분산 실시간 시스템에서의 자원 제어 기법
(A Resource Control Technique in Distributed Real-Time Systems)

이 은 미 † 허 신 ††
(Eun-Mi Lee) (Shin Heu)

요 약 본 논문에서는 분산 실시간 시스템에서 공유되는 자원들에 대한 실시간적 특성을 분석하고, 태스크가 이들 자원을 요청했을 때, 봉쇄시간을 예측하는 자원 관리자를 제안한다. 분산 환경에서, 우리는 봉쇄의 주요 원인인 우선순위 역전 문제와 함께 원격 봉쇄의 문제점을 고려해야 한다. 본 논문에서 우선순위 역전 문제는 동기 프로토콜로 잘 알려진 Priority Ceiling Protocols(PCP)를 사용하여 해결하였다. 또한, 원격 봉쇄의 문제에 대해서는, 전역자원을 다른 지역자원들 보다 우선적으로 수행함으로써 원격 태스크들로 인한 봉쇄시간을 예측할 수 있도록 하였다. 본 논문의 자원 관리자는 할당된 자원과 태스크들의 관계 목록을 이용하여, 요청된 자원의 상태에 따른 봉쇄요인을 분석하고, 그 결과로 태스크가 자원 수행을 마칠 때까지 소요되는 봉쇄시간의 상한값을 결정한다. 또한, 이러한 상한값의 타당성을 수학적으로 증명하였다.

Abstract In this paper, we analyze the real-time characteristics of resources sharing in distributed real-time systems, and present a resource manager that can estimate the blocking time of each task that requests a resource locally (or remotely). In distributed environments, we should consider the priority inversion problem that is a main cause of the blocking and also, consider the remote blocking problem. In this paper, we solve the priority inversion problem using the Priority Ceiling Protocol (PCP) known as synchronization protocols. For the remote blocking problem, the blocking time due to the remote tasks is predictable by executing a global resource prior to other local resources. The resource manager analyzes the blocking factors according to the state of a resource by using relation lists between resources and tasks, and determines an upper bound of its blocking time until a task completes the execution of the resource. Furthermore, we analytically prove the validity of this upper bound.

1. 서 론

일반적으로 비 실시간 응용에서는 모든 태스크들에 대해 적절한 응답성을 제공하도록 자원들을 관리하는데 비해, 실시간 응용에서는 각 태스크들에 대해 명시적으로 지정된 실시간 특성 즉, 시간적인 요구조건을 만족해야 한다. 즉, 응답의 정확성뿐만 아니라 응답시간도 태스크 수행의 성공여부의 필수적인 조건이 된다. 이러한 시간적 요구조건은 태스크가 계산을 완료해야 하는 마감시한 제약을 갖도록 설정하여, 마감시한 이내에 계산을 완료하지 못하면 그 결과는 치명적이 될 수 있다[1]. 이러한 태스크의 시간 제약 조건을 만족시키기 위한 스케줄링 문제는 지속적으로 연구되어 왔으며, 많은 연구성과들이 산업 현장에 응용되고 있다.

하지만, 최근 실시간 연구분야는 한층 더 복잡, 다양해지고 분산 지향적인 실시간 플랫폼 구조를 필요로 하고 있으며, 이에 따라 다양한 요구조건들을 수용해야 하는 문제에 직면하고 있다. 이러한 요구조건들 중에서 자원 제어의 문제는 한층 중요해지고 있다. 분산 실시간 시스템에서 다수의 공유되는 자원을 허용함은 전체 시스템의 효율을 최대한으로 높일 수 있는 장점 때문에 불가피한 것으로, 이는 프로세서에 대한 태스크들의 스케줄링 정책에 있어 근본적인 고려요건이 되고 있다. 일반적으로 실시간 시스템에서 자원 제어와 할당 문제에

† 정 회 원 : 한양대학교 전자계산학과
emlee@cse.hanyang.ac.kr
†† 종신회원 : 한양대학교 전자계산학과 교수
shinheu@cse.hanyang.ac.kr
논문접수 : 1999년 8월 12일
심사완료 : 2000년 12월 29일

대한 해결법을 제안하는데 있어서 요구되는 속성은 예측성(predictability)과 한계성(boundedness)을 들 수 있다[2]. 예측성은 자원 할당에 관련된 결정이 시스템 수행 이전에 예측되어야 함을 의미하며, 한계성은 태스크의 실행시간을 자원 접근에 관련해서 한계 지어야 하고, 이 한계는 스케줄 가능성을 결정할 수 있도록 수행 전에 계산되어야 한다. 이러한 속성들을 만족시키기 위한 방법은 결정적인 수행동작들로 제한되어야 하고, 이러한 요건을 만족시키지 못하면 태스크 집합의 스케줄가능성은 어려워지고, 최악의 경우 태스크의 마감시한 보장은 실현될 수 없다.

본 논문에서는 분산 실시간 시스템에서 공유되는 자원들에 대한 실시간적 특성을 분석하고, 태스크가 이들 자원을 요청했을 때, 봉쇄시간(blocking time)을 예측하는 자원 관리자를 제안한다. 자원 관리자는 태스크로부터 자원 요청이 들어왔을 때, 요청한 자원의 상태에 따른 봉쇄요인을 분석하고, 그 결과로 태스크가 요청한 자원의 임계구역 수행을 마칠 때까지 소요되는 봉쇄시간의 상한값을 결정한다. 이러한 상한값은 요청한 태스크가 지역적으로 접근했는지, 전역적으로 접근했는지, 또는 요청한 자원이 지역적으로 허용된 자원인지 전역적으로 허용되었는지에 따라 선택적으로 결정될 수 있다. 본 논문에서는 전역자원을 위한 별도의 프로세서를 지정하지 않았으며, 임의로 할당된 태스크들과 자원들간의 몇 가지 관계 목록들을 이용해서, 각 태스크들의 봉쇄시간을 예측할 수 있도록 하였다. 또한, 구해지는 각 경우의 상한값의 타당성을 수학적으로 증명하였으며, 기존연구인 MPCP(Multiprocessor Priority Ceiling Protocol)의 예제와 비교하여 봉쇄시간을 감소시킬 수 있음을 보였다.

본 논문의 구성은 다음과 같다. 2 장에서는 자원 제어문제를 고려할 때, 주요 동기가 되는 우선순위 역전문제와 원격 봉쇄 문제점을 언급하고, 이를 해결하는데 주력해온 기존 연구 방법들에 대해 알아보고, 3장에서는 본 논문에서 제안하는 자원 관리자의 개념과 역할을 소개한다. 4장에서는 자원 관리자를 적용했을 때 계산되는 봉쇄시간의 상한값을 수학적으로 분석하고, 5장에서 구한 상한값이 기존 연구 방법보다 감소됨을 보이고, 6장에서 결론을 맺는다.

2. 기존 연구

2.1 우선순위 역전

우선순위 역전 (priority inversion)문제는 상호배제적으로 허용되는 자원에 대해서, 태스크가 자원을 요청했을 때, 이 자원이 다른 낮은 우선순위 태스크에 의해 사용 중이라면, 낮은 우선순위 태스크가 자원의 임계구역 수행을 종료할 때까지 높은 우선순위 태스크가 기다리게 될 때 발생한다[4].

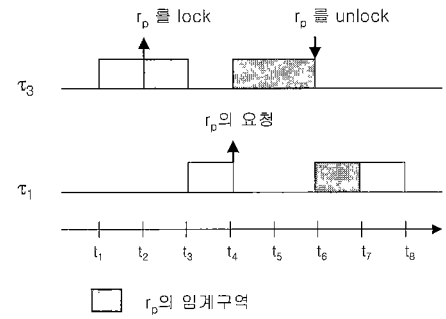

그림 1 우선순위 역전

그림 1의 설명을 위해, $i < j$ 일 때, 태스크 τ_i의 우선순위가 태스크 τ_j의 우선순위보다 높다고 하자. t_1 시각에 τ_3가 도착하고 수행 중 자원 r_p를 요청하여 그 임계구역을 수행 중일 때, t_3 시각에서 새로운 높은 우선순위 태스크 τ_1이 도착한다면, τ_1은 낮은 우선순위 태스크 τ_3를 선점하고 수행을 시작한다. 이때, 만일 τ_1도 t_4 시각에서 같은 자원 r_p를 요청한다면, 이 자원 r_p는 이미 선점 당한 τ_3에 의해 lock이 걸려 있으므로 τ_1은 r_p를 거절당하고, τ_3가 r_p를 lock을 해제할 때까지 기다려야 한다. 이는 자원의 임계구역 수행에 대해 비선점적이기 때문이며, 이때, 높은 우선순위 태스크는 낮은 우선순위 태스크에 의해 봉쇄(block) 된다고 하고, 이 봉쇄구간은 그림 1에서 t_4에서 t_6 까지의 구간이 된다. 실시간 시스템에서 이러한 우선순위 역전 현상이 문제가 되는 것은 높은 우선순위 태스크가 낮은 우선순위 태스크에 의해 봉쇄되는 시간을 예측할 수 없다는 것이다. 예를 들어 t_4시각에서 τ_1이 봉쇄상태에 들어가고, τ_3가 r_p를 재개한 후에 τ_3보다 높은 우선순위를 갖는 또 다른 태스크 τ_2가 새로 도착하는 경우가 생길 수 있다. 이때, τ_2는 τ_3을 선점하게 되고, 임의의 자원을 요청하고 그 임계구역 수행에 들어가거나 τ_1과 같은 이유로 봉쇄 상태가 될 수 있다. 이때, 문제는 τ_1이 봉쇄되는 시간이 τ_2로 인해 더 길어질 수 있다는 점이다. 이 시간은 τ_2와 같은 또 다른 태스크들의 도착으로 더 지연될 수 있으며, 결과적으로 이 봉쇄시간은 예측할 수 없다. 따라서, τ_1의 봉쇄시간이 단순히 봉쇄시키는 낮은 우선순위 태스크의 하나의 자원 임계구역 시간동안이 아니라, 봉쇄되

고 있는 동안 다른 중간 우선순위 태스크들에 의해 선점될 수 있음으로 해서 봉쇄 시간이 예측 불가능하게 지연될 수 있다. 이는 실시간 시스템에서의 한계성(boundedness) 요구조건에 위배된다.

