https://doi.org/10.5392/JKCA.2021.21.06.012

애플리케이션 특징에 따른 CFS 태스크 우선순위 제어 기법

Task Priority Control Method based on the Characteristics of Applications in CFS

장준혁*, 이예나**, 홍지만***
한남대학교 컴퓨터공학과*, 숭실대학교 컴퓨터학과**, 숭실대학교 컴퓨터학부***

Joonhyouk Jang(jhjang@hnu.kr)*, Yena Lee(ynlee.oslab@gmail.com)**,
Jiman Hong(jiman@ssu.ac.kr)***

요약

비례 지분 스케줄러는 각 태스크에 상대적인 CPU 시간을 할당하고 태스크의 지분에 따라 실행할 태스크를 결정한다. 본 논문에서는 대표적인 비례 지분 스케줄러인 리눅스 Completely Fair Scheduler(CFS)에서 애플리케이션의 특징과 태스크 우선순위의 상관관계를 실험한다. 그리고 애플리케이션의 특징에 따라 태스크 우선순위 정밀도를 제어하는 기법을 제안한다. 제안 기법을 리눅스에서 구현하여 유의미한 실험 결과를 확인하였다.

■ 중심어 : | 스케줄링 | 우선순위 | 리눅스 | CFS |

Abstract

A proportional share scheduler allocates CPU time to tasks and determines which task will be dispatched according to their priorities. In this paper, we investigate the correlation between the characteristics of applications and task priorities in the Linux Completely Fair Scheduler(CFS), which is one of the representative proportional share schedulers. We also propose a method of controlling the granularity of priority assignments according to the characteristics of applications. We implemented the proposed technique in a Linux system and confirmed the meaningful experimental results.

■ keyword : | Scheduling | Priority | Linux | CFS |

I. 서 론

운영체제의 스케줄러는 준비 상태인 태스크 중에서 CPU 자원을 할당할 태스크를 결정한다. 스케줄러는 공정하고 효율적인 분배를 위해 CPU 사용률, 대기 시간, 공정성 등을 고려해야 하고, 이를 위해 다양한 스케줄링 기법이 활용된다[1]. 이 중 비례 지분 스케줄링은 각 태스크에 지분을 부여해서 스케줄링 우선순위를 결정하는 기법이다. 비례 지분 스케줄러는 실행 중인 각 태스크가 가진 상대적인 지분에 따라 CPU자원을 공평하게 할당한다[2].

버전 2.6 이전 리눅스 커널은 O(1) 스케줄러를 사용하였다. 리눅스의 O(1) 스케줄러는 스케줄링 알고리즘에 소요되는 시간이 태스크 개수에 관계없이 항상 일정하기 때문에, 태스크 개수가 많아져 스케줄링 함수를 빈번하게 호출하더라도 전체 시스템 성능은 크게 차이

* 본 논문은 ICCC 2020 국제학술대회 우수논문입니다.
* 본 연구는 2016년 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No. NRF-2016R1D1A1B01016073, 가상화 기반 오픈 게이트웨이 플랫폼과 오케스트레이션 보안 서비스 프레임워크 설계 및 구현).
접수일자 : 2021년 04월 07일
수정일자 : 2021년 05월 25일
심사완료일 : 2021년 05월 25일
교신저자 : 홍지만, e-mail : jiman@ssu.ac.kr

나지 않도록 설계되었다. 하지만 대화식(Interactive) 태스크가 많은 경우 평균 이하의 성능을 보인다.

현재 리눅스 커널은 O(1) 스케줄러의 단점을 보완한 완전 공평 스케줄러(CFS, Completely Fair Scheduler)를 사용하고 있다. CFS 스케줄러는 비례 지분 스케줄러의 일종으로서, 각 태스크에 비례 지분에 해당하는 가중치(weight)를 부여한다. CFS 스케줄러는 가중치에 반비례하는 가상 실행시간(Virtual Runtime, vruntime)을 기준으로 런큐(Run Queue)에 있는 태스크들을 정렬하며, CFS 런큐는 레드블랙 트리(Red-Black Tree, RB-Tree)로 구현된다[3-8]. 그러나, CFS 스케줄러도 각 태스크 별로 가상 실행시간과 태스크의 나이스(nice) 값에 따른 가중치 값을 구하기 위한 계산량이 많으며, 특정 애플리케이션 태스크가 많은 경우 BSD의 ULE 스케줄러보다 성능이 떨어진다[4].

