**운영체제 과제2**

**학번:** **2014182011**

**이름:**  **김태화**

**목 차**

1. **CFS (Completely Fair Scheduler)**
2. **CFS의 장/단점**

1) 장점

2) 단점

**3. 다른 스케줄러와의 비교**

**4. 구현 방법**

1. **CFS (Completely Fair Scheduler)**

1) CFS

- CFS는 2.6.23 이후로 리눅스의 기본 스케줄러로 사용되고 있는 알고리즘

- 프로세스 실행을 위해 CPU 자원 할당을 관리

- 상호작용 성능을 극대화하면서도 전반적인 CPU 이용을 극대화하는 것을 목표

- 기본 원리는 모든 프로세스를 공평하게 실행하는 것

**2. CFS의 장/단점**

1) 장점

- 프로세스에 공평한 양의 프로세서가 제공

- 현재 실행할 수 없는 작업이 나중에 프로세서가 필요할 때 대기했던

시간에 상응하는 프로세서 시간을 받을 수 있도록 보장

- 레드-블랙 트리를 이용해 스스로 밸런스를 조정하고, 빠르고 효율적으로

삽입 삭제가 가능하다.

- 우선 순위별로 실행 큐를 관리하지 않아도 되서 효과적인 방법

2) 단점

-

**3. 다른 스케줄러와의 비교**

1) 기존의 O(1) 스케줄러

- 다른 Nice값(Timeslice)를 가지는 작업을 실행을 할 때

ContextSwitching은 Timeslice와 같은 간격으로 발생하게 된다.

- Nice 값 0과 1을 가지는 두개의 작업이 실행될 때는 두 작업의

Timeslice는 100ms, 95ms가 할당될 것이다. 하지만 18과 19의

Nice 값을 가지게 되는 작업이 실행될 때는 10ms, 5ms 의

Timeslice를 가지게 된다.

- Timeslice가 TimerTick의 영향을 받아 변화한다.

- Interactive Task의 값을 우선시하는 문제가 있다. Expired 큐에

존재하는 작업의 실행이 지연되게 된다.

2) CFS

- Nice값을 작업이 할당 받을 proportion of processor에 대한 weight로

사용하게 된다. 각각의 작업들은 실행 가능한 작업들의 weight의 총 합을

기준으로 자신의 weight에 비례하게 Timeslice를 할당 받게 된다.

O(1)에서 일어나던 Nice값을 절대적인 Timeslice와 분리함으로써 해결

하였다.

- 두개의 작업을 실행한다고 가정하였을 때 각각의 작업에게 weight를

부여하고 그 값에 따른 비율로 Timeslice를 부여하게 된다. Nice값이

어떻게 부여되더라도 weight의 비율에 따라 Timeslice를 부여하기

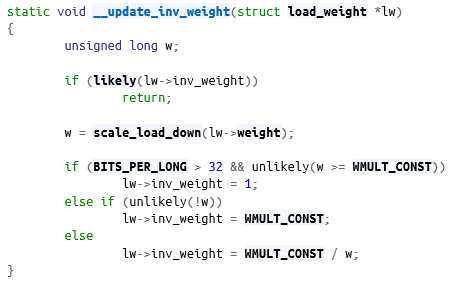
때문에 위의 사항과 같이 기존의 문제를 해결하였다.

**4. 구현 방법**



- 런 큐에 기반을 두지 않고 레드-블랙 트리를 사용해 타임라인을

구현하였다.



- Weight값에 따라서 업데이트를 진행한다.