04/02 스터디 노트

날짜	@2025년 4월 2일
태그	

1. CPU 스케줄링 알고리즘

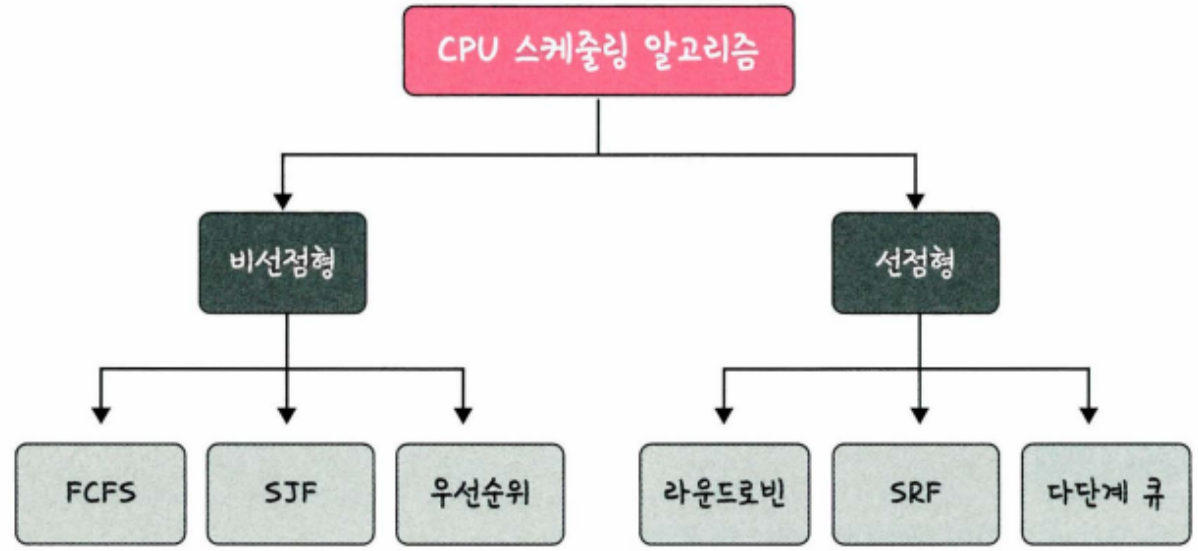
CPU 스케줄러

> 운영체제에서 CPU를 여러 프로세스에 효율적으로 할당하는 역할을 하는 핵심 구성 요소

- CPU 스케줄링 알고리즘에 따라 프로세스에서 해야 하는 일을 스레드 단위로 CPU에 할당
- 여러 프로세스가 동시에 실행될 수 없음
    - CPU 스케줄링 알고리즘을 기반으로 프로세스를 선택
    - 공정성과 성능을 고려하여 CPU 자원을 분배

