2001 년 한국정보처리학회 추계 학술발표논문집 제 8 권 제 2 호

우선순위 역전시간 최소화를 위한 uCOS 에서의 확장 MuTexS 설계 및 구현*

°이재호, 김선자, 김흥남
한국전자통신연구원 인터넷정보가전연구부
email: {jhlee, sjkim, hnkim}@etri.re.kr

The Design and Implementation of Advanced MuTexS For Minimizing Priority Inversion Time In uCOS

Jae-Ho Lee°, Heung-Nam Kim, Sun-Ja Kim
Internet Appliance Technology Dept. ,ETRI

요 약

본 논문은 실시간 운영체제에서 높은 우선순위를 가지는 태스크가 낮은 우선순위를 가지는 태스크가 사용중인 공유자원을 기다리는 과정에서 발생하는 우선순위 역전현상을 해결하기 위한 효과적인 방법에 대해 언급한다. 우선순위 역전현상은 실시간 운영체제의 주요 특징인 태스크 수행 완료의 바운드 타임을 예측하기 어렵게 만들어 실시간 운영체제를 사용하는 가장 큰 목적인 결정성(determinism)을 보장 받지 못하게 된다. 이를 해결하기 위해 논문에서 구현된 커널은 비교적 크기가 작으면서도 실시간 운영체제의 핵심적 특징을 잘 갖추고 있는 **uCOS**(Micro C/OS) 커널을 사용하였으나, 유일한 우선순위만을 갖는 uCOS 의 제약사항을 보완하고 Priority Inheritance Protocol 을 이용한 **MuTexS** (Mutual Exclusion Semaphore)를 구현하기위해 커널의 자료구조를 확장하여 수정된 스케줄링 방식을 사용하였다.

1. 서론

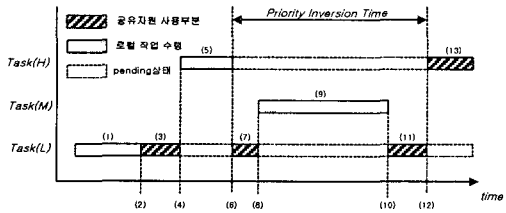
마이크로 프로세스를 가지고 있는 내장형 시스템에서 동작되는 태스크들을 효율적으로 관리하기 위해 범용 운영체제와 사용목적이 다른 실시간 운영체제가 쓰인다. 실시간 운영체제는 각종 태스크들이 올바른 수행 결과를 데드라인에 맞춰 출력할 수 있는 결정성을 가져야 하는데 이러한 요소를 파괴하는 것이 우선순위 역전 현상이다.
본 논문에서는 이를 해결하기 위해 웹에 공개된 실시간 운영체제를 분석하여 우선순위 상속기법을 적용하기 위해 커널을 수정하여 흔히 MuTexS 라고 불리는 Mutual Exclusion Semaphore 를 구현하였다.

본 논문의 2 장에서 우선순위 역전문제(**PIP**: Priority Inversion Problem)의 개념을 기술하고, uCOS 실시간 운영체제의 기본 구조와 특히 PIP 를 해결하기 위해 uCOS 에서 사용한 방법을 관련연구로서 기술하였고, 3 장에서는 기존 uCOS 가 가지는 커널 구조상의 제약사항을 극복하기 위해 수정된 자료구조를 제시하고 이를 이용한 향상된 스케줄링 방식과 효율적인 MuTexS 를 설계 및 구현하였으며, 4 장에서는 구현된 커널에 대한 검증실험 및 향후 연구과제를 기술하였다.

2. 관련연구

2.1 실시간 운영체제에서의 PIP

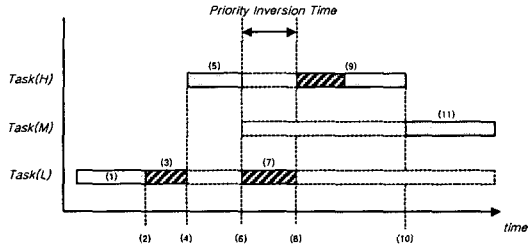
[그림 1]은 실시간 운영체제에서 동작되는 태스크들이 공유자원을 lock/unlock 하는 과정에서 발생되는 우선순위 역전현상이 일어나는 시나리오를 보이며, 각 태스크 우선순위는 (H>M>L)과 같다.



