비주기 태스크의 응답시간을 개선하기 위해 확장한 슬랙 스틸링 알고리즘

최 만 억† · 한 대 만†† · 구 용 완†††

요 약

본 논문은 고정 우선순위를 가지는 주기 태스크와 동적으로 발생하는 비주기 태스크를 스케줄링하는 슬랙 스틸링(slack stealing) 알고리즘의 문제점을 개선한다. 슬랙 스틸링 알고리즘은 비주기 태스크의 발생에 따라 슬랙 스틸링 서버가 적합한 우선순위를 비주기 태스크에 부여하여 즉시 서비스 가능하도록 함으로써 불필요한 대기시간을 최소화하고 있다. 하지만, 슬랙 스틸링을 수행하기 위해서는 임의의 시점까지 주기적 태스크의 수행 시간을 구해야 한다. 그리고, 주기적 태스크의 수행 시간은 슬랙 알고리즘을 적용하는 동안 매 시간 마다 다시 구해지고 있다. 이때 사용되는 시간 복잡도는 계산에 적용되는 태스크의 수가 n이라면 $O(n)$으로 나타난다. 본 논문에서는 스케줄링된 주기적 태스크의 슬랙타임과 수행시간을 테이블에 저장하여 비주기 태스크가 사용하는 슬랙을 구함으로서 동적으로 발생하는 비주기적 태스크의 복잡도를 $O(\log n)$으로 감소시키고 응답시간을 향상시킨다. 본 논문에서 제안한 알고리즘을 모의 실험을 통하여 증명한다.

Extended Slack Stealing Algorithm for Improve Response Time of Aperiodic Tasks

Man-Uk Choi† · Dae-Man Han†† · Yong-Wan Koo†††

ABSTRACT

This paper intends to improve the problems of the slack stealing algorithm scheduling for periodic tasks with fixed priority and aperiodic tasks which occur dynamically. The Slack stealing algorithm reduces unnecessary *waiting time* by making the service possible immediately when the slack stealing server gives suitable priority to aperiodic tasks according to the status of aperiodic tasks arrivals at runtime But to performs the slack stealing, we must calculate execution time of periodic tasks till the point of random And, execution time of periodic tasks is being repeatedly calculated every hours while the slack algorithm is applied. We show time complexity when is used as to O(n) if the number of tasks which is applied to the calculation is n In this paper, due to stored in tables slack times and the execution times of the scheduled periodic tasks, the complexity of aperiodic tasks which is occurring dynamically reduced to $O(\log n)$ and improves the response times. we prove the algorithm proposed in this paper through the simulation.

1. 서 론

실시간 시스템은 범용 컴퓨터 시스템과는 달리, 시스템의 시간적 정확성에 유의해야 하는 시스템으로, 시간적 제약의 만족이 중요시되는 항공기 제어, 무기관리, 핵발전소등에서 넓게 사용되고 있다. 실시간 시스템의 연구분야는 성능을 좌우할 수 있는 예측성(Predictability), 스케줄링 가능성(Schedulability), 적합성(Feasibility)등의 세 가지 방향으로 연구가 진행되고 있으며 실시간 시

† 정 회 원 수원대학교 대학원 전자계산학과
†† 준 회 원 수원대학교 대학원 전자계산학과
††† 종신회원 수원대학교 전자계산학과 교수
논문접수 2000년 1월 18일, 심사완료 2000년 6월 20일

스템에 사용되는 대부분의 실시간 스케줄링 알고리즘이 비주기적 태스크의 응답시간 개선에 초점을 맞추어 연구가 진행되고 있다.

본 논문에서 중점적으로 다루는 슬랙 스틸링 알고리즘은 슬랙을 계산할 때 각 스케줄링 시점에서 모든 우선순위 레벨에 대하여 검색을 하므로 계산량이 방대하여 그것을 실제 시스템에 적용하기에는 무리가 따른다. 온라인(On-line)상에서 최대 슬랙 $A^*(t)$를 구하는 과정에서 주기적 태스크의 수행시간 계산으로부터 발생하는 오버헤드 때문에 실행시간에 비주기 태스크가 발생한 시점에서 가용슬랙을 구하는 방법이 비주기적 태스크의 응답시간 개선이라는 근본적인 목적을 고려할 수 없게 된다 따라서, 본 논문에서는 슬랙 스틸링으로부터 발생되는 근본적인 오버헤드를 감소하는 개선된 알고리즘을 제시한다.

본 논문의 구성은 다음과 같다 2장 관련 연구에서는 비주기적 태스크와 주기적 태스크의 혼합된 형태를 위한 스케줄링 알고리즘 중에서 슬랙 스틸링 알고리즘을 구체적으로 설명하고 문제점을 제시한다. 3장에서는 개선된 슬랙 스틸링 알고리즘을 제시한다. 4장에서는 시뮬레이션 방법과 배경을 설명하고, 시뮬레이션을 통한 실험 및 평가를 기술한다. 5장에서는 결론 및 향후 연구방향에 대하여 서술한다

2. 관련연구

주기 태스크와 비주기 태스크의 혼합된 형태의 슬랙 스틸링 알고리즘을 살펴보고 기존의 알고리즘으로부터 발생하는 문제점을 제기한다

2.1 슬랙 스틸링(slack stealing) 알고리즘

슬랙(slack)은 주기적인 태스크가 실행을 종료했을 때 종료시한까지 사용되지 않는 부분을 말한다. 슬랙 스틸링 알고리즘에는 정적 슬랙 스틸링 알고리즘과 동적 슬랙 스틸링 알고리즘이 있다. 두 알고리즘은 모든 종류의 비주기 태스크에 대해 최소 응답시간(response time)을 제공한다는 관점에서 최적이라고 증명되었다[9]. 정적 슬랙 스틸링 알고리즘은 연성 종료시한을 갖는 비주기 태스크와 경성 종료시한을 갖는 비주기 태스크에 대해 적용하여 사용되는 실시간 스케줄링 알고리즘이다. 이 알고리즘은 주기 태스크에 종료시한 단조형으로 우선순위를 할당하고 비주기 태스크를 위한 주기적인 서버를 두지 않고 비주기 태스크가 도착하면 도착 시간에 해당하는 오프라인 시 구해놓은 주기 태스크의 각 우선순위 레벨에서의 가능한 슬랙(slack)들의 최소값을 비주기 태스크의 실행을 위하여 활용한다.

