다중 프로세서 시스템에서 문맥교환을 줄이기 위한 변형된 LLF 스케줄링 알고리즘

(A Modified Least-Laxity First Scheduling Algorithm for Reducing Context Switches on Multiprocessor Systems)

오 성 흔 † 길 아 라 †† 양 승 민 †††

(Sung-Heun Oh) (Ara Khil) (Seung-Min Yang)

요 약 최소 여유시간 우선(Least-Laxity First, LLF) 스케줄링 알고리즘은 작은 여유시간을 가진 태스크가 높은 우선순위를 갖는 스케줄링 알고리즘으로써 단일 프로세서 시스템에서 최적이고 다중 프로세서 시스템에서 준최적으로 증명되었다. 그러나 이 스케줄링 알고리즘은 여유시간 충돌이 발생하였을 때 태스크 간에 빈번한 문맥교환이 발생하게 되는 문제점이 있어 실용적이지 못하다.

본 논문에서는 과도한 문맥교환을 일으키는 LLF의 문제점을 해결하기 위해 다중 프로세서 시스템을 위한 MLLF/MP(Modified Least-Laxity First on Multiprocessor) 스케줄링 알고리즘을 제안한다. MLLF/MP는 태스크의 여유시간 역전이 발생하더라도 마감시간을 놓치지 않는 범위에서 태스크를 연속적으로 수행시킴으로써 빈번한 문맥교환이 발생하는 것을 방지한다. MLLF/MP 또한 다중 프로세서 시스템에서 준최적임을 증명한다. 모의 실험 결과를 통하여 MLLF/MP는 LLF보다 적은 스케줄링 오버헤드를 가짐을 보인다.

키워드 : 실시간 시스템, 실시간 스케줄링 알고리즘, 실시간 태스크, 스케줄링 최적성, 문맥교환 오버헤드, 여유시간

Abstract The Least-Laxity First(or LLF) scheduling algorithm assigns the highest priority to a task with the least laxity, and has been proved to be optimal for a uni-processor and sub-optimal for a multi-processor. However, this algorithm is impractical to implement because laxity tie results in the frequent context switches among tasks.

In this paper, a Modified Least-Laxity First on Multiprocessor(or MLLF/MP) scheduling algorithm is proposed to solve this problem, i.e., laxity tie results in the excessive scheduling overheads. The MLLF/MP is based on the LLF, but allows the laxity inversion. MLLF/MP continues executing the current running task as far as other tasks do not miss their deadlines. Consequently, it avoids the frequent context switches. We prove that the MLLF/MP is also sub-optimal in multiprocessor systems. By simulation results, we show that the MLLF/MP has less scheduling overheads than LLF.

Key word : real-time system, real-time scheduling algorithm, real-time tasks, scheduling optimality, context switching overhead, laxity

1. 서 론

실시간 응용이 보다 높은 성능을 요구하게 되면서 다중 프로세서 시스템을 이용한 실시간 응용들이 증가되어 왔다. 다양한 다중 프로세서 시스템 구조가 제안되었으며 구현되어 왔다. Dertouzous와 Mok은 다중 프로세서 시스템에서 사용되는 마감시간 우선(earliest-dead line first, EDF) 스케줄링 알고리즘과 최소 여유시간 우선(least-laxity first, LLF) 스케줄링 알고리즘에 대한 연구를 수행하였다[1, 2]. EDF는 적은 문맥교환을 갖지만 다중 프로세서 시스템에서 LLF에 비해 낮은 스케줄링 성공률을 갖는다. LLF는 단일 프로세서 시스템

† 비 회 원 : (주)디지캡 책임 연구원
honestly@digicaps.com
†† 종신회원 : 숭실대학교 컴퓨터학부 교수
ara@computing.soongsil.ac.kr
yang@comp.soonsil.ac.kr
논문접수 : 2001년 12월 17일
심사완료 : 2002년 11월 29일

에서 최적이고 다중 프로세서 시스템에서 준최적(sub-optimal)이라는 높은 스케줄링 능력을 갖지만 여유시간 충돌(laxity-tie)이 발생하였을 때 과도한 문맥교환을 일으키는 단점을 갖는다. 이러한 LLF의 과도한 문맥교환을 줄이기 위한 연구로써 단일 프로세서 시스템에서 사용되는 LLF/MP(Least-Laxity First with Minimum Preemption)와 다중 프로세서 시스템에서 사용되는 EDZL(Earliest Deadline Zero Laxity) 스케줄링 알고리즘이 제안되었다[5, 6].

본 논문에서는 다중 프로세서 시스템에서 문맥교환 횟수의 발생을 감소시킴으로써 LLF보다 낮은 스케줄링 오버헤드를 갖는 MLLF/MP(Modified Least-Laxity First on Multiprocessor) 스케줄링 알고리즘을 제안한다. 이 스케줄링 알고리즘에서는 태스크의 여유시간 우선순위 역전(laxity priority inversion)이 발생하더라도 마감시간을 놓치지 않는 범위에서 현재 수행되는 태스크를 계속 수행시킴으로써 빈번한 문맥교환이 발생하는 것을 방지한다. MLLF/MP는 LLF와 같이 다중 프로세서 시스템에서 준최적의 특성을 가지고 있다. 본 논문에서는 MLLF/MP의 준최적성을 수학적으로 증명하고 모의 실험 결과를 통하여 LLF, EDZL, MLLF/MP의 문맥교환 발생비율을 비교함으로써 MLLF/MP의 향상된 성능을 보인다.

본 논문의 구성은 다음과 같다. 2장에서는 기존의 스케줄링 알고리즘 연구를 살펴 본 후 3장에서 태스크 모델을 제시한다. 4장에서는 다중 프로세서 시스템에서의 LLF와 LLF/MP의 특성을 살펴 본 후 MLLF/MP 스케줄링 알고리즘을 제시하고 준최적성을 수학적으로 증명한다. 5장에서의 모의 실험을 통하여 LLF, EDZL, MLLF/MP의 문맥교환 발생비율을 비교한다. 6장에서 결론으로 논문을 맺는다.

