실시간 운영체제의 우선순위 역전 현상에 대한 해결 기법

김인혁° 김재광 고광선 엄영익
성균관대학교 정보통신공학부
{kkojiband°, linux, rilla91, yieom}@ece.skku.ac.kr

A Scheme for Resolving Priority Inversions in Real-time Operating Systems

Inhyuk Kim°, Jaekwang Kim, Kwangsun Ko, Young Ik Eom
School of Information and Communication Engineering, Sungkyunkwan University

요 약

실시간 운영체제는 정해진 시간 내에 작업처리를 완료해야 하는 분야에 주로 사용되고 있으며, 최적의 실시간 운영체제를 설계 및 개발하기 위해서는 반드시 필요한 몇 가지 조건들이 있다. 본 논문에서는 실시간 운영체제에 필요한 조건 중에서 우선순위 역전 현상을 해결하는 기법을 제안한다. 기존에 우선순위 역전 현상을 해결하기 위하여 Basic Priority Inheritance 프로토콜, Priority Ceiling Emulation 프로토콜 등이 제안되었다. 그러나 이러한 기법들은 복잡한 형태의 우선순위 역전 현상에 대해서는 해결이 불가능하거나, 실행 시 비효율성 등의 문제가 발생하기 때문에 실제로는 여러 가지 기법들과 혼용되어 사용되었다. 이에 본 논문에서는 재귀적인 형태의 자료구조를 사용하여 우선순위 역전 현상을 효과적으로 해결하는 기법을 보이고, 기존 기법들과 비교한다.

1. 서 론

현재 실시간 운영체제는 우주탐사선, 미사일 등과 같이 정해진 시간 내에 작업처리를 완료해야하는 분야에 다방면으로 사용되고 있으며, 효과적인 스케줄링 정책, 인터럽트 처리 지연 최소화, 우선순위 역전 현상 해결 등의 조건을 만족시켜야한다. 이 중에서 우선순위 역전 현상은 실행 시 자주 발생하는 문제는 아니지만, 우주탐사선이나 미사일과 같이 몇 달 또는 몇 십 년씩 재가동 없이 동작하는 시스템의 경우 단 한 번의 오류가 시스템을 실패로 이끌게 되므로 반드시 해결해야 하는 문제이다. 이를 해결하기 위하여 여러 기법이 제안되었지만 완벽한 해결책은 제시되지 않은 상태이며, 일반적으로 다른 기법들과 혼용하여 해당 문제를 해결하고 있다. 이에 본 논문에서는 기존의 기법보다 더 효율적으로 우선순위 역전 현상을 해결하는 기법을 제안한다.

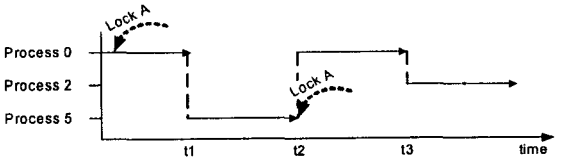
본 논문의 구성은 다음과 같다. 2장에서는 우선순위 역전 현상과 기존에 제안된 기법들에 대해 설명하고 3장에서는 2장에 소개된 기법들의 문제점들과 우선순위 역전 현상을 해결하기 위한 Recursive Data Structure 기반 프로토콜을 제안한다. 마지막 4장에서는 결론 및 향후 연구계획을 설명한다.

2. 관련연구

본 장에서는 우선순위 역전 현상에 대해 설명하고, 이를 해결하기 위하여 기존에 제시된 대표적인 기법인 Basic Priority Inheritance 프로토콜과 Priority Ceiling Emulation 프로토콜에 대해 설명한다.