[그림 1] Priority Inversion Problem

(1)-(3) Task(L)이 자신의 작업을 수행하다가 공유자원을 접근하기 위해 semaphore 를 얻어 계속 자신의 일을 수행 중이다.
(4)-(5) Task(L)은 Task(H)에 의해 preemption 되어 Task(H)가 CPU 사용권을 얻어 자신의 일을 수행한다.
(6) Task(H)가 수행 중 Task(L)에 의해 잠겨진 공유자원을 사용하려고 하나, 이미 Task(L)이 점유하고 있으므로 semaphore 를 얻기 전까지 자원에 접근할 수 없으므로 CPU 사용권을 다른 태스크에 넘겨야 한다.

(7)-(9) Task(L)은 semaphore 에 의해 묶어놓은 공유자원을 접근하여 하던 작업을 계속 수행하던 중 Task(M)에 의해 preemption 되고, Task(M)이 CPU 사용권을 가지고 자신의 작업을 수행한다.
(10)-(11) Task(M)이 수행을 마치고 스케줄링이 일어날 때, Task(H)가 사용하고자 하는 자원은 아직도 Task(L)의 semaphore 에 의해 묶여 있으므로, Task(L)이 공유자원을 해제시킬 때까지 블로킹 당하게 되므로 Task(L)이 수행된다.
(12)-(13) Task(L)이 공유 자원의 사용을 마치고 해제 시켰을 때 비로서 Task(H)가 이 자원을 얻어서 자신의 작업을 재개하게 된다. 그러나, 위 과정에서 Task(M)의 응용프로그램 수행시간에 따라 Task(H) 수행완료 시점이 예측할 수 없는 시간이라면 실시간 운영체제의 결정성을 보장하지 못하는 오류가 발생하게 된다.

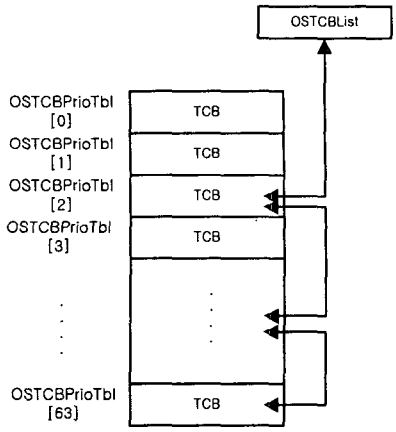


[그림 2] Priority Inheritance

[그림 2]는 PIP 를 해결하기위해 priority inheritance 의 개념을 도입하였으며 동작 시나리오는 다음과 같다.
(1)-(5) [그림 1]의 시나리오와 동일하게 수행된다.
(6)-(7) Task(H)가 수행도중 Task(L)에 의해 잠겨진 공유자원을 사용하기 위해 semaphore 를 요청하나, 커널은 Task(L)에 의해 이미 사용하고 있음을 알린다. 이때 발생하게 되는 PIP 를 해결하기위해 Task(L)의 우선순위를 Task(H)와 동일한 우선순위로 고쳐서 Task(M)에 의해 preemption 당하지 않고 계속해서 공유자원에 접근하여 자신의 작업을 수행한다.
(8) Task(L)은 공유 자원의 사용을 마치고, semaphore 를 커널에 반환하고 이때 앞서 높여 주었던 우선순위를 다시 원래대로 갱신한다.
(9) Task(H)가 쓰고자 하는 공유자원을 접근하기 위해 Task(L)이 해제한 semaphore 를 얻어 자신의 작업을 마친다.
(10)-(11) Task(L)이 원래의 우선순위로 돌아갔으므로, 스케줄 되어 Task(M)이 자신의 작업을 수행한다.

