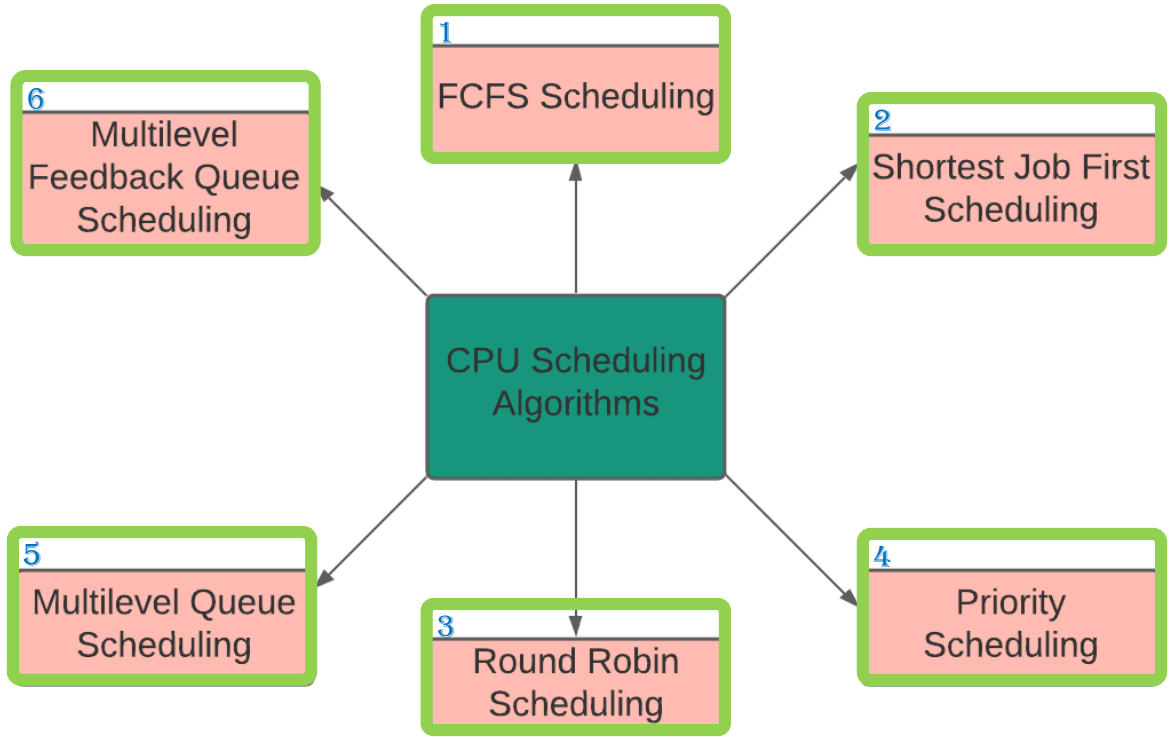
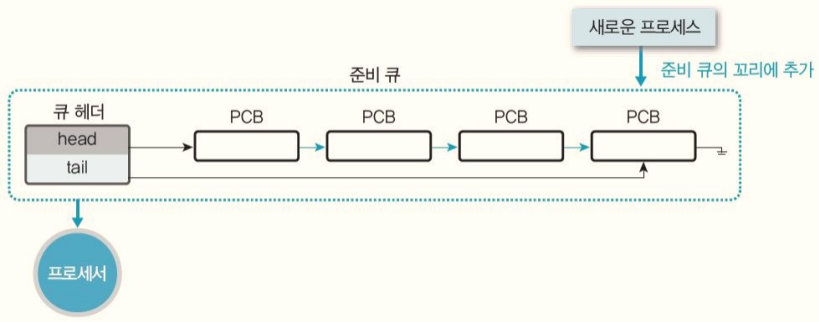
CHAPTER 5
CPU SCHEDULING

