실시간 운영체제에서 Recursive Semaphore 설계 및 구현

이원용°, 김용희, 손필창, 이철훈
충남대학교 컴퓨터공학과
{nissikr°, yonghee, pcson, clee}@cnu.ac.kr

Design and Implementation of Recursive Semaphore for Real-Time Operating Systems

Won-Yong Lee°, Yong-Hee Kim, Pil-Chang Son, Cheol-Hoon Lee
Dept. of Computer Engineering, Chungnam National University

요 약

실시간 시스템의 개발 및 운영에 사용되는 실시간 운영체제는 여러 개의 태스크가 동시에 작업할 수 있는 멀티태스킹 환경과 각 태스크에 우선순위를 부여하여 가장 높은 우선순위의 태스크가 CPU 를 선점하는 스케줄링 방법, 그리고 태스크간 동기화 및 통신을 위한 메커니즘을 제공하고 있다. 또한 여러 태스크들에 의해 사용되는 공유자원을 관리하기 위해 세마포어(Semaphore)를 사용하여 태스크간 동기화를 제공한다. 그러나 하나의 태스크가 세마포어를 이용하여 공유자원에 대해 여러 번 접근 할 경우, 데드락(Deadlock)을 일으킬 소지가 많다. 본 논문에서는 실시간 운영체제인 *i*RTOS™ 에서 데드락을 방지하기 위한 재귀적 세마포어(Recursive Semaphore)를 설계 및 구현하였다.

1. 서 론

실시간 시스템은 시스템 내부 또는 외부에서 발생하는 이벤트에 대하여 이벤트 발생시간과 그 이벤트 처리가 끝날 때까지의 지연 시간이 미리 제시된 시간(Deadline)을 넘지 않는 시스템을 의미한다.

실시간 시스템에서 여러 개의 태스크가 하나의 자원을 접근하려 할 때 자원의 값이 바뀔 경우 다른 태스크에 영향을 끼치게 된다. 이러한 자원을 공유자원이라 한다. 실시간 시스템에서는 공유자원을 안전하게 사용하기 위해서 세마포어를 사용하고 있다.

본 논문은 실시간 운영체제 중에서 안정성 및 성능에서 인정받아 상용화 된 *i*RTOS™ 를 연구대상으로 하였고, 태스크가 공유자원을 획득하고 또 다시 공유자원을 호출 할 경우 생기는 문제점을 예방하기 위해 재귀적 세마포어를 설계하고 이를 구현하였다.[1][2]

본 논문의 구성은 2 장에서 관련연구로 *i*RTOS™의 구성과 제공하는 커널 서비스에 대하여 설명하고, 3 장에서는 재귀적 세마포어의 설계 및 구현한 내용을 기술한다. 4 장에서는 테스트 환경 및 결과를 설명하고, 5 장에서는 결론 및 향후 연구과제를 기술한다.

2. 관련 연구

2.1. *i*RTOS™

실시간 운영체제는 코드실행을 시간에 따라 정확히 관리하고, 시스템 자원을 관리하며, 응용 프로그램 개발을 위한 일관된 기반을 제공하는 운영체제이다. 실시간 운영체제인 *i*RTOS™ 는 기본적으로 멀티태스킹 환경 및 우선순위를 기반으로 하는 선점형 스케줄러를 제공하고 태스크간 동기화 및 통신을 위해서 세마포어, 메시지 큐, 메시지 메일박스 등을 제공한다.

2.1.1 태스크 스케줄링 정책

*i*RTOS™ 에서 생성되는 모든 태스크들은 가장 높은 우선순위인 0 부터 가장 낮은 우선순위인 255 까지의 256 개의 우선순위 중 하나를 부여 받고 별도의 테이블로 관리된다. 실행 준비된 태스크들 중에서 가장 높은 우선순위를 갖는 태스크가 CPU 점유권을 가져가는 선점형 스케줄링 정책을 사용하고, 시간적 정확성(Determinism)을 제공하며 같은 우선순위를 갖는 태스크에 대해서는 FIFO 또는 Round-Robin 방식으로 스케줄링 한다.[3][6]

2.1.2 태스크간 통신 및 동기화

① 세마포어(Semaphore)

세마포어는 수행중인 여러 태스크가 동기화 또는 상호배제를 목적으로 획득하거나 반환할 수 있는 커널 서비스이다.