2.1. 우선순위 역전 현상

우선순위 역전 현상이란 실시간 운영체제에서 우선순위가 낮은 프로세스가 사용 중인 자원을 우선순위가 높은 프로세스가 요청하여 할당받지 못하고 대기하는 현상을 의미한다. 이에 대한 구체적인 동작과정은 그림 1에서 보인다. 논문 전개 편의상 프로세스 0의 우선순위가 가장 낮고 프로세스 5의 우선순위가 가장 높다고 가정한다. 프로세스 0이 자원 A를 소유한 상태에서 프로세스 5가 실행된 후, 자원 A를 요청하면 프로세스 5는 프로세스 0이 자원 A를 해제할 때까지 대기 상태가 된다. 이때 프로세스 2가 실행되면 프로세스 0은 대기 상태가 되고 프로세스 5는 프로세스 2에 의해 실행되지 못한다. 따라서 프로세스 5가 프로세스 0과 프로세스 2때문에 실행되지 못하므로 이러한 상황을 우선순위 역전 현상이라고 한다. 이상의 과정은 그림 1이 보이는 바와 같다.



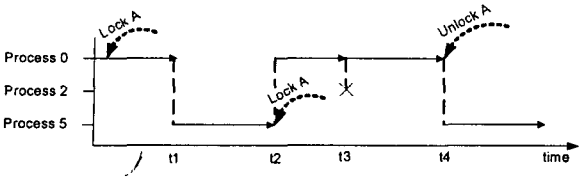
(그림 1) 기본적인 우선순위 역전 현상

t_1 : 프로세스 5 실행
t_2 : 프로세스 5가 자원 A 요청 (자원 A는 프로세스 0이 사용 중이므로 프로세스 5는 대기상태가 됨)
t_3 : 프로세스 2 실행

그림 1에서 보이는 바와 같이 프로세스 5는 자원 A를 얻기 위해 대기하는 시간이 부정확하기 때문에 실시간 운영체제에서는 치명적인 문제가 된다 [1].

2.2. Basic Priority Inheritance 프로토콜

Basic Priority Inheritance 프로토콜은 프로세스 0이 자원 A를 소유한 상태에서 프로세스 5가 실행하여 자원 A를 요청하면, 프로세스 0이 자원 A를 소유하는 동안 프로세스 5는 대기상태가 된다. 이때 프로세스 0은 임시로 프로세스 5의 우선순위를 상속받아서 실행되기 때문에 프로세스 0이 실행되는 동안 프로세스 0과 프로세스 5의 중간 우선순위를 가진 프로세스 2에 의해 선점되지 않는다. 그림 2는 Basic Priority Inheritance 프로토콜의 동작 과정을 보인다.



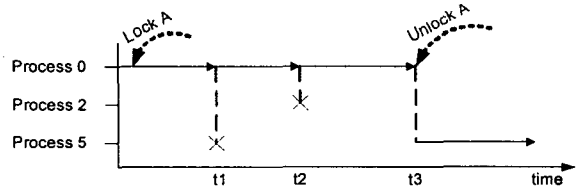
(그림 2) Basic Priority Inheritance 프로토콜의 동작과정

t1 : 프로세스 5 실행
t2 : 프로세스 5가 자원 A 요청 (자원 A는 프로세스 0이 사용 중이므로 프로세스 5는 대기상태가 됨)
t3 : 프로세스 2 실행
t4 : 프로세스 0이 자원 A 해제

그림 2가 보이는 바와 같이 프로세스 0은 임시로 프로세스 5의 우선순위를 상속받기 때문에 프로세스 2가 실행 가능 상태가 되어도 프로세스 0은 선점되지 않고 실행을 끝낸 후, 자원 A를 프로세스 5에게 넘겨줄 수 있다. 하지만 이 알고리즘은 단순한 우선순위 역전 현상만 해결이 가능하기 때문에 단독으로 사용될 수 없는 단점이 있다 [2].

