DOI: 10.3745/KIPSTA.2009.16A.6.437

다중프로세서 시스템을 위한 여유시간 기반의 온라인 실시간 스케줄링 알고리즘

조 규 억[†] · 김 용 석[††]

요 약

마감시간을 기준으로 스케줄링하는 EDF (Earliest Deadline Fisrt) 알고리즘이나 여유시간을 기준으로 스케줄링 하는 LLF (Least Laxity First) 알고리즘과 같은 기존의 방식들은 다중프로세서 시스템에서는 스케줄링 성공률이 급격히 낮아지거나 문맥교환 횟수가 지나치게 높아 현실적인 적용에 무리가 있다. 이 둘을 적절히 조합하여 성능을 개선한 것으로서 EDZL (Earliest Deadline Zero Laxity)은 EDF를 기본으로 하고 여유시간이 0에 도달한 태스크에 대해서는 우선적으로 실행하도록 하는 방식이다. 본 논문에서는 LLF와 같이 최소 여유시간의 태스크를 우선적으로 실행하되 문맥교환은 여유시간이 0에 도달한 태스크가 발생할 경우에만 이루어지도록 한 LLZL (Least Laxity Zero Laxity) 알고리즘을 제안한다. 시뮬레이션 평가결과 LLZL은 준최적으로 알려져 있는 LLF에 근접한 높은 스케줄링 성공률을 보이면서도 문맥교환 오버헤드는 EDF와 비슷한 낮은 수준을 유지하였다. EDZL과의 비교에 있어서도 스케줄링 성공률, 문맥교환 횟수 두 가지 측면 모두 나은 성능을 보인다.

키워드 : 온라인 스케줄링, 비주기적 태스크, 여유시간, 실시간 태스크, 다중프로세서 시스템

A Laxity Based On-line Real-Time Scheduling Algorithm for Multiprocessor Systems

Kyueok Cho[†] · Yong-Seok Kim[††]

ABSTRACT

For multiprocessor systems, Earliest Deadline First (EDF) based on deadline and Least Laxity First (LLF) based on laxity are not suitable for practical environment since EDF has low schedulability and LLF has high context switching overhead. As a combining of EDF and LLF to improve the performance, Earliest Deadline Zero Laxity (EDZL) was proposed. EDZL is basically the same as EDF. But if the laxity of a task becomes zero, its priority is promoted to the highest level. In this paper, we propose Least Laxity Zero Laxity (LLZL) which is based on LLF. But context switching is allowed only if the laxity of a task on rady queue becomes zero. Simulation results show that LLZL has high schedulability approaching to LLF and low context switching overhead similar to EDF. In comparison with EDZL, LLZL has better performance in both of schedulability and context switching overhead.

Keywords : On-Line Scheduling, Aperiodic Task, Laxity, Realtime Task, Multiprocessor System

1. 서 론

실시간 태스크의 작업들이 점차 복잡해짐에 따라 실시간 시스템에서도 다중프로세서를 사용하는 것이 일반화 되고 있는 추세이다. 이에 따라 다중프로세서 환경에서의 실시간 스케줄링 알고리즘에 관한 연구도 활발히 진행 중에 있다. 다중프로세서 환경에서는 단일프로세서 환경에서와 달리 태스크를 어느 프로세서에서 실행하는가에 대한 정책에 따라서도 스케줄링 성공률이 영향을 받게 된다. 이러한 복잡성으로 인하여 다중프로세서 환경에서의 실시간 스케줄링에 관한 연구는 시작단계이고, 상대적으로 예측이 가능한 주기적인 태스크 모델에 연구가 집중되어 있다. 따라서 다중프로세서 환경에서의 비주기적 태스크 모델에 적용할 스케줄링 알고리즘들은 많이 나와 있지 않다. 주기적인 태스크 모델에서는 태스크의 발생을 예측할 수 있기 때문에 현재의 상황뿐만이 아니라 미래의 상황을 고려하여 스케줄링 함으로서 전체적인 스케줄링 성공률을 높일 수 있다. 그러나 비주기적 태스크 모델에서는 미래의 태스크 발생을 예측할 수 없기 때문에 현재 상황만을 고려하여 스케