CPU 스케줄링 알고리즘

▼ 그림 3-33 CPU 스케줄링 알고리즘



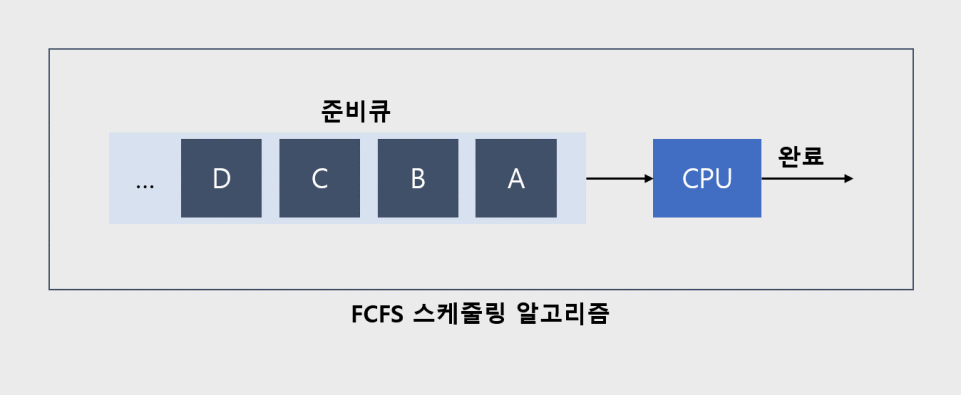
> 어떤 프로그램에 CPU 소유권을 줄 것인지 결정

- 목표 설정
  - CPU 이용률은 높게
  - 주어진 시간에 많은 일을 하게
  - 준비 큐에 있는 프로세스는 적게
  - 응답 시간은 짧게

1-1. 비선점형 방식

- 프로세스가 스스로 CPU 소유권을 포기하는 방식
- 강제로 프로세스를 중지하지 않음
  - 프로세스가 CPU를 할당 받으면 작업이 끝날 때까지 강제로 소유권이 빼앗기지 않음
  - 컨텍스트 스위칭으로 인한 부하가 적다.

1-1-1. FCFS



FCFS 스케줄링 알고리즘

First Come, First Served

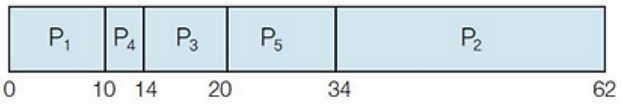
1. 개념
   - 가장 먼저 도착한 프로세스를 가장 먼저 처리하는 알고리즘
   - FIFO 방식과 유사
2. 장점
   - 공정성
     - 먼저 온 프로세스가 먼저 실행되므로 단순하고 공정함.
   - 대기 큐를 사용하여 구현이 간단하고, 관리하기 쉬움
   - 오버헤드 없음
     - Context Switching이 적음
3. 단점
   - convoy effect 발생 (준비 큐에서 대기 시간이 증가하는 현상)
     - 앞에서 길게 수행되는 프로세스가 있다면 뒤에 있는 짧은 프로세스들이 대기해야 함
     - 비효율적
   - 긴 평균 대기 시간
4. 예시

프로세스	도착 시간	실행 시간
P1	0	5
P2	1	3
P3	2	2

- 실행 순서: **P1 → P2 → P3**
  - P1: 0 time에 도착 → 5 time 동안 점유
  - P2: 1 time에 도착 → P1이 끝날 때까지 기다림 (5 - 1) → 3 time 동안 점유
  - P3: 2 time에 도착 → P1, P2가 끝날 때까지 기다림 (5 - 1 + 3 - 1) → 2 time 동안 점유
- 평균 대기 시간: **[(0) + (5 - 1) + (8 - 2)] / 3 = 3.33초**