이러한 우선순위 역전 문제를 해결하기 위해 여러 가지 해결법들이 발표되었다[4-8]. 이러한 해결법들은 적절한 프로토콜을 사용하여 높은 우선순위 태스크가 봉쇄되는 시간을 예측 가능하게 하는데 주력해 왔으며, 기본적으로 Basic Priority Inheritance Protocols(BPIP)과 Priority Ceiling Protocols(PCP) 동기화 프로토콜을 기반으로 하고 있다. BPIP에서는 한 개 이상의 높은 우선순위 태스크들이 낮은 우선순위 태스크에 대해 봉쇄되고 있을 때, 낮은 우선순위 태스크는 자신의 원래 우선순위를 무시하고 높은 우선순위 태스크들 중 가장 높은 우선순위를 상속받아 이 우선순위에서 자원의 임계구역을 수행한다. 그림 1에서 τ_3는 t_4 시각에서 τ_1의 우선순위를 상속받아 이 우선순위에서 r_p의 임계구역을 수행한다. 이 임계구역을 종료한 뒤에는 자신의 원래 우선순위로 복귀한다. 이와 같은 정책으로, τ_3가 τ_1의 우선순위에서 r_p의 임계구역을 수행하는 동안 다른 중간 우선순위 태스크에 의해 선점당하는 것을 피할 수 있다. Rajkumar 등은 [4]에서 이 프로토콜을 사용하면, 높은 우선순위 태스크가 봉쇄되는 최대 시간은 이 태스크를 봉쇄시키는 각각의 낮은 우선순위 태스크들의 가장 긴 임계구역 수행 동안임을 밝혔다. 하지만, 이 BPIP로는 데드락과 다중 봉쇄로 인한 문제를 해결할 수 없으며, 이러한 문제를 해결하기 위해 확장된 것이 PCP이다. PCP는 기본적으로 BPIP의 개념을 그대로 사용하면서, 각 자원들에 각각의 ceiling 값을 할당한다. ceiling 값은 각 자원을 사용하려는 태스크들 중 가장 높은 우선순위를 갖는 태스크의 우선순위 값이다. 이 정책은 임의의 태스크 τ가 다른 태스크의 임계구역을 선점하고 자신이 요청한 자원의 임계구역을 실행하려 할 때, 이 새로운 임계구역이 실행되는 우선순위는 모든 선점된 임계구역들의 상속된 우선순위보다 높아야 함을 보장한다. 만일 이 조건이 만족되지 않으면, 태스크 τ는 요청한 임계구역에 진입하는 것을 거부당하고 보류된다. 이때, τ를 봉쇄시키는 낮은 우선순위 태스크는 τ의 우선순위를 상속받는다. 이 정책 역시, 봉쇄시간의 상한이 낮은 우선순위 태스크의 가장 긴 임계구역 구간임이 증명되고 있다[4,8].

2.2 원격봉쇄의 문제점

우선순위 역전 문제는 분산 실시간 환경에서는 한층 더 복잡해진다. 분산 실시간 환경에서 자원은 지역적이거나 전역적이다. 전역자원이라 함은 자원을 관리하고 있는 프로세서가 아닌 다른 프로세서에서 수행 중인 태스크에 의해 요청되는 자원이다. 이때, 전역자원의 요청과 해제 사이의 임계구역은 전역 임계구역이라 하고, 반면에 지역자원에 의한 임계구역은 지역 임계구역이라 한다. 분산 실시간 환경에서는 태스크가 원격으로 자원을 요청할 때, 이 자원에 대해 이미 lock을 갖고 수행중인 높은 우선순위 태스크에 의해 봉쇄되거나, 또는 태스크가 지역적으로 같은 프로세서의 자원을 요청했을 때에도, 이 자원이 다른 프로세서에서 원격 요청을 한 태스크에 의해 점유됨으로써 봉쇄되는 경우가 있다. 전역자원인 경우의 근본 문제는 높은 우선순위 태스크가 수행 중일 때, 낮은 우선순위 태스크의 자원요청이 도착하는 상황이 발생하기 때문이다. 이러한 경우는 그림 2에 나타나 있다.

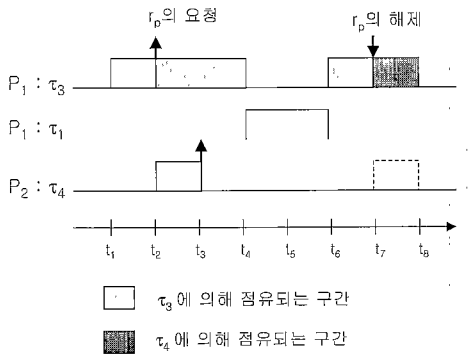

그림 2 원격 봉쇄

그림 2에서 프로세서 P_1의 τ_3가 t_1시각에서 수행을 시작하고, t_2시각에 요청한 자원 r_p를 점유하고 임계구역을 수행할 때, 다른 프로세서 P_2에서 수행 중이던 낮은 우선순위 태스크 τ_4가 t_3시각에 전역적으로 자원 r_p를 요청해 올 수 있다. 이때, 프로세서 P_1의 τ_3가 낮은 우선순위 태스크의 요청을 무시하고 자원의 임계구역 수행을 계속하면 프로세서 P_2의 τ_4는 요청한 자원에 대해 원격 봉쇄상태에 들어간다. 이때, τ_4를 봉쇄시킨 태스크 τ_3는 수행 도중 같은 프로세서의 높은 우선순위 태스크 τ_1에 의해 선점될 수 있고, 높은 우선순위 태스크의 수행이 끝난 후 다시 임계구역 수행을 재개한다. 이때, 문제는 프로세서 P_2의 τ_4는 자신을 봉쇄시킨 태스크가 있는 프로세서에서 발생하는 이러한 지연을 계속해서 기다려야 한다. 그림 2에서 이 구간은 $[t_4, t_6]$이고, 이는 프로세서 P_1에서 자원의 임계구역 수행을 선점시키는

높은 우선순위 태스크들의 수행시간으로, 이 시간을 정확히 예측하기란 어렵다. 이는 단일 프로세서에서의 봉쇄와는 달리 원격의 낮은 우선순위 태스크가 높은 우선순위 태스크에 의해 봉쇄되는 현상으로, 이와같이 원격 태스크로 인해 봉쇄되는 경우를 원격 봉쇄(remote blocking)라 하며, 분산 환경을 고려한다면, 이 봉쇄의 요인을 분석하고 이를 봉쇄 상한값에 추가해야 한다.

Rajkumar 등[8]은 이러한 원격 봉쇄의 문제점을 언급하고, 이를 해결하기 위해 PCP를 확장한 MPCP를 개발하였다. MPCP에서 프로세서는 응용 프로세서와 동기 프로세서 두 그룹으로 구분된다. 응용 프로세서에서는 응용 작업들만을 수행하고, 동기 프로세서에서는 전역자원들이 할당되어 관리된다. 이 동기 프로세서에는 전역자원들과 함께 응용 태스크들도 할당되는데, 이때 응용 태스크들은 다른 응용 프로세서에 할당된 태스크들 보다 낮은 우선순위 태스크들이다 이와 같이 전역자원과 낮은 우선순위 태스크들을 같은 프로세서에 할당함으로써, 현재 수행 중인 태스크보다 낮은 우선순위를 갖는 원격 태스크로 부터의 전역자원 요청을 방지할 수 있다. 즉, 이러한 방법으로 MPCP에서는 앞서 설명한 원격봉쇄를 일으키는 요인을 제거하고 있다. MPCP는 응용프로세서든, 동기 프로세서든 상관없이 PCP를 사용한다. 이때, 전역자원을 허가받은 태스크가 허가받지 않은 태스크에 의해 선점되는 것을 막기 위해, 각 전역 자원의 ceiling 값은 시스템에서 태스크들 중 가장 높은 우선순위 값보다도 높아야 한다. 다시 말하면, 전체 시스템의 태스크들 중 가장 높은 우선순위 값을 π_{max}라 하고, 각 전역자원 R을 요청하는 태스크들 중 가장 높은 우선순위 태스크의 우선순위 값을 π_R이라 할 때, R의 ceiling은 $\pi_R + \pi_{max} + 1$이다. MPCP의 스케줄 가능성 분석은 단일 프로세서에서의 PCP와 같고 각 태스크가 봉쇄되는 요인은 다음의 4가지이며, 봉쇄시간의 상한은 이들 4개 값의 합이다.