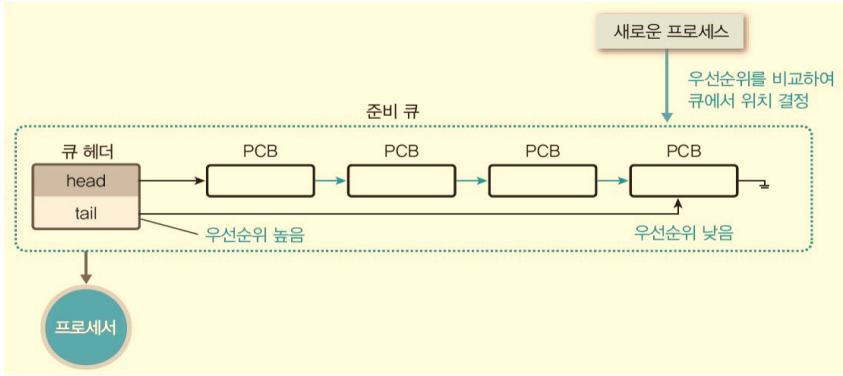
Scheduling Algorithm Optimization Criteria

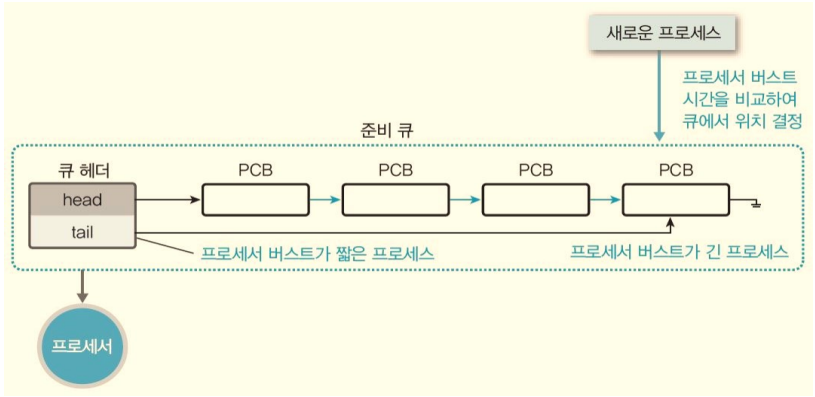
- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

