

프로세서 스케줄링



8th Week

Kim, Eui-Jik

Contents

- 소개
- 스케줄링 수준
- 선점형/비선점형 스케줄링
- 우선순위
- 스케줄링 목적
- 스케줄링 알고리즘

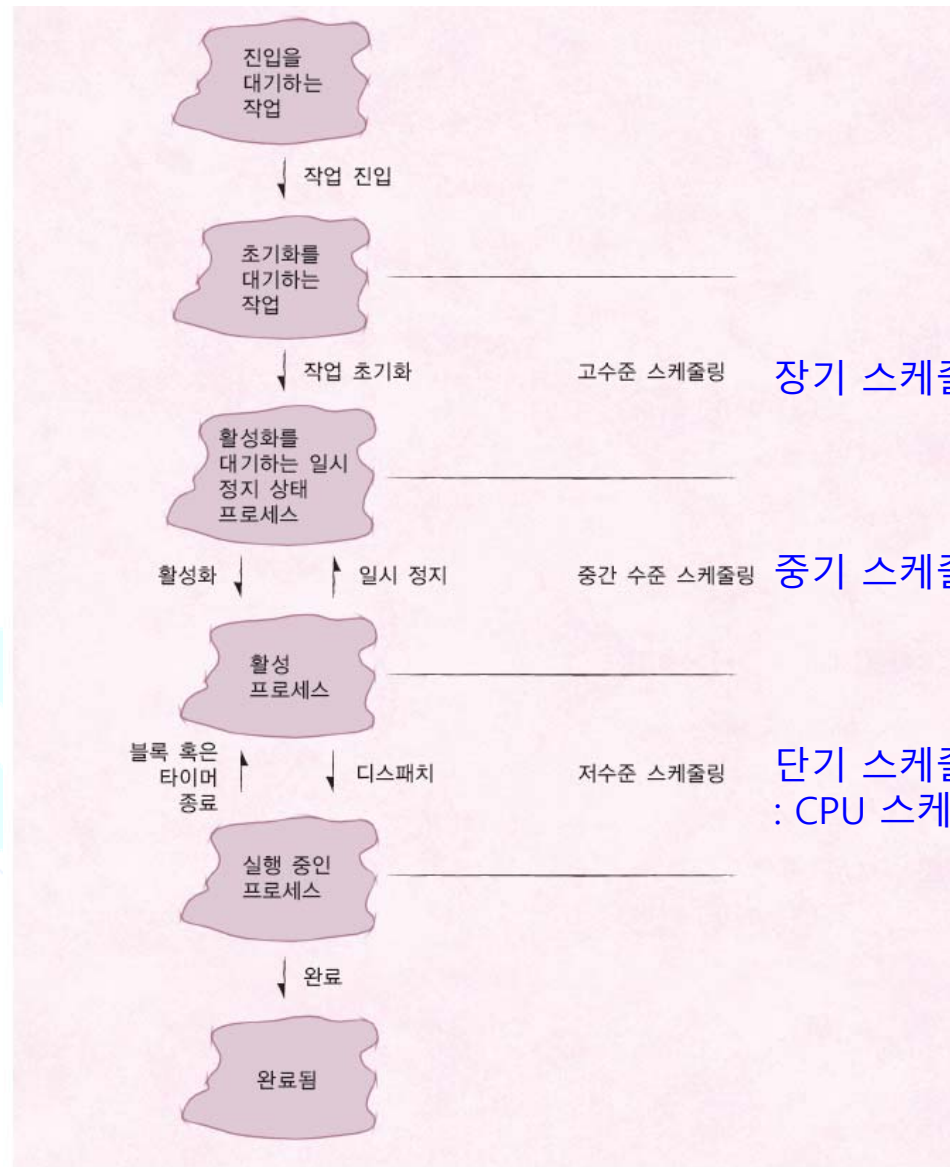
소개

- 프로세서 스케줄링 정책
 - 주어진 시간에 시스템이 실행할 프로세스를 선택하는 작업
 - 여러 성능 요건의 충족이 필요
 - 처리량 최대화
 - 지연시간 최소화
 - 무기한 연기 방지
 - 명시된 데드라인 전에 작업 완료
 - 프로세서 활용도 극대화
- 프로세서 스케줄링
 - 프로세스들이 작업을 수행하기 위해서는 CPU 스케줄러에 의해서 CPU를 할당 받아야 한다.
 - 이러한 할당 작업은 운영체제에 의해서 수행되며, CPU가 언제, 어떤 프로세스에게 할당되는지 결정하는 작업을 프로세서 스케줄링 또는 CPU 스케줄링이라고 한다.

스케줄링 수준(단계)

- 고수준 스케줄링
 - Long-term job scheduling
 - 작업 스케줄링, 장기 스케줄링, 승인 스케줄링(admission scheduling)
 - 시스템이 어떤 작업에 자원을 얻으려고 경쟁하게 해줄지 결정
 - 임의의 시간에 시스템에 존재하는 프로세스의 총 수를 지정
- 중간수준 스케줄링
 - Mid-term swapping
 - Swapping: 물리 메모리가 부족한 경우, 시스템 전체적인 성능을 향상 시키기 위해서 특정한 프로세스가 사용하는 메모리를 반납하고, 프로세스가 사용했던 메모리 내용을 디스크로 이주시키는 것을 의미함
 - 어떤 프로세스가 프로세서를 얻으려고 경쟁할 수 있는지 결정(활성화)
 - 단기적인 시스템 부하의 높낮이에 따라 결정
- 저수준 스케줄링
 - Short-term CPU scheduling
 - 활성화된 프로세스 중 어떤 프로세스에 프로세서를 할당할지 결정
 - 우선순위 부여