슬랙 스틸링 알고리즘은 임격한 제한시간을 가지는 주기 태스크의 시간 제한을 준수하면서 비주기 태스크의 응답시간을 최소화하기 위하여 처리기 시간을 주기 태스크로부터 스틸링(stealing)하는 방식으로 메모리 시스템에서의 사이클 스틸링(cycle stealing)과 개념적으로 유사한 방법이다[11] 오프라인으로 태스크의 정보를 테이블로 구축하는 경우 테이블의 크기를 태스크의 하이퍼주기(hyperperiod)의 크기만큼 구성하게 된다. 초월주기는 모든 주기 태스크의 최소 공배수(LCM)가 되는 시간구간이다. 전체 태스크 집합에 대하여 태스크의 정보를 기록할 수 있는 테이블을 구성하기 위해서는 너무 많은 메모리가 낭비되기 때문에 초월주기를 테이블의 크기로 생성한다. 즉, 각 태스크의 $LCM(\tau_1, \tau_2, \cdots, \tau_n)$을 구하여 테이블로 구성한다. 주기 태스크 수행시 구성된 테이블은 비주기 태스크가 발생하게 되면 테이블을 참조하여 가용슬랙을 구하게 된다. 이때 테이블에 구성된 슬랙은 태스크가 제출된 시점부터 생성된 슬랙이므로 임의의 구간에서 사용할 수 있는 슬랙을 구하기 위하여 주기 태스크의 수행 크기를 요구하게 되면 테이블로부터 실제 사용 가능한 슬랙을 생성하여 비주기 태스크에 할당한다

다음은 슬랙 스틸링을 위한 가정과 슬랙을 생성하는 과정을 기술한다.

- 슬랙 스틸링을 위한 가정
  - 문맥교환(context switching)의 오버헤드(overhead)는 스케줄링 시 적용되지 않는다.
  - 태스크의 준비시간은 주기의 시작과 일치한다.
  - 태스크들간의 동기적인 문제나 충돌은 발생하지 않는다.
  - 선점(preemption)이 가능하다.

- 수식의 정의에 사용되는 기호와 의미

$U_i(t)$ 구간 [0, t]에서의 전체 계산량

$A_i(t)$: 비주기 태스크에 서비스를 위한 누적 계산량

$P_i(t)$: 우선순위 i의 태스크와 우선순위가 i 이상인 주기태스크의 계산 시간

A_u : 구간 $[C_{u-1}, C_u]$에서 우선순위 i로 비주기 테스

크에 할당된 시간

$I_i(t)$: 유휴구간과 i-레벨보다 우선순위가 낮은 주기 태스크의 수행시간

C_{ij} : i-레벨에서 j번째 태스크의 최악수행시간

D_{ij} : 주기 태스크 τ_i의 j번째 태스크의 제한시간

【정리 1】 $P_i(t)$는 임의의 시간 t 이전에 준비 시간이 경과한 우선순위 i 이상의 모든 주기 태스크의 계산량이다. C_i를 태스크 τ_i의 최대(최악) 계산시간이라 하고, T_i를 주기라고 할 때 $P_i(t)$는 다음 【수식 1】로 정의될 수 있다[9].

【수식 1】 $$P_i(t) = jC_i + \sum_{k=1}^{i-1}\left(\left\lceil \frac{t}{T_k} \right\rceil \times C_k \right) \quad [11]$$

【정리 2】 누적 슬랙 A_{ij}는 구간 $[0, C_{ij}^{'}]$에서 우선순위 레벨 i와 그보다 높은 우선순위에서 비주기 태스크를 처리할 수 있는 최대 가용슬랙이다.(단, $C_{ij}^{'} \le d_{ij}$, C_{ij}는 τ_{ij}의 최악 수행시간이다.)

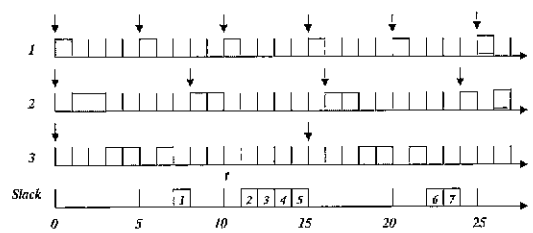
임의의 시간구간 [0, t] 사이에서 누적되는 슬랙 A_{ij}는 시간 t에서 주기적 태스크의 수행시간 $P_i(t)$를 제거한 최대값으로 【수식 2】로 정의되어 있다.

【수식 2】 $$\min_{\{0 \le t \le D_{ij}\}} \left\{ \frac{A_{ij} + P_i(t)}{t} \right\} \le 1 \quad [11]$$

【증명 1】 (그림 2-1), <표 2-1>를 참조하여 【수식 2】에 대한 결과를 다음과 같이 얻을 수 있다.

