다중처리기 상의 실시간 태스크를 위한 고정 우선순위 스케줄링 알고리즘의 성능 향상

(Enhancing Fixed Priority Scheduling Algorithms for Real-Time Tasks on Multiprocessors)

박 민 규 † 한 상 철 †† 김 희 헌 ††† 조 성 제 †††† 조 유 근 †††††
(Minkyu Park) (Sangchul Han) (HeeHeon Kim) (Seongje Cho) (Yookun Cho)

요 약 본 논문은 다중처리기 상에서 고정 우선순위 스케줄링 알고리즘의 성능을 향상시키는 기법을 제시한다. 이 기법은 여유시간(laxity)이 0인 작업(job)에 가장 높은 우선순위를 부여하고 다른 작업들보다 우선적으로 스케줄 한다. 이 기법을 적용한 고정 우선순위 알고리즘은 원래의 알고리즘을 strictly - dominate한다. 즉, 원래의 고정 우선순위 알고리즘이 스케줄 할 수 있는 모든 태스크 집합(task set)을 스케줄 할 수 있으며, 원래의 고정 우선순위 알고리즘이 스케줄하지 못하는 일부 태스크 집합을 스케줄 할 수 있다. 또한 모의실험을 통하여 제안한 기법을 적용한 알고리즘이 스케줄 가능한 태스크 집합의 수와 스케줄 보장 이용률(schedulable utilization bound) 측면에서 원래의 알고리즘보다 우수함을 보인다.

키워드 : 다중처리기, 실시간, 우선순위 스케줄링, 여유시간, 스케줄 보장 이용률

Abstract This paper presents a scheme to enhance fixed priority scheduling algorithms on multiprocessors. This scheme gives the highest priority to jobs with zero laxity and schedules them prior to other jobs. A fixed priority algorithm employing this scheme strictly dominates the original one; it can schedule all task sets schedulable by the fixed priority algorithm and some task sets not schedulable by the fixed priority algorithm. Simulation results show that the proposed scheme improves fixed priority algorithms in terms of the number of schedulable task sets and schedulable utilization bound.

Key words : multiprocessor, real-time, priority-driven scheduling, laxity, schedulable utilization bound

1. 서 론

실시간 시스템을 위한 우선순위 알고리즘(priority driven algorithm)은 정적 우선순위(static priority) 알고리즘과 (작업수준) 고정 우선순위(job-level fixed priority) 알고리즘, (작업수준) 동적 우선순위(job-level dynamic priority) 알고리즘으로 분류된다. 정적 우선순위 알고리즘은 각 태스크(task)에게 고유한 우선순위를 부여하고, 태스크가 생성한 모든 작업(job)은 태스크에게 부여된 우선순위와 동일한 우선순위를 갖는다. RM(Rate Monotonic)[1]과 DM(Deadline Monotonic)[2]이 대표적인 정적 우선순위 알고리즘이다. 고정 우선순위 알고리즘에서는 생성되는 작업마다 다른 우선순위가 부여되며 일단 부여된 우선순위는 작업이 수행을 마칠 때까지 변하지 않는다. 이 부류에 속한 알고리즘으로는 EDF(Earliest Deadline First)[1], EDF-US[m/(2m-1)][3], fpEDF[4] 등이 있다. 동적 우선순위 알고리즘에서는 작업의 우선순위가 시간에 따라 변할 수 있다. LLF (Least Laxity - First)[5]는 잘 알려진 동적 우선순위 알고리즘이다.

고정 우선 순위 알고리즘의 장점은 구현이 용이하고 선점(preemption) 횟수가 적다는 것이다. 전체 선점 횟수는 작업의 수에 의해 한정된다[6]. 그러나 고정 우선

· 본 연구는 두뇌한국21 사업과 서울대학교 컴퓨터연구소의 지원으로 수행되었음

† 정 회 원 : 서울대학교 컴퓨터공학과
mkpark@ssrnet.snu.ac.kr
†† 학생회원 : 서울대학교 컴퓨터공학과
schan@ssrnet.snu.ac.kr
††† 비 회 원 : 서울대학교 컴퓨터공학과
hhkim@ssrnet.snu.ac.kr
†††† 정 회 원 : 단국대학교 정보컴퓨터학부 교수
sjcho@dku.edu
††††† 종신회원 : 서울대학교 컴퓨터공학부 교수
cho@cse.snu.ac.kr

논문접수 : 2004년 8월 16일
심사완료 : 2004년 9월 22일

순위 알고리즘의 처리기 이용률은 만족스럽지 않다. Baruah는 m개의 처리기 상에서 고정 우선순위 스케줄링 알고리즘의 스케줄 보장 이용률(schedulable utilization bound)이 $(m+1)/2$을 넘지 못함을 보였다[4]. 한편, 동적 우선순위 알고리즘은 다중처리기 상에서 고정 우선순위 알고리즘보다 더 우수하고[7] 더 높은 스케줄 보장 이용률을 갖는다고 알려져 있다. 그러나 동적 우선순위 알고리즘은 많은 수의 선점을 유발한다.

