우선순위 역전을 해결하기 위한 세마포어의 구현*

양희권[0], 윤기현, 성영락[†], 이철훈
충남대학교 컴퓨터공학과, [†]국민대학교 전자정보통신공학부
{hkyang, khyoon, chlee}@cc.cnu.ac.kr, [†] yeong@mail.kookmin.ac.kr

Implementation of Semaphores to Prevent Priority Inversion

Hui-Kwon Yang[0], Ki-Hyun Yoon, Yeong Rak Seong[†], and Cheol-Hoon Lee
Dept. of Computer Engineering, Chungnam National Univ.
[†] School of Electrical Engineering, Kookmin Univ.

요 약

실시간 운영체제(Real-Time OS)는 우선순위 기반의 선점형 스케줄링을 제공하는 운영체제로서 시간 결정성(Determinism)을 보장하는 특징이 있다. 그러나, 우선순위가 높은 태스크가 우선순위가 낮은 태스크에 의해 CPU 를 점유 당하는 우선순위 역전(Priority Inversion)이 발생하여 시간 결정성이 보장되지 못하면 시스템의 심각한 결함을 야기할 수 있다. 본 논문에서는 우선순위 역전을 해결하기 위하여 Priority Inheritance Protocol 을 적용한 세마포(Semaphore)의 구현에 대해 기술한다.

1. 서 론

실시간 운영체제는 멀티 태스킹을 지원하는 운영체제로서 우선순위 기반의 선점형 스케줄러를 제공한다. 일반적으로 실시간 운영체제에서는 실행 준비 상태의 태스크들 중 가장 높은 우선순위의 태스크가 CPU 를 점유해야 하지만 임계구역(critical section)에 대한 상호 배타적 실행이 허용되기 때문에 높은 우선순위의 태스크가 실행 준비 상태가 되었음에도 불구하고 낮은 우선순위의 태스크에 의해 CPU 를 점유하지 못하는 문제가 발생할 수 있는데, 이를 우선순위 역전(Priority Inversion)이라 한다. 우선순위 역전이 발생하면 시간 결정성이 보장되지 못할 뿐만 아니라 중요한 태스크가 실행되지 못함으로 인한 시스템의 심각한 결함을 유발할 수 있기 때문에 실시간 운영체제의 설계에 있어 이 문제에 대한 고려가 필요하다. 우선순위 역전의 해결 방안으로는 우선순위 역전의 원인이 되는 태스크의 우선순위를 현재 실행 준비 상태에 있는 최상위 태스크의 수준으로 우선순위를 일시적으로 올려주어 실행하게 하는 Priority Inheritance Protocol 과 우선순위 역전 발생 가능 영역을 수행하는 태스크들의 우선순위 한계를 지정하여 해당 영역이 수행될 때 임의의 우선순위로 실행되게 하는 Priority Ceiling Protocol 이 있다. 본 논문에서는 우선순위 역전을 능동적으로 해결하기 위해 세마포에 Priority Inheritance Protocol 을 적용하였다.

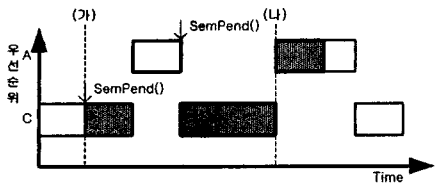
본 논문의 구성은 2 장에서 우선순위 역전의 해결방안에 대한 기반 연구를 설명하고 3 장에서는 Priority Inheritance 방식을 이용하여 구현한 세마포를 설명한다. 4 장에서는 테스트 환경 및 결과를 설명하고 5 장에서는 결론 및 향후 과제를 기술한다.

2. 기반연구

2.1 우선순위 역전(Priority Inversion)

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 위해 세마포를 제공한다. 임의의 우선순위를 가진 태스크가 실행중에 보다 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 역전이라 한다. 우선 순위 역전이 발생하게 되면 기존의 높은 우선순위를 가진 태스크의 실행은 세마포를 가진 태스크가 세마포를 반납할 때까지 실행이 지연된다. 더욱이, 이들 두 태스크 사이의 우선순위를 가진 태스크에 의해 높은 우선 순위를 가진 태스크의 실행 지연시간은 더 길어지게 되고, 시스템에 치명적인 오류를 낳게 된다. 아래 [그림 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

*본 논문은 산업자원부 중기거점과제 연구비 지원에 의한 것임



[그림 1] 우선순위 역전

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

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 할당받아 실행할 수 있도록 하는 방식이다.