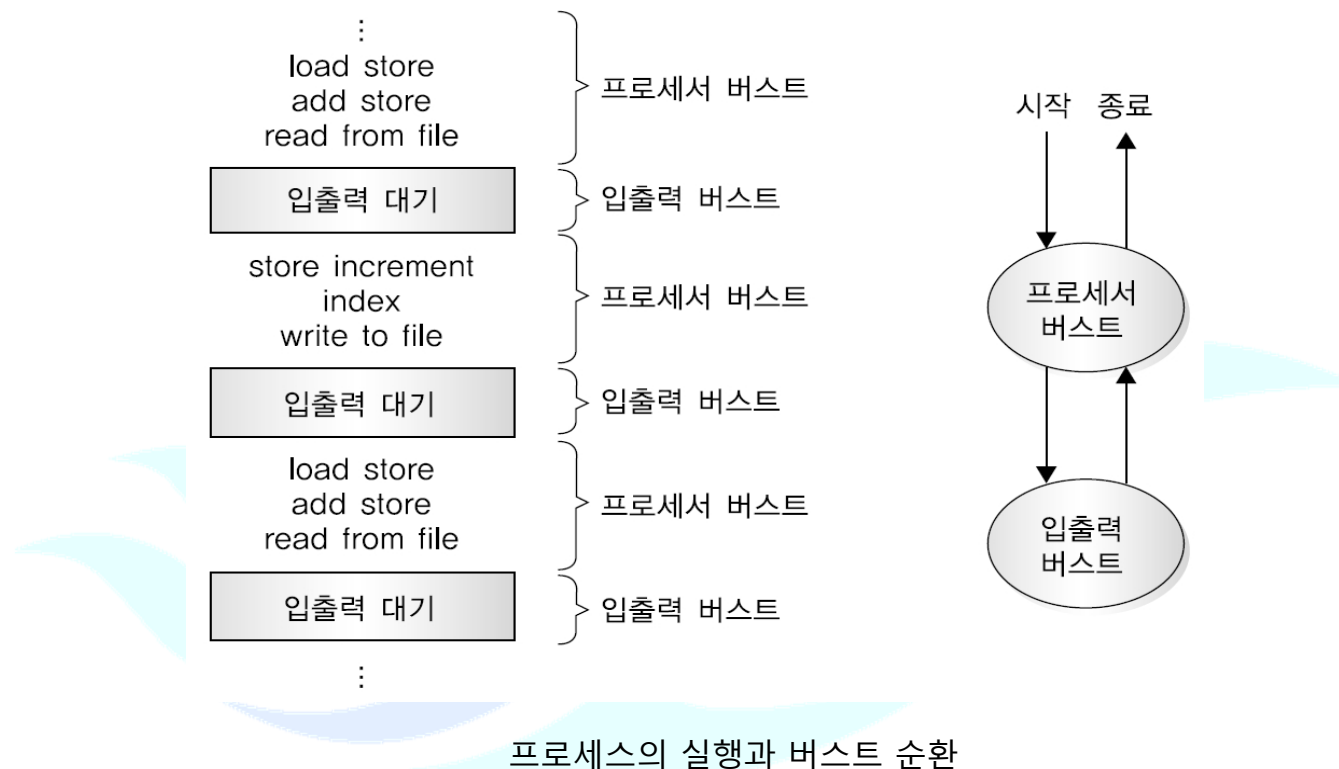
스케줄링 수준(단계)



프로세스 실행

- 다중 프로그래밍 시스템에서 입출력 작업을 수행하는 동안 프로세스가 대기 상태에 있을 때 다른 프로세스에게 CPU를 할당해줌으로써 컴퓨터 자원을 최대로 활용
- 프로세스의 실행은 실행(프로세서 버스트; CPU burst)과 입출력 대기(입출력 버스트; I/O burst)의 순환으로 구성
 - 프로세스들은 이들 두 상태 사이에서 교대로 진행
 - 프로세서 버스트, 즉 프로세서에서 실행되고 있는 프로세스의 상태로 시작하고 끝남
 - 입출력 중심의 프로세스(I/O-bound process)는 짧은 CPU burst가 많이 있으며, CPU 중심의 프로세스(CPU-bound process)는 다수의 긴 CPU burst를 가짐
 - 입출력 중심의 프로세스와 CPU 중심의 프로세스의 적절한 혼합은 좋은 성능을 낼 수 있게 함

프로세스 실행



선점형/비선점형 스케줄링

■ 선점형 스케줄링

- 프로세서에서 프로세스가 실행 중일 때 시스템이 프로세서의 쟁취가 가능
- 우선순위가 높은 프로세스에 대한 응답 시간 향상
- 대화식 시분할 시스템에서 반응 시간 보장
- 여러 프로세스를 메인 메모리에 유지하는 것이 필요

■ 비선점형 스케줄링

- 프로세스가 프로세서를 할당 받으면 작업을 완료하거나 자발적으로 프로세서를 반납할 때까지 프로세서에서 실행 가능
- 중요한 프로세스라도 앞의 프로세스가 완료될 때까지 대기

우선순위

- 정적 우선순위
 - 처음에 정해진 우선순위가 고정
 - 쉬운 구현
 - 낮은 오버헤드
 - 환경 변화에 대한 반응이 불가능
- 동적 우선순위
 - 상황에 따라 우선순위의 변화
 - 복잡한 구현
 - 높은 오버헤드
 - 시스템의 반응성 향상

스케줄링 목적

- 시스템 유형에 따라 다른 목적
 - 처리량 최대화
 - '수용할 만한' 시간 안에 반응을 보이는 대화식 프로세스의 수 최대화
 - 자원활용도 최대화
 - 무기한 연기 회피
 - 오버헤드 최소화
 - 예측 가능성 보장
- 대부분의 스케줄러가 가지는 공통된 목적
 - 공정성
 - 예측 가능성
 - 규모 확장성

