우선순위 역전을 해결하기 위한 iRTOS 에서의 확장 MuTexS 설계 및 구현

강희성[0], 손필창, 정충희[†], 이철훈
충남대학교 컴퓨터공학과, [†]한국원자력안전기술원
{hskang[0], pcson, clee}@cnu.ac.kr, [†]ch.jeong@kins.re.kr

Design and Implementation of Advanced MuTexS For Prevent Priority Inversion in iRTOS

Hui-Sung Kang[0], Pil-Chang Son, Choong-Heui Jeong[†], Cheol-Hoon Lee
Dept. of Computer Engineering, Chungnam National University*,
[†]Korea Institute of Nuclear Safety

요 약

실시간 시스템에서는 두 개 이상의 태스크가 공유자원을 사용한다. 이러한 자원에 의해서 높은 우선순위 태스크가 낮은 우선순위 태스크에 의해서 CPU 를 점유 당하는 우선순위 역전현상(Priority Inversion)이 발생한다. 우선순위 역전 문제는 실시간 시스템의 스케줄 가능성과 예측성에 심각한 결함을 야기할 수 있다. 이를 해결하기 위해 본 논문에서는 비교적 크기가 작으면서도 실시간 운영체제의 핵심적 특징을 잘 갖추고 있는 iRTOS™ 커널을 사용하였고 우선순위 역전을 해결하기 위해서 Priority Inheritance Protocol 을 사용하여 확장된 MuTexS 를 구현하였다.

1. 서 론

실시간 운영체제는 각종 태스크들이 올바른 수행 결과를 데드라인에 맞춰 처리할 수 있는 결정성(Determinism)을 가져야 하는데 이러한 요소를 파괴하는 것이 우선순위 역전 현상(Priority Inversion)이다.

우선순위 역전의 해결방안으로는 우선순위 역전의 원인이 되는 태스크의 우선순위를 현재 실행 준비 상태에 있는 최상위 태스크의 수준으로 우선순위를 일시적으로 올려주어 실행하게 하는 우선순위 상속 프로토콜(Priority Inheritance Protocol)과 우선순위 역전 발생 가능 영역을 수행하는 태스크들의 우선순위 한계를 지정하여 해당 영역이 수행될 때 임의의 우선순위로 실행되게 하는 우선순위 상한 프로토콜(Priority Ceiling Protocol)이 있다. 본 논문에서는 우선순위 역전을 능동적으로 해결하기 위해 세마포(Semaphore)에 우선순위 상속 프로토콜을 적용하였다.

본 논문의 구성은 2 장에서 우선순위 역전문제(PIP: Priority Inversion Problem)의 개념을 기술하고, 3 장에서는 우선순위 상속 프로토콜 방식을 이용한 MuTexS(Mutual Exclusion Semaphore)의 설계 및 구현을 다룬다. 4 장에서는 테스트 환경 및 결과를, 마지막으로 5 장에서는 결론 및 향후 과제에 대해 기술한다.

2. 관련 연구

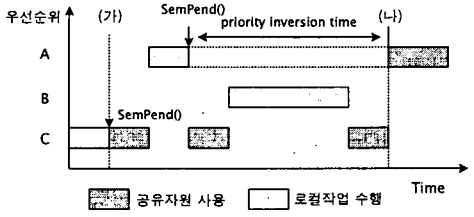
2.1 실시간 운영체제에서의 우선순위 역전문제

우선순위 기반의 선점형 스케줄링을 하는 실시간 운영체제는 실행 준비 상태의 태스크들 중 가장 높은 우선순위를 가진 태스크에게 CPU 를 할당한다. 그리고, 태스크들이 공유 자원을 접근하고자 할 때 상호 배타적 실행을 위해서 세마포를 제공한다. 임의의 우선순위를 가진 태스크가 실행 중에 낮은 우선순위를 가진 태스크에 의해 점유된 세마포를 필요로 할 경우, 실행되던 태스크는 우선순위가 높음에도 불구하고 세마포를 가진 태스크에게 CPU 의 점유를 빼앗기게 되며 이를 우선순위 역전이라고 한다. 우선 순위 역전이 발생하게 되면 기존의 높은 우선순위를 가진 태스크가 세마포를 반납할 때까지 실행이 지연된다. 더욱, 이들 두 태스크 사이의 우선순위를 가진 태스크에 의해 높은 우선순위를 가진 태스크의 실행 지연시간은 더 길어지게 되고, 시스템에 치명적인 오류가 발생한다.

[그림 2-1]에서 우선순위가 낮은 태스크 C 가 이미 세마포를 점유하고 있기 때문에 보다 높은 우선순위인 태스크 A 는 세마포가 반납될 때까지 CPU 를 얻지 못하게 된다. 이러한 경우를 '우선순위 역전' 이라 한다. 심각한 경우, 우선순위가 낮은 태스크 C 가 세마포를 점유하고 있는 (가)와 (나)의 시기에 A 와 C 의 중간 우선순위를 가진 태스크 B 가 실행 준비상태가 된다면 C 조차도 B 에게 CPU 를 빼앗기게 되고, 결국엔 가장

* 본 논문은 원자력안전기술원의 실시간 운영체제 평가기술에 관한 연구과제의 수행결과임.