2. 관련 연구

대표적인 정적 우선순위 스케줄링 방법인 비율단조(rate-monotonic, RM) 스케줄링 알고리즘은 주기가 짧은 태스크에 높은 우선순위를 할당하는 방법으로써 동적 우선순위 스케줄링 방식과는 달리 설계 시 태스크의 우선순위가 결정되며 실행 시 우선순위가 바뀌지 않는다[3]. 정적 우선순위 스케줄링 방법은 동적 우선순위 스케줄링 방법에 비해 구현이 간단하지만 프로세서의 이용률은 낮다. 비율단조 스케줄링의 경우 n개의 태스크에 대해서 프로세서 이용률 U가 $U \leq n(2^{1/n}-1)$을만족하면 스케줄 할 수 있고 n이 무한대로 수렴할 때는 U는 0.693이 된다[3]. Dhall과 Liu는 다중 프로세서 시스템에서 비율 단조 스케줄링 알고리즘을 사용하는 연구를 수행하였다[7]. 동적 우선순위 스케줄링 방법의 예로서는 EDF와 LLF 스케줄링 알고리즘이 있다[2, 3, 4]. EDF에서는 현재 시점에서 가장 가까운 마감시간을 갖는 태스크가 가장 높은 우선순위를 갖고 LLF에서는 최소 여유시간을 갖는 태스크가 가장 높은 우선순위를 갖는다. 이때 여유시간이란 태스크가 선점 당하지 않고 수행을 마쳤을 때 마감시간까지의 여분시간이다. 두 스케줄링 알고리즘 모두 선점 가능한 단일 프로세서 상에서 최적인 알고리즘[1]으로 증명되었다[2, 3, 4]. 즉, 프로세서 이용률 U가 1보다 작거나 같은 태스크 집합에 대해서 항상 실행 가능한 스케줄[2]을 만들어 낸다. 그러나 LLF는 그 이론적인 우수성에 비해 여유시간 충돌에 의한 빈번한 문맥교환으로 인하여 실용적이지 못한 단점을 갖는다[5, 6, 8]. 즉, 최소 여유시간을 갖는 태스크가 여러 개 존재하여 여유시간 충돌이 발생하였을 때 이 태스크들은 스케줄링 시점마다 번갈아 수행되면서 빈번한 문맥교환을 일으킨다. 이는 여유시간 충돌이 일어난 태스크 중에서 수행되지 않은 태스크의 여유시간은 수행된 태스크의 여유시간보다 작아지기 때문이다.

LLF의 단점인 빈번한 문맥교환을 방지하면서도 단일 프로세서 시스템에서 최적인 LLF/MP스케줄링 알고리즘이 제안되었다[5]. 이 스케줄링 알고리즘은 최소 여유시간을 가진 태스크에게 수행할 수 있는 우선권을 주되 여유시간 역전(inversion)이 발생하더라도 다른 태스크의 마감시간을 어기지 않는 범위에서 연속적으로 수행시킴으로써 LLF의 빈번한 문맥교환을 방지한다. LLF/MP는 단일 프로세서에서 최적일 뿐만 아니라 최소 선점 특성을 갖는다. 그러나 LLF/MP는 다중 프로세서 시스템에서 준최적이 아니다.

EDF, LLF는 모두 단일 프로세서 시스템에서 최적인 스케줄링 알고리즘으로써 같은 스케줄링 능력을 갖지만 다중 프로세서 시스템에서는 서로 다른 스케줄링 능력을 갖는다[1, 4]. 예를 들어 두 개의 프로세서를 갖는 다중 프로세서 시스템에서 표 1과 같은 태스크 집합을 EDF와 LLF로 스케줄링 하였을 때 EDF는 실행 가능

1) 어떤 알고리즘이 스케줄 가능한 태스크 집합 T에 대해 항상 실행 가능한 스케줄을 찾을 수 있다면 이 알고리즘은 최적 알고리즘(optimal algorithm)이다.

2) 임의의 스케줄이 태스크 집합 T에 대해 모든 태스크들을 마감시간 내에 완료하도록 프로세서 시간을 할당하면 이 스케줄은 실행 가능한 스케줄(feasible schedule)이다. 적어도 한 스케줄링 알고리즘에 의해 실행 가능한 스케줄을 갖는 태스크 집합 T를 스케줄 가능(schedulable)한 태스크 집합 T라고 한다.

한 스케줄을 갖지 못하는 반면 LLF는 실행 가능한 스케줄을 갖는다(그림 1 참조). 즉, EDF는 다중 프로세서 시스템에서 최적이 아니다.

표 1 태스크 집합 예제 1

	잔여 수행시간	마감시간	여유시간
T_1	1	2	1
T_2	1	2	1
T_3	3	3	0

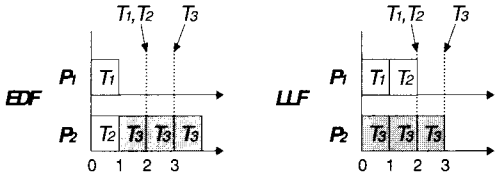

그림 1 다중 프로세서 시스템에서의 EDF와 LLF 스케줄링

Dertouzos와 Mok은 태스크의 마감시간, 최악수행시간, 수행시작시간에 대한 완전한 지식을 미리 갖지 않는 이상 두 개 이상의 프로세서를 갖는 다중 프로세서 시스템에서 마감시간 기반의 최적인 스케줄링 알고리즘은 존재하지 않는다고 증명하였다[1]. LLF는 다중 프로세서 시스템에서 준최적(sub-optimal)임이 증명되었다[1, 6]. 현재 시점에 수행 준비된 태스크 집합이 실행 가능한 스케줄을 가질 때 어떤 스케줄링 알고리즘이 이 태스크 집합에 대해서 항상 실행 가능한 스케줄을 만들어 낼 수 있다면 이 스케줄링 알고리즘을 준최적이라고 말한다. Dertouzos와 Mok에 의해 최적인 스케줄링 알고리즘이 존재하지 않는다고 증명된 이상 준최적성(sub-optimality)이 다중 프로세서 시스템에서 실시간 스케줄링 알고리즘의 성능 척도가 된다.

그러나 다중 프로세서 시스템에서도 LLF는 여유시간 충돌이 발생하는 경우 단일 프로세서 시스템에서와 같이 빈번한 문맥교환을 일으키는 단점을 갖는다. 다중 프로세서 시스템에서 LLF는 작은 여유시간을 갖는 순서대로 최대 m개의 태스크를 항상 수행시킨다. 여기서 m은 프로세서의 개수이다. 만약 태스크의 개수가 m보다 작거나 같다면 모든 태스크는 각각의 프로세서에서 수행될 수 있다. 그러나 LLF에 의해 수행 선택된 태스크들 중에서 선택되지 않은 임의의 태스크와 같은 여유시간을 가진 태스크가 존재하는 경우 여유시간 충돌이 발생한다. 예를 들어 표 2과 같이 동일한 여유시간을 가진 4개의 태스크 T_1, T_2, T_3, T_4와 두 개의 프로세서가 있다고 하자. 그림 2는 표 2의 태스크 집합에 대한 LLF 스케줄을 보여준다. LLF는 여유시간 충돌이 일어나면 충돌된 태스크 중 임의의 태스크를 선택한다. 임의적으로 LLF는 시간 0에서 T_1과 T_3을 수행하기로 선택하였다고 하자. T_1과 T_3을 수행하면 수행되지 않은 T_2와 T_4의 여유시간은 감소하여 수행 선택된 T_1, T_3보다 작은 여유시간을 갖게 된다. LLF는 시간 1에서 T_2와 T_4를 수행시킨다. 시간 2에서 다시 T_1, T_2, T_3, T_4는 여유시간 충돌을 일으킨다. 그림 2에서 볼 수 있듯이 문맥교환은 T_1과 T_3이 종료할 때까지 스케줄링 시점마다 계속 일어난다. 이와 같이 LLF는 단일 프로세서 시스템에서는 최적이고 다중 프로세서 시스템에서는 준최적이라는 매우 좋은 스케줄링 능력을 갖지만 여유시간 충돌 시 빈번한 문맥교환을 일으킴으로써 스케줄링 오버헤드가 크다는 단점을 가지고 있다.

표 2 태스크 집합 예제 2

	잔여 수행시간	마감시간	여유시간
T_1	3	6	3
T_2	4	7	3
T_3	3	6	3
T_4	4	7	3

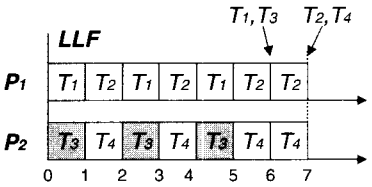

그림 2 LLF의 빈번한 문맥교환

다중 프로세서 시스템에서 LLF의 빈번한 문맥교환의 단점을 해결하기 위해 EDZL 스케줄링 알고리즘이 제안되었다[6]. EDZL은 태스크의 마감시간과 여유시간을 동시에 고려한 LLF와 EDF의 혼합 스케줄링 방식으로써 낮은 문맥교환 오버헤드와 높은 스케줄링 능력을 갖는다. EDZL의 기본 방식은 다음과 같다. EDZL은 여유시간이 0이 되는 태스크가 존재하지 않는 이상 EDF 방식으로 태스크를 스케줄링 한다. 스케줄링 도중 여유시간이 0인 태스크가 발생하면 현재 프로세서에서 수행

중인 태스크들 중에서 가장 큰 마감시간을 가지면서 여유시간이 0이 아닌 태스크와 문맥교환을 하여 여유시간이 0이 된 태스크를 해당 프로세서에서 수행시킨다. EDZL은 두 개의 프로세서를 가진 시스템에서 준최적이지만 3 개 이상의 프로세서를 가진 시스템에서는 더 이상 준최적이 아니다[6].

3. 시스템 및 태스크 모델

본 논문에서 제안하는 스케줄링 알고리즘은 다음과 같은 가정 하에서 동작한다.

가정 1 태스크는 m개의 다중 프로세서 상에서 스케줄링 된다.

가정 2 모든 태스크는 서로 독립적이며 항상 선점 가능하다.

다중 프로세서 시스템에서 태스크 스케줄링은 다음과 같은 제약사항을 갖는다.

- **태스크 제약사항** : 한 태스크는 동시에 두 개 이상의 프로세서에서 수행될 수 없다.
- **프로세서 제약사항** : 프로세서는 임의의 시점에서 한 개의 태스크만 수행할 수 있다.

태스크 T_i의 최악수행시간(worst case execution time), 그리고 상대적 마감시간(relative deadline)을 각각 c_i, d_i로 표기한다. 임의의 시간 t에서 태스크의 상태는 태스크의 절대적 마감시간까지 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.

$$L_i(t) = D_i(t) \quad E_i(t)$$

여유시간이란 태스크가 시간 t부터 선점 당하지 않는다고 가정하고 수행을 마쳤을 때 마감시간까지의 여분 시간을 말한다. 만일 여유시간이 0이라면 이 태스크는 선점 당하지 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[1]. 태스크가 수행되면 여유시간은 감소하지 않는다. 그러나 태스크가 수행되지 않으면 여유시간은 감소한다.

4. MLLF/MP(Modified Least-Laxity First on Multiprocessor) 스케줄링 알고리즘

본 장에서는 MLLF/MP 스케줄링 알고리즘을 제안하기 전에 LLF와 LLF/MP 스케줄링 알고리즘의 작동 방식을 먼저 살펴 본다. 다음 MLLF/MP 스케줄링 알고리즘을 제안하고 그 특성을 살펴 본다.