스케줄링 알고리즘

- 스케줄링 알고리즘
 - 각 프로세스가 언제, 얼마 동안 실행할지 결정
 - 여러 요인에 대한 고려
 - 선점/비선점 여부
 - 우선순위
 - 실행 시간
 - 완료 시까지 남은 시간
 - 공정성(모든 프로세스들은 공정하게 스케줄링 되어야 하며 무한정 스케줄링이 지연되는 일이 없어야 함)

스케줄링 알고리즘

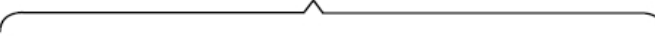
■ 알고리즘 성능 평가 기준

- 스케줄링 알고리즘 선택 시, 각 알고리즘의 특성을 고려해야 함
- 프로세서 사용률 (Max CPU utilization)
 - 프로세서를 실행상태로 항상 유지하여 유휴상태가 되지 않도록 함
- 처리율 (Max throughput)
 - 단위 시간당 완료되는 작업 수가 많도록 짧은 작업을 우선 처리하거나 인터럽트 없이 작업을 실행
- 반환시간 (Min turnaround time)
 - 작업이 시스템에 맡겨져 메인메모리에 들어가기까지의 시간, 준비 큐에 있는 시간, 실행시간, 입출력 시간 등 작업 제출 후 완료되는 순간까지의 소요시간이 최소화되도록 일괄처리 작업을 우선 처리함
- 대기시간 (Min waiting time)
 - 작업의 실행시간이나 입출력시간에는 실제적인 영향을 미치지 못하므로 준비 큐에서 기다리는 시간이 최소화되도록 사용자 수를 제한함
- 반응시간 (Min response time)
 - 의뢰한 시간에서부터 반응이 시작되는 시간까지의 간격
 - 대화형 시스템에 중요한 사항으로, 대화식 작업을 우선 처리하고 일괄처리 작업은 대화식 작업의 요구가 없을 때 처리함

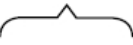
※ 스케줄링 시 프로세서 사용률과 처리율을 최대화하고, 반환시간, 대기시간, 반응시간은 최소화하는 것이 바람직함

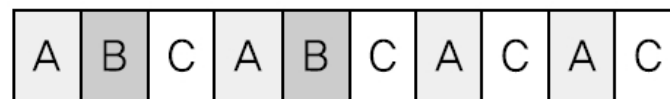
스케줄링 알고리즘

- 프로세스의 반환, 대기, 반응 시간
 - 프로세스 A, B, C가 A, B, C 순으로 같은 시간에 도착했다 가정함

반환시간 

대기시간 

반응시간 

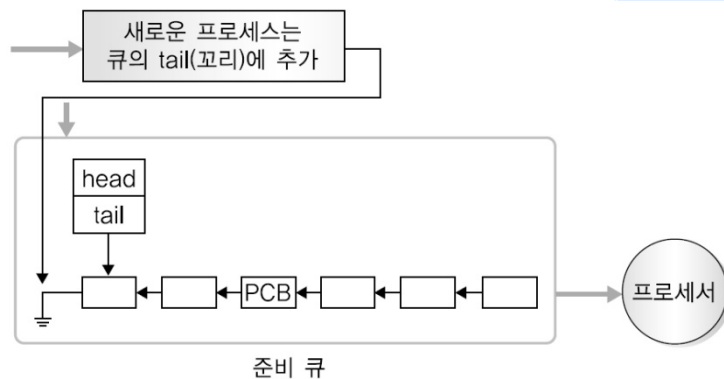


시간

프로세스 C의 반환시간, 대기시간, 반응시간

(1) 선입 선처리 스케줄링 (FCFS, First-Come-First-Served)

- 선입 선처리 스케줄링 (FCFS, First-Come-First-Served)
 - 프로세서를 요구 순서대로 할당
 - 비선점 기법으로 프로세서 스케줄링 알고리즘 중 가장 간단함
 - 선입선출 (FIFO, First-In First-Out) 큐로 구현
 - 일괄처리 시스템에서는 효율적이나, 대화식 시스템에서는 사용자의 빠른 응답 요구에 적합하지 않음



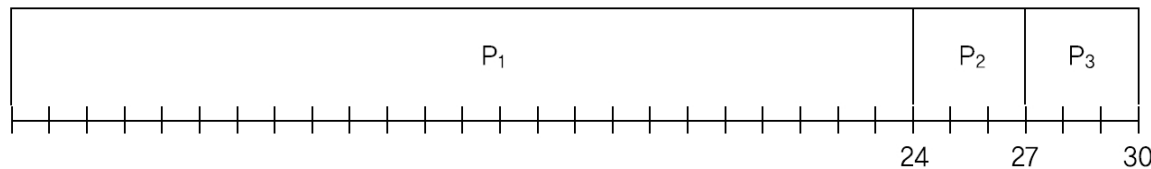
선입 선처리 스케줄링

- 새로운 작업이 시스템에 들어오면 프로세스의 PCB는 준비 큐의 마지막에 연결됨
- 차례가 되면 준비 큐의 앞부분에 있는 프로세스는 프로세서를 할당 받고 준비 큐에서 삭제
- 성능이 좋지 않는 경우가 많으며 평균 대기시간이 긴 경우도 있음

(1) 선입 선처리 스케줄링 (FCFS, First-Come-First-Served)

