공유메모리 다중 프로세서 실시간 시스템에서의 동기화 프로토콜

강 승엽* , 하 란
홍익대학교 전자계산학과

An Extended Real-Time Synchronization Protocols for Shared Memory Multiprocessors

Seung-yup Kang, Rhan Ha
Dept. of Computer Science, Hongik University

요약

작업들이 자원을 공유하는 경우 예측하기 어려운 지연시간이 발생한다 다중 프로세서 시스템에서의 자원공유로 인한 지연시간은 더욱 예측하기 어렵다 실시간 시스템의 스케줄 가능성 검사를 위해서는 이러한 지연시간을 정확히 예측해야 한다 선점가능한 우선순위 구동 CPU 스케줄링 알고리즘에 의해서 다른 우선순위의 작업과의 동기화는 우선순위 역전 문제를 야기한다. 본 논문에서는 다중 프로세서에서의 동기화 프로토콜을 제안하고 직업의 지연시간을 분석한다 다른 프로세서에 할당된 작업들이 수행중인 자원을 요구할 때, 자원을 수행하는 직업의 우선순위를 높여줌으로써 자원수행을 빠르게 종료하게 한다. 이로 인해 자원에 의한 지연을 최소화한다. 특히, 높은 우선순위의 작업의 경우 더욱 작은 지연시간을 갖게 한다 시뮬레이션을 통한 Shared Memory Protocol[5]과의 비교, 분석 결과 성능의 향상을 보임을 알 수 있다. 다양한 작업집합에 대한 지연시간을 분석하였다.

1. 서 론

다중 프로세서 시스템에서 작업들은 많은 자원을 공유하게 된다 그러나 배타적으로 공유되는 자원들은 두 개의 작업이 동시에 같은 자원을 사용하지 못하므로 실시간 스케줄링 기법은 동기화를 필요로 하며 자원의 동기화는 작업 수행을 지연시킨다. 선점가능한 우선순위 구동 CPU 스케줄링 알고리즘의 경우 다른 우선순위 작업과의 동기화는 우선순위 역전 문제를 야기한다 즉, 낮은 우선순위 작업이 높은 우선순위 작업을 기다리게 하면서 먼저 수행되는 경우가 발생하는 것이다 결정성 실시간 작업의 스케줄 가능성을 체크하기 위해서는 동기화에 의해 발생되는 작업의 지연시간을 고려해야 한다. 그러므로, 실시간 동기화의 목적은 각 작업의 최악경우 지연시간을 최소화하는 것이다. 특히, 높은 우선순위 작업의 지연시간을 최소화해야 한다.

본 논문은 선점가능한 우선순위 구동 스케줄링 알고리즘 환경에서 실시간 동기화 문제를 고찰한다. 특히, 각 프로세서가 메모리를 공유하고 작업들이 프로세서에 정적으로 할당되어 있는 분산 또는 병렬 시스템에 대해 논한다

본 논문의 구성은 2절에서는 기존에 연구된 실시간 동기화 프로토콜에 대해 설명하며, 3절에서는 단일 프로세서에서는 발생하지 않고 다중 프로세서에서 발생하는 동기화에 따른 지연시간에 대해 설명한다. 4절에서는 제안된 프로토콜을 설명하고 5절에서는 지연시간의 분석에 대해 논하며 마지막 6절에서는 정리와 향후 연구에 대해 논한다

2. 관련 연구

실시간 시스템의 스케줄가능성을 분석하기 위해서는 자원공유로 발생하는 지연시간을 정확하게 분석하는 것이 필요하다[1,2]. 그리고 이러한 지연시간을 최소로 하는 것은 더 많은 작업을 스케줄 가능하게 하므로 중요하다

