論 文

다중처리기 시스템에서 실시간 쓰레드 동기화에 관한 연구

正會員 朴政亨　　正會員 宋周錫

A Study on Real-Time Thread Synchronization in Multiprocessor Systems

Jung Hyung Park, Joo Seok Song *Regular Members*

要　　約

본 논문에서는 다중처리기 시스템, 다중 쓰레드 구조에서 GCS의 내포로 인한 교착상태의 발생을 방지하고 실시간 다중 쓰레드 스케쥴링의 기본이 되는 우선순위에 기초한 쓰레드간의 동기화에 관한 알고리즘을 설계하였다. 우선순위 상속 프로토콜과 우선순위 상한 프로토콜을 이용하여 다중처리기 시스템에 대한 개념을 확장하고, 광역 임계구역이 다른 임계구역에 내포된 경우 유발될 수 있는 교착상태를 해결할 수 있는 방법을 제시하고, 주기적인 쓰레드들의 클래스별 제한시간 실패율을 시뮬레이션으로 조사해 보았다.

ABSTRACT

This paper presents a priority-based algorithm for synchronization among threads which prevents a deadlock in multiprocessor multithread system caused by the inclusion of GCS and is the basis of multithread scheduling. It expands the concept of multiprocessing systems using the priority inheritance protocol and the priority ceiling protocol, presents a method for solving deadlock which can occur when a global critical section is nested in another critical section, and examines the deadline miss ratio of periodic threads for each class through simulation.

I. 서　　론

스케쥴링은 한정된 하드웨어의 자원을 이용하여 사용자의 작업을 적절한 형태로 최대한 빨리 수행되도록 하는 것으로 운영체제의 핵심이라고 할 수 있다.[4] 실시간 시스템에서는 각 사건의 종류에 따라 제한 시간의 보장과 응답시간의 준수를 필요로 한다.[10][12] 또한 작업의 제한시간을 만족하는 범위내에서는 자원

三星綜合技術院
Samsung Advanced Institute of Technology
論文番號 : 9372
接受日字 : 1993年 5月 10日

의 유용성을 증대시키는 방향으로 스케쥴링을 행하나, 제한시간을 넘기게 되는 경우는 공평성의 원칙이 무시되고 긴급한 작업의 제한시간을 만족시키기 위하여 긴급하지 않은 작업의 소모를 감수하더라도 우선적으로 스케쥴링을 해야 한다.[3]

쓰레드는 하나의 프로세스를 하나이상의 수행단위로 분할한 것으로 더 높은 병렬성을 지원하나, 동기화의 문제등 여러가지 비용의 낭비요소가 많아서 구현에 어려움이 있다.[1]. 그러나 더 높은 수준의 병렬성을 지원하기 위해서 연구가 필요하다. 프로세스 단위로 우선순위에 따라 스케쥴링을 행하는 경우, 하나의 프로세스는 하나의 프로세스에 할당되어 처리되므로 제한된 시간내에 처리가 어려운 경우가 발생할 수 있으나, 쓰레드로 분할되어 처리를 하는 경우는 처리속도가 증가되므로 제한시간 실패율을 감소시킬 수 있다.

본 논문에서는 이러한 실시간 다중 쓰레드 스케쥴링의 기본이 되는 쓰레드간의 동기화에 관한 알고리즘을 설계하였다. 광역 임계구역(Global Critical Section: GCS)이 다른 임계구역에 내포된 경우 발생할 수 있는 교착상태를 해결할 수 있는 방법을 제시하고 주기적인 쓰레드들의 클래스별 제한시간 실패율을 시뮬레이션으로 조사해 보았다.

2. 실시간 시스템 동기화 문제

2.1 Rate-Monotonic 알고리즘

엄격한 제한시간을 가지는 주기적인 태스크들의 스케쥴링의 문제는 1973년 Liu와 Layland에 의해 처음으로 연구되었다[6][7][10][13]. 그들의 논문에서 적절한 정적 우선순위와 동적 우선순위 스케쥴링 알고리즘을 제시하였으며 최악의 경우에 대한 결정을 하였다. 주기가 긴 태스크들에 비해 주기가 짧은 태스크들에 높은 우선순위를 부여하는 적절한 정적 우선순위 알고리즘을 'Rate-Monotonic' 알고리즘이라 한다[7].

수행시간을 C_i라 하고 제한시간을 T_i라 할때 이들 태스크들이 제한시간을 제한시간을 만족시키기 위한 충분조건은 다음과 같다[11][13].

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \cdots + \frac{C_n}{T_n} \le n(2^{1/n} - 1)$$

즉 $\sum u_i \le n(2^{1/n} - 1)$

n=2일때 0.83에서 $n \to \infty$일때 $\log_e 2 = 0.693$으로 점차적으로 감소한다. 따라서 임의의 크기를 가진 주기적인 태스크의 집합은 rate-monotonic 알고리즘이 사용되며 CL 전체 효율이 0.693보다 크지 않으면 모든 제한시간을 만족시킬수 있다[7].

이 rate-monotonic 알고리즘은 다음과 같은 중요한 성질을 가진다[7].

