동적 우선순위 상한 프로토콜의 스케줄링 가능성 조건 개선

(An Improvement of the Schedulability Condition in Dynamic Priority Ceiling Protocol)

오 성 흔 † 양 승 민 ††
(Sung-Heun Oh) (Seung-Min Yang)

요 약 실시간 시스템에서 태스크가 공유 자원을 사용할 때 제한되지 않은 우선순위 역전이 발생할 수 있다. 이는 실시간 태스크의 스케줄링 가능성 보장을 불가능하게 한다. 따라서 우선순위 역전을 제한하는 자원 접근 프로토콜들이 연구되었으며 주기적 태스크 집합의 스케줄링 가능성 분석을 위한 충분조건이 제시되었다.

본 논문에서는 동적 우선순위 상한 프로토콜을 사용할 경우 개선된 스케줄링 가능성 충분조건을 제시한다. 제시된 방법에서는 높은 우선순위의 태스크들의 마감시간을 놓치지 않는 범위 내에서 낮은 우선순위를 가진 태스크가 계속 수행될 수 있다는 사실을 이용하였다. 이렇게 함으로써 높은 우선순위를 가진 태스크가 낮은 우선순위를 가진 태스크를 위해서 허용해 줄 수 있는 시간은 높은 우선순위를 가진 태스크의 최악 블록킹 시간에서 제외될 수 있다. 태스크의 최악 블록킹 시간이 감소하게 되므로 동적 우선순위 상한 프로토콜의 스케줄링 가능성 충분조건은 개선된다.

Abstract When tasks access shared resources in real-time systems, the unbounded priority inversion may occur. In such cases it is impossible to guarantee the schedulability of real-time tasks. Several resource access protocols have been proposed to bound the duration of priority inversion and sufficient conditions are given to guarantee the schedulability of periodic task set.

In this paper, we show an improved sufficient condition for schedulability when the dynamic priority ceiling protocol is used. Our approach exploits the fact that a lower priority task can continue to execute as far as the higher priority tasks do not miss their deadlines. This permitting execution time of the higher priority tasks for a lower priority task can be excluded from the worst-case blocking time of the higher priority tasks. Since the worst-case blocking time of tasks can be reduced, the sufficient condition for schedulability of dynamic priority ceiling protocol becomes further tight.

1. 서 론

실시간 시스템은 시스템의 정확성이 논리적 정확성 뿐만 아니라 결과를 마감시간에 맞추는 시간적 정확성에 의존하는 시스템이다. 실시간 시스템 내의 태스크들이 마감시간 내에 수행되지 못한다면 태스크 자체의 쓸모가 사라지거나 시스템 고장이 발생할 수 있다. 그러므로 태스크들이 마감시간 내에 수행될 수 있도록 적절한 실시간 스케줄링 기법을 사용해야 하며 주어진 태스크들이 이 스케줄링 기법으로 스케줄링 가능한지 시스템 설계 시 분석 하게 된다.

실시간 태스크들을 스케줄링 하기 위한 한가지 방법으로 각 태스크의 우선순위에 기반하여 스케줄링 하는 것이 있다. 우선순위 기반 스케줄링 기법에서는 가장 높은 우선순위를 가진 태스크를 수행시킨다. 우선순위 기반 스케줄링 기법은 우선 순위를 주는 방법에 따라 비율단조(Rate-Monotonic 또는 RM) 스케줄링 알고리즘과 같은 정적 우선순위 스케줄링 방법과 마감시간 우선

· 본 연구는 한국과학재단 특정기초 연구과제지원(과제번호 1999-1-303-006-3)으로 수행되었음.
† 학생회원 숭실대학교 컴퓨터학과
honestly@realtime.soongsil.ac.kr
†† 종신회원 · 숭실대학교 컴퓨터학과 교수
yang@computing.soongsil.ac.kr
논문접수 : 2001년 1월 15일
심사완료 : 2001년 8월 28일

(Earliest-Deadline-First 또는 EDF) 스케줄링 방법과 최소 여유시간 우선(Least-Laxity-First 또는 LLF) 스케줄링 방법과 같은 동적 우선순위 스케줄링 방법으로 나뉜다[1, 2, 3, 4].

이 스케줄링 관련 연구에서는 한 태스크의 수행은 다른 태스크의 수행에 의존하지 않고 독립적이라는 가정을 하고 있다. 그러나 대부분의 시스템에서 태스크들은 공동의 목적을 달성하기 위해 서로의 협동을 요구하므로 실제로는 서로 독립적이지 않다. 태스크들은 프로세서를 제외한 다른 하드웨어 장치들을 서로 공유하기도 하며 공유 자료들을 이용하여 서로의 수행에 직접적으로 또는 간접적으로 영향을 미친다. 이렇게 공유되는 개체들을 일반적으로 자원(Resource)이라고 한다. 대부분의 자원은 한번에 한 태스크에 의해서만 사용할 수 있도록 상호배제(Mutually Exclusive) 적인 접근을 요구한다. 자원을 서로 공유하게 됨에 따라 높은 우선순위를 가진 태스크가 낮은 우선순위를 가진 태스크에 의해서 수행이 중지되는 우선순위 역전(Priority Inversion) 현상이 발생할 수 있다. 높은 우선순위를 가진 태스크 T_i와 보다 낮은 우선순위를 가진 태스크 T_j가 상호배제적으로 같은 자원을 접근한다고 하자. 만약 T_j가 이미 자원을 사용하고 있을 때 T_i가 이 자원을 사용하려고 하면 T_i는 T_j가 자원의 사용을 마칠 때까지 기다리게 되는 우선순위 역전 현상이 발생한다. 또한 T_i보다는 낮지만 T_j보다는 높은 우선순위를 가진 다른 태스크들이 T_j를 계속 선점(Preemption)함으로써 T_i의 우선순위 역전 기간은 더욱 길어질 수 있다. 일반적으로 태스크가 자원의 접근을 요청하여 접근이 허용될 때까지 기다리는 시간(이후 블록킹 시간)은 제한(unbounded)되어져 있지 않다. 실시간 시스템의 보다 높은 예측성을 위해서 블록킹 시간은 제한(bounded)되어져야 한다. 공유자원의 접근을 조정하는 적절한 프로토콜을 사용함으로써 이 무제한적인(unbounded) 우선순위 역전 문제를 해결할 수 있다.