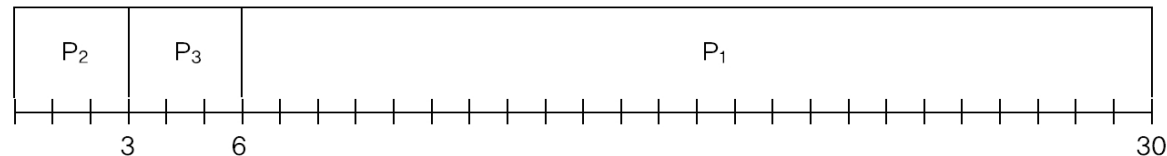
- 작업의 평균 변환 시간 계산
 - 프로세서 버스트 시간을 알고 있는 다음 세 가지 작업의 평균 변환 시간을 계산함
 - 작업이 P_1, P_2, P_3 의 순서로 들어왔다면, 다음과 같은 간트 도표(Gantt Chart)로 나타냄

프로세스	실행시간
P_1	24
P_2	3
P_3	3



P_1, P_2, P_3 의 순서로 들어왔을 경우의 간트 도표

- 반환시간은 P_1 이 24, P_2 가 27, P_3 가 30이므로, 평균 반환시간은 $27 = (24 + 27 + 30) / 3$
- 대기시간은 P_1 이 0, P_2 가 24, P_3 가 27이므로, 평균 대기시간은 $17 = (0 + 24 + 27) / 3$



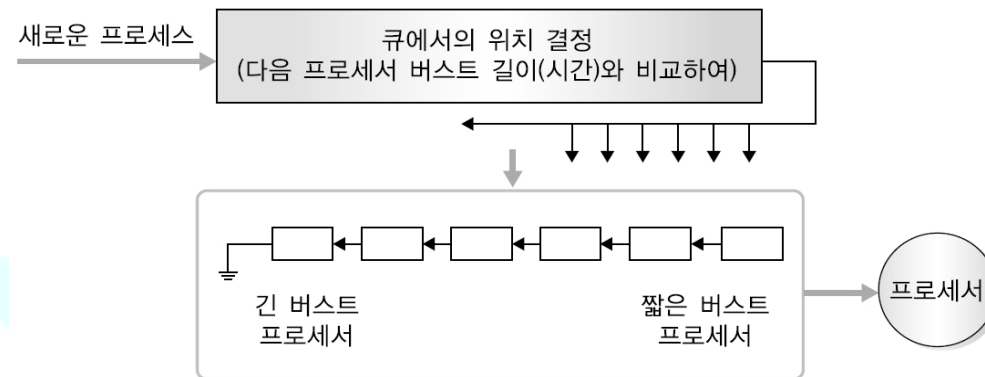
P_2, P_3, P_1 의 순서로 들어왔을 경우의 간트 도표

- 평균 반환시간은 $13 = (3 + 6 + 30) / 3$ 이 되고, 평균 대기시간은 $3 = (0 + 3 + 6) / 3$ 으로 단축됨

※ 프로세스의 실행시간에 따라 평균 반환시간과 평균 대기시간은 큰 폭으로 변화함

(2) 최소 작업 우선 스케줄링 (SJF, Shortest-Job-First)

- 최소 작업 우선 스케줄링(SJF, Shortest Job First)
 - 프로세서 버스트 시간이 가장 짧은 작업에 프로세서를 할당
 - 이때 두 프로세스가 다음 순서로 동일한 프로세서 버스트를 가진다면 선입 선처리 스케줄링을 적용함



최소 작업 우선 스케줄링

- 다음과 같은 작업들이 준비상태 큐에 있을 때, 최소 작업 우선 스케줄링을 사용하면 그림과 같은 간트 도표가 이루어지며, 평균 반환시간은 $13 = (3 + 9 + 16 + 24) / 4$

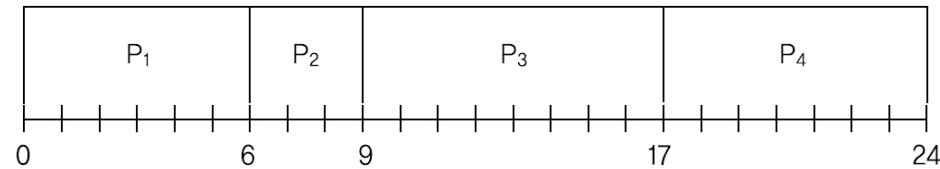
프로세스	실행시간
P ₁	6
P ₂	3
P ₃	8
P ₄	7



최소 작업 우선 스케줄링을 사용한 간트 도표

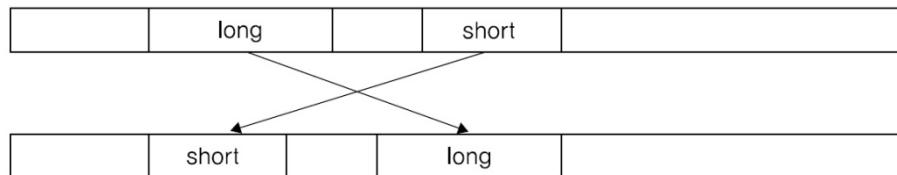
(2) 최소 작업 우선 스케줄링 (SJF, Shortest-Job-First)

- 대기시간은 P2가 0, P1은 3, P4는 9, P3은 16이므로 평균 대기시간은 $7(=(0+3+9+16)/4)$
- 선입 선처리 스케줄링(FCFS) 사용 시 간트 도표



선입 선처리 스케줄링을 사용한 간트 도표

- 평균 반환시간은 $14(=(6+9+17+24)/4)$ 이며, 평균 대기시간은 $8(=(0+6+9+17)/4)$
- 최소 작업 우선 알고리즘은 주어진 작업 집합에 대해 평균 대기시간이 최소이므로 최적 알고리즘
 - [그림 a] 실행시간이 짧은 작업(Short)을 실행시간이 긴 작업(Long) 앞에 놓음
 - 긴 작업의 대기시간은 증가하였으나, 짧은 작업의 대기시간이 줄어 평균 대기시간은 감소
 - [그림 b] 프로세스 실행시간이 각각 5, 11, 7, 3인 경우



