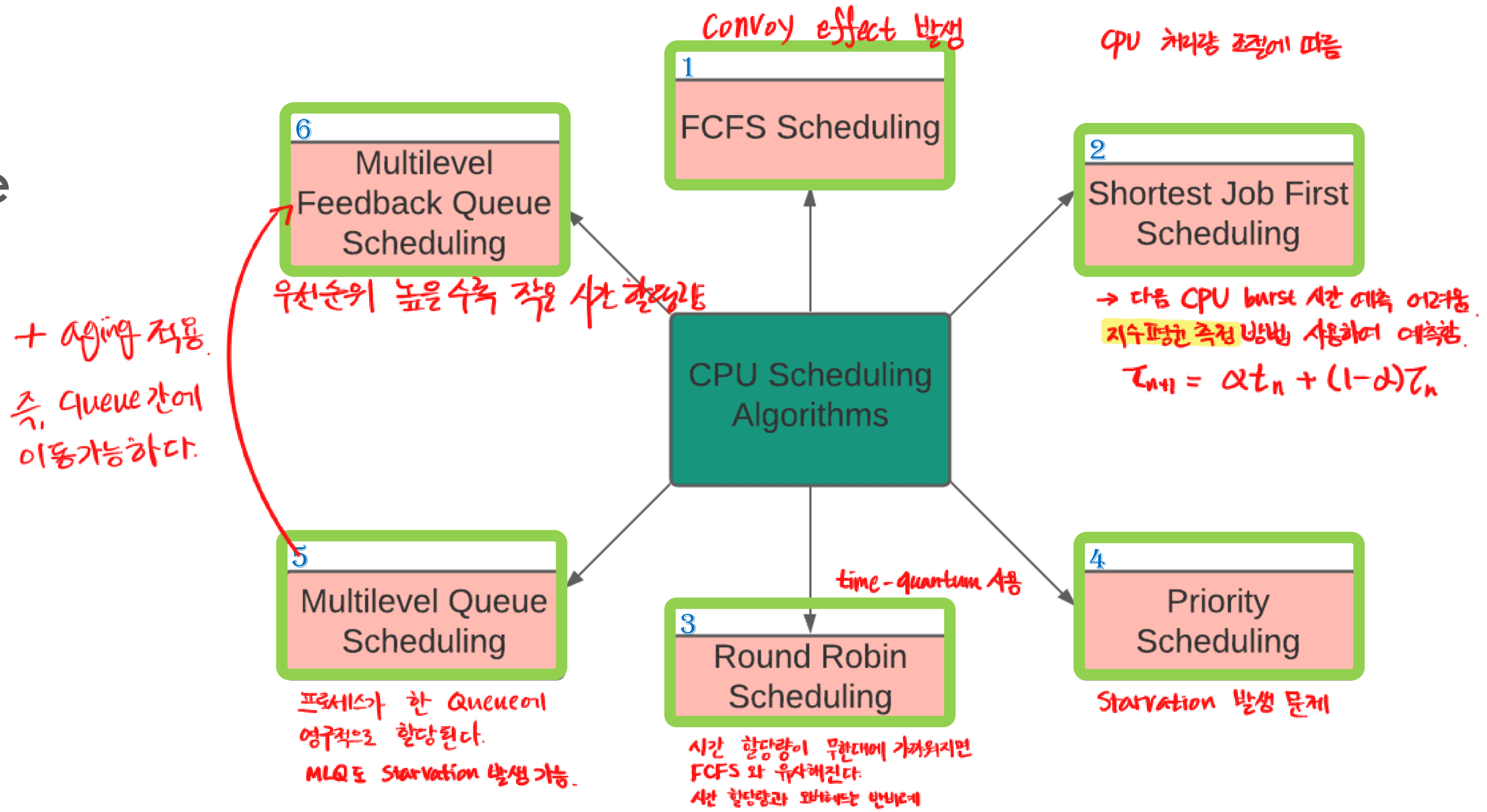
CHAPTER 5
CPU SCHEDULING

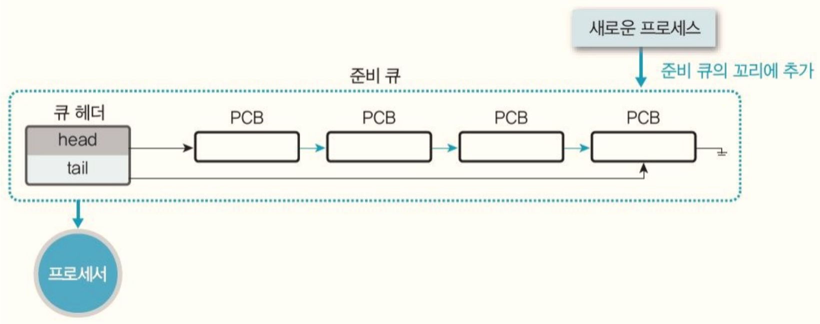
10. Scheduling Algorithms

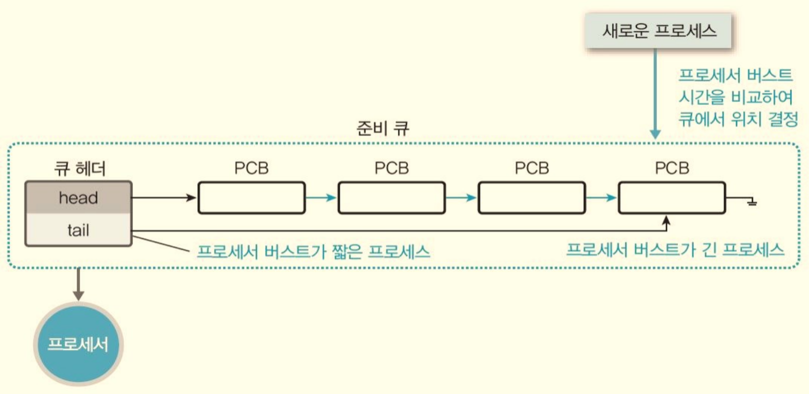
Scheduling Algorithm Optimization Criteria

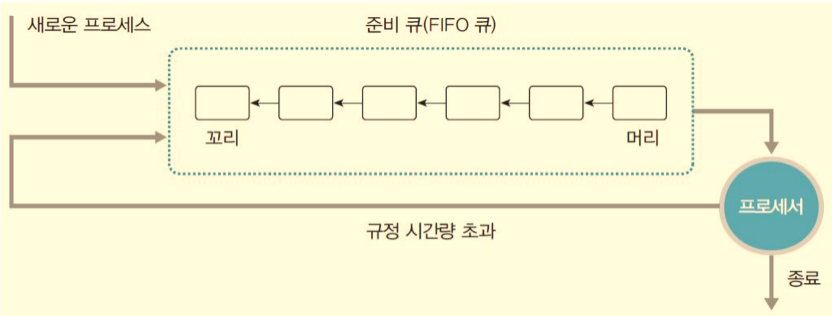
- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