본 논문은 고정 우선순위 알고리즘의 성능을 향상시키는 기법을 제안한다. 이 기법은 여유시간(laxity)이 0인 작업에게 최고 우선순위를 부여하여 다른 작업들보다 우선적으로 스케줄 한다. 여유시간이 0인 작업은 즉시 스케줄 되지 않으면 마감시간(deadline)을 지키지 못한다. 여유시간이 0보다 큰 나머지 작업들에게는 고정 우선순위 알고리즘에 따라 우선순위를 부여한다. 이 기법은 임의의 고정 우선순위 알고리즘에 적용할 수 있으며, 이 기법을 적용한 고정 우선순위 알고리즘은 원래의 알고리즘을 strictly dominate한다. 즉, 원래의 알고리즘이 스케줄 할 수 있는 모든 태스크 집합(task set)을 스케줄 할 수 있고, 원래의 알고리즘이 스케줄 하지 못하는 일부 태스크 집합을 스케줄 할 수 있다. 본 논문에서는 무작위 생성(randomly generated) 태스크 집합을 사용한 모의실험을 통해 제안한 기법을 평가하였다. 모의실험 결과, 제안한 기법은 스케줄 가능한 태스크 집합의 수와 스케줄 보장 이용률 측면에서 고정 우선순위 스케줄링 알고리즘의 성능을 향상시킨다.

본 논문의 구성은 다음과 같다. 2장에서는 시스템 모델과 고정 우선순위 스케줄링 알고리즘을 간략히 설명한다. 3장에서는 제안한 기법을 제시하고 이 기법을 채택한 고정 우선순위 알고리즘의 특성을 논의한다. 4장에서는 모의실험을 통해 제안한 기법을 평가하고, 5장에서 결론을 맺는다.

2. 고정 우선순위 알고리즘

본 논문은 m개의 동일 처리기(identical processor) 상에서 주기 태스크(periodic task)의 선점 스케줄링(preemptive scheduling)을 다룬다.

태스크 집합 $\tau=\{\tau_1, \tau_2, \ldots, \tau_n\}$는 n개의 독립적인 주기 태스크로 구성되며, 주기 태스크는 $\tau_i=(C_i, P_i)$로 정의된다. C_i는 최악 수행시간(worst case execution - time)이고 P_i는 주기를 나타낸다. 모든 태스크는 동시에 시작하며 각 태스크의 상대적 마감시간(relative deadline)은 주기와 같다고 가정한다. 각 태스크는 시각 $(k-1)P_i$마다 작업을 생성하고, 이 작업은 마감시간 kP_i까지 C_i만큼의 수행을 요구한다. 작업 J_i의 상태는 (c_i, d_i)로 표현되며, c_i는 남은 수행시간, d_i는 (절대적) 마감시간까지 남은 시간이다. 작업이 도착(arrive)할 때 $c_i=C_i$, $d_i=P_i$으로 주어지며, c_i와 d_i는 시간에 따라 변한다. 작업의 여유시간은 $l_i=d_i-c_i$으로 정의된다. 주기 태스크의 이용률(utilization)은 $u_i=C_i/P_i$으로 정의하고 태스크 집합의 전체 이용률은 $U(\tau)=\Sigma_{i=1}^{n}u_i$으로 정의한다.

EDF는 마감시간이 가장 빠른 작업에게 가장 높은 우선순위를 부여한다. EDF는 효율적 구현이 가능하며 전체 선점회수가 전체 작업의 수의 2배를 넘지 않는다[6]. 그러나 다중처리기 상에서 EDF의 스케줄 보장 이용률은 매우 낮을 수 있다. 특히, Dhall 등은 EDF가 이용률이 높은 태스크와 낮은 태스크가 혼합되어 있는 태스크 집합을 잘 스케줄 하지 못함을 보였다[8].

EDF-US[m/(2m-1)]은 태스크들을 이용률이 높은 태스크 그룹과 이용률이 낮은 태스크 그룹으로 나눈다. 이용률이 $m/(2m-1)$보다 높은 태스크에 의해 생성된 작업에게는 최고 우선순위를 부여하고, 이용률이 $m/(2m-1)$보다 낮은 태스크에 의해 생성된 작업에게는 EDF에 따라 우선순위를 부여한다. 이 알고리즘은 전체 이용률이 $m^2/(2m-1)$ 이하인 모든 태스크 집합을 스케줄 할 수 있다. 이 후로 본 논문에서는 EDF-US[m/(2m-1)]을 EDF-US로 표기한다.