4.1 LLF와 LLF/MP 스케줄링 알고리즘의 특성

단일 프로세서 시스템에서 LLF의 특징을 먼저 살펴본다. 그림 3과 같이 단일 프로세서 시스템에서 $L_i(t) \leq L_j(t)$를 만족하는 두 태스크 T_i와 T_j가 있을 때 LLF의 스케줄링을 살펴 보자. T_i의 여유시간이 T_j와 같거나 보다 작으므로 T_i가 $L_j(t)-L_i(t)$만큼 수행된 후 시간 t'에 T_i와 T_j의 여유시간이 같아지게 되어 여유시간 충돌이 발생한다. 만약 $L_j(t)-L_i(t) \geq E_i(t)$라면 T_i와 T_j의 여유시간이 같아지기 전에 T_i가 종료하게 되므로 여유시간 충돌은 발생하지 않는다. 그러나 $L_j(t)-L_i(t) < E_i(t)$인 경우 T_i와 T_j의 여유시간 충돌이 발생하면 시간 t'부터 두 태스크 중 한 태스크가 수행을 종료할 때까지 계속 여유시간 충돌을 일으키면서 번갈아 수행을 하게 된다. 만약 $E_i(t)-(L_j(t)-L_i(t)) \leq E_j(t)$라면 즉, $D_i(t) \leq D_j(t)$라면 그림 3(a)에서 볼 수 있듯이 T_j가 T_i보다 먼저 종료된다. 그러나 $E_i(t)-(L_j(t)-L_i(t)) > E_j(t)$라면 즉, $D_i(t) > D_j(t)$라면 그림 3(b)에서 볼 수 있듯이 T_i가 T_j보다 먼저 종료된다. 여유시간 충돌이 발생하였을 때 그림 3(a), (b)와 같이 두 태스크를 번갈아 수행하도록 하지 않고 그림 3(c), (d)와 같이 두 태스크의 수행시간을 서로 교환하여 태스크가 연속적으로 수행될 수 있도록 할 수 있다. 그림 3(c)에서 볼 수 있듯이 $D_i(t) \leq D_j(t)$라면 시간 t부터 T_i를 선점 없이 수행시켜 T_j의 수행을 시작하기 전에 T_i를 수행 종료시킬 수 있다. 그러나 $D_i(t) > D_j(t)$라면 그림 3(d)에서 볼 수 있듯이 시간 t'부터 T_i를 $D_j(t)-L_i(t)$만큼 T_j보다 먼저 수행시킬 수 있다. 이와 같이 했을 때 T_i와 T_j의 종료시점은 LLF 스케줄에 의한 T_i와 T_j의 종료시점보다 늦지 않다. 이것은 LLF/MP 스케줄링 알고리즘의 기본 개념이다[5].

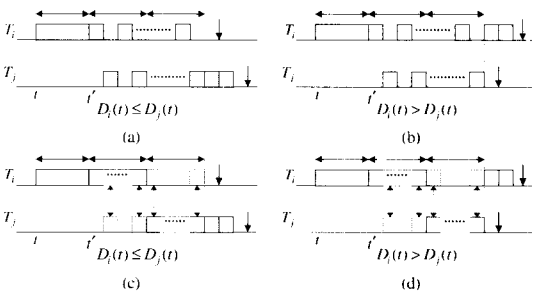

그림 3 LLF 스케줄과 여유시간 우선순위 역전

표 3 태스크 집합 예제 3

	잔여 수행시간	마감시간	여유시간
T_1	5	7	2
T_2	4	7	3
T_3	7	9	2

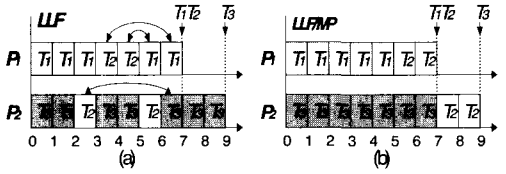

그림 4 LIF와 LLF/MP 스케줄

LLF/MP는 단일 프로세서 시스템에서 낮은 문맥교환을 가지면서도 최적이지만 다중 프로세서 시스템에서는 준최적이 아니다. 이 사실을 확인하기 위해 반례(counter-example)를 보인다. 그림 4에서는 표 3의 태스크 집합에 대한 LLF 스케줄과 LLF/MP 스케줄을 보여 준다. LLF/MP는 시간 0에서 태스크 T_1과 T_3를 수행 선택하며 5단위시간 만큼 T_2보다 먼저 수행한다. LLF/MP는 그림 4(a)에서 표기한 바와 같이 T_1과 T_2의 수행시간을 각각 T_3의 수행시간과 교환한다. 그러나 그림 4(b)를 보면 LLF/MP는 T_2의 마감시간을 놓치게 됨을 볼 수 있다.

LLF/MP가 T_2의 마감시간을 놓치는 이유는 다음과 같다. 그림 4(a)에서 볼 수 있듯이 프로세서 P_1에서 수행되는 시간 4의 T_2의 수행시간과 P_1에서 수행되는 시간 5의 T_1의 수행시간을 교환하였을 때 시간 5에서 T_2는 두 프로세서에서 동시에 수행되도록 수행시간이 교환된다. 그리고 P_1에서 수행되는 시간 3의 T_2의 수행시간과 P_1에서 수행되는 시간 6의 T_1의 수행시간을 교환하고, P_2에서 수행되는 시간 2의 T_2의 수행시간과 P_2에서 수행되는 시간 6의 T_3의 수행시간을 교환하였을 때 시간 6에서 T_2는 두 프로세서에서 동시에 수행되도록 수행시간이 교환된다. 즉, LLF/MP는 T_2가 시간 5와 6에서 두 프로세서에서 동시에 수행되도록 수행시간을 교환한다. 그러나 태스크 제약사항에 의해 T_2는 두 프로세서에서 동시에 수행될 수 없으므로 T_2의 수행은 지연되고 결국 T_2는 마감시간을 놓치게 된다. 이 반례에서 본 것처럼 LLF/MP는 태스크 제약사항을 만족시키지 않고 태스크의 수행시간을 교환할 수 있기 때문에 단일 프로세서 시스템에서 사용되는 LLF/MP를 다중 프로세서 시스템에서 그대로 사용할 수 없다.

4.2 MLLF/MP(Modified Least-Laxity First on Multiprocessor) 스케줄링 알고리즘

본 절에서는 제안하는 MLLF/MP 스케줄링 알고리즘은 수행시간 교환이라는 기본 개념을 사용하되 태스크 제약사항을 만족하는 스케줄링 알고리즘이다. MLLF/MP는 LLF/MP처럼 최소 여유시간을 가진 태스크에게 수행할 수 있는 우선권을 주되 여유시간 역전이 발생하더라도 다른 태스크의 마감시간을 어기지 않는 범위에서 연속적으로 수행시킨다. 그리고 4.1절의 반례에서 본 바와 같이 LLF/MP의 잘못된 수행시간 교환을 방지하기 위해 태스크의 여유시간이 음수가 되지 않는 범위에서 수행시간이 교환될 수 있도록 한다.