따라서, 본 논문에서는 리눅스에서 비실시간 애플리케이션 태스크들의 특성에 기반하여 성능이 개선된 CFS 스케줄러를 제안한다. 제안한 CFS 스케줄러는 기존 나이스 값에 기반한 가중치 테이블을 간소화하여 태스크 별 가상 실행시간을 계산하는 방법을 개선한다. 또한 제안한 CFS 스케줄러의 성능을 평가하기 위해 제안한 CFS 스케줄러를 리눅스 커널 5.7에 구현하고 기존 CFS 스케줄러 상에서 여러 개의 비실시간 애플리케이션들을 실행시켜 제안한 CFS 스케줄러가 비실시간 애플리케이션 태스크의 성격에 따라 시스템의 성능을 개선할 수 있음을 보인다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구를 소개하며, 3장에서는 기존의 CFS 스케줄러를 분석한다. 4장에서는 제안한 CFS 스케줄러에 대해 설명한다. 5장에서는 제안한 CFS 스케줄러의 성능을 평가 하고 6장에서 결론을 맺는다.

II. 관련 연구

운영체제에서 모든 태스크들은 실행을 시작하면서부터 경쟁적으로 운영체제에 CPU 사용을 요구하고 운영체제는 스케줄러를 통해 요청한 태스크들에게 CPU를 공평하게 할당한다. CPU나 태스크 입장에서 CPU의 사용은 실행 시간을 의미한다[4].

범용 운영체제에 실행 시간을 할당하는 스케줄러는 태스크들에게 공정해야 하며, 특정 태스크가 기아(Starvation) 상태에 빠지지 않게 해야 한다. 또한 CPU를 효율적으로 사용해야 하며 스케줄링로 인한 비용이 크지 않아야 한다.

운영체제의 기본 태스크 스케줄링 기법으로는 FCFS(First Come First Service), SJF(Shortest Job First), SRN(Shortest Remaining Time Next), RR(Round Robin), MLFQ(Multi-Level Feedback Queue) 등이 있다[4].

이러한 스케줄러 기법 중 실제 범용 운영체제에서는 RR과 MLFQ 스케줄러의 변형을 실제 구현에서 사용하고 있다. 또한 실제 스케줄러 구현에서는 운영체제마다 다르게 정의된 PCB(Process Control Block) 구조체에서 정의된 맴버 변수(nice, prioirty 등의 변수)를 활용하여 고유한 특성을 갖는 스케줄러를 사용한다.

최근 범용 운영체제에서 적용된 스케줄러로는 공평몫(Fair-Share) 스케줄러, 완전 공평 몫(CFS, Completely Fair-Share) 스케줄러, O(1) 스케줄러, ULE 스케줄러 등이 있다[4]. 공평 몫 스케줄러는 하나의 사용자 애플리케이션이 여러 개의 태스크들로 구성되어 있음을 고려하여 개발되었다. 또한 공평 몫 스케줄러는 사용자 관점에서는 개별 태스크보다는 자신의 태스크 집합(즉, 애플리케이션 전체)이 어떻게 동작하는지에 관심을 두고, 스케줄링의 단위를 개별 태스크가 아닌 태스크 집합 단위로 하여 CPU를 공평하게 태스크에게 할당한다.

기존 리눅스 커널 2.4의 스케줄러를 대체하여 완전히 새로운 스케줄러로 만든 것이 O(1) 스케줄러이며, 이는 리눅스 커널 2.5부터 적용되었다. O(1) 스케줄러는 우선순위 기반의 타임슬라이스(Time Slice, 시간 할당량) 할당의 문제를 해결하고 있으나, 없어진 큐(Expired Queue)에 있는 태스크의 기아 (Starvation) 현상이 생길 수 있다. O(1) 스케줄러는 태스크의 개수가 많을 때 성능이 크게 떨어진다.

