제 18 회 한국정보처리학회 추계학술발표대회 논문집 제 9 권 제 2 호 (2002.11)

Priority Ceiling Protocol 을 이용한 Mutual Exclusion Semaphore 설계 및 구현

안희중, 박희상, 이철훈
충남대학교 컴퓨터공학과
e-mail : hjahn@ce.cnu.ac.kr

Design and Implementation of Mutual Exclusion Semaphores Using The Priority Ceiling Protocol

Hee-Joong Ahn, Hee-Sang Park, Cheol-Hoon Lee
Dept. of Computer Engineering, Chung-Nam 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, 태스크간 동기화 및 통신을 위한 메커니즘을 제공하고 있다. 그리고 여러 태스크들에 의해 사용되는 공유자원을 관리하기 위해 세마포어를 사용하여 태스크간에 동기화를 제공한다. 하지만 세마포어만으로 공유자원을 관리하게 되면 더 높은 우선순위의 태스크가 실행 준비 되어 있음에도 불구하고 상대적으로 낮은 우선순위의 태스크가 CPU 를 선점하는 우선순위 역전이 발생하여 실시간 운영체제의 핵심인 시간 결정성을 만족하지 못해 시스템에 심각한 문제를 발생 시킬 수 있다. 본 논문에서는 실시간 운영체제인 iRTOS™에서 우선순위 역전을 예방하기 위한 방법중 하나인 Priority Ceiling Protocol 을 이용한 Mutual Exclusion Semaphore 를 설계하고 구현한 내용을 기술한다.

1. 서론

실시간 시스템은 기존의 시스템과 달리 시스템의 수행결과가 기능적으로 정확해야 할 뿐만 아니라 결과가 도출되는 시간도 제약조건을 만족하는 시스템을 의미한다. 이러한 실시간 시스템의 개발이나 운영에 사용되는 운영체제가 실시간 운영체제이다.

실시간 운영체제는 독립적인 여러 개의 태스크가 동시에 수행할 수 있는 멀티태스킹(Multi-Tasking) 환경을 지원하며 각 태스크는 자신의 우선순위(Priority)를 가지고 있으며 실행 준비된 태스크들 중에서 우선순위가 가장 높은 태스크가 CPU 를 선점하는 스케줄링을하며 태스크간 동기화 및 통신을 위해서 세마포어, 메시지 큐, 메시지 메일박스를 제공하고 있다. 여러 개의 태스크가 하나의 자원을 접근하려 할 때 데이터의 불일치가 발생할 수 있다. 이러한 자원을 공유자원이라 하며 이를 효율적으로 관리하기 위해서 실시간 운영체제는 세마포어를 사용하고 있다. 그러나 세마포어만을 사용하여 공유자원을 관리하게 되면 실행 준비된 태스크들 중에서 우선순위가 높은 태스크가 있음에도 불구하고 상대적으로 우선순위가 낮은 태스카가 CPU 를 선점하는 우선순위 역전(Priority Inversion) 문제가 발생할 수 있다. 이를 예방하는 방법으로 Mutual Exclusion Semaphore 를 사용하며 구현하는 방법에 따라서 높은 태스크의 우선순위를 계승하는 Priority Inheritance Mutual Exclusion Semaphore 와 우선순위를 주어진 우선순위까지 높혀주는 Priority Ceiling Mutual Exclusion Semaphore 로 구분한다. 본 연구팀은 실시간 운영체제 중에서 안정성 및 성능에서 인정받은 iRTOS™를 연구대상으로 하였고 태스크가 공유자원을 획득하면 주어진 우선순위로 증가하는 Priority Ceiling Mutual Exclusion Semaphore 를 설계 하고 이를 구현하여 우선순위 역전을 예방하도록 하였다.

본 논문의 구성은 2 장에서 iRTOS™의 구성과 제공

하는 커널 서비스에대한 기반연구를 설명하고 3 장에서는 Priority Ceiling Mutual Exclusion Semaphore 를 설계하고 구현한 내용을 설명한다. 그리고 4 장에서는 테스트 환경 및 결과를 설명하고 5 장에서는 결론 및 향후 연구과제를 기술한다.

2. 기반연구

2.1. iRTOS™

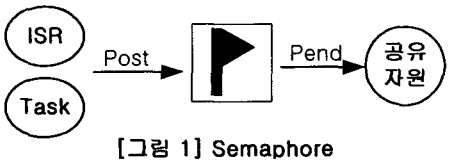
실시간 운영체제는 중요한 태스크에 높은 우선순위를 부여하여 CPU 를 선점할 수 있도록 함으로써 중요한 태스크가 정해진 시간내에 정확히 끝낼 수 있도록 시간 결정성(Time Determinism)을 보장하는 운영체제이다. 실시간 운영체제인 iRTOS™는 기본적으로 멀티태스킹 환경 및 우선순위를 기반으로 하는 선점형 스케줄러를 제공하고 태스크간 동기화 및 통신을 위해서 세마포어, 메시지 큐, 메시지 메일박스를 제공한다[1].

2.1.1. 태스크 스케줄링 정책

실시간 운영체제에서 생성되는 모든 태스크들은 가장높은 우선순위인 0 부터 가장낮은 우선순위인 63 까지의 64 단계의 우선순위중 하나를 부여받고 별도의 테이블로 관리되어 정해진 시간에 실행 준비된 태스크들 중에서 가장 높은 우선순위를 갖는 태스크를 찾을 수 있다. 태스크가 다른 우선순위를 갖을 경우에는 실행 준비된 태스크들 중 가장 높은 우선순위를 갖는 태스크가 CPU 를 선점하는 우선순위 기반의 선점형 스케줄링 정책을 사용하여 실시간 운영체제의 시간 결정성을 보장해 주며 같은 우선순위를 갖는 경우에는 주어인 타임 슬라이스(Time Slice)동안 차례로 수행되는 라운드 로빈(Round-Robin) 정책을 사용한다.

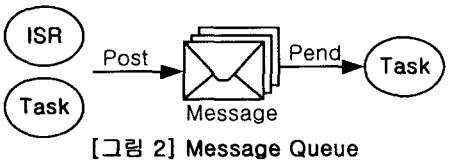
2.1.2. Inter-Task Synchronization and Communication

① 세마포어 (Semaphore)
세마포어는 여러 태스크가 한 자원에 대하여 상호 배타적인 접근을 허용하여 공유자원을 효과적으로 관리하고 태스크들간 동기화에 사용된다.



[그림 1] Semaphore

② 메시지 큐 (Message Queue)
태스크나 Interrupt Service Routine 이 다른 태스크에 여러 개의 메시지를 전달할 때 사용한다.

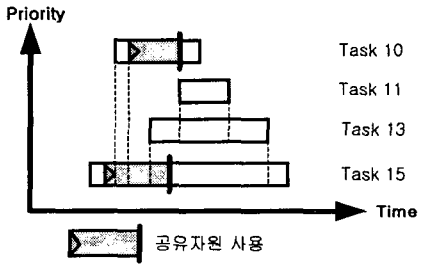


[그림 2] Message Queue

③ 메시지 메일박스(Message Mailbox)
메시지 큐의 특수한 경우로 태스크로 전달 가능한 메시지가 한 개이며 빠르게 메시지를 전달할 때 사용된다.

2.2. 우선순위 역전(Priority Inversion)

우선순위 기반의 선점형 스케줄링을 하는 실시간 운영체제에서 실행 준비가 된 태스크들 중에서 우선순위가 가장 높은 태스크가 CPU 를 선점하여 실행하지만 세마포어만을 사용하여 공유자원을 관리하면 실행 준비된 태스크중에 상대적으로 우선순위가 높은 태스크가 있음에도 불구하고 상대적으로 낮은 우선순위를 갖는 태스크가 CPU 를 점유하는 경우가 발생하는데 이를 우선순위 역전이라 한다. 우선순위 역전이 발생하면 우선순위가 높은 태스크가 상대적으로 낮은 우선순위의 태스크 때문에 중요한 작업을 수행하지 못하고 대기하게 되며 이 태스크의 대기 시간을 예측할 수 없기 때문에 실시간 운영체제의 핵심인 시간 결정성을 깨뜨리게 되어 시스템에 심각한 문제를 야기시킬수 있다.