2.3. Priority Ceiling Emulation 프로토콜

Priority Ceiling Emulation 프로토콜은 실시간 운영체제가 초기화 될 때 자원 A의 우선순위를 자원 A를 사용하는 프로세스 중에서 가장 높은 우선순위를 가지는 프로세스보다 높게 설정한다. 즉, 프로세스 0이 자원 A를 소유하면 자원 A의 우선순위를 임시로 상속받아 자원 A에 접근하는 다른 프로세스에 의해서 선점되지 않도록 한다. 그림 3은 Priority Ceiling Emulation 프로토콜의 동작과정을 보인다.



(그림 3) Priority Ceiling Emulation 프로토콜의 동작과정

t1 : 프로세스 5 실행
t2 : 프로세스 2 실행
t3 : 프로세스 0이 자원 A 해제 (프로세스 5 실행)

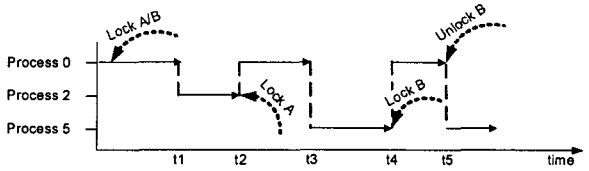
그림 3에서 보이는 바와 같이 프로세스 0이 자원 A를 소유하면 작업이 완료될 때까지 프로세스 2와 5에 의해서는 선점되지 않는다. 그러나 프로세스 2와 5가 자원 A를 요구하지 않는 상황에서도 프로세스 0은 항상 새로운 우선순위를 할당받아야 하기 때문에 불필요한 성능저하가 발생한다 [3].

3. 제안 기법

본 절에서는 기존에 제시된 기법의 문제점을 살펴보고 본 논문에서 제안하는 Recursive Data Structure 기반 프로토콜에 대해 설명한다.

3.1. 기존 기법의 문제점

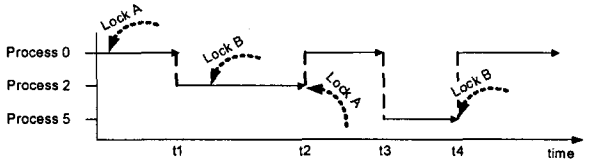
기존에 제시된 기법에서는 그림 4와 5가 보이는 바와 같이 두 가지 복잡한 형태의 우선순위 역전 현상을 해결할 수 없다. 그림 4는 첫 번째, 프로세스 0이 두 개의 자원 A와 자원 B를 소유한 상태에서 우선순위가 높은 프로세스 2와 5가 각각 자원 A와 B를 요청하는 경우를 보인다.



(그림 4) 자원을 동시에 여러 개 소유했을 경우의 우선순위 역전 현상

t1 : 프로세스 2 실행
t2 : 프로세스 2가 자원 A 요청 (자원 A는 프로세스 0이 사용 중이므로 프로세스 2는 대기상태가 됨)
t3 : 프로세스 5 실행
t4 : 프로세스 5가 자원 B 요청 (자원 B는 프로세스 0이 사용 중이므로 프로세스 5는 대기상태가 됨)
t5 : 프로세스 0이 자원 B 해제

그림 5는 두 번째, 프로세스 0이 자원 A를 소유하고 있는 상태에서 프로세스 2가 자원 B를 소유한 뒤 자원 A를 요청하여 대기 상태가 되고, 프로세스 5가 자원 B를 요청하는 경우를 보인다.



(그림 5) 재귀적 형태의 우선순위 역전 현상

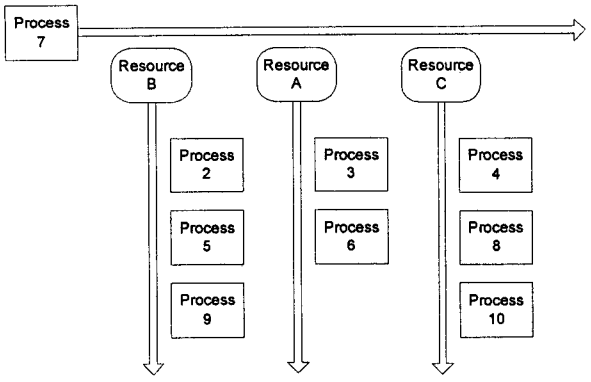
t1 : 프로세스 2 실행
t2 : 프로세스 2가 자원 A 요청 (자원 A는 프로세스 0이 사용 중이므로 프로세스 2는 대기상태가 됨)
t3 : 프로세스 5 실행
t4 : 프로세스 5가 자원 B 요청 (자원 B는 프로세스 2가 사용 중이므로 프로세스 5는 대기상태가 됨)