2.2 uCOS 실시간 운영체제에서 PIP 해결 방법

uCOS 에서는 64 개의 우선순위를 가지며, 어떠한 Task 도 동일한 우선순위를 가지고 커널에 등록될 수 없으므로, 동일한 우선순위를 가진 태스크들에게 부여되는 타임 퀀텀을 가지고 수행되는 라운드 로빈(Round Robin)방식의 스케줄링을 지원하지 못한다. [그림 3]은 uCOS 의 태스크 관리 구조로서 커널을 초기화 할 때 64 개의 TCB(Task Control Block)배열을 생성하여 OSTCBPrioTbl[64]에 등록시키고, Task 의 생성, 삭제, 스케줄링은 Task 들간의 연결 구조인 OSTCBList 를 사용한다.



[그림 3] uCOS 의 태스크 관리구조

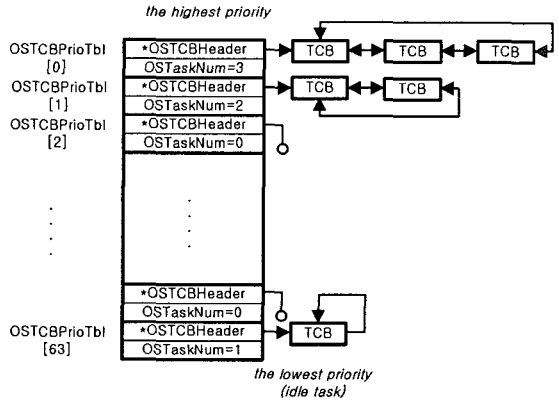
이러한 스케줄링 구조에서는 PIP 를 해결하기 위해 Priority Inheritance Protocol 을 사용할 수 없으므로 이와 유사한 개념을 가진 Priority Ceiling Protocol 을 사용한다. uCOS 에서 생성된 MuTexS 는 응용 프로그램 작성자에 의해 동일한 자원을 점유하는 태스크들의 우선순위를 고려하여 새로운 태스크가 등록되는 것처럼 적당한 우선순위로 OSTCBPrio[64]에 등록된다. 이때 부여 받게 되는 우선순위 값의 결정은 동일한 자원을 사용하는 태스크 중 가장 높은 우선순위 값보다 더 높은 우선순위를 부여해야 한다. 이것은 낮은 우선순위를 가진 태스크가 자원을 점유하면서 suspend 되어 있는 동안, 이 보다 높은 우선순위의 태스크가 자원을 사용하여 자신의 작업을 수행하려고 하면 다른 태스크에 의해 사용중인 자원이 해제 될 때까지 기다리는 시간을 최소화 시키기 위한 것이다. MuTexS 를 생성할 때 예약해둔 우선순위는 자원을 소유하고 있는 태스크의 우선순위를 일시적으로 높여 작업을 신속히 끝내고 자원을 해제하도록 하기 위해 쓰이며, 자원을 해제 하는 과정에서 변경 되었던 태스크의 우선순위가 본래의 값으로 복원된다.

2.3 uCOS 실시간 운영체제의 제한사항

Priority Ceiling Protocol 을 사용하는 uCOS 커널의 한계점은 다음과 같다. 첫째 커널을 초기화 할 때 사용되지 않을 태스크들을 위한 64 개의 TCB 를 미리 생성하므로, 이에 따른 메모리 오버헤드가 존재한다. 둘째, 태스크가 동일한 우선순위를 갖도록 지원하지 못하므로 MuTexS 를 생성할 때 마다 태스크가 등록되어야 할 OSTCBPrio[64]에 등록해 주어야 한다. 따라서 많은 태스크들이 여러 개의 자원을 공유하면서 동시 접근의 안전성을 보장하기 위해 많은 MuTexS 를 생성하여 사용하는 경우에 있어서, 64 개로 제한된 우선순위가 부족할 수 있으므로 그러한 환경에는 적합하지 못하다. 셋째, MuTexS 를 위한 우선순위 값을 응용프로그램 작성자가 직접 지정해 주어야 하므로 여러 개의 MuTexS 를 사용하는 경우 응용프로그램에서 태스크가 사용하지 않는 우선순위를 찾아서 일일이 할당해줘야 하는 번거로움이 있다.
위와 같은 단점을 극복하기 위해서 uCOS 실시간 운영체제가 Priority Inheritance Protocol 을 지원할 수 있도록 커널의 자료구조를 수정하고 이를 이용하여 스케줄러 모듈을 수정하고 3 절과 같이 효율적인 MuTexS 모듈을 구현하였다.