정리 1. (Srinivasan and Baruah [3]) EDF-US[m/(2m-1)]은 m개의 처리기상에서 전체이용률이 $U(\tau)\le m^2/(2m-1)$인 임의의 주기 태스크 집합을 스케줄 할 수 있다.

fpEDF는 이용률이 1/2을 넘는 태스크에 의해 생성된 작업 중 이용률이 가장 높은 태스크의 작업부터 최대 $m-1$개의 작업에게 최고 우선순위를 부여하고, 나머지 작업들은 EDF에 의해 우선순위를 부여한다. fpEDF는 전체 이용률이 $(m+1)/2$ 이하인 모든 태스크 집합을 스케줄 할 수 있다.

정리 2. (Baruah[4]) fpEDF은 m개의 처리기 상에서 전체 이용률이 $U(\tau)\le(m+1)/2$인 임의의 주기 태스크 집합을 스케줄 할 수 있다.

3. ZL기법(Zero Laxity Scheme)

본 논문은 여유시간이 0보다 큰 작업보다 여유시간이 0인 작업을 우선적으로 스케줄 하는 ZL기법을 제시한다. 여유시간이 0인 작업은 즉시 스케줄 되지 않으면 마감시간을 지키지 못하므로 긴급한 작업이라고 할 수 있다. 따라서 ZL기법은 긴급한 작업이 긴급하지 않은 다른 작업보다 우선적으로 스케줄 될 수 있도록 한다. 그림 1은 ZL기법을 적용한 고정 우선순위 알고리즘의 기술이다. FA는 임의의 고정 우선순위 알고리즘이고, FA/ZL은 ZL기법

```
1: BEGIN
2: Jobs are assigned priorities according to FA
3: FOR each ready job J DO
4:       IF J.laxity == 0 THEN
5:             J.priority = ∞
6:       END IF
7: EDN FOR
8: END
```

그림 1 FA/ZL 알고리즘

을 적용한 FA이다.

ZL기법이 부여한 우선순위는 FA가 부여한 우선순위보다 항상 높다. 예를 들어, EDF-US는 이용률이 $m/(2m-1)$보다 높은 태스크의 작업에게 최고 우선순위를 부여한다. 그러나 EDF-US/ZL은 여유시간이 0인 작업에게 최고 우선순위를 부여하고, 여유시간이 0보다 크고 이용률이 $m/(2m-1)$보다 높은 태스크의 작업에게 두 번째로 높은 우선순위를 부여한다. 따라서 어떤 작업의 여유시간이 0이 되면, 그 작업은 여유시간이 0보다 큰 임의의 작업을 선점할 수 있다.

3.1 FA/ZL 알고리즘의 특성

스케줄링 알고리즘 A가 m개의 처리기 상에서 태스크 집합 $\tau=\{\tau_1, \tau_2, \ldots, \tau_n\}$을 스케줄 하고 ($m<n$), 시각 t에 알고리즘 A가 부여한 우선순위가 가장 높은 m개의 작업의 집합을 $\sigma_A(t)=\{j_1, j_2, \ldots, j_m\}$라고 하자. 또한, 작업들의 배열 순서는 우선순위 순서라고 하자. 즉, j_i의 우선순위는 j_{i+1}의 우선순위보다 높다. 정리 3은 고정 우선순위 스케줄링 알고리즘 FA가 m개의 처리기 상에서 태스크 집합 τ을 스케줄 할 수 있으면 FA에 의한 스케줄과 FA/ZL에 의한 스케줄이 동일함을 보인다.

정리 3. 태스크 집합 τ가 고정 우선순위 알고리즘 FA에 의해 스케줄 가능하면, 임의의 시각 t에 $\sigma_{FA}(t)=\sigma_{FA/ZL}(t)$이다.

증명. 시각 t에 준비된 작업(ready job)의 집합이 $\{j_1, j_2, \ldots, j_k\}$이고 FA에 의한 우선순위 순서라고 하자 ($k \le n$). 준비된 작업 중에 우선순위가 가장 높은 작업에게 처리기가 할당되므로 $\sigma_{FA}(t)=\{j_1, j_2, \ldots, j_m\}$이다. 이 때, $\sigma_{FA}(t)$의 부분집합 $\{j_1', j_2', \ldots, j_s'\}$을 고려하자 ($s \le m$). 이 집합에 속한 작업들은 여유시간이 0보다 크고, 우선순위 순서이다.

그러면, $\sigma_{FA}(t)-\{j_1', j_2', \ldots, j_s'\}$은 $\sigma_{FA}(t)$에 속한 작업 중에 여유시간이 0인 작업의 집합이다.

$\{j_{m+1}, j_{m+2}, \ldots, j_k\}$에 속한 모든 작업은 여유시간이 0보다 크다. 만약 $\{j_{m+1}, j_{m+2}, \ldots, j_k\}$에 속한 어떤 작업의 여유시간이 0이면 이 작업은 마감시간을 지키지 못하므로 태스크 집합 τ가 FA에 의해 스케줄 가능하다는 가정에 모순이다.

