가용한 공유자원이 있거나 가용한 메시지가 존재하면 t_PendingLock 필드에 상관없이 공유자원이나 메시지를 할당하고, 가용한 공유자원이 없거나 가용한 메시지가 존재하지 않으면 태스크를 PENDING 상태로 변경시켜야 된다. 그러나 t_PendingLock 필드에 MK_PLOCK 으로 세팅되어 있으므로 태스크의 상태를 변경하지 않고 에러처리를 하여 태스크 상태 변경으로 인한 문제점을 해결한다.

4. 테스트 환경 및 결과

본 논문에서 기술하고 있는 실시간 운영체제는 32-bit CPU 를 대상으로 설계되었으며, 컴파일러는 ARM SDT 2.51 을 사용하였다. 컴파일한 실행 이미지는 약 23K 정도 이며, 이를 삼성에서 제작한 ARM920T 기반의 S3C2400 Evaluation Board 에 다운로드하여 테스트하였다.



[그림 3] 테스트 환경

5. 결론 및 향후 연구과제

본 논문은 실시간 운영체제에서 태스크가 PENDING 상태로 변경될 수 있는 함수를 인터럽트나 시그널에서 부적절하게 사용함으로써 발생할 수 있는 실시간 운영체제의 데이터 충돌 및 시스템 붕괴와 같은 최악의 상황을 방지 할 수 있도록 PENDING LOCK/UNLOCK 함수를 설계 및 구현하여 개발자의 편의성과 실시간 운영체제의 안정성 및 신뢰성을 높여 주었다.

공유자원이나 메시지 등의 자원을 볼 수 있는 Resource Viewer 와 같은 디버깅 환경과 더 나아가 통합 개발환경 구현에 대한 부분은 계속 연구되어야 할 것이다.

6. 참 고 문 헌

[1] Embedded System, RTOS, http://www.inestech.com
[2] Jean J. Labrosse, uC/OS The Real-Time Kernel, R&D Publications,1995.
[3] WindRiver, VxWorks Progammer's Guide, 1997.
[4] Accelerated Technology, *Nucleus PLUS Internals Manual*, 1996.
[5] IEEE Std 1003.1b, *Portable Operating System*, 1993.