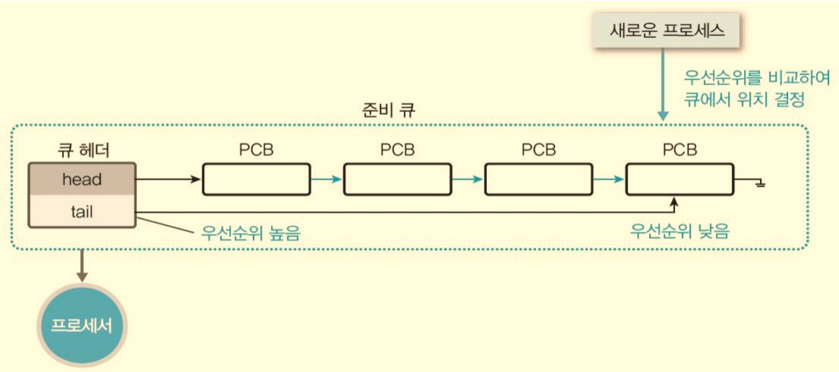


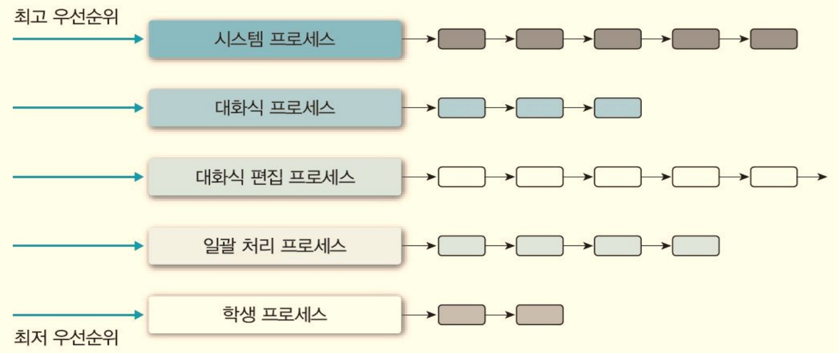
Scheduling Algorithm

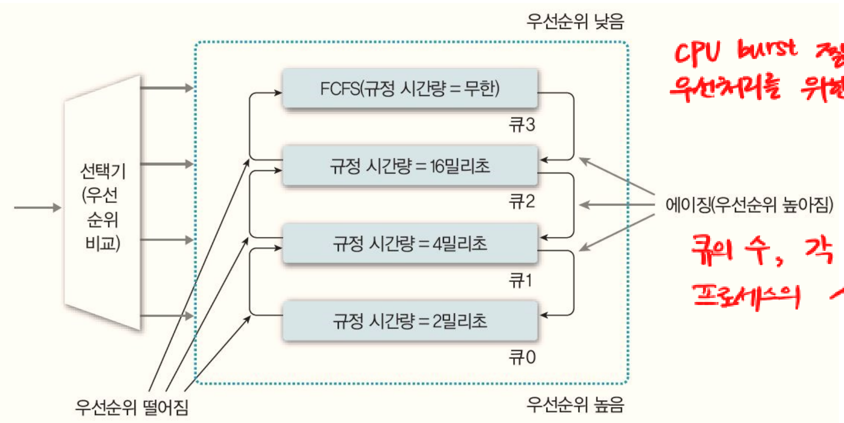












CPU burst 짧은 프로세스 우선처리를 위한 설계

큐의 수, 각 큐의 스케쥴링 알고리즘
프로세스의 상위/하위 큐로의 이동

First- Come, First-Served (FCFS) Scheduling

Process	Burst Time$[ms]$
P_1	24
P_2	3
P_3	3

- Suppose that the processes arrive in the order: P1 , P2 , P3
  The Gantt Chart for the schedule is: 시간에 따른 일의 진행 상황 표시하는 차트

Turnaround - time

- Waiting time for P1 = 0; P2 = 24; P3 = 27$[ms]$
- Average waiting time: $(0 + 24 + 27)/3 = 17[ms]$

FCFS Scheduling (Cont.)

Suppose that the processes arrive in the order:

$$P_2 \,,\; P_3 \,,\; P_1$$

- The Gantt chart for the schedule is:

- Waiting time for $P_1 = 6; P_2 = 0; P_3 = 3[ms]$
- Average waiting time: $(6 + 0 + 3)/3 = 3[ms]$
- Much better than previous case
- Convoy effect - short process behind long process
  - Consider one CPU-bound and many I/O-bound processes

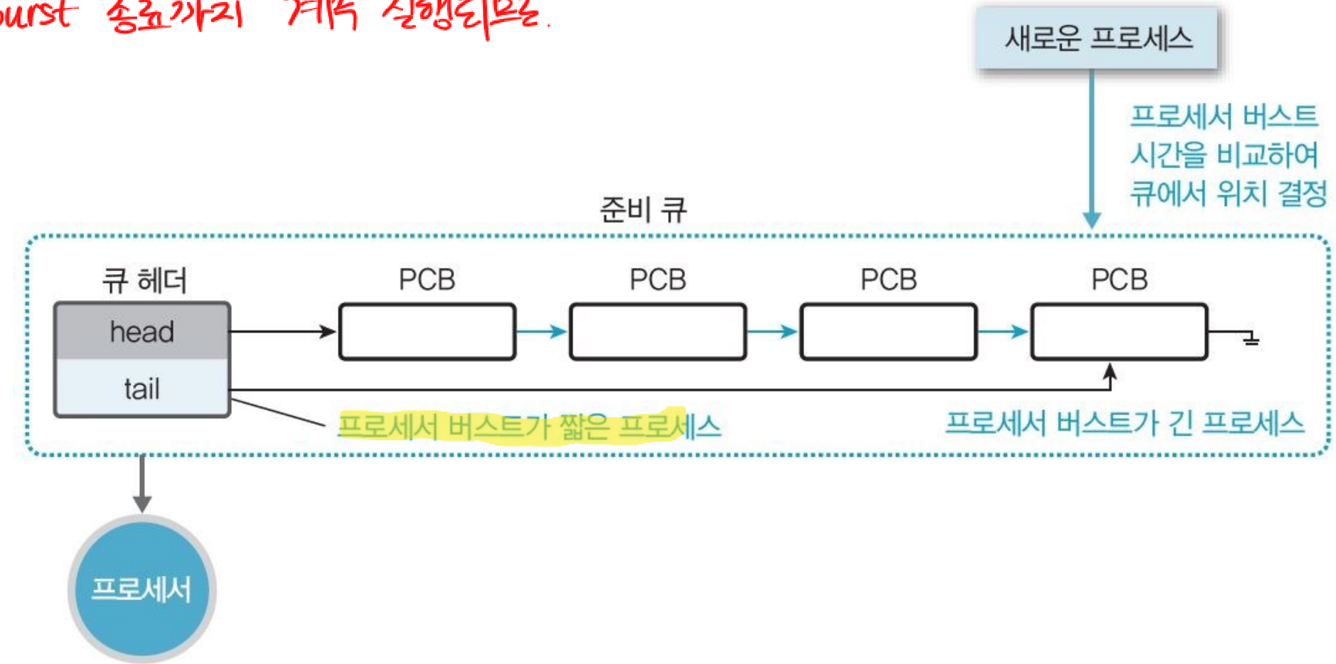
convoy effect
CPU burst가 긴 Process가 먼저 실행되면,
뒤에 있는 짧은 process 들이 오랜 시간 기다려야함.
(I/O bound process가 CPU bound process 뒤에
있을 때 등등)

Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst
  - Use these lengths to schedule the process with the shortest time
- SJF is optimal – gives minimum average waiting time for a given set of processes

  대기시간이 최소화되는 방식.
  - The difficulty is knowing the length of the next CPU request
  - Could ask the user

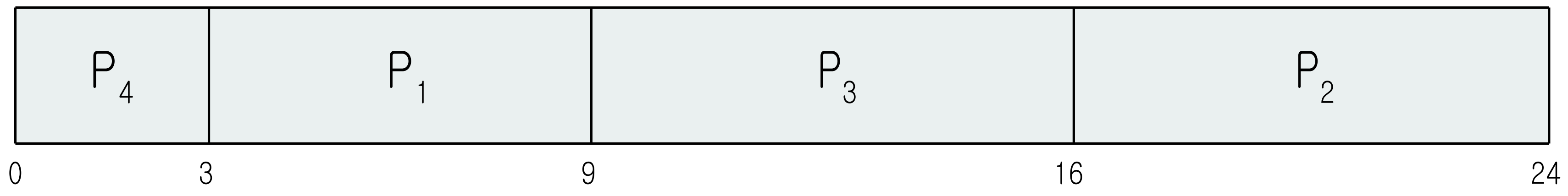
SJF는 비선점형 – CPU burst 종료까지 계속 실행되므로.



Example of SJF

Process	Burst Time[ms]
P_1	6
P_2	8
P_3	7
P_4	3

- SJF scheduling chart



- Average waiting time = (3 + 16 + 9 + 0) / 4 = 7[ms]

Determining Length of Next CPU Burst

- Can only estimate the length – should be similar to the previous one
  - Then pick process with shortest predicted next CPU burst

Prediction? how?

- Can be done by using the length of previous CPU bursts, using exponential averaging

1. $t_n =$ actual length of n^{th} CPU burst
2. $\tau_{n+1} =$ predicted value for the next CPU burst
3. $\alpha,\ 0 \le \alpha \le 1$
4. Define: $\tau_{n+1} = \alpha\, t_n + (1-\alpha)\tau_n.$

상수값은 반영 안함

t-2 : 10 , t-1 : 5 → t = ? 7.5

현재값과 과거값의 중요도에 따른 비율 나누어서 값을 구한다

weight.

- Commonly, α set to ½
- Preemptive version called shortest-remaining-time-first

Prediction of the Length of the Next CPU Burst

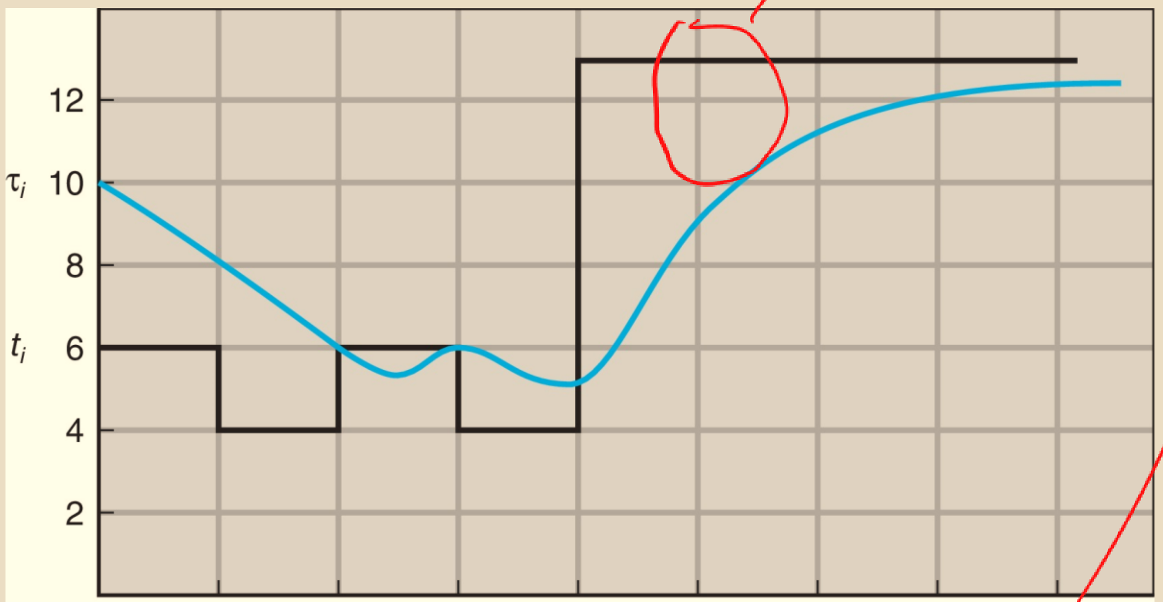
시간 Point 예제 - 주식 시황앞에 따른 확률 변화

현재값이 중요하면 α값을 크게 잡는다. (13 13 ...)

$$\tau_{n+1} = \alpha\, t_n + (1-\alpha)\tau_n$$

$$\tau_0 = 10$$

$$\alpha = \frac{1}{2}$$



$\tau_1 = \alpha$ $\tau_2 = \frac{1}{2}$ $\tau_3 = \frac{1}{2}$ $\tau_4 = \frac{1}{2}4 + \frac{1}{2}6$