Priority Ceiling Protocol(이하 PCP)[3]은 단일 프로세서에서의 동기화를 해결하고 있으며 다중 프로세서 시스템으로 확장된 개념으로는 Multi-processor Priority Ceiling Protocol(이후 MPCP)[4], Shared Memory Protocol(이후 SMP)[5]과 Scalable Real-time Synchronization Protocol(이후 SRSP)[6]이 있다 MPCP의 경우 다른 프로세서에 있는 작업들이 같은 자원을 요구하는 경우 이 자원의 수행은 동기화 프로세서라는 특별한 프로세서에서 수행하게 함으로써 공유되는 자원으로 인한 지연을 최소화한다. 그러나 공유되는 자원이 많은 경우 병목현상에 의한 수행의 지연을 유발한다 SMP의 경우 자원의 수행이 자신의 프로세서에서 수행이 이루어지는 장점이 있으나 다른 프로세서와 공유되는 자원의 수행시 무조건 높은 우선순위로 높여 줌으로써 자원을 사용하지 않는 높은 우선순위의 작업에게 또 다른 지연을 발생시킨다 SRSP는 모든 자원을 수행하기 전에 예약함으로써 내포된 자원(nested resource)의 수행을 가능하게 하였으나 자원을 수행하지 않는 작업의 지연시간에 대한 고려가 부족하다

3. 다중 프로세서에서의 동기화

다중 프로세서에서 자원은 동일한 프로세서에 할당된 작업들만이 요구하는 자원과 서로 다른 프로세서에 할당된 작업들이 요구하는 자원으로 나누어질 수 있다 모든 자원이 동일한 프로세서에 할당된 작업들에 의해서만 요구되는 경우라면 프로세서들을 각각 하나의 단일 프로세서로서 고려하여 각각에 PCP를 적용하면 된다 그러나, 다른 프로세서에 할당된 작업들이 요구하는 자원이 있을 때는 원격 블로킹이라는 새로운 지연을 발생시킨다. J_k는 작업을 나타내며 k의 값이 작을수록 우선순위가 높은 작업을 나타낸다

원격 블로킹(Remote Blocking)의 개념

서로 다른 프로세서에 할당된 작업들이 요구하는 자원들의 공유로 인해 다중 프로세서 시스템에서는 단일 프로세서에서는 발생하지 않는 원격 블로킹이라는 새로운 지연이 발생하게 된다 그림 1의 경우가 대표적인 원격 블로킹의 예이다 J_1은 J_3과 함께 공유하고 있는 자원을 수행하기 위해 프로세서 2에 있는 자신보다 낮은 우선순위의 J_3의 수행이 완료될 때까지 기다려야 한다 또 다른 원격 블로킹의 예는 그림 2의 경우이다 그림 2의 경우 우선순위 역전은 발생하지 않지만 J_3이 프로세서 2에 있는 작업들에 의해 오랜 시간동안 지연된다 즉, 다른 프로세서에 있는 작업에 의해 오랜 시간 지연을 당하게 되는 것이다. 이것은 프로세서당 스케줄가능성의 분석을 어렵게 만든다 다른 프로세서의 작업들에 의해 공유되는 자원(이후 전역자원)의 수행이 시스템에 있는 모든 작업들의 우선순위보다 높은 우선순위로 이루어지도록 함으로써 이러한 원격 블로킹은 해결할 수 있다. 즉, 그림 1의 경우 J_3의 자원 수행이 J_2보다 높은 우선순위로 이루어지므로 J_2에 의해 선점되지 않는다 그러므로 J_1은 J_2의 수행에 의해 지연되지 않는다. 그림 2의 경우 J_2의 자원 수행이 J_0보다 높은 우선순위로 이루어지므로 J_3는 J_0, J_1에 의해 선점되지 않으므로 J_3은 다른 프로세서의 작업에 의해 지연되지 않는다.