자원을 접근하는 태스크들의 스케줄링 문제를 다루기 위해 많은 자원 접근 프로토콜들이 개발되었다. 정적 우선순위 시스템에서 사용되는 우선순위 상속 프로토콜(Priority Inheritance Protocol 또는 PIP)과 우선순위 상한 프로토콜(Priority Ceiling Protocol 또는 PCP)이 개발되었다[5]. 이를 EDF 스케줄링 기법에서 사용하기 위해 확장된 프로토콜이 개발되었다[6, 7]. 정적, 동적 우선순위 시스템 모두에서 사용할 수 있는 스택 자원 기법(Stack Resource Policy 또는 SRP)도 제안되었다. 이 연구들에서는 자원 접근 프로토콜의 운영 방식뿐만 아니라 이를 사용하는 주기적 태스크 집합의 스케줄링 가능성(Schedulability)을 분석하기 위한 조건도 제시하고 있다. 이 중 Chen이 제안한 동적 우선순위 상한 프로토콜(Dynamic Priority Ceiling Protocol 또는 DPCP)은 우선순위 역전 기간을 제한(bound)시키고 PIP에서 발생하는 교착상태(Deadlock)와 연속된 블록킹(Chained Blocking) 현상을 방지하는 자원 접근 프로토콜이다[6].

본 논문에서는 DPCP의 스케줄링 가능성 조건을 개선한다. DPCP의 스케줄링 가능성 조건은 태스크가 블록킹되는 시간을 태스크의 수행시간으로 간주하여 구해진 충분조건이다. 본 논문에서는 EDF와 LLF와 같은 동적 우선순위 스케줄링 기법에서 우선순위 역전이 발생한다고 해서 높은 우선순위를 가진 태스크가 반드시 마감시간을 놓치는 것은 아님을 보인다. 높은 우선순위를 가진 태스크는 낮은 우선순위를 가진 태스크에게 먼저 수행될 수 있도록 어느 정도 수행시간을 양보해 줄 수 있다. 즉, 낮은 우선순위를 가진 태스크가 이미 자원을 사용하여 높은 우선순위를 가진 태스크가 블록킹되었을 때 높은 우선순위를 갖는 태스크는 낮은 우선순위를 가진 태스크가 수행을 더 진행할 수 있도록 허용해 줄 수 있다. 이렇게 허용된 수행시간은 DPCP의 스케줄링 가능성 조건에서 고려한 태스크의 최악 블록킹 시간에서 제외시킬 수 있으므로 최악 블록킹 시간을 줄일 수 있다. 태스크가 블록킹되는 시간을 보다 정확히 구하여 감소시킴으로써 DPCP의 스케줄링 가능성 조건을 보다 개선할 수 있다.

논문의 구성은 다음과 같다. 2장에서는 본 논문에서 사용하는 정의와 표기법을 설명한다. 3장에서는 관련연구로써 DPCP에 대해 간략히 소개한다. 4장에서는 우선순위 역전 현상 특성을 살펴보고 개선한 DPCP의 스케줄링 가능성 조건을 제시한다. 5장에서 결론으로 논문을 맺는다.

2. 정의 및 표기

태스크들끼리 공유하는 자원은 일반적으로 자료구조, 변수들의 집합, 메모리의 공간, 파일 또는 하드웨어 디바이스의 레지스터 집합 등이 될 수 있다. 상호배제 방식으로 접근을 허용하는 자원은 자원의 일치성(Consistency)을 항상 유지해야 한다. 상호배제 방식으로 수행되는 코드의 일부분을 임계구역(Critical Section)이라고 한다. 대부분의 운영체제 커널은 임계구역의 접근을 조정하기 위해 세마포어(Semaphore)라는 동기화 기법

을 제공한다. 각 임계구역은 해당하는 세마포어를 가지고 있다. 태스크는 임계구역 안으로 들어가기 전에 임계구역을 보호하는 세마포어를 잠근다. 태스크는 임계구역에서 나올 때 세마포어의 잠금을 푼다. 일단 태스크에 의해서 세마포어가 잠기면 세마포어의 잠금이 풀릴 때까지 다른 태스크는 임계구역 내로 들어올 수 없다. 임계구역에 들어가고자 하는 태스크는 이 임계구역 내에 들어가 있는 다른 태스크가 잠금을 풀 때까지 기다려야 한다. 태스크가 공유자원 접근을 위해 기다리는 것을 블록킹(Blocking) 되었다고 한다. 만약에 임계구역 내에 다른 태스크가 존재하지 않는다면 곧바로 임계구역 내로 들어가 세마포어를 잠그고 자원을 사용한다. 태스크가 임계구역 내에서 나올 때 이 임계구역에 대해서 블록킹된 다른 태스크가 자원에 접근할 수 있도록 허용한다.