FA/ZL은 여유시간이 0인 작업에게 최고 우선순위를 부여하므로 $\sigma_{FA}(t)-\{j_1', j_2', \ldots, j_s'\}$에 속한 $m-s$개의 작업들은 $\sigma_{FA/ZL}(t)$에 속한다. 또한, FA/ZL은 여유시간이 0보다 큰 작업 $\{j_1', j_2', \ldots, j_s', j_{m+1}, j_{m+2}, \ldots, j_k\}$ 중에서 s개의 작업을 추가로 선택한다. FA/ZL의 정의에 의하여 우선순위가 가장 높은 $j_1', j_2', \ldots, j_s'$을 선택한다. 그러므로

$$\sigma_{FA/ZL}(t)=(\sigma_{FA}(t)-\{j_1', j_2', \ldots, j_s'\}) \cup \{j_1', j_2', \ldots, j_s'\} = \sigma_{FA}(t). \quad \square$$

스케줄링 알고리즘 A_2가 스케줄 할 수 있는 모든 태스크 집합을 스케줄링 알고리즘 A_1이 스케줄 할 수 있으면, A_1은 A_2를 dominate한다고 말한다. 또한, A_1이 A_2를 dominate하고, A_1은 스케줄 할 수 있지만 A_2는 스케줄 할 수 없는 태스크 집합이 존재하면 A_1은 A_2를 strictly dominate한다고 말한다.

Dominate 관계는 알고리즘의 스케줄 능력을 비교하는 좋은 기준이다. 하지만 dominate 관계가 알려진 알고리즘들은 많지 않다. 특히, Kalyanasundaram 등은 동적 우선순위 알고리즘인 LLF가 고정 우선순위 알고리즘인 EDF를 dominate하지 못함을 보였다[9]. 정리 3은 임의의 고정 우선순위 알고리즘 FA에 대해, FA가 스케줄 할 수 있는 임의의 태스크 집합이 FA/ZL에 의해 스케줄 가능하다, 즉, FA/ZL이 FA를 dominate함을 의미한다. 따라서 스케줄러의 수행 오버헤드(runtime overhead)를 무시할 때 FA/ZL은 FA보다 우수하다고 말할 수 있다. 다음의 따름정리는 정리 3으로부터 얻어지는 결과이다.

따름정리 1. EDF/ZL은 EDF를 strictly dominate한다.

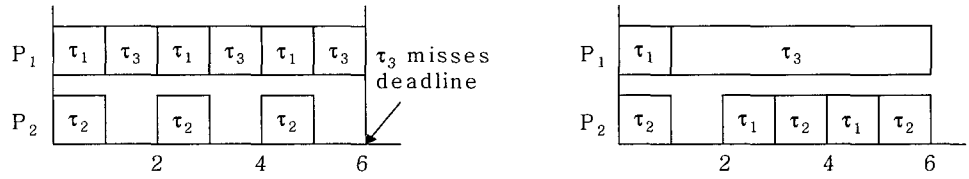
증명. 정리 3에 의해 EDF/ZL은 EDF를 dominate한다. 또한, 2개의 처리기 상에서 EDF/ZL은 태스크 집합 $\tau=\{\tau_1=(1,2), \tau_2=(1,2), \tau_3=(5,6)\}$을 스케줄 할 수 있으나 EDF는 스케줄 할 수 없다. 그림 2는 2개의 처리기 상에서 τ의 스케줄이다.

따름정리 2. EDF-US/ZL은 EDF-US를 strictly dominate한다.

증명. 정리 3에 의해 EDF-US/ZL은 EDF-US를 dominate한다. 또한, 2개의 처리기 상에서 EDF-US/ZL은 태스크 집합 $\tau=\{\tau_1=(1,2), \tau_2=(3,4), \tau_3=(3,4)\}$을 스케줄 할 수 있으나 EDF-US는 스케줄 할 수 없다. 그림 3은 2개의 처리기 상에서 τ의 스케줄이다.

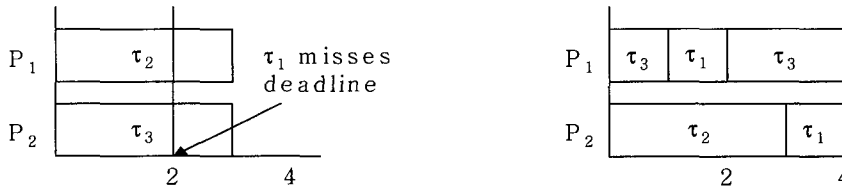
따름정리 3. fpEDF/ZL은 fpEDF를 strictly dominate한다.