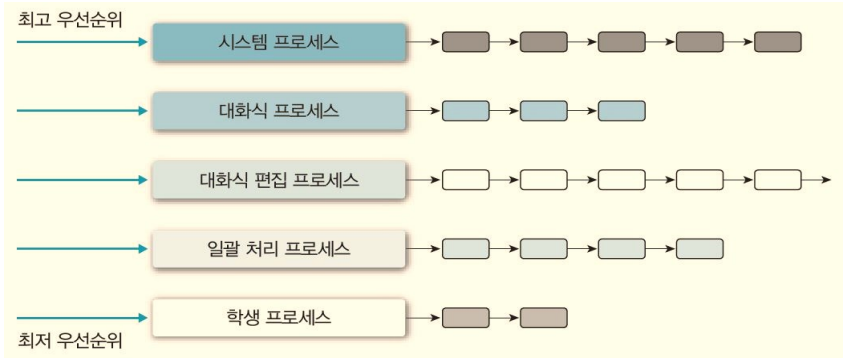


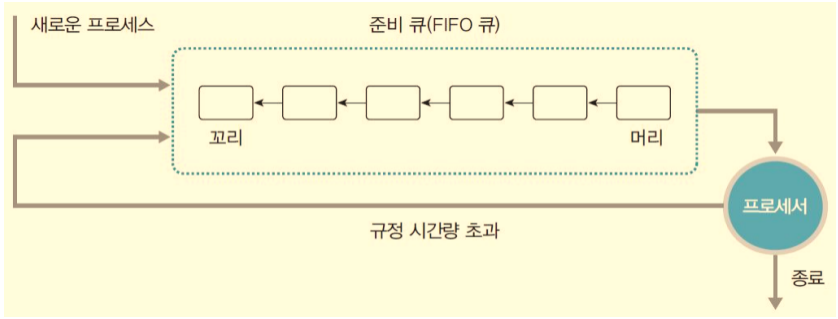
Scheduling Algorithm

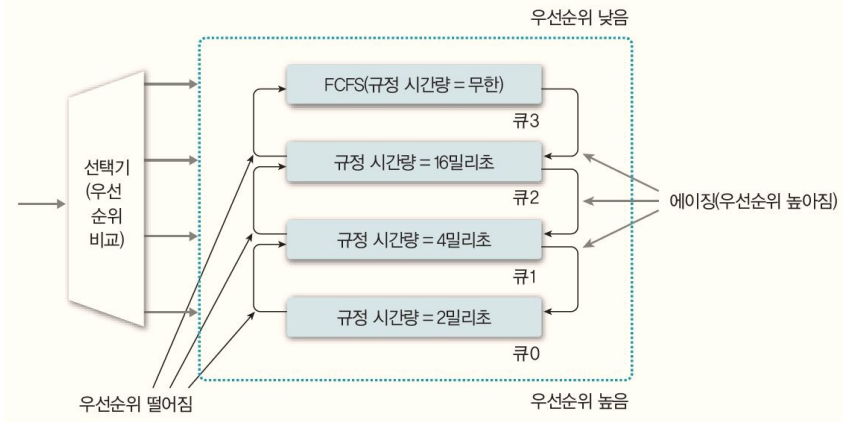








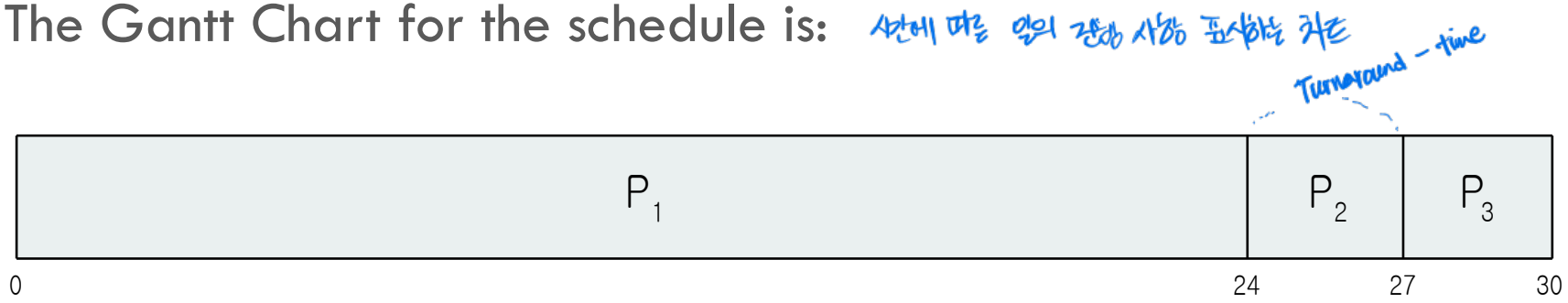




First- Come, First-Served (FCFS) Scheduling