본 논문에서의 가정과 태스크와 임계구역, 세마포어에 대한 표기법은 다음과 같다.

- 모든 태스크는 주기적 태스크이고 단일 프로세서 상에서 수행된다.
- 태스크 T_i의 주기와 최악 수행시간, 그리고 상대적 마감시간을 각각 p_i, c_i, d_i로 표기한다.
- 모든 주기적 태스크는 시간 0에 시작하며 $p_i = d_i$이다.
- 태스크는 자체적으로 수행을 중지하지 않는다.
- 시간 t에서 태스크 T_i의 상태는 태스크의 마감시각까지 남은 마감시간(Deadline) $D_i(t)$와 태스크가 수행 종료하기까지 남은 잔여 수행시간(Remaining Execution Time) $E_i(t)$로 나타낼 수 있다. 이 태스크의 여유시간(Laxity) $L_i(t)$는 다음과 같다.

$$L_i(t) = D_i(t) - E_i(t)$$

  여유시간이란 태스크가 시간 t부터 선점되지 않는다고 가정하고 수행을 마쳤을 때 마감시간까지의 여분 시간을 말한다. 만일 여유시간이 0이라면 이 태스크는 선점되지 않고 종료할 때까지 수행되어야 마감시간을 놓치지 않는다. 음수의 여유시간을 갖는 태스크는 마감시간보다 잔여 수행시간이 더 크므로 이미 마감시간을 보장할 수 없는 태스크이다. 여유시간은 태스크의 급한 정도를 나타낸다[3].
- 태스크 T_i가 수행 도중 잠글 수 있는 세마포어의 리스트를 SL_i라고 한다. 예를 들어 T_i의 수행 도중 세마포어 S_x와 S_y를 잠그려고 한다면 $SL_i = \{S_x, S_y\}$이다.
- 임계구역 Z_x은 해당하는 세마포어 S_x를 갖는다. 태스크가 임계구역 내에 들어간 후 선점되지 않고 수행하여 임계구역에서 나올 때까지 걸리는 시간을 $E(Z_x)$라고 표기한다.
- 교착상태를 피하기 위해 태스크에 의해 사용되는 자원은 적절히 내포(properly nested)되어 있다고 가정한다 [5, 6]. 즉, 한 임계구역은 다른 임계구역 내에 완전히 포함되거나 임계구역은 서로 겹치지 않아야 한다.
- 시간 t에서 세마포어 S를 잠글 수 있는 가장 높은 우선순위를 가진 태스크의 우선순위를 세마포어 S의 동적 우선순위 상한(Dynamic Priority Ceiling) 또는 우선순위 상한(Priority Ceiling)이라고 하고 $c_t(S)$로 표기한다 [6]. EDF 스케줄링 기법에서는 가장 빠른 마감시간을 가진 태스크의 우선순위가 가장 높다. 현재 시간에서 세마포어 S의 우선순위 상한은 $c(S)$로 표기한다. EDF스케줄링 기법을 사용하는 경우 현재 주기의 태스크가 종료되면 태스크의 우선순위는 다음 주기의 마감시간으로 변하기 때문에 세마포어의 우선순위 상한은 동적으로 변하게 된다.

3. 동적 우선순위 상한 프로토콜 (DPCP)

본 장에서는 본 논문에서 고려하는 DPCP의 운영 방식과 특징을 기술한다. DPCP는 상한 프로토콜(Ceiling Protocol)과 잠금 프로토콜(Locking Protocol)로 구성된다[6]. 상한 프로토콜은 세마포어의 우선순위 상한 값을 정의한다. 잠금 프로토콜은 언제 잠금 요청을 허락할 것인지를 결정한다. 상한 프로토콜과 잠금 프로토콜은 다음과 같다.

- 상한 프로토콜 : 세마포어 S의 우선순위 상한 값은 이 세마포어를 잠갔거나 아니면 나중에 잠글 수 있는 가장 높은 우선순위를 갖는 태스크의 우선순위와 항상 같다.
- 잠금 프로토콜 : 세마포어 S에 대해 잠금 요청을 한 태스크 T_i는 $D_i(t) < c(S_H)$를 만족해야만 세마포어 S를 잠글 수 있다. 여기서 S_H는 T_i가 아닌 다른 태스크에 의해 잠긴 세마포어들 중 가장 높은 우선순위 상한을 가진 세마포어이다. 만약 $D_i(t) \geq c(S_H)$를 만족하여 세마포어 S를 잠글 수 없다면 태스크 T_i는 블록킹된다. 그리고 S_H를 잠근 태스크 T_j는 T_j가 S_H의 잠금을 풀 때까지 T_i의 우선순위를 상

속 받는다.