증명. 정리 3에 의해 fpEDF/ZL은 fpEDF를 dominate한다. 또한, 2개의 처리기 상에서 fpEDF/ZL은 태스



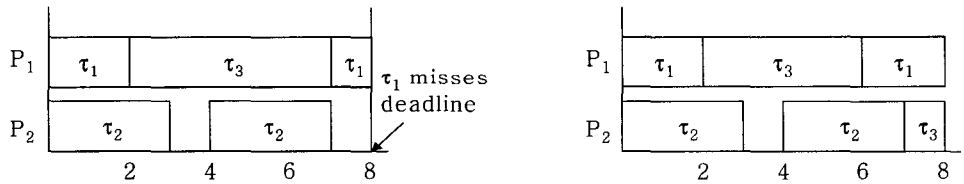
(a) EDF schedule (b) EDF/ZL scheduleA

그림 2 태스크 집합 $\tau = \tau_1 = (1,2), \tau_2 = (1,2), \tau_3 = (5,6)$



(a) EDF-US schedule (b) EDF-US/ZL schedule

그림 3 태스크 집합 $\tau = \tau_1 = (1,2), \tau_2 = (3,4), \tau_3 = (3,4)$



(a) fpEDF schedule (b) fpEDF/ZL schedule

그림 4 태스크 집합 $\tau = \{\tau_1 = (2,4), \tau_2 = (3,4), \tau_3 = (5,8)\}$

크 집합 $\tau = \{\tau_1 = (2,4), \tau_2 = (3,4), \tau_3 = (5,8)\}$을 스케줄 할 수 있으나 fpEDF는 스케줄 할 수 없다. 그림 4는 2개의 처리기 상에서 τ의 스케줄이다.

따름정리 4. EDF-US/ZL은 m개의 처리기 상에서 전체 이용률이 $U(\tau) \leq m^2/(2m-1)$인 임의의 주기 태스크 집합을 스케줄 할 수 있다.

증명. 정리 1과 정리 3에 의해 성립한다.

따름정리 5. fpEDF/ZL은 m개의 처리기 상에서 전체 이용률이 $U(\tau) \leq (m+1)/2$인 임의의 주기 태스크 집합을 스케줄 할 수 있다.

증명. 정리 2와 정리 3에 의해 성립한다.

3.2 ZL기법의 구현 방안

ZL기법은 여유시간을 참조하여 우선순위를 부여하기 때문에 시간의 흐름에 따라 작업의 우선순위가 바뀔 수 있다. 따라서 ZL기법은 고정 우선순위 스케줄링에서 발생하지 않는 수행 오버헤드를 수반하다. 본 절은 ZL기법의 수행 오버헤드에 대해 논의하고 효율적인 구현 방안을 제시한다.

LLF와 같이 여유시간을 참조하여 우선순위를 부여하는 알고리즘은 다음과 같은 수행 오버헤드를 수반한다. 첫째, 작업의 여유시간은 시간에 따라 변하므로 모든 준비된 작업(ready job)의 여유시간을 지속적으로 계산해야 한다. 둘째, 여유시간이 변함에 따라 우선순위를 지속적으로 변경해야 한다. 셋째, 우선순위가 변함에 따라 선점과 문맥교환이 발생한다.

ZL기법은 여유시간을 참조하여 우선순위를 부여하지만 수행 오버헤드는 크지 않다. 여유시간이 0이 된 작업은 최고 우선순위를 부여받으며, 그 우선순위는 작업이 완료할 때까지 변하지 않는다. 따라서 ZL기법을 적용함으로써 추가되는 우선순위 변경과 그로 인한 선점 및 문맥교환은 작업 당 최대 한 번이다.

또한 여유시간을 매 시간마다 계산하는 수행 오버헤드는 여유시간 큐(laxity queue)와 타이머를 이용하여 최소화할 수 있다. 그림 5는 처리기의 수가 3일 때 여유시간 큐의 일례(一例)이다. 준비된 작업은 총 7개이다. 이 중에 3개의 작업 j_1과 j_2, j_3은 처리기를 할당받아 수행 중이고 나머지 작업 j_4와 j_5, j_6, j_7은 우선순위가 낮아 처리기를 할당 받지 못하였다. 여유시간 큐의 각 노드는 현재 준비된 작업 중에 처리기를 할당받지 못한 작업들을 가리키며, 여유시간 순으로 정렬되어 있다. 여유시간

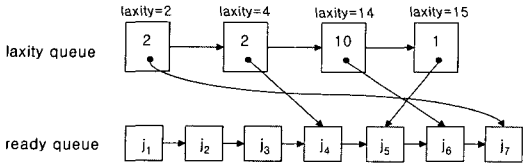


그림 5 여유시간 큐를 이용한 여유시간 계산 방안

큐의 첫째 노드는 여유시간이 가장 작은 작업의 여유시간을 기록하고, 나머지 노드는 앞 노드와의 여유시간 차이를 기록한다. 처리기를 할당받지 못한 작업의 여유시간은 모두 동일하게 감소하므로 여유시간의 차이만을 기록함으로써 나머지 노드의 갱신을 피할 수 있다.