(i) local blocking : 임의의 프로세서 P에 한정된 태스크가 같은 프로세서의 낮은 우선순위 태스크들에 의해 봉쇄되는 구간으로 이 상한은 $(n_i^G + 1) \times cs_i^L$이다. 이때 n_i^G는 프로세서 P에서 태스크 τ_i가 종료 전에 진입하는 전역 임계구역들의 수이고, cs_i^L는 τ_i를 봉쇄하는 가장 긴 지역 임계구역 구간이다.

(ii) global blocking : 태스크 τ_i가 전역 자원을 요청했을 때, 바로 들어가지 못하고 봉쇄되는 구간으로 이 상한은 $n_i^G \times cs_i^G$이다. 이때, cs_i^G는 τ_i를 봉쇄하는 가장 긴 전역 임계구역 구간이다.

(iii) remote blocking : A_p를 τ_i보다 높은 우선순위를 가지면서 프로세서 P가 아닌 다른 동기 프로세서에서 수행하고 전역 자원에 접근하는 태스크들의 집합이라 할 때, 이 태스크들 τ_k ($\in A_p$)의 최대 봉쇄시간 $CS_k \times \lceil p_i / p_k \rceil$ 의 합이다. 이때, CS_k는 태스크 τ_k가 동기 프로세서에서 단독으로 전역 임계구역을 수행하는 시간이고, p_i 와 p_k 는 각각 τ_i 와 τ_k의 주기이다.

(vi) deffered blocking : A_q를 같은 프로세서 P에 할당된 전역 임계구역에 접근하는 낮은 우선순위 태스크들의 집합이라 할 때, 이 태스크들 τ_k ($\in A_q$)의 전역 임계구역의 수행에 의해 선점되는 구간이다. 이 상한은 각 태스크 τ_k의 최대 봉쇄 시간 $CS_k \times \lceil p_i / p_k \rceil$ 의 합이다.

이와 같이 MPCP에서는 근본 문제인 원격 봉쇄문제를 제거하면서 태스크의 봉쇄 상한값을 결정하였다. 하지만, 이 프로토콜은 구조적으로 전역 자원의 임계구역 수행을 위한 전담 프로세서를 필요로 하고, 그 상한값이 지나치게 비관적이어서, 자원의 임계구역이 커지면, 봉쇄한계도 불필요하게 증가할 수 있다는 단점이 있다. 또한, 동기 프로세서에 할당되어 있는 낮은 우선순위 태스크의 지연시간을 지나치게 지연시킨다는 단점이 있다.

하지만, 본 논문에서는 MPCP의 동기 프로세서와 같은 전담 프로세서를 두지 않으며, 각 프로세서에는 지역적으로 요청되는 자원과 전역적으로 요청되는 자원 모두를 필요에 따라 할당할 수 있다. 이러한 전제하에 요청한 자원이 지역인지 전역인지에 따라 그 봉쇄시간을 구분하여 결정짓도록 하였다. 따라서 본 논문의 경우, 위와 같은 원격 봉쇄 문제에 당면하게 되고 이를 해결해야 한다. 단일 프로세서에서 예측 불가능한 봉쇄시간을 해결하기 위해 사용하는 PCP의 기본 방법은 우선순위 상속이다. 즉, 봉쇄되는 높은 우선순위 태스크의 우선순위 값을 봉쇄시키는 태스크의 자원 실행 우선순위로 상속함으로써, 중간 우선순위 태스크의 수행을 방지할 수 있다. 하지만, 원격 봉쇄의 경우 이를 그대로 적용할 수 없는데, 이는 봉쇄된 원격 태스크의 우선순위가 봉쇄시키는 태스크의 자원 실행 우선순위 보다 낮을 수 있기 때문이다. 이때, 봉쇄된 태스크의 우선순위보다 높은 우선순위 태스크는 도중에 도착하여 수행을 시작할 수 있으므로 그림 2에서와 같은 상황은 여전히 발생한다. 따라서, 원격 봉쇄를 일으키는 주요 원인인 높은 우선순위 태스크의 수행을 방지하는 것이 주요 관건이며, 본 논문에서는 전역자원에 상속되는 우선순위 값을 전역자원의 ceiling 값으로 취함으로써 이를 해결하고자 한다.

위의 그림 2를 예로 들어 보다 자세히 설명하면 다음과 같다. t_3시각에서 τ_4가 자원을 요청했을 때, 발생할

수 있는 경우는 τ_4가 봉쇄되는 경우와 요청한 자원을 허가 받는 경우이다. 전자의 경우는 τ_4가 요청한 자원이 r_p이거나 PCP의 조건을 만족하지 못하는 경우이고, 후자의 경우는 PCP의 조건을 만족하는 경우로 이때, 요청한 자원은 r_p가 아닌 다른 자원인 경우이다. 본 논문에서는 이 두 경우의 r_p와 다른 자원 r_q은 모두 전역 자원에 의해 접근되는 자원들로 그 ceiling 값은 각각 MPCP에서처럼 전체 시스템에서 가장 높은 우선순위 값보다도 높은 값으로 설정된다. 즉, 각 자원 r_p(또는 r_q)를 요청하는 태스크들 중 가장 높은 우선순위 값 + 전체 시스템에서 가장 높은 우선순위 값 + 1이다. 이때, 본 논문에서는 t_3시각에서 τ_4로부터 자원 요청이 들어왔을 때, r_p나 r_q의 실행 우선순위를 자신의 ceiling값으로 재설정하도록 하였다. 이와같이 하면 t_3시각에서 상속되는 실행 우선순위는 적어도 프로세서 P_1에서 가장 높은 우선순위 태스크 τ_1의 우선순위보다는 크기 때문에 τ_4가 봉쇄되거나 자원 임계구역 수행 도중에 프로세서 P_1에서의 어떠한 지역 태스크로부터 선점되는 일은 발생하지 않는다. PCP와 마찬가지로 r_p나 r_q의 임계구역 수행이 종료되면, 이전의 실행 우선순위로 돌아간다. 이때, 프로세서의 큐에는 봉쇄되고 있던 τ_4와 함께, 도착했으나 수행에 들어가지 못한 높은 우선순위 태스크가 대기하고 있을 수 있다. 이런 경우 봉쇄되고 있는 원격 태스크 τ_4에게 자원 r_p를 우선적으로 허용한다. 이와 같이 원격 태스크를 봉쇄시키고 있는 지역 태스크의 자원 임계구역 수행이나 원격 태스크에 의해 수행되고 있는 자원 임계구역의 실행 우선순위를 프로세서에서 가장 높은 우선순위로 유지함으로써, 다른 높은 우선순위 태스크의 선점을 막을 수 있고, 자원을 수행하는 프로세서에서는 원격 태스크로 부터의 요청을 PCP 조건을 만족하는 범위 내에서 우선적으로 수행할 수 있도록 하였다.

또 다른 고려해야 할 문제로는 하나의 전역 자원에 대한 원격 태스크들간의 경쟁이다. 그 예로 그림 2에서 프로세서 P_1에서 t_2시각에 수행을 시작한 자원 r_p가 전역자원으로 다른 프로세서 P_3의 원격 태스크로부터 요청된 자원이고, t_3시각에 프로세서 P_2의 τ_4가 이 자원 r_p를 요청하는 경우이다. 이 경우는 각 전역자원에 대한 ceiling값을 정할 때, 각 전역자원에 대해 접근하려는 태스크들 중 가장 높은 우선순위 태스크들의 우선순위에 따라 ceiling 값이 달라지므로 이러한 ceiling값의 차이에 따라 허가 여부를 결정할 수 있다.

3. Resource Manager

자원 제어를 위해서는 적절한 동기 프로토콜이 필요하고, 동기화로 인한 시간 정도를 분석하여 스케줄링의 한 요소로 정의하여야 한다. 본 논문의 자원 제어 방법에서 동기 프로토콜은 기본적으로 PCP를 사용한다. 하지만 PCP는 분산 환경의 자원 제어에는 그대로 적용할 수 없으므로, 분산된 태스크들의 예측성 있는 봉쇄시간 상한값을 구하기 위해 본 논문에서는 자원 관리자를 제안한다. 자원 관리자는 프로세서 당 하나씩 할당되어 있고, 그 프로세서에 할당된 태스크나 다른 프로세서의 태스크, 그리고 할당된 자원들의 목록과 상태를 유지한다. 즉, 태스크의 자원 요청이 도착하면, 자원을 허가할지 또는 대기시킬지를 결정하고, 각 경우에 대해 예측 가능한 봉쇄시간을 포함한 자원 한계시간을 계산하여 태스크에 되돌린다.