(1) 이것은 일시적인 과부하가 발생했을때 가장 중요한 태스크들의 시간적 요구조건을 만족시키는 것을 보장할 수 있다.
(2) 이것은 우선순위 상한 프로토콜(priority ceiling protocol)을 사용한 동기화에 이용될 수 있다.
(3) 불명확한 계산이 허용되는 태스크들을 스케쥴링하는데 편리하게 사용될 수 있다.
(4) 이것은 프로세서, I/O 제어기, 통신매체에서 쉽게 구현될 수 있다.

태스크들이 공유데이타를 사용하는 경우 이 데이타에 대한 액세스를 순서화시켜야 한다. 이러한 공유데이타에 대한 액세스에서 우선순위가 낮은 태스크가 우선순위가 높은 태스크의 수행을 방해하는 경우가 발생할 수 있다. 이러한 현상을 우선순위 반전(priority inversion)의 문제라 한다[10].

2.2 우선순위 반전의 문제

우선순위 반전은 두개이상의 태스크들이 공유자원을 액세스하려고 할때 발생한다. 우선순위가 낮은 태스크가 먼저 액세스를 하고 나중에 우선순위가 높은 태스크가 액세스를 하려는 경우, 우선순위가 높은 태스크는 우선순위가 낮은 태스크가 공유자원에 대한 액세스를 끝낼때까지 블럭된다. 우선순위 반전의 문제를 그림으로 표시하면 아래의 [그림1]과 같다.

위의 그림에서 우선순위가 낮은 태스크 L이 임계구역에서 선점되고 있는 경우 우선순위가 높은 태스크 H는 임계구역에 들어가려고 시도하나 실패하여 블럭된다. H는 L이 임계구역을 떠나기 전까지는 수행을 할 수가 없게 되어 계속 대기하게 되며, H와 L사이의 우선순위를 가진 다른 태스크들이 L의 수행을 방해하

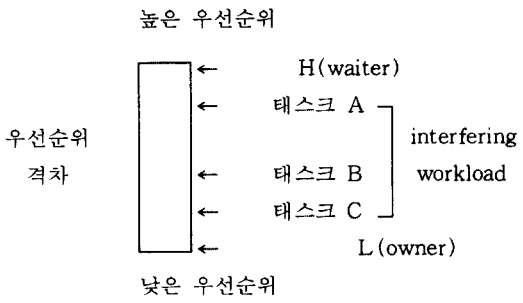


그림 1. 우선순위 반전

게 된다. 따라서 H는 높은 우선순위를 가지고 있으면서 L의 우선순위와 같은 역할밖에 수행을 하지 못해 계속 대기하게 되어 제한시간을 만족시키지 못하는 경우가 많이 발생하게 된다[6][13][11][13].

이러한 우선순위 반전은 상호배제의 문맥에서 종종 발생하며 많은 실시간 시스템에서 이용하는 정적 스케쥴링 기법에서 발생하기 쉬우며 상속 프로토콜로 해결할 수 있다[10][11][13].

3. 단일처리기 시스템 동기화

단일처리기 시스템에서 동기화 방법을 살펴보기전에 다음과 같은 가정을 한다.

- ◆ 쓰레드는 프로세서에서 자신만 수행을 시작되면, 끝날때까지 프로세서를 계속 사용한다.
  즉 쓰레드는 I/O와 같은 수행을 하기 위해 스스로 블럭시키지 않는다는 것이다. 즉 프로세서내에서의 수행만을 가정한다.
- ◆ 쓰레드내의 임계구역은 완전히 내포되고 임계구역의 끝이나 수행이 끝나면 가지고 있는 모든 록을 해제한다.
- ◆ 쓰레드 τ_1, τ_2, ···, τ_n우선순위가 τ_1이 가장 높고, 점차 낮아지는 순서로 가정한다.
- ◆ 각 쓰레드는 주기적인 쓰레드와 비주기적인 쓰레드로 구분한다. 주기적인(periodic) 쓰레드는 일정간격으로 발생하는 같은 타입의 연속된 것이고, 비주기적인(aperiodic) 쓰레드는 불규칙적인 간격으로 발생하는 같은 타입의 쓰레드를 말하며, 모든 쓰레드는 초기에 우선순위를 지정받는다.

3.1 우선순위 상속 프로토콜

우선순위 상속 프로토콜의 기본적인 개념은 하나의 쓰레드가 하나이상의 우선순위가 높은 쓰레드들을 블럭시킬때 원래의 할당된 우선순위를 무시하고 블럭된 쓰레드중에서 우선순위가 가장 높은 쓰레드의 우선순위로 임계구역이 수행되도록 하는 것이다[11][13].

기본적인 우선순위 상속 프로토콜은 다음과 같이 정의된다.[13].

(1) 실행되기 위해 대기중인 쓰레드들중 가정 높은 우선순위를 가지는 쓰레드 τ가 프로세서에 할당된다.

(2) 만일 τ가 더 높은 우선순위의 쓰레드를 블럭시킬때 τ는 자신에 의해 블럭된 쓰레드들의 우선순위중에서 가장 높은 우선순위를 물려받는다.

(3) 우선순위 상속은 이행적(transitive)이다.

(4) 만약 τ가 블럭되지 않고 수행중인 쓰레드의 할당된 우선순위나 상속받은 우선순위보다 높으면 다른 쓰레드를 선점할 수 있다.