<표 2-1> Periodic Hard Tasks

Periodic Hard Tasks			
Priority	*Period*	*Deadline*	*wc_excution*
1	5	5	1
2	8	7	2
3	15	12	3



(그림 2-1) <표 2-1>의 Periodic Hard Task에서 발생된 Slack

∴ 임의의 시간 $t=10$에서 가용슬랙 A_{31}의 값은 다음과 같이 나타낼 수 있다

$\min_{\{0 \le 10 \le D_{31}\}} \left\{ \frac{A_{31} + P_3(10)}{10} \right\} \le 1$으로부터 1개의 가용슬랙 A_{ij}를 찾아낼 수 있다

이러한 방법을 반복하여 태스크 3까지 스케줄링이 완료된 경우에 가용슬랙을 $T_3=10$에서 $T_3+D_3=22$ 사이의 가용슬랙 값을 구해 보면 다음과 같은 순서쌍 집합으로 나타낼 수 있다.

$(\delta, A_{ij}) = \{(10, 1), (11, 1), (12, 2), (13, 3), (14, 4), (15, 5), (16, 5), (17, 5), (18, 5), (19, 5), (20, 5), (21, 5), (22, 5)\}$ 실행 시간에, $\delta=20$인 시간에 가용슬랙 A_{ij}는 5로 나타난다.

(단, A_{31}는 우선순위 레벨 i가 3임을 나타내고 j가 1임을 나타낸다, i는 우선순위, j는 인스턴스)

【정리 3】 누적 슬랙 함수 $A_i(0,t)$는 각 태스크 i가 그들의 모든 인스턴트들이 데드라인을 지나지 않으면서, 범위 [0, t]에서 우선순위 i와 그보다 높은 우선순위에서 비주기 태스크가 처리 할 수 있는 최대의 가용슬랙 값이다[9].

$A_i(0,t)$는 태스크 i의 모든 인스턴스의 스케줄링 가능성을 보장하도록 하며 다음 【수식 3】과 같이 정의된다.

【수식 3】 $A_i(0,t) = A_{ij}$, $C_{ij-1} \le t \le C_{ij}^{'}$, $j \ge 1$ [9]

슬랙 스틸링 알고리즘의 오퍼레이션은 경성 우선순위 태스크 i의 각 요청 τ_{ij}에서 누적된 슬랙 A_{ij}를 기록한 테이블에 의존한다. 그 값은 누적 슬랙 함수 $A_i(0,t)$를 정의하는데 사용되고, 각 태스크 i가 자신들의 데드라인을 만나게 되는 [0, t] 사이에서 우선순위가 i이며 그리고 그보다 높은 우선순위에서 비주기 태스크가 스케줄링 될 수 있도록 한다 누적 슬랙 함수 $A_i(0,t)$는 태스크 i가 자신의 데드라인을 지나치지 않음을 보장하는 시간 t에서 비주기 태스크를 처리할 수 있도록 하는 $A_i^*(t)$를 계산하는데 사용한다 슬랙 스틸링 알고리즘은 비주기적 태스크를 위해 활용할 수 있는 실제적인 처리시간 $A^*(t)$를 결정하는 함수 $A_i^*(t)$를 사용한다

【정리 4】 $I_i(t)$는 i-레벨 이하에서의 활용되지 못하는 유휴시간과 우선순위가 i보다 작은 주기 태스크의 계산량이다[9].

이상의 결과는 시구간 [0, t]를 기준으로 비주기 태스크에 할당 가능한 시간인 $A^*(t)$를 구한 것으로 비

주기 태스크가 시점 0에서부터 충분한 양이 도착되어 수행준비 상태일 때 사용 가능하다. 그런데 비주기 태스크의 도착 시간은 예측 가능하지 못하므로 [0, t] 사이의 임의의 시점 s에서 비주기 태스크에 할당 가능한 계산 시간은 $A^*(t)$를 이용하여 알 수 없다

【수식 4】
$$A^*(s,t) = \min_{(1\le j\le n)} A_j(s,t) = \min_{(1\le i\le M)} A_i(s,t) = A_y-(A'(s)+I_i(s)), \quad C_{y-1}\le t\le C_y,\ j\ge 1 \qquad [9]$$

위의 【수식 4】에서 $A'(s)$는 [0, s]에서 높은 비주기 태스크를 위해 이미 사용된 비주기 태스크들의 계산시간이고, $I_i(s)$는 [0, s]에서 idle시간을 포함한 level-i의 비사용 구역이다.

A_y에서 $A'(s)+I_i(S)$를 제외한 값으로 $A_i(s,t)$가 정의되는 이유는 A_y는 구간 [0, t]에서의 $A_i(t)$와 $I_i(t)$로 구성되는 고정적인 값이기 때문이다.

2.2 알고리즘의 문제점

슬랙 스틸링 알고리즘은 비주기 태스크의 발생에 따라 슬랙 스틸링 서버가 적합한 우선순위를 비주기 태스크에 부여하여 즉시 서비스 가능하도록 함으로서 불필요한 대기시간을 최소화하고 있다. 하지만, 슬랙 스틸링을 수행하기 위해서는 임의의 시점까지 주기적 태스크의 수행 시간을 구해야 한다. 그리고, 주기적 태스크의 수행시간은 슬랙 알고리즘을 적용하는 동안 매 시간 마다 다시 구해지고 있다. 이때 사용되는 시간 복잡도는 계산에 적용되는 태스크의 수가 n이라면 $O(n)$으로 나타난다. 따라서, 매 시간마다 불필요한 계산을 반복하게 되므로 실행시간이나 매우 짧은 시간 내에 슬랙을 계산하기 위해서는 개선된 슬랙 알고리즘을 요구하게 된다