Basic Priority Inheritance 프로토콜은 기본적인 형태의 우선순위 역전 현상은 해결이 가능하지만, 그림 4와 5가 보이는 바와 같은 복잡한 형태의 우선순위 역전 현상은 해결이 불가능하다. 또한, Priority Ceiling Emulation 프로토콜은 그림 4가 보이는 바와 같은 형태의 우선순위 역전 현상은 해결이 가능하지만, 그림 5가 보이는 바와 같은 재귀적인 형태의 우선순위 역전 현상은 해결이 불가능하며, 불필요한 경우에도 우선순위 상속이 발생해서 실행 시 성능이 떨어지는 단점이 있다 [4].

기본적으로 우선순위 역전 현상은 실시간 운영체제의 모든 프로세스와 자원들 사이에서 재귀적인 형태로 발생한다. 기존에 제시된 기법들은 이런 재귀적인 관계는 고려하지 않았기 때문에 그림 4와 그림 5에서 보이는 바와 같이 복잡한 형태의 우선순위 역전 현상은 해결이 불가능하다.

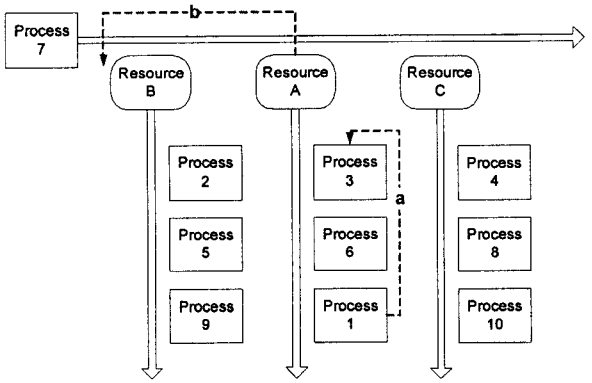
3.2. Recursive Data Structure 기반 프로토콜

본 논문에서 제안하는 Recursive Data Structure 기반 프로토콜은 재귀적인 형태의 자료구조를 사용함으로써 복잡한 우선순위 역전 현상을 해결한다. Recursive Data Structure 기반 프로토콜이 동작하기 위한 자료구조는 그림 6이 보이는 바와 같다.



(그림6) Recursive Data Structure 기반 프로토콜이 동작하기 위한 자료구조

그림 6에서 보이는 바와 같이 프로세스 7은 자원 A, B, C를 소유하고 있으며, 각 자원들에는 해당 자원을 요청한 프로세스가 대기하는 큐가 존재한다. 각 자원별로 대기 중인 프로세스는 우선순위에 의해 정렬이 되어있으며, 프로세스 7이 소유 중인 자원들도 각 자원의 첫 번째 대기 프로세스의 우선순위에 의해 정렬되어 있다. 그러므로 첫 번째 자원의 첫 번째 대기 프로세스가 항상 가장 높은 우선순위를 갖게 된다. 즉, 프로세스 7은 자원 B의 첫 번째 대기 프로세스인 프로세스 2의 우선순위를 상속받게 된다. 그림 6에서 보이는 자료구조는 모든 프로세스마다 가지고 있기 때문에 재귀적 자료구조라 한다. 그림 6에서 보이는 바와 같은 상황에서 프로세스 1이 자원 A를 요청하고 대기 상태에 들어간 뒤의 상황은 그림 7과 같다.