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:



- 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:

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

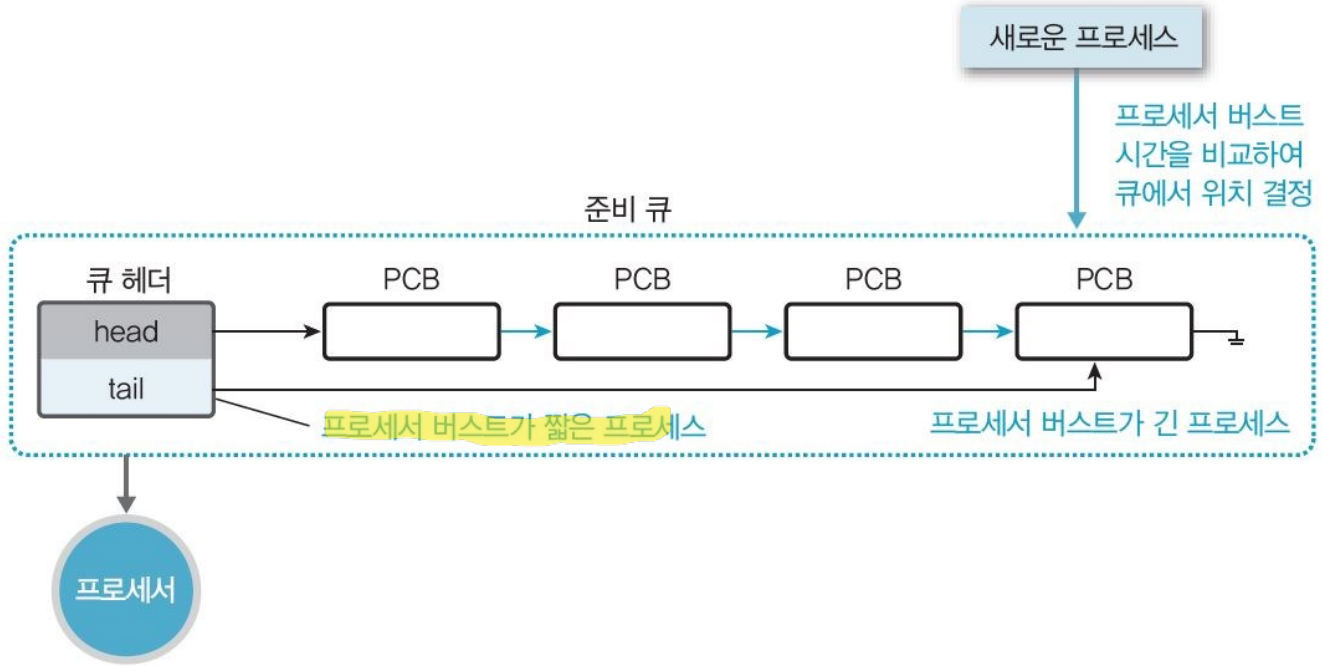
- The Gantt chart for the schedule is:

- 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

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
  - The difficulty is knowing the length of the next CPU request
  - Could ask the user

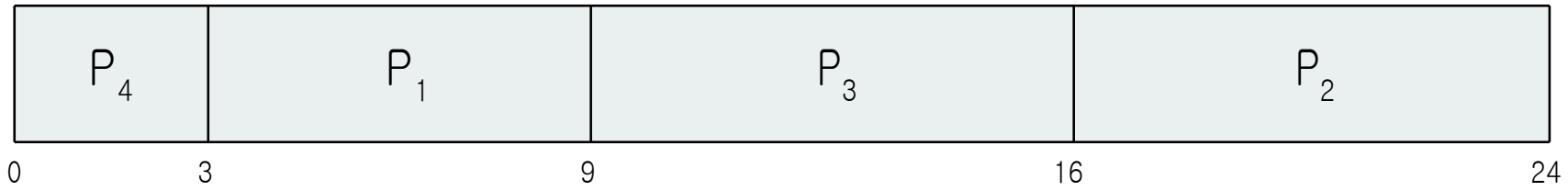
대기시간이 최소화되는 방식.



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. τ_{n+1} = predicted value for the next CPU burst
3. α, $0 \le \alpha \le 1$ 가중치
4. Define: $\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n$.

weight

현재값과 과거값의 중요도에 따른 비율 나누어서 값을 구한다

상수값은 반영 안함

t_{-2} : 10, t_{-1} : 5 → t = ? 7.5

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

Prediction of the Length of the Next CPU Burst

사건 point 여러 – 주식 상황에 따른 확률 변화

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

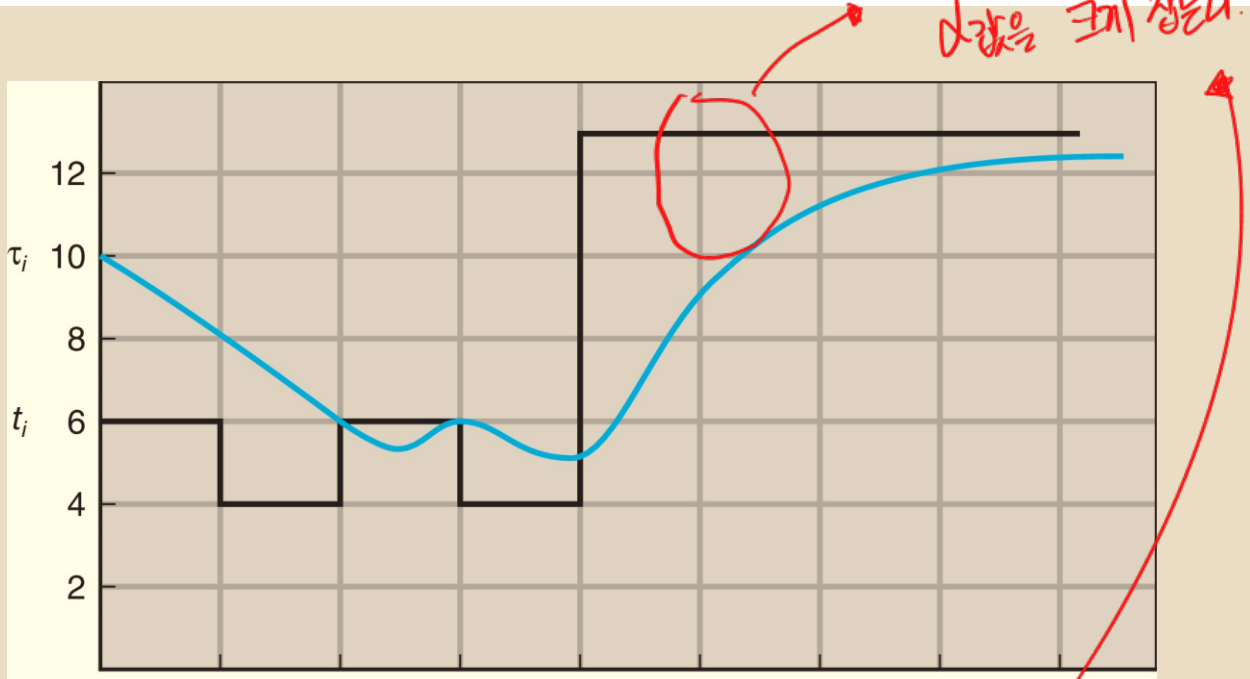
$$\tau_0 = 10$$

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

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

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

$$\vdots$$



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:

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

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

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

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

$$\vdots$$

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

자주로그로 증가하는 이유

$\cdots$

Exponential - average ⇒ 지수함수로 증가한다.

- 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

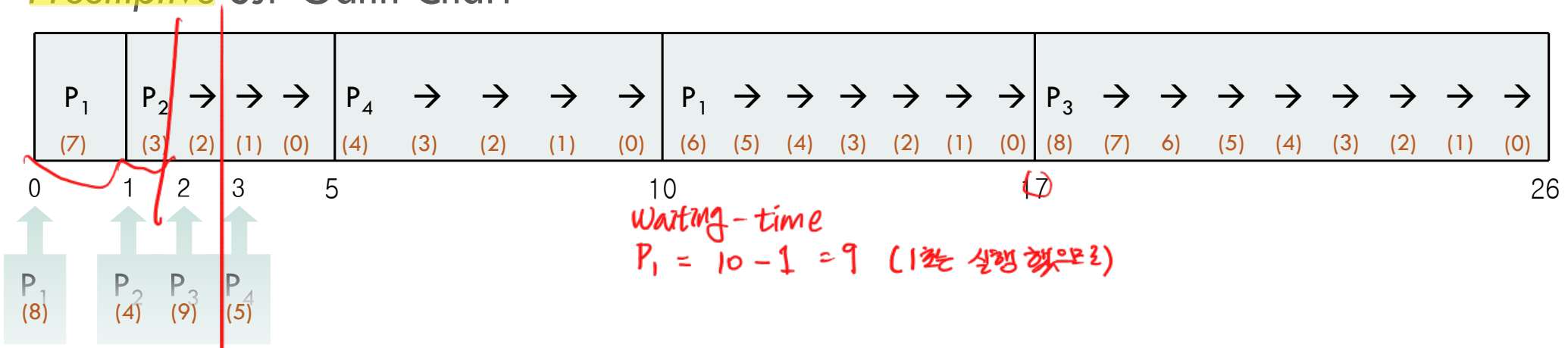
Process	Arrival Time	Burst Time[ms]
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