3. 확장커널의 MuTexS 설계 및 구현

3.1 라운드 로빈 스케줄러 지원을 위한 커널 수정



[그림 4] 라운드로빈 스케줄러를 지원하는 확장 커널 구조

2 절에서 관련연구로서 소개한 Priority Inheritance Protocol 을 사용하기 위해서는 커널의 스케줄러가 라운드 로빈 방식을 지원해야 하므로 기존의 TCB 를 관리하는 자료구조와 스케줄링을 담당하는 모듈을 [그림 5]와 같이 수정하였다.

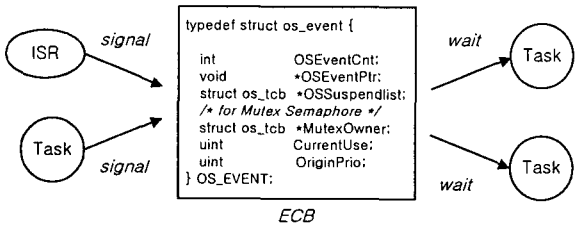
```
OSScheduler()
{
(1)  SavedQuantum = CurrentTask's TimeQuantum;

(2)  OSTCBHighRdy = "get a new high priority task" ;

(3)  if ( SavedQuantum == 0 )
         CurrentTask's TimeQuantum = Default TimeQuantum;

(4)  if ( OSTCBHighRdy == CurrentTask )
         if ( SavedQuantum == 0
         && OSTCBPrioTbl[CurrentTask's Priority].OSTaskNum >1 )
             CPU Context Switch to CurrentTask's Neighbor
     else {
(5)      if ( SavedQuantum == 0 ) {
             CurrentTask's Quantum = 0;
             "Put CurrentTask into Scheduling Ready List";
         }
(6)      CPU Context Switch to OSTCBHighRdy;
     }
}
```

[그림 5] 수정된 스케줄러 알고리즘

(1) 현재 태스크의 남아 있은 퀀텀 값을 저장한다.
(2) 스케줄링을 기다리는 태스크 중 우선순위가 가장 높은 태스크를 얻는다.
(3) 현재 태스크의 퀀텀 값이 0 이면, default 퀀텀 값을 설정한다.
(4) 과정 (2)에서 새롭게 결정된 태스크의 우선순위가 현재 태스크의 우선순위와 같고, 동일한 우선순위의 또 다른 태스크가 존재하면 그 태스크로 문맥 전환을 한다.
(5)-(6) 과정 (2)에서 새롭게 결정된 태스크의 우선순위가 현재 태스크의 우선순위와 다르면 새로운 태스크로 문맥전환을 한다. 이때 현재 태스크의 퀀텀 값이 0 이었다면, (3)에서 설정했던 default 퀀텀 값을 다시 0 으로 설정하고 스케줄링 대상이 될 수 있도록 Ready List 에 넣어준다.

3.2 MuTexS 모듈 구현 내용

실시간 운영체제에서는 태스크들간의 동기화와 통신을 위해서 semaphore, 메시지 메일박스, 메시지 큐 등을 제공한다. 여기에 실시간 운영체제에서 발생될 수 있는 PIP 을 해결 하기 위해 MuTexS 모듈을 구현하였다. [그림 6]은 기존 이벤트 관련 서비스를 제공하기 위한 ECB(Event Control Block) 구조체에 MuTexS 구현을 위해 세가지 필드를 추가하였다.



[그림 6] Event Control Block

- OSEventCnt : 이벤트의 type 이 semaphore 일 때 사용되어 semaphore 의 개수를 나타낸다.
- OSEventPtr : 이벤트의 종류가 메시지 메일박스 또는 메시지 큐일 때 사용되며, 메시지나 데이터 구조를 가리키는 포인터로 사용된다.
- OSSuspendlist : 이벤트에 Suspend 된 태스크들의 리스트 헤더를 가리키는 포인터로 사용되며, Suspend 된 태스크들은 우선순위에 따라 단일 연결 리스트로 구성된다.
- MutexOwner : MuTexS 를 소유하고 있는 TCB 를 가리킨다.
- CurrentUse : MuTexS 자원의 사용 여부를 나타낸다.
- OriginPrio : Priority Inversion 이 발생한 경우, lower priority task 의 우선순위가 높아지는데, 이때 변경되기 이전의 우선순위를 저장하기 위해 사용된다. MuTexS 를 사용을 마친 태스크는 이를 반환할 때 OriginPrio 필드를 이용하여 본래의 우선순위로 복귀하게 된다.

