



Scheduling



목차



프로세스 스케줄링

CPU Scheduling

발생 상황

Scheduling이 필요한 이유

Process 상태

프로세스 상태 변경



Scheduling Algorithm 구분

성능 척도

시스템 입장에서 성능 척도

프로그램 입장에서 성능 척도

선점형 vs 비선점형 방식

Preemptive 방식

Non-Preemptive 방식



Scheduling Algorithm

FIFO(FCFS)

SJF

HRN

RR

SRT

MLQ

MLFQ



참고



프로세스 스케줄링

CPU Scheduling

- Scheduler는 언제 어떤 프로세스를 선택해서 CPU에서 실행시키는지 선택하는 모듈
- CPU 효율 극대화를 위해 적절한 스케줄링이 필요

- 메모리에 여러 개의 프로세스를 올려놓고, CPU의 가동시간을 적절히 나누어 각각의 프로세스에게 분배하여 실행

발생 상황

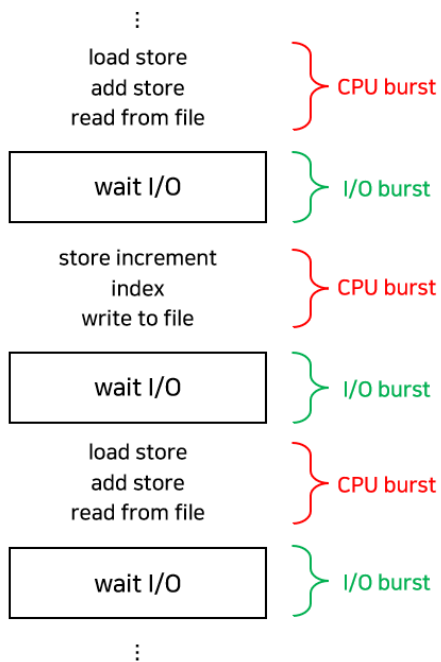
1. 실행 상태에 있던 프로세스가 I/O 요청 등에 의해 Block 상태가 되는 경우
2. Timer 인터럽트 발생에 의해 준비 상태가 되는 경우
3. I/O 요청으로 Block 상태에 있던 프로세스의 I/O 작업이 완료되어 인터럽트가 발생하고, 그 결과 이 프로세스의 상태가 준비 상태로 바뀌는 경우
4. CPU 실행 상태에 있는 프로세스가 종료되는 경우



I/O 작업의 예시

1. 파일 읽기
2. 파일 쓰기
3. 네트워크 통신
4. 사용자 입력
5. 그래픽 출력

Scheduling이 필요한 이유



- 프로세스는 두 단계의 반복으로 구성된 사이클 형태로 수행

1. CPU만 사용하는 단계 (CPU burst)

2. I/O만 사용하는 단계 (I/O burst)

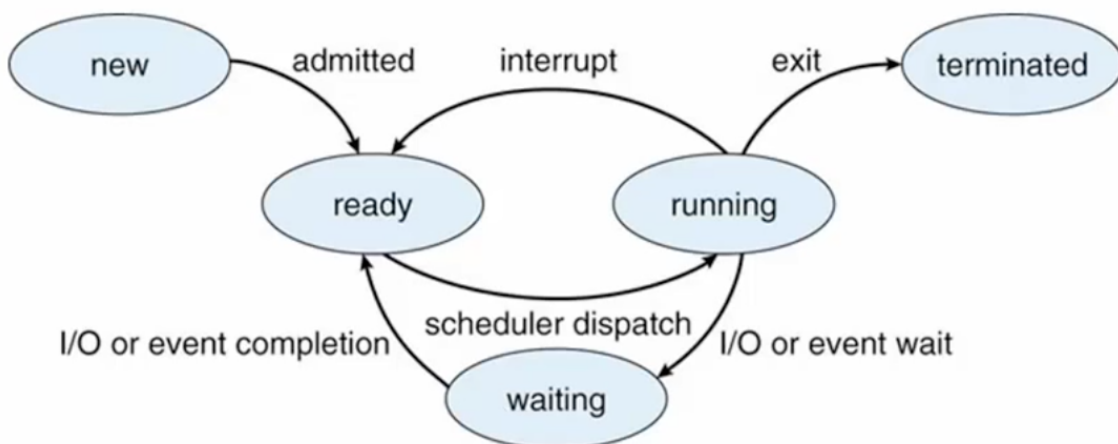
- CPU burst time의 분포에 따라 프로그램의 특성을 나타낼 수 있다