[그림 a] 짧은 작업 우선 처리

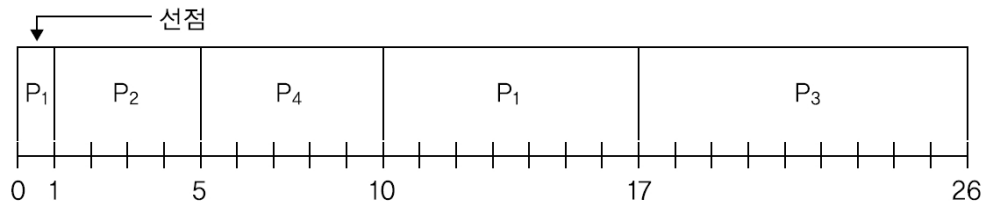
5	11	7	3	$W = 0 + 5 + 16 + 23 = 44$
5	7	11	3	$W = 0 + 5 + 12 + 23 = 40$
5	7	3	11	$W = 0 + 5 + 12 + 15 = 32$
5	3	7	11	$W = 0 + 5 + 8 + 15 = 28$
3	5	7	11	$W = 0 + 3 + 8 + 15 = 26$

[그림 b] 최소 작업 우선 스케줄링이 최적임을 증명

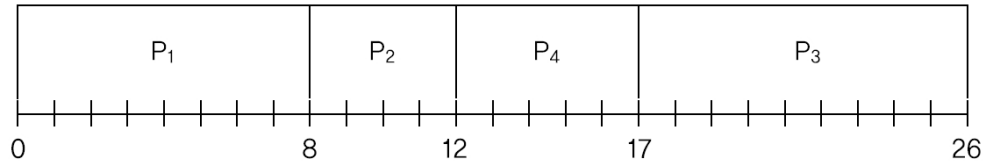
(2) 최소 작업 우선 스케줄링 (SJF, Shortest-Job-First)

- 최소 작업 우선 스케줄링(SJF)은 선점 또는 비선점이 가능
 - 아래의 간트 도표는 선점과 비선점 최소 작업 우선 스케줄링을 나타냄

(a) 선점



(b) 비선점



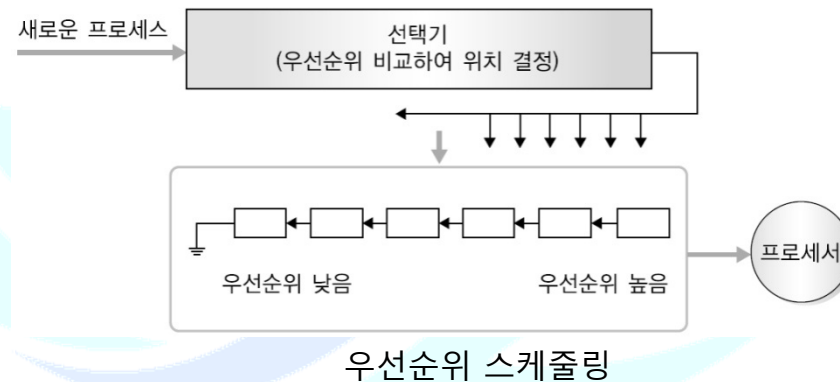
선점과 비선점 최소 작업 우선 스케줄링을 사용한 간트 도표

프로세스	도착시간	실행시간
P ₁	0	8
P ₂	1	4
P ₃	2	9
P ₄	3	5

- 선점인 경우 : 최소 잔여시간 우선 스케줄링(SRT: Shortest-Remaining-Time)
 - 평균 반환시간은 $13 = (P_1 + P_2 + P_3 + P_4) / 4 = ((17-0) + (5-1) + (26-2) + (10-3)) / 4 = 52/4$
 - 평균 대기시간은 $6.5 = (P_1 + P_2 + P_3 + P_4) / 4 = (10-1) + (1-1) + (17-2) + (5-3)) / 4$
 - 시분할 시스템에 유용, 문맥교환 시간이 소요됨
(빈번한 선점이 발생하는 경우, 오버헤드 증가)
- 비선점인 경우
 - 평균 반환시간은 $14.25 = (P_1 + P_2 + P_3 + P_4) / 4 = (8-0) + (12-1) + (26-2) + (17-3)) / 4 = 57/4$
 - 평균 대기시간은 $7.75 = (P_1 + P_2 + P_3 + P_4) / 4 = (0 + (8-1) + (17-2) + (12-3)) / 4$

(3) 우선순위 스케줄링(Priority Scheduling)

- 우선순위 스케줄링(Priority Scheduling)
 - 각 프로세스마다 고유한 우선 순위 할당
 - 준비 큐에 도착한 프로세스와 현재 실행 중인 프로세스의 우선순위를 비교
 - 프로세스들의 우선순위를 비교하여 최고 우선순위를 가진 프로세스에 프로세서를 할당
 - 우선순위가 같은 프로세스들은 선입 선처리 순으로 스케줄링됨

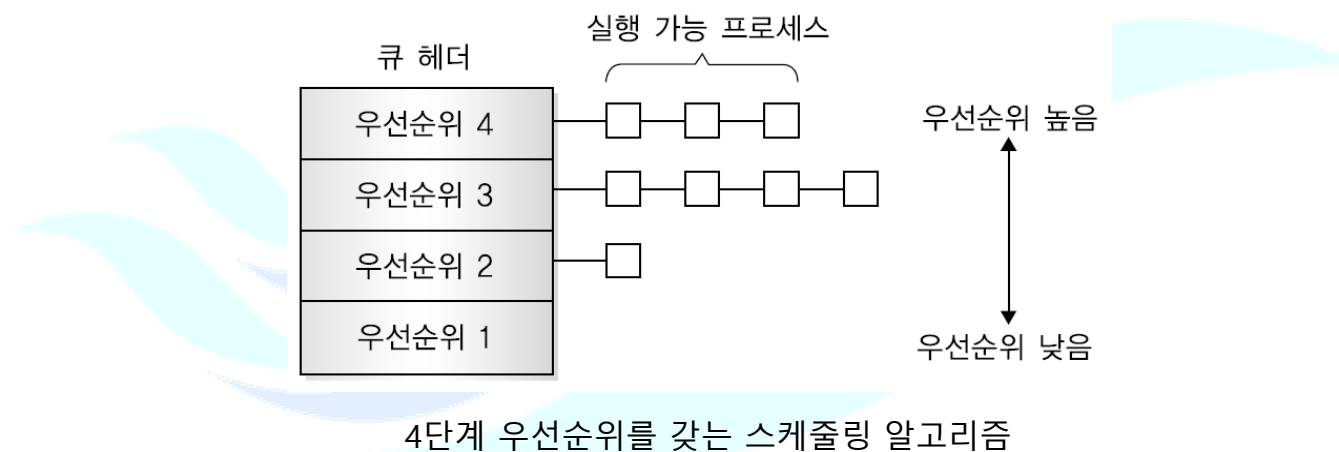


- 최소 작업 우선 알고리즘은 우선순위 알고리즘에 속함
 - 우선순위(P)는 예측된 다음 프로세서 버스트의 역(τ), 즉 ' $P = 1/\tau$ '
 - 프로세서 버스트가 클수록 우선순위가 낮으며, 그 반대로 성립됨

(3) 우선순위 스케줄링(Priority Scheduling)

■ 선점 또는 비선점이 가능함

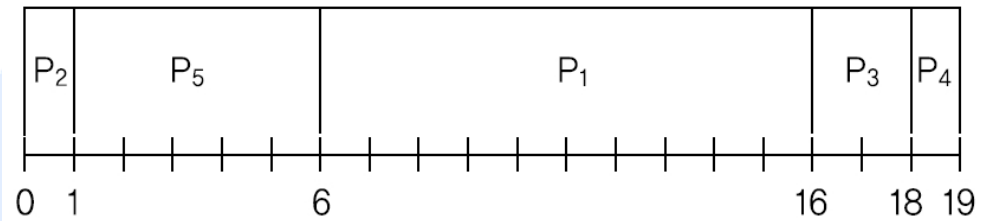
- 선점 우선순위 스케줄링 알고리즘: 새로 도착한 프로세스의 우선순위가 현재 실행되는 프로세스의 우선순위보다 높으면 프로세서를 선점함
- 비선점 우선순위 스케줄링 알고리즘: 단순히 준비 큐의 머리 부분에 새로운 프로세스를 넣음



(3) 우선순위 스케줄링(Priority Scheduling)

- P1~P5 프로세스들이 같은 시간에 도착했다 가정함.
- 우선순위 스케줄링을 이용하여 스케줄링한 결과는 아래의 간트 도표와 같으며, 평균 대기 시간은 8.2임

프로세스	실행시간	우선순위
P ₁	10	3
P ₂	1	1
P ₃	2	3
P ₄	1	4
P ₅	5	2



우선순위 스케줄링을 이용한 간트 도표

- 문제점
 - 주요 문제는 무한정지와 기아
 - 우선 순위가 높은 프로세스들이 계속 들어오면 우선순위가 낮은 프로세스들은 무한정 기다려야 함
- 해결 방법
 - 에이징(Aging)
 - 오랫동안 시스템에서 대기하는 프로세스들의 우선순위를 점진적으로 증가시키는 기법

(4) 순환 할당(RR, Round-Robin) 스케줄링

■ 순환 할당(Round-Robin) 스케줄링

■ 시분할 시스템에 효과적

- **규정 시간량(Time Quantum)** 또는 **시간 할당량(Time Slice)**이라 하는 작은 단위의 시간을 정의
- 각 프로세스에게 일정한 타임 쿼텀(time quantum)을 할당, 프로세스는 주어진 시간 간격만큼 수행한 후 프로세서를 양도함
- 시간 할당량은 일반적으로 10 ~100 msec
- 준비 큐는 순환 큐(**Circular Queue**)로 설계, 프로세서 스케줄러가 준비 큐를 돌아가면서 한 번에 한 프로세스에 정의된 규정 시간 만큼 프로세서를 제공
- 준비 큐는 **FIFO** 큐임

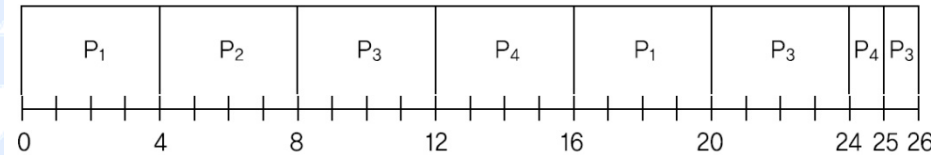


순환 할당 스케줄링

(4) 순환 할당(RR, Round-Robin) 스케줄링

- 타임 쿼텀이 극단적으로 큰 경우는 FIFO 스케줄링 기법과 동일함
- 타임 쿼텀이 작은 경우는 문맥전환의 빈번하게 발생하므로 오버헤드가 커짐
- [Example 1] 규정 시간량을 4로 한 경우 순환 할당 스케줄링의 간트 도표는 [그림a]와 같음