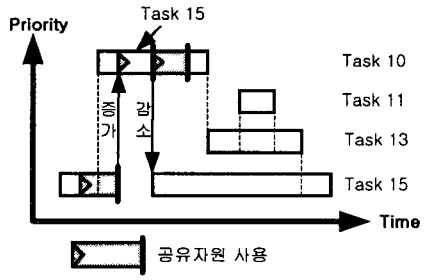
[그림 3] Priority Inversion

[그림 3]에서 태스크의 우선순위는 Task 10 이 가장 높고 Task 11, Task 13, Task 15 순으로 우선순위를 갖고 있다. 우선순위가 가장 낮은 Task 15 가 생성되어 공유자원을 소유하고 작업을 수행하던 중에 Task 10 이 생성되어 CPU 를 선점하고 작업을 수행중 공유자원을 요구하면 Task 15 가 공유자원을 소유하고 있기 때문에 공유자원을 얻지 못하고 Task 15 가 공유자원을 해제하기를 기다리며 대기 상태로 이동하여 Task 15 가 다시 CPU 를 선점하여 작업을 수행한다. 이때 공유자원과 상관없는 Task 13 이 생성되면 현재 수행중인 태스크 15 가 상대적으로 우선순위가 낮기 때문에 Task 13 이 CPU 를 선점하고 자신의 작업을 수행한다. 따라서 Task 10 은 Task 13 이 공유자원을 사용한 작업이 끝나고 공유자원을 해제하기를 기다리고 있었으나 Task 13 이 CPU 를 선점하여 두 태스크의 작업이 끝나기를 기다리게 된다. 그러나 공유자원을 소유하지 못해 대기하고 있으므로 Task 15 가 공유자원을 해제하기를 기다리는 것은 어쩔 수 없으나 상대적으로 우선순위가 낮은 Task 13 이 공유자원과 상관없이 CPU 를 선점하게 되는 문제가 발생하였다. Task 10 보다 낮은 중간 단계의 우선순위를 갖는 태스크가 연쇄적으로 CPU 를 선점하면 Task 10 이 언제 수행될지 예측하기가 불가능하므로 시간 결정성을 깨뜨리는 결과가 된다. 이와 같은 문제가 우선순위 역전이며 이를 예방하기 위해

인터럽트 방지나 태스크의 선점을 방지하는 방법을 제시할 수 있지만 이는 시스템에 또다른 문제점을 야기시키는 원인이 된다. 따라서 효과적으로 우선순위 역전 문제를 예방하기 위해서는 현재 공유자원을 소유하고 작업을 수행중인 태스크의 우선순위를 변경하는 방법을 사용한다. 우선순위를 변경하는 방법에 따라서, 상대적으로 높은 우선순위를 상속받는 Priority Inheritance Mutexes Semaphore 와 주어진 우선순위로 바로 변경하는 Priority Ceiling Mutexes Semaphore 가 있다.

2.3. Priority Inheritance Mutexes Semaphore

공유자원을 소유한 태스크가 공유자원을 이용한 작업을 수행하는 중에 상대적으로 우선순위가 높은 태스크가 공유자원을 요구하면 공유자원을 소유하고 있는 태스크의 우선순위를 공유자원을 요구하는 태스크의 우선순위를 상속받아 동일하게 증가시키며 이를 이용한 세마포어를 Priority Inheritance Mutexes Semaphore 라 한다. 우선순위가 공유자원을 요구한 태스크와 동일하기 때문에 이보다 낮은 우선순위를 갖는 태스크에 CPU 를 선점당하지 않고 공유자원을 이용한 작업을 수행할 수 있게 된다. 이와 같은 방법으로 우선순위 역전을 예방하며 높은 우선순위의 태스크가 공유자원을 요구했을 때 이 태스크의 시간 결정성을 보장한다. 공유자원을 다 사용하고 해제하게 되면 우선순위가 원래대로 되돌아간다.

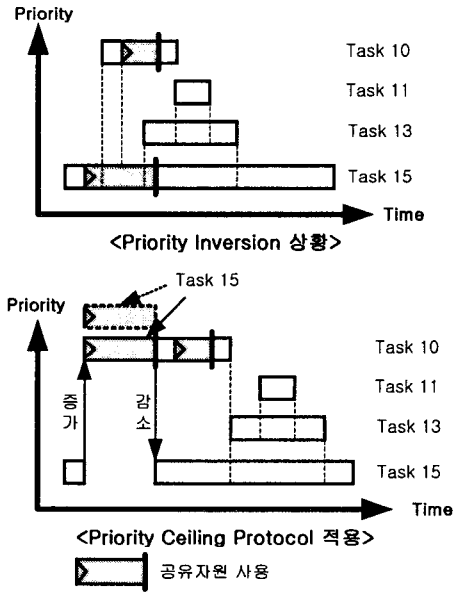


[그림 4] Priority Inheritance Mutexes

상대적으로 높은 우선순위의 태스크가 공유자원을 요구하면 자동으로 현재 공유자원을 소유하고 있는 태스크의 우선순위가 증가되므로 프로그래머가 API 함수를 사용하기 용이하지만 구현이 복잡하고 여러 개의 태스크가 여러 개의 공유자원을 요구하게 되면 교착상태(Deadlock)에 빠질 수 있다[2][3].