※ 본 논문은 부분적으로 강원대학교 정보통신연구소의 지원을 받았음.
† 준 회 원 : 강원대학교 컴퓨터정보통신공학과 석사과정
†† 정 회 원 : 강원대학교 컴퓨터학부 교수(교신저자)
논문접수 : 2009년 4월 14일
수 정 일 : 1차 2009년 8월 17일
심사완료 : 2009년 8월 19일

줄링 하게 된다. 따라서 시스템이 운용되는 전체 시간 안에 발생하는 모든 태스크에 대해 마감시간을 충족시키기는 어렵게 되므로 최적의 스케줄링 알고리즘은 존재할 수 없다[7].

단일프로세서 환경의 경우 EDF(Earliest Deadline First)와 LLF(Least Laxity First)가 최적으로 알려져 있다. 그러나 다중프로세서 환경에서는 EDF의 경우 문맥교환 비율은 낮지만 스케줄링 성공률이 낮으며 LLF는 스케줄링 성공률은 높지만 문맥교환 횟수가 많아 현실적인 적용에는 무리가 있다. 따라서 이러한 스케줄링 알고리즘들을 적절히 조합하여 성능을 개선한 알고리즘들이 활발히 연구되고 있다. 그 중 대표적인 것으로 EDZL(Earliest Deadline Zero Laxity)이 있다. EDZL은 많은 연구를 통해 EDF와 LLF보다 좀 더 현실적 적용이 가능한 것으로 알려져 있다. 본 논문에서는 다중프로세서 환경에서도 준최적으로 알려져 있는 LLF의 단점을 보완하여 LLF에 근접한 스케줄링 성공률을 보이면서 EDF와 비슷한 문맥교환 비율을 갖는 LLZL(Least Laxity Zero Laxity) 알고리즘을 제안하고 그에 대한 성능을 평가하였다. EDZL과의 비교에 있어서도 LLZL이 좀 더 나은 성능을 보인다.

2. 관련 연구

단일프로세서 환경에서와 달리 다중프로세서 환경에서는 태스크를 어느 프로세서에서 실행시킬 것인가, 할당된 프로세서에서 다른 프로세서로의 이동을 어떻게 할 것인가와 같은 추가적인 고려사항이 발생한다. 이러한 것들로 인하여 단일프로세서 상에서 최적으로 알려져 있는 스케줄링 알고리즘들도 다중프로세서 환경에서는 오히려 좋지 않은 성능을 보여주는 경우가 있다. Baruah등은 위와 같은 사항을 고려하여 다중프로세서 환경에서의 스케줄링 알고리즘을 분류하고 그들 간의 관계를 보였다[1].

다중프로세서 환경에서는 스케줄링 성공률에 있어서 마감시간을 기준으로 한 알고리즘보다 여유시간을 기준으로 한 알고리즘이 일반적으로 더 유리하다[2]. 마감시간을 기준으로 하게 되면 마감시간은 크지만 상대적으로 계산시간이 길어 여유시간이 짧은 태스크에 대해서는 효과적으로 스케줄링 할 수 없게 된다. 그러나 여유시간이 작다는 것은 상대적으로 급한 태스크를 의미하며 이러한 태스크를 우선적으로 스케줄링 하게 되면 효과적으로 스케줄링 할 수 있게 된다.