이러한 우선순위 상속 프로토콜은 교착상태를 방지하지 못한다. 또한 연속된 블럭킹(Chain of blocking)으로 블럭킹 지연시간이 길어질 수 있다. 이 우선순위 상속프로토콜의 교착상태의 문제를 해결할 수 있는 방법이 우선순위 상한 프로토콜이다[11][13].

3.2 우선순위 상한 프로토콜

이 프로토콜의 목적은 교착상태와 연속된 블럭킹의 형성을 방지하고자 하는 것으로, 기본적인 아이디어는 쓰레드τ가 다른 쓰레드의 임계구역을 선점하고 자신의 임계구역 z를 수행할 때 우선순위가 다른 모든 선점된 임계구역의 상속된 우선순위보다 높아야 한다는 것이다[11][13]. 이 조건을 만족시키지 못하면 쓰레드 τ는 임계구역에 들어가는 것이 블럭되고 τ를 블럭시킨 쓰레드가 τ의 우선순위를 상속받는다[13].

우선순위 상한 프로토콜은 다음과 같이 정의 된다.

(1) 쓰레드 τ가 임계구역에 들어가기 전에 세마포어 S를 로킹해야 한다. 만일 τ의 우선순위가 τ-외의 다른 쓰레드에 의해 현재 사용되는 모든 세마포어의 가장 높은 우선순위 상한보다 높지 않은

경우 쓰레드 τ는 블럭될 것이고 세마포어 S에 대한 로킹은 거부된다.

(2) 쓰레드 τ는 임계구역에서 수행되며 그보다 우선순위가 높은 쓰레드를 블럭시키지 않으면 지정된 우선순위로 수행한다. 만약 쓰레드 τ가 우선순위가 높은 쓰레드를 블럭시키면 그에 의해 블럭된 쓰레드의 가장 높은 우선순위를 상속받는다. τ가 임계구역을 빠져나올때 임계구역에 들어갈 때의 우선순위로 되돌아간다. 우선순위 상속은 이행적(transitive)이다[11][13].

(3) 임계구역에 들어가려고 시도하지 않는 쓰레드 τ는 우선순위가 수행되고 있는 쓰레드보다 우선순위가 높은 경우는 그 쓰레드를 선점할 수 있다.

이 우선순위 상한 프로토콜은 우선순위 상속으로 인한 기존의 직접 블럭킹과 간접 블럭킹(push-through blocking)에 또다른 형태의 블럭킹을 유발시킨다. 이 블럭킹은 세마포어 S_0는 사용중이 아니지만 수행중인 쓰레드의 우선순위가 다른 세마포어의 우선순위 상한보다 높지않기 때문에 사용하지 못하게 되는 것으로 'ceiling blocking'이라 한다[13]. 이것은 교착상태와 연속된 블럭킹의 방지에 필요하다. 그러나 최악의 블럭킹 지연시간이 개선된다.

4. 다중처리기 시스템 동기화

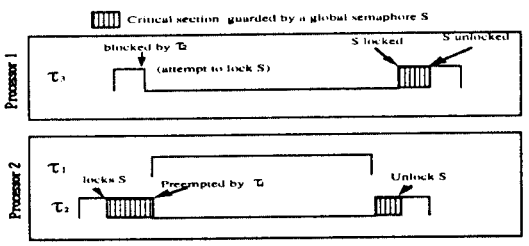
4.1 요구조건

다중처리기 시스템에서 동기화를 이루기 위해서는 'Test & Set', 'Fetch-and-Add'와 같은 기본적인 명령어(basic instruction)들이 존재해야 한다[1][2][4][5]. 이 기본적인 명령어들은 상호 협력하는 프로세스들간의 동기화나 통신을 위해 사용되며 대개 특수 목적의 하드웨어에 의해 지원된다[2].

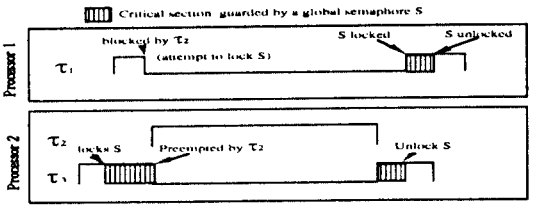
공유메모리 다중처리기 시스템은 대개 메모리내의 공유된 데이타를 이용하여 동기화를 이룬다. 또한 동기화 프로토콜은 여러개의 프로세서에서 여러개의 프로세스나 쓰레드가 동시에 처리되기 때문에 이진 세마포어를 사용한 'busy-waiting' 프로토콜을 주로 사용한다. 이 프로토콜의 장점은 동기화 변수의 변화를 빨리 인식한다는 장점이 있다. 이 세마포어의 로킹과 해제는 기본적으로 제공하는 기본 명령어들을 이용하여 원자적으로 수행된다. 이 원자적 연산(atomic operation)이 제공되지 않으면 시스템 전체의 동기화를 이룰 수 없게 된다.

4.2 원거리 블럭킹

쓰레드들이 여러개의 프로세서에서 수행이 될때 블럭킹 지연이 더 길어질 수 있다[10][11]. 단일 프로세서에서의 블럭킹 개념이 여러개의 프로세서 환경에서는 자원의 공유로 인한 개념의 확장이 필요하다. 다중처리기 시스템에서 쓰레드들이 그들의 제한시간을 만족시키는가에 대한 결정은 자신의 프로세서에서 발생하는 선점과 블럭킹 뿐만 아니라 다른 프로세서에서의 여러 쓰레드들에 의해 발생하는 대기시간을 고려해야 한다. 이것을 원거리 블럭킹(remote blocking)이라 한다[10]. 이러한 원거리 블럭킹이 최소화되어야 한다. 원거리 블럭킹의 예는 다음과 같다.