본 논문에서 사용하는 용어는 다음과 같이 정의된다

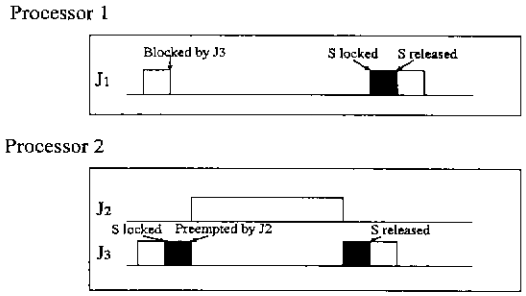

그림 1 : 원격 블로킹

- P_지연시간 자신과 같은 프로세서에 있는 자신보다 높은 우선순위를 갖는 작업에 의한 선점시간을 제외한 모든 지연시간 즉, 낮은 우선순위의 작업에 의한 지연과 다른 프로세서에 있는 작업들에 의한 지연을 의미한다
- 전역자원 : 서로 다른 프로세서에 할당된 작업들이 요구하는 자원.
- 지역자원 : 동일한 프로세서에 할당된 작업들만이 요구하는 자원. 다른 프로세서에 할당된 작업들은 이 자원을 요구하지 않는다
- 실링(ceiling) 자원에게 부여되는 값으로 지역자원의 경우 그 자원을 요구하는 작업들 중 가장 높은 우선순위를 갖는 작업의 우선순위를 값으로 하고, 전역자원의 경우는 시스템에서 가장 높은 우선순위보다 하나 더 높은 우선순위를 값으로 한다
- H_우선순위 : 시스템에 있는 모든 작업들의 우선순위보다 높은 우선순위이다. (SMP의 경우, normal priority를 의미하며 작업에 따라 각각 다른 우선순위가 부여된다[4] 본 논문에서 제안된 프로토콜에서는 전역자원의 실링을 나타내며 일정한 값을 갖는다)
- 충돌 : 어떤 작업이 수행하고 있는 자원을 다른 프로세서에 있는 작업이 요구하는 경우, 또는 두 개의 작업이 동시에 같은 자원을 요구하는 경우에 충돌이 발생했다고 한다.

4. 프로토콜

진역자원의 수행이 작업들에게 할당된 모든 우선순위보다 더 높은 H_우선순위로 이루어진다면 자원 수행시에 다른 작업에 의해 선점되지 않기 때문에 원격 블로킹은 해결될 수 있다 그러나, SMP의 경우와 같이 전역자원의 수행에 무조건 H_우선순위를 부여하는 것은 전역자원을 사용하지 않는 작업에게 다음과 같은 또 다른 지연을 초래하게 된다

첫째, 충돌이 발생하지 않은 상태라도 무조건 전역자원의 수행이 H_우선순위로 이루어지기 때문에, 원격 블로킹이 발생하지 않은 경우에도 높은 우선순위 작업은 낮은 우선순위 작업에 의해 불필요하게 지연된다. 즉, 충돌이 발생하지 않은 경우에도 우선순위 역진이 발생한다. 그림 1에서 프로세서 1의 J_1이 릴리즈되지 않은 상태라면 프로세서 2의 J_2는 J_3의 자원수행에 의해 지연될 필요가 없다

예제 1 : 같은 프로세서에 있는 작업 J_1, J_2, J_3이 각각 다른 프로세서에 있는 작업이 수행중인 전역자원 r, s, t를 요구하여 블록되어 있는 경우에 J_0이 릴리즈 되어 수행중이라면, J_0은 J_1, J_2

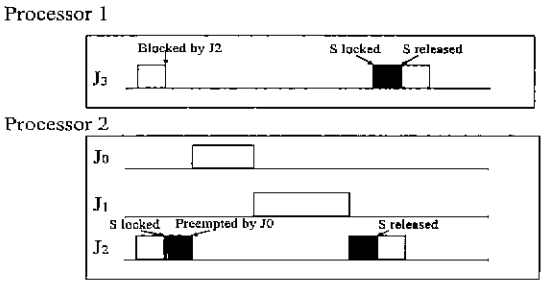

그림 2 원격 블로킹

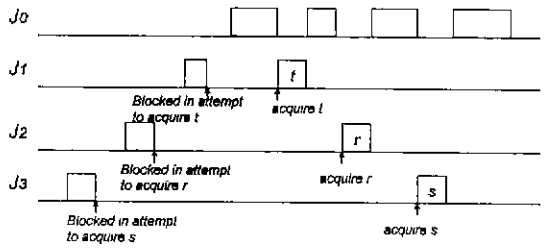

그림 3 : 예제 1의 스케줄

J_3의 전역자원이 사용가능해질 때마다 J_1, J_2, J_3의 전역자원수행에 의해 지연된다. (그림 3 참조)

둘째, 예제 1을 나타내는 그림 3에서와 같이 전역자원을 수행하지 않는 작업은 여러 개의 낮은 우선순위를 갖는 작업들에 의해 계속해서 지연될 수 있다 즉, 최악의 경우, 자신과 같은 프로세서에 할당된 자신보다 낮은 우선순위를 갖는 모든 작업들의 전역자원 수행시간 동안 지연될 수 있다.