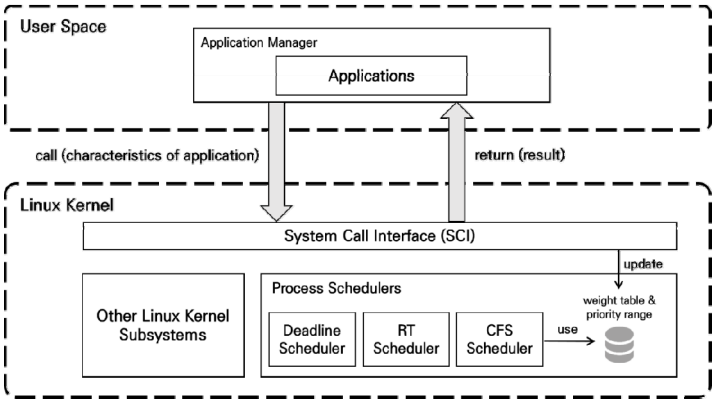

그림 1. 리눅스 시스템 구조

이는 O(1) 스케줄러 알고리즘이 O(1) 성능 복잡도를 갖고 있으나, O(1)이 되게 하기 위해서 태스크의 우선순위를 먼저 정렬해야 하는 문제점이 있기 때문이다.

리눅스 커널 2.6.23부터는 CFS 스케줄러가 구현되었다. 각 태스크는 실행 가능한 태스크들의 가중치의 총합을 기준으로 자신이 가지고 있는 가중치 만큼만 비례적으로 타임슬라이스를 할당 받는다. 예를 들어 프로세서가 100ms 동안 실행 된다고 가정하고 같은 나이스(nice) 값을 가지는 태스크가 두 개 있다고 할 때, 두 태스크가 나이스 값 -19를 가지든 20을 가지든 상관없이 타임슬라이스는 50ms씩을 얻게 된다. 만일 4개의 태스크만 존재하고 모두 같은 나이스 값을 가진다면 4개의 태스크는 모두 25ms의 타임슬라이스를 얻게 된다. 나이스 값을 절대적인 타임슬라이스와 분리함으로써 O(1) 스케줄러 등 기존 리눅스 커널 스케줄러의 문제점을 해결한다.

기존 리눅스 커널 스케줄러에서 동일한 나이스 값을 갖는 두 개의 태스크가 100ms의 동일한 타임슬라이스를 가지고 있고 스케줄러는 20ms 주기로 문맥교환(Context Swtiching)을 수행한다고 가정하면, 태스크의 20ms 주기로 문맥교환이 수행되기 때문에 각 태스크는 40ms와 20ms씩 실행이 되고 특정 시간에는 태스크 간 실행시간의 공정성은 떨어진다. 두 태스크가 공정하게 수행된다면 60ms의 시간이 지난 후에 두 태스크는 30ms씩 실행이 되어야 한다. 이러한 상황에서 CFS 스케줄러는 기존 리눅스 커널의 불공정함을 극복하기 위해 다음번 스케줄러에서 20ms를 실행한 태스크를 선택하여 40ms를 실행한 태스크와 공평하게 CPU를 차지할 수 있게 한다.

결국 두 태스크의 실행 시간 차이는 최대 문맥교환 주기인 20ms를 넘어서지 않게 된다.

CFS 스케줄러는 기본 런큐 배열을 사용하지 않고 모든 태스크의 실행 상태를 관리하기 위해 레드블랙 트리 구조를 사용한다. 레드블랙 트리는 균형 트리의 일종으로 트리의 삽입과 삭제 오버헤드가 O(logN)인 효율적인 균형 트리 중 하나이다. CFS 스케줄러의 레드블랙 트리 내에서 가장 왼쪽에 있는 태스크는 응답성이 가장 높다. 즉, 다음번 실행될 태스크가 되며 문맥교환의 대상이 된다. CFS 스케줄러의 핵심은 레드블랙 트리 구조에 태스크를 새로 추가하기 위해 키 값을 어떻게 결정하는가 하는 것이다. 또한 CFS 스케줄러의 다른 특징은 나노초(Nano Second) 단위의 정밀도로 동작하는 것이다.