새로 도착한 작업은 FA/ZL에 따라 우선순위를 부여받고 준비 큐(ready queue)에 삽입된다. 이 작업이 처리기를 할당 받지 못하면 여유시간 큐에 이 작업에 대한 노드가 삽입된다. 만약 이 작업이 다른 작업을 선점하면, 선점되어 수행이 중단된 작업에 대한 노드가 여유시간 큐에 삽입된다. 어떤 작업이 수행을 종료하면 그 작업을 수행하던 처리기는 다른 작업에게 할당 되고, 새로 처리기를 할당 받은 작업에 대한 노드가 여유시간 큐에서 삭제된다.

타이머는 여유시간 큐의 첫째 노드가 갱신될 때마다 재설정되며, 첫째 노드의 여유시간만큼 시간이 지난 후에 만기(expire)되도록 설정된다. 타이머가 만기되면 여유시간 큐의 첫째 노드를 삭제하고, 해당 작업에게 최고 우선순위를 부여한다. 이 때 선점이 발생하면 선점된 작업에 대한 노드가 여유시간 큐에 삽입된다.

FA/ZL은 LLF와 달리 매 시간마다 여유시간을 갱신할 필요가 없고, 여유시간 큐에 새로운 노드가 삽입될 때 그 노드에 대해서만 여유시간을 계산한다. 또한, 작업의 여유시간이 0이 되면 그 작업은 최고 우선순위를 받고 더 이상 우선순위의 변경이 없다. 따라서 우선순위 변경과 선점, 문맥교환 등에 의해 발생하는 수행 오버헤드가 크지 않다.

4. 실험 및 평가

본 실험은 무작위로 생성한 태스크 집합을 사용하여 제안한 ZL기법을 평가한다. 태스크 집합은 6개의 그룹, $G_u(u=2,3,\ldots,7)$으로 나누어 생성하였다. 각 그룹에는 25,000개의 주기 태스크 집합이 속하고, G_u에 속한 태스크 집합들은 전체 이용률이 구간 $(u-1, u]$에 균등하게 분포한다.

태스크 집합은 다음과 같이 생성하였다. 먼저 태스크 집합의 목표 이용률을 구간 $(u-1, u]$에서 균등하게 선택한다. 그 다음에 태스크 집합의 전체 이용률이 목표 이용률에 도달할 때까지 주기 태스크를 하나씩 생성하여 태스크 집합에 추가한다. 각 주기 태스크의 생성은 주기 생성 단계와 수행시간 생성 단계로 나뉘어 생성된다. 주기 생성 단계에서는 Goossens 등이 제안한 알고리즘[10]을 사용하여 주기를 생성한다. 이 알고리즘은 모의실험이 적절한 시간에 완료될 수 있도록 hyperperiod의 크기를 제어한다. 본 실험에서 hyper-period의 최대 크기는 $(2^4\times3^2\times5^2\times7\times11)$이고, 주기의 최대 크기는 300이다. 수행시간 생성 단계에서는 최악 수행시간을 $[1, period)$의 구간에서 무작위로 선택한다.

표 1은 m개의 처리기 상에서 전체 이용률이 m 이하인 모든 태스크 집합에 대해 각 알고리즘이 스케줄 성공한 태스크 집합의 백분율이다. ZL기법은 스케줄 가능한 태스크 집합의 수를 상당히 증가시켰다. 특히, EDF-/ZL은 EDF보다 10~16% 더 많은 태스크 집합을 스케줄 할 수 있었다. EDF-US와 fpEDF는 ZL 기법에 의해 6~7% 정도 향상되었다.

EDF는 이용률이 높은 태스크가 포함된 태스크 집합을 잘 스케줄 하지 못한다[4,8]. 이용률이 높은 태스크의 마감시간이 상대적으로 길면 그 태스크는 수행을 완료할 기회가 적다. 이용률이 높은 태스크의 작업은 초기 여유시간이 작으므로 여유시간이 곧 0이 되지만, 처리기는 마감시간이 짧은 다른 작업에게 할당되기 쉽다. 그러나 EDF/ZL은 여유시간이 0인 작업을 즉시 스케줄 하므로 이러한 상황에 잘 대처한다.

EDF-US와 fpEDF는 EDF의 단점을 보완하도록 고안되었기 때문에 EDF보다 더 많은 태스크 집합을 스케줄 할 수 있다. 하지만 EDF-US와 fpEDF는 전체 이용률이 스케줄 보장 이용률보다 높고, 짧은 상대적 마감시간과 낮은 이용률을 가진 태스크가 많이 포함된 태스크

표 1 스케줄 성공한 태스크 집합의 백분율

처리기 수	이용률	EDF	EDF/ZL	EDF-US	EDF-US/ZL	fpEDF	fpEDF/ZL
2	(1, 2]	83.2%	93.7%	87.9%	94.2%	88.5%	93.0%
3	(1, 3]	80.4%	93.8%	86.8%	93.7%	87.6%	93.6%
4	(1, 4]	80.4%	94.7%	87.6%	94.1%	88.1%	94.5%
5	(1, 5]	79.8%	94.9%	88.0%	94.2%	88.4%	94.9%
6	(1, 6]	79.8%	95.5%	88.6%	94.7%	88.7%	95.5%
7	(1, 7]	79.8%	95.8%	89.0%	95.0%	89.0%	95.8%