EDF는 단일 프로세서 환경에서 최적으로 알려진 스케줄링 알고리즘이다[3, 4]. 이것은 태스크의 마감시간을 기준으로 마감시간이 빠른 것을 우선적으로 실행하는 방식을 취하고 있다. 마감시간에 따라 결정된 우선순위가 변하지 않으므로 문맥교환 횟수가 높지 않은 장점이 있다. 또한 상대적으로 구현의 용이함도 EDF의 장점이라 할 수 있다. 그러나 마감시간만을 고려하여 우선순위를 결정하기 때문에 다중프로세서 환경에서는 여유시간이 작은 태스크와 긴 태스크가 혼재해 있을 경우 적절히 스케줄링 할 수 없게 되고 따라서 스케줄링 성공률도 낮아지게 된다[5].

LLF 또한 단일프로세서 환경에서는 최적으로 알려진 스케줄링 알고리즘이다[6]. 이것은 태스크들의 여유시간을 기준으로 여유시간이 가장 짧은 태스크를 우선적으로 실행하는 방식을 취하고 있다. 현재 시점에서 급한 태스크들에게 우선순위를 높게 주게 됨으로서 다중프로세서 환경에서도 스케줄링 성공률이 상당히 높다. 그러나 비주기적 태스크 모델의 예측할 수 없는 특성에 따라 단일프로세서 환경에서와 같이 최적의 성능을 보이진 않지만, 준최적성을 만족하게 된다. 준최적성이란 현재 대기중인 태스크가 스케줄링 가능한 태스크 집합일 경우 항상 적합한 스케줄을 생성하는 알고리즘을 준최적성을 만족한다고 정의한다[7]. LLF의 준최적성이라는 장점에도 여유시간은 마감시간과는 달리 시간의 흐름에 따라 변화하는 값이므로 각 태스크들의 상대적인 여유시간의 차이는 빈번하게 바뀌게 된다. 따라서 LLF는 문맥교환 횟수가 지나치게 높아지게 되고 현실적인 적용에는 무리가 있다[1].

EDZL은 EDF방식에 여유시간이 0이 되는 태스크를 추가적으로 고려해서 성공률을 높인 알고리즘이다[7]. 대기큐에 여유시간이 0인 태스크가 존재하지 않으면 EDF와 마찬가지로 마감시간에 따라 우선순위를 부여하여 스케줄링한다. 여유시간이 0인 태스크가 발생하게 되면 긴급히 실행하지 않을 경우 마감시간을 충족하지 못하기 때문에 현재 실행중인 태스크들 중 여유시간이 0보다 크면서 마감시간이 가장 늦는 태스크와 문맥교환을 하여 실행할 수 있게 한다. 또한 EDZL은 EDF가 스케줄링 할 수 있는 태스크 집합, 즉 여유시간이 0인 태스크가 발생하지 않는 태스크 집합에 대해서는 EDF와 동일하게 스케줄링하게 된다. 여유시간이 0인 태스크가 발생하는 경우에 다른 태스크들보다 높은 우선순위를 할당하여 긴급히 실행하도록 하여 마감시간을 충복시킬 수 있기 때문에 EDF에 비해 스케줄링 성공률이 높다. EDZL이 두 개의 프로세서 환경에서는 준최적이고 스케줄링 성공률과 문맥교환 횟수면에서 여유시간 값이 비교적 충분한 태스크들에 대해서 현실적 사용가능성이 입증된바 있다[7-8].

3. LLZL 알고리즘

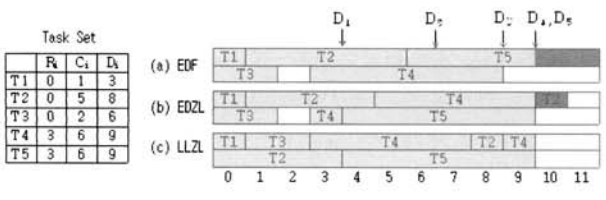
본 논문에서는 m개의 프로세서 상에서 비주기적 태스크 집합을 대상으로 하는 경성 실시간 스케줄링을 다룬다. 태스크 집합은 n개의 선점 가능한 비주기적 태스크로 구성되며 태스크 T_i는 (R_i, C_i, D_i)로 표현된다. 각각 도착시간, 계산시간, 마감시간을 의미한다. 여유시간 L_i는 $D_i-(T+C'_i)$로 정의하고, 여기서 T는 현재 시간이고 C'_i는 남아있는 계산시간이다. 여유시간은 태스크의 긴급한 정도를 의미한다. 여유시간이 음수이면 그 태스크는 마감시간을 충족할 수 없음을 의미하고 여유시간이 0이면 즉시 그 작업을 시작하지 않을 경우 마감시간을 충족할 수 없게 되므로 즉시 시

작하여야 한다.