높은 우선순위인 A 의 대기시간이 길어지게 된다. 이는 곧 시간결정성을 위반하고 시스템에 심각한 오류를 낳게 된다.



[그림 2-1] 우선순위 역전

2.2. Priority Inheritance Mutex

실행중인 태스크가 보다 낮은 우선순위의 태스크가 가지고 있는 Mutex 를 할당 받고자 할 때, 현재 실행중인 태스크를 대기 상태로 만들고 자신의 우선순위를 상속하여 Mutex 를 가지고 있는 태스크가 임계구역을 안정적으로 수행할 수 있도록 하는 방식이다. 우선순위를 상속 받은 태스크가 Mutex 를 해제하면, 원래의 우선순위로 복귀하고 대기하던 상위의 태스크가 Mutex 를 할당 받아 실행할 수 있도록 하는 방법이다.

2.3. Priority Ceiling Mutex

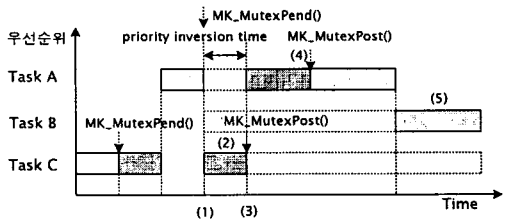
태스크가 Mutex 를 할당 받을 때, Mutex 생성시 미리 정해진 우선순위로 태스크의 우선순위를 증가시켜 Mutex 를 해제할 때까지 안정적으로 실행할 수 있도록 하는 방법이다. 프로그래머가 시스템 설계 시 Mutex 를 사용하게 될 태스크들의 우선순위 중 가장 높은 값을 기본값으로 설정하면 태스크들이 Mutex 를 사용하는 동안에는 우선순위가 기본값으로 변경되도록 함으로써 우선순위 역전을 미연에 방지하도록 한 방법이다.

2.4. *i*RTOS™ 실시간 운영체제

*i*RTOS™에서는 256 개의 우선순위를 가지며, 동적으로 추가, 삭제될 수 있다. 또한 많은 태스크들이 여러 개의 자원을 공유하면서 동시 접근의 안정성을 보장하기 위해서는 많은 MuTexS 를 생성하여 사용해야 한다. 이를 위해서 동일한 우선순위를 가진 태스크들에 라운드 로빈(Round Robin) 방식의 스케줄링을 지원한다.

3. 설계 및 구현

3.1 동작 시나리오



[그림 3-1] 우선순위 상속 동작 시나리오

[그림 3-1]는 우선순위 역전문제를 해결하기 위해 우선순위 상속개념을 도입하였으며 세부적인 동작 시나리오는 [표 3-1] 같다.

[표 3-1] 세부적인 동작 시나리오

(1) (2)	Task A 가 수행도중 Task C 에 의해 잠겨진 공유자원을 사용하기 위해 세마포를 요청하나, 커널은 Task C 에 의해 이미 사용하고 있음을 알린다. 이때 발생하게 되는 PIP 를 해결하기 위해 Task C 의 우선순위를 Task A 와 동일한 우선순위로 고쳐서 Task B 에 의해 선점당하지 않고 계속해서 공유자원에 접근하여 자신의 작업을 수행할 수 있게 한다.
(3)	Task C 는 공유자원의 사용을 마치고, 세마포를 커널에 반환하고 이때 앞서 높여 주었던 우선순위를 다시 원래대로 갱신한다.
(4)	Task A 가 쓰고자 하는 공유자원을 접근하기 위해 해제한 세마포를 얻어 자신의 작업을 수행한다.
(5)	Task C 가 원래의 우선순위로 돌아갔으므로, 스케줄링 되어 Task B 가 자신의 작업을 수행한다.

3.2 MuTexS 구조체 설계

[그림 3-2]에서와 같이 MuTexS 구현을 위해 다음의 MuTexS 구조체를 구현하였다.

```
typedef struct mk_mutexs_struct {
    int s_Count;
    <생 략>
    Struct MK_TASK  *s_Owner; // 추가
    uint    s_CurrentUse;  // 추가
    uint    s_OriginPrio;  // 추가
} MK_MUTEXS
```

[그림 3-2] 수정한 세마포 구조체

① s_Owner : MuTexS 를 소유하고 있는 TCB
② s_CurrentUse : MuTexS 자원의 사용여부
③ s_OriginPrio : Priority Inversion 이 발생할 경우, lower priority task 의 우선순위가 높아지는데, 이때 변경되기 이전의 우선순위를 저장하기 위해 사용, MuTexS 사용을 마친 태스크는 이를 반환할 때 s_OriginPrio 필드를 이용하여 본래의 우선순위로 복구한다.

3.4 MuTexS 할당 / 반납

MuTexS 를 구현한 API 들은 [표 3-2]와 같다.