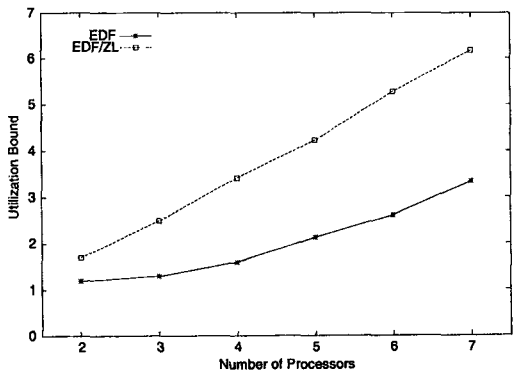


그림 6 EDF와 EDF/ZL의 스케줄 보장 이용률

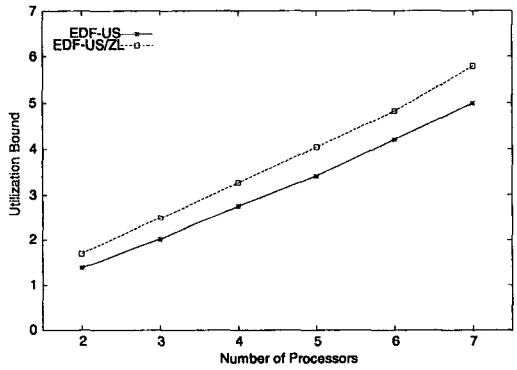


그림 7 EDF-US와 EDF-US/ZL의 스케줄 보장 이용률

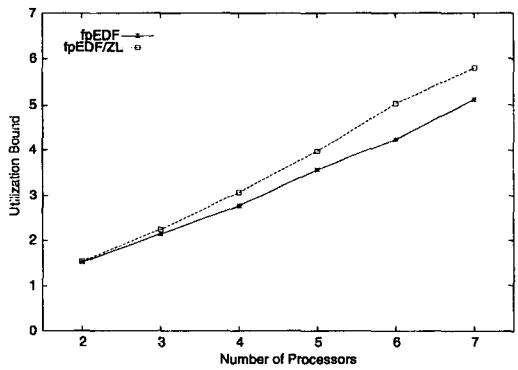


그림 8 fpEDF와 fpEDF/ZL의 스케줄 보장 이용률

집합을 스케줄 실패하기 쉽다. 최고 우선순위를 부여받은 이용률이 높은 태스크들은 여유시간이 충분해도 수행을 완료할 때까지 처리기를 독점적으로 사용한다. 수행 기회를 얻지 못한, 이용률이 낮은 태스크들은 여유시간이 빠르게 감소하고 결국 마감시간을 지키지 못한다. ZL기법은 여유시간이 0인 이용률이 낮은 태스크에게 우선적으로 수행 기회를 준다. 여유시간이 0이 된 작업은 즉시 이용률이 높은 태스크를 선점할 수 있으므로 마감시간을 지킬 수 있다.

그림 6은 EDF와 EDF/ZL의 처리기 수에 따른 스케줄 보장 이용률을 보여준다. 그림에서 보듯이 ZL기법은 EDF의 스케줄 보장 이용률을 현저하게 향상시켰다. 처리기의 수가 7일 때 EDF의 스케줄 보장 이용률은 3.34이지만 EDF/ZL의 스케줄 보장 이용률은 6.18이다.

그림 7과 그림 8은 ZL기법에 의해 향상된 EDF-US와 fpEDF의 스케줄 보장 이용률을 보여준다. EDF-US와 fpEDF은 본래 스케줄 보장 이용률이 높기 때문에 향상된 정도가 EDF만큼 크지 않다. 실험결과에서 EDF-US와 fpEDF의 스케줄 보장 이용률은 이론적인 스케줄 보장 이용률보다 다소 높다. 그 이유는 각 알고리즘의 최악의 경우가 본 실험에 사용한 태스크 집합에 포함되지 않을 수 있기 때문이다.

5. 결 론

본 논문은 고정 우선순위 스케줄링 알고리즘의 성능을 향상시키는 ZL기법을 제안하였다. ZL기법은 여유시간이 0인 작업에게 최고 우선순위를 부여하고 나머지 작업들에게는 고정 우선순위 알고리즘에 따라 우선순위를 부여한다. ZL기법을 적용한 고정 우선순위 알고리즘은 원래의 알고리즘이 스케줄 할 수 있는 모든 태스크 집합을 스케줄 할 수 있으며, 원래의 알고리즘이 스케줄 할 수 없는 일부 태스크 집합을 스케줄 할 수 있다.

모의실험을 통한 성능을 평가 결과, ZL기법은 스케줄 가능한 태스크 집합의 수와 스케줄 보장 이용률의 측면에서 고정 우선순위 알고리즘의 성능을 상당히 향상시킨다. 특히, ZL기법을 EDF에 적용할 때 현저한 성능 향상을 보였다.

또한, 본 논문은 ZL기법의 수행 오버헤드가 크지 않음을 보이고, ZL기법을 효율적으로 구현할 수 있는 구현 방안을 제시하였다. 향후 연구 과제로서 제시한 ZL기법 구현 방안을 다중처리기 실시간 시스템에 구현하여 ZL기법의 수행 오버헤드 분석이 필요하다.

참 고 문 헌

[1] Liu, C. L. and Layland, J. W., "Scheduling Algorithms for Multiprogramming in a HardReal-Time Environment," Journal of the ACM, Vol.20, No.1, pp. 46-61, 1973.

[2] Audsley, N. C., Burns, A., Richardson, M., and Wellings, A., "Hard Real-Time Scheduling: The Deadline Monotonic Approach," In Proceedings of IEEE Workshop on Real-Time Operating Systems and Software, pp. 133-137, 1991.

[3] Srinivasan, A. and Baruah, S., "Deadline-based Sc-

heduling of Periodic Task Systems on Multiprocessors," Information Processing Letters, Vol.84, No.2, pp. 93-98, 2002.
[4] Baruah, S. K.. "Optimal Utilization Bounds for the Fixed-Priority Scheduling of Periodic Task Systems on Identical Multiprocessors," IEEE Transactions on Computers, Vol.53, No.6, pp.781-784, 2004
[5] Leung, J., "A New Algorithm for Scheduling Periodic Real-Time Tasks," Algorithmica, Vol.4, pp. 209-219, 1989.
[6] Goossens, J., Funk, S., and Baruah, S., "Priority-Driven Scheduling of Periodic Task Systems on Multiprocessors," Real-Time Systems, Vol.25, Issue.2-3, pp. 187-205, 2003.
[7] Carpenter, J., Funk, S., Holman, P., Srinivasan, A., Anderson, J., and Baruah, S., Handbook of Scheduling: Algorithms, Models, and Performance Analysis, chapter 2, Chapman Hall/CRC Press, 2004.
[8] Dhall, S. K. and Liu, C. L., "On a Real-Time Scheduling Problem," Operations Research. Vol.26, No.1, pp. 127-140, 1978.
[9] Kalyanasundaram, B., Pruhs, K. P., and Torng, E., "Errata: A New Algorithm for Scheduling Periodic, Real-Time Tasks," Algorithmica, Vol.28, pp. 269-270, 2000.
[10] Goossens, J., and Macq, C., "Limitation of the Hyper-Period in Real-Time Periodic Task Set Generation," In Proceedings of the 9th International Conference on Real-Time Systems, pp. 133-148, 2001.

박 민 규
1991년 서울대학교 컴퓨터공학과 졸업(공학사). 1993년 서울대학교 컴퓨터공학과 졸업(공학석사). 1993년~현재 서울대학교 컴퓨터공학과 박사과정. 관심분야는 스케줄링 알고리즘, 실시간시스템, 분산시스템

한 상 철
1998년 연세대학교 컴퓨터과학과 졸업(공학사). 2000년 서울대학교 컴퓨터공학과 졸업(공학석사) 2000년~현재 서울대학교 컴퓨터공학과 박사과정. 관심분야는 실시간 시스템, 스케줄링 알고리즘

김 희 헌
1999년 고려대학교 컴퓨터과학과 졸업 2000년~현재 서울대학교 전기·컴퓨터공학부 석박사 통합과정. 관심분야는 다중처리기상에서의 실시간 스케줄링 분산시스템

조 성 제
1989년 서울대학교 컴퓨터공학과(학사). 1991년 서울대학교 대학원 컴퓨터공학과(공학석사). 1996년 서울대학교 대학원 컴퓨터공학과(공학박사). 1996년~1997년 서울대학교 컴퓨터신기술연구소 객원연구원. 2001년 미국 University of California, Irvine 객원 연구원. 1997년~현재 단국대학교 정보컴퓨터학부 컴퓨터과학전공 조교수. 관심분야는 컴퓨터보안, 시스템 소프트웨어, 실시간 시스템, 분산 시스템 등

조 유 근
1971년 서울대학교 건축공학과 학사 1978년 미네소타대학교 컴퓨터과학 박사 1979년~현재 서울대학교 컴퓨터공학부 교수. 1984년~1985년 미네소타대학교 교환 교수. 1993년~1995년 서울대학교 중앙교육연구전산원장. 1999년~2001년 서울대학교 공과대학 부학장. 2001년~2002년 한국정보과학회 회장. 관심분야는 운영체제, 알고리즘 설계 및 분석, 암호학