LLZL은 LLF에 기반하면서 문맥교환 횟수를 줄이도록 한 알고리즘이다. 앞에서 소개한 것과 같이 LLF는 준최적임에도 문맥교환이 빈번하게 일어나기 때문에 현실적으로 적용하기에는 무리가 있다. LLZL에서는 이러한 문제를 해결하기 위해 문맥교환은 여유시간이 0인 태스크가 발생하는 시점에만 하도록 하였다. 만일 여유시간이 0인 태스크가 발생하지 않는다면 문맥교환은 일어나지 않으며, 실행중인 태스크가 종료되는 시점에 대기 큐에서 여유시간이 가장 작은 태스크를 실행하도록 한다. 여유시간이 0인 태스크가 발생하면 현재 실행중인 태스크들 중 여유시간이 가장 큰 태스크와 문맥교환을 수행한다. 여유시간이 다양하게 분포되어 있는 태스크 집합의 경우 마감시간을 우선적으로 고려하게 되는 EDZL에 비해 LLZL은 태스크의 계산시간이 함께 고려되는 여유시간으로 스케줄링 하기 때문에 스케줄링 성공률이 더욱 높아질 수 있다. 예를 들어 태스크 집합 {(0, 1, 3), (0, 5, 8), (0, 2, 6), (3, 6, 9), (3, 6, 9)} 이 두 개의 프로세서에서 실행이 된다면 (그림 1)-(a)와 (그림 1)-(b)에서 보여 지는 것과 같이 EDF와 EDZL의 경우 마감시간을 충족하지 못하는 태스크가 발생하게 되지만 (그림 1)-(c)와 같이 LLZL의 경우 모든 태스크에 대해서 마감시한을 충족시킬 수 있게 된다. 반대로 EDF나 EDZL로 스케줄링이 가능한 집합에 대하여 LLZL로는 실패하는 특수한 경우도 존재하지만 평균적으로는 LLZL 의 성공률이 높게 나온다.

LLZL의 스케줄링 시점은 크게 세 가지 경우로 구분하여 나타낼 수 있다. 먼저 새로운 태스크가 발생했을 경우 만일 태스크가 할당되어 있지 않은 프로세서가 있을 경우 발생한 태스크를 그 프로세서에 할당하고 실행되게 한다. 이미 모든 프로세서에서 태스크가 실행중이라면 새로 발생한 태스크는 대기 큐에 삽입된다. 두 번째로 실행중인 태스크가 종료되면 LLZL은 여유시간을 기준으로 정렬되어 있는 대기 큐에서 가장 짧은 여유시간을 가진 태스크를 프로세서에 할당한 후 실행되게 한다. 마지막으로 여유시간이 0인 태스크가 발생하면 즉시 실행하지 않으면 마감시간을 충족할 수 없으므로 현재 프로세서에서 실행중인 태스크들 중 여유시간이 가장 긴 태스크를 찾아내어 이를 선점하도록 하여 여유시간이 0인 태스크가 마감시간을 충족할 수 있게 한다. (그림 2)는 LLZL에 대한 알고리즘을 보여준다.