1. 짧고 많은 CPU burst가 존재하는 프로그램 → I/O-bound job

2. 길고 적은 CPU burst가 존재하는 프로그램 → CPU-bound job

- 이러한 여러 job이 섞여있기 때문에 CPU 스케줄링이 필요
- CPU Scheduler는 메모리에서 Ready 상태의 프로세스 중 어떤 프로세스를 CPU에 할당해줄지 선택

Process 상태



- New : 프로세스가 생성되는 중
- Running : CPU에서 명령이 실행되는 중

- Waiting : 프로세스가 어떤 Event(입출력 완료, signal 수신)이 발생하기를 기다리는 중
- Ready : 프로세스가 CPU에 할당되어 실행되기를 기다리는 중
- Terminated : 프로세스 실행 종료

프로세스 상태 변경

- CPU Scheduling으로 인해 프로세스의 상태가 변경된다
 1. Running → Waiting(Blocked) : I/O 요청 혹은 자식의 종료를 위해 wait() 함수를 호출한 경우
 2. Running → Ready : 인터럽트가 발생한 경우
 3. Waiting → Ready : I/O 작업이 끝난 경우
 4. Terminate

Scheduling Algorithm 구분

성능 척도

시스템 입장에서 성능 척도

- CPU 이용률 : 전체 시간 중 CPU가 쉬지 않고 일한 시간
- 처리량 : 단위 시간 당 수행 완료한 프로세스의 수

프로그램 입장에서 성능 척도

- 소요 시간 : 프로세스가 Ready queue에서 대기한 시간부터 작업을 완료하는데 걸리는 시간
- 대기 시간 : 프로세스가 Ready queue에서 대기한 시간
- 응답 시간 : 프로세스가 CPU를 요청한 후 첫 번째 실행을 시작하는 시간

CPU 이용률과 처리량이 높을수록, 소요/대기/응답 시간이 짧을 수록 좋다



Process Queue

Job Queue (Long-term scheduler)

- 프로세스가 시스템에 들어오면 Job Queue에 놓임
- RAM으로 올라올 프로그램을 정하는 Queue
- 하드 디스크에 있는 프로그램이 실행되기 위해 메인 메모리에 할당될 순서를 기다리는 Queue
- 디스크와 메모리 사이의 Scheduling을 담당
- 어떤 프로세스에 메모리를 할당해 Ready Queue로 보낼지 결정

Ready Queue (Short-term scheduler)

- CPU 점유 순서를 기다리는 Queue
- CPU와 메모리 사이의 Scheduling을 담당
- Ready Queue에 존재하는 프로세스 중 어떤 프로세스를 Running할지 결정

Device Queue

- I/O를 하기 위한 여러 장치가 있는데, 각 장치를 기다리는 Queue가 각각 존재

선점형 vs 비선점형 방식

Preemptive 방식

- 우선 순위가 높은 프로세스가 현재 프로세스를 중단시키고 CPU를 점유
- 우선순위가 높은 프로세스들이 계속 들어오는 경우 오버헤드 초래
- Round Robin , Multi-Level Queue , Multi-Level Feedback Queue 등의 방식이 있다