예제를 통해 DPCP의 동작을 설명한다. 태스크 T_1, T_2, T_3의 세마포어 리스트는 각각 $\{S_x\}$, $\{S_y, S_x\}$, $\{S_y\}$ 고 하자. 각 태스크의 발생시간과 마감시간은 그림 1과 같다. 세마포어 S_x와 S_y의 우선순위 상한은 가장 빠른 마감시간을 가진 T_2의 마감시간과 같다. 시간 t_0에 T_3가 발생하여 수행되다가 시간 t_1에 세마포어 S_y를 잠근다. 세마포어 S_x와 S_y가 사용 중에 있지 않기 때문에 T_3는 세마포어 S_y를 잠글 수 있다. 이 때 $c(S_H)$는 T_2의 마감시간이 된다. 시간 t_2에 T_3보다 빠른 마감시간을 갖는 T_1가 발생하여 T_1를 선점하고 수행한다. 시간 t_3에 T_1가 세마포어 S_x를 잠그려고 시도한다. 그러나 $D_1(t_3) > c(S_H) = D_2(t_3)$이므로 T_1은 T_3가 세마포어 S_y의 잠금을 풀 때까지 기다리게 된다. 이후 시간 t_4에 T_2가 발생하여 T_2를 선점하고 수행한다. 시간 t_5에 세마포어 S_y를 잠그려고 시도하지만 $D_3(t_5) = c(S_H)$이므로 T_2는 T_3가 세마포어 S_y의 잠금을 풀 때까지 기다리게 된다. 시간 t_6에 T_3가 세마포어 S_y의 잠금을 푼다. T_2가 세마포어 S_y를 잠그고 수행한다. 시간 t_7에 T_2는 세마포어 S_y의 잠금을 풀고 세마포어 S_x를 잠그려고 시도한다. 이 때 어떠한 세마포어도 잠겨 있지 않기 때문에 T_2는 세마포어 S_x를 잠글 수 있다. 시간 t_8에 T_2의 수행이 종료되고 T_1이 수행된다. 이 때 비로서 T_1은 세마포어 S_x를 잠그고 임계구역 내에 들어갈 수 있게 된다. T_1이 종료된 후 T_3가 수행된다.

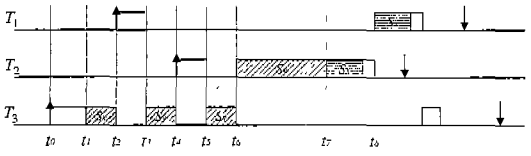


그림 1 DPCP의 예제

만약 동일한 태스크에 대해서 PIP를 사용했다면 T_1은 시간 t_3에 세마포어 S_x를 잠글 수 있다. 그러나 이렇게 하면 T_2는 수행 도중 두 번 블록킹된다. 한번은 T_3에 의해서 그리고 한번은 T_1에 의해서 블록킹 된다. 높은 우선순위를 가진 태스크는 수행 도중 한 개 이상의 임계구역 내에 들어갈 수 있다. 최악의 경우 각 임계구역 내에 들어가려고 할 때마다 낮은 우선순위를 갖는 태스크가 이미 임계구역 내에 들어가 있는 경우가 발생할 수 있다. 이렇게 되면 태스크가 수행되는 도중에 블록킹되는 수가 많아지게 된다. 이러한 현상을 연속된 블록킹이라고 한다. 태스크가 수행되는 도중 블록킹되는 수가 많아지면 태스크의 종료시간이 지연되어 마감시간을 놓치는 경우가 발생할 수 있다.

DPCP에서 태스크는 처음 임계구역에 들어가기 전에만 블록킹되는 특성을 가지고 있다. 일단 태스크가 처음 임계구역 내에 들어가게 되면 이 태스크보다 낮은 우선순위를 갖는 태스크에 의해 다시는 블록킹 되지 않는다. 이는 교착상태가 불가능하고 연속된 블록킹이 발생하지 않음을 나타낸다.

태스크 T_i를 블록킹시킬 수 있는 임계구역의 집합을 태스크의 블록킹 집합 β_i이라고 정의한다. DPCP에서 태스크 T_i는 T_i 또는 T_i보다 높은 우선순위를 갖는 태스크가 접근할 수 있는 임계구역 내에 낮은 우선순위를 갖는 태스크 T_j가 이미 들어가 있고 T_i가 이 T_j를 선점하려고 할 때에만 블록킹된다.

$p_i < p_j$를 만족하는 태스크 T_i와 T_j가 있을 때 두 태스크의 마감시간에 따라 다음과 같은 경우들이 존재한다(그림 2 참조).

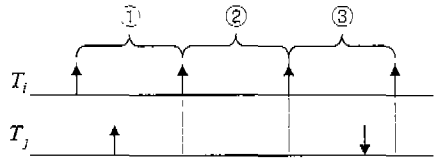


그림 2 EDF에서 동적으로 변하는 태스크의 우선순위

① T_i가 T_j보다 먼저 또는 같이 요청되고 T_i의 마감시간이 T_j보다 빠르다.

② T_j가 T_i보다 먼저 요청되고 T_i의 마감시간이 T_j보다 빠르다.

③ T_j가 T_i보다 먼저 요청되고 T_j의 마감시간이 T_i보다 빠르거나 같다.

①의 경우 T_i가 T_j보다 빠른 마감시간을 가지기 때문에 T_i는 T_j에 의해 선점되지 않는다. ③의 경우 T_j의 마감시간이 T_i보다 빠르므로 T_i는 T_j에 의해 선점되지 않는다. T_i가 T_j에 의해 선점되는 경우는 ②의 경우 만이다[6, 8]. 반대로 주기가 긴 T_i는 어떠한 경우라도 T_i를 선점할 수 없다. 이는 T_i가 먼저 요청되었을 때 T_j가 T_i보다 빠른 마감시간을 가질 수 없기 때문이다. 그러므로 EDF에서 T_i가 선점할 수 있는 태

스크는 T_i보다 긴 주기를 갖는 태스크이다.