P_1: 7

Remaining-time 기준

SJF 에서만 Shortest-remaining-time first가 더 알맞은 용어일

- *Preemptive* SJF Gantt Chart



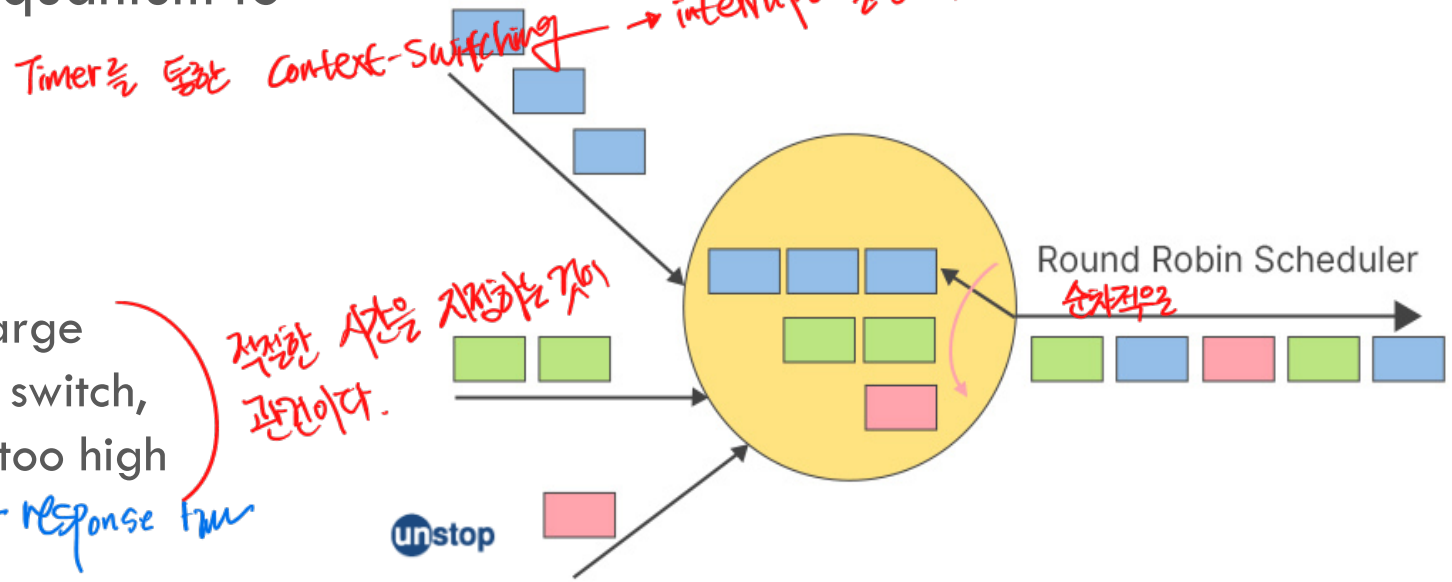
Waiting-time

$P_1 = 10 - 1 = 9$ (1초는 실행 했으므로)

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

Round Robin (RR)

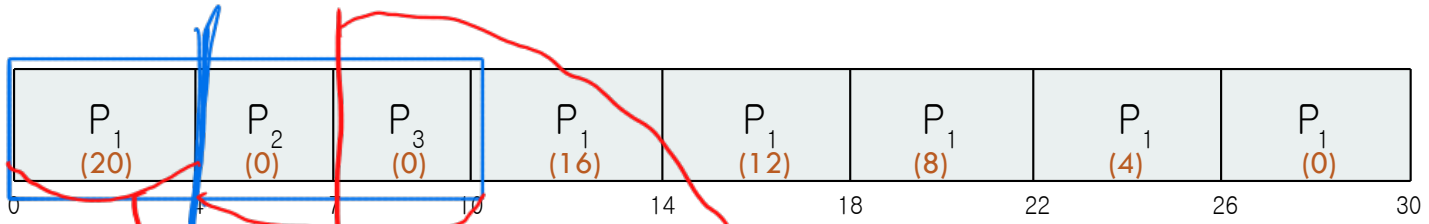
- Each process gets a small unit of CPU time (**time quantum** q), usually 10~100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are n processes in the ready queue and the time quantum is q, then each process gets $1/n$ of the CPU time in chunks of at most q time units at once. No process waits more than $(n-1)q$ time units.
- Timer interrupts every quantum to schedule next process
- Performance
  - q large $\Rightarrow$ FIFO
  - q small $\Rightarrow$ q must be large with respect to context switch, otherwise overhead is too high



Example of RR with Time Quantum = 4ms

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

- The Gantt chart is:

