6장 과제

11731017 김민경

1. CPU 스케줄링의 기본목표는 무엇인가?

CPU 스케줄링의 기본적인 목표는 시스템 처리율을 높이고 CPU의 유휴시간을 줄여서 효과적으로 사용하는데 있다. CPU 스케줄링이란 메모리에서 실행을 기다리는 프로세스 중 하나를 선택하는 과정이다.
예를 들어 I/O 와 같은 자원을 이용하는 프로세스가 있는 경우에 CPU는 다른 프로세스를 사용하도록 함으로서 CPU의 사용량을 극대화 하도록 한다. 또한 CPU의 유휴시간이 존재할 때 이를 줄이고자한다. 운영체제는 Ready Queue 에서 프로세스를 선택하여 실행되도록 하는데, 이때 스케줄러는 메모리에 있는 실행 준비가 되어있는 프로세스들을 선택하고 CPU 에게 그들 중 하나를 실행할것을 할당해주도록 해서 CPU의 IDLE TIME 을 줄인다. 하지만 ~~예외적으로~~ 실시간 시스템에서도 CPU 스케줄링을 하는데, 이는 CPU 활용률과 처리율 향상이 궁극적 목표는 아니다. 이와 같은 경우는 기한을 정해놓고 처리율을 향상시키는 것이 아니라 응답시간을 최소로 줄여야 하는 것이 우선이기 때문이다.

2. CPU 스케줄링 알고리즘들의 목표나 평가척도 등 스케줄링의 기준들을 6가지만 나열하고 간단히 정의하라.

스케줄링 알고리즘을 최적화하기 위해서는 다양한 기준들이 존재한다.
첫째, 높은 CPU 활용률이다. 컴퓨터의 전체 가동시간에 대한 CPU 사용시간의 비율을 높이도록한다. CPU는 이상적으로 시간을 100% 활용하도록 해야한다.
둘째, 처리율 극대화. 단위 시간당 처리하는 프로세스의 개수를 극대화한다.
셋째, 공정성. 모든 프로세스에게 CPU 사용시간을 공평하게 배분한다.
넷째, 응답시간 최소화. 대화식 사용자에 대한 응답시간을 최소화하기 위해 프로세스를 스케줄링한다.
다섯째, 대기시간 최소화. 프로세스가 준비 리스트에서 CPU 를 할당받을 때까지 대기 시간을 최소화한다.

여섯째, 소요시간 최소화. 프로세스가 컴퓨터 시스템에 진입한 시점에서 완료 시점까지 걸린 시간을 최소화한다.

3. CPU 스케줄링 알고리즘은 I/O 집중 프로세스를 선호하는가 CPU 집중 프로세스를 선호하는가? 그 이유는 무엇인가?

CPU 스케줄링 알고리즘은 대부분 I/O 집중 프로세스를 선호한다.
I/O 집중 프로세스는 CPU 집중 프로세스보다 상대적으로 긴 IDLE TIME을 가지고 있다. 다른 JOB과 비교할 때, I:O 장치는 CPU의 다른 명령보다 훨씬 처리 속도가 느리다. 따라서 스케줄러의 역할은 I:O 바운드 프로세스에게 가장 많은 시간을 주어서 CPU로 하여금 CPU 바운드 프로세스를 과도하게 기다리는 시간이 없도록 하며 대기 시간을 줄이도록 하는데 있다. 그리고 사용자들이 직접 입출력을 담당하는 경우에 동시다발적으로 빠르게 처리를 해주어야 하는 경우도 있다.
사용자들과 상호작용을 더 쉽고 빠르게 하기 위해 I/O 집중 프로세스를 더 우선시 하기도한다.

4. 스케줄링은 선점, 비선점으로 구분된다. 이를 구분하는 기준은 무엇인가?
그리고 현대의 운영체제 대부분은 어떤 방식을 취하는가? 비선점 알고리즘을 사용할 때 가장 좋은 점은 무엇이고, 가장 큰 문제점은 무엇이라고 생각되는가?

선점 스케줄링과 비선점 스케줄링을 구분하는 기준은 커널이 현재 실행 중인 프로세스나 스레드를 강제로 중단 시킬 수 있는지의 여부이다. 선점형 스케줄링의 경우 프로세스가 실행 중일때 언제든 프로세스를 중단 시킬 수 있다. 하지만 비선점형 스케줄링의 경우 만약 작업이 CPU에 할당이 되면 그 작업이 완료될 때까지 CPU가 제어권을 회수할 수 없다.
현대의 운영체제 대부분은 선점형 스케줄링을 사용한다. 선점형 같은 경우는 타임 슬라이스나 우선순위와 같은 기준이 있어서 중단될 수 있다. 따라서 비선점형 스케줄링 보다 비교적 유연한 처리가 가능하기 때문에 CPU활용률이 더 높다. 그리고 선점형 스케줄링은 단위시간당 task를 완료하는 개수가 더 많아 CPU 활용률이 높아서 현대의 운영체제에 적합하다.

비선점 스케줄링을 처리할 때 좋은 점은 프로세스 당 소요시간이 빠르다는 것이다. 그리고 순차적인 작업 처리가 가능하기 때문에 데드락 기아현상이 발생하지 않는다. 반면 다양한 정책 반영과 우선순위에 따른 처리를 반영할 수 없기 때문에 비효율적이다.

5. FCFS, RR, SJF, SRTF을 간단히 설명하고, 가장 현실적으로 많이 사용되는 것은 어떤것일까? 그 이유는 무엇이라고 생각되는가?

FCFS 는 큐에 도착한 순서대로 프로세스를 실행시키는것을 말한다. 이는 비선점 스케줄링으로 프로세스별로 큐에 도착한 시간을 기준으로 실행하며 별도의 우선순위는 없다. 따라서 기아현상은 발생하지 않는다.

SJF는 프로세스들에게 주어진 별도의 우선순위는 없고 실행시간이 가장 짧은 프로세스를 먼저 실행시켜 프로세스들의 평균 대기 시간을 최소화한다. 이는 비선점 스케줄링으로 실행을 끝날 때까지 중단시키지 않는다. 하지만 짧은 실행시간을 가진 프로세스가 계속 큐에 도착하면 긴 프로세스는 기아 현상이 발생할 수 있다.

SRTF는 프로세스들에게 우선순위는 없고 남은 실행시간이 가장 짧은 프로세스를 먼저 실행한다. 이는 선점형 스케줄링 방식으로 남은 실행시간보다 실행시간이 더 짧은 프로세스가 큐에 도착하면 현재 프로세스의 실행을 강제 중단시키고 새 프로세스를 실행한다. 따라서 짧은 실행시간을 가진 프로세스가 지속적으로 큐에 도착하면 긴 프로세스는 기아 현상이 발생가능하다.

RR는 일정 시간 간격으로 프로세스들을 번갈아 실행시키는 것으로 준비 리스트를 원형 큐로 구축하여 사용한다. 이는 타임 슬라이스가 지나면 강제로 중단시켜 큐에 삽입하는 것으로 선점 스케줄링 방식이다. 이와 같이 일정 할당 시간이 있기 때문에 기아 현상이 발생하지 않는다.

이와 같은 스케줄링 방식 중 가장 많이 사용되는 방식은 RR 방식이다. SJF, SRTF 는 대기 시간이 가장 작기 때문에 효율적인 스케줄링 방식이지만 프로세스의 예상 실행 시간을 예측할 수 없기 때문에 실질적으로 구현하기 어렵다는 한계가 있다. 그리고 비선점 방식인 FCFS과 같은 경우는 대기시간이 너무 길어서 보편적으로 사용하기에는 무리가 있다. 따라서 기아현상이 없고 구현이 간단하다는 FCFS 의 장점과 SJF, SRTF 의 대기시간이 상대적으로 적다는 장점을 적절히 혼합하여 RR를 구현하였다.

6. 스케줄링 알고리즘 중 대기 시간이 가장 짧은 알고리즘은 어떤 것인가?

SRTF는 남은 실행 시간이 가장 짧은 프로세스에 대해 먼저 스케줄링을 하기 때문에 SJF 보다 대기시간을 최소화 할 수 있다. 그리고 타임 슬라이스가 작은 RR의 경우에도 실행시간이 짧은 것이 먼저 실행된다는 점에서 SRTF와 비슷한 효과를 가질수 있지만 RR은 완료되지 못하고 기다리는 스레드의 개수가 많아질 수 있기 때문에 평균 대기시간이 SRTF 보다 길다.

7. 디스패치란 무엇이며 디스패처란 무엇인가?

디스패치란 '급파하다', '배달하다' 라는 뜻으로 예인 스케줄링된 프로세스나 스레드를 CPU로 보내 실행시키는것이다. 하지만 실제로 프로세스나 스레드를 CPU에 보낼수는 없다. 따라서 CPU 레지스터에 스케줄링된 스레드나 프로세스의 컨텍스트를 복귀시켜 CPU가 프로세스나 스레드를 실행시키도록 한다. 따라서 프로세스나 스레드를 CPU에 보낼수 있도록 커널에 실체가 있는 코드로 정의한 디스패처를 만들어 CPU에 정보를 디스패치하는 역할을 하도록 한다. 구체적으로는, 시스템 호출이나 인터럽트 서비스 루틴의 마지막 단계에서 실행되는 스케줄링 코드를 통해 선택된 프로세스나 스레드가 CPU 에의해 실행되도록 디스패처 작업을 실행한다. 작업과정은 다음과 같다. 첫째, 현재 실행 중인 프로세스와 스케줄러에 의해 선택된 프로세스(스레드) 사이의 컨텍스트 스위칭 과정. 둘째, 커널 모드에서 사용자 모드로의 전환. 셋째, 선택된 스레드가 이전에 실행을 중단한 상태에서 실행을 시작하도록 점프하는 과정. 여기서 디스패처 코드의 실행시간은 가능한 짧도록 작성되어야 한다.

8. 수업 시간에 다룬 알고리즘들에 대해 preemptive 알고리즘을 골라 적어라.

SRTF, RR, Priority Scheduling, MLQ, MLFQ 는 preemptive 알고리즘들이다.
SRTF는 큐에서 남은 실행시간 보다 실행시간이 더 짧은 프로세스가 큐에 도착했을 때 스케줄링 된다. RR는 타임 슬라이스가 지나면 강제로 중단시켜 큐에 삽입하고, 새 프로세스를 선택하는 스케줄링에 돌입한다. Priority Scheduling은 더 높은 우선순위의 프로세스가 큐에 도착하면 프로세스의 실행을 중단시키고 더 높은 순위의 프로세스를 실행시킨다. 그리고 MLQ 와 MLFQ 같은 경우에는 프로세스 실행 중 높은 순위의 큐에 프로세스가 도착했을 때 실행을 중단하고 새로 도착한 프로세스를 스케줄하면 선점 스케줄링이 된다.

9. 다른 알고리즘에 비해 RR 이 가진 장점과 단점은 무엇인가?

RR은 타임 슬라이스가 있어서 순차적으로 실행되므로 공정하다. 그리고 큐에 도착한 순서대로 타임 슬라이스에 맞춰 스케줄되므로 기아 현상이 없다. 또 구현이 용이한 장점 등의 이유로 CPU 스케줄링의 목표와 가장 부합하기 때문에 가장 많이 사용된다. FCFS과 SJF가 가진 극단성에 균형을 취하는 것으로 평가된다.

단, 잦은 스케줄링으로 인해 스케줄링 때마다 소요되는 컨텍스트 스위칭 오버헤드가 크고 타임 슬라이스 할당량에 따라 처리 결과가 달라진다는 단점도 있다.
만약 타임 슬라이스가 크다면 RR은 FCFS에 가까운 알고리즘이 된다.
타임 슬라이스가 작다면 RR은 SJF나 SRTF의 스케줄링 결과와 가깝게 된다.

10. 멀티 코어 시스템에서는 단일 코어의 스케줄링 알고리즘을 사용하게 되면 발생하는 2가지 문제점을 설명하라.

첫째, 멀티 코어 시스템에서 각 코어는 CPU 내 독립적인 캐시 메모리를 가지고 있다. 코어는 캐시에 적재된 코드만 실행하므로 스레드의 코드와 데이터는 실행 전에 메모리로 부터 캐시로 먼저 복사된다. 기본적으로 프로세스 내부의 캐시의 정보를 저장해야 하는데, 이때 만약 실행시키기로 결정된 새 스레드가 해당 코어에서 실행된 적이 없다면 캐시를 비우는 시간과 새로운 스레드의 데이터와 코드를 저장하는 시간 등의 컨텍스트 스위칭의 시간이 길어진다. 따라서 코어 친화성을 이용하여 스레드가 특정 CPU에서 실행되도록 제한하여 동일한 코어에서 계속 실행되도록 한다.

둘째, 잘못된 알고리즘으로 인해 코어의 유휴시간이 증가하여 어떤 코어에서는 작업이 몰리고 어떤 코어는 놀게 된다. 이는 전체 시스템의 처리율을 떨어뜨리게 되므로 부하 불균형 문제가 발생한다. 따라서 프로세스나 스레드를 무작위로 코어에게 할당하지 않아야 한다. 그리고 각 코어 당 대기 큐를 할당하여 idle 상태의 코어가 존재하지 않도록 작업을 적절히 분배해야 한다.

11. processor affinity란 무엇인가? 이것은 사용자가 결정하는가? 운영체제가 결정하는가? 당신의 생각을 말해보라.