3. 개선한 슬랙 스틸링 알고리즘

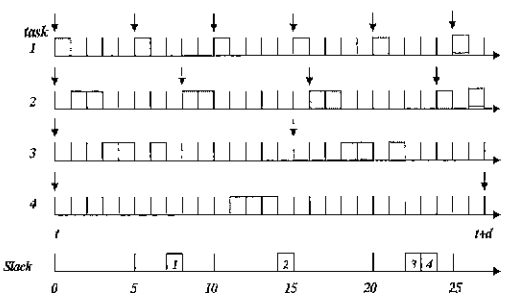
스케줄링 시 비주기적 태스크가 도착하게 되면 현재 슬랙이 있는지를 판단해야 한다. 이 판단에 필요한 시간 복잡도는 모든 대역폭 보존 알고리즘들이 $O(1)$로 나타난다. 반면에 슬랙 스틸링 알고리즘은 모든 우선순위 수준에 대하여 이를 검사해야 하기 때문에 시간 복잡도는 $O(n)$이다. 따라서 슬랙 스틸링 알고리즘의 슬랙 계산 시간이 다른 대역폭 보존 서버에서 나타나는 복잡도에 근접할 수 있도록 한다.

3.1 슬랙 계산 방법

주기 태스크가 수행하는 동안 주기적 태스크의 값을 테이블에 저장하고 이후에 슬랙 타임을 계산하는데 사용하게 된다면 태스크들이 실행되는 도중에 비주기 태스크를 위한 슬랙을 계산하게 되는 경우 최대한 빠른 시간에 슬랙을 구할 수 있다. 그러므로, 비주기 태스크 스케줄링 방법 중 알고리즘의 복잡성과 응답시간을 개선할 수 있다. 본 알고리즘은 비주기적 태스크 스케줄링의 궁극적인 목표가 모든 주기적 태스크의 마감시간을 만족하면서 비주기적 태스크에 대한 응답시간을 최대한 빠르게 함과 동시에 알고리즘의 구현이 용이해야 하고, 슬랙을 계산하는 방법이 단순해야 한다는 관점으로 접근한다. 다음 <표 3-1>은 주기적 태스크 집합으로부터 슬랙 계산 방법을 설명하고 제안된 주기적 태스크 집합의 테이블을 다음과 같이 나타낸다.

〈표 3-1〉 슬랙 계산을 위한 Periodic Task Set

Periodic Hard Tasks			
Priority	*Period*	*Deadline*	*wc_excution*
1	5	5	1
2	8	7	2
3	15	12	3
4	30	27	3



(그림 3-1) 〈표 3-1〉의 Periodic Task Set에서 발생된 Slack

<표 3-1>로부터 다음과 같은 과정을 통하여 가용슬랙을 구한다.

① 【수식 1】에서 $P_i(t) = jC_i + \sum_{k=1}^{i-1}\left(\left\lceil \frac{t}{T_k}\right\rceil \times C_k\right)$ 값은 다음과 같다.

$P_1(t+d)=6,\ P_2(t+d)=14,$

$P_3(t+d)=20,\ P_4(t+d)=23$

② 【수식 2】 $\min_{(C_{j-1}\le t\le D_i)}\left\{\frac{A_y+P_i(t)}{t}\right\}\le 1$로부터 다음과 같이 가용슬랙을 계산한다.

A_y는 $[t, t+d]$ 사이에서 정적 슬랙 스틸링 방법으로 가용슬랙의 순서쌍 (δ, A_y)은 다음 <표 3-2>와 같이 나타난다.

〈표 3-2〉 (δ, A_y) 가용 슬랙의 순서쌍

(δ, A_y)	1	2	3	4	5	6	7	8	…	26	27
task 1	0	1	2	3	4	4	5	6	…	20	21
task 2	0	0	0	1	2	2	3	4	…	13	13
task 3	0	0	0	0	0	0	0	1	…	7	7
task 4	0	0	0	0	0	0	0	1	…	4	4

③ 【수식 2】로부터 구해진 값은 테이블에 저장된다. 저장되는 부분은 [알고리즘 1]에 기술한 테이블 Slack_Table[i][Current_Time].accumlation_Slack에 저장된다. (기존의 slack 알고리즘은 이 부분이 없으며 개선된 slack 알고리즘에 추가된 부분이다).

④ 【수식 4】로부터 구간별 슬랙을 구하도록 한다

⑤ 【수식 4】에 의해 구간별 가용슬랙이 구해지고 그 값을 테이블로 저장을 한다.

⑥ 비주기 태스크가 도착한 경우.

㉠ 기존의 slack 알고리즘.

비주기 태스크가 도착하게 되면 기존의 slack 알고리즘은 비주기 태스크의 가용 슬랙 계산을 위해 주기 태스크의 계산시간 【수식 1】을 연산하기 시작한다. 따라서, 【수식 1】은 태스크의 개수가 n이면 n번의 연산횟수를 나타내어 Time-Complexity가 $O(n)$이 된다.

㉡ 개선한 slack 알고리즘.

비주기 태스크가 도착하게 되면 [알고리즘 1]에 기술한 테이블 Slack_Table[i][Current_Time].accumlation_Slack으로부터 주기 태스크의 크기를 참조하여 사용하므로 【수식 1】 부분을 수행하지 않으므로 Time-Complexity가 $O(1)$이 된다. 이 결과는 태스크의 수가 작은 경우이고, 태스크의 수가 증가함에 따라 이 값은 $O(\log n)$으로 나타난다.

모든 주기적 태스크의 수행시간을 시간 단위 별로 기록하기 위한 테이블의 크기가 너무 커지면 메모리 관리측면에서 비효율적이므로 모든 태스크의 하이퍼주기(H) 동안만 스택에 저장하여 사용한다 모든 태스크의 주기로 최소 공배수를 하이퍼주기를 사용하는 이유는 모든 태스크의 주기로 생성된 최소 공배수를 중심으로 태스크들의 수행이 반복되기 때문이다.