CPU burst (t_i)		6	4	6	4	13	13	13	...
"guess" (τ_i)	10	8	6	6	5	9	11	12	...

$$\tau_1 = \alpha\, t_0 + (1-\alpha)\tau_0$$

$$\tau_2 = \alpha\, t_1 + (1-\alpha)\tau_1$$

$\vdots$

Examples of Exponential Averaging

- $\alpha = 0$
  - $\tau_{n+1} = \tau_n$
  - Recent history does not count
- $\alpha = 1$
  - $\tau_{n+1} = \alpha\, t_n$
  - Only the actual last CPU burst counts
- If we expand the formula, we get:

$$\tau_{n+1} = \alpha\ t_n + (1-\alpha)\tau_n$$

$$\tau_n = \alpha\ t_{n-1} + (1-\alpha)\tau_{n-1}$$

지수로그로 증가하는 이유

$$\tau_{n-1} = \alpha\ t_{n-2} + (1-\alpha)\tau_{n-2}$$

$$\vdots$$

$$\tau_1 = \alpha\ t_0 + (1-\alpha)\tau_0$$

$$\tau_{n+1} = \alpha\ t_n + (1-\alpha)\alpha t_{n-1} + \cdots \qquad \cdots$$

Exponential - average

⇒ 지수함수적으로 줄어든다.

- Since both α and $(1 - \alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor

Example of Shortest-remaining-time-first

- Now we add the concepts of varying arrival times and preemption to the analysis

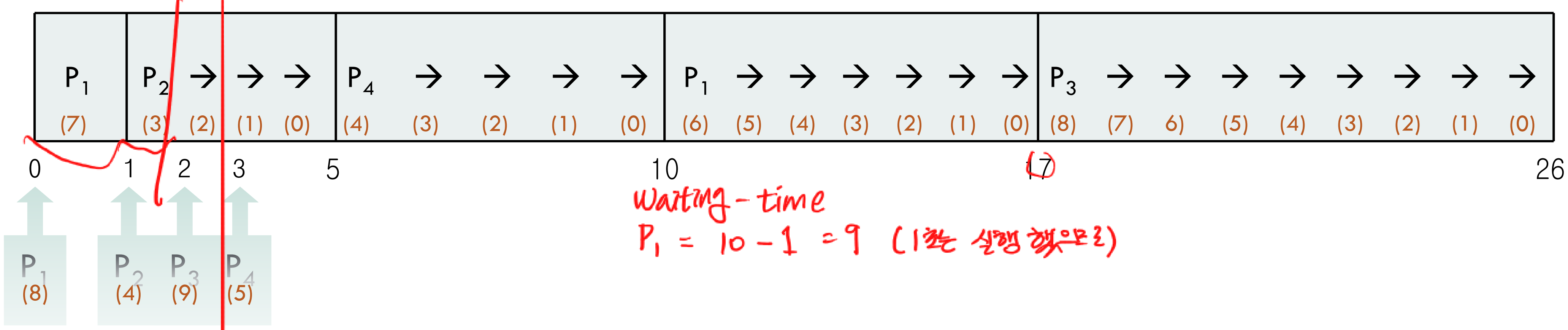
Process	Arrival Time	Burst Time[ms]
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

P_1: 7

Remaining-time 기준

SJF 이지만 Shortest-remaining-time first가 더 알맞은 용어임

- Preemptive SJF Gantt Chart



Waiting-time

P_1 = 10 - 1 = 9 (1초는 실행했으므로)

- Average waiting time = [(10-1)+(1-1)+(17-2)+(5-3)]/4 = 26/4 = 6.5 [ms]

구분된 시간이 있다.

- Each process gets a small unit of CPU time (**time quantum** q), usually 10~100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are n processes in the ready queue and the time quantum is q, then each process gets $1/n$ of the CPU time in chunks of at most q time units at once. No process waits more than $(n-1)q$ time units.

내 차례.

- Timer interrupts every quantum to schedule next process

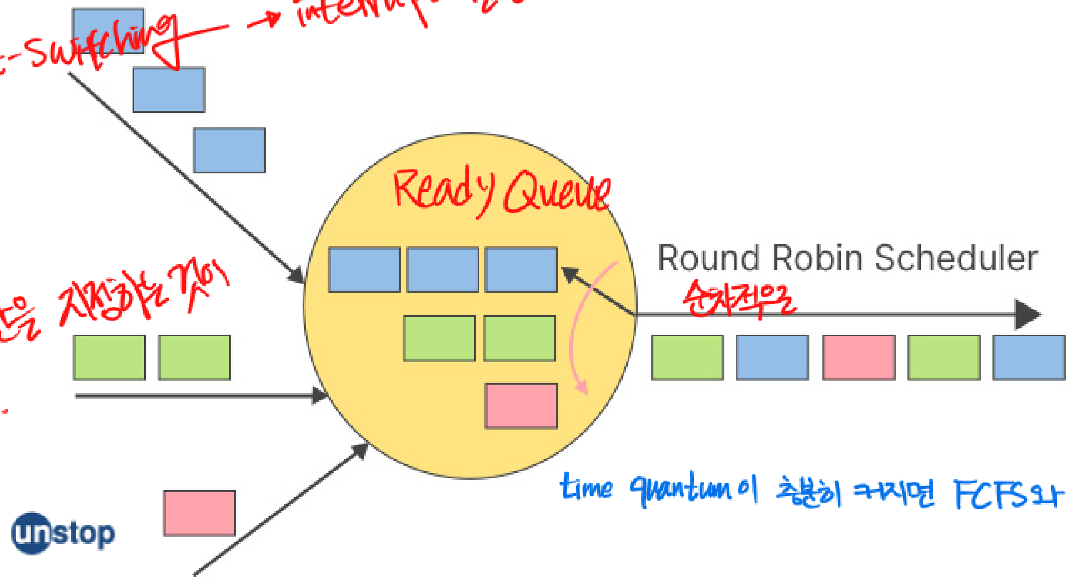
Timer를 통한 Context-Switching → interrupt 발생 시킴.