2.3 Priority Ceiling Mutex

태스크가 Mutex 를 할당받을 때, Mutex 생성시 미리 정해진 우선순위로 태스크의 우선순위를 증가시켜 Mutex 를 해제할 때까지 안정적으로 실행할 수 있도록 하는 방법이다. 프로그래머가 시스템 설계시 Mutex 를 사용하게될 태스크들의 우선순위중 가장 높은 값을 기본값으로 설정하면 태스크들이 Mutex 를 사용하는 동안에는 우선순위가 기본값으로 변경되도록 함으로써 우선순위 역전을 미연에 방지하도록 한 방법이다.

3. 우선순위 역전을 해결하기 위한 세마포의 구현

일반적으로 단일 공유자원을 다수의 태스크들이 접근하고자 할 때, 우선순위 역전이 발생할 수 있다. 그리고, 두개 이상의 다중 공유자원에 있어서도 개체의 수가 부족해질 경우 우선순위 역전의 소지가 있다. 시스템에서 시간결정성(Determinism)이 중요시 될 때 우선순위 역전을 최단시간내에 해결해야 할 필요가 있다. 본 연구에서는 우선순위 역전에 대비한 별도의 Mutex 를 제공하는 대신에 Priority Inheritance Protocol 을 적용한 세마포를 제공함으로써 세마포를 사용하는데 있어 예기치 못한 우선순위 역전의 문제를 커널이 능동적으로 해결할 수 있도록 하였다.

3.1 세마포 구조체

[그림 2]에서와 같이 이미 구현된 세마포 구조체에 다음의 몇 가지 변수를 추가하여 세마포가 할당된 태스크들을 이중연결 리스트로 관리할 수 있도록 하였다.

```
struct sa_List {
        MK_Task          *sa_Task;
        struct sa_List   *sa_Prev;
        struct sa_List   *sa_Next;
}
typedef struct mk_semaphore_struct {
    int      s_Count;
  < 생 략 >
    struct sa_List  *s_AllocList;
```

[그림 2] 세마포 구조체

이 리스트는 우선순위 역전이 발생할 경우 우선순위를 상속할 태스크를 선택하기 위한 것으로 세마포 생성시 s_Count 만큼 동적으로 생성된다. 비어있는 리스트는 s_AllocListFree 에 의해 관리되고 태스크 구조체 포인터들의 리스트는 시작점을 가리키는 s_AllocListStart 와 최근에 추가된 노드를 가리키는 s_AllocListCur 에 의해 관리된다. s_HiPrio 는 세마포가 할당된 태스크들중 가장 높은 우선순위를 의미한다.

3.2 세마포 할당/반납

세마포가 할당 가능하면, 세마포가 할당된 태스크리스트에 태스크 포인터를 추가하고 현재 태스크의 우선순위가 리스트에 있는 태스크들보다 높으면(s_HiPrio > MK_GetCurrentPriority()) s_HiPrio 를 변경한 후 세마포를 할당한다. 여분의 세마포가 없다면 현재 태스크를 대기상태로 만들고, s_HiPrio 와 현재 태스크의 우선순위를 비교한다. s_HiPrio 가 현재 태스크의 우선순위보다 낮으면, 우선순위 역전으로 간주하고 태스크리스트에서 실행 준비 상태에 있는 임의의 태스크를 선택한다. 임의의 태스크를 선택할 때, 태스크 리스트에서 우선순위가 가장 높은 태스크를 선택하는 방법과 세마포가 할당된지 오래된 태스크를 선택하는 방법을 생각해 볼 수 있다. 본 논문에서는 세마포가 할당된지 오래된 태스크에 기회를 제공하도록 하였는데, 우선순위에 의한 방법에 비해 세마포의 활용도를 높이고 리스트 탐색과 관련한 연산을 줄여주는 장점이 있다. 이에 따라 가장 오래된 노드를 가리키는 s_AllocStart 에서 시작하여 실행 준비 상태에 있는 태스크를 찾는다. 우선순위를 상속받을 태스크가 선택되면, 현재 태스크의 우선순위를 상속하고, 스케줄러를 호출하여 나머지 임계구역을 수행하도록 한다. 이 때, 우선순위를 상속 받은 태스크의 본래 우선순위를 기억하고 있다가 세마포가 해제되거나 Timeout 되어 세마포를 기다리던 태스크가 깨어나면, 우선순위를 상속받았던 태스크는 원래의 우선순위로 환원된다. 다음의 [그림 3]은 세마포를 할당