3.1 가정과 정의

본 논문 전반에 배경이 되는 몇 가지 가정과 정의는 다음과 같다.

가정

1. 시스템 환경은 독립된 프로세서 P_i ($\in P = \{P_1, P_2, P_3, \cdots, P_l\}$)들로 구성된 분산환경이다.
2. 분산 할당되어 있는 태스크들의 우선순위는 전역적으로 순서화되어 있다.
3. 프로세서에는 하나 이상의 태스크 τ_i($\in T = \{\tau_1, \tau_2, \tau_3, \cdots, \tau_m\}$)가 정적으로 할당되어 있으며 각 태스크들은 선점적이다.
4. 프로세서에는 하나 이상의 자원($R = \{r_1, r_2, \cdots, r_n\}$)이 정적으로 할당되며, 각 자원들은 비선점적으로 수행된다.
5. 시스템에 할당되어 있는 임의 자원 r_k는 상호 배제적으로 허용된다.
6. 자원 임계구역 수행시간은 전담 알고리즘에 의해 계산된다.
7. 내포된(nested) 전역 임계구역과 임계구역간의 오버랩(overlap)은 허용하지 않는다.
8. 한 프로세서에서 하나의 태스크는 동일한 임계구역에 대해 한번 이상 접근하지 않는다.

본 논문에서는 자원들에 대해 같은 프로세서 내의 태스크로부터 접근하는 경우와 다른 프로세서의 태스크로부터 접근하는 경우를 구분하기 위해, 지역 태스크와 원격 태스크로 명명하고, 태스크들에 대해서도 같은 프로세서내의 자원에 접근하면 지역 자원으로, 다른 프로세서의 자원에 접근하는 경우 전역 자원으로 구분한다. 구분 없이 태스크나 자원을 표기한 경우는 둘 모두에 대해 말하고 있음을 미리 밝혀둔다. 프로세서에 할당되어 있는 태스크들과 자원들은 가정에 의해 정적으로 할당

되며, 자원에 대한 태스크의 요청여부도 미리 할당되어 있다고 가정한다. 또한, 본 논문에서 제안하고 있는 자원 관리자는 프로세서 당 하나씩 존재하며, 관리하는 각 자원들에 대한 태스크의 할당 정보를 수행 시작 전에 알고 있다고 가정한다. 이러한 정보는 다음의 정의 1과 2에서 정의한 예약관계와 충돌 관계에 따른 리스트이며, 이 정보는 자원 요청과 수행에 따른 봉쇄 상한값을 결정짓는데 기본이 된다.

정의 1. 예약관계

임의 자원 r_p에 대해 태스크가 수행 중 언제든 사용하기로 되어 있다면, 이 자원은 태스크에게 예약되어 있다고 하고, 지역 자원 r_p에 예약되어 있는 태스크들의 집합을 $T_{local}^{(p)}$, 전역 자원 r_q에 예약되어 있는 태스크들의 집합을 $T_{global}^{(q)}$로 표기한다.

이러한 예약 관계는 그림 3과 같이 도시해 볼 수 있다.

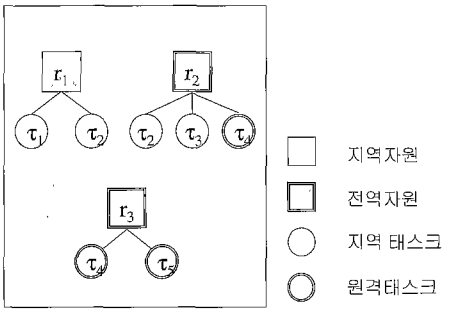

그림 3 A Resource Manager

분산 환경임을 감안한다면, 태스크들은 다른 프로세서에 할당된 자원을 요청하는 경우도 고려하여야 하며, 자원 관리자에 예약되어 있는 자원이 지역 태스크들에 의해 지역적으로 사용되는지 원격 태스크에 의해 전역적으로 사용되는지를 구분하여야 한다. 그림 3 에서 보듯이 한 자원에 대해 지역 태스크에게만 예약되어 있는 경우도 있고, 원격 태스크들에게만, 또는 지역 태스크와 원격 태스크에 동시에 예약되어 있을 수도 있다. 또한, 정의 1로 부터 T_{local}과 T_{global}를 정의할 수 있는데, T_{local}은 적어도 한번 지역적으로 자원을 요청하는 태스크들의 집합이고, T_{global}은 적어도 한번 전역적으로 자원을 요청하는 태스크들의 집합이다. 즉, 시스템의 모든 자원 r_p와 r_q에 대해 $T_{local} = \bigcup_p T_{local}^{(p)}$와 $T_{global} = \bigcup_q T_{local}^{(q)}$이다.

정의 2. 충돌관계

임의의 두 태스크가 공통된 자원에 대해 예약되어 있다면, 이 태스크들은 충돌관계에 있다고 하고, 지역 자원 r_p에 대해 태스크 τ_i와 충돌 관계에 있는 태스크들의 집합을 $\hat{T}_{local}(p, i)$, 전역 자원 r_q에 대해 태스크 τ_j와 충돌 관계에 있는 태스크들의 집합을 $\hat{T}_{global}(q, j)$로 표기하고 다음과 같이 정의할 수 있다.

$$\hat{T}_{local}(p,i) = \{ \tau_i, \tau_k \mid \exists\, \tau_i \in T_{local}^{(p)};\ \tau_k \in T_{local}^{(p)} \}$$

$$\hat{T}_{global}(q,j) = \{ \tau_j, \tau_k \mid \exists\, \tau_j \in T_{global}^{(q)};\ \tau_k \in T_{local}^{(q)} \cup T_{global}^{(q)} \}$$

정의 1과 2에서 임의 프로세서에 속한 태스크 τ_i에 대해 자원이 지역적으로만 사용될 때는 $T_{local}^{(p)}$와 $\bigcup_i \hat{T}_{local}(p, i)$에 속한 태스크들은 같다. 하지만, $T_{global}^{(q)}$와 $\bigcup_i \hat{T}_{global}(q, i)$에 속한 태스크들은 반드시 같다고 볼 수 없는데, 이는 τ_i 와 전역적 충돌 관계에 있는 태스크들 중에는 지역 태스크도 포함될 수 있기 때문이다. 즉, 예약관계와 충돌 관계는 다음과 같은 관계가 있음을 알수 있다.

$$T_{local}^{(p)} = \left\{ \bigcup_i \hat{T}_{local}(p,i) \mid \forall\, \tau_i \in T, \exists\, r_p \in R \right\}$$

$$T_{global}^{(q)} + T_{local}^{(q)} = \left\{ \bigcup_j \hat{T}_{global}(q,j) \mid \forall\, \tau_j \in T, \exists\, r_q \in R \right\}$$

다음은 본 논문에서 사용하는 표기들이다.

$C_{lcs}(p,i)$: 자원 r_p에 대한 태스크 τ_i의 지역 임계구역 수행시간

$C_{gcs}(q,j)$: 자원 r_q에 대한 태스크 τ_j의 전역 임계구역 수행시간

$T_{local}^{(p)}$: 자원 r_p에 대해 지역적으로 예약관계에 있는 태스크들의 집합

$T_{global}^{(q)}$: 자원 r_q에 대해 전역적으로 예약관계에 있는 태스크들의 집합

$\hat{T}_{local}(p, i)$: 자원 r_p에 대해 태스크 τ_i와 지역적 충돌관계에 있는 태스크들의 집합

$\hat{T}_{global}(q, j)$: 자원 r_q에 대해 태스크 τ_i와 전역적 충돌관계에 있는 태스크들의 집합

ϕ_i : 태스크 τ_i의 우선순위

$ceil(r_p)$: 자원 r_p 의 ceiling값

3.2 자원 관리자의 동작

자원관리자는 태스크로부터의 자원 요청이 들어오면, 동작을 시작한다. 먼저 태스크가 요청한 자원을 즉시 허가해 줄 수 있는지 대기시켜야 하는지를 결정한다. 이때 자원 관리자는 두 경우에 대해 태스크가 봉쇄되는 상한값을 계산하여 되돌린다.

경우 1. 태스크에게 자원을 허가하는 경우, 자원관리자는 자원들과 태스크들의 예약관계와 현재 상태에 따라 자원의 임계구역 수행을 완료할 때까지 태스크가 봉쇄되는 상한값을 평가한다.

경우 2. 태스크에게 요청한 자원에 대해 봉쇄시키는 경우, 최대의 봉쇄시간을 예측하고, 자원을 허가 받을 때까지 대기시킨다.