III. 기존 CFS 스케줄러 분석

[그림 1]은 리눅스 시스템의 구조이다. 리눅스 커널에는 5개의 스케줄러(Stop, Deadline, RT, CFS, Idle Task)가 존재한다. 유저 태스크는 우선순위에 따라 실시간 태스크와 비실시간 태스크로 나눠진다. 비실시간 태스크는 SCHED_NORMAL, SCHED_BATCH, SCHED_IDLE 중 하나의 스케줄링 정책을 갖고 CFS 스케줄러에서 동작한다.

리눅스 커널은 0 이상 139 이하의 140개 우선순위를 사용한다. 우선순위가 0이상 99이하인 태스크는 RT

스케줄러에 의해 스케줄되고, 우선순위가 100 이상 139이하 태스크는 CFS 스케줄러에 의해 스케줄된다. 수식 (1)과 같이, CFS 스케줄러에서 태스크 우선순위(priority)는 사용자 공간에서 설정 가능한 나이스(nice) 값을 통해 결정된다. 태스크의 나이스 값이 작을수록 우선순위가 높은 태스크이다. 나이스 값의 범위는 -20 이상 19이하, 기본값 DEFAULTPRIO는 120이다.

$$priority = nice + DEFAULTPRIO \quad (1)$$

리눅스 CFS 스케줄러의 실제 의사결정은 의사결정 시점에 각 태스크의 가상 실행시간에 의해 결정되며, 태스크에 부여된 가중치는 태스크 가상 실행시간의 증감 속도를 결정한다. 수식 (2)는 나이스 값을 통해 태스크의 가중치(weight)를 계산하는 식이며, [표 1]은 나이스 값에 대응되는 가중치를 미리 계산한 상수들의 테이블 prio_to_weight 테이블이다. 예를 들어, 실행 중인 태스크가 2개일 때 나이스 값이 1 감소하면 가중치는 약 25% 감소하고 CPU 점유율에 해당하는 타임슬라이스(tslice)는 약 10% 증가한다.

$$weigth \approx \frac{1024}{1.25^{nice}} \quad (2)$$

표 1. 리눅스 커널 내부의 priority_to_weight 테이블

나이스 (nice) 값	가중치(weight)				
-20~-16	88761	71755	56483	46273	36291
-15~-11	29154	23254	18705	14949	11916
-10~-6	9548	7620	6100	4904	3906
-5~-1	3121	2501	1991	1586	1277
0~+4	1024	820	655	526	426
+5~+9	335	272	215	172	137
+10~+14	110	87	70	56	45
+15~+19	26	29	23	18	15

$$tslice = \frac{task's\ weight}{runqueue's\ weight} \times latency \quad (3)$$

수식 (3)과 같이, 태스크의 타임슬라이스(tslice)는 가중치와 스케줄링 지연시간(latency)에 의해 정해진다. 스케줄링 지연시간은 런큐에서 태스크들을 적어도 한 번씩 실행시키기 위한 시간 간격을 의미한다. 타임슬라이스는 스케줄링 지연시간을 각 태스크의 가중치로 나눈 값이다. 따라서 우선순위가 낮은 태스크일수록 가중치가 크고, 많은 타임슬라이스를 할당 받게 된다. 즉, 타임슬라이스는 런큐에 존재하는 태스크 개수와 우선순위 비율에 따라 결정된다.

IV. 애플리케이션 특성에 따른 CFS 스케줄러

제안 CFS 스케줄러에서는 태스크 가중치에 따른 가상 실행시간 및 CPU 시간 변화를 확인하기 위해 리눅스 커널의 priority_to_weight 테이블을 수정하였다. 먼저 priority_to_weight 테이블에서 기존 제공하는 40개 가중치를 n개씩 그룹화하여 40/n개 가중치로 수정하였고, 나이스 값에 따라 부여되는 가중치는 수정 전 해당 범위의 n개 가중치들의 평균값으로 대체하였다. 수정된 priority_to_weight 테이블은 n = 2일 때 [표 2]와 같이 20개의 가중치를 사용한다. 이때 비실시간 태스크의 우선순위는 +120 ~ +139가 된다. n = 5일 때는 [표 3]과 같이 8개의 가중치를 사용하게 된다.