LLF의 경우 매 시점에 따라 변화하는 여유시간을 기준으로 스케줄링하게 되므로 태스크들의 우선순위 관리가 복



	R_i	C_i	D_i
T1	0	1	3
T2	0	5	8
T3	0	2	6
T4	3	6	9
T5	3	6	9

(그림 1) EDF, EDZL과 LLZL의 비교 예

```
if ( a new task T is arrived )
  if ( idle processor exists )
    assign T to the idle processor
  else
    insert T into the ready queue

if ( the task on processor P is finished )
  select the least laxity task T from the ready queue
  assign T to processor P

if ( a task T on ready queue becomes zero laxity )
  T preempts the most laxity task
      among running tasks
```

(그림 2) LLZL 알고리즘

잡해진다. 그러나 LLZL은 LLF와는 달리 (그림 2)에서와 같이 3가지 상황에서만 문맥교환이 발생하므로 좀 더 간단하게 여유시간을 관리할 수 있다. 시스템내의 모든 태스크는 동일한 시스템시간을 갖게 되므로 태스크의 여유시간 L_i를 D_i-C'i로 태스크가 최소한 이 이전에 실행을 시작해야 하는 시점으로 정의함으로서 시간에 따라 여유시간을 갱신하지 않더라도 태스크가 스케줄링 되는 시점에 해당 태스크에 대해서만 여유시간을 갱신해주게 됨으로서 효율적인 관리가 가능하게 된다. 이와 같이 LLZL은 LLF를 좀 더 단순하게 구현하게 되지만 다음의 성능 평가에서 보는 바와 같이 LLF가 가진 스케줄링 성공률의 장점을 크게 감소시키지 않으면서 문맥교환 횟수를 큰 폭으로 줄일 수 있게 된다.

4. 성능 평가

4.1 시뮬레이션 모델

성능평가를 위해 시뮬레이션을 통하여 LLZL을 기존의 EDF, LLF, 및 EDZL과 비교하였다. 시뮬레이션은 태스크의 평균 발생간격의 변동, 시스템의 평균 부하의 변동, 및 태스크의 평균 여유시간의 변동에 따라 실시하였다. 태스크가 발생하는 간격 (λ)은 지수분포를 따르고, 태스크의 평균 발생빈도는 $f = 1/\lambda$ 이다. 시스템의 평균 부하 ρ에 대하여 태스크의 평균 실행시간은 $C = \rho/f$ 이다. 지정된 시스템 평균 부하에 대하여 태스크의 발생간격이 줄어들면 태스크의 평균 실행시간은 늘어난다. 태스크별의 실행시간은 1부터 2C까지의 구간에 균등분포를 따른다. 태스크의 여유시간은 태스크의 실행시간에 여유시간 비율을 곱해서 결정하며, 평균 비율 r에 대하여 태스크별 여유시간 비율은 0부터 2r까지의 구간에 균등분포를 따른다.

성능 평가는 스케줄링 성공률과 문맥교환율로 비교하였다. 스케줄링 성공률은 스케줄링을 실시한 태스크 집합 개수대비 성공적으로 스케줄링이 이루어진 태스크집합 개수의 비율을 의미하고, 문맥교환율은 태스크 개수 대비 스케

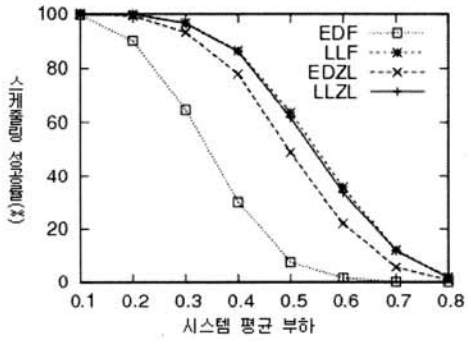
줄링 동안 발생한 문맥교환 회수의 비율이다.