본 논문에서는 이러한 무조건적인 우선순위의 상승으로 인해 발생되는 지연시간을 줄임으로서 각 작업의 응답시간을 줄인다 SMP와 같이 전역자원의 수행시 무조건 H_우선순위를 부여하지 않고 처음에는 자원의 수행이 할당된 원래 우선순위로 이루어지고 충돌이 발생하는 순간에 H_우선순위를 갖게 한다면 첫 번째의 문제는 해결될 수 있다. 즉, 충돌이 발생하지 않는 상태에서는 보다 높은 우선순위를 갖는 작업이 이를 선점할 수 있다.

제안하는 프로토콜은 다음과 같다.

1. 지역자원의 처리는 PCP[3]의 정책을 따른다
2. 전역자원의 실링은 시스템의 작업들에 할당된 가장 높은 우선순위보다 하나 더 높은 값을 취한다.
3. 전역자원의 수행은 그 자원을 수행하는 작업에 할당된 우선순위로 이루어지며 충돌이 발생할 경우에만 우선순위를 전역자원의 실링 값으로 높여준다. 자원의 수행이 끝나면 작업에게 처음 할당된 원래의 우선순위로 돌아간다
4. 어떤 작업의 우선순위가 같은 프로세서에서 수행이 시작되고 아직 종료되지 않은 자원의 실링보다 큰 경우에만 자신의 자원을 수행할 수 있다.
5. 자원의 수행을 시작하려 할 때, 자신의 원래 할당된 우선순위가 현재 프로세서에서 수행중인 작업의 우선순위보다 크지 않다면 선점할 수 없다

예제 1에서 보여지는 두 번째의 문제점은 프로토콜의 5번 항목에 의해서 해결이 가능하다. 즉, J_1, J_2, J_3의 할당된 우선순위가 J_0의 우선순위보다 낮기 때문에 J_0을 선점하여 자신의 진역자원을 수행할 수 없다. 그러므로 J_0은 낮은 우선순위를 갖는 J_1, J_2, J_3에 의해서 지연되지 않는다

보조정리 *작업 J_k가 전역자원을 수행하고 있는 동안에는 그 프로세서에서는 다른 어떠한 자원도 수행이 시작될 수 없다.*

정리 *자원을 수행하지 않는 작업이 자신보다 낮은 우선순위를 갖는 작업의 전역자원수행에 의해 지연되는 횟수는 최대한 한번이다.*

전술한 두 가지의 문제점은 본 논문에서 제안한 프로토콜의 정의와 정리에 의해 해결된다. 즉, 자원을 사용하지 않는 높은 우선순위의 작업에 대한 지연시간을 최소화한다

5. P_지연시간 분석 및 시뮬레이션

본 절에서는 작업이 수행을 종료하기까지의 지연되는 시간을 계산한다. P_지연시간은 같은 프로세서에 할당된 높은 우선순위의 작업에 의한 선점시간을 제외한 지연되는 모든 시간을 의미하며 지연요소들은 다음과 같다.

1 $H_{k,z}$: J_i와 다른 프로세서에 할당된 작업들 중 진역자원 k를 요구하는 J_i보다 높은 우선순위를 갖는 작업들의 자원수행시간의 합 진역자원을 수행하려 할 때, 다른 프로세서에 있는 높은

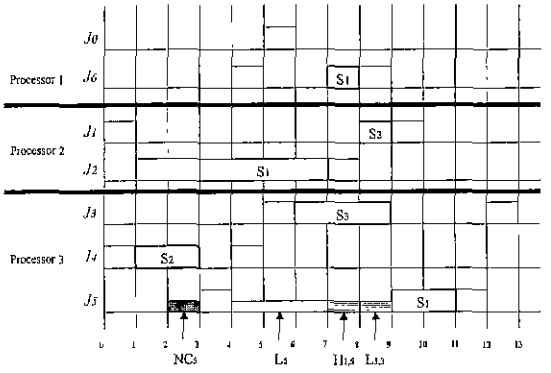

그림 4 지연시간 분석의 예제

우선순위의 작업이 그 전역자원을 사용하고 있는 경우이다. 자신과 같은 전역자원을 사용하는 다른 프로세서에 할당된 높은 우선순위의 작업들의 자원수행시간의 합으로 바운드할 수 있다