- Performance
  - q large ⇒ FIFO
  - q small ⇒ q must be large with respect to context switch, otherwise overhead is too high

적절한 시간을 지정하는 것이 관건이다.

dispatcher – response time

dispatch – latency

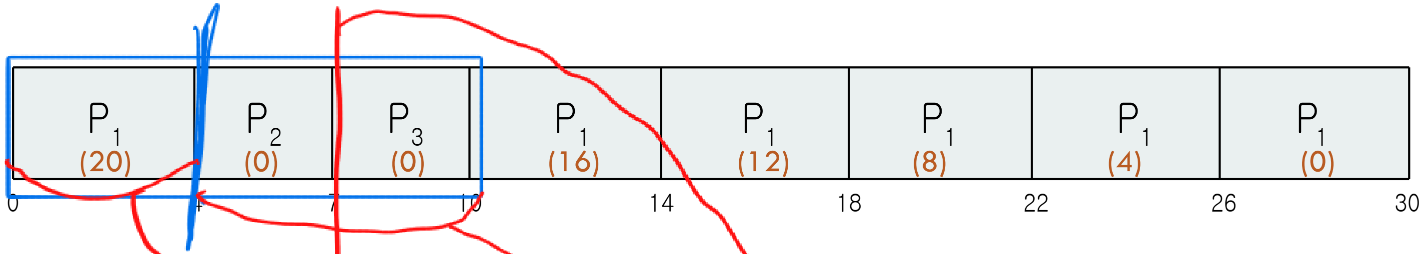


time quantum이 충분히 커지면 FCFS와 같아진다.

Example of RR with Time Quantum = 4ms

Process	Burst Time[ms]
P_1	24
P_2	3
P_3	3

- The Gantt chart is:



  - Average waiting time = [(10-4)+(4)+(7)]/3 = 17/3 = 5.66 [ms]

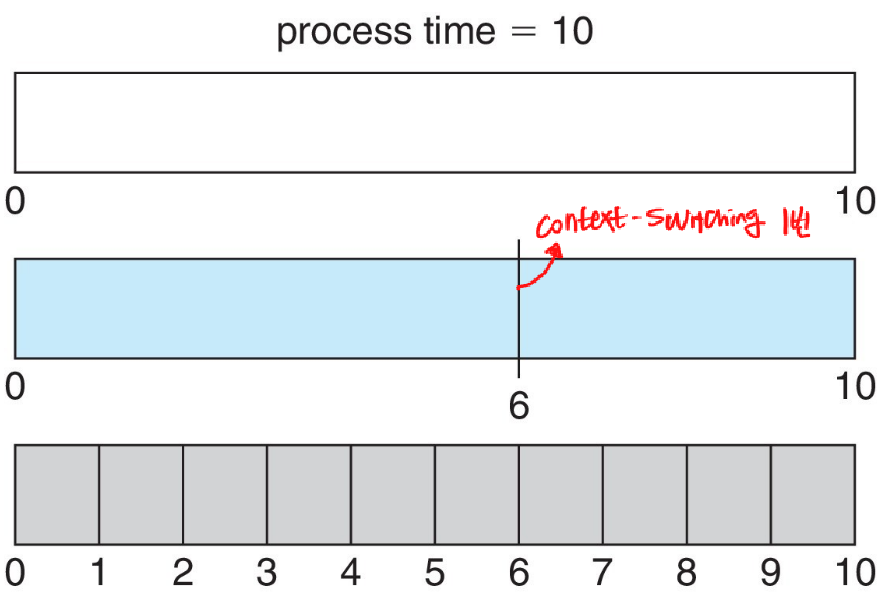
- Typically, higher average turnaround than SJF, but better ***response***

Turnaround는 길어질 수 있지만, Response가 빠르기 때문에 용이하다.

- q should be large compared to context switch time 퀀텀이 너무 크면
- q usually 10ms to 100ms, context switch < 10 μsec 약 $\frac{ms}{1000}$

1000배 차이

Time Quantum and Context Switch Time

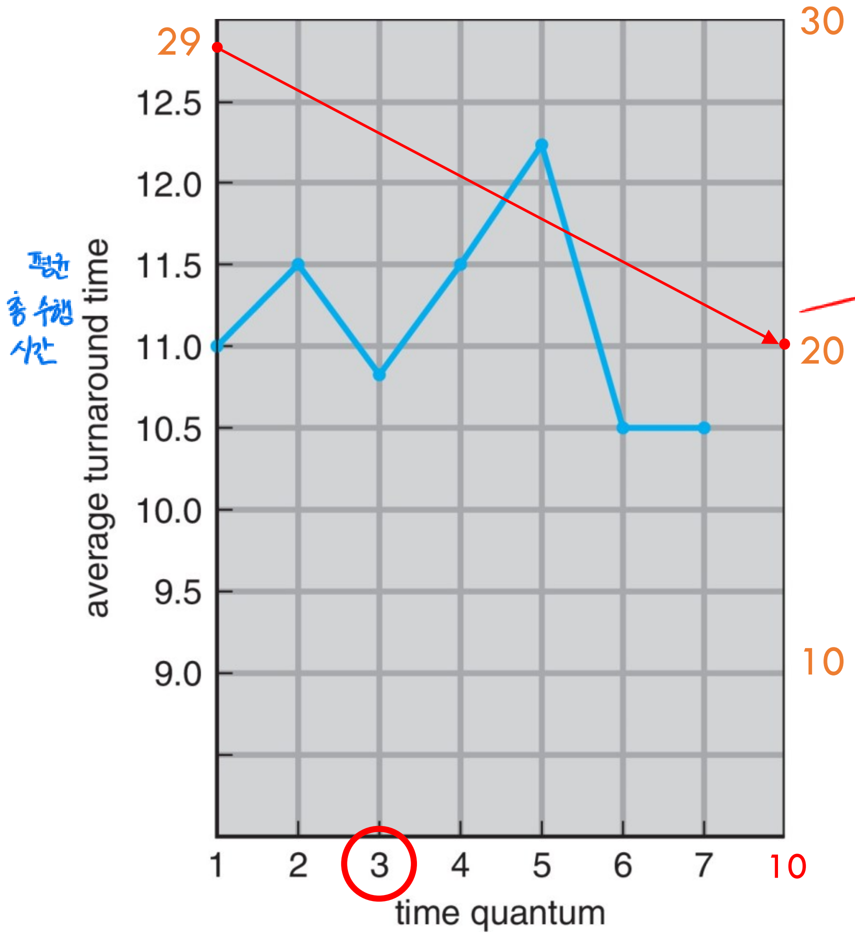


quantum	context switches	
12	0	1 time quantum
6	1	2 time quanta
1	9	10 time quanta

성능차이가 크다.
context-switching

- Each process gets a small unit of CPU time (time quantum q), usually 10~100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- q usually 10*ms* to 100*ms*, context switch < 10 μsec 적당한 time Quantum 지정이 관건이다.