MuTexS 를 구현한 API 들은 다음과 같다.

- OSMutexMake(OS_EVENT *pevent)

공유자원을 배타적으로 접근할 수 있도록 binary semaphore 를 생성시킨다.

- OSMutexGet(OS_EVENT *pevent, uint timeout, uint *err)

MuTexS 가 가용하면, 이를 소유할 태스크의 우선순위와 주소를 저장하고 복귀한다. 한편 mutex 가 사용 중이면, 이를 소유하고 있는 태스크와 현재 태스크의 우선순위를 비교한다. 만약, mutex 를 요청한 태스크의 우선순위가 더 높다면, MuTexS 를 소유한 태스크의 우선순위를 현재 mutex 를 요청한 태스크의 우선순위 만큼 높여준다(Priority Inheritance Technique). 현재 태스크는 mutex 를 무한정 기다리지 않도록 timeout 값을 설정하고, resheduling 을 통해 CPU 를 점유할 다음 태스크를 선정한다.

- OSMutexRelease(OS_EVENT *pevent)

MuTexS 를 소유한 태스크만이 OSMutexRelease() 함수를 이용하여 사용한 MuTexS 를 해제할 수 있다. MuTexS 를 반환하는 태스크의 우선순위와 OS_EVENT 구조체 내에 저장된 본래의 우선순위(OriginPrio)가 동일한가 검사한다. 만약 다르다면, 이전에 priority inheritance 가 발생한 것이므로 원래의 우선순위로 복귀시킨다. 이때 MuTexS 를 기다리는 태스크가 존재하면, MuTexS 를 점유할 수 있도록 MuTexS 를 새로운 태스크에 할당 시키고, 만약 이를 기다리던 태스크가 없다면

MuTexS 가 사용 가능함을 표시하고 rescheduling 을 통해 CPU 를 점유할 태스크를 선정하여 수행을 계속한다.

4. 결론 및 향후 연구 과제

우선순위 역전 현상은 실시간 운영체제의 가장 큰 특징인 태스크 수행 완료의 예측성과 결정성을 파괴하는 요소이다. 따라서 본 논문에서는 이러한 예측할 수 없는 priority inversion time 을 줄이기 위한 방법으로 Priority Inheritance 방법을 사용하여 MuTexS 를 구현 하였다. [그림 7]의 실험 코드는 세 개의 태스크를 생성시키고(TaskH/TaskM/TaskL), 두 개의 태스크(TaskH/TaskL)가 공유자원을 배타적으로 사용하기 위해 semaphore 를 사용한 경우와 MuTexS 를 이용한 경우를 보여준다. [그림 8]의 실험 결과는 Semaphore 를 사용하여 공유자원에 접근 한 경우(왼쪽 그림)에는 MuTexS 를 사용한 경우(오른쪽 그림)보다 훨씬 길고, 예측하기 힘든 priority inversion time 을 갖게 된다. 즉, semaphore 만을 사용한 실험에서 TaskH 의 우선순위가 가장 높음에도 불구하고 TaskM 의 수행시간에 의존하는 경우가 발생하므로, TaskH 수행완료의 time-bound 예측이 어렵다는 문제가 발생하는 것이다. 이는 MuTexS 를 사용함으로써 PIP 를 완전히 해결할 수는 없지만, Semaphore 만을 사용하는 경우보다 priority inversion time 이 훨씬 줄어들기 때문에 실시간 운영체제를 사용함으로써 얻는 태스크 수행 완료시간의 바운드 타임을 보장해 줄 수 있는 측면에서 매우 중요한 기능을 제공하는 요소이다.