2. $L_{i,p}$ 작업 J_i와 같은 프로세서 p에 할당된 낮은 우선순위를 갖는 작업들의 자원수행시간(전역, 지역자원 모두)의 합 전역자원의 수행을 시작하려 할 때, 낮은 우선순위의 작업이 다른 전역자원 또는 지역자원을 사용하고 있는 경우이다.
3. L_i : J_i와 같은 전역자원을 요구하는 낮은 우선순위를 갖는 작업들의 자원수행시간중 가장 긴 자원수행시간. 전역자원을 수행하려 할 때, 낮은 우선순위의 작업이 그 전역자원을 사용하고 있는 경우이다. 이 경우, 그 낮은 우선순위의 작업의 전역자원 수행이 끝나면 바로 수행이 시작될 수 있으므로 같은 전역자원을 요구하는 낮은 우선순위의 작업들 중 가장 긴 자원수행시간으로 바운드 된다.
4. NC_i J_i와 같은 프로세서에 있는 낮은 우선순위의 작업들의 전역자원 수행시간 중 가장 긴 전역자원 수행시간. J_i가 자원을 수행하지 않는 경우 낮은 우선순위의 작업의 전역자원에 의해 선점당하는 경우이다. 낮은 우선순위의 작업들 중에서 가장 긴 전역자원 수행시간으로 바운드 될 수 있다.

열거한 지연요소들 모두 합한 것이 하나의 작업의 최악 경우 P_지연시간이 된다. 즉, J_i의 P_지연시간은 다음과 같다.

$$B_i = H_{k,i} + L_{i,p} + L_i + NC_i$$

그림 4는 P_지연시간에 대한 예제를 나타낸다. 3개의 프로세서와 7개의 작업 그리고, 3개의 자원을 사용하는 시스템에서 우선순위 5의 지연시간을 통해 지연요소를 설명한다.

- 1초에 J_4와 J_5은 자원1을 요구한다 이때, J_4의 우선순위가 더 크기 때문에 J_4가 자원1을 수행하게 된다. 충돌이 발생한 경우이기 때문에 J_4는 7(시스템에 있는 작업들의 우선순위보다 하나 더 큰 수가 실링 값이 된다 즉, 실링 값은 7이다.)을 우선순위로 사용하게 된다
- 2.5초에 릴리스된 J_5는 J_4가 실링 값으로 수행이 이루어지므로 지연된다. 이때 지연되는 지연요소는 NC_5이다.
- 3초에 J_4는 자원수행을 마치고 자신의 원래 우선순위인 4를 되찾게 된다. 그러므로, J_5는 J_4를 선점할 수 있다.
- 4초에 J_3는 자원2를 요구하는데 이는 프로세서 2에 있는 J_2가 수행을 하고 있기 때문에 지연된다. 이는 L_7이다.
- 7초에 J_2의 자원2 수행이 끝나지만 프로세서 1에 있는 J_5보다 높은 우선순위의 J_6이 자원2를 요구하고 있기 때문에 J_5는 다시 지연된다 이 경우는 $H_{1,5}$이다.
- 8초에 J_6의 자원2의 수행이 끝나지만 J_5와 같은 프로세서 3에 있는 J_3보다 낮은 J_2이 나른 공유자원 3을 수행하고 있기 때문에 J_5는 다시 지연된다 이는 $L_{3,3}$이다
- 9초에 J_5는 자원1을 수행한다

즉, J_5의 P 지연시간은 지연요소 NC_5, L_5, $H_{1,5}$ $L_{3,3}$을 모두 합한 시간이 된다. SMP는 요소 4의 경우, 같은 프로세서에 할당된 모든