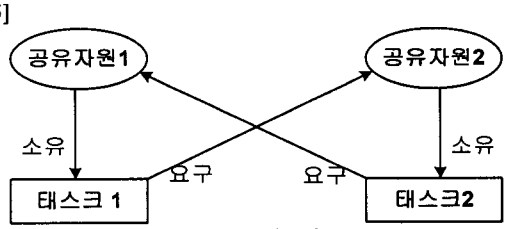
② 메시지 큐(Message Queue)

태스크나 ISR(Interrupt Service Routine)이 다른 태스크에 여러 개의 메시지를 전달할 때 사용한다.

③ 메시지 메일박스(Message Mailbox)
메시지 큐의 특수한 경우로 하나의 메시지를 빠르게 전달할 때 사용한다.

2.1.3 데드락(Deadlock)

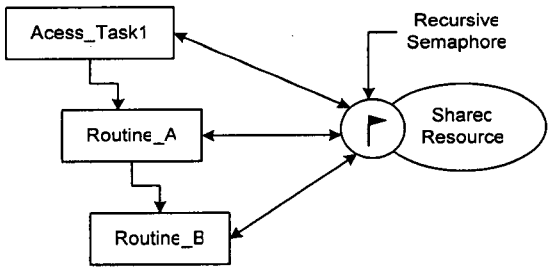
둘 이상의 태스크가 각각 리소스를 점유한 상태에서 상대 태스크에 의해 점유된 리소스를 원한다면 블록 상태로 대기하게 되는데, 이런 현상을 데드락이라 한다.[5]



[그림 2-3] 데드락

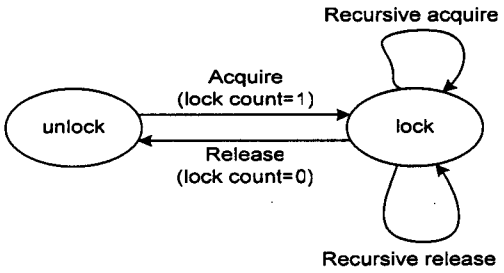
3. 재귀적 세마포어 설계 및 구현

재귀적 세마포어는 공유자원에 독점적으로 접근하고자 하는 태스크가 그 공유자원에 여러 차례 접근 할 때 유용하다. 재귀적 세마포어를 사용하면 데드락을 일으키지 않고 리소스를 여러 차례 획득 할 수 있다.



[그림 3-1] 공유자원에 접근

[그림 3-1]에서 Access_Task1 이 Routine A 를 호출하고 Routine A 는 Routine B 를 호출하는데 3 개의 루틴이 모두 공유자원을 사용한다고 가정하자. 이런 상황에서 일반적인 세마포어를 사용하면 Access _Task1 은 데드락이 발생해서 더 이상 진행하지 못하고 블로킹 될 것이다. 태스크에서 루틴을 호출하면 그 루틴은 그 태스크에 일부분으로 봐야 하므로 Routine A 와 Routine B 는 Access_Task1 의 일부로 동작하는 것이다. 따라서 Routine A 와 Routine B 가 세마포어를 획득 하려고 시도하는 것은 Access_Task1 이 세마포어를 다시 획득하려는 것과 같다. 이런 경우, Access_Task1 은 이미 자신이 획득한 세마포어를 다시 요구함으로써 데드락이 발생하게 된다. 이런 상황을 방지하기 위해 재귀적 세마포어를 사용하는 것이다. Access_Task1 이 재귀적 세마포어를 사용하면 태스크 자신 또는 태스크가 호출한 루틴이 같은 공유자원을 호출해도 데드락이 발생하지 않는다. 따라서 태스크와 Routine A, B 가 안전하게 루틴을 끝낼 수 있다.