<표 3-2>에서 각 시간 별로 정적 슬랙 스틸링 알고리즘에 의해 가용슬랙이 테이블에 저장되고 이 테이블에 주기적 태스크의 수행시간을 첨가하여 <표 3-3>을 구성한다 <표 3-3>이 하이퍼주기 만큼 구성되면 그 이후에 발생하는 비주기적 태스크를 위한 슬랙 스틸링 알고리즘의 수행은 다음과 같이 수행된다

① 스케줄링 가능성 분석

② RMS 방법에 의해 주기 태스크 스케줄링

③ 【수식 2】에서 계산되는 주기적 태스크의 수행시간 $P_i(t)$를 구하여 테이블 생성

④ 누적 가용슬랙 A_y는 테이블을 참조하여 가용슬랙을 구함

⑤ 주기적 태스크의 수행시간 요구시 테이블 참조

⑥ 구간별 가용슬랙 생성

〈표 3-3〉 (δ, A_y) 가용 슬랙과 주기적 태스크($P_i(t)$)의 수행시간

(δ, A_y)	1		2		3		4		5		6		7		8		·	26		27	
task 1	0	1	1	1	2	1	3	1	4	1	4	2	5	2	6	2	·	20	6	21	6
task 2	0	1	0	2	0	3	1	3	2	3	2	4	3	3	4	4	··	13	13	13	14
task 3	0	1	0	2	0	3	0	4	0	5	0	6	0	7	1	7	·	7	20	7	20
task 4	0	1	0	2	0	3	0	4	0	5	0	6	0	7	1	7		4	23	4	23

3.2 알고리즘

슬랙 스틸링 알고리즘이 스케줄링 시 모든 레벨에 위치한 주기적 태스크를 고려하여 슬랙을 계산하는 것과는 달리, 개선된 알고리즘은 스케줄링 시 해당하는 주기적 태스크의 크기를 미리 테이블로 구성하여 사용하므로 슬랙 스틸링 알고리즘의 연산 횟수를 줄이며 알고리즘의 복잡성을 감소시킨다. 그리고, 미리 만들어진 테이블을 이용하므로 실시간 시스템모델에서 요구하는 예측성이 높아진다. 슬랙 스틸링을 사용하여 온라인 시 연산횟수가 $O(n)$으로 나타나는 데 비해 개선된 알고리즘은 $O(\log n)$으로 나타난다.

[알고리즘 1]은 주기적 태스크의 수행시간을 저장하는 개선된 슬랙 스틸링 알고리즘의 의사코드이다. 이 알고리즘은 태스크 집합 전체에 대하여 수행을 하며,

전체 주기적 태스크의 우선순위를 순서로 스케줄링을 시작한다. slack[MAX]는 전체 태스크 집합에 대하여 주기적 태스크가 사용 중 인지를 검토하고 현재의 우선순위에서 보다 높은 우선순위의 태스크 집합에 대하여 슬랙을 찾아낸다 각 태스크 집합으로부터 검색된 슬랙은 Slack_Table[i][j].accumulation_Slack 테이블에 저장된다. 이후에 발생되는 사용가능 슬랙을 계산하는 경우에 이 테이블을 참조하게 된다. [알고리즘 1]에서 사용된 $hp(i)$의 수행은 [알고리즘 2]에 나타내었다. [알고리즘 2]에서는 전체 태스크 집합 중에서 우선순위가 현재의 스케줄링 위치에서 i 레벨보다 상위의 우선순위를 가지는 태스크 집합에 대하여 주기 태스크의 수행시간을 구한다. 기존의 슬랙 스틸링 알고리즘은 [알고리즘 2] 부분을 슬랙이 요구되는 시점에서 태스크 집합의 개수 만큼을 수행하게 된다. 그러나 개선된 알고리즘에서는 테이블 내에서 참조하게 되므로 [알고리즘 2]를 스케줄링 시에 구성하고 슬랙(slack)요청이 있을 경우에만 참조를 하여 가용슬랙을 구할 수 있다. 알고리즘에 나타나는 $hp(i)$는 우선순위가 i 이거나 또는 i 이상인 태스크의 집합을 의미한다

```
while(jobRemain()) { // 태스크 집합이 자신의 수행시간을 모두 수행
  for( MAX ){     // MAX 까지 수행 // 전체 태스크 대한 슬랙 검사
 if ( slack[MAX] == 0 // 슬랙이 있다면)
 while( ∀j ∈ hp(i) ) { // i보다 높은 우선 순위의 태스크 집합
 if( Blankslack[k+current_Time]  =='B'){
      // 스케줄링 될 수 있는 부분인지 검토
   Blankslack[k+current_Time] ='P',
      //높은 우선순위 주기 태스크가 사용 중
   PeriodeExcution();  // 모든 주기 태스크의 수행시간 검사
   Slack_Table[i][Current_Time ].accumulation_Slack = 누적 슬랙;
   }
    else{
     current_Time++;   // 다음 슬랙으로 이동
        }
    }
  }
}
```

[알고리즘 1] 개선된 슬랙 알고리즘의 의사코드

```
PeriodeExcution() {
while( ∀j ∈ hp(i) == TRUE){
 P_i(t) = jC_i + Σ_{k=1}^{i-1} ( ⌈ t/T_k ⌉ × C_k ) ;
 Slack_Table[i][Current_Time].period_excution =  P_i(t)
 // 주기 태스크 수행시간 저장
 // i 레벨의 우선순위에 현재의 시간을 표시하는 테이블 위치에 저장
 // 우선순위가 i인 주기 태스크와 i 이상의 주기 태스크 집합의 수행시간 계산.
    }
 return  P_i(t) ; // 주기 태스크의 수행시간 반환
  }
```

[알고리즘 2] 주기 태스크의 수행시간 계산 알고리즘의 의사코드

4. 시뮬레이션 및 평가

본 장에서는 시뮬레이션 모델을 통하여 슬랙 스틸링 알고리즘, 그리고 본 논문에서 제시한 알고리즘의 성능을 비교 평가한다.