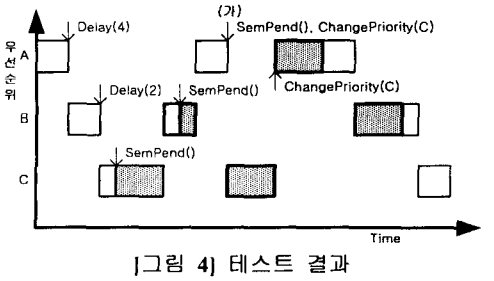
하는 SemPend()함수의 일부분으로 우선순위 역전을 해결하기위해 임의의 태스크를 선정해서 우선순위를 상속하고, 세마포가 반납되면 다시 환원하는 과정에 대한 소스 코드이다. 세마포가 반납되면, 태스크 리스트에서 관련 노드를 제거하고 s_HiPrio, s_Count 값을 재조정한 후, 세마포 반납 처리를 완료한다.

```
<생략>
if (pTask->t_Priority < pSemaphore->s_HiPrio) {
  /* Priority Inversion */
  tmpList = pSemaphore->s_AllocStart;
  do {
    if (tmpList->sa_Task->t_Status == MK_TASK_Ready) {
      SelTask = tmpList->sa_Task;
      break;
    }
    tmpList = tmpList->sa_Next;
  } while(tmpList != pSemaphore->s_AllocStart);
  if (SelTask) {
    oldPrio = SelTask->t_Priority;
    MK_TaskChangePriority(SelTask, pTask->t_Priority);
    pSemaphore->HiPrio = pTask->t_Priority;
  }
}
```

[그림 3] SemPend() 함수

4. 테스트 환경 및 결과

본 연구는 ARM-920T CPU 가 탑재된 SMDK2400 보드에서 SDT2.51 ARM 용 통합개발환경으로 진행되었다. 커널은 본 연구팀에서 자체 개발한 iRTOS™을 사용하여 세마포만을 새롭게 구현하였다. 아래 [그림 4]의 테스트 결과를 보면 태스크들의 우선순위가 A> B> C 이고 이들은 모두 초기상태 s_Count = 2 인 세마포(S)를 사용한다. (가) 시점에 이르러 가장 높은 우선순위를 가진 태스크 A가 MK_SemPend()를 호출하면 사용가능한 세마포(S)가 없기 때문에(s_Count = -1), 태스크 A는 대기상태가 되고 세마포(S)를 가진 태스크들 중 C가 선택되어 A 의 우선순위를 상속받아 나머지 임계구역을 수행한 후 세마포(S)를 반납하면, 태스크 C 의 우선순위는 이전의 값을 회복하고 태스크 A 가 세마포를 할당받아 작업을 수행하게 되는 것을 알 수 있다.



[그림 4] 테스트 결과

5. 결론 및 향후 과제

본 논문에서는 실시간 운영체제에서 사용되는 세마포에 Priority Inheritance Protocol 을 적용함으로써, 커널에 의해 우선순위 역전을 능동적으로 해결할 수 있도록 하였다. 그러나 우선순위 역전 해결 기능이 없던 세마포에 비해, 우선순위 역전을 탐지하고 해결하는데 대한 연산량과 메모리 사용량이 증가되는 문제를 안고 있다. 특히 증가된 연산의 대부분은 세마포를 사용하는 태스크들에 대한 리스트를 관리하는데에 관한 것이다. 따라서, 향후 이들의 측면에서 시스템의 부담을 감소하고 우선순위 역전을 더욱 빨리 해결할 수 있는 방안에 대한 연구가 진행되어야 할것이다.

6. 참고문헌

[1] http://www.inestech.com
[2] David Stepner 외 2 명, "Embedded Application Design Using a Real-Time OS", IEEE, 1999.
[3] Jean, J. Labrosse, "μC/OS The Real-Time Kernel", R&D Publications, 1995.
[4] C.M.Krishna, Kang G.Shin, "Real-Time Systems", McGraw-Hill, 1997
[5] David Kalinsky, "Mutexes Prevent Priority Inversions", Embedded Systems Programming, 1998.
[6] Orv Balcom, "Simple Task Scheduler Prevents Priority Inversion", Embedded Systems Programming, 1995.
[7] 이재호, "CalmRISC 용 실시간 운영체제 설계 및 구현", 충남대학교 석사학위논문, 2001.
[8] 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.