(a) 원거리 블럭킹의 예 1



(b) 원거리 블럭킹의 예 2

그림 2. 원거리 블럭킹의 예

위의 그림(a)에서는 τ_1이 세마포어S에 의해 시작되는 임계구역을 수행하려고 한 그보다 우선순위가 낮은 τ_3가 그 부분을 수행하고 있으므로 블럭된다. 그 도중

τ_2는 임계구역과 상관없는 부분을 수행하므로 τ_3를 선점하에 수행한다. 이때 τ_1는 τ_3에 의해서만 블럭되어야 하나, τ_2에 의해서 블럭되는 것과 같다. (b) 에서는 τ_3가 τ_1에 의해 블럭되는 것으로 (a) 보다는 문제가 덜 심각하다.

위의 예에서 보면 원거리 블럭킹은 단일처리기 시스템에서의 블럭킹과는 많은 차이가 있다는 것을 알 수 있다. 블럭킹이 세마포어가 없는 상황에 대해서도 쓰레드가 기다려야 하는 경우에 대해서도 발생할 수 있다. 따라서 다중처리기 시스템에서의 동기화 프로토콜은 쓰레드의 원거리 블럭킹의 시간이 다른 쓰레드의 임계구역에 한정되어야 하고 비임계구역에 대한 자연이 영향을 받지 않아야 한다[10].

4.3 광역 우선순위 상한과 지역 우선순위 상한

시스템 전체에서 쓰레드에 할당된 가장 높은 우선순위를 P_H라 하자. 단일처리기 시스템에서 우선순위 상한 프로토콜에서는 P_H보다 높은 우선순위에서 수행될 임계구역이 존재하지 않는다. 따라서 모든 지역 세마포어의 우선순위 상한은 P_H보다 낮거나 같다. 그러나 광역 임계구역은 지정된 우선순위보다 더 높은 우선순위를 가지고 수행될 필요가 있을때는 P_H보다 더 높아야 한다. 이러한 조건은 지역 임계구역에 대해서는 해당되지 않는다. 따라서 지역 임계구역에 대한 우선순위 상한과 광역 임계구역 상한의 두가지 우선순위 상한을 정의해야 한다.

정의 : 지역 세마포어(local semaphore) S의 우선순위 상한은 S를 사용하는 태스크중에서 가장 높은 우선순위로 정의된다.

정의 : 세마포어 S_i를 액세스 하는 태스크의 가장 높은 우선순위를 PS_i라 하자. 그러면 광역 세마포어 S_{Gi}의 우선순위 상한은 다음과 같이 정의된다.
- S_{Gi}의 우선순위 상한은 P_H보다 높아야 한다.
- 만약 S_{Gi}와 S_{Gi}가 광역 세마포어이고 $PS_i > PS_i$ 이면 S_{Gi}의 우선순위 상한은 S_{Gi}의 우선순위 상한보다 높아야 한다.

이러한 조건으로 광역 세마포어의 기본 우선순위 상한(base priority ceiling) P_G를 P_H보다 더 높게 정의한다. 즉 $P_G > P_H$이다. 그러면 광역 세마포어의 우선순위 상한은 $P_G + PS_i$로 주어질 수 있다.

4.4 광역 임계구역 실행 우선순위

GCS의 우선순위 조건은 다음과 같다.
- 블럭킹 지연은 임계구역에 대한 지연이 되어야 한다.
- 만약 τ_j가 τ_i를 블럭시키면 τ_j는 블럭킹 지연시간 동안 τ_i의 우선순위를 포함해야 한다.

GCS내에서 수행중인 쓰레드는 원거리 쓰레드를 블럭시킬때만 우선순위가 P_H보다 높아야 한다. 그러나 우선순위의 변화는 다중처리기 환경에서 구현하기가 힘들다. 그러므로 GCS에서 수행중인 쓰레드는 항상 P_H와 우선순위 상한이하의 우선순위를 가지도록 한다. 이것을 우선순위 할당규칙이라 한다[10].

5. 내포된 GCS 동기화 알고리즘 설계

5.1 시스템 모델

본 논문에서 제시하는 동기화는 공유메모리 다중처리기 시스템(shared-memory multiprocessor system)이다. 이들의 구성은 아래의 그림 3과 같다.

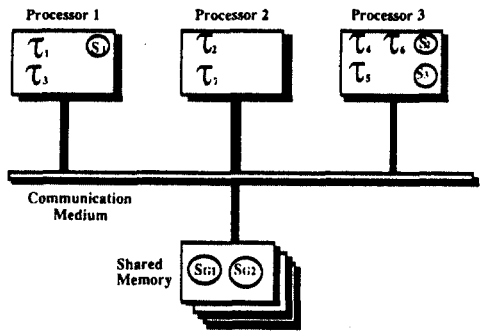


그림 3. 시스템 모델

여러개의 프로세서와 공유메모리 모듈은 버스에 의해 서로 연결되어 있으며 이 버스에 의한 지연은 고려하지 않는다. 각 프로세서는 각자의 지연 메모리와 캐쉬를 가진다[2]. 이 지역 메모리는 그 프로세서에 할당된 쓰레드와 데이타를 가지고 있으며, 캐쉬는 공유

데이타를 가지고 있으며 캐쉬 일관성 문제는 고려하지 않는다.