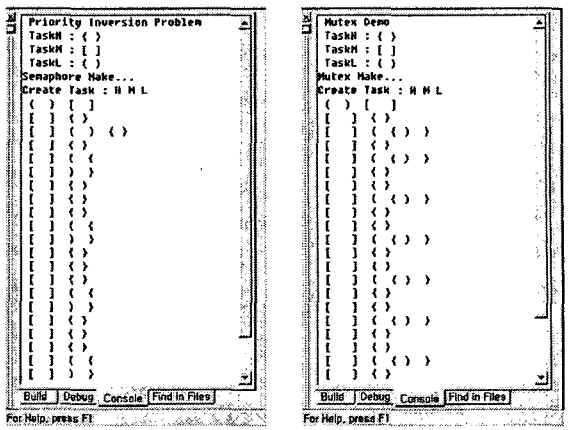
```
void TaskH(void *Para)
{
  uint Reply;
  for(;;) {
    printf(" {");

    OSSemGet(&Resource,0,&Reply);
    // OSMutexGet&Resource,0,&Reply);

     / * --- user code ---*/

    OSSemRelease(&Resource);
    //OSMutexRelease(&Resource);

    printf(" } ");
    OSTimeDly(2);
  }
}
```

```
void TaskM(void *Para)
{
  uint Reply;
  printf("M₩n");
  for(;;) {

    printf(" [ ");

     /* --- user code --- */

    printf(" ] ");
    OSTimeDly(2);
    printf("₩n");
  }
}
```

```
void TaskL(void *para)
{
  uint Reply;
  for(;;) {
    printf(" (");

    OOSSemGet(&Resource,0,&Reply);
    // OSMutexGet&Resource,0,&Reply);

     /* --- user code --- */

    OSSemRelease(&Resource);
    //OSMutexRelease(&Resource);

    printf(" )");
    OSTimeDly(2);
  }
}
```

```
OS_EVENT Resource;

int main(void)
{
  char Id1='H';
  char Id2='M';
  char Id3='L';

  OSInit();

  OSSemMake(&Resource);
  // OSMutexMake(&Resource);

  OSTaskCreate(TaskH,  . . .  , 10);
  OSTaskCreate(TaskM,  . . .  , 20);
  OSTaskCreate(TaskL,  . . .  , 30);

  OSStart();
}
```

[그림 7] MuTexS 를 이용한 검증코드



[그림 8] MuTexS 를 이용한 실험 결과

본 논문에서는 실시간 운영 체제에서 태스크들간에 공유자원을 사용하는데 발생하는 우선순위 역전현상을 해결하기 위해 Priority Inheritance Protocol 을 사용하기 위해 라운드 로빈을 지원하는 스케줄러와 일반적인 Semaphore 구조를 확장한 MuTexS 모듈을 설계 하고 구현하였다. 본 연구에 이어 계속 연구되어져야 할 부분은 운영체제의 정확성 및 신뢰성의 향상을 위해 스케줄링과 관련하여 구현된 모듈이 얼마나 태스크 완료의 예측성을 제공하는지 갖가지 환경에서의 로직 분석기에 의한 타이밍 분석이 필요할 것이다. 아울러 다른 실시간 운영체제와의 비교 기준을 정의하고, 이 기준에 따른 성능 평가가 수반되어야 할 것이다.

5. 참고문헌

[1] L Sha. ,R. Rajkumar and J. P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization", IEEE Transactions On Computers, Vol. 39, No 9. SEPTEMBER 1990

[2] John B. Goodenough, Lui Sha, "The Priority Ceiling Protocol: A Method for Minimizing the Blocking of High-Priority Ada Tasks", CMU/SEI-88-SR-4, March, 1988, http://www.sei.cmu.edu/

[3] Jean J.Labrosse, "uCOS The Real-Time Kernel", R&D Publications Lawrence,Kansas 66046

[4] Jean J.Labrosse, "UC/OS-II and Mutual Exclusion Semaphores", Application Note AN-1002, www.ucos-ii.com

[5] David E. Simon, An Embedded Software Primer, Addison-Wesley, 1999

[6] C.M.Krishna, Kang G. Shin, Real Time Systems, The McTGraw Hill Companies INC. ,1998