1-1-2. SJF

프로세스	도착 시간	실행 시간
P_1	0	10
P_2	1	28
P_3	2	6
P_4	3	4
P_5	4	14



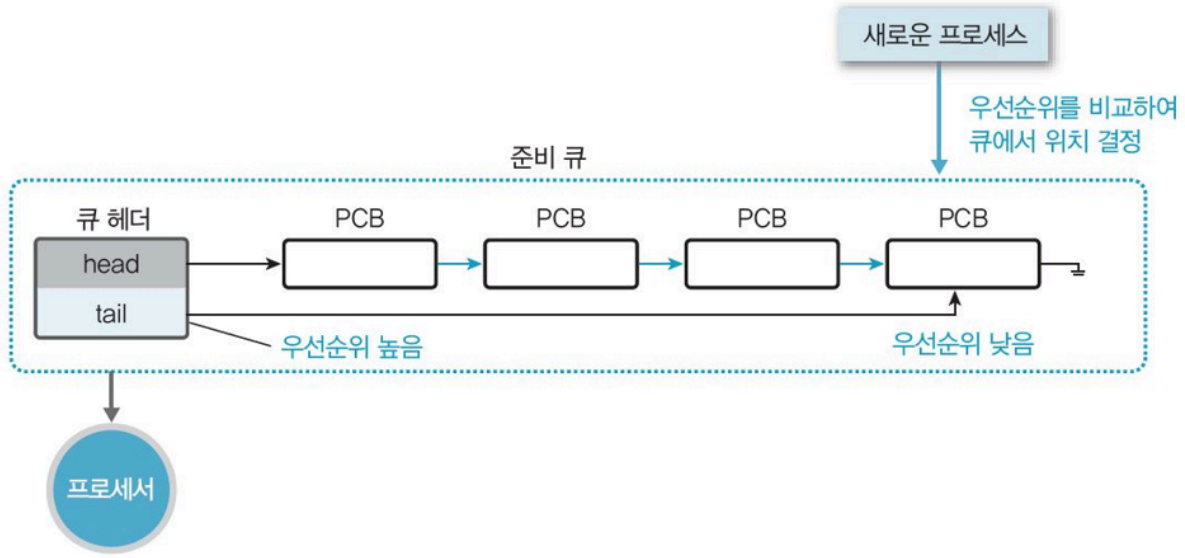
Shortest Job First

1. 개념
   - 실행 시간이 가장 짧은 프로세스를 가장 먼저 실행하는 알고리즘
2. 장점
   - 평균 대기 시간이 최소화됨
     - 짧은 프로세스를 우선 실행하여 전체적인 응답 속도가 향상됨
   - 처리량 증가
     - 짧은 작업을 먼저 처리하므로 많은 프로세스를 빠르게 실행할 수 있음
3. 단점
   - 기아 현상(Starvation) 발생
     - 실행 시간이 긴 프로세스가 계속해서 밀릴 수 있음 → 영원히 실행 X
   - 정확한 실행 시간 예측이 어려움
     - 프로세스의 실행 시간을 미리 알아야 하는데, 현실적으로 예측하기 어려움
4. 예시

프로세스	도착 시간	실행 시간
P1	0	6
P2	2	8
P3	4	7
P4	6	3

- 실행 순서: **P1 → P4 → P3 → P2** (실행 시간이 짧은 순서대로 실행)
- 평균 대기 시간 계산 후 최적의 값 도출 가능

1-1-3. 우선순위



1. 개념
   - 각 프로세스에 우선순위를 부여하고, 우선순위가 높은 프로세스를 먼저 실행
2. 장점
   - 긴급한 프로세스를 빠르게 실행 가능
     - 중요한 작업을 먼저 처리할 수 있음 ex. 실시간 시스템
   - 우선 순위를 이용한 유연한 스케줄링 가능
     - 시스템 자원 관리에 효과적
3. 단점
   - 기아 현상 발생 가능
     - 낮은 우선순위를 가진 프로세스가 계속 밀려 실행되지 않을 수 있음

        - 그래서 aging 기법을 사용해서 보완함. → 오래된 작업일수록 우선순위를 높이는 방법
    - 우선순위 결정이 어려움
        - 어떤 기준으로 우선순위를 설정할지 명확하지 않으면 비효율적인 스케줄링이 됨
    - 비효율적 자원 사용 가능성
        - 높은 우선순위 프로세스가 적을 경우 CPU가 유휴 상태(Idle)가 될 수 있음
4. 예시

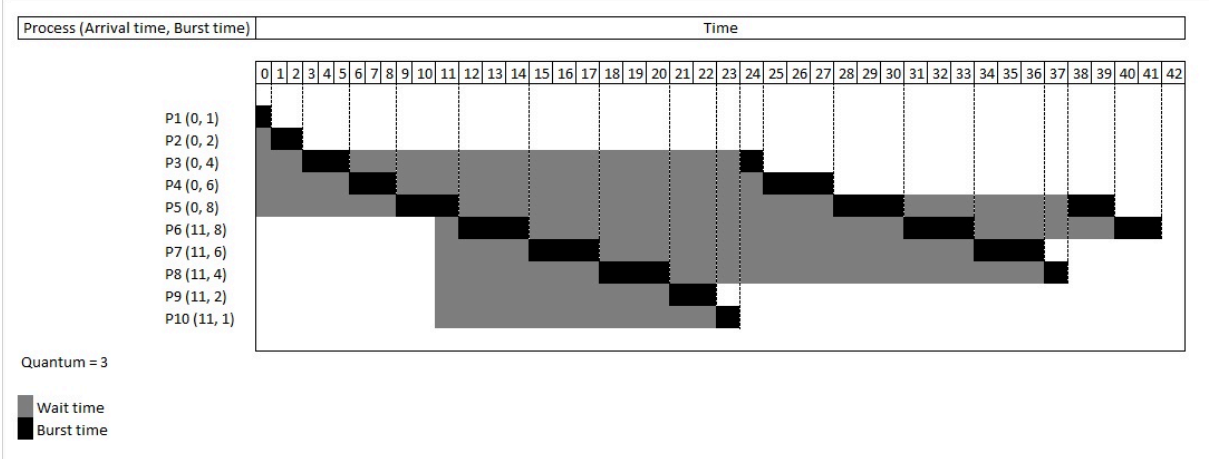
프로세스	도착 시간	실행 시간	우선순위(낮을수록 높음)
P1	0	5	3
P2	1	3	1
P3	2	4	4
P4	3	2	2

- 실행 순서: **P2 → P4 → P1 → P3** (우선순위 순으로 실행)

1-2. 선점형 방식

- 운영체제가 CPU를 강제로 회수하여 다른 프로세스에 할당하는 방식
- **응답성과 시스템 자원 활용도를 높이기 위해** 현대 운영체제에서 널리 사용됨
- 대표적인 예: 라운드 로빈(RR), 최단 남은 시간 우선(SRTF), 다단계 큐(Multilevel Queue)

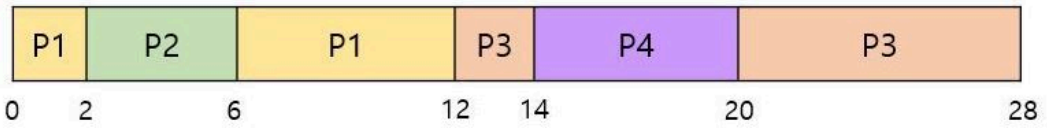
1-2-1. 라운드 로빈



1. 개념
    - 모든 프로세스에 동일한 시간 조각(Time Quantum, Time Slice)을 부여
    - 해당 시간 내에 작업이 끝나지 않으면 CPU를 반환하고 준비 큐의 뒤로 이동
    - 각 프로세스는 순환적으로 CPU를 할당받음 (선점형 FCFS와 유사)
2. 장점
    - 응답 시간이 짧아져 대화형 시스템(Interactive System)에 적합
    - **공정성 보장**: 모든 프로세스가 동일한 기회를 받음
    - CPU 점유가 짧은 작업은 빠르게 완료 가능
3. 단점
    - 타임 퀀텀 설정이 민감
        - 너무 크면 FCFS가 되고,
        - 너무 작으면 문맥 교환(Context Switching) 오버헤드가 커짐
            - 대기 시간이 누적되기 쉬움 → 많은 프로세스가 있을 때 효율이 저하됨
4. 예시
    - 프로세스: P1(도착:0, 실행:5), P2(도착:1, 실행:3), P3(도착:2, 실행:4)
    - 타임 퀀텀: 2
    - 실행 순서: `P1(2) → P2(2) → P3(2) → P1(3) → P2(1) → P3(2)`
    - 평균 대기 시간 감소, 짧은 작업(P2)은 비교적 빨리 끝남