프로세스	도착시간	실행시간
P ₁	0	8
P ₂	1	4
P ₃	2	9
P ₄	3	5



[그림a] 규정 시간량을 4로 한 순환 할당 스케줄링의 간트 도표

- P1은 처음 4만큼의 시간을 실행한 후 아직 4가 남아 있지만 P2가 프로세서를 선점, 실행됨
- P2는 시간을 4만큼만 원하므로 4시간 후 종료되며 P3이 들어옴
- P3은 4시간을 실행한 후 P4에 프로세서를 넘겨줌
- 이후 앞서 종료되지 않은 P1이 실행됨

(4) 순환 할당(RR, Round-Robin) 스케줄링

- [Example 2] 타임 쿼텀(Time Quantum) = 20 일 때, RR 스케줄링

<u>Process</u>	<u>Burst Time</u>
----------------	-------------------

P_1	53
-------	----

P_2	17
-------	----

P_3	68
-------	----

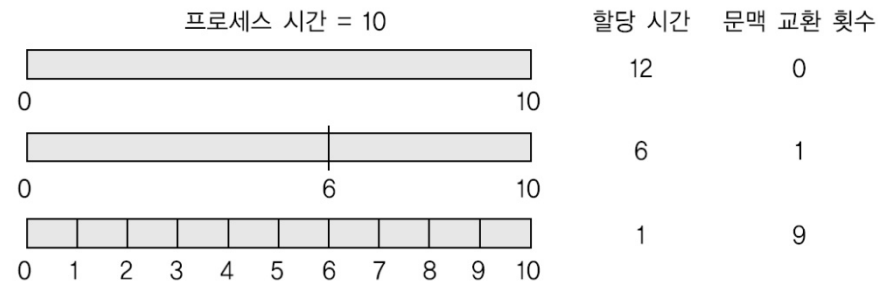
P_4	24
-------	----

P ₁	P ₂	P ₃	P ₄	P ₁	P ₃	P ₄	P ₁	P ₃	P ₃
----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------

0	20	37	57	77	97	117	121	134	154	162
---	----	----	----	----	----	-----	-----	-----	-----	-----

(4) 순환 할당(RR, Round-Robin) 스케줄링

- 타임 쿼텀(Time Quantum)의 크기에 따른 문맥 전환 관계

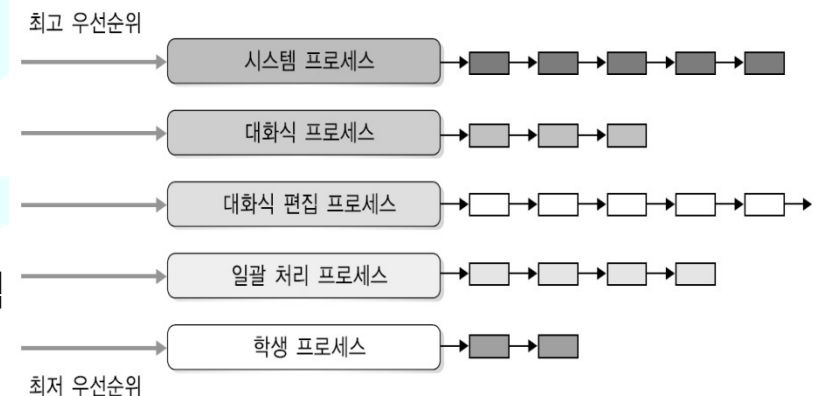


문맥교환 횟수를 증가시키는 작은 시간 할당량

- 쿼텀이 짧은 경우 문맥전환의 오버헤드로 인하여 처리율(throughput)이 떨어짐
- 문맥전환 오버헤드가 3 msec인 경우
 - 짧은 쿼텀 사용 시, Time Quantum = 15 msec:
 $\text{overhead} = 3 / (3 + 15) = 16.67 \%$
 - 긴 쿼텀 사용 시, Time Quantum = 97 msec:
 $\text{overhead} = 3 / (3 + 97) = 3 \%$

(5) 다단계 큐(MLQ, Multi-Level Queue) 스케줄링

- 다단계 큐(Multi-Level Queue) 스케줄링
 - 프로세스는 속성에 따라서 다양한 그룹으로 나뉘어짐(interactive, batch, etc.)
 - 준비 큐(ready queue)를 몇 개의 큐로 분할 함
 - 각 큐는 자신만의 독자적인 스케줄링 알고리즘을 가짐
 - 큐 사이: 고정된 우선순위의 선점식 스케줄링
 - 각 큐는 우선 순위가 다르며, 높은 우선 순위의 큐에 작업이 있는 경우 해당 큐의 프로세스가 먼저 수행
 - 한 큐에서 다른 큐로 작업 이동이 불가능
 - 큐 5개를 가진 다단계 큐 스케줄링 알고리즘
 - 각 큐는 순서대로 절대적인 우선순위를 가짐
 - 앞의 세 가지 큐가 비어있어야, 네번째 일괄처리 큐에 있는 프로세스가 실행됨

