CPU 스케쥴링

프로세스 스케줄링이란 CPU를 사용하려고 하는 프로세스들 사이의 우선 순위를 관리하는 일

스케줄링 종류

선점(preemptive) vs 비선점(Non-preemptive)

현재 CPU를 점유하고 있는 입장에서 선점, 비선점을 뜻한다. 즉 어떤 프로세스가 현재 점유중인 상태에서 해당 프로세스가 끝나기 전에 다른 프로세스로 점유가 넘어갈 수 있으면 선점형, 해당 프로세스가 끝날 때까지 다른 프로세스로 점유가 넘어가지 않으면 비 선점형!

개인적으로 용어가 와닿지는 않음...

- 선점형 스케줄링
    - RR(Round Robin), MLFQ(Multi Level Feedback Queue), SRTF(Shortest Remaining Time First)
- 비선점형 스케줄링
    - FCFS(First Come First Served), SJF(Shortest Job First), HRN(Highest Response Ratio Next)

선점형이 좋아보이지만 선점형 방식에서는 스위칭 오버헤드가 많이 발생하므로 이를 고려하여 결정해야한다.

FCFS(First Come First Served) Scheduling

- 동작방식: 먼저 CPU에 도착한 작업부터 차례로 CPU를 점유하는 **비선점형 스케줄링**
- 장점: 구현하기 매우 쉬움, 간단한 시스템에 자주 사용 됨
- 단점: CPU에 먼저 도착한 하나의 긴 프로세스로 인해 처리 시간이 짧은 프로세스도 오래 대기해야하는 문제가 발생할 수 있다. (반환 시간과 응답시간 모두 길어지는 문제점)

SJF(Shortest Job First) Scheduling

- 동작방식: 처리시간이 짧은 작업부터 CPU에 할당하는 **비선점형 스케줄링**
- 장점: 최소의 평균 대기 시간을 보장한다. 즉 최적의 CPU 효율을 가진다.

- 단점: 수행시간이 긴 프로세스는 계속 후순위로 밀려나는 기아 현상이 발생한다. 또한 각 프로세스의 CPU 사용시간(작업 처리시간)을 모르는 경우 사용하기가 어렵다.

SRTF(Shortest Remaining Time First)

- 동작방식: SJF의 선점형 방식! 현재 실행 중인 프로세스보다 더 짧은 실행 시간을 가진 새로운 프로세스가 도착하면 현재 실행 중인 프로세스를 중단하고 CPU를 새로운 프로세스에 할당하는 **선점형 스케줄링** 방식
- 장점: 처리 시간이 짧은 프로세스는 대기 시간과 응답 시간이 크게 감소.
- 단점: 여전히 기아 현상을 막을 수는 없다. 잦은 변경으로 오버헤드 발생, SJF와 마찬가지로 각 프로세스의 CPU 사용시간(작업 처리시간)을 모르는 경우 사용하기가 어렵다.

Round Robin(RR)

- 동작방식: 각 프로세스는 우선순위 없이 준비 큐에 들어온 순서대로 일정한 시간 간격(타임 슬라이스) 동안 CPU를 사용한다. 타임 슬라이스를 다 사용했음에도 작업이 끝나지 않은 프로세스는 대기 큐의 맨 뒤로 가서 대기하게 되며, 다음 프로세스로 스위칭이 일어나는 **선점형 스케줄링** 방식이다. 타임 슬라이스 크기에 따라 효율성이 결정된다. (크면 FCFS 방식과 가까워지고 작으면 스위칭 오버헤드가 커진다.)
- 장점: 일반적으로 프로세스의 응답시간이 빠르며 기아 현상이 발생하지 않는다.
- 단점: 잦은 스위칭으로 오버헤드가 크며, 우선 순위가 없기 때문에 중요한 작업에 더 많은 CPU 시간을 할당하고 싶을 때 사용하기는 부적절하다. (거의 유일하게 우선순위가 없는 스케줄링 방식인듯..?)

Multi Level Feedback Queue

- 동작방식: 여러 개의 큐를 이용하며, 프로세스가 우선순위에 따라 다른 큐로 이동하는 방식으로 동작한다. 프로세스에 대한 정보가 없을 때 짧은 작업을 먼저 실행시켜 반환시간을 줄일 수 있고 응답 시간을 줄일 수 있다. (모두 최적은 아님)

  어떤 작업이 CPU를 자주 양보하면(입 출력 대기) 해당 작업의 우선순위를 높게 유지한다. 한 작업이 긴 시간 동안 CPU를 집중적으로 사용하면 MLFQ는 해당 작업의 우선순위를 낮춘다.

구체적인 규칙은 다음과 같다.

- **규칙 1** : Priority(A) > Priority(B) 이면, A가 실행된다. (B는 실행X)
- **규칙 2** : Priority(A) = Priority(B) 이면, A와 B는 RR 방식으로 실행된다.
- **규칙 3** : 작업이 시스템에 진입하면, 가장 높은 우선 순위, 즉 맨 위 큐에 놓여진다.

    - 해당 작업의 처리 시간이 짧은지 긴지 알 수 없으므로 우선 가장 높은 우선 순위를 주고 CPU 사용시간에 따라 작업의 우선순위를 조정.
- ~~**규칙 4a** : 주어진 타임 슬라이스를 모두 사용하면 우선순위는 낮아진다. 즉, 한 단계 아래 큐로 이동한다.~~
- ~~**규칙 4b** : 타임 슬라이스를 소진하기 전에 CPU를 양도하면 같은 우선 순위를 유지한다.~~
    - 4a, 4b 적용시 문제점:
        - 시스템에 너무 많은 대화형 작업이 존재하면 그들이 모든 CPU 시간을 소모하게 될 것이고 따라서 긴 실행 시간 작업은 CPU 시간을 할당받지 못하는 기아 현상 발생
        - 사용자가 스케줄러를 속여서 지정된 몫보다 더 많은 시간을 할당하도록 하게 만들 수 있다. (ex. 타임 슬라이스가 끝나기 전에 아무 파일을 대상으로 입출력 요청을 내려 CPU를 양도하는 방식으로 우선순위를 유지)
- **규칙 4** : 주어진 단계에서 시간 할당량을 소진하면 (CPU를 몇 번 양도하였는지 상관없이 총 시간으로 계산), 우선순위는 낮아진다. (아래 단계의 큐로 이동한다.)
    - 시간 할당량을 얼마로 하는지가 가장 중요!
- **규칙 5** : 일정 기간 **S**가 지나면, 시스템의 모든 작업은 최상위 큐로 이동시킨다.
    - 기아 현상을 해결하기 위해 만들어진 규칙. CPU를 오래 점유하는 프로세스여도 최상위 큐에 존재하는 동안 RR형식으로 다른 프로세스들과 CPU를 공유하기 때문에 굶지 않음을 보장 받는다.
    - CPU를 많이 사용하여 우선 순위가 낮아진 프로세스가 입출력 위주로 작업이 변경되었을 때, 우선순위 상향을 통해 스케줄러가 변경된 특성에 적합한 스케줄링 방법을 적용할 수 있게 된다.

장점: 프로세스의 정보가 없을 때, 즉 여러 상황에서 평균적인 효율성이 좋음.

단점: 환경설정의 어려움(타임 슬라이스 설정, 최상위 큐로 이동시키는 시간 설정 등)

HRN(Highest Response Ratio Next)

HRN은 각 프로세스에 대한 응답 비율을 계산하고, 가장 높은 응답 비율을 가진 프로세스에 CPU를 할당하는 방식으로 동작하는 **비선점 방식**. 이 알고리즘은 우선순위와 대기 시간을 고려하여 프로세스를 스케줄링한다.

1. 응답 비율 계산:
    - 각 프로세스에 대해 다음과 같이 응답 비율을 계산한다.
    - 응답 비율 = (대기 시간 + 실행 시간) / 실행 시간
2. 가장 높은 응답 비율 선택:
    - 응답 비율이 가장 높은 프로세스에 CPU를 할당합니다.
    - 응답 비율이 높을수록 프로세스에게 높은 우선순위가 부여되며, 대기 시간이 긴 프로세스나 실행 시간이 짧은 프로세스에게 유리합니다.
3. 계속적인 갱신:
    - 프로세스가 대기하는 동안 대기 시간이 증가하고, 실행 중에는 실행 시간이 감소하므로 응답 비율은 계속해서 갱신됩니다.
    - 이러한 갱신은 새로운 프로세스가 도착하거나 현재 실행 중인 프로세스가 종료될 때 발생합니다.

HRN 알고리즘은 프로세스의 대기 시간과 실행 시간을 고려하여 가장 높은 응답 비율을 가진 프로세스를 우선적으로 처리하므로, 응답 시간을 최소화하고 프로세스의 대기 시간을 공평하게 분배하는 데 도움이 된다. 그러나 HRN은 계산량이 많고 복잡하며, 실제 시스템에서 사용하기 어려울 수 있다.

실제 운영체제에서 사용되고 있는 CPU 스케쥴링 기법

- **Best Job Scheduling Algorithm for Mac Operating System**

One of the commonly employed scheduling algorithms on Mac operating systems is the **Multilevel Feedback Queue (MLFQ) algorithm.** MLFQ is often used to optimize the execution time of tasks on the CPU. The Multilevel Feedback Queue algorithm is a dynamic priority scheduling algorithm that categorizes jobs into multiple priority queues. The MLFQ algorithm allows for jobs to move between queues based on runtime, processing time, and priority.

In a Mac operating system, the MLFQ algorithm can handle a mix of interactive tasks, background tasks, and real-time tasks effectively by assigning appropriate priorities. This algorithm ensures that interactive tasks, which require fast response times, are given higher priority, while background tasks

and less time-sensitive jobs are assigned lower priorities. This approach helps maintain system responsiveness, prompt completion time for tasks, and fast response time for new jobs.

- **Best Job Scheduling Algorithm for Windows Operating System**

In the Windows operating system, the default job scheduling algorithm used is the **Multilevel Feedback Queue algorithm**. Windows also supports a subset of scheduling algorithms through its Windows Scheduler API, including First Come First Served (FCFS), Shortest Job First (SJF), Round Robin, and priority scheduling.

The Windows Scheduler API allows developers and system administrators to customize job scheduling algorithms based on specific job requirements.

- **Best Job Scheduling Algorithm for Linux Operating System**

Choosing the best job scheduling algorithm for Linux depends on the specific Linux distribution, kernel version, and requirements of the system. A widely-used and highly regarded scheduling algorithm for Linux is the **Completely Fair Scheduler (CFS).**

The Completely Fair Scheduler is a process scheduler that fairly distributes CPU time among active processes. It utilizes a red-black tree data structure to maintain a queue of runnable processes, allowing for programmatic scheduling decisions. The CFS algorithm considers factors like process priority, virtual runtime, and load balancing to ensure fair distribution and optimal resource utilization.