Core affinity란 코어에서 실행중인 스레드가 중단 되었을 때 컨텍스트 스위칭이 발생하게 되는데 이에 드는 비용을 줄이기 위해 프로세스나 스레드가 특정 CPU에서 실행되도록 제한하는 스케줄러의 특징을 말한다. 이는 기본적으로 운영체제에서 결정하는데 코어마다 run queue를 두어서 코어 친화성을 이루어서 스케줄링 시 컨텍스트 스위칭 시간이 늘어나지 않도록 한다. 다만 만약 processor affinity에 대한 개념을 잘 이해하고 있는 사용자의 경우 사용자가 옵션으로 직접 결정할 수 있기도 하다.

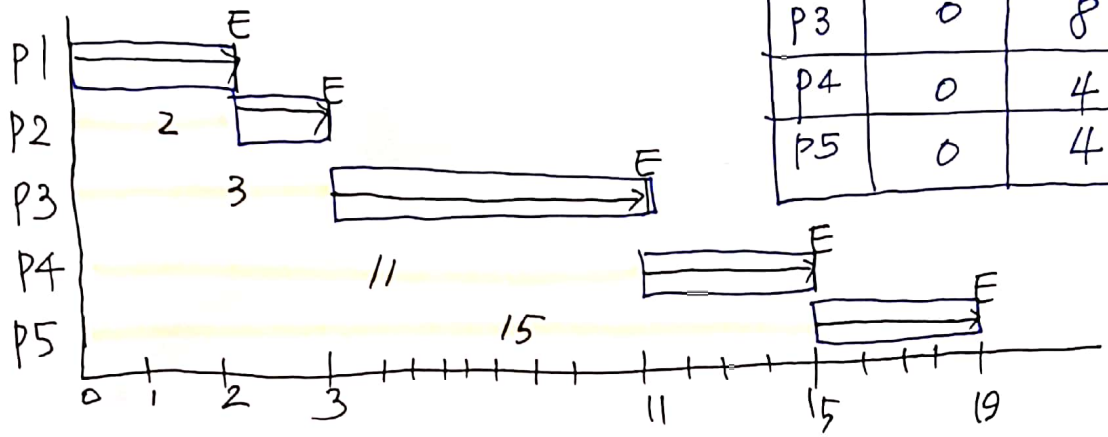
12. 다음 프로세스의 집합을 생각해보자. CPU burst는 단위가 밀리초이다.
프로세스들은 모두 0 시간에 도착해서 차례 P1, P2, P3, P4, P5 순서로 도착한다.

(a) FCFS., SJF, Non-preemptive Priority 스케줄링, RR (타임슬라이스 2밀리초) 각각의 스케줄링이 일어나는 과정을 그려라.

(b) 각 알고리즘의 총 처리시간, 평균 대기시간을 계산하라.

	도착시간	실행시간	우선순위
P1	0	2	2
P2	0	1	1
P3	0	8	4
P4	0	4	2
P5	0	4	3

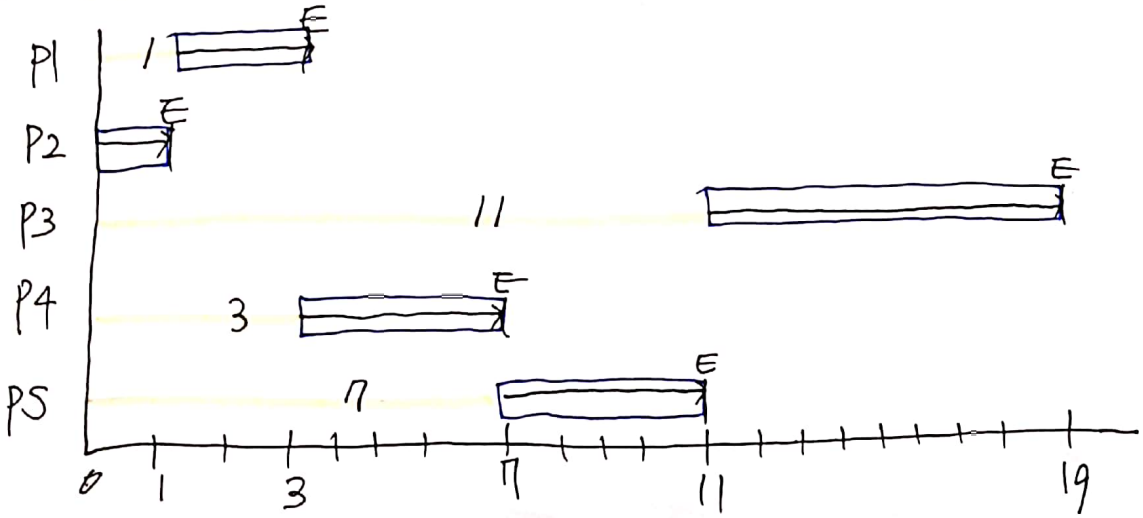
1) FCFS.