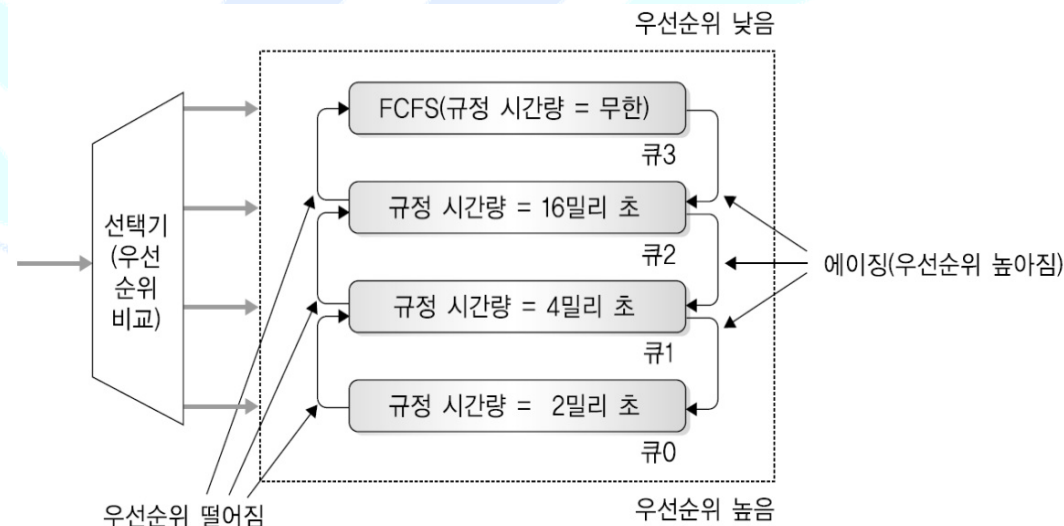


다단계 큐 스케줄링

(6) 다단계 피드백 큐(MLFQ, Multi Level Feedback Queue) 스케줄링

■ 다단계 피드백 큐 스케줄링

- 새로운 프로세스가 들어오면 높은 우선 순위를 할당해주어 처리가 될 수 있게 하고, 처리가 완료되지 못했을 때에 점차 낮은 우선 순위 큐로 이동하여 작업이 처리 될 수 있게 함
- 프로세스가 큐 사이를 이동 할 수 있게 변형
- CPU를 오래 사용한 프로세스는 낮은 우선 순위 큐로 이동
- 프로세스가 보다 하위 단계의 큐로 이동할수록 주어진 할당 시간은 점차 크게 설정함
- 에이징(aging)기법이 도입되기도 함
 - 낮은 큐에 머물러 있는 프로세스의 기아 방지를 위해서 일정한 시간마다 낮은 큐의 프로세스를 높은 우선 순위 큐로 이동시키는 방법



(7) 최고 응답률 우선 스케줄링 (HRN, Highest Response-Rate Next)

■ HRN(Highest Response-Rate Next) 스케줄링

■ 한슨(Brinch Hansen)이 개발

- 최소 작업 우선(SJF) 기법의 약점이었던 긴 작업과 짧은 작업 간의 지나친 불평등을 어느 정도 보완한 기법
- 비선점 스케줄링 기법으로 각 작업의 우선순위는 작업이 서비스 받을 시간 뿐만 아니라 서비스를 기다린 시간, 두 가지의 함수로 나타냄
- 한 작업이 프로세서를 차지하면 작업이 종료될 때까지 실행됨
- HRN 기법의 가변적 우선순위는 다음 식에 따라 계산됨

$$\text{우선순위} = \frac{\text{대기한 시간} + \text{서비스를 받을 시간}}{\text{서비스를 받을 시간}}$$

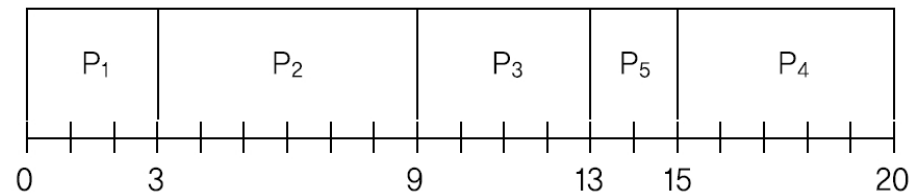
- '서비스를 받을 시간'이 분모에 있으므로
'서비스를 받을 시간'이 짧은 작업이 우선순위가 높음
- '대기한 시간'이 분자에 있으므로 '서비스를 받을 시간'이 긴 작업도
'대기한 시간'이 큰 경우에 우선순위가 높아짐
- 따라서, 시스템의 응답시간은 다음과 같이 나타낼 수 있음

$$\text{시스템의 응답시간} : \text{대기한 시간} + \text{서비스를 받을 시간}$$

(7) 최고 응답률 우선 스케줄링 (HRN, Highest Response-Rate Next)

■ 작업 5개를 이용한 HRN 스케줄링

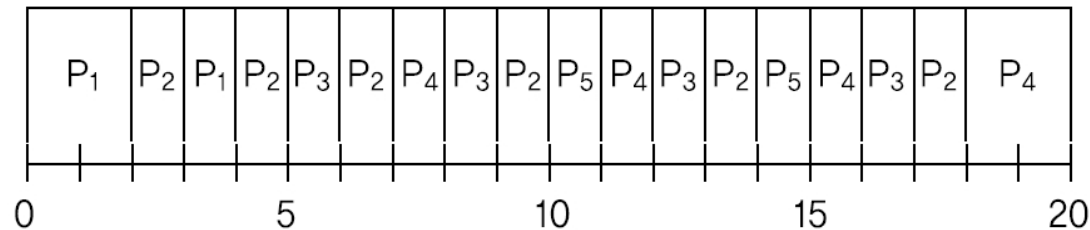
프로세스	도착시간	실행시간
P ₁	0	3
P ₂	2	6
P ₃	4	4
P ₄	6	5
P ₅	8	2



HRN 스케줄링의 간트 도표

$$\text{대기시간} : 0 + (3 - 2) + (9 - 4) + (15 - 6) + (20 - 15) = 10$$

■ 순환할당 스케줄링의 간트 도표(규정 시간량=1)



순환할당 스케줄링의 간트 도표

$$\text{대기시간} : (3 - 2) + (17 - 5 - 2) + (16 - 3 - 4) + (18 - 3 - 6) + (14 - 1 - 8) = 6.8$$