MLLF/MP에서 태스크는 예산(budget)시간을 갖는데 이 시간은 태스크가 MLLF/MP에 의해 수행 선택될 때 지정된다. 시간 t에서 태스크 T_i의 예산시간을 $B_i(t)$라고 한다. T_i가 프로세서에서 수행됨에 따라 $B_i(t)$는 감소한다. MLLF/MP 스케줄링 알고리즘은 그림 5와 같다.

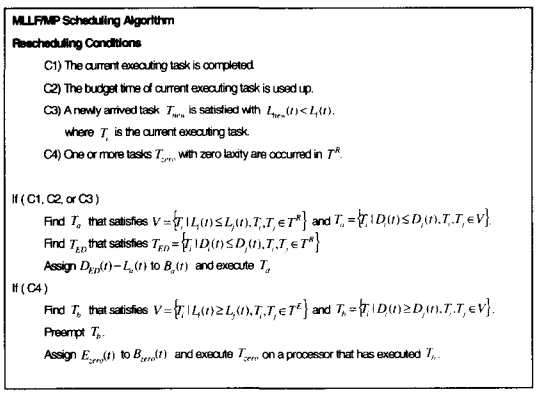

그림 5 MLLF/MP 스케줄링 알고리즘

그림 5에서 T^R은 현재 프로세서에서 수행되지 않는 수행 준비된 태스크들의 집합이다. T^E는 프로세서에서 현재 수행되는 태스크들의 집합이다.

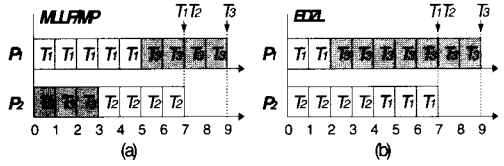

그림 6 MLLF/MP와 EDZL 스케줄

그림 6(a)와 그림 6(b)는 표 3의 태스크 집합에 대한 MLLF/MP 스케줄과 EDZL 스케줄을 보여 준다. MLLF/MP는 시간 0에서 가장 작은 여유시간을 가진 태스크 T_1과 T_3을 수행 선택하여 각각 $D_1(0) - L_1(0) = 5$, $D_2(0) - L_3(0) = 5$만큼씩 보다 먼저 수행시키게 된다. 그러나 시간 3에서 T_2의 여유시간이 0이 되므로 P_2에서 수행되던 T_3을 선점하여 T_2를 P_2에서 수행시킨다. 시간 5에서 T_1이 수행 종료하고 T_3가 수행된다. 두 스케줄링 알고리즘은 표 3의 태스크 집합에 대해 각각 2번의 문맥 교환을 일으킨다.

EDZL은 3개 이상의 프로세서를 가진 다중 프로세서 시스템에서는 최적이 아닌 반면 MLLF/MP는 2개 이상의 프로세서의 개수를 가진 다중 프로세서 시스템에서 LLF와 같이 준최적이다. 표 4는 3개의 프로세서를 가진 다중 프로세서 시스템에서 EDZL은 실행 가능한 스케줄을 찾지 못하지만 MLLF/MP는 실행 가능한 스케줄을 찾는 태스크 집합의 예를 보여 준다. 그림 7은 표 4의 태스크 집합에 대한 EDZL과 MLLF/MP의 스케줄을 보여 준다.

표 4 태스크 집합 예제 4

	잔여 수행시간	마감시간	여유시간
T_1	2	2	0
T_2	2	2	0
T_3	4	5	1
T_4	4	5	1
T_5	1	4	3

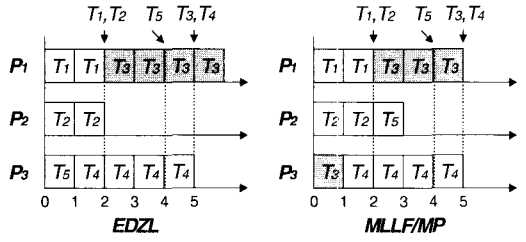

그림 7 표 4의 태스크 집합에 대한 EDZL와 MLLF/MP의 스케줄

4.3 MLLF/MP 스케줄링 알고리즘의 특성

본 절에서는 수학적 증명을 통하여 MLLF/MP가 준최적임을 보인다.

정리 1 LLF에 의해 스케줄링 가능한 태스크 집합에 대해 MLLF/MP는 실행 가능한 스케줄을 만들어 낸다.

증명 : 시간 t에서 LLF에 의해 스케줄링 가능한 태스크 집합이 MLLF/MP에 의해 스케줄링 되었을 때 시간 $t+1$의 태스크 집합은 여전히 LLF에 의해 스케줄링 가능함을 증명한다.

LLF와 MLLF/MP 모두 동시에 최대 m개의 태스크를 수행시킬 수 있다. LLF는 가장 작은 여유시간을 가진 m개의 태스크를 수행시킨다. 시간 t에서 두 스케줄링 알고리즘이 수행시킨 태스크들이 같다면 시간 $t+1$의 태스크 집합은 여전히 LLF에 의해 스케줄링 가능하다. 시간 t에 여유시간이 0이 되는 태스크에 대해서는 두 스케줄링 알고리즘 모두 우선적으로 수행시키므로 이러한 태스크는 LLF와 MLLF/MP에 의해서 수행된다. 여기서는 두 스케줄링 알고리즘이 서로 다르게 수행시키는 태스크에 대해서만 고려한다. 시간 t에서 LLF는 T_j를 수행시키지 않고 T_i를 수행시키고 MLLF/MP는 T_i를 수행시키지 않고 T_j를 수행시키는 태스크 T_i와 T_j가 존재한다고 하자. LLF 와 MLLF/MP가 서로 다른 태스크를 수행시키기 위해서는 $L_i(t)<L_j(t)$를 만족해야 한다. 만약 $L_i(t) \geq L_j(t)$이라면 두 스케줄링 알고리즘은 모두 T_j를 수행시킬 것이기 때문이다. MLLF/MP가 시간 t에서 T_j를 선택하여 수행시키는 이유는 시간 t보다 이전 시간 t'에서 $L_j(t') \leq L_i(t')$와 $L_j(t') < L_{ED}(t')$를 만족하였기 때문이다. 그리고 시간 t에서 $D_{ED}(t)-L_j(t)>0$를 만족하고 $L_j(t)>0$이기 때문이다. T_{ED}는 T_i와 같거나 보다 빠른 마감시간을 가진 태스크이다. 즉, $D_{ED}(t) \leq D_i(t)$이다. 그러므로 식 (1)을 만족한다.