p	r	2			4			8		
	n \ u	30	50	70	30	50	70	30	50	70
2	16	99.97	99.87	100	100	100	100	100	100	100
	32	100	100	99.94	100.02	99.96	100	99.87	99.92	100
	64	100.01	99.98	99.93	100	99.94	99.82	100	100	100
	128	100	99.99	99.86	100	100	99.8	100.03	100	99.99
	256	100	99.98	99.97	100	100	99.97	100	100	99.99
	512	100	99.99	99.98	100	100	100	99.99	100	99.96
4	16	100	100	98.28	99.89	100	100	100	100	100
	32	100	99.5	98.59	100.02	100	99.98	100	100	99.95
	64	100	99.42	99.25	100	99.89	99.83	100	99.82	99.93
	128	99.98	99.97	99.95	100	99.95	99.85	100	99.99	99.77
	256	100	99.92	99.67	100	99.99	99.92	100	100	99.95
	512	100	99.98	99.85	100	99.99	99.9	100	100	99.93
8	16	100	99.97	99.63	99.69	100	98.9	100	100	100
	32	99.95	100.06	98.75	100	99.73	99.51	100.03	99.96	99.25
	64	99.8	99.4	99.89	100.03	99.84	98.75	99.99	99.75	99.6
	128	99.91	99.53	97.93	99.98	99.59	98.77	99.98	99.9	99.84
	256	99.93	100	99.99	100	99.87	98.32	99.97	99.95	99.86
	512	99.97	99.71	99.56	99.99	99.93	99.33	100	99.98	99.89

표 1 : SMP와의 시뮬레이션 결과

전역자원을 사용하는 모든 낮은 우선순위의 작업들의 자원수행시간의 합이 된다 이는 우선순위가 높은 작업일수록, 전역자원의 수행이 많아질수록 더욱 커지게 된다. 그러므로, 이를 한번으로 바운드 함으로써 우선순위가 높은 작업의 지연을 줄일 수 있다.

표 1은 SMP와의 시뮬레이션 결과를 나타낸다. p는 프로세서의 수, r은 자원의 수, n은 작업들의 수, u는 작업의 총 수행시간에 대한 자원 수행시간의 비율을 나타낸다. 결과 값은 SMP와 제안된 프로토콜에 의한 모든 작업의 총 응답시간의 비율이다. 결과에서 알 수 있듯이 프로세서가 많은 경우와 자원의 수가 적을수록 더 좋은 결과를 나타내는데 이는 전역자원의 수가 적으면 다른 전역자원에 의한 지연이 줄고 전역자원의 수가 많아지면 다른 전역자원에 의한 지연시간이 길어지기 때문에 SMP와 비슷한 결과를 보이기 때문이다. 그러므로, 작업들의 효율적인 프로세서 할당방법을 사용하여 전역자원의 수를 줄인다면 더욱 좋은 결과를 가져올 것이다. 또한, 자원의 수행시간의 비율이 클수록 좋은 결과를 보이는데 이로써 자원 수행에 의한 지연시간이 SMP보다 작다는 것을 알 수 있다.

6. 결론

작업들이 자원을 공유함으로써 공유된 자원을 수행하는데 긴 지연시간이 발생된다 이러한 지연시간을 줄이기 위해 전역자원의 수행에 높은 우선순위를 부여하는데 이는 전역자원을 사용하지 않는 작업들이 전역자원을 수행하는 작업들에 의해 또 다른 지연시간을 갖게 한다. 일반적으로 전역자원에 의한 지연은 전역자원이 수행되고 있는 도중에 발생하는 충돌에 의한 것이다 그러므로, 충돌이 발생한 경우에만 높은 우선순위를 부여함으로써 전역자원을 사용하고 있지 않는 작업들의 불필요한 지연을 줄일 수 있다. 이는 우선순위가 높은 작업에 더욱 높은 효과를 갖는다

7. 참고문헌

[1] C. L. Liu, J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real time environment", JACM 20(1), 1973.

[2] L. Sha, J. P. Lehoczky, Rajkumar, "Solutions for some practical problems in prioritized preemptive scheduling", IEEE Real_Time System Symposium, 1986.

[3] L. Sha, R. Rajkumar and J. P. Lehoczky, "Priority inheritance. an approach to real-time synchronization", IEEE Transactions on Computers, vol.39, no.9, September 1990, pp.1175-1185.

[4] R. Rajkumar, "Real-time synchronization protocol for shared memory multiprocessors", IEEE Real-Time System Symposium, 1990.

[5] R. Rajkumar, L. Sha, J. Lehoczky, "Real-time synchronization protocols for multi-processors", IEEE Real-Time System Symposium, 1988

[6] I. Rhee, G. R. Martin, "A scalable real-time synchronization protocol for distributed systems", IEEE Real-Time System Symposium, 1995.