1-2-2. SRF

✓ **SRT Scheduling**

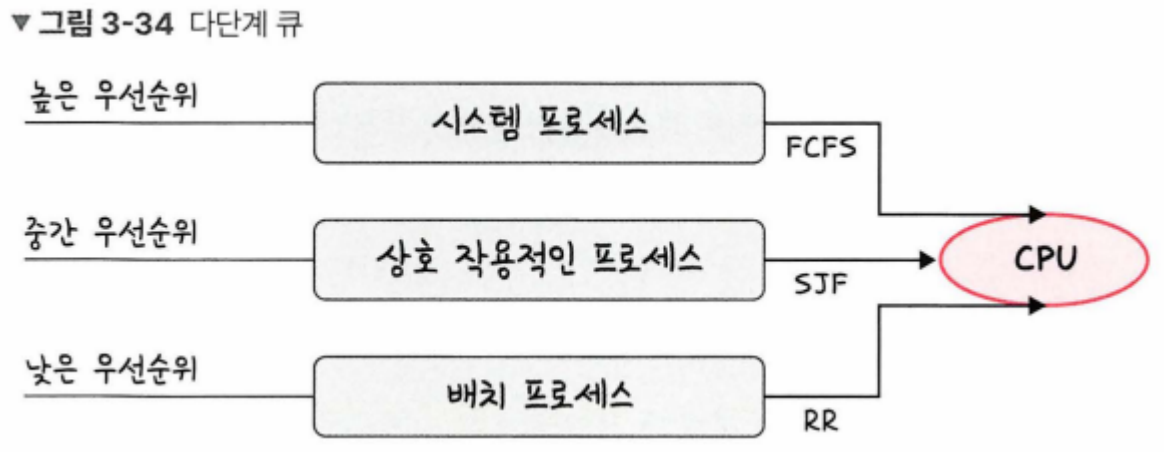


Process	Arrival Time	Burst Time	Waiting Time	Turnaround Time
P1	0	8	0+4	12
P2	2	4	0	4
P3	10	10	2+6	18
P4	14	6	0	6

- Average Waiting Time : (4+0+8+0)/4 = 3
- Average Turnaround Time : (12+4+18+6)/4 = 10

1. 개념
    - **SJF의 선점형 버전**으로, 가장 남은 실행 시간이 짧은 프로세스를 먼저 실행
    - 실행 중 더 짧은 작업이 도착하면 현재 작업을 중지하고 새로운 작업을 수행
    - 항상 **최소 남은 시간** 기준으로 프로세스를 선택
2. 장점
    - 평균 대기 시간과 평균 응답 시간이 모든 스케줄링 알고리즘 중 가장 낮음
    - 짧은 작업에 대해 매우 유리, 전체 처리 속도가 향상됨
3. 단점
    - 실행 시간 예측이 필요
    - 실제 운영체제에서는 작업의 남은 시간을 정확히 알기 어려움
        - 기아 현상 발생 가능
    - 긴 작업이 계속해서 짧은 작업에 밀려 실행되지 않을 수 있음
4. 예시
    - 프로세스: P1(도착:0, 실행:8), P2(도착:1, 실행:4), P3(도착:2, 실행:2), P4(도착:3, 실행:1)
    - 실행 순서: `P1(1) → P2(0~1) → P3(0~2) → P4(0~1) → P3(1~2) → P2(나머지) → P1(남은 시간)`
    - 도중에 더 짧은 작업이 들어오면 선점 발생

1-2-3. 다단계 큐



1. 개념
    - 프로세스를 **성격에 따라 여러 개의 큐로 분류**하고 각 큐에 다른 스케줄링 정책 적용
    - 예: 시스템 프로세스, 사용자 프로세스, 대화형/일괄 작업 프로세스 등으로 분리
    - 큐 간에는 **우선순위 기반**으로 스케줄링됨 (상위 큐가 먼저 실행됨)
2. 장점
    - 다양한 유형의 작업을 효율적으로 처리 가능
    - 각 큐에서 최적의 스케줄링 방식을 선택할 수 있어 유연함
    - 큐 간 이동이 없다면 스케줄링 오버헤드가 적음
3. 단점
    - 프로세스가 큐를 잘못 배정 받으면 성능 저하가 일어남
    - 기아 현상: 하위 큐의 작업이 상위 큐에 밀려 오랫동안 실행되지 않을 수 있음
    - 큐 간 이동이 불가능하면 적응성(유연성) 부족
4. 예시
    - 큐 1: 실시간 작업 (라운드 로빈, 타임 퀀텀 5ms)
    - 큐 2: 대화형 사용자 작업 (라운드 로빈, 타임 퀀텀 10ms)
    - 큐 3: 백그라운드 배치 작업 (FCFS)

      → 실시간 작업이 모두 끝나야 그 다음 큐의 작업이 실행됨