4.2 시스템의 부하에 따른 평가

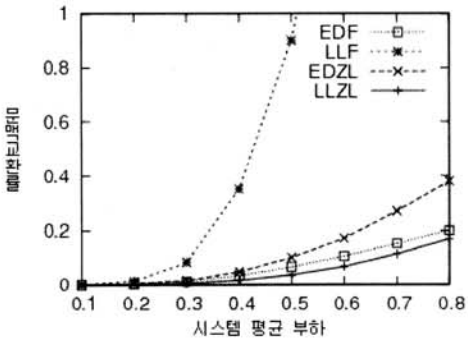
시스템 부하의 변화에 따른 평가 결과 LLZL은 스케줄링 성공률 측면에서 준최적인 LLF에 상당히 근접해 있고 EDZL보다 우수한 결과를 보여주었다. 문맥교환율 측면에서도 문맥교환 회수가 적은 것으로 알려진 EDF과 거의 흡사한 결과를 보여주었다. 시뮬레이션에 적용한 파라미터로는 프로세서의 개수는 5개, 단위시간당 태스크의 평균 발생빈도는 0.04, 그리고 태스크의 평균 여유시간 비율은 r=0.5을 적용하였다. 각 시행에 있어 모든 스케줄링 알고리즘에는 동일한 태스크 집합을 적용하였다.

(그림 3)은 시스템 부하의 변화에 따른 스케줄링 성공률을 보여준다. LLZL은 준최적으로 알려진 LLF와의 비교에 있어서 큰 차이를 보이지 않았다. LLZL은 기본적으로 여유시간에 우선순위를 두고 스케줄링하게 되고 여유시간이 0인 태스크에 대해서만 즉시 실행하도록 함으로서 여유시간을 지속적으로 관리하는 LLF보다는 낮지만 상당히 근접한 스케줄링 성공률을 가지게 된다. 또한 EDZL과의 비교에 있어서 LLZL의 스케줄링 성공률이 약 10%정도 높게 나타나고 있다. 이는 마감시간을 기준으로한 방법보다 여유시간을 기준으로한 방법이 스케줄링 성공률 측면에서 좀 더 좋은 성능을 보이기 때문이다.

시스템 부하의 변화에 따른 문맥교환율 측면에서도 (그림 4)에서 보는바와 같이 LLZL은 문맥교환 회수가 적은 것으로 알려진 EDF과 거의 흡사한 결과를 보여주었다. LLF의 경우 시간에 따라 변화하는 여유시간에 따라 선점이 이루어지기 때문에 상대적으로 태스크의 계산시간이 긴 상태에서는 문맥교환율이 지나치게 높고 따라서 실제 시스템에 적용하기가 곤란하다. EDF 및 EDZL의 경우 LLF와 달리 실시간으로 변하는 값이 아닌 고정되어 있는 마감시간을 기준으로 선점이 이루어지기 때문에 태스크의 성격에 따른 변화가 크게 나타나지 않는다. EDZL은 기본적으로 EDF방식을 취하고 여유시간이 0인 태스크가 발생하는 경우 추가적인 문맥교환이 이루어지게 되므로 EDF보다 높은 문맥교환율이 발생하게 된다. 반면에 LLZL의 경우 여유시간에 따라 정렬된 대기 큐를 이용하고 여유시간이 0인 태스크가 발생할 때에만 문맥교환이 이루어지게 되므로 EDF, EDZL과 마찬가지로 태스크의 성격에 따른 변화가 크지 않다.