표 2. priority_to_weight 테이블의 그룹화(n=2)

나이스 (nice) 값	가중치(weight)				
-10~-5	80258	51378	32723	20980	13433
-5~-1	8584	5502	3514	2246	1432
0~+4	922	591	379	244	155
+5~+9	99	63	41	26	17

표 3. priority_to_weight 테이블의 그룹화(n=5)

나이스 (*nice*) 값	가중치(*weight*)			
-5~-1	59912	19595	6415	2095
0~+4	690	226	73	22

다음으로 priority_to_weight 테이블의 가중치를 세분화하였다. 세분화는 실시간 태스크 우선순위 범위를 감소시켜 비실시간 태스크의 우선순위 범위를 확장했다. 수정된 우선순위 범위를 기반으로 새롭게 추가된 우선순위 값에 대응되는 가중치를 계산하여 새로운 priority_to_weight 테이블을 생성했다. [표 4]는 나

이스 값 범위를 -30 이상 29이하로 확장한 경우 제공되는 총 60개의 가중치를 나타낸다.

표 4. priority_to_weight 테이블의 세분화(60개 weight)

나이스 (*nice*) 값	가중치(*weight*)				
-30 ~ -26	827181	661744	529396	423516	338813
-25 ~ -21	271051	216840	173472	138778	111022
-20 ~ -16	88761	71755	56483	46273	36291
-15 ~ -11	29154	23254	18705	14949	11916
-10 ~ -6	9548	7620	6100	4904	3906
-5 ~ -1	3121	2501	1991	1586	1277
0 ~ 4	1024	820	655	526	423
5 ~ 9	335	272	215	172	137
10 ~ 14	110	87	70	56	45
15 ~ 19	36	29	23	18	15
20 ~ 24	12	9	8	6	5
25 ~ 29	4	3	2	2	2

사용자 공간에서 나이스 값을 설정하는 시스템콜이 호출되면 변경된 나이스 값이 set_user_nice()를 통해 task_struct 구조체에 반영된다. 이때, NICE_TO_PRIO 매크로에 의해 해당 나이스 값에 대한 우선순위 값이 추출된다. [표 5]는 태스크 우선순위 범위 확장을 위해 수정된 나이스 값 및 우선순위 관련 매크로 값이다.

표 5. 수정된 우선순위 관련 매크로 값

매크로 이름	값	
	기존	수정
MAX_NICE	19	24
MIN_NICE	-20	-25
MAX_USER_RT_PRIO	100	90

V. 성능 평가

1. 실험 환경

실험을 위해 [표 6]과 같은 환경의 머신 두 대를 준비하였으며, 둘 중 한 머신에서만 우선순위 및 가중치와 관련된 변수와 함수 일부분을 수정하였다.

표 6. 실험 환경

환경	설명
CPU	Intel Core i5-6400 2.7GHz
Memory	DDR4 4GB
Storage	SSD 256GB
Operating System	Linux 5.7.12

실험에서는 다양한 애플리케이션을 사용하여 애플리케이션의 우선순위가 애플리케이션의 성능에 미치는 영향을 평가하였다. fibo는 피보나치 수열을 얻는 프로그램으로 대표적인 배치(batch) 애플리케이션이다. sysbench는 멀티스레드 벤치마크 도구로서 대표적인 대화형 애플리케이션이다. 그 외 리눅스 벤치마크 테스트 툴인 Phoronix 등을 사용하여 실험을 진행하였다. 명확한 결과를 얻기 위해 애플리케이션들은 단독으로 실행되지 않고 다른 애플리케이션과 동시에 실행하였다.

수정된 가중치 범위에 따른 가상 실행시간을 측정하기 위해 리눅스 커널에서 제공하는 프로파일 도구인 ftrace를 사용하였다. ftrace의 트레이스 결과를 분석하여 가중치 값의 범위에 따른 가상 실행시간 변화를 측정하였다.