[그림 3-2] 재귀적 세마포어의 상태 다이어그램

[그림 3-2]와 같이 태스크가 재귀적 세마포어를 획득할 때 iRTOS™ 커널은 그 태스크를 재귀적 세마포어의 소유자로 등록하고 lock count 를 사용해서 현재 공유자원을 몇 번 호출 했는지를 나타낸다. unlock 상태를 풀어주기 위해서는 lock count 만큼 풀어줘야 한다. 이런 경우에 lock count 는 현재 상태가 lock 인지 unlock 인지를 나타내며, 또한 재귀적으로 lock 한 횟수까지 나타낸다.[2]

3.1 재귀적 세마포어 생성 및 삭제

```
STATUS MK_CreateRecursiveSemaphore
       (pRecursiveSemaphore, pName, Options)
STATUS MK_DeleteRecursiveSemaphore
       (pRecursiveSemaphore)
```

[그림 3-3] 재귀적 세마포어 생성/삭제

[그림 3-3]에서 MK_CreateRecursiveSemaphore() 함수는 Recursive Semaphore 를 생성하는 함수이다. pRecursive Semaphore 는 세마포어 제어블록이고 pName 은 세마포어의 이름, Options 는 세마포어에서 자원할당을 기다리는 태스크에 대한 스케줄링 정책으로서 태스크의 우선순위에 따라 스케줄링 하는 MK_Recursive Semaphore _PRIO 와 먼저 자원을 기다린 태스크에 스케줄링 하는 MK_RecursiveSemaphore _FIFO 가 있다.

3.1 재귀적 세마포어 획득 및 반환

```
STATUS MK_ObtainRecursiveSemaphore
       (int RecursiveSemaphoreID, int timeout)
STATUS MK_ReleaseRecursiveSemaphore
       (int RecursiveSemaphore)
```

[그림 3-4] 재귀적 세마포어 획득/반환

공유자원을 소유한 상태에서 다시 획득하려 함으로써 데드락이 발생할 가능성이 있는 경우 MK_Obtain RecursiveSemaphore()를 사용하여 공유자원을 획득할 수 있다. 태스크가 공유자원을 획득하면 여러 번 공유자원에 접근한다 할지라도 데드락을 피할 수 있게 된

다. 세마포어를 획득할 때마다 획득한 횟수가 몇 번인지 알 수 있게 카운트 값을 증가 시킨다. 그리고 timeout 은 공유자원을 곧 바로 획득하지 못하는 경우에 실시간 운영체제의 시간 결정성을 보장하기 위해서 제공하는 시간 제약사항이다.

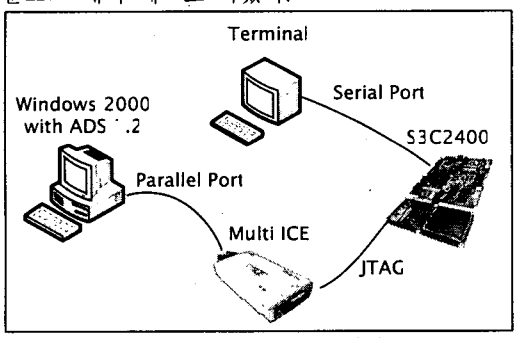
```
typedef struct mk_recursive_semaphore_struct
{
    MK_U32_t    s_Magic;
    MK_INT_t    s_Lock_Count;
    MK_U8_t     s_OwnerName;
    struct mk_pending_list_struct       s_PendingList;
    struct mk_recursive_semaphore_struct  *s_Prev;
    struct mk_recursive_semaphore_struct  *s_Next;
    MK_S8_t                *s_pName;
} MK_RecursiveSemaphore;
```