T_i의 블록킹 집합은 T_i가 선점할 수 있는 태스크 즉, T_i보다 긴 주기를 갖는 태스크가 접근할 수 있는 임계구역만을 포함한다[6]. T_i의 블록킹 집합 β_i은 다음과 같이 정의된다.

$$\beta_i = \{Z_x \mid Z_x \in SL_i \cap SL_j \text{ and } p_i < p_j\} \cup \{Z_x \mid Z_x \in SL_j \cap SL_k \text{ and } p_k < p_i < p_j\}$$

예를 들어 주기가 각각 16, 18, 20이고 $SL_1=\{S_1, S_2\}$, $SL_2=\{S_1, S_3\}$, $SL_3=\{S_2, S_3\}$인 태스크 T_1, T_2, T_3가 있을 때 각 태스크의 블록킹 집합은 $\beta_1=\{S_1, S_2\}$, $\beta_2=\{S_2, S_3\}$, $\beta_3=\{\ \}$이다.

DPCP에서 태스크는 자신의 처음 임계구역 내에 들어가기 전에 블록킹되므로 최악 블록킹 시간(Worst-case Blocking Time) B_i는 태스크 T_i의 블록킹 집합 β_i에서 가장 긴 임계구역의 크기가 된다. 는 다음과 같이 구해진다.

$$B_i = \max\{E(Z_x) \mid Z_x \in \beta_i\}$$

DPCP를 사용하는 n개의 주기적 태스크 집합이 스케쥴링 가능하기 위한 충분조건은 다음과 같다[6].

$$\sum_{i=1}^{n} \frac{c_i + B_i}{p_i} \le 1$$

이 조건은 최악 블록킹 시간을 태스크 수행의 추가적인 수행으로 간주함으로써 구해진다.

4. DPCP의 개선된 스케줄링 가능성 조건

본 장에서는 3장에서 언급된 DPCP의 스케쥴링 가능성 조건을 보다 개선한다. 기본적인 접근 방식은 태스크 T_i가 블록킹되는 최악 블록킹 시간 B_i을 보다 정확히 계산하여 감소시킴으로써 DPCP의 스케쥴링 가능성 조건을 개선하는 것이다. 먼저 높은 우선순위를 가진 태스크는 낮은 우선순위를 가진 태스크가 어느 정도 먼저 수행될 수 있도록 허용하여도 즉, 어느 정도의 우선순위 역전 현상이 발생하더라도 태스크가 반드시 마감시간을 놓치지는 것이 아님을 보인다. 이는 높은 우선순위를 가진 태스크가 접근하는 임계구역 내에 낮은 우선순위를 가진 태스크가 이미 들어가 있을 때 낮은 우선순위를 가진 태스크가 어느 정도 계속 수행할 수 있도록 허용해 줄 수 있음을 의미한다. 이 허용해 준 시간은 태스크가 블록킹된 시간이 아니라 우선순위 역전이 발생했음에도 불구하고 수행을 허용해 준 시간이다. 그러므로 이 허용된 시간은 태스크의 최악 블록킹 시간에서 제외할 수 있다. 이러한 방식으로 태스크의 최악 블록킹 시간을 감소시킬 수 있다. 마지막 절에서는 DPCP의 기존 스케쥴링 가능성 조건을 만족하지 않지만 개선된 스케쥴링 가능성 조건을 만족하는 주기적 태스크 집합의 예를 보인다. 그리고 이 주기적 태스크 집합은 DPCP 운영 방식에 의해서 실행 가능한 스케줄을 가짐을 보인다.

4.1 우선순위 역전 허용

다음 정리 1은 EDF와 LLF 스케쥴링 기법에서 마감시간, 여유시간 우선순위 역전 현상이 발생해도 태스크가 반드시 마감시간을 놓치는 것은 아님을 보인다.

정리 1. 시간 t에 $D_i(t) \le D_j(t)$를 만족하는 스케쥴링 가능한 태스크 T_i와 T_j가 있을 때 T_i의 마감시간을 어기지 않는 최대의 여유시간을 δ라 할 때 δ는 $D_i(t) - L_j(t)$이다. 즉, δ동안 T_j를 수행시킨 후 T_i를 수행시켜도 마감시간을 어기지 않는다.

증명:

시간 t에 T_i와 T_j는 스케쥴링 가능하므로 다음의 식 (1)를 만족한다.

$$L_i(t) \ge 0 \text{ and } L_j(t) \ge E_i(t) \quad (1)$$

T_i를 수행하기 전에 T_j를 δ만큼 수행시킨 후의 시간 $t+\delta$에 태스크 T_i와 T_j가 여전히 스케줄 가능하기 위해 다음의 식 (2)와 (3)을 만족해야 한다.