[표 3-2] MuTexS 생성 / 할당 / 반납 API

MK_MutexCreate()
공유자원을 배타적으로 접근할 수 있도록 binary semaphore 를 생성시킨다
MK_MutexPend()

MuTexS가 가용하면, 이를 소유할 태스크의 우선순위를 저장하고 복귀한다. 한편 MuTexS가 사용중이면 이를 소유하고 있는 태스크와 현재 태스크의 우선순위를 비교한다. 만약, MuTexS를 요청한 태스크의 우선순위가 더 높다면, MuTexS를 소유한 태스크의 우선순위를 현재 요청한 태스크의 우선순위만큼 높여준다(Priority Inheritance Technique). 현재 태스크는 MuTexS를 무한정 기다리지 않도록 Timeout 값을 설정하고 재스케줄링을 통해 CPU를 점유할 다른 태스크를 설정한다.
MK_MutexPost()
MuTexS를 소유한 태스크만이 MK_MutexPost 함수를 이용하여 사용한 MuTexS를 해제할 수 있다. MuTexS를 반환하는 태스크의 우선순위와 MK_SEMAPHORE 구조체 내에 저장된 본래의 우선순위(s_OriginPrio)가 동일한가 검사한다. 만약 다르다면, 이전에 우선순위 상속이 발생한 것이므로, 원래의 우선순위로 복귀시킨다. 이때 MuTexS를 기다리는 태스크가 존재하면, MuTexS를 점유할 수 있도록 MuTexS를 새로운 태스크에 할당시키고, 만약 이를 기다리는 태스크가 없다면 MuTexS가 사용 가능함을 표시하고 재스케줄링을 통해 CPU를 점유할 태스크를 선정하여 수행을 계속한다.

4. 테스트 환경 및 결과

본 연구는 ARM-920T CPU가 탑재된 S3C2400 보드에서 ADS ARM 용 통합개발환경으로 진행되었다. 커널은 본 연구팀에서 자체 개발한 *j*RTOS™을 사용하여 세마포만을 새롭게 구현하였다.

테스트는 세 개의 태스크를 생성시키고(Task A/Task B/Task C), 두 개의 태스크(Task A/Task C)가 공유자원을 배타적으로 사용하기 위해 기존의 세마포를 사용하는 방법과 새롭게 구현한 MuTexS를 사용하는 방법을 비교하였다. 이는 전체 태스크의 프로세서 이용률을 [그림 4-1]와 같은 방식으로 측정하였으며 결과는 [그림 4-2]과 같다.

$$\text{CPU Useage} = 100 \times \left(1 - \frac{\text{MK\_IdleCtr}}{\text{MK\_MaxCtr}}\right)$$

[그림 4-1] 프로세서의 이용률 계산 방법

구분	기존 세마포	확장 MuTexS
프로세서 이용률	72%	89%

[그림 4-2] 프로세서 이용률 결과

결과에서 보는 것과 같이 일반적인 세마포를 사용하면 MuTexS를 사용했을 경우보다 우선순위 역전현상으로 발생한 지연시간으로 프로세서 이용률이 낮은 것으로 나타났다. 이는 Task A의 우선순위가 가장 높음에도 불구하고 Task B의 수행시간에 의존하는 경우가 발생하므로 Task A의 수행완료 바운드 타임 예측이 힘들다는 문제도 발생하게 된다. 이는 MuTexS를 사용함으로써 PIP를 완전히 해결할 수는 없지만, 세마포만을 사용한 경우보다 우선순위 역전시간이 훨씬 줄어들기 때문에 실시간 운영체제를 사용함으로써 얻는 태스크 수행 완료시간의 바운드 타임을 보장해줄 수 있는 측면에서 매우 중요한 기능을 제공하는 요소이다.

5. 결론 및 향후 과제

본 논문에서는 실시간 운영체제에서 태스크들간에 공유자원을 사용하는데 발생하는 우선순위 역전현상을 해결하기 위해 세마포에 우선순위 상속 프로토콜을 적용함으로써, 커널에 의해 우선순위 역전을 능동적으로 해결할 수 있도록 하였다. 본 연구에 이어 계속 연구되어져야 할 부분은 중첩(nested)된 MuTexS에 의해 발생하는 데드락(deadlock)을 해결하는 방법과 운영체제의 정확성 및 신뢰성의 향상을 위해 스케줄링과 관련하여 구현된 모듈이 얼마나 태스크 완료의 예측성을 제공하는지 갖가지 환경에서의 분석이 필요할 것이다. 아울러 다른 실시간 운영체제와의 비교 기준을 정의하고, 이 기준에 따른 성능 평가가 수반되어야 할 것이다.

참고문헌

[1] David Stepner 외 2 명, " *Embedded Application Design Using a Real-Time OS*" , IEEE, 1999.

[2] Jean, J, Laborosse, " *μ C/OS The Real-Time Kernel*" , R&D Publications, 1995.

[3] C.M.Krishna, Kang G.Shin, " *Real-Time Systems*" , McGraw-Hill, 1995.

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

[5] Orv Balcom, " *Simple Task Scheduler Prevents Priority Inversion*" , Embedded Systems Programming, 1995.

[6] 이재호, " *The Design and Implementation of the real-time operating system for CalmRISC*" , 충남대학교 석사학위논문, 2001

[7] 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