5.2 기존 연구의 문제점

Ragunathan Rajkumar[10] 에 의해 제시된 다중처리기 동기화 방법은 광역 임계구역이 내포되지 않은 환경에서의 동기화로서 이를 광역 임계구역이 내포되지 않은 환경에서의 동기화로서 이들 광역 임계구역이 내포된 경우는 교착상태가 발생할 수 있다. 이 교착상태가 발생하는 예제를 보자.

예제 :

$\tau_1 = \{\cdots, P(S_1), \cdots, P(S_{G1}), \cdots, V(S_{G1}), \cdots, V(S_1), \cdots\}$

$\tau_2 = \{\cdots, P(S_{G1}), \cdots, P(S_{G2}), \cdots, V(S_{G2}), \cdots, V(S_{G1}), \cdots\}$

$\tau_3 = \{\cdots, P(S_{G1}), \cdots, P(S_1), \cdots, V(S_1), \cdots, V(S_{G1}), \cdots\}$

$\tau_4 = \{\cdots, P(S_2), \cdots, V(S_2), \cdots, P(S_{G1}), \cdots, V(S_{G1}), \cdots, P(S_2), \cdots, V(S_2), \cdots\}$

$\tau_5 = \{\cdots, P(S_3), \cdots, V(S_3), \cdots, P(S_{G2}), \cdots, P(S_{G1}), \cdots, V(S_{G1}), \cdots, V(S_{G2}), \cdots\}$

$\tau_6 = \{\cdots, P(S_2), \cdots, V(S_2), \cdots, P(S_3), \cdots, V(S_3), \cdots\}$

$\tau_7 = \{\cdots, P(S_{G2}), \cdots, V(S_{G2}), \cdots\}$

표 1. 예제의 각 세마포어의 우선순위

Semaphore	Priority Ceiling
S_1 (local)	$P(\tau_1)$
S_2 (local)	$P(\tau_4)$
S_3 (local)	$P(\tau_5)$
S_{G1} (global)	$P_G + P(\tau_1)$
S_{G2} (global)	$P_G + P(\tau_2)$

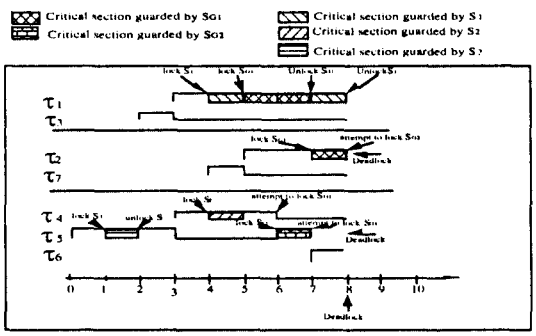


그림 4. 예제의 쓰레드의 수행순서

표 2. 예제의 임계구역의 실행 우선순위

thread	critical section Guarded by	Normal priority
τ_1	S_1	$P(\tau_1)$
	S_{G1}	$P(\tau_2)$
τ_2	S_{G1}	$P_G + P(\tau_1)$
	S_{G2}	$P_G + P(\tau_5)$
τ_3	S_1	$P(\tau_1)$
	S_{G1}	$P_G + P(\tau_2)$
τ_4	S_2	$P(\tau_5)$
	S_{G1}	$P_G + P(\tau_1)$
τ_5	S_3	$P(\tau_5)$
	S_{G1}	$P_G + P(\tau_1)$
	S_{G2}	$P_G + P(\tau_2)$
τ_6	S_2	$P(\tau_4)$
	S_3	$P(\tau_5)$
τ_7	S_{G2}	$P_G + P(\tau_5)$

위의 예제에 대한 수행과정은 [그림 4]와 같으며 교착상태가 발생한다.

5.3 내포된 GCS의 동기화

GCS가 GCS에 내포된 경우에 대한 동기화는 앞절에서의 예와 같이 교착상태의 문제가 발생한다. 이러한 문제를 해결하기 위하여 교착상태 발생조건 중 'hold and wait'의 조건을 만족시키지 않도록 하여 내포된 임계구역을 실행할 때 사용되는 세마포어를 한꺼번에 로킹한다. 여러개의 세마포어를 한번에 로킹하기 위해서는 세마포어를 로킹하는 부분을 또다른 하나의 임계구역으로 설정해야 한다. 광역세마포어를 로킹하는 부분을 임계구역으로 설정해서 광역세마포어를 동시에 로킹하도록 하는 것이다. 이러한 방법을 이용하여 내포된 GCS의 동기화를 이루기 위해서 새로운 광역 이진 세마포어(global binary semaphore)를 도입하여 광역 임계구역에 사용되는 세마포어의 록을 임계구역으로 설정한다. 또한 내포된 임계구역의 우선순위 상한은 기존의 방법에서는 P_G에 그 세마포어가 사용되는 쓰레드의 우선순위를 더한 값이 되었으나 내포된 GCS에서는 그보다 높게 설정하여 가장 우선적으로 수행이 되도록 한다. 실행우선순위를 더 높게 설정하기 위해 내포된 두 임계구역의 우선순위를 합산하여 내포된 임계구역의 우선순위 상한으로 설정한다. 앞의 예제에서의 내포된 임계구역의 수행을 시

작할 때 다음과 같은 연산을 수행하도록 한다.