Turnaround Time Varies With The Time Quantum



process	time
P_1	6
P_2	3
P_3	1
P_4	7

10
10
10

동시에 같은 시간에 시작

time quantum을 많이 줄 때 유리한 경우

Q=1 일 때,

P_1 15 P_2 9 P_3 3 P_4 17

Q=2

P_1 14 P_2 10 P_3 5 P_4 17

time-quantum = 3 일 때.

P_1	P_2	P_3	P_4	P_1	P_4	P_4

0 3 6 7 10 13 16 17

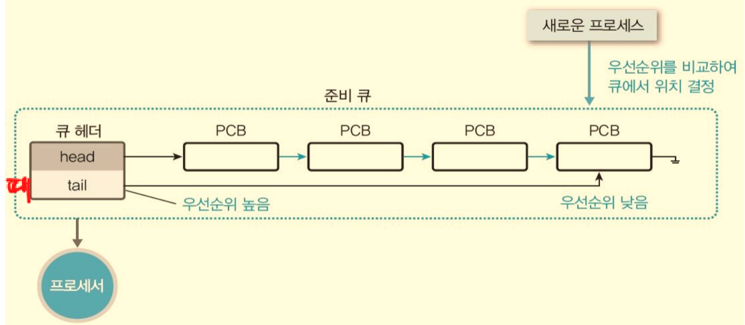
Process 바로 종료 후 다음 프로세스 실행

	Turn around Time
P1	13
P2	6
P3	7
P4	17
ATA	10.75

❖ 80% of CPU bursts should be shorter than q

- A priority number (integer) is associated with each process Process 별로 integer으로 우선순위 지정
- The CPU is allocated to the process with the highest priority 더 낮은 숫자가 더 우선순위 높음
  (smallest integer ≡ highest priority, 3bit or 14bit)
  - Preemptive Process 교체 발생 Ready-Queue 에서 발생하는 동작
  - Nonpreemptive Process

  Preemptive는 Ready-Queue에 우선순위 우선인 Proc 들어올 때 교체됨.



- SJF is priority scheduling where priority is the inverse of predicted next CPU burst time
- Problem ≡ **Starvation** – low priority processes may never execute 기아
- Solution ≡ **Aging** – as time progresses increase the priority of the process 에이징

우선순위가 동일한 프로세스들은 FCFS 순서로 스케줄링

Example of Priority Scheduling

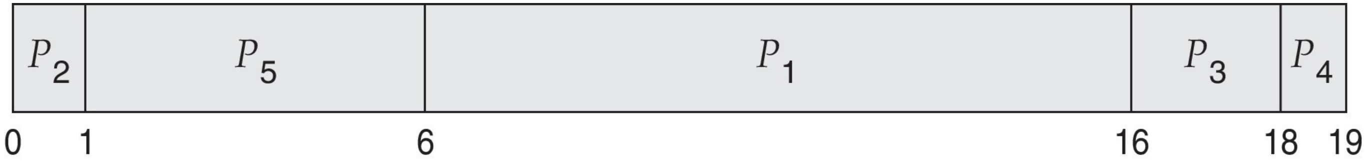
Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

동시 시작?

3bits or 14bits의 순서 숫자 제공

도착 시간 차이에 따른 선점/비선점 변화 계산.

- Priority scheduling Gantt Chart



- Waiting time
  - $6+0+16+18+1=41\text{msec}$
- Average waiting time = 8.2 msec

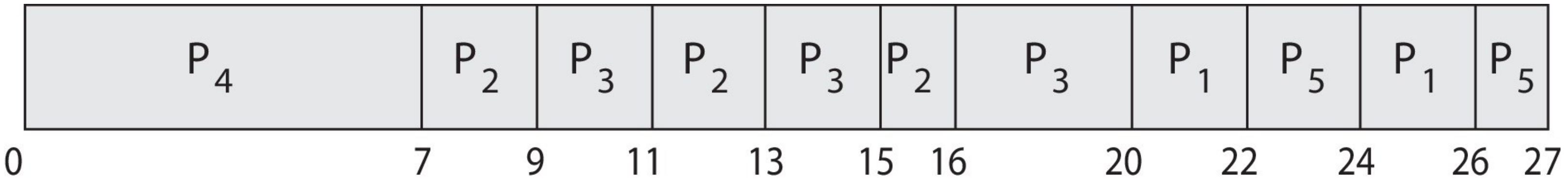
Priority Scheduling w/ Round-Robin

Process	Burst Time	Priority
P_1	4	3
P_2	5	2
P_3	8	2
P_4	7	1
P_5	3	3

우선순위가 같으면 time-quantum 지정하여 RR 수행

- Run the process with the highest priority. Processes with the same priority run round-robin
- Gantt Chart with 2 ms time quantum

★ waiting time 과 turn-around time 계산



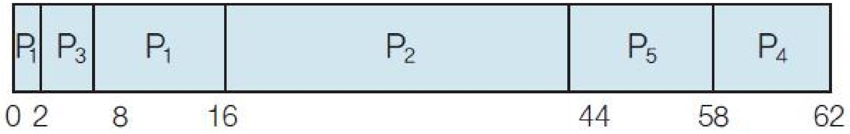
☆ 계산 방법 이해

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

대기(우선순위 낮음)

P_1보다 우선순위 높아서 먼저 실행

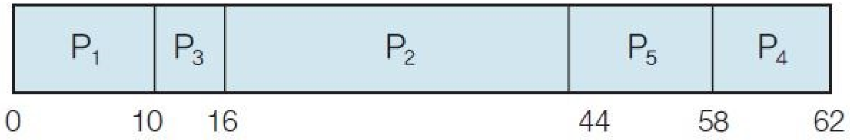
(a) 준비 큐



(b) 선점 우선순위 간트 차트

뺏김

동일 우선순위는 FCFS?