$$0<L_i(t)<L_j(t)<D_{ED}(t) \leq D_i(t) \quad (1)$$

식 (1)을 통해 $L_i(t)<L_j(t)<D_i(t)$임을 알 수 있는데 이 식의 각 항에서 $L_i(t)$를 감산하면 식 (2)를 얻는다.

$$0<L_j(t)L_i(t)<E_i(t) \quad (2)$$

식 (2)를 보면 T_i의 잔여수행시간 $E_i(t)$가 $L_j(t)-L_i(t)$보다 크므로 LLF에서는 T_i가 $E_i(t)$만큼 수행되기 이전에 T_i와 T_j의 여유시간이 같아지므로 T_j는 T_i가 $E_i(t)$만큼 수행되기 이전에 어떤 프로세서에서 적어도 한번 수행된다. LLF에 의해서 시간 t에서 수행되는 T_i의 수행시간과 T_i가 $E_i(t)$만큼 수행되기 이전에 T_j가 수행된 수행시간을 교환하여도 T_i와 T_j의 마감시간은 어기지 않는다.

만약 시간 교환을 하였을 때 그림 8과 같이 T_i가 동시에 두 프로세서에서 수행되도록 시간이 교환되는 경우가 발생할 수 있다. 그러나 $L_i(t)>0$이기 때문에 T_i와 T_j가 수행시간 교환을 하여도 T_i는 마감시간을 놓치지

않는다. 그러므로 시간 t에서 LLF에 의해 스케줄링 가능한 태스크 집합이 MLLF/MP에 의해 스케줄링 되었을 때 시간 $t+1$의 태스크 집합은 여전히 LLF에 의해 스케줄링 가능하다.

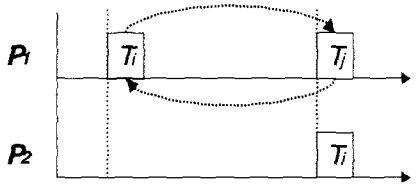

그림 8 수행시간 교환

정리 2 MLLF/MP는 준최적 스케줄링 알고리즘이다.

증명 : 이는 준최적성의 정의와 LLF의 준최적성, 정리 1에 의해서 증명된다.

5. 모의 실험

본 장에서는 모의 실험 결과를 제시함으로써 제안한 MLLF/MP 스케줄링 알고리즘의 향상된 성능을 보인다. 이 모의 실험에서는 두 개 이상의 프로세서를 갖는 다중 프로세서 시스템에서 LLF, EDZL, MLLF/MP의 문맥교환 발생비율과 스케줄링 성공비율을 비교한다.

본 실험에서는 크게 프로세서의 개수가 변하는 경우와 태스크의 평균 여유시간이 변하는 경우로 나누어 수행한다. 각 실험에 대한 성능 분석의 척도로써 문맥교환 발생비율과 스케줄링 성공비율을 비교한다. 문맥교환 발생비율의 비교는 비교 대상의 스케줄링 알고리즘이 모두 실행 가능한 스케줄을 갖는 태스크 집합에 대해서만 수행하였다.

본 실험에서는 2, 4, 6, 8 개의 프로세서를 가진 다중 프로세서 시스템을 고려한다. 각 다중 프로세서 시스템에 대해서 서로 다른 임의의 태스크 집합 1000개를 모의 실험하여 평균을 구하였다. 본 실험에서는 다음과 같은 가정을 한다.

- 실험 대상인 다중 프로세서 시스템의 프로세서 개수를 m=2, 4, 6, 또는 8이라고 한다.
- 각 태스크 집합은 30×m개의 태스크를 가진다.
- 각 태스크는 수행시작시간과 최악수행시간, 여유시간의 속성을 갖는다. 상대적 마감시간은 최악수행시간과 여유시간의 합으로 구해진다.
  - 수행시작시간은 포아송 분포(poisson distribution)를 갖는다. λ는 단위 수행시간에 발생할 수 있는 태스크의 개수로써 본 실험에서는 이 값을 0.01×m으로 지정한다.
  - 최악수행시간은 정규 분포(normal distribution)를 갖는다. 최악수행시간의 평균은 50으로 지정하였고 최악수행시간의 표준편차는 25로 지정하였다.
  - 여유시간은 정규 분포(normal distribution)를 갖는다. 프로세서의 개수가 변하는 경우의 실험에서는 여유시간의 평균은 25로 지정하였고 여유시간의 표준편차를 10으로 지정하였다. 평균 여유시간이 변하는 경우의 실험에서는 평균 여유시간을 15에서 35까지 단위 5씩 변화 시키면서 수행하였다. 그리고 각 평균 여유시간에 대해 여유시간의 표준편차는 10으로 고정하였다.

본 실험에서는 스케줄링 가능성을 높이기 위해 태스크 집합을 구성하는 태스크의 개수를 적게 하는 대신 각 다중 프로세서 시스템에 대해 실험하는 태스크 집합의 개수를 크게 하였다. 그리고 태스크 사이의 여유시간 충돌이 발생하는 확률을 증가시키기 위해서 최악수행시간의 크기를 크게 하였다.