P(new)
  lock S_{G1}
  lock S_{G2}
  execution priority = $P_G + P(S_{G1}) + P_G + P(S_{G2})$
  $= 2P_G + P(S_{G1}) + P(S_{G2})$
  V(new)

이렇게 함으로써 내포된 임계구역을 동기화 시키는데 있어서 교착상태 발생의 원인이 되는 'hold and wait' 발생조건이 만족시키지 않게 되어 교착상태가 발생하지 않게 된다. 우선순위 상한을 더 높게 주는 것은 다른 임계구역보다 먼저 수행되게 하기 위함이다.

만일 첫번째 세마포어 S_{G1}을 로킹하였으나 두번째 세마포어 S_{G2}를 로킹하지 못하면 이미 획득한 세마포어에 대한 로킹을 해제시키고 다시 시도하게 한다. 이렇게 하여 두개의 세마포어를 로킹하지 못하면 해당 임계구역을 수행하지 못하게 하며, 두 세마포어를 로킹한 경우, 실행 우선순위 상한을 설정하고 해당임계구역을 수행하도록 한다. 이러한 방법을 앞절의 예제에 적용하면 다음의 순서로 수행한다.

- 시간 t_0에 프로세서 3에서 τ_5가 시작되어 수행을 시작한다.
- 시간 t_1에 프로세서 3에서 τ_5가 지역 세마포어 S_3를 로킹하고 임계구역을 수행한다.
- 시간 t_2에 프로세서 3에서 τ_5가 임계구역의 수행을 끝마치고 세마포어 S_3를 해제시킨다. 프로세서 1에서는 τ_3의 수행이 시작된다.
- 시간 t_3에 프로세서 1에서는 τ_1이 시작되어 τ_3를 선점하고 수행을 시작한다. 프로세서 3에서는 τ_4가 시작되어 τ_5를 선점하고 수행을 시작한다.
- 시간 t_4에 프로세서 1에서는 τ_1이 지역 세마포어 S_1을 로킹하고 임계구역을 수행한다. 프로세서 3에서는 τ_4가 지역 세마포어 S_2를 로킹하고 임계구역을 수행한다.
- 시간 t_5에 프로세서 1에서는 τ_1이 광역 세마포어 S_{G1}을 로킹하고 광역 임계구역을 수행한다. 프로세서 3에서는 τ_4가 임계구역의 수행을 끝내고 S_2를 해체시킨다. 프로세서 2에서는 τ_2가 시작된다.
- 시간 t_6에 프로세서 1에서는 τ_1이 광역 임계구역의 수행을 계속하고 프로세서 2에서는 τ_2가 비임계구역을 계속 수행한다. 프로세서 3에서는 쓰레드 τ_4가 광역 세마포어 S_{G1}을 로킹하려고 시도하나 실패하여 블럭되고 τ_5가 깨어나 내포된 광역 임계구역에 대한 세마포어 S_{G2}과 S_{G1}을 로킹하려고 시도하나 S_{G1}을 로킹하지 못하여 S_{G2}를 해제하고 블럭된다.
- 시간 t_7에 프로세서 1에서는 S_{G1}을 해제시키고 지역 임계구역을 수행하며 프로세서 3에서는 τ_4가 S_{G1}을 로킹하고 광역 임계구역을 수행한다. 프로세서 2에서는 τ_2가 S_{G1}과 S_{G2}에 대한 로킹을 시도하나 실패하여 블럭된다.
- 시간 t_8에 프로세서 1에서는 τ_1이 수행을 끝마치고 프로세서 2에서는 τ_2가 S_{G1}과 S_{G2}를 로킹하고 내포된 광역 임계구역을 수행한다. 프로세서 3에서는 τ_4가 임계구역의 수행을 끝마치고 비임계구역의 수행을 시작한다.
- 시간 t_9에서 프로세서 1에서는 τ_3이 제시작되고, 프로세서 2에서는 τ_2가 내포된 광역 임계구역을 계속 수행한다. 프로세서 3에서는 τ_4가 지역 세마포어 S_2를 로킹하고 임계구역을 수행한다.

이와 같은 방법으로 앞절의 예제에서 발생한 교착상태를 해결할 수 있다. 이러한 시간적인 수행과정은 아래의 그림 5와 같다.

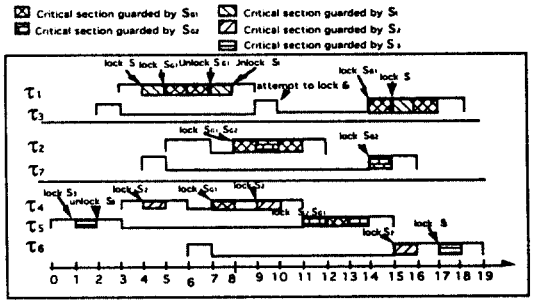


그림 5. 내포된 광역 임계구역의 동기화 순서

6. 성능 분석

위에서 제시한 동기화 방법을 이용하여 동기화 시킬 경우에 대한 각 클래스별 제한시간 실패율을 썬 호환 기종에서 C언어와 이용이 가능한 SMPL을 이용하여

위에서 제시한 알고리즘을 시뮬레이션으로 알아보았다.

6.1 시뮬레이션 모델 및 패러미터

시뮬레이션의 대상이 되는 각 쓰레드는 대기 큐에 각 클래스 별로 일정한 주기를 가지고 도착하고 주기가 가장 짧은 클레스 1이 우선순위가 가장 높고 주기가 가장 긴 클래스 5가 가장 낮은 우선순위를 가지도록 하는 rate-monotonic 알고리즘에 따르도록 하였다.

시뮬레이션 모델은 하나의 대기 큐를 가지고 4개의 프로세서가 밀결합 형태로 연결된 시스템이고 M/M/1특성을 가진다. 도착하는 쓰레드는 쓰레드 ID, 도착시간, 주기, 클레스, 우선순위, 서비스시간을 가진다. 도착하는 쓰레드는 도착즉시 부하가 가장 낮은 CPU에 스케쥴링되어 해당 CPU에서 수행된다.

시뮬레이션에 사용되는 패러미터는 다음과 같다.

표 3. 시뮬레이션에 사용되는 패러미터 값

총 노드 수	4
쓰레드의 클래스	5
광역 임계구역	2
스케쥴링 방법	우선순위 위주
각 클래스의 Interarrival Time	30, 60, 90, 120, 150
서비스 시간	Interarrinal Time의 33%, 50%, 67%, 75%, 90%

6.2 결과분석

아래의 그림들은 평균 서비스 시간이 발생주기의 50%일때와 90%일때의 제한시간 실패율을 각 클래스 별로 나타낸 것이다.

아래의 그림 8은 평균 서비스 시간이 발생주기의 33%에서 90%로 변해 갈때 클레스별 실패율을 나타낸 것이다.

아래의 그림 9는 평균 서비스 시간이 발생주기의 33%에서 90%로 변해갈때 각 CPU 효율의 변화를 나타낸 것이다.

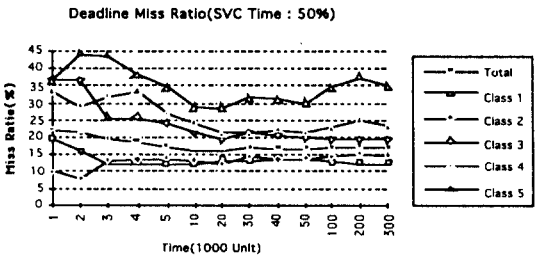


그림 6. 주기의 50%일때 제한시간 실패율

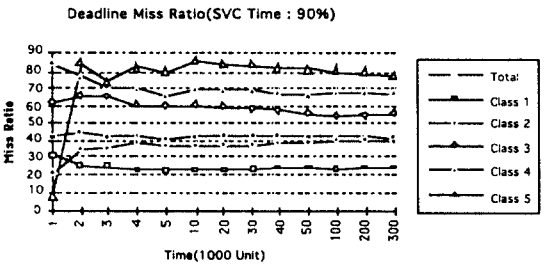


그림 7. 주기의 90%일때 제한시간 실패율

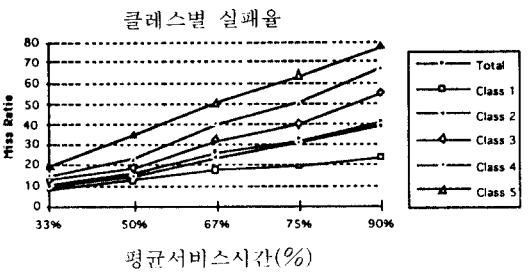


그림 8. 변화에 따른 클래스별 실패율

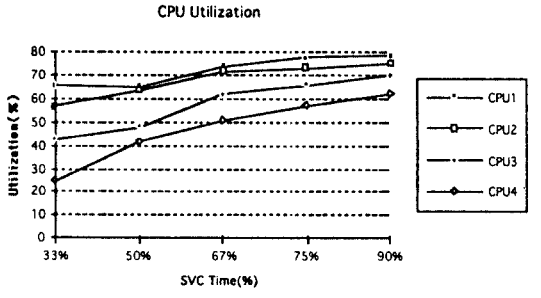


그림 9. 변화에 따른 CPU 효율

시뮬레이션 결과 나타나는 특징을 살펴보면 평균서

비스 시간이 주기의 33%일때에는 시스템 전체의 부하가 높지 않아서 우선순위가 낮은 클래스의 쓰레드도 제한시간을 만족시키는 비율이 높고 실패하는 경우는 20%정도에 지나지 않는다. 그러나 서비스 시간이 50%로 증가하는 경우 클래스 5의 실패율은 35%정도에 이른다. 그러나 클래스 1의 실패율은 크게 증가하지 않는다. 서비스 시간이 더 증가하여 90%에 이르면 클래스 5는 실패율이 80%에 이른다. 그러나 클래스 1의 실패율은 20%에서 크게 증가하지 않으며 교착상태는 발생하지 않는다.