(c) 비선점 우선순위 간트 차트

안뺏김

(b) 선점 예의 반환시간과 대기시간

프로세스	반환시간	대기시간
P_1	(16 − 2) = 14	(8 − 2) = 6
P_2	(44 − 1) = 43	(16 − 1) = 15
P_3	(8 − 2) = 6	(2 − 2) = 0
P_4	(62 − 3) = 59	(58 − 3) = 55
P_5	(58 − 4) = 54	(44 − 4) = 40
평균 반환시간 : 35.2[= (14 + 43 + 6 + 59 + 54/5]		평균 대기시간 : 23.4[= (6 + 15 + 0 + 55 + 40)/5]

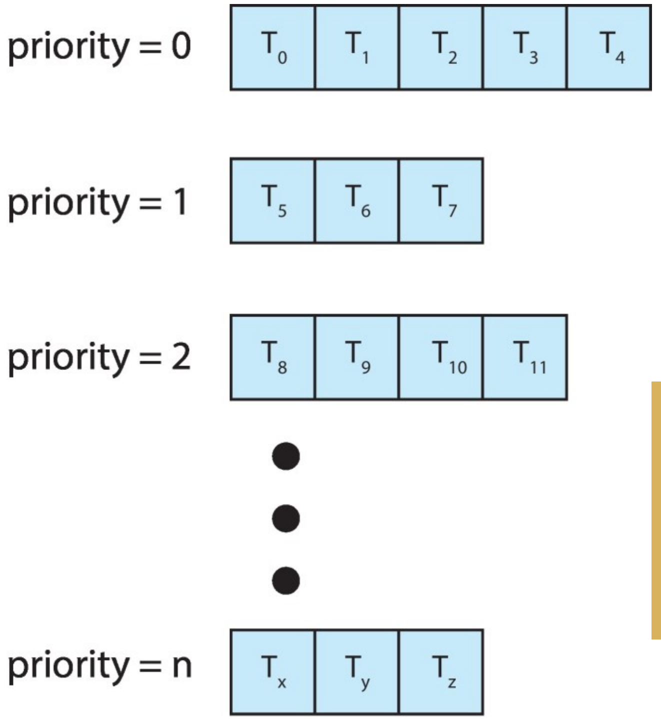
(c) 비선점 예의 반환시간과 대기시간

프로세스	반환시간	대기시간
P_1	10	0
P_2	(44 − 1) = 43	(16 − 1) = 15
P_3	(16 − 2) = 14	(10 − 2) = 8
P_4	(62 − 3) = 59	(58 − 3) = 55
P_5	(58 − 4) = 54	(44 − 4) = 40
평균 반환시간 : 36[= (10 + 43 + 14 + 59 + 54)/5]		평균 대기시간 : 23.6[= (0 + 15 + 8 + 55 + 40)/5]

Multilevel Queue

- With priority scheduling, have separate queues for each priority.
- Schedule the process in the highest-priority queue!

우선순위와 RR의 동일 분배 방식 사용

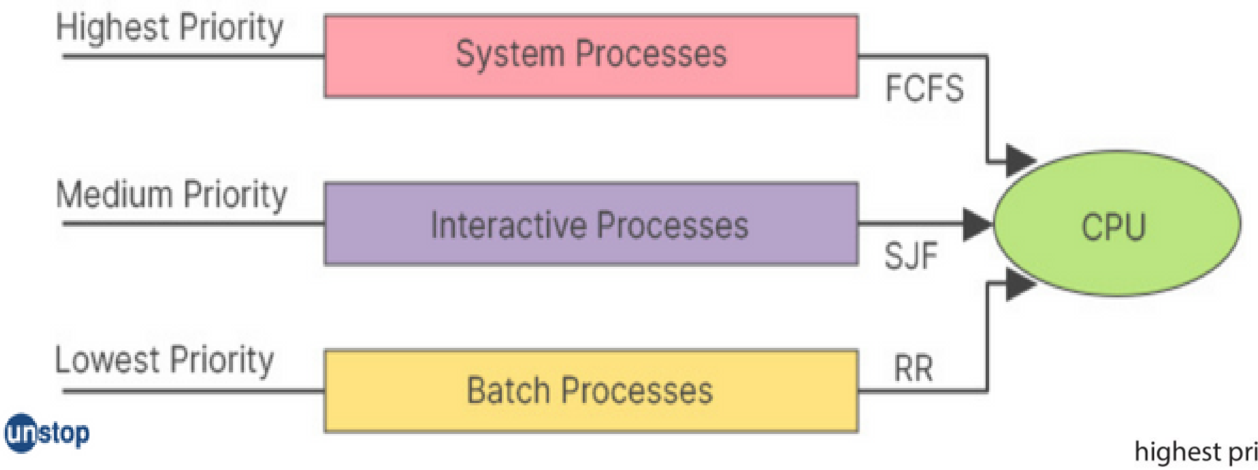


우선 순위 높은 것 (Priority가 작은 것) 부터
순서대로 처리함.
(Process, I/O 처리 등을 기반으로 Priority 나눔)

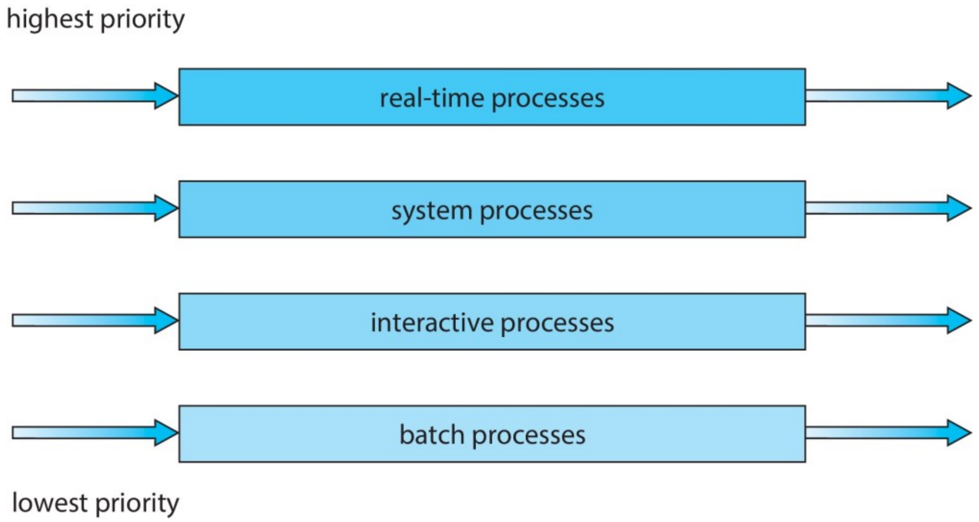
- 각 작업을 서로 다른 묶음으로 분류할 수 있을 때 사용
  - 준비 상태 큐를 종류별로 여러 단계로 분할
  - 작업을 메모리의 크기나 프로세스의 형태에 따라 특정 큐에 지정
  - 각 큐는 자신만의 독자적인 스케줄링 갖음

❖ Prioritization based upon process type

각 Queue 간의 Process 이동이 불가한 문제점 존재
→ Starvation 발생 가능



배분하는 방법?