2. 실험 결과

[그림 2]는 가중치 값 범위에 따른 특정 태스크의 가상 실행시간 변화를 나타낸다. 가중치 값의 범위가 증가할수록 가중치 값이 더욱 세밀해지므로 가상 실행시간이 감소되는 것을 확인할 수 있다. 단, 세분화된 가중치 값에 의해 모든 태스크의 가상 실행시간이 감소 되었기 때문에, 다음번에 실행될 태스크 선택에는 변화가 없다. 따라서 가상 실행시간의 감소가 애플리케이션 성능 저하를 의미하지는 않는다.

[그림 3]은 서로 다른 특성을 가지는 두 애플리케이션 fibo와 sysbench를 우선순위의 값 및 범위가 수정된 리눅스 환경에서 실행한 결과이다. CFS 스케줄러는 애플리케이션의 특성에 상관없이 대화형 프로세스와 배치 프로세스에게 모두 CPU 사용시간을 공정하게 배분함을 확인 할 수 있으며, 비실시간 프로세스의 우선순위 범위를 확장 또는 축소하더라도 공정한 스케줄링이 가능하다는 것을 확인할 수 있다. 또한, 리눅스의 가중치 계산식을 사용하여 새롭게 계산된 가중치를 추가해도 애플리케이션 성능에는 큰 변화가 없음을 확인할 수 있다.

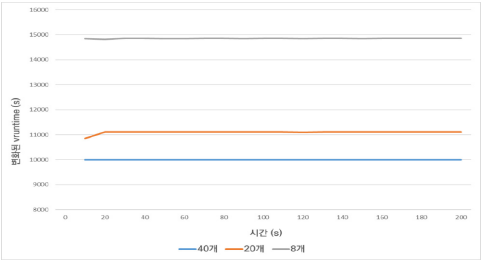

그림 2. 가중치 값 범위에 따른 태스크 가상 실행시간 차이

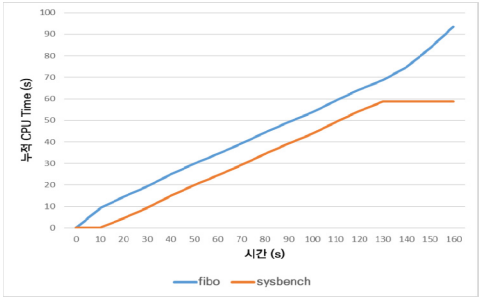

그림 3. 수정된 가중치를 적용했을 때 누적 CPU 시간

[표 7]과 [그림 4]는 다양한 애플리케이션을 우선순위의 값 및 범위가 수정된 리눅스 환경에서 실행한 결과이다. 과학 계산 애플리케이션들은 mflops를 측정했으며, 그외 애플리케이션들은 실행시간을 측정하여 비교하였다. 리눅스 CFS 스케줄러는 애플리케이션의 특성에 관계없이 대화형 프로세스와 배치 프로세스에게 모두 CPU 사용 시간을 공평하게 배분하는 것을 목적으로 한다. 하지만 가중치 값의 범위를 축소 혹은 확장했을 때, [그림 4]에서 fibo, c-ray와 같은 일부 배치 프로세스에서 더 좋은 성능을 보인다는 것을 확인할 수 있었다.

표 7. 가중치 값 범위에 따른 애플리케이션 성능 비교

애플리케이션	20	40	50	성능 측정 기준
fibo	292	295	290	실행시간
c-ray	256.64	322.29	254.36	실행시간
scimark	463.82	517.78	466.03	Mflops
himeno	2196.64	2527.94	2227.9	Mflops
sysbench	296.1187	291.1518	292.1518	실행시간