5.1 프로세서의 개수가 변하는 경우

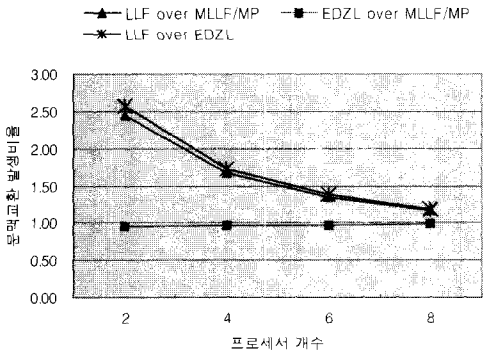

그림 9 프로세서 개수 변화에 따른 문맥교환 발생비율

그림 9는 프로세서 개수의 변화에 따른 LLF, EDZL, MLLF/MP의 문맥교환 발생비율을 보여 준다. 실험 결과를 통해 EDZL과 MLLF/MP는 LLF에 비해 적은 문맥교환 횟수를 가짐을 알 수 있다. 그림 9에서 프로세서 개수가 증가할수록 MLLF/MP와 EDZL에 대한 LLF의 문맥교환 발생비율이 감소하는 것을 볼 수 있다. LLF에서는 프로세서의 개수가 증가할수록 프로세서에서 수행되는 태스크의 개수도 역시 증가하기 때문에 수행되지 않음으로써 여유시간이 줄어드는 태스크의 개수가 감소

되고 결국 여유시간 충돌이 일어날 확률은 감소하게 된다. 만일 여유시간 충돌이 발생하더라도 프로세서에서 수행되고 있는 모든 태스크들이 동시에 여유시간 충돌이 발생하는 것이 아니라 일부 태스크들 사이에서 먼저 발생하게 되고 시간이 여유시간 충돌이 발생하는 태스크의 개수가 갈수록 서서히 증가하게 된다. 반면 시간이 지나갈수록 현재 프로세서에서 수행되는 태스크의 잔여 수행시간은 계속 줄어들고 혹시 여유시간 충돌이 발생하더라도 적은 수의 여유시간 충돌을 일으킨 후에 수행을 종료하게 된다. 그러므로 프로세서의 개수가 증가할수록 LLF는 적은 문맥교환 횟수를 갖게 된다. MLLF/MP는 여유시간 충돌이 발생하는 것과 상관없이 예산시간을 지정하여 태스크를 수행시키므로 프로세서 개수의 증가에 따른 문맥교환 횟수의 감소는 LLF에 비하여 작다. EDZL도 여유시간 충돌이 발생하는 것과 상관없이 마감시간이 빠른 태스크를 우선적으로 수행시키므로 프로세서 개수의 증가에 따른 문맥교환 횟수의 감소는 LLF에 비하여 작다. 그러므로 프로세서의 개수가 증가할수록 MLLF/MP와 EDZL에 대한 LLF의 문맥교환 발생비율은 줄어들게 된다. 그림 10은 프로세서 개수의 변화에 따른 LLF에 대한 EDZL, MLLF/MP의 스케줄링 성공비율을 보여 준다.

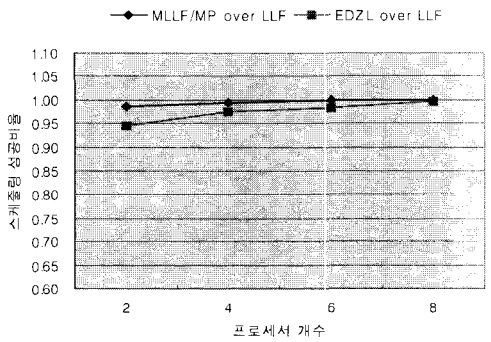

그림 10 프로세서 개수 변화에 따른 스케줄링 성공비율

5.2 평균 여유시간이 변하는 경우

그림 11은 프로세서의 개수가 4개일 경우 태스크의 평균 여유시간의 변화에 따른 LLF, EDZL, MLLF/MP의 문맥교환 발생비율을 보여 준다. 그림 11을 보면 태스크의 평균 여유시간이 증가함에 따라 MLLF/MP와 EDZL에 대한 LLF의 문맥교환 발생비율이 증가됨을 볼 수 있다. 여유시간의 표준편차를 고정하였기 때문에 평균 여유시간이 변하더라도 태스크들 사이의 여유시간 차이는 크게 변하지 않는다. 그리고 태스크의 최악수행시간도 고정되었기 때문에 평균 여유시간이 변하더라도 여유시간 충돌이 발생할 확률은 거의 변하지 않는다. 다만 태스크의 상대적 마감시간이 길어지기 때문에 스케줄링 성공률은 증가하게 된다. 그러므로 LLF는 평균 여유시간의 변화에 상관 없이 거의 비슷한 문맥교환 횟수를 보인다.

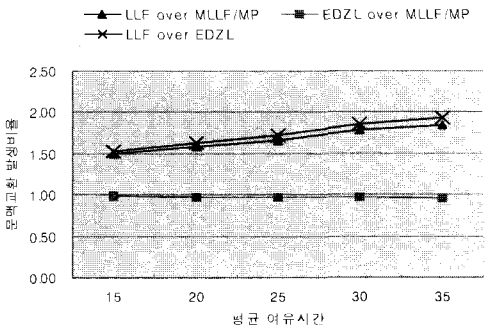

그림 11 평균 여유시간 변화에 따른 문맥교환 발생비율

LLF는 스케줄링 시점마다 가장 작은 여유시간을 가진 태스크들을 항상 수행시킨다. 그러므로 여유시간 충돌이 발생한 경우 여유시간 충돌이 일어난 태스크들은 번갈아 수행되기 때문에 여유시간이 충돌 된 태스크들 사이의 여유시간 차이는 항상 그리 크지 않다. 그러나 MLLF/MP의 경우 여유시간 역전이 발생하더라도 현재 수행하던 태스크를 계속 수행시키고 EDZL의 경우 마감시간이 빠른 태스크를 우선적으로 계속 수행시키기 때문에 수행되지 않는 태스크가 여유시간 0을 갖게 되는 속도는 LLF에 비해 매우 빠르며 태스크의 여유시간이 0이 될 확률 또한 LLF보다 커지게 된다. MLLF/MP와 EDZL은 프로세서에서 수행되지 않는 태스크가 0이라는 여유시간을 갖게 되면 반드시 문맥교환을 일으키게 된다. 그러나 평균 여유시간이 증가할수록 MLLF/MP와 EDZL에 의해 여유시간 0을 가진 태스크들이 발생할 확률은 줄어들게 되고 이로 인해 문맥교환이 발생할 확률 또한 줄어들게 된다. 그러므로 태스크의 평균 여유시간이 커짐에 따라 LLF의 문맥교환 횟수는 거의 변함이 없는 반면 MLLF/MP와 EDZL의 문맥교환 횟수가 줄어들기 때문에 MLLF/MP와 EDZL에 대한 LLF의 문맥교환 발생비율은 높아지게 된다. 그림 12는 프로세서 개수의 변화에 따른 LLF에 대한 EDZL, MLLF/MP의 스케줄링 성공비율을 보여 준다.