Non-Preemptive 방식

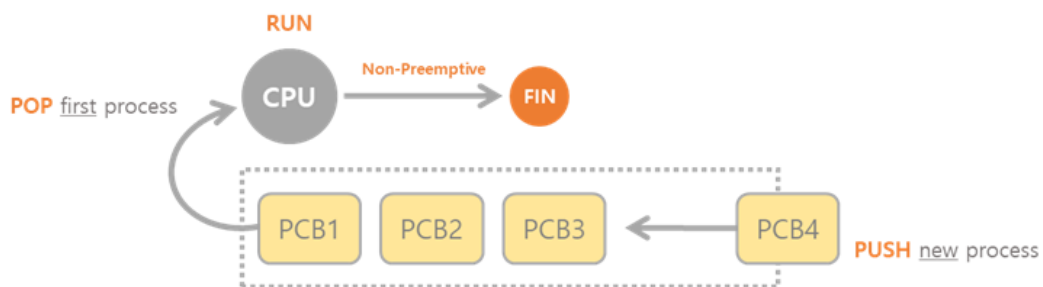
- 하나의 프로세스가 CPU를 할당받으면 작업 종료 후 CPU 반환 전까지 다른 프로세스가 CPU를 점유할 수 없음

- 모든 프로세스에 대한 요구를 공정하게 처리
- 짧은 작업을 수행하는 프로세스가 긴 작업이 종료될 때까지 대기하는 상황이 발생
- 전체 시스템 입장에서 낮은 처리율
- Priority, Shortest Job First, First In First Out, Highest Response Ratio Next 등의 스케줄링 방식이 있다



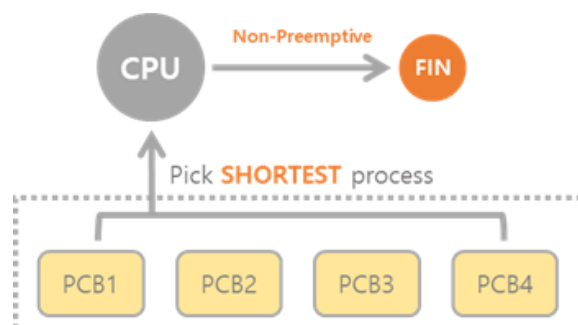
Scheduling Algorithm

FIFO(FCFS)



- First in First Out Sheduling
- 비선점 (Non-Preemptive) 방식
- 먼저 대기 큐에 들어온 작업에게 CPU를 먼저 할당
- 중요하지 않은 작업이 중요한 작업을 기다리게 할 수 있다
- 처리시간이 큰 프로세스가 CPU를 먼저 차지하면 전체 시스템 성능이 저하되는 Convey Effect가 발생할 수 있음

SJF

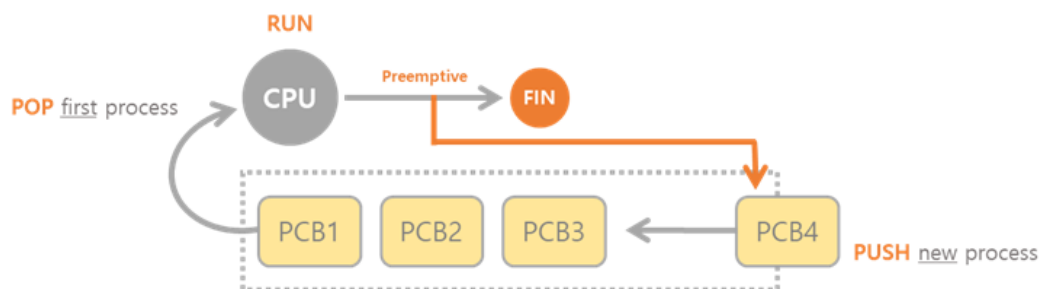


- Shortest Job First
- 비선점 (Non-Preemptive) 방식
- 실행시간이 짧은 프로세스 순서로 CPU를 할당
- 소요시간이 큰 프로세스는 계속 실행되지 않는 Starving 문제 발생

HRN

- Highest Response Ratio Next
- 비선점 (Non-Preemptive) 방식
- 기다린 시간 + CPU 사용시간으로 우선순위를 설정
 - $\text{우선순위} = (\text{대기시간} + \text{CPU 사용시간}) / \text{CPU 사용시간}$
- SJF의 Starving을 해결하기 위한 방법이나 여전히 공정성이 낮아 사용하지 않음

RR



- Round Robin Scheduling
- 선점 (Preemptive) 방식
- FIFO 스케줄링 기법을 Preemptive 기법으로 구현한 스케줄링
- 프로세스는 FIFO 형태로 대기 큐에 적재되지만, 주어진 시간 할당량(time slice) 안에 작업을 마쳐야 함
- 할당량을 다 소비하고도 작업이 끝나지 않은 프로세스는 다시 대기 큐의 맨 뒤로 돌아간다

SRT

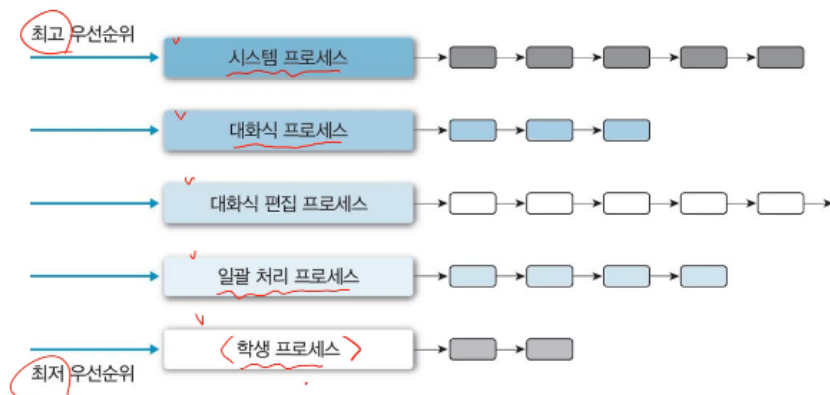
4개의 프로세스가 1분의 차이를 두고 연속적으로 입력되는 경우 가정



- Shortest Remaining Time First
- 선점 (Preemptive) 방식
- 남은 처리 시간이 더 짧다고 판단되는 프로세스가 Ready Queue에 생기면 언제라도 프로세스가 선점됨

MLQ

MLQ (Multi-level Queue)



Queue의 구성은 정책에 따라 결정

- Multi-Level Queue
- 작업 (or 우선순위) 별로 별도의 Ready Queue를 가짐

- 각각의 Queue는 자신만의 스케줄링 기법 사용
- 각각의 프로세스는 Queue간 이동이 허용되지 않음
- Queue 사이에는 우선순위 기반의 스케줄링 사용

MLFQ

- Multi-Level Feedback Queue
- 선점 (Preemptive) 방식
- 피드백에 따라 우선 순위를 조정하여 Queue간 이동을 허용
- 설계 및 구현이 복잡하고 스케줄링 부하가 크다
- 무한 대기 현상 존재 가능

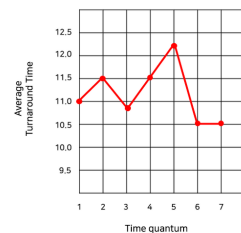
참고

▼ 링크

[운영체제(OS)] 5. 프로세스 스케줄링(Process Scheduling)

[목차] 1. CPU Scheduling 2. Scheduling Criteria 3. Scheduling Algorithm 4. Multiple-Processor Scheduling 참고) - <https://parksb.github.io/article/9.html> - KOCW 공개강의 (2014-


 <https://rebro.kr/175>



Process	Time
P1	6
P2	3
P3	1
P4	7


[운영체제] 프로세스 스케줄링

컴퓨터 하드웨어인 CPU는 여러개의 프로세스를 동시에 실행할 수 없다. 언제나 한순간에 오직 1개의 프로세스를 수행한다. 다만, 운영체제의 멀티태스킹(Multi tasking)과 스케줄링 기법으로 여

 <https://velog.io/@xxhaileypark/운영체제-프로세스-스케줄링>

T oday I L earn

[CS📖] 프로세스 스케줄링(선점, 비선점)

 프로세스 스케줄링이란 무엇일까? > 우리의 뇌는 동시에 두 가지 생각을 못한다 CPU 역시 동시에 프로세스를 관리하지 못한다 <https://velog.io/@dasssseul/CS-프로세스-스케줄링선점-비선점>

어디서든 쓸모있는
CS 지식 