[그림 3-5] 재귀적 세마포어 제어블록

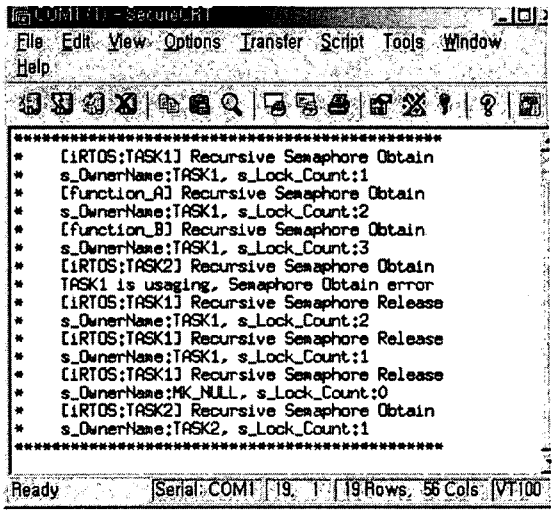
재귀적 세마포어 제어블록인 [그림 3-5]에서 s_Magic 은 재귀적 세마포어의 식별자이고, s_Lock_Count 는 현재 카운트 값을 나타내며, s_OwnerName 은 공유자원을 사용하는 현재 태스크 이름을 나타낸다. 즉 s_OwnerName 이 MK_Null 이면 태스크가 재귀적 세마포어를 얻어서 사용중이므로 다른 태스크들은 PENDDING 상태에서 대기하다가 MK_Null 이 된 후에 재귀적 세마포어를 획득할 수 있다. s_PendingList 는 Recursive Semaphore 에서 자원을 기다리는 태스크의 리스트이다. 그리고 각각 s_Prev 와 s_Next 는 이전과 다음 Recursive Semaphore 를 가리키고 s_Name 은 Recursive Semaphore 의 이름이다.

4. 테스트 환경 및 결과

본 논문에서 기술하고 있는 실시간 운영체제인 *i*RTOS™ 는 32-bit Processor 를 대상으로 설계하였으며, 컴파일러는 ARM ADS 1.2 을 사용하였으며, 삼성에서 제작한 ARM920T 기반의 S3C2400 Evaluation Board 에 다운로드 해서 테스트 하였다.



[그림 4-1] 테스트 환경



[그림 4-2] 테스트 결과

[그림 4-2]는 Task1 이 재귀적 세마포어를 획득한 후에 내부 함수에 의해 추가적으로 2 번 더 획득을 했다. 그런 다음에 TASK2 가 재귀적 세마포어를 획득하려다가 s_OwnerName 이 MK_Null 이 아니므로 실패를 한다. 재귀적 세마포어를 TASK1 이 반환 후에 TASK2 가 재귀적 세마포를 얻는다.

5. 결론 및 향후 과제

본 논문은 실시간 운영체제에서 세마포어를 사용함으로써 발생하는 데드락을 방지해서 시스템 붕괴와 같은 최악의 상황을 피해갈 수 있도록 재귀적 세마포어를 설계 및 구현하여 실시간 운영체제의 안전성 및 신뢰성을 높여 주었다.

향후 연구과제로 실시간 시스템에서 안정성을 더욱 강화하기 위해 재귀적 세마포어를 사용하는 시점을 정확히 알 수 있도록 연구가 진행되어야 할 것이다.

참고 문헌

[1] Jean, J. Labrosse, *"μ C/OS-II The Real-Time Kernel"*, R&D Books, 1999

[2] Qing Li, Caroline Yao, *"RTOS 를 이용한 실시간 임베디드 시스템 디자인"*, CMP Books, 2003

[3] WindRiver, *VxWorks Programmer's Guide*, 1997.

[4] IEEE std 1003.1b, *Portable Operating System*, 1993.

[5] David Kalinsky, "*Mutexes Prevent Priority Inversions*", Embedded Systems Programming, 1998.

[6] 안희중, 박희상, 이철훈, "*Design and Implementation of Mutual Exclusion Semaphores Using the Priority Ceiling Protocol*", 정보처리학회 추계학술발표논문집, vol.9, no.2, pp.555-558, Nov.2001