시뮬레이션의 결과를 보면 각 클래스의 평균 서비스 시간이 증가하더라도 우선순위가 높은 클래스 1은 제한시간 실패율이 크게 증가하지 않는 반면 클래스 5의 실패율은 급격히 증가하는 것을 볼 수 있다. CPU의 효율의 변화를 보면 같은 종류의 CPU를 사용하더라도 각 CPU의 효율이 다르게 나타나는 것을 볼 수 있다. 이것은 주로 할당하는 쓰레드의 클래스가 어느 클래스인가에 따라 달라진다.

7. 결 론

선점(Preemption)이 허용되는 범용 실시간 시스템을 설계하는 데 있어서 그 성능은 동가화의 방법에 따라 크게 좌우된다[1][4]. 다양한 우선순위에 대해 수행을 하기위해 우선순위 위주의 스케쥴링을 할때 선점과 블럭킹이 발생하게 된다. 이 선점과 블럭킹의 결과 교착상태가 발생할 수 있고, 제한시간을 만족시키지 못하는 경우가 발생한다[6][10]. 따라서 임계구역의 수행시간을 가능한 단축시켜야 하고 필수적인 경우에만 임계구역으로 설정하는 것이 필요하다.

단일처리기의 실시간 시스템에서의 동기화는 우선순위 상한으로 가능하며 교착상태가 발생하지 않는다. 광역 임계구역이 내포되지 않는 경우, 다중처리기에서의 동기화는 광역 우선순위 상한 프로토콜로써 가능하다. 그러나 광역 임계구역이 내포된 경우에는 교착상태가 발생한다.

본 논문에서는 다중처리기 시스템에서 광역 임계구역이 내포되는 것을 허용하는 환경에서의 실시간 동기화 방법과 알고리즘을 제시하였다. 이 방법은 교착상태를 유발하는 네가지 조건중에서 'hold and wait' 의 조건이 되지 않도록 하였다. 이 방법은 다중처리기 시스템에서 교착상태 발생의 원인이 되는 광역임계구역을 수행할 때 광역 세마포어의 로킹을 한번에 수행하도록 하는 방법으로 교착상태를 해결하였다. 또한 주기에 대한 평균 서비스 시간이 증가하여 시스템 전체 부하가 증가하더라도 우선순위가 높은 쓰레드의 제한시간 실패율을 어느 정도 보장할 수 있도록 하였다.

쓰레드 수준의 스케쥴링에서 성능은 프로세서를 쓰레드로 분할하는 과정에서 얼마나 적절히 효과적으로 분할할 수 있는가에 크게 좌우된다.

참 고 문 헌

1. Anne Dining, "A Survey of Synchronization Methods for Parallel Computers," IEEE Computer, Vol.22, No.7, pp.66-77, 1989.
2. Michel Dubois and Christoph Scheurich, "Synchronization, Coherence, and Event Prdering in Multiprocessors," IEEE Computer, Vol.21, No.2, pp.9-21, 1988.
3. Borko Furht et al., "Real-Time UNIX Systems : Design And Application Guide," Kluwer Academic Publishers, pp.1-33, 1991.
4. Danial D. Gajski, Jih-Kwon Peir, "Essential Issues in Multiprocessor Systems," IEEE Computer, Vol.18, No.6, pp.9-27, 1985.
5. Gary Graunke and Shreekant Thakker, "Synchronization Algorithms for Shared-Memory Multiprocessors," IEEE Computer, Vol.23, No.6, pp.60-69, 1990.
6. Mark Heuser, "An Implementation of Real-Time Thread Synchronization," USENIX Summer Conference, Anaheim, CA, pp.97-105, 1990.
7. J. Lehoczky, L Sha and Y. Ding, "The Rate Monotonic Scheduling Algorithm : Exact Characteration And Average Case Behavior, "Proc. of the IEEE Real-Time Systems Symposium," IEEE Computer Society press, pp. 199-171, 1989.

8. J. L. Peterson and A. Siberschatz, "Operating System Concepts 2nd ed.," Addison-wesley Press, 1988.
9. Mike L. Powell, S. R. Kleiman et al., "SunOS Multi-Thread Architecture," The SPARC Technical Papers, pp.339-372, Springer-Verlag, 1991.
10. Ragunathan Rajkumer, "Real-Time Synchronization Protocols for Shared Memory Multiprocessors," Proc of the 10th International Conference of Distributed Computing Systems, IEEE Computer Society Press, pp. 116-123, 1991.
11. Ragunathan Rajkumar, "Synchronization in real-time systems : A priority inheritance approach," Kluwer Academic Publishers, 1991.
12. Karsten Schwan, Hongyi Zhou and Ahmed Gheith, "Real-Time Thread, Operating System Review, AcM Press, Vol.26, No.1, pp.35-46, 1992.
13. Lui Sha, Ragunathan Rajkumar and John P. Lehoczky, "Priority Inheritance Protocols : An Approach to Real-Time Synchronization," IEEE Trans. on Computers, Vol.39, No.9, pp. 1175-1185, 1990.

朴 政 亨 정회원

1991年 2月 : 연세대학교 전산과학과 졸업(학사)
1993年 2月 : 연세대학교 전산과학과 졸업(석사)
1993年 2月 - 현재 : 삼성종합기술원 근무

※ 관심분야 : Realtime OS /processing, Multiprocessor OS