이상의 실험 결과를 통하여 준최적인 MLLF/MP는 LLF에 비하여 보다 작은 문맥교환을 일으킴으로써 LLF에 비해 스케줄링 오버헤드가 적음을 알 수 있다.

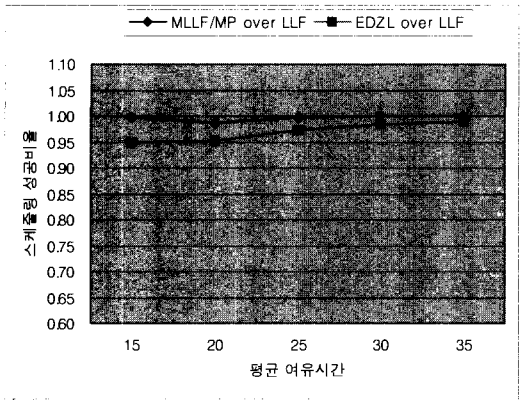

그림 12 평균 여유시간 변화에 따른 스케줄링 성공률

6. 결 론

LLF는 단일 프로세서 시스템에서 최적이고 다중 프로세서 시스템에서 준최적이라는 이론적인 우수성을 가졌음에도 불구하고 여유시간 충돌이 발생하는 경우 빈번한 문맥교환을 일으키는 단점으로 인해 실용적이지 못하였다. 본 논문에서 제안한 MLLF/MP는 다중 프로세서 시스템에서 LLF와 같이 준최적이지만 보다 적은 문맥교환 오버헤드를 갖는 스케줄링 알고리즘이다. MLLF/MP는 여유시간 충돌이 발생하였을 때 여유시간 역전이 발생하더라도 태스크를 연속시켜 수행함으로써 빈번한 문맥교환을 막아 스케줄링 오버헤드를 줄인다. 본 논문에서는 MLLF/MP의 준최적성을 수학적으로 증명하였으며 모의 실험 결과를 통하여 LLF에 비해 적은 문맥교환 발생비율을 가짐을 보였다.

LLF와 MLLF/MP는 모두 재스케줄링 시 가장 작은 여유시간을 가진 태스크들을 프로세서 개수 만큼 찾아 수행 선택한다. 그러나 MLLF/MP는 수행 선택되지 않은 태스크 중에서 가장 빠른 마감시간을 가진 태스크를 찾아 태스크의 예산시간을 계산해야 하는 추가적인 오버헤드를 갖는다. 재스케줄링 오버헤드 자체만으로는 MLLF/MP가 LLF보다 크다. 그러나 LLF는 매 시점마다 재스케줄링을 해야 하는 반면 MLLF/MP는 태스크의 예산시간이 모두 소모되거나 여유시간이 0이 된 태스크가 발생했을 경우에만 재스케줄링이 일어나므로 LLF와 같이 매 시점마다 재스케줄링을 하지 않는다. 이렇듯 전체적인 스케줄링 오버헤드 비교는 단편적으로 이루어 질 수 없으며 구현 방법과 실제 태스크 구성에 따라 달라질 수 있다. 그리고 LLF, EDZL, MLLF/MP와 같이 태스크의 여유시간을 고려하는 스케줄링 알고리즘은 공통적으로 매 시간 마다 태스크의 여유시간을 변경하고 관리야 하는 스케줄링 오버헤드를 갖는다. 그러므로 제안한 MLLF/MP 뿐만 아니라 LLF, EDZL 구현을 위한 효율적인 자료구조 및 접근 방법의 연구가 필요하다.

참 고 문 헌

[1] M.L. Dertouzos and A.K. Mok, "Multiprocessor On-Line Scheduling of Hard-Real-Time Tasks," IEEE Transactions on Software Engineering, Vol. 15, No. 12, December 1989

[2] 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

[3] 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

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

[5] 오성혼, 양승민, "실시간 시스템을 위한 최소 여유시간 우선 기반의 최소 선점을 갖는 스케줄링 알고리즘," 한국정보과학회 논문지(A), 제26권, 제4호, pp.443-454, 1999년 4월

[6] 조성제, 이석균, 유해영, "산발적인 경성 실시간 태스크를 위한 온라인 스케줄링 알고리즘," 한국정보과학회 논문지(A), 제25권, 제7호, pp. 708-718, 1998년 7월

[7] S.K. Dhall and C.L. Liu, "On a Real-Time Scheduling Problem," Operations Research, Vol. 26, No. 1, pp. 127-140, February 1978.

[8] Michael B. Jones, Joseph S. Barrera III, Alessandro Forin, Paul J. Leach, Daniela Rosu and Marcel-Catalin Rosu, "An Overview of the Rialto Real-Time Architecture," In Proceedings of the Seventh ACM SIGOPS European Workshop, Connemara, Ireland, pages 249-256, September, 1996

오 성 훈

1996년 2월 인천대학교 정보통신공학(학사). 1998.8 숭실대학교 컴퓨터학과(석사).2002.2 숭실대학교 컴퓨터학과(박사). 2002.3 ~ 현재 (주)디지캡 책임 연구원. 관심분야는 Real-Time Scheduling, Operating System, Real-Time Communication 등

양 승 민

1978년 2월 서울대학교 전자공학(학사). 1978 ~ 1981 삼성전자 연구원. 1983년 4월 Univ. of South Florida 전산학(석사). 1986년 12월 Univ. of South Florida 전산학(박사). 1988 ~ 1993 Univ. of Texas at Arlington 조교수. 1993 ~ 현재 숭실대학교 컴퓨터학부 교수. 1996 ~ 1998 국회도서관 정보처리국장. 관심분야는 Real-Time System, Operating System, Fault-Tolerant System 등

길 아 라

1987년 2월 이화여자대학교 전산학과 이학사. 1990년 2월 한국과학기술원 전산학과 석사. 1997.2한국과학기술원 전산학과 박사. 1995.3 ~ 1997.1(주)새롬 기술선임연구원.1997 ~ 현재 숭실대학교 컴퓨터학부 교수. 관심분야는 Real-time Operating System, Real-time Communication, Multimedia Networking, Computer-Telephony Integration 등