(그림 3) 시스템 평균 부하에 따른 스케줄링 성공률



(그림 4) 시스템 평균 부하에 따른 문맥교환율

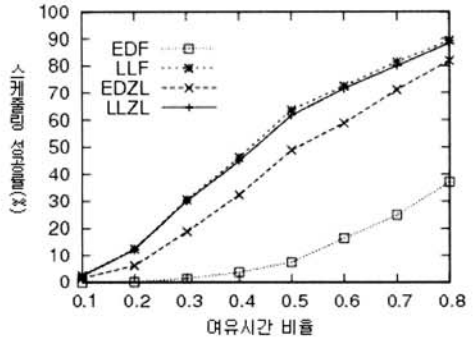
LLZL은 EDF와 비교해서도 근소한 차이로 낮은 문맥교환율을 보임에 주목할 필요가 있다. EDF의 경우 태스크의 긴급한 정도를 나타내는 여유시간을 고려하지 않고 마감시간만을 고려하기 때문에 불필요한 문맥교환을 하게 되는 경우가 발생할 수 있다. 그러나 LLZL의 경우 새로 발생한 태스크가 비록 마감시한은 짧지만 비교적 큰 여유시간이 있다면 현재 실행하던 것을 완료한 후에 실행할 수 있으므로 불필요한 문맥교환을 줄일 수 있게 된다. 그러나 이후에 보여줄 (그림 6)에서와 같이 평균 여유시간 비율이 0.3이하로 줄어들면 EDF가 조금 더 우수한 결과를 보여주게 된다.

EDZL과 LLZL은 여유시간이 0인 태스크에 가장 높은 우선순위를 부여하여 즉시 처리하도록 한다는 측면에서 비슷하다. LLZL은 여유시간이 0인 태스크가 발생할 경우에만 문맥교환이 이루어지므로 여유시간이 0인 태스크가 발생하는 비율은 (그림 4)에서 LLZL의 문맥교환률과 동일하다. EDZL에서 여유시간이 0인 태스크가 발생하는 비율은 LLZL에 비해 약간 높게 나타난다. 이것은 (그림 4)에서 EDZL의 문맥교환률과 LLZL의 문맥교환률 사이의 값으로서 나타난다. EDZL의 문맥교환률에서 새로 생성된 태스크의 마감시간이 실행중인 태스크보다 일러서 발생하는 문맥교환을 제외한 것이다.

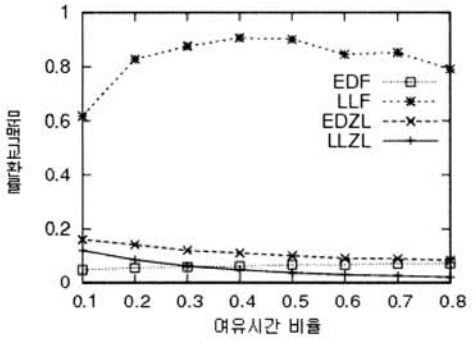
4.3 여유시간 및 태스크 발생빈도에 따른 평가

평균 여유시간 비율과 태스크의 평균 발생빈도를 변화시키더라도 시스템 부하를 변화시키는 경우와 동일하게 LLZL의 스케줄링 성공률은 LLF에 근접하고 문맥교환률은 EDF에 근접한 결과를 보여주었다. 시뮬레이션에 적용한 파라미터로는 프로세서의 개수는 5개, 시스템의 평균 부하는 평균 0.5를 적용하였다. 태스크 발생 빈도를 변화시키는 경우에는 여유시간 비율 0.5를 적용하고 여유시간을 변화시키는 경우에는 태스크 발생빈도 0.04를 적용하였다.

(그림 5)와 (그림 6)은 여유시간에 따른 스케줄링 성공률과 문맥교환율을 보여준다. 여유시간이 상대적으로 긴



(그림 5) 여유시간에 따른 스케줄링 성공률



(그림 6) 여유시간에 따른 문맥교환율

태스크 모델에 대해서는 (그림 3)에서와 마찬가지로 LLZL이 LLF와 근접한 스케줄링 성공률을 보이면서 EDF보다 낮은 문맥교환율을 보여준다. 그러나 여유시간이 줄어들게 되면 여유시간이 0인 태스크가 발생하는 비율이 커지게 됨으로써 EDF보다 문맥교환율이 커지게 됨을 보여준다. 그러나 EDZL 또한 이러한 현상이 발생하게 되므로 여유시간의 변화에 따른 실험에서도 LLZL이 여전히 EDZL보다 나은 성능을 보이는 것으로 나타났다. 태스크의 평균 발생빈도를 변화시켰을 경우에도 마찬가지의 결과를 보여주었다.