3. Priority Ceiling Mutexes Semaphore 설계 및 구현

Priority Ceiling Mutexes Semaphore 는 태스크가 공유자원을 획득하면 그 공유자원을 요구하는 태스크와는 상관없이 미리 주어진 우선순위로 정적으로 증가하여 공유자원을 이용한 작업을 수행하도록 공유자원을 관리하는 방법이다. 프로그래머는 하나의 공유자원에 대하여 Mutexes Semaphore 를 생성할 때 공유자원을 획득한 태스크의 우선순위를 그 공유자원을 요구하는 모든 태스크들의 우선순위와 동일하거나 높게 정해야 하며 다른 태스크들에 영향을 주지 않는 범위 내에서 신중하게 결정해야 한다. 그리고 Priority Inheritance Mutexes Semaphore 와 마찬가지로 공유자원을 해제하면 태스크의 우선순위는 이전의 우선순위로 되돌아가게 된다.

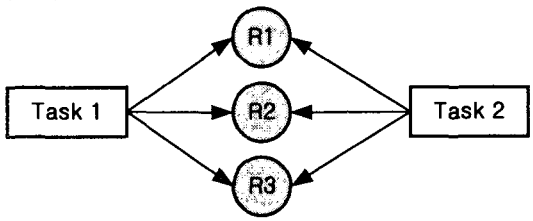


[그림 5] Priority Ceiling Mutexes

[그림 4]에서 Task 15 가 생성된 후 공유자원을 획득하면 다른 태스크인 Task 10, Task 11, Task 13 와는 상관없이 미리 Mutexes Semaphore 생성시 프로그래머에 이해 주어진 우선순위로 바로 증가하여 공유자원을 이용한 작업을 수행한다. 그 후 Task 10, Task 13, Task 11 이 생성되지만 Task 15 의 우선순위가 Task 10 보다 크거나 동일하기 때문에 계속 CPU 를 선점하고 다른 태스크의 방해 없이 작업을 수행한다. 그리고 Task 15 가 공유자원을 이용한 작업을 마치고 공유자원을 해제하면 Task 15 의 우선순위는 이전의 우선순위로 변경되고 실행준비 된 태스크들 중 우선순위가 가장 높은 Task 10 이 CPU 를 선점하여 작업을 수행함으로 우선순위 역전이 발생하지 않게 되어 Task 10 의 시간 결정성을 보장해 준다.

Priority Inheritance Mutexes Semaphore 는 [그림 6]에서와 같이 여러 개의 태스크가 여러 개의 공유자원을 요구할 때 교착상태에 빠질수 있다. 높은 우선순위의 Task1 과 낮은 우선순위의 Task2 는 공유자원 R1, R2, R3 를 획득해야만 어떠한 작업을 수행할 수 있을 때 Task2 가 먼저 R1 만을 얻은 후 Task1 이 생성되어 CPU 를 선점하고 R1, R2 를 얻고 R3 을 얻으려 할 때 이미 Task2 가 R3 을 소유하고 있으므로 R3 가 해제되기를 기다린다. Task2 가 깨어나 R1, R2 를 얻으려 하지만 Task1 이 소유하고 있어 R1, R2 가 해제되길 기다리게 된다. 결과적으로 Task1, Task2 는 서로의 자원을 기다리는 교착상태에 빠지게 된다. 그러나 Priority Ceiling Mutexes Semaphore 는 위의 예에서 우선순위가 낮은 Task2 가 R3 를 얻으면 Task1 의 우선순위보다 크

거나 동일한 우선순위로 증가하므로 Task1 이 생성되어도 Task1 에 CPU 를 선점당하지 않고 R1, R2 얻어 필요한 작업을 수행할 수 있어 교착상태가 발생하지 않는다.



[그림 6] Deadlock 발생 조건

3.1. Mutexes Semaphore Creation/Deletion

mutexID MK_CreateMutexSemaphore
(MK_MSCB *mutex, int priority, int option)
STATUS MK_DeleteMutexSemaphore
(MK_MSCB *mutex)

MK_CreateMutexSemaphore() 함수는 Mutexes 세마포어를 생성하는 함수이다. priority 는 태스크가 공유자원을 소유했을 때 증가하는 우선순위이며 프로그래머가 신중히 결정해야 하며 option 은 태스크가 공유자원을 반납할 때 공유자원을 기다리는 태스크들 중 어떠한 태스크를 스케줄링할 것인지 결정하는 것으로 태스크의 우선순위에 따라 스케줄링하는 MK_MUTEX_PRIO 와 선입선출의 MK_MUTEX_FIFO 가 있다[4][5][6].