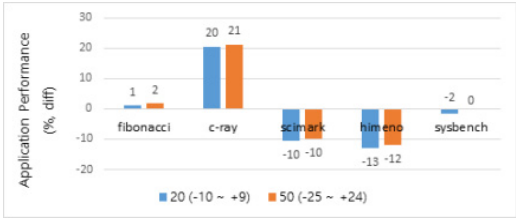

그림 4. 가중치 값 범위에 따른 애플리케이션 성능 비교

VI. 결 론

본 논문에서는 리눅스에서 비실시간 애플리케이션 태스크들의 특성에 기반하여 가중치 값을 변화시켜 실행 성능을 개선할 수 있는 CFS 스케줄러를 제안한다. 이를 위하여 태스크 우선순위를 결정하는 가중치 값의 범위와 개수를 다양화할 수 있도록 기존 리눅스 커널을 수정하고, 서로 다른 특성을 가진 애플리케이션들을 동시에 실행하여 애플리케이션들의 실행 성능을 측정하였다. 실험을 통해 가중치 값에 따라 타임슬라이스와 가상 실행시간과 같은 스케줄링 요소에는 차이가 발생하지만 다음에 실행될 프로세스의 선택이 달라지지는 않음을 확인했다. 또한 리눅스 커널에서 다수의 애플리케이션들이 동시에 실행되었을 때 CPU 코어를 공유하기 때문에 단독 실행할 때보다 애플리케이션 성능이 떨어지지만 이는 가중치 값 차이에 의한 것이 아니라 실행 가능한 태스크의 개수 차이에 의한 것이라는 점도 확인할 수 있었다.

참 고 문 헌

[1] R. S. Jamale, S. Dhotre, and P. T. Patil, “A Survey on Response Time Analysis Using Linux Kernel Completely Fair Scheduler for Data Intensive Tasks,” J. of Control Theory and Applications, Vol.9, No.44, pp.351-357, 2016.

[2] A. Roca, S. Rodriguez, A. Segura, K. Marquet, and V. Beltran, “A Linux Kernel Scheduler Extension for Multi-core Systems,” Proc. of 2019 IEEE 26th International Conference on

High Performance Computing, Data, and Analytics (HiPC), pp.353-362, 2019.
[3] C. S. Wong, I. K. T. Tan, R. D. Kumari, J. W. Lam, and W. Fun, "Fairness and interactive performance of O(1) and CFS Linux kernel schedulers," Proc. of 2008 International Symposium on Information Technology, pp.1-8, 2008.
[4] A. S. Woodhull and A. S. Tanenbaurn, *Operating Systems Design and Implementation (2nd Ed.)*, Prentice-Hall, 1997.
[5] C. S. Pabla, *Completely Fair Scheduler, Linux Journal*, Slashdot Media, 2009.
[6] M. Rouven and Y. Wiseman, "Medium-Term Scheduler as a Solution for the Thrashing Effect," The Computer Journal, Vol.49, No.3, pp.297-309, 2006.
[7] S. Dhotre and S. Patil, "Cause of process starvation for linux completely fair scheduler with apache server," Proc. of 2017 International Conference on Computing Methodologies and Communication (ICCMC), pp.814-819, 2017.
[8] M. Åsberg, T. Nolte, S. Kato, and R. Rajkumar, "ExSched: An External CPU Scheduler Framework for Real-Time Systems," Proc. of 2012 IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, pp.240-249, 2012.

저 자 소 개

장 준 혁(Joonhyouk Jang) 정회원

- 2009년 2월 : 서울대학교 전기컴퓨터공학부(학사)
- 2015년 8월 : 서울대학교 전기컴퓨터공학부(박사)
- 2021년 3월 ~ 현재 : 한남대학교 컴퓨터공학과 조교수

〈관심분야〉 : 시스템소프트웨어, 운영체제

이 예 나(Yena Lee) 정회원

- 2019년 2월 : 숭실대학교 컴퓨터학부(학사)
- 2021년 2월 : 숭실대학교 컴퓨터학과(석사)
- 2021년 1월 ~ 현재 : KT Digital Works Biz Project

〈관심분야〉 : 시스템소프트웨어, 운영체제

홍 지 만(Jiman Hong) 정회원

- 1994년 2월 : 고려대학교 컴퓨터학과 (학사)
- 2003년 2월 : 서울대학교 컴퓨터공학부 (석사, 박사)
- 2007년 3월 ~ 현재 : 숭실대학교 컴퓨터학부 교수
- ACM SigAPP Chair

〈관심분야〉 : 시스템소프트웨어, 운영체제