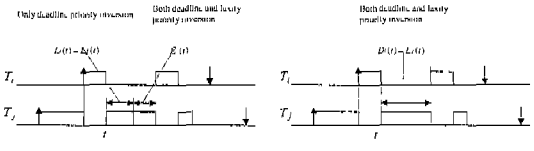
$$L_i(t+\delta) \ge 0 \quad (2)$$

$$L_j(t+\delta) \ge E_i(t+\delta) \quad (3)$$

시간 $t+\delta$까지 T_j는 수행되지 않았으므로 $L_i(t+\delta) = L_i(t) - \delta$이다. 식 (2)는 다음 식 (4)에 의해 만족된다.

$$\begin{aligned} L_i(t+\delta) &= L_i(t) - \delta \ge L_i(t) - D_i(t) + L_j(t) \\ &= L_j(t) - E_i(t) \ge 0 \end{aligned} \quad (4)$$

시간 $t+\delta$에서 $L_j(t+\delta)$와 $E_i(t+\delta)$는 각각 $L_j(t)$와 $E_i(t)$이므로 식 (3)은 식 (1)에 의해 만족된다.■



(a) $L_j(t) \le L_i(t)$ (b) $L_j(t) > L_i(t)$

그림 3 마감시간과 여유시간 우선순위 역전

마감시간과 여유시간 우선순위 역전 현상을 살펴보기 위하여 T_i와 T_j의 여유시간에 따라 두 가지 경우로 나눌 수 있다. 먼저 $L_j(t) \le L_i(t)$인 경우를 살펴보자(그림 3(a) 참조). 시간 t이후에 T_j가 $L_i(t) - L_j(t)$만큼 수행될 때에는 EDF 스케쥴링 측면에서 마감시간 우선순위 역전이 발생한다. 왜냐하면 T_i의 마감시간이 T_j보다

빠르기 때문이다. 그러나 T_j의 여유시간은 T_i보다 작거나 같으므로 LLF 스케줄링 측면에서 여유시간 우선순위 역전은 발생하지 않는다. 그러므로 T_j가 $L_i(t)-L_j(t)$만큼 수행될 때까지는 LLF 스케줄링 기법으로 스케줄링 하는 것과 같다. 이후에 T_j가 $E_i(t)$만큼 더 수행될 때에는 마감시간과 여유시간 우선순위 역전이 모두 발생한다.

$L_j(t)>L_i(t)$인 경우를 살펴보자(그림 3(b) 참조). T_j가 $D_i(t)-L_i(t)$만큼 수행되는 동안 EDF, LLF 스케줄링 측면 모두에서 마감시간, 여유시간 우선순위 역전이 발생한다. 그러나 정리 1에 의해 알 수 있듯이 마감시간과 여유시간 우선순위 역전이 발생한다고 해서 태스크가 반드시 마감시간을 놓치게 되는 것은 아니다.

다음 정리 2는 $p_i<p_j$를 만족하는 T_i가 T_j에 의해 블록킹되는 경우 T_j가 T_i보다 먼저 수행될 수 있도록 T_i가 허용해 줄 수 있는 시간을 구한다.

정리 2. $p_i<p_j$인 태스크 T_i와 T_j가 있다. 그림 4와 같이 시간 t에 T_j, 그리고 시간 t'에 T_i가 요청되고 $t'+p_i<t+p_j$이다. (즉, T_i의 마감시간이 T_j보다 빠르다.) 이 때 $t'\le t''<t+p_i$를 만족하는 임의의 시간 t''에 T_i가 T_j에 대해서 허용해 줄 수 있는 최소 우선순위 역전 기간 λ_{ij}은 $p_i-p_j+c_j$이다.

증명:

정리 1에서와 같이 시간 t''에서 $D_i(t'')<D_j(t'')$이면 T_i보다 T_j를 $D_i(t'')-L_j(t'')$만큼 먼저 수행시켜도 T_i는 마감시간을 어기지 않는다.

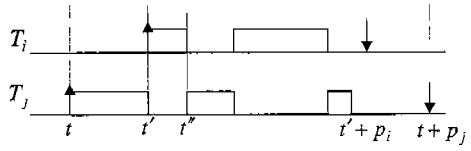


그림 4 우선순위 역전 허용

구간 (t,t')에서 T_j가 ε만큼 수행되었다고 하자. $0\le\varepsilon\le t'-t$이다. 시간 t'에서 태스크 T_j의 여유시간은 다음과 같다.

$$L_j(t')=p_i-c_j-(t'-t)+\varepsilon \quad (5)$$

구간 (t',t'')에서 T_j가 ε'만큼 수행되었다고 하자. $0\le\varepsilon'<c_i$이다. 구간 (t',t'')에서 T_j가 수행될 수 있다면 T_i가 수행될 것이기 때문에 T_j는 절대로 수행되지 않는다. 시간 t''에서 태스크 T_j의 여유시간은 다음과 같다.

$$L_j(t'')=p_i-c_j-(t'-t)+\varepsilon-(t''-t') \quad (6)$$

시간 t''에서 $D_i(t'')-L_j(t'')=p_i-p_j+c_j-\varepsilon+(t'-t)$만큼 T_j를 먼저 수행시킬 수 있다.

$D_i(t'')-L_j(t'')=p_i-p_j+c_j-\varepsilon+(t'-t)\ge p_i-p_j+c_j$ 이므로 T_i가 T_j에게 먼저 수행할 수 있도록 허용해 줄 수 있는 최소 우선순위 역전 기간 λ_{ij}는 $p_i-p_j+c_j$가 된다. ■

정리 2에서 고려한 태스크 T_i와 T_j에 대해서 λ_{ij}는 T_i의 수행 가능한 구간 $(t',t'+p_i)$의 임의의 시점에서 T_j에게 허용해 줄 수 있는 수행시간의 하한값(lower bound)이다. 실제의 경우 구간 (t,t')에서 T_j가 수행된 정도에 따라 T_i가 T_j의 수행을 허용해 줄 수 있는 수행시간은 λ_{ij} 이상이 될 수 있다.