3.2. Mutexes Semaphore Obtain/Release

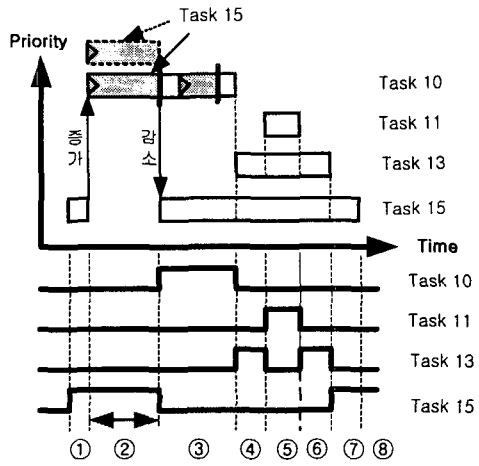
STATUS MK_ObtainMutexSemaphore
(int mutexID, int timeout)
STATUS MK_ReleaseMutexSemaphore (int mutexID)

공유자원을 소유하려 할 때 우선순위 역전이 발생할 가능성이 있는경우 MK_ObtainMutexSemaphore() 함수를 사용하여 공유자원을 획득할 수 있다. 공유자원을 획득하게 되면 태스크의 우선순위가 Mutexes 세마포어 생성시 주어진 우선순위로 변경된다. timeout 은 공유자원을 바로 획득하지 못할경우에 실시간 운영체제의 시간 결정성을 보장하기 위해서 제공되는 시간 제약사항이다. timeout 이 양수값이면 그때까지 공유자원을 기다린다는 의미이고, 0 이면 그 즉시 얻지 못하면 바로 에러처리 함을 의미하며, -1 이면 공유자원을 얻을 때 까지 무한정 기다린다는 의미이다.

MK_ReleaseMutexSemaphore() 함수를 이용하여 공유자원을 다 사용한후 해제하면 태스크의 우선순위는 원래의 우선순위로 되돌아간다[4][5][6].

4. 테스트 환경 및 결과

본 논문의 구현은 삼성에서 제작한 CPAD3 용 통합 개발환경인 CalmSHINE16 으로 컴파일 하였으며 실시간 운영체제인 iRTOS™ 커널의 이미지는 약 21Kbyte 이다. 컴파일한 이미지를 CPAD3 용 Emulation Board 로 다운로드하여 실행 하였다.



[그림 7] CPU 점유 그래프

[그림 7]에서 High Edge 는 CPU 를 점유한 상태를 의미하며 예상 결과를 나타내었다. ②구간에서 공유자원을 획득한 Task 15 의 우선순위가 증가하여 Task10 에 선점당하지 않고 CPU 를 점유하여 공유자원 해제시까지 작업을 수행한다. Task 15 가 공유자원을 해제하면 원래의 우선순위로 돌아가고 ③구간에서 그 다음 우선순위가 높은 Task 10 이 CPU 를 선점하고 작업을 수행하는 결과를 보여주고 있다[7].

5. 결론 및 향후 연구과제

본 논문에서는 멀티태스킹 환경 및 우선순위 기반의 선점형 스케줄링을 하는 iRTOS™에서 공유자원을 세마포어만으로 관리할 때 발생하는 우선순위 역전을 예방하기위한 하나의 방법으로 Priority Ceiling Mutual Exclusion Semaphore 를 설계하고 구현하여 실시간 운영체제의 시간 결정성을 깨뜨려 시스템에 심각한 영향을 줄 수 있는 우선순위 역전을 예방하게 되었다.

POSIX(Portable Operating System Interface for UNIX)에서는 두가지의 Mutexes 세마포어를 표준으로 정하고 있어 Priority Inheritance Mutexes Semaphore 의 교착상태를 예방하기 위한 방안을 강구하여 사용자의 상황에 맞도록 두개의 Mutexes 세마포어를 사용할 수 있도록 iRTOS™에서 POSIX 를 지원하는 부분은 계속 연구되어야 할 것이다.

참고문헌

[1] http://www.inestech.com
[2] David Kalinsky, "Mutexes Prevent Priority Inversions", 1998
[3] Orv Balcom, "Simple Task Scheduler Prevents Priority Inversion", 1995
[4] WindRiver, "VxWorks Programmer's Guide 5.3.1"
[5] Nucleus PLUS Reference Manual
[6] Jean j. Labrosse, "μ C/OS-II and Mutual Exclusion Semaphores", 2000
[7] IEEE Std 1003.1b, "Portable Operating System", 1993