본 논문에서는 다음과 같이 스케줄링 환경에 대한 가정을 한다.

- 단일 처리기 시스템 모델
- 연성 실시간 태스크 집합을 모델로 한다.
- 문맥 교환(Context Switching) 오버헤드는 고려하지 않는다.
- 태스크간의 선점 (Preemption) 가능

4.1 시뮬레이션 모델

시뮬레이션은 IBM PC상에서 NT4.0 OS를 기반으로 하였으며, CPU 400Mhz, RAM 64M을 사용하고, 알고리즘은 C++언어로 기술하였다 알고리즘 구현 시 프로그램을 수행하고 반환되는 시간의 측정은 Time 함수를 사용하였다

주기 태스크를 위한 부분은 고정 우선순위를 부여하여 스케줄링하고 비주기 태스크를 위한 부분은 동적인 스케줄링 방법이 사용된다. 본 논문에서 제시된 개선된 스케줄링 알고리즘은 주기 태스크를 위한 부분에서 사용된다 먼저 도착한 태스크 집합에 대하여 CPU 사용율을 고려하여 스케줄링이 될 수 있는지를 검토한다. 주기적 태스크의 정보를 테이블을 참조하여 예측하고 스케줄러(scheduler)로부터 우선순위를 할당받는다. 우선순위는 RMS(Rate Monotonic Scheduling)방법으로 우선순위를 할당하고, 주기적 태스크를 스케줄링 한다. 비주기적 태스크 집합이 Run-Time 시에 도착하면 혼합 스케줄러에 의해 주기적 태스크와 비주기적 태스크를 혼합하여 스케줄링 한다. 슬랙 스틸링 알고리즘에 의하여 혼합 스케줄링을 하는 동시에 주기적 태스크 집합이 스케줄링 된 이후에 발생하는 슬랙(slack)을 비주기 태스크를 위하여 할당한다. 실험에 사용될 주기 태스크의 집합과 비주기 태스크의 집합을 설정한다. 제시한 알고리즘이 비주기 태스크 집합의 수가 증가하는 경우 주기 태스크 집합의 수행시간을 구하면서 발생하는 오버헤드를 감소시키는 방법을 사용하므로 시뮬레이션 모델에서는 비주기 태스크가 전체 태스크 집합에서 어느 정도의 비율로 증가된 때 응답시간의 개선을 보이는

가에 중점을 두어 성능을 평가한다. 그리고 비주기 태스크의 비율에 따른 CPU의 이용율을 검토하여 주기태스크의 개수에 따라 어느 정도의 이용율이 나타나는가를 시뮬레이션하고 그 결과를 나타낸다.

4.2 응답시간 계산

[알고리즘 3]에서는 각각의 우선순위 레벨에서 비주기 태스크의 실행크기 만큼 수행을 하며 현재의 사용공간이 어떤 태스크가 사용중 인지를 표시해 주는 배열변수 slackBlank[]을 이용하여 비주기 태스크가 사용될 수 있는 슬랙인 경우 대기시간(waiting time), 반환시간(turnaround time), 응답시간(response time)을 구한다. 알고리즘에서는 정확한 클럭(cluck)으로부터 시스템 호출을 이용하여 응답시간을 알아내기 위하여 Time함수를 사용하였으며, Time 함수는 슬랙을 구하기 시작하는 시간부터 반환되는 시간까지의 시스템 시간을 호출하여 사용한다. 따라서, 시뮬레이션 과정에서는 본 알고리즘이 시스템에 따라 차이를 나타낼 수도 있다

```
for(j=0, i=0;  j<(aperiod[j].a + aperiod[j].c ; j++) {
      // 비주기 태스크의 수행 크기만큼 수행
if ( slackBlank[j] == 'A' || slackBlank[j]=='B')
{   // 비주기 태스크가 사용될 수 있거나 빈 공간일 경우
    waitingTime = currentTime - aperiod[j].a;  // 대기시간
    TurnArroundTime = waitingTime + aperiod[j].c ; // 반환시간
    responseTime = TurnArroundTime - aperiod[j].a; // 응답시간
    time4= clock();     // Time 함수를 사용하여 시스템으로부터
                        알고리즘의 응답시간 계산
...
   }
}
```

[알고리즘 3] 응답시간 계산을 위한 알고리즘의 의사코드

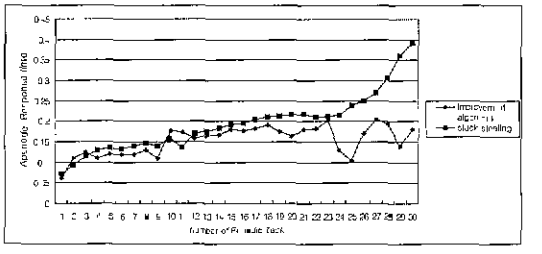
4.3 비주기 태스크의 비율에 따른 성능평가

성능평가에 사용된 주기 태스크와 비주기 태스크의 백분율에 대한 표를 다음과 같이 나타낸다.