(그림 7) 자원 A를 요청한 프로세스 1이 추가되었을 경우 재귀적 자료구조의 변화

그림 7에서 보이는 바와 같이 프로세스 1이 자원 A의 대기 프로세스 목록에 추가되어 재정렬되어야 하며, 전체 자료구조는 정렬되어있으므로 a의 경우 재정렬 시간은 O(n)이고, b의 경우 재정렬 시간도 O(n)이 된다. 그리고 실제 실행 시에는 하나의 프로세스가 여러 개의 자원을 동시에 소유하거나, 여러 프로세스가 자원을 동시에 요청하는 경우가 적으므로 재정렬에 드는 비용은 낮다.

프로세스 7이 새로운 자원을 요청하거나 소유 중인 자원을 해제하거나, 또는 대기 중인 프로세스의 우선순위가 상속에 의해 변경되는 경우에도 그림 7에서 보이는 바와 같은 방법으로 재정렬되고 프로세스 7은 새로운 우선순위를 상속받게 된다.

따라서, 본 논문에서 제시하는 Recursive Data Structure 기반 프로토콜을 사용할 경우 연관된 프로세스들의 우선순위가 재귀적으로 상속되어 그림 4와 그림 5에서 보이는 바와 같은 복잡한 형태의 우선순위 역전 현상도 해결이 가능하다. 이에 대해 기존 기법과 비교하면 표 1과 같다.

[표 1] 기존 기법과 제안기법과의 성능 비교

해결방법 / 비교기준	Basic Priority Inheritance 프로토콜	Priority Ceiling Emulation 프로토콜	Recursive Data Structure 기반 프로토콜
A	O	O	O
B	X	O	O
C	X	X	O
D	낮음	높음	중간

표 1에서 사용된 비교기준은 다음과 같다.

A : 우선순위 역전 현상 기본 형태
B : 자원을 동시에 여러 개 소유했을 경우의 우선순위 역전 현상 (그림 4 참조)
C : 재귀적 형태의 우선순위 역전 현상 (그림 5 참조)
D : 실행 시 추가비용

표 1이 보이는 바와 같이 Basic Priority Inheritance 프로토콜은 실행 시 추가비용이 낮으나 B와 C의 경우 해결이 불가능하다. Priority Ceiling Emulation 프로토콜의 경우 C는 해결이 불가능하고 실행 시 추가비용이 높다. 반면 본 논문에서 제안한 Recursive Data Structure 기반 프로토콜은 A, B, C 모든 경우에 해결이 가능하며 실행 시 추가비용이 Priority Ceiling Emulation 프로토콜에 비해 낮다 [5].

4. 결론 및 향후계획

본 논문에서는 실시간 운영체제에서 문제가 되는 우선순위 역전 현상을 해결하기 위한 기법으로 Recursive Data Structure 기반 프로토콜을 제안하였다. Recursive Data Structure 기반 프로토콜의 장점으로는 기존의 제시된 기법으로는 해결이 불가능했던 다수의 자원을 동시에 소유하는 경우와 재귀적 형태의 우선순위 역전 현상 해결 가능, 간단한 구현, 실행 시 성능상의 효율성 등이 있다. 향후 연구 계획으로는 제안 기법에 대한 실제 구현 및 테스트를 통해 실질적인 성능평가를 실시하고자 한다.

[참고 문헌]

[1] TimeSys Inc., Priority Inversion: Why You Care and What to Do About It. A White Paper, 2004.
[2] 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, Sep. 1990, pp. 1175-1185.
[3] J. B. Goodenough and L. Sha, The Priority Ceiling Protocol, Special Report CMU/SEI-88-SR-4, Mar. 1998.
[4] V. Yodaiken, "The Dangers of Priority Inheritance," Department of Computer Science New Mexico Institute of Technology, 2001.
[5] V. Yodaiken, Against Priority Inheritance, FSMLABS Technical Paper, Jul. 2003.