5. 결 론

비주기적 태스크 모델에서는 비주기적 태스크가 갖는 예측할 수 없는 특성으로 인하여 최적의 온라인 스케줄링 알고리즘이 존재할 수 없다. 또한 기존의 단일프로세서에서 적용하던 스케줄링 알고리즘들은 다중프로세서 환경에서 비주기적 태스크들을 적절히 스케줄링 하지 못한다. LLF와 EDF는 스케줄링 성공률과 문맥교환 비율 사이에서 한가지 측면에서만 만족할만한 성능을 보이고 다른 측면에서는 낮은 성능을 갖는다.

본 논문에서 제시한 LLZL은 여유시간을 기본으로 하여 긴급한 태스크가 발생할 때에만 문맥교환이 이루어 지도록 하는 알고리즘으로서, 시뮬레이션을 통한 성능 평가 결과 LLF에 근접한 스케줄링 성공률을 보이면서도 EDF과 유사한 문맥교환율을 보이는 것으로 나타났다. 또한 EDF의 대안으로서 널리 알려진 EDZL보다도 우수한 성능을 보여주었다.

LLZL은 긴급한 태스크가 발생하지 않으면 여유시간에 따라서 문맥교환 없이 순서대로 스케줄링한다. 이때 여유시간이 유사한 태스크가 2개 이상일 경우 어떤 태스크를 선택하여 스케줄링 하는가에 따라 스케줄링 성공률 또한 달라질 것이다. 따라서 여유시간이 유사한 태스크가 존재할 경우의 태스크 선택 정책과 과부하에서의 처리방향에 대한 연구가 필요하다.

참 고 문 헌

[1] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, and S. Baruah. A categorization of real-time multiprocessor scheduling problems and algorithms. In Joseph Y. Leung, editor, Handbook on Scheduling Algorithms, Methods, and Models, pages 30.1-30.19. Chapman Hall/CRC, Boca Raton, Florida, 2004.

[2] Hong, J.; Tan, X.; Towsley, D., "A performance analysis of minimum laxity and earliest deadline scheduling in a real-time system," Computers, IEEE Transactions on, Vol.38, No.12, pp.1736-1744, Dec., 1989.

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

[4] Liu, J. W., Real-Time Systems, p.70, Prentice Hall, 2000

[5] Locke, D., "Best-Effort Decision Making for Real-Time Scheduling," Doctoral Dissertation, Computer Science Department, Carnegie-Mellon University, 1986.

[6] Leung, J., "A New Algorithm for Scheduling Periodic Real-Time Tasks," Algorithmica, Vol.4, pp.209-219, 1989.

[7] Cho, S., Lee, S., Ahn, S., and Lin, K., "Efficient Real-Time Scheduling Algorithms for Multiprocessor Systems," IEICE Trans, Commun., Vol. E85-B, No.12, pp.2859-2867, 2002

[8] 박민규, 한상철, 김희헌, 조성제, 조유근, "다중처리기 상의 실시간 스케줄링 알고리즘의 우월 관계 및 성능", 정보과학회논문지: 시스템 및 이론 제 32 권 제 7호, 2005.8.

조 규 억

e-mail : ckebabo@gmail.com

2008년 강원대학교 컴퓨터정보통신공학부 (학사)

2008년~현 재 강원대학교 컴퓨터정보통신공학부 석사과정

관심분야 : 운영체제, 실시간시스템

김 용 석

e-mail : yskim@kangwon.ac.kr

1984년 서울대학교 해양학과(학사)

1986년 KAIST 전기및전자공학과(공학석사)

1989년 KAIST 전기및전자공학과(공학박사)

1990년~1995년 한국생산기술연구원 및 전자부품연구원 선임연구원

1995년~현 재 강원대학교 컴퓨터학부 교수

관심분야 : 운영체제, 임베디드시스템, 실시간시스템