〈표 4-1〉 시뮬레이션을 위한 주기 태스크 집합

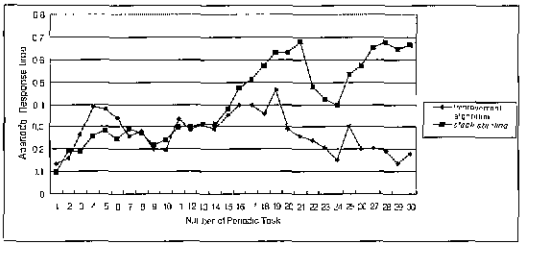
	Periodic Task set with 60%, 30% Aperiodic Task					
Task	Period	Computation	Period	Computation	Period	Computation
1	200	12	1650	20	1563	6
2	150	12	750	19	1450	8
3	120	4	685	14	453	12
4	113	14	650	18	450	2
5	355	5	500	9	589	9
6	1156	5	507	8	589	2
7	1506	15	695	8	622	4
8	1509	8	696	5	441	4
9	1606	9	636	4	580	8
10	1607	20	635	3	450	17

〈표 4-1〉에서 주어진 주기 태스크 값의 설정은 주기가 큰 태스크와 작은 태스크를 골고루 분포시키고, 수행시간을 2에서 20 사이의 임의적인 값으로 발생되는 값을 선정하였다.



(그림 4-1) 비주기 태스크가 60%일 경우 성능평가 (단위 ms)

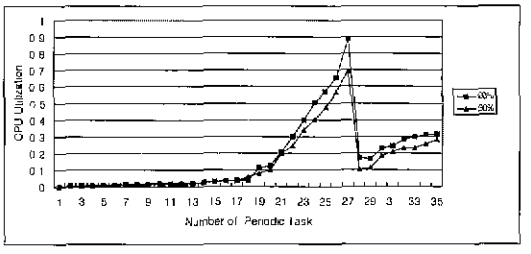
(그림 4-1)에 나타난 결과는 비주기 태스크가 전체의 60%일 경우에 비주기 테스크의 응답시간을 나타낸 것이다. 그리고 기존의 슬랙 스틸링 알고리즘은 주기 태스크의 수가 20미만일 경우에는 개선된 알고리즘과 유사한 응답시간을 나타냈고 20이상으로 증가할 경우 비주기 태스크에 할당 할 수 있는 슬랙(slack)을 계산하기 위하여 주기태스크의 수행시간을 계산하는 오버헤드(overhead)가 증가하기 시작하여 주기 태스크의 개수가 20일 경우에 응답시간이 0.2(ms)로 나타나고, 30개에서 최고 0.399(ms)까지 증가하였다.



(그림 4-2) 비주기 태스크가 30%일 경우 성능평가 (단위 : ms)

(그림 4-2)에 나타난 결과는 전체태스크의 수에서 비주기 태스크의 비율이 30%일 경우에 응답시간을 나타낸 것이다. 주기 태스크의 개수가 15일 때까지는 유사한 응답시간을 나타냈으며 기존의 슬랙 스틸링 알고리즘은 19개의 주기 태스크가 발생하면서 응답시간이 0.7(ms)로 가장 높게 나타났다 이때 개선된 알고리즘은 0.48(ms)정도로 나타나 기존의 슬랙 스틸링 알고리

즘보다는 응답시간이 적게 나타났지만 개선된 알고리즘도 슬랙이 감소할 경우에는 응답시간이 증가하는 것으로 나타났다 60%의 비주기 태스크를 포함하는 경우에는 그래프에서 비교적 안정적인 그래프 모양으로 응답시간의 편차가 비교적 적었지만 비주기 태스크의 양이 30%로 감소하여 주기 태스크의 양이 70%로 증가하면 슬랙을 계산하는 시간의 증가와 요구하는 슬랙의 양이 증가하여 불규칙적인 응답시간이 반환된다. 개선된 알고리즘에서는 주기 태스크의 수가 증가하여도 같은 조건으로 비주기 태스크가 분포할 경우 응답시간이 비교적 큰 폭으로 증가하지는 않았다. 따라서, 주기 태스크의 수가 증가하면서 슬랙 계산시간이 증가하는 것으로 나타나고 있다. 그리고, 태스크의 수가 28이상이 되면서 각각 60%, 30%의 비주기 태스크를 포함할 경우 모두 응답시간이 큰 폭으로 증가하고 있다.



(그림 4-3) 비주기 태스크의 비율에 따른 CPU 이용율

(그림 4-3)에 나타난 결과는 비주기 태스크의 비율에 따른 개선된 알고리즘의 CPU 이용율을 나타낸 것이다. 비주기 태스크의 비율이 전체의 비주기 태스크의 비율이 60%, 30%인 경우에 유사한 이용율을 보이지만 비주기 태스크의 비율이 작을수록 근소한 차이로 감소된 이용율을 볼 수 있다. 이결과는 적절한 시점에서 필요로 하는 만큼의 슬랙이 제공되지 못하기 때문이다. 따라서 비주기 태스크에 제공될 수 있는 슬랙은 적절한 타임에서 제공될 수 있어야 한다.

5. 결 론

실시간 시스템에 적합한 스케줄링 알고리즘은 마감시간(deadlinc)이내에 태스크 집합을 수행해야 한다 Polling Server와 함께 대분분의 대역폭 보존 알고리즘들은 비주기적 태스크 스케줄링 알고리즘 보다 반응속도를 높이는 결과를 나타내었지만 많은 제한점을 가지고 있다. 이 제한점을 보안한 슬랙 스틸링 알고리즘이 개발되었지만 슬랙 스틸링 알고리즘은 가용슬랙(A^*)을 구하기 위해 임의의 시점까지 주기적 태스크의 수행시간($P_i(t)$)을 구하는 오버헤드가 발생하여 실제 시스템에 적용하기에는 이렵다.

본 연구에서는 이러한 문제를 보완하기 위하여 가용슬랙을 구성하는 테이블 내에 주기적 태스크의 수행시간을 테이블로 구성하고 비주기 태스크를 위한 슬랙(slack)이 요구될 때 슬랙 계산을 위해 테이블을 이용하는 개선된 슬랙 스틸링 알고리즘을 제안하였다

성능 평가 결과 개선한 슬랙 스틸링 알고리즘에서는 기존의 슬랙 스틸링 알고리즘과 비교했을 때 비주기 태스크의 비율이 60%에서는 평균 0.07(ms)감소하였으며, 30%에서는 0.15(ms)로 감소하여 기존의 슬랙 스틸링 알고리즘보다 낮은 응답시간(response time)을 나타내었다. 따라서, 주기 태스크의 개수가 많아질 경우 발생하는 기존 슬랙 알고리즘의 오버헤드를 개선하였다.

향후연구 과제로는 개선된 슬랙 스틸링 알고리즘을 실시간 시스템에 적용하여 스케줄링 할 수 있도록 하고, 멀티 프로세서 환경에서 스케줄링 될 수 있는 방안이 요구된다. 그리고, 실시간 환경을 요구하는 RT-CORBA에 실시간 스케줄링 알고리즘을 적용하는 방법을 연구하고 있다.

참 고 문 헌

[1] Alia Atlas and Azer Bestavros, "Slack stealing job admission control," Technical Report BUCS-TR-98-009, Boston University, Computer Science Department, 1998.

[2] Alia Atlas and Azer Bestavros, "Maintaining quality of service or multimeia systems using statistical rate monotonic scheduling." Technical Report BUCS-TR-98-011, Boston University, Computer Science Department, 1998.

[3] B. Sprunt, L. Sha, and J. P. Lehoczky, "Aperiodic Task Scheduling for Hard Real_Time Systems," Real-Time Systems, Vol.1, pp.27-69, 1989.

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

[5] J, Lehoczky, L. Sha, and Y. Ding, "The Rate Monotonic Scheduling Algorithm : Exact Characterization and Average Case Behavior," In Proceedings of the 10th Real-Time Systems Symposium, pp. 166-171,1989.

[6] LuiSha, John P. Lehoczky, and Ragunathan Rajkumar. "Solutions for some practical problems in prioritized preemptive scheduling," IEEE Real-Time Systems, 1994.

[7] M. Spuri and G. C. Buttazzo, "Efficient Aperiodic Service under Earliest Deadline Scheduling," Proceedings of Real-Time Systems Symposium, pp.2-11, 1994.

[8] Neil C. Audsley, Alan Burns, Robert I. Davis, Ken W. Tindell, and Andy J. Wellings, "Fixed priority preemptive scheduling : An historical perspective, Real-Time System," 1993.

[9] Robert Davis, "Guaranteeing X in Y : On-line Acceptance Tests for Hard Aperiodic Tasks Scheduled by the Slack Stealing Algorithm," University of York, Y01 5DD, England, 1994.

[10] R. I. Davis, "Scheduling Slack Time in Fixed Priority Preemptive System," University of York, England, 1993.

[11] R. I. Davis, "Approximate Slack Stealing Algorithm for Fixed Priority Pre-emptive System," University of York, England, 1994

[12] R. I. Davis, "Dual Priority Scheduling : A Means of Providing Flexibility in Hard Real-Time System," Dept, Computer Science. University of York, 1996.

[13] S. R. Thuel and J. P. Lehoczky, "On-line Scheduling of Hard Deadline Aperiodic Tasks in Fixed-Priority System." In Processing of the 14th Real-Time Systems Symposium, 1993

[14] T. M. Ghazalie and T P Baker, "Aperiodic Servers in a Deadline Scheduling Environment," Real-Time System, Vol.9, No.1, pp 31-67, 1995

최 만 억

e-mail : muchoi@cs.suwon.ac.kr

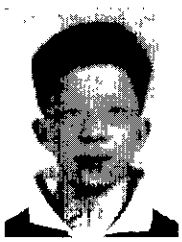
1990년 대전산업대학교 전자계산학과 졸업(학사)

1994년 수원대학교 대학원 전자계산학과 졸업(이학석사)

1999년 수원대학교 대학원 수학과 박사 수료

1995년~현재 수원대학교 시간강사

관심분야 : Open Distributed System, Client Server System, Distributed Object System(Real Time CORBA)



한 대 만

e-mail : han38@hanmail.net

1998년 한국 방송통신 대학 전자계산학과 졸업(학사)

2000년 수원대학교 대학원 전자계산학과 졸업(이학석사)

2000년~현재 수원대학교 대학원 전자계산학과 박사과정 재학

관심분야 : Open Distributed System, Real-Time Scheduling, Real-Time CORBA, Object Oriented Programming, ASP

구 용 완

e-mail : ywkoo@cs.suwon.ac.kr

1976년 중앙대학교 전자계산학과 졸업(학사)

1982년 중앙대학교 대학원 전자계산학과 석사과정 졸업(석사)

1988년 중앙대학교 대학원 전자 계산학과 박사과정 졸업(박사)

1983년~현재 수원대학교 자연과학대학 전자계산학과 정교수, 수원대학교 대학원(일반, 교육, 산업경영) 전자계산학과 주임교수, 동 대학교 전자계산소 소장

관심분야 : Operating System을 근간으로 한 Distributed System 및 System Software, Open System, Multimedia, Real Time System, Computer Network, Distributed Data Base, Software Engineering, 인터넷 응용, 전자상거래