질문	답변
CPU 스케줄링 알고리즘이 무엇인가요?	CPU는 하나의 프로세스 작업이 **끝나면 다음 프로세스** 작업을 수행해야 합니다. 이때 **어떤 프로세스를 다음에 처리**할 지 선택하는 알고리즘을 CPU Scheduling 알고리즘이라고 합니다. 따라서 상황에 맞게 CPU를 **어떤 프로세스에 배정하여 효율적으로 처리하는가**가 관건입니다. FCFS(First-Come First-Served)는 **먼저 도착한 프로세스를 먼저 실행**시키는 방식입니다. 하지만 이 방식은 **평균 대기 시간이 길어**질 수 있습니다. SJF(Shortest-Job-First)는 CPU **사용시간이 짧은 프로세스를 먼저** 실행하여 **평균 대기 시간을 최소화**합니다. RR(Round Robin)은 **정해진 시간** 동안 각 프로세스에게 CPU를 할당하고, 시간이 지나면 CPU를 **빼앗아 다음 프로세스에게 주는 방식**입니다.
선점형 스케줄링과 비선점형 스케줄링이 무엇인가요?	선점형 스케줄링은 CPU가 현재 실행 중인 프로세스를 **강제로 중단**시키고, **다른 프로세스를 실행**할 수 있는 기술입니다. 비선점형 스케줄링은 실행 중인 프로세스가 **I/O나 인터럽트가 발생해 종료**되거나 **대기 상태**가 될 때까지 계속해서 실행됩니다. 선점형 스케줄링 알고리즘에는 Round Robin, Priority-based 등이 있습니다. 비선점형 스케줄링 알고리즘에는 FCFS, SJF 등이 있습니다.
FCFS (비선점)	준비 큐에 **먼저 도착한 프로세스가 먼저 CPU를 점유**하는 방식이다. (일명 '선착순' 처리방식) CPU를 할당받으면 그 시간 동안 다른 프로세스는 CPU를 사용할 수 없습니다. 할당되었던 **CPU가 반환될 때만 스케줄**링이 이루어진다. 단점 Convoy Effect 발생 **소요 시간이 긴 프로세스가** 짧은 프로세스보다 **먼저 도착**해서 뒤에 프로세스들이 **오래 기다려**야 하는 현상

SJF(Shortest-Job-First) (선점, 비선점)	**CPU 실행시간이 짧은 프로세스**에게 CPU를 **먼저 할당**하는 방식이다. 선점, 비선점 모두 가능하다. **대기 시간 측면**에서 옵티멀(최적의) 방법 (선점형은 SRTF/SRT Shortest-Remaining-Time-First 라고 하기도 한다) **starvation 발생 (기아 상태)** 효율성만 고려해서 짧은 프로세스에게만 CPU 줌 **영원히 CPU 못얻는 프로세스**도 생김 누가 짧게 쓰고 갈게 쓰는지 미리 알 수 없음 따라서 **과거의 CPU 버스트를 통해 예측**한다 이러한 작업 방식은 실시간 시스템에 적합
Priority 우선순위 스케줄링 (선점, 비선점)	**우선순위가 높은 프로세스가 먼저 선택**되는 스케줄링 알고리즘이다. **선점, 비선점 모두 가능**하다. 문제 기아 상태 : **우선순위가 낮은 프로세스**는 절대로 실행될 수 없는 상황이 발생 해결책 Aging : **시간이 지날수록** 우선순위가 낮은 프로세스의 우선순위를 높임 Starvation(기아) 현상 **CPU의 점유를 오랜 시간 동안 하지 못하는 현상**을 의미한다. Priority 스케줄링 방식에서 우선순위가 **매우 낮은 프로세스가 ready queue에서 대기**하고 있다고 가정해보자. 이 프로세스는 아무리 오래 기다려도 CPU를 점유하지 못할 가능성이 매우 크다. 실제 컴퓨터 환경에서는 새로운 프로세스가 자주 ready queue에 들어온다. 이러한 프로세스가 모두 우선순위가 높은 상태라면 이미 기다리고 있던 우선순위가 낮은 프로세스는 하염없이 기다리고만 있는 상태로 남아있을 수 있다. 이를 해결하는 방법 중 하나는 **aging**이 있다. ready queue에서 기다리는 동안 일정 시간이 지나면 **우선 순위를 일정량 높여주는 것**이다. 그러면 우선순위가 매우 낮은 프로세스라 하더라도, 기다리는 시간이 길어질수록 우선순위도 계속 높아지므로 수행될 가능성이 커진다.

Round Robin 스케줄링 (선점)	**대기 큐의 순서**대로 처리하되 **정해진 시간만큼만 작업을 처리**하고 다음 작업으로 넘기는 알고리즘 **일정시간**을 정해 하나의 프로세스가 **일정 시간동안 수행**하고 다시 **대기 상태**로 돌아간다. 그리고 다음 프로세스 역시 같은 시간동안 수행한 후, 대기한다. 이러한 작업을 모든 프로세스가 돌아가면서 진행하며, **마지막 프로세스가 끝나면 다시 처음** 프로세스로 돌아와서 작업을 반복한다. 일정 시간을 Time Quantum(Time Slice)라고 부른다. 일반적으로 10 ~ 100msec 사이의 범위를 갖는다. 한 프로세스가 종료되기 전에 time quantum이 끝나면 다른 프로세스에게 **CPU를 넘겨주기 때문에 선점형 스케줄링**의 대표적인 예시다.
Multi-level Queue 스케줄링	**큐를 여러 개 분리**해놓고 각 **큐 별로 스케줄링** 알고리즘 지정 CPU를 기다리기 위해 여러 줄로 줄을 서는 것 각 큐마다 **다른 규칙**을 지정할 수도 있다.(ex. 우선순위, CPU 시간 등) 모든 프로세스는 가장 위의 큐에서 CPU의 점유를 대기한다. 이 상태로 진행하다가 이 큐에서 **기다리는 시간이 너무 오래 걸린다면 아래의 큐로 프로세스**를 옮긴다. 이와 같은 방식으로 대기 시간을 조정할 수 있다. 만약, 우선순위 순으로 큐를 사용하는 상황에서 우선순위가 낮은 아래의 큐에 있는 프로세스에서 starvation 상태가 발생하면 이를 우선순위가 높은 위의 큐로 옮길 수도 있다. 대부분의 상용 운영체제는 여러 개의 큐를 사용하고 각 큐마다 다른 스케줄링 방식을 사용한다. 프로세스의 성격에 맞는 스케줄링 방식을 사용하여 최대한 효율을 높일 수 있는 방법을 선택한다.

Multi-level Feedback Queue 스케줄링	일반적으로 **큐를 여러개** 두고 **피드백을 받아서 옮겨갈수** 있도록하는 알고리즘 멀티레벨 피드백 큐는 여러 개의 큐를 가진 스케줄링 방법 중 하나입니다. **각 큐는 우선순위에 따라 정렬**되어 있고, 높은 우선순위 큐부터 처리됩니다. 낮은 우선순위의 큐로부터 높은 우선순위의 큐로 프로세스가 이동할 수 있습니다. 프로세스가 도착하면 가장 높은 우선순위의 큐에 할당됩니다. 만약 그 프로세스가 CPU를 짧게 사용한다면(**작업이 짧다면), 그 큐에서 처리**됩니다. 하지만 작업 시간이 긴 경우, 다음으로 낮은 우선순위의 큐로 이동하게 됩니다. 여기서 피드백이란 실행 중인 프로세스가 다른 큐로 이동할 때 제어를 받는 것을 의미합니다. 프로세스가 긴 작업을 요청할 때 다른 큐로 이동하고, 그 큐에서도 실행 시간이 길면 다시 한 단계 아래로 이동하는 것을 반복합니다. 이렇게 여러 큐를 사용하면 **짧은 작업을 빠르게 처리**하고, 긴 작업을 밀어내어 **CPU를 오래 점유하는 것을 방지**할 수 있습니다. 이 방법은 프로세스의 종류나 실행 시간에 따라 적합한 큐로 효과적으로 할당하여 시스템의 효율성을 높일 수 있습니다. 또한, I/O 바운드 프로세스와 CPU 바운드 프로세스를 구분하여 처리함으로써 시스템의 성능을 향상시킬 수 있습니다.