- A process can move between the various queues; aging can be implemented this way
- Multilevel-feedback-queue scheduler defined by the following parameters:
  - number of queues
  - scheduling algorithms for each queue 선택기
  - method used to determine when to upgrade a process
  - method used to determine when to demote a process
  - method used to determine which queue a process will enter when that process needs service

우선순위 낮음
FCFS (무한 밀리초)
RR — 16 밀리초
4 밀리초
2 밀리초
우선순위 높음

Aging 방법 사용하여
Queue 간에 process 이동
(우선순위 떨어지거나 높아지거나)

Example of Multilevel Feedback Queue

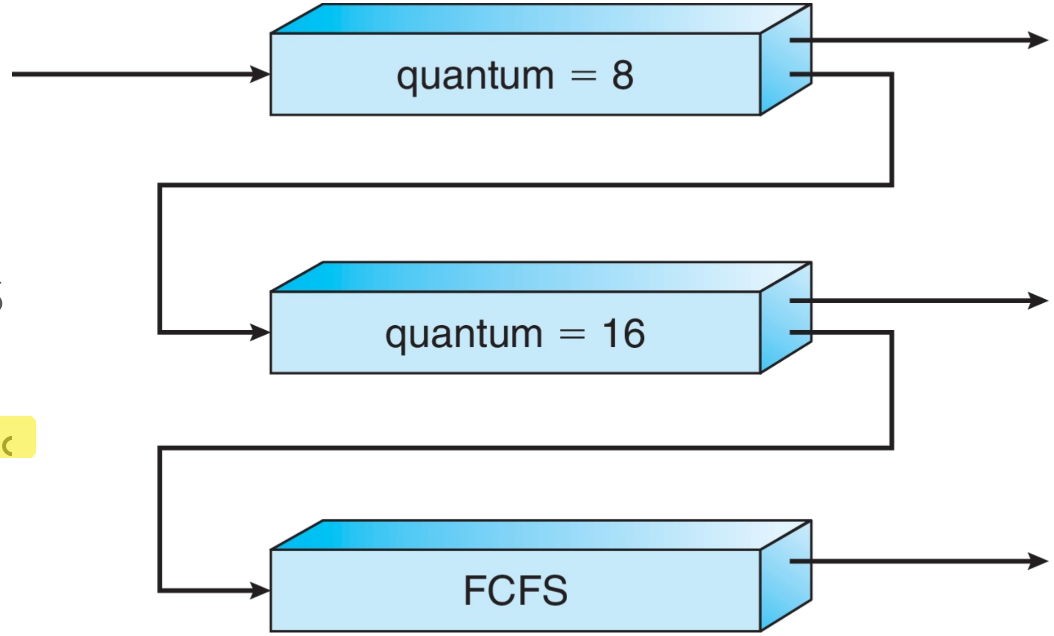
Three queues:

- Q0 – RR with time quantum 8 milliseconds
- Q1 – RR time quantum 16 milliseconds
- Q2 – FCFS

Scheduling

- A new job enters queue Q0 which is served FCFS
  - When it gains CPU, job receives 8 milliseconds
  - If it does not finish in 8 milliseconds, job is moved to queue Q1

    제한시간 내에 못 끝내면 우선순위 떨어짐
- At Q1 job is again served FCFS and receives 16 additional milliseconds
  - If it still does not complete, it is preempted and moved to queue Q2



제일 높은 Priority 부여 (새 프로세스에 대해)
이후 강등하여 분배.

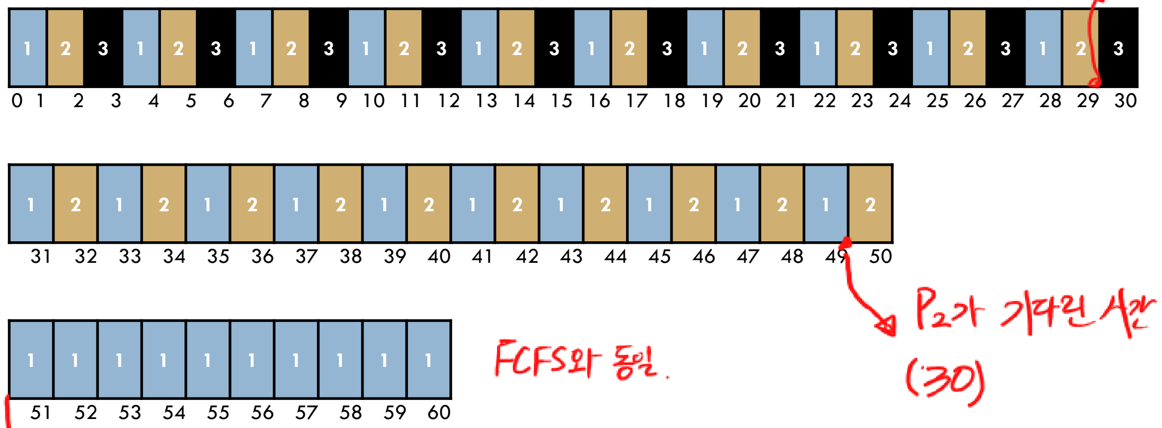
프로세스	실행 시간
P_1	30
P_2	20
P_3	10

(a) 준비 큐

												종료						종료	종료
큐 1:	P_1	P_2	P_3															P_2	P_1
큐 2:				P_1	P_2	P_3													
큐 3:							P_1	P_2	P_3	P_1	P_2	P_3	P_1	P_2	P_1	P_2	P_1		
시간	1	2	3	5	7	9	13	17	21	25	29	32	36	40	44	48	52	53	60

(b) 간트 차트

(30-10)



FCFS와 동일.

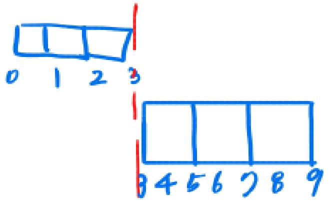
P_2가 기다린 시간 (30)

P1이 기다린 시간 (50 − 20 = 30 ; 20초는 이전에 사용)

$Q_1=1$

$Q_2=2$

$Q_3=4$



(a) 반환시간

프로세스	라운드 로빈[RR] 스케줄링	다단계 피드백 큐[MLFQ] 스케줄링
P_1	60	60
P_2	50	53
P_3	30	52
평균 반환시간 : $46\frac{2}{3}$[=(60+50+30)/3]		평균 반환시간 : $48\frac{1}{3}$[=(60+53+32)/3]

(b) 대기시간

프로세스	라운드 로빈[RR] 스케줄링	다단계 피드백 큐[MLFQ] 스케줄링
P_1	30	(53−23)=30
P_2	30	(52−19)=33
P_3	20	(29−7)=22
평균 대기시간 : 30[=(30+30+30)/3]		평균 대기시간 : $28\frac{1}{3}$[=(30+33+22)/3]

Scheduling 방법 각 특성 및 turn around time.
Waiting time 계산방법

End of Chapter 5