경우 1에서 태스크에게 자원을 허가하는 경우는 현재 프로세서가 idle한 상태이거나 다른 태스크가 수행하고 있는 상태일 수 있다. 전자의 경우는 당연히 요청받은 자원을 허가해 줄 수 있으나, 후자의 경우는 몇 가지 조건에 따라 허가 여부가 다를 수 있다. 이미 수행중인 태스크가 점유하고 있는 자원이 요청한 자원과 같은 경우는 자원을 허가하지 못하고, 요청한 태스크를 봉쇄시킨다. 또한, 점유된 자원이 요청한 자원과 다를 경우라도 자원의 ceiling값과 요청한 태스크의 우선순위 값에 따라서도 자원의 허가 여부가 다르다. 자원을 허가해 줄 수 있는 상태는 다음과 같이 요약해 볼 수 있다.

(i) 점유된 자원과 요청한 자원이 다르고, 점유된 자원의 ceiling값이 요청한 태스크의 우선순위보다 낮은 상태

(ii) 자원을 요청한 태스크가 원격 태스크이고, 요청한 자원이 이미 점유되어 있는 자원과 다른 상태

(ii)는 원격 태스크로부터의 자원 요청시 지역 태스크의 수행이나 사용 중인 지역 자원에 대해 우선적으로 허가받을 수 있음을 의미한다. 단, 이때 발생하는 지연은 자원 관리자에서 그 상한값을 평가할 수 있도록 하여 태스크의 스케줄 시에 적용되도록 한다. 일단 자원을 허가 받아 수행에 들어가면 자원의 ceiling 값을 이 프로세서에서 수행할 지역 태스크들 중 가장 높은 우선순위보다 큰 값으로 재 설정한다.

경우 2는 자원 허가를 받지 못하고 봉쇄 상태에 들어가는 경우로, 다음의 두 가지 상태이다.

(i)′ 프로세서 내의 어떤 전역 자원이 다른 원격 태스크에 의해 점유되어 수행되고 있는 상태

(ii)′ 요청한 자원이 다른 태스크에 의해 점유되어 있는 상태

위의 (i)′과 (ii)′ 모두 봉쇄상태로 들어가면 자원관리자는 자원을 봉쇄시키고 수행중인 자원의 우선순위를 요청한 태스크의 실행 우선순위로 재 설정한다. 이렇게 함으로써 봉쇄시키는 자원이 종료되는 즉시 요청한 자원을 수행할 수 있도록 한다.

자원 관리자로 자원 요청이 도착했을 때, 자원 관리자가 측정해서 스케줄러로 되돌리는 시간은 크게 두 가지로 구분할 수 있는데, 경우 1에서처럼 요청한 자원을 허가 받아 수행을 시작했을 때, 수행을 마치기 전까지 수행 도중에 봉쇄되는 시간과 경우 2의 이유로 요청한 자원을 바로 허가 받지 못하고 임계구역의 수행에 들어가기 전까지 봉쇄되는 시간이다. 편의 상 전자를 내부적 봉쇄라 하고 후자를 외부적 봉쇄라 한다. 본 논문의 목적은 이러한 내부적 봉쇄와 외부적 봉쇄의 상한값을 구하는 것으로 결과적으로 이 두 형태의 봉쇄 구간을 합한 값을 자원에 대한 예측 가능한 봉쇄시간으로 설정하는 것이다.

4. 봉쇄시간 상한값의 분석

태스크로부터 자원에 대한 요청이 도착하면 현재 예약되어 있는 자원과 태스크들의 상태에 대한 정보를 가지고 허가를 할지 봉쇄시킬지를 결정한다. 두 경우 모두 자원 관리자는 요청한 자원의 임계구역 수행을 종료하기까지 봉쇄되는 상한값을 계산할 수 있으며, 이때, 태스크가 지역인지 원격인지, 또한 요청한 자원이 지역인지 전역인지에 따라 다르게 평가된다. 이 절에서는 각 경우에 대한 봉쇄시간 상한값을 보다 면밀하게 평가해 보기로 한다.

4.1 지역 태스크의 경우

자원을 요청한 태스크가 지역 태스크인 경우에 계산되는 경우이다. 구하고자 하는 봉쇄시간의 주요 요인은 태스크를 봉쇄시키는 자원의 임계구역 시간이다. 임계구역 시간은 지역자원의 경우와 전역자원의 경우를 구별하여야 하며, 태스크 τ_i의 지역자원 r_p의 임계구역 수행시간을 $C_{lcs}(p,i)$로, 전역자원 r_q의 임계구역 수행시간을 $C_{gcs}(q,i)$로 표기하기로 한다. 또한, 각 태스크 τ_i의 우선순위는 ϕ_i로 표기하고, $i < j$ 일 때, 태스크 τ_i의 우선순위가 태스크 τ_j의 우선순위보다 높다고 한다. 자원 r_p의 ceiling 값은 $ceil(r_p)$로 표기한다.

보조정리 1. 지역 태스크 τ_i는 자원 r_p를 요청하여 임계구역을 수행 중일 때, 모든 지역 태스크 $\tau_k \in \{\bigcup_j \hat{T}_{local}(q,j) \mid \forall \tau_j \in T, \exists r_q \in R; \phi_j > \phi_i, r_q \neq r_p\}$에 의해 최대 $\sum_{\tau_k} C_{lcs}(q,k)\ \varepsilon_k$ 동안 봉쇄될 수 있다. 이때 $ceil(r_p) \geq \phi_k$ 이면 $\varepsilon_k = 0$이고 $ceil(r_p) < \phi_k$ 이면 $\varepsilon_k = 1$이다.

증명 태스크 τ_i가 자원 r_p에 대한 lock을 갖고 있지만 다른 지역 태스크에 의해 봉쇄되는 경우는 임의의 새로운 태스크 τ_k가 도착하여 τ_i를 선점하여 자원을 요청하는 경우이다. 따라서 태스크 τ_k의 우선순위는 τ_i보다 커야 한다. 즉, $\phi_k > \phi_i$ 이어야 한다. 이때, 선점하는 태스크 τ_k가 요청한 임의 자원 r_q에 대해 허가받아 임계구역에 들어가면 τ_i는 이 구간동안 봉쇄된다. PCP 원칙에 따른다는 가정에 의해 태스크 τ_k가 요청한 자원을 허가 받으려면, (i)요청한 자원이 현재 lock이 걸려있는

r_p가 아니어야 하고, (ii)τ_k의 우선순위가 자원 r_p의 ceiling값보다 커야한다. 따라서, (i)의 조건을 위해, 자원 r_p와 충돌관계에 있는 태스크들을 제외한 다른 자원 r_q에 대해 지역적 충돌 관계에 있는 태스크들 τ_k의 지역적 자원 임계구역 수행시간의 합이어야 하고, (ii)의 조건을 위해, τ_k는 r_p의 ceiling값에 따라 선택적이어야 한다. 이들 조건을 모두 고려할 때, 위의 봉쇄 상한값은 타당하다. ■

위의 보조정리에서 태스크 τ_k는 태스크 τ_i를 선점하지만 수행 도중 한번도 자원을 요청하지 않고 수행을 마치는 경우를 생각해 볼 수도 있다. 이 경우는 자원의 임계구역 수행으로 인한 봉쇄이기보다는 태스크 선점으로 인한 지연으로 볼 수 있으며, 또한, 임계구역에 들어갔다고 해도 임계구역 수행을 완료한 후 나머지 실행으로 인한 지연도 같은 이유로 지연될 수 있다. 자원 관리자는 관리되는 자원에 대해 요청해온 경우에만 취급하므로, 이러한 지연시간에 대한 정보를 알 수 없다. 본 논문에서 이 값은 봉쇄시간에 포함시키지 않으며, 단지 자원에 대한 충돌 관계에 의해 결정지어지는 봉쇄 구간의 상한값으로 제한하여 논한다.

보조정리 2. 지역 태스크 τ_i는 지역 자원 r_p을 요청하여 임계구역을 수행 중일 때, 모든 원격 태스크 $\tau_k \in T_{global}$와 τ_k가 요청하는 자원 r_q에 의해 최대 $\sum_{\tau_k} C_{gcs}(q,k)$ 동안 봉쇄될 수 있다.

증명 태스크 τ_i가 사용 중인 자원 r_p는 지역 자원이므로, 다른 원격 태스크와 충돌관계에 있지 않다. 즉, 새로 도착하는 원격 태스크 τ_k는 r_p를 요청하지 않는 것은 분명하다. 또한, 정의 3에 의해 자원 관리자는 지역 태스크가 수행 중일 때 도착하는 모든 원격 태스크들의 자원 요청을 우선적으로 처리한다. 다시 말하면, r_p의 수행이 완료된 직후 다른 지역 태스크가 도착해도 대기 중인 원격 태스크가 요청한 자원을 허가 받을 수 있다. 즉, 우선순위와 관계없이 모든 지역 태스크는 도착하는 원격 태스크들에 의해 선점된다. 이렇게 도착하는 원격 태스크들은 T_{global}에 포함되는 태스크들 중 하나이고, 최대 봉쇄시간은 이 태스크들의 전역 자원 임계구역 $C_{gcs}(q,k)$ 의 합이다. ■

보조정리 3. 지역 태스크 τ_i는 전역자원 r_p를 요청하여 임계구역을 수행 중일 때, 모든 원격 태스크 $\tau_k \in \left\{ \bigcup_j \hat{T}_{global}(q,j) \mid \forall \tau_j \in T,\ \exists r_q \in R;\ r_q \neq r_p \right\}$에 의해 최대 $\sum_{\tau_k} C_{gcs}(q,k)$ 동안 봉쇄될 수 있다.

증명 지역 태스크 τ_i가 r_p의 임계구역을 수행하는 도중, 도착하는 원격 태스크의 자원 요청은 요청한 자원에 따라 선점 여부가 다르다. 요청한 자원이 이미 점유되어 있는 자원 r_p라면 선점할 수 없으므로 이는 무시된다. 하지만 다른 전역자원에 대한 요청은 다른 지역 태스크들에 우선하므로 지역 태스크 τ_i의 자원 임계구역 수행을 선점할 수 있다. 따라서 이때 봉쇄되는 시간은 그 자원의 전역 임계구역 시간이다. 즉, 선점하는 태스크들은 r_p를 제외한 다른 전역자원 r_q와 충돌관계에 있는 태스크들의 집합 $\bigcup_j \hat{T}_{global}(q,j)$에 속한다. 따라서 이들 태스크들의 전역 자원 임계구역 동안의 합이 최대 봉쇄시간이 된다. ■

정리 1. 지역 태스크 τ_i가 자원 r_p를 요청하여 임계구역을 수행 중일 때 태스크 τ_i는 임계구역 수행을 완료할 때까지 모든 $\tau_j \in \left\{ \bigcup_l \hat{T}_{local}(q,l) \mid \forall \tau_l \in T,\ \exists r_q \in R;\ \phi_l > \phi_i,\ r_q \neq r_p \right\}$와 τ_k 에 대해, 최대 다음의 상한값 동안 봉쇄될 수 있다. 단, r_p가 지역자원이면, $\tau_k \in T_{global}$, r_p가 전역자원이면 $\tau_k \in \left\{ \bigcup_m \hat{T}_{global}(r,m) \mid \forall \tau_m \in T, \exists r_r \in R;\ r_q \neq r_p \right\}$ 이다.

$$\sum_{\tau_j} C_{lcs}(q,j)\ \varepsilon_j + \sum_{\tau_k} C_{gcs}(r,k)$$

이때, $ceil(r_p) \geq \phi_j$ 이면 $\varepsilon_j = 0$이고 $ceil(r_p) < \phi_j$ 이면 $\varepsilon_j = 1$이다.

증명 지역 태스크 τ_i가 자원 r_p을 요청하면, 자원 관리자는 r_p가 지역자원인지, 전역자원인지에 따라 다른 봉쇄 상한값을 계산해야 한다. (i) 먼저 r_p가 지역자원인 경우, 다른 지역 태스크가 도착하는 경우에 대한 상한값은 보조정리 1에서 증명되었고, 원격 태스크로 인한 상한값은 보조정리 2에서 증명되었으므로 이 두 상한값의 합으로 나타낼 수 있다. (ii) 또한, r_p가 전역자원인 경우, 다른 지역 태스크의 도착에 의한 상한값은 보조정리 1에 나타낸 바와 같이 지역 태스크인 경우와 같은 상한값임을 알 수 있다. 원격 태스크로 인한 봉쇄 상한값은 보조정리 3에서 증명되었으므로 이 두 값의 합이 r_p가 전역자원인 경우의 상한값으로 나타낼 수 있다. ■

이 정리는 지역 태스크가 자원의 임계구역을 수행하는 도중에 발생할 수 있는 내부적 봉쇄에 관한 것이다. 지역 태스크는 요청한 자원이 지역적으로 허용되는 자원인지 전역적으로 허용되는 자원인지에 따라 봉쇄 상한값이 다르게 결정된다.

정리 2. 지역 태스크 τ_i는 자원 r_p를 요청하였으나, lock을 얻지 못하고 봉쇄될 때, 임계구역에 들어가기 전에 τ_i보다 낮은 우선순위를 갖는 모든 테스크 τ_k, $\tau_k \in T_{local} - \{\tau_i\}$와 τ_k가 요청하는 자원 r_q에 의해 최

대 $\max(C_{lcs}(q,k))$동안 봉쇄될 수 있다.

증명 (i) 요청한 자원이 지역자원이라면, 요청한 자원이 이미 다른 태스크에 의해 lock이 걸려 있거나 PCP의 조건을 만족시키지 못하는 경우, 즉, 이 태스크의 우선순위가 현재 lock이 걸려 있는 자원의 ceiling보다 높지 않은 경우이다. 이는 참고문헌 [8]의 정리 2 에서 밝힌 대로 전형적인 우선순위 역전의 경우이다. 따라서 PCP 법칙에 의해서 낮은 우선순위 지역 태스크의 가장 긴 지역 임계구역 동안이다. (ii) 요청한 자원이 전역자원이라면, lock을 갖고 있는 태스크가 지역 태스크인 경우와 원격 태스크인 경우 두 가지로 볼 수 있다. 지역 태스크라면, 우선순위 원칙에 따라 요청한 테스크보다 낮은 우선순위 태스크인 경우이므로 이는 (i)과 같은 경우로, 같은 시간 동안 봉쇄될 수 있다. 하지만, 원격태스크인 경우는 요청하는 지역 태스크의 수행자체가 보류되므로 이런 상황은 발생하지 않는다. 즉, 고려하지 않아도 된다. (i)과 (ii)로 부터 요청한 자원이 지역이든 전역이든, PCP 원칙을 그대로 적용할 수 있으므로, 정리에 의해 가장 긴 낮은 우선순위 태스크의 자원 임계구역 시간이다. ■

정리 2의 상한값은 MPCP에서 τ_i를 봉쇄시키는 가장 긴 임계구역 시간으로 정의한 값과 같다[8]. 본 논문에서는 기본 프로토콜로 PCP를 사용하므로 정리 2에서와 같이 지역 태스크의 봉쇄 상한값은 MPCP에서 증명된 값과 같다. 본 논문에서는 이처럼 임계구역에 들어가기 직전에 봉쇄되는 경우를 외부적 봉쇄라 하였으며, 위의 정의 2는 이러한 지역 태스크의 외부적 봉쇄시간 상한값으로 볼 수 있다. 따라서, 정리 1과 정리 2로 부터 지역 태스크가 자원을 요청했을 때, 봉쇄되는 최대 시간은 정리 1의 내부적 봉쇄시간과 정리 2의 외부적 봉쇄시간의 합이다

4.2 원격 태스크의 경우

자원을 요청한 태스크가 다른 프로세서에 할당된 태스크인 경우이다. 이때, 원격 태스크의 자원요청은 모든 지역 태스크들에 우선하도록 함으로써, 원격봉쇄로 인한 문제를 최소화한다.

정리 3. 원격 태스크 τ_i는 전역 자원 r_p의 임계구역 수행을 시작해서 완료할 때까지 모든 태스크 $\tau_k \in \{\bigcup_q T_{global}(q) | \forall r_q \in R; r_q \neq r_p\}$에 대해 최대 $\sum_{\tau_j} C_{gcs}(q,k)\,\varepsilon_k$ 동안 봉쇄된다. 이때, $ceil(r_p) \geq \phi_k$ 이면 $\varepsilon_k = 0$이고 $ceil(r_p) < \phi_k$ 이면 $\varepsilon_k = 1$이다.

증명 먼저, (i) 원격 태스크 τ_i가 일단 전역자원 r_p를 허가 받아 사용 중이면, 정의 3의 원격 태스크 우선 원칙에 따라 다른 지역 태스크들에 의해 중지되는 일은 없다. 이는 r_p가 아닌 다른 자원에 대한 지역 태스크의 요청에 대해서도 같다. 따라서 어떠한 지역 태스크의 도착이나 자원 요청으로 인한 봉쇄는 고려할 필요가 없다. (ii) 하지만, 다른 원격 태스크로부터의 자원 요청에 의한 중지는 고려해야 한다. 이때, 다른 원격 태스크가 요청한 자원이 이미 점유된 r_p이면 이 태스크는 r_p가 해제될 때까지 기다려야 하며, τ_i의 봉쇄시간에 영향을 주지 않는다. 따라서, r_p에 대해 전역적으로 충돌관계에 있는 태스크들은 배제되고, r_p가 아닌 다른 자원을 요청하는 원격 태스크의 경우만 고려하면 된다. 따라서 (i)과 (ii)의 두 경우를 만족하기 위해서는 r_p가 아닌 임의의 자원 r_q에 전역적으로 예약되어 있는 태스크들의 임계구역 수행시간을 고려해야 한다. 이 태스크들은 자원 r_p가 아닌 모든 자원 r_q에 대해 $\bigcup_q T_{global}^{(q)}$ 에 속한 태스크들로 볼 수 있고, 이 태스크들의 전역 임계구역 수행시간의 합이 봉쇄시간의 상한값이 된다. 이때, PCP원칙에 의해 rq를 요청한 임의의 원격 태스크 τj의 우선순위 ϕk는 점유되어 있는 자원 rp의 ceiling값보다 높아야 한다. ■

이 정리는 원격 태스크가 자원의 임계구역을 수행하는 도중 발생할 수 있는 내부적 봉쇄에 관한 것이다. 원격 태스크는 모든 지역 태스크에 우선하므로 봉쇄를 일으키는 요인은 다른 원격 태스크들로 인한 것이다.

정리 4. 원격 태스크 τ_i는 전역자원 r_p를 요청하였으나 허가받지 못할 때, 임계구역에 들어가기 전에 모든 $\tau_j \in T_{local}^{(p)}$, $\tau_k \in T_{global}^{(p)} - \{\tau_i\}$. $\tau_l \in T_{global} - \{\tau_i\}$에 대해 최대 $\max(C_{lcs}(p,j), C_{gcs}(p,k)) + \sum_{\tau_l} C_{gcs}(q,l)$ 동안 봉쇄될 수 있다.

증명 전역자원 r_p를 요청한 원격 태스크 τ_i가 봉쇄되는 상태는 두 가지로 볼 수 있다. 먼저, (i) 요청한 자원 r_p가 다른 태스크에 의해 점유되어 있는 경우와 (ii) r_p가 어떤 태스크에 의해서도 점유되어 있지 않음에도 허가 받지 못하는 경우이다. (i)의 경우부터 먼저 살펴보면, r_p를 점유하고 있는 태스크는 지역 태스크일 수도, 전역 태스크일 수도 있다. r_p를 점유한 태스크가 지역 태스크 τ_j라고 할 때, τ_i가 이 태스크에 의해 봉쇄되는 시간은 많아야 τ_j의 r_p에 대한 지역적 임계구역 수행시간 중 가장 긴 시간이다. 즉, $\tau_j \in T_{local}^{(p)}$ 이면, $max(C_{lcs}(p,j))$ 시간만큼 봉쇄된다. 이때, τ_i가 지역 태스크에 의해 봉쇄되는 경우는 단 한번의 이 시간 동안이다. 왜냐하면, τ_i는 원격 태스크이고, 정의 3에 의해 원격 태스크는 이후에 도착하는 다른 지역 태스크들 보다

우선하므로 τ_i이 자원의 임계구역을 종료할 때까지 어떠한 지역 태스크들의 도착은 지연되기 때문이다. 하지만, 만일 r_p를 점유한 태스크가 전역 태스크 τ_k라고 한다면, τ_k는 τ_i를 제외하고 자원 r_p에 대해 전역적으로 예약되어 있는 태스크들 중 하나이고, 많아야 가장 긴 태스크의 전역적 임계구역 수행시간 동안 봉쇄된다. 즉, $\tau_k \in T_{global}^{(p)} - \{\tau_i\}$ 이면, $max(C_{gcs}(p,k))$ 시간만큼 봉쇄된다. 따라서, 자원 r_p가 이미 다른 태스크에 의해 점유되어 있을 때, τ_i가 이로 인해 봉쇄되는 최대 시간은 위의 지역 태스크인 경우와 전역 태스크인 경우 중 가장 긴 시간동안이다. 또한, (ii)의 경우는 τ_i가 r_p를 요청했을 때, r_p가 free한 상태일지라도 다른 원격 태스크가 r_p 가 아닌 다른 전역 자원의 임계구역을 수행하고 있는 도중으로 PCP의 요건을 충족하지 못하거나, 다른 원격 태스크와 자원에 대해 충돌하는 상태인 경우이다. 이 경우는 프로세서 내에 예약된 모든 원격 태스크들의 전역적 임계구역 수행시간의 합으로 최대 봉쇄시간을 설정해야 한다. 따라서 위의 (i)과 (ii)의 두 경우를 고려할 때, 이 상한값은 타당하다. ■

이 정리는 원격 태스크가 자원을 요청했으나 허가 받지 못하고 봉쇄되는 경우로, 자원의 임계구역 수행에 들어가기 전에 봉쇄되는 외부적 봉쇄에 관한 것이다. 결과적으로 원격 태스크가 자원을 요청했을 때, 봉쇄되는 최대 시간은 정리 3의 내부적 봉쇄시간과 정리 4의 외부적 봉쇄시간의 합으로 볼 수 있다.

5. 분석 예제

자원 관리자에서 계산되는 봉쇄 시간의 상한값은 태스크의 특성과 자원 요청을 하는 시점의 상태에 따라 선별적으로 계산된다는 점이 기존 방식과의 차이이다. 본 절에서는 본 논문의 결과를 검증하기 위해 다음과 같은 예제를 들어 기존 방식과 본 논문의 결과를 비교해 본다.

예제 1. 문헌 [8]의 예제 3.4의 예

예제 2. 자원 관리자 방식을 위한 임의의 할당 예

예제 3. 예제 2를 편중되게 할당한 예

예제 1은 [8]의 MPCP를 사용한 결과의 봉쇄시간과 본 논문의 자원 관리자 (이하 RM으로 표기)를 이용한 결과를 비교해 보기 위한 것으로, 분석을 위해 필요한 태스크의 주기는 그대로 이용하였으나, 지역 자원 임계구역과 전역 자원 임계구역 수행 시간은 편의상 모두 1로 하였으며, 다른 예제 2와 3에도 마찬가지로 적용하였다. 예제 2와 3은 본 논문에서의 RM의 사용 예를 살펴보기 위해 임의로 할당한 예로, 두 경우 모두 MPCP를 사용한 경우의 봉쇄시간도 분석하여, 그 차이를 분석하였다. 예제 2는 비교적 지역자원과 전역 자원의 할당이 고르게 분포된 예로써, 자원의 수를 7로 증가시키고 두 프로세서 P_1 와 P_3 는 지역 태스크와 원격 태스크 모두, P_2는 지역 태스크에 의해서만 접근되도록 할당하였다. 반면에, 예제 3은 예제 2를 다소 편중되게 할당한 예로써, P_1에서는 지역적 접근만 발생하도록 하였고, P_2와 P_3는 원격 자원에 의한 접근만 발생하도록 할당하였다. 또한, 비교를 위해 각 예에서 태스크의 수는 동일하게 하며, 각 예제는 세 가지 경우로 구분하여 살펴보는데, (a) MPCP에서 동기 프로세서를 사용한 경우, (b) (a)의 할당에 RM방식을 적용한 예, 그리고 (c) RM 방식을 적용하기 위해 임의로 할당한 예이다. 이는 본 논문의 RM방식에서는 MPCP에서와 같이 별도의 동기 프로세서를 두지 않기 때문이며, (b)의 경우는 동기 프로세서를 임의의 한 프로세서로 간주하고 적용하였다. 따라서 비교 결과는 하나의 예에 대해 3가지 형태의 봉쇄 상한값이 평가된다. 각 예제의 자원과 태스크들의 할당은 표 1과 같다.

예제 1에서 각 태스크가 접근하는 자원들의 목록은 τ_1 (r_3, r_1), τ_2(r_3, r_2), τ_3(r_1, r_4), τ_4(r_2, r_4), τ_5(r_3, r_5), τ_6(r_4, r_5)이라고 하고, 예제 2에서 각 태스크가 접근하는 자원들의 목록은 τ_1(r_4, r_1), τ_2(r_4, r_5 ,r_2), τ_3(r_1, r_5, r_6), τ_4(r_3, r_2, r_6), τ_5(r_5, r_7), τ_6(r_7, r_6)이라고 한다. 예제 3은 예제 2를 다소 편중되게 재 할당한 것으로 자원들의 목록은 예제 2의 예를 그대로 따르도록 한다.

표 1 각 예제의 태스크와 자원의 할당 예

프로세서	예제 1		예제 2		예제 3
	(a),(b)	(c)	(a),(b)	(c)	(c)
P_1	τ_1, τ_3 (r_1)	τ_1, τ_2 (r_1, r_3)	τ_1, τ_3 (r_1)	τ_1, τ_3 (r_1, r_4)	$\tau_1, \tau_2, \tau_3, \tau_4, \tau_5, \tau_6$ (r_1, r_5, r_6)
P_2	τ_2, τ_4 (r_2)	τ_3, τ_4 (r_2, r_4)	τ_2, τ_4 (r_2, r_3)	τ_2, τ_4 (r_2, r_3)	none (r_2, r_3)
P_3	τ_5, τ_6 (r_3, r_4, r_5)	τ_5, τ_6 (r_5)	τ_5, τ_6 (r_4, r_5, r_6, r_7)	τ_5, τ_6 (r_5, r_6, r_7)	none (r_4, r_7)

MPCP에서는 태스크의 봉쇄 요인을 네 가지로 구분하여 이들의 합으로 상한값을 계산하지만, 본 논문의 자원 관리자는 두 가지로 구분하여 합계를 계산한다. 즉, 자원의 요청 시 다른 지역 태스크들로 인한 봉쇄요인과 다른 원격 태스크들로 인한 봉쇄요인이다. 또한, 어떤 태스크의 경우 할당 상태에 따라서 지역이든 전역이든

두 번 이상 접근이 발생할 수도 있는데, 이런 경우는 이 값들 중 최대값을 그 봉쇄시간으로 하였다. 이들 각 예제들로부터 평가되는 각 경우의 봉쇄시간 상한값은 각 그림 4, 5에 나타내었다.

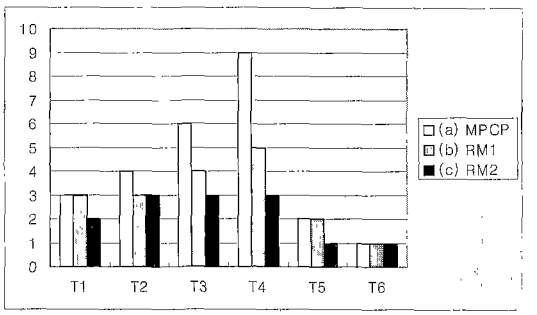

그림 4 예제 1의 비교 결과

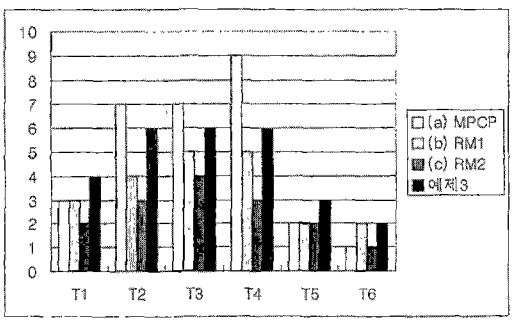

그림 5 예제 2와 예제 3의 비교 결과

먼저, MPCP의 경우와 예제 1의 결과를 보면, 전반적으로 자원 관리자를 사용하는 경우 봉쇄시간이 다소 감소되었음을 볼 수 있다. 특히 전역 자원에 접근하려는 태스크들 τ_1, τ_2, τ_3, 그리고, τ_4의 경우 지역적 접근과 전역적 접근의 경우 모두 고르게 감소하였는데, 이는 자원 관리자가 선택적으로, 봉쇄시간에 영향을 끼칠 수 있는 태스크들만을 봉쇄시간 계산에 포함시켰기 때문이며, τ_5와 τ_6은 모두 지역적 접근의 경우만 봉쇄된 예로 MPCP와 비교해 볼 때, 봉쇄시간이 크게 증가하지 않았다. 실제로, 이 두 태스크는 다른 태스크의 선점으로 인한 지연도 지나치게 크지 않음을 알 수 있었다. 이는 MPCP의 경우 동기 프로세서에 강제적으로 할당시키는 낮은 우선순위 태스크가 지나치게 지연되는 단점이 개선된 것으로 본다. 예제 2와 예제 3은 [8]의 예제와는 무관하게 본 논문의 자원 관리자를 사용한 경우를 보인다. 예제 2의 결과는 비교적 고르게 자원과 태스크들을 할당한 것으로, 각 경우의 봉쇄시간이 작은 값으로 나타났지만, 예제 3의 결과는 봉쇄시간이 다소 증가하였다. 예제 3은 편중된 예를 들기 위해 고의로 할당시킨 것으로, 실제로는 발생하지 않지만, 이러한 경우라도 봉쇄시간이 지나치게 증가되지 않음을 알 수 있다. 일반적으로 자원과 태스크의 할당은 예제 1과 예제 2와 같이 지역 태스크와 전역 태스크가 비슷한 비율로 할당된다고 보며, 이들 경우 자원 관리자를 이용할 때, 다소 감소된 봉쇄시간을 예측할 수 있다.

6. 결론 및 향후 연구과제

본 논문에서는 분산 실시간 환경에서 자원 제어를 위한 동기화 방법으로 자원 관리자를 제안한다. 자원 관리자는 태스크로부터 자원 요청이 도착했을 때 자원의 상태에 따라 자원의 허가 여부를 결정하고 태스크가 요청한 자원으로 인한 지연 등을 평가한다. 자원 관리자의 주요 목적은 관리하고 있는 태스크들과 자원들의 예약 관계와 이들간의 충돌관계 등을 조사하여 다른 자원의 점유로 인한 봉쇄시간 등을 예측하고, 이러한 봉쇄시간의 상한값을 결정하는 것이다. 또한, 본 논문의 자원 관리자를 사용함으로써, 전역자원만을 위한 특별한 프로세서를 두지 않고도, 태스크들과 자원을 융통성 있게 할당할 수 있고, 이러한 할당된 자원의 상태에 따라 봉쇄시간을 선택적으로 계산할 수 있다. 또한, 본 논문에서는 자원 관리자에서 계산되는 각 경우의 봉쇄시간의 타당성을 수학적으로 증명하였으며, 실제 예를 통해 기존 연구인 MPCP의 봉쇄시간 상한값을 비교하였다. 그 결과로 지나치게 큰 값으로 설정된 요인을 제거함으로써, 상한값을 감소시킬 수 있었으며, 낮은 우선순위 태스크의 지나친 지연시간을 상당 부분 감소시킬 수 있었다. 제안하는 자원 관리 기법은 멀티미디어, 분산된 시뮬레이션, 분산된 데이타베이스 등의 실시간 응용 문제에서 end-to-end 시간제약을 만족시키는 스케줄링 문제에 자원의 제약 조건을 해결하는데 사용될 수 있을 것으로 보며, 이는 기존의 MPCP와 rate monotonic 접근법이 갖고 있는 분산환경에서의 메시지 스케줄링에 대한 제약점을 극복할 것으로 본다.

본 논문의 자원 제어 기법은 몇 가지 제약점을 가지고 있다. 첫 번째는 2.2절에서와 같은 원격 봉쇄를 제거하기 위해 전역자원의 실행 우선순위를 ceiling값과 같도록 함으로써 초래될 수 있는 문제점이다. 이는 원칙적으로 PCP의 우선순위 상속방법을 사용하는 것과 차이가 없지만, 실질적으로 높은 우선순위 태스크의 수행을 필요이상 지연시킬 수 있다. 특히, 전역자원의 임계구역 구간이 긴 경우나 원격 요청이 빈번한 경우는 심각해

질 수 있다. 하지만, 실질적으로 전체 태스크의 수행시간에 대한 임계구역의 구간은 길지 않으므로, 이는 큰 제약이 될 수 없을 것으로 보지만, 보다 일반성을 갖기 위해서는 이러한 제약은 개선되어야 한다. 또 하나는 자원 관리자에 의해 구해지는 봉쇄 상한값을 태스크들의 스케줄 가능성 분석의 한 요인으로 적용하는 문제로의 확장이다. MPCP의 경우 rate monotonic 스케줄링 방법에 적용하고 있으나, 본 논문에서 제안한 자원 관리자는 보다 동적 환경에 적합한 스케줄링 방법에 적용 가능할 것으로 보며, 이 부분은 더 연구되어야 할 부분이다.

참 고 문 헌

[1] John P. Lehoczky, Lui Sha, J. K. Strosnider and Hide Tokuda, "Fixed priority Scheduling Theory for Hard Real-Time Systems," *Foundations of real-time computing : Scheduling and Resource Management,* KAP, 1991.

[2] Neil C. Audsley, "Resource control for hard real-time systems: A review," Technical report YCS_91-159, department of Computer Science, University of York, 1991

[3] A. K. L. Mok, "Fundamental design problems of distributed Systems for the hard real-time system environment," MIT/LCS/TR-297, Laboratory of Computer Science, Massachusetts Institute of Technology (1983)

[4] Lui Sha, Ragunathan Rajkumar, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," *IEEE Trans. on Software Engineering,* vol.39, pp.1,175-1,185, Sept. 1990.

[5] Ragunathan Rajkumar, Lui Sha, John P. Lehoczky, and Krithi Ramamrithan, "An Optimal Priority Inheritance Policy For Synchronization in Real-Time Systems," *Advanced in real-time Systems,* Prentice Hall, 1995.

[6] Akira Nakamura, "An Investigation of Real-Time Synchronization," PhD. Thesis, Computer Laboratory, Wolfson College of University of Cambridge, Dec. 1993.

[7] P.G. Jansen and F. Gausevles, "Priority Inheritance in real-Time Micro Kernels," International Symp. on Comp. and Inf. Sciences VII, Antalya, Turkey, Nov. 1992.

[8] Ragunathan Rajkumar, *Synchronization in Real-Time Systems : A Priority Inheritane Approach,* KAP, 1991.

이 은 미

1989년 한양대학교 수학과 학사. 1991년 한양대학교 전자계산학 석사. 현재 한양대학교 전자계산학 박사과정. 관심분야는 분산처리 시스템, 실시간 운영체제, 실시간 통신

허 신

1973년 서울대학교 전기공학과 졸업(공학사). 1979년 Univ. of Southern California 전자계산학 석사학위 취득. 1986년 Univ. of South Florida 전자계산학 박사학위 취득. 1986년 ~ 1988년 The Catholic University of America 조교수. 1988년 ~ 현재 한양대학교 교수. 관심분야는 분산처리 시스템, 결함허용 시스템, 실시간 운영체제.