4.2 DPCP의 개선된 스케줄링 가능성 조건

임계구역 $Z_x\in\beta_i$에 대해서 $Z_x\in SL_j$를 만족하고 $p_i<p_j$를 만족하는 태스크 T_i와 T_j가 존재한다고 하자. 앞에서 언급하였듯이 DPCP에서는 임계구역 Z_x안에 T_j가 이미 들어가 있고 T_i가 T_j를 선점하려고 할 때에만 T_i가 블록킹된다. 임계구역 Z_x는 T_i가 들어 갈 수 있는 임계구역이거나 T_i가 들어 가지는 않지만 T_i보다 짧은 주기를 가진 태스크가 들어갈 수 있는 임계구역이다. 어쨌든 간에 T_i는 T_j에 의해서 최대 $E(Z_x)$만큼 블록킹된다.

정리 2에서 살펴 보았듯이 시간 t에서 T_i가 T_j를 선점하려고 할 때 T_i는 T_j가 최소 λ_{ij}만큼 계속 수행할 수 있도록 허용해 줄 수 있다. 이것은 T_i가 T_j에 의해서 블록킹되는 것이 아니라 마감시간, 여유시간 우선순위 역전이 일어났음에도 불구하고 T_j가 더 수행할 수 있도록 T_i가 허용해 주는 것이다. 만약 $E(Z_x)\le\lambda_{ij}$라면 T_i는 T_j가 임계구역 Z_x에서 나올 때까지 먼저 수행될 수 있도록 허용해 줄 수 있으므로 이 경우 T_i는 T_j에 의해서 블록킹되지 않는다. 그러나 $E(Z_x)>\lambda_{ij}$라면 T_i는 T_j에 의해서 최대 $E(Z_x)-\lambda_{ij}$만큼 블록킹될 수 있다.

T_i보다 긴 주기를 가지면서 임계구역 $Z_x\in\beta_i$에 들어갈 수 있는 태스크 T_j는 한 개 이상 존재한다. T_i는 이런 태스크들에 의해서 블록킹될 수 있는데 이 태스크들의 λ_{ij}중 가장 작은 값을 Δ_i^x라고 한다. $E(Z_x)-\Delta_i^x$는 T_i가 임계구역 Z_x에 대해서 블록킹될 수 있는 최대 길이가 된다. $E(Z_x)-\Delta_i^x$를 B_i^x라고 표기한다. 만약 $E(Z_x)-\Delta_i^x$이 음수라면 B_i^x는 0이다. 한편 T_i의 블록킹 집합은 한 개 이상의 임계구역으로 구성된다. T_i는 블록킹 집합의 어떠한 임계구역으로 인해서도 블록킹될

수 있다. T_i의 블록킹 집합의 임의의 임계구역 중 가장 큰 B_i^x를 B_i^*라고 표기한다. DPCP의 B_i와 구별하기 위해 B_i^*로 표기한다. 정리하면 다음과 같다.

$$\Delta_i^x = \min\{\lambda_{ij} \mid Z_x \in SL_j \cap \beta_i \text{ and } p_i < p_j\}$$

$$B_i^x = \max\{E(Z_x) - \Delta_i^x, 0\}$$

$$B_i^* = \max\{B_i^x \mid Z_x \in \beta_i\}$$

DPCP의 스케줄링 가능성 조건은 다음과 같이 개선된다. 이 조건 또한 B_i^*를 테스크의 추가적인 수행시간으로 간주하여 구해진다.

$$\sum_{i=1}^{m} \frac{c_i + B_i^*}{p_i} \le 1$$

B_i^*는 최소한 B_i보다 크지 않기 때문에 개선된 스케줄링 가능성 조건은 DPCP의 기존 스케줄링 가능성 조건과 최소한 같거나 보다 좋다.

4.3 개선된 스케줄링 가능성 조건을 이용한 DPCP의 예제

다음 표 1의 특성을 갖는 테스크 T_1, T_2, T_3와 표 2와 같이 임계구역의 수행시간을 갖는 임계구역 Z_1, Z_2, Z_3가 존재한다.

표 1 태스크 정보

	p_i	c_i	$p_i - c_i$	SL_i	β_i	B_i
T_1	16	3	13	S_1, S_2	S_1, S_2	2
T_2	18	5	13	S_1, S_3	S_2, S_3	4
T_3	20	10	10	S_2, S_3	$\{\}$	0

표 2 임계구역 정보

$E(Z_1)$	$E(Z_2)$	$E(Z_3)$
1	2	4

각 태스크의 B_i^*를 다음과 같이 구한다.

- $\lambda_{12}, \lambda_{13}, \lambda_{23}$은 각각 3, 6, 8이다.
- B_1^*의 계산
  - $Z_1 \in \beta_1$이고 $Z_1 \in SL_j$, $p_1 < p_j$를 만족하는 태스크 T_j는 T_2뿐이므로 $\Delta_1^1 = \lambda_{12} = 3$이다. $E(Z_1) - \Delta_1^1 = -2$이므로 B_1^1은 0이다.
  - $Z_2 \in \beta_1$이고 $Z_2 \in SL_j$, $p_1 < p_j$를 만족하는 태스크 T_j는 T_3뿐이므로 $\Delta_1^2 = \lambda_{13} = 6$이다. $E(Z_2) - \Delta_1^2 = -4$이므로 B_1^2은 0이다.
  - 그러므로 B_1^*는 0이다.
- B_2^*의 계산
  - $Z_2 \in \beta_2$이고 $Z_2 \in SL_j$, $p_2 < p_j$를 만족하는 태스크 T_j는 T_3뿐이므로 $\Delta_2^2 = \lambda_{23} = 8$이다. $E(Z_2) - \Delta_2^2 = -6$이므로 B_2^2은 0이다.
  - $Z_3 \in \beta_2$이고 $Z_3 \in SL_j$, $p_2 < p_j$를 만족하는 태스크 T_j는 T_3뿐이므로 $\Delta_2^3 = \lambda_{23} = 8$이다. $E(Z_3) - \Delta_2^3 = -4$이므로 B_2^3은 0이다.
  - 그러므로 B_2^*는 0이다.
- T_3는 어떤 태스크로부터도 블록킹되지 않으므로 B_3^*는 0이다.

다음의 식에서 볼 수 있듯이 이 태스크 집합은 DPCP의 기존 스케줄링 가능성 조건으로는 실행 가능하지 않다.

$$\frac{c_1 + B_1}{p_1} + \frac{c_2 + B_2}{p_2} + \frac{c_3 + B_3}{p_3} = 0.3125 + 0.5 + 0.5 = 1.3125 > 1$$

그러나 개선된 DPCP의 스케줄링 가능성 조건에 의하면 이 테스크 집합은 실행 가능하다.

$$\frac{c_1 + B_1^*}{p_1} + \frac{c_2 + B_2^*}{p_2} + \frac{c_3 + B_3^*}{p_3} = 0.1875 + 0.2778 + 0.5 = 0.9653 < 1$$

그림 5와 같이 이 주기적 태스크 T_1, T_2, T_3는 실행 가능하다.

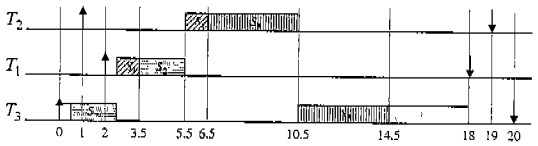


그림 5 표 1의 주기적 태스크에 대한 DPCP의 예제

5. 결 론

실시간 시스템의 존재하는 태스크가 자원을 공유할 때 무제한적인 우선순위 역전 문제를 해결하기 위해 적절한 자원 접근 프로토콜을 이용한다. 자원 접근 프로토콜은 태스크의 우선순위 역전 시간을 제한시킴으로써 실시간 시스템의 예측성을 높인다. 또한 이 자원 접근 프로토콜을 사용하는 주기적 태스크 집합의 스케줄링 가능성을 분석할 수 있다.

본 논문에서는 자원 접근 프로토콜의 하나인 DPCP의 스케줄링 가능성 조건을 보다 개선하였다. 이는 EDF와

LLF 스케줄링 알고리즘에서 마감시간, 여유시간 우선순위 역전 현상을 일시적으로 허용하더라도 태스크의 마감시간을 맞출 수 있다는 사실을 이용하였다. 우선순위 역전이 발생하더라도 높은 우선순위를 가진 태스크는 낮은 우선순위를 가진 태스크에게 수행시간을 양보할 수 있으므로 태스크의 최악 블록킹 시간을 보다 정확히 구하였다. 이를 통해 충분조건인 DPCP의 스케줄링 가능성 조건을 보다 개선하였다.

참 고 문 헌

[1] C.L. Liu and J.W. Layland, Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment, *Journal of the Association for Computing Machinery*, 20(1), 1973.

[2] Cheng S.-C, J.A. Stankovic, and K. Ramamritham, Scheduling Algorithms for Hard Real-Time Systems A Brief Survey, Tutorial on Hard Real-Time Systems, J.A. Stankovic and K. Ramamritham (Eds.), *IEEE Computer Society*, 1988.

[3] M.L. Dertouzos, Control Robotics: the Procedural Control of Physical Processes, *Information Processing 74*, North-Holland Publishing Company, 1974.

[4] A.K. Mok, Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment, *Ph.D. Thesis*, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts, May 1983.

[5] L. Sha, R. Jajkumar, and J. P. Lehoczky, Priority inheritance protocols: An Approach to real-time synchronization, *IEEE Transactions on Computer*, Vol. 39, No. 9, September 1990.

[6] Min-Ih Chen and Kwei-Jay Lin, Dynamic Priority Ceilings: A Concurrency Control Protocol for Real-Time Systems, *The Journal of Real-Time Systems*, 2, 1990, pp. 325-346.

[7] M. Spuri, Efficient Deadline Scheduling in Real-Time Systems, *Ph.D. Thesis*, Scuola Superiore S. Anna, July 1995.

[8] T.P. Baker, Stack-Based Scheduling of Real-Time Processes, *The Journal of Real-Time Systems* 3, 1991.

오 성 흔

1996년 2월 인천대학교 정보통신공학(학사). 1998년 8월 숭실대학교 전자계산학(석사). 1998년 9월 ~ 현재 숭실대학교 컴퓨터학과 박사과정. 관심분야는 실시간 스케줄링, 운영체제, 실시간 통신 프로토콜 등임.

양 승 민

1978년 2월 서울대학교 전자공학(학사). 1983년 4월 (미) Univ. of South Florida 전산학(석사). 1986년 12월 (미) Univ. of South Florida 전산학(박사). 1988년 ~ 1993년 (미) Univ. of Texas at Arlington 조교수. 1993년 ~ 현재 숭실대학교 컴퓨터학과 부교수. 1996년 ~ 1998년 국회도서관 정보처리국장. 관심분야는 실시간 시스템, 운영체제, 결함허용 시스템 등임.