총 처리시간 = (2+1+8+4+4). = 19 ms

평균 대기시간 = (0+2+3+11+15)/5 = 31/5 = 6.2 ms

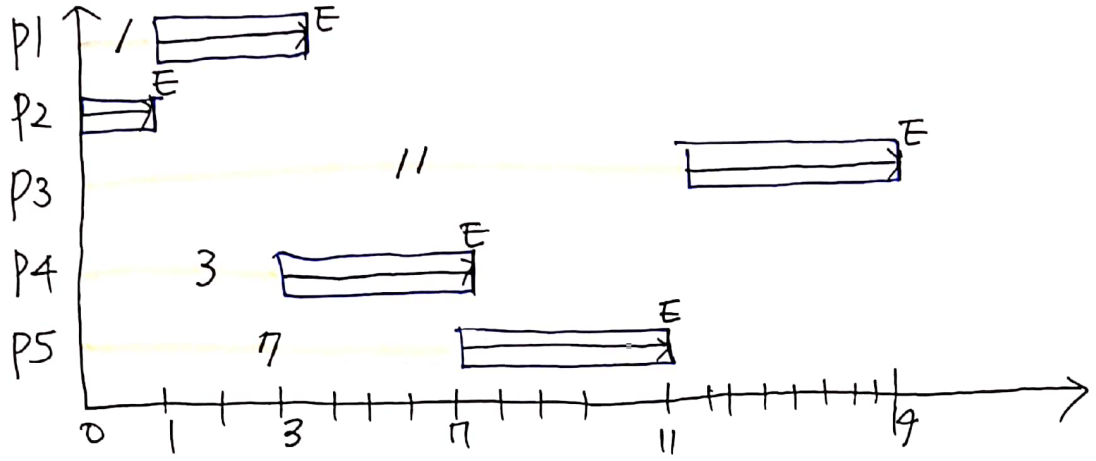
2) SJF



총 처리시간 = (1+2+4+4+8) = 19 ms

평균 대기시간 = (0+1+3+7+11)/5 = 22/5 = 4.4 ms

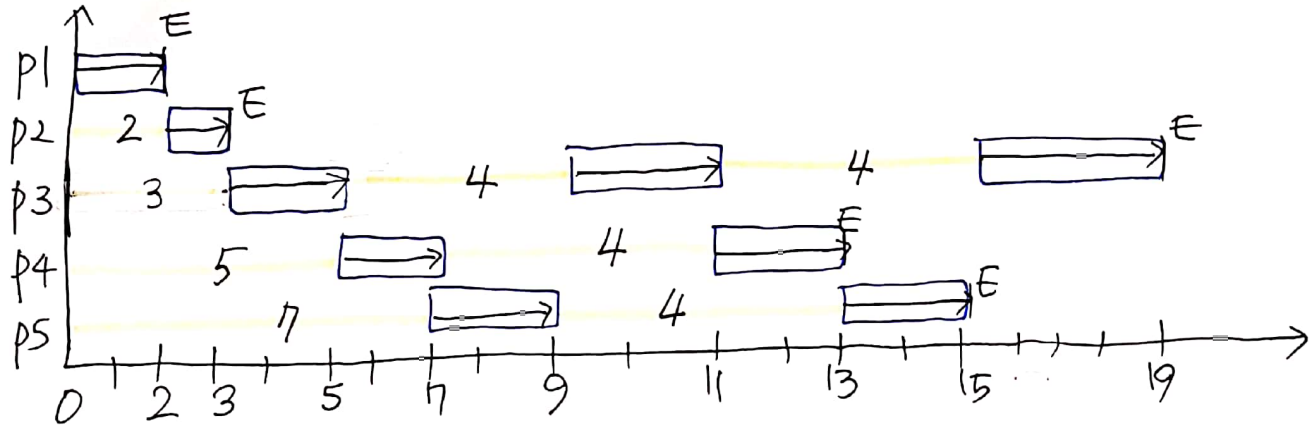
3) Non-preemptive Priority.



총처리시간 = (1+2+4+4+8). = 19 ms

대기시간 = (0+(1+3+7+11)/5 = 22/5 = 4.4ms

4) RR



총처리시간 = (2+1+8+4+4) = 19 ms

대기시간 = (0+2+11+9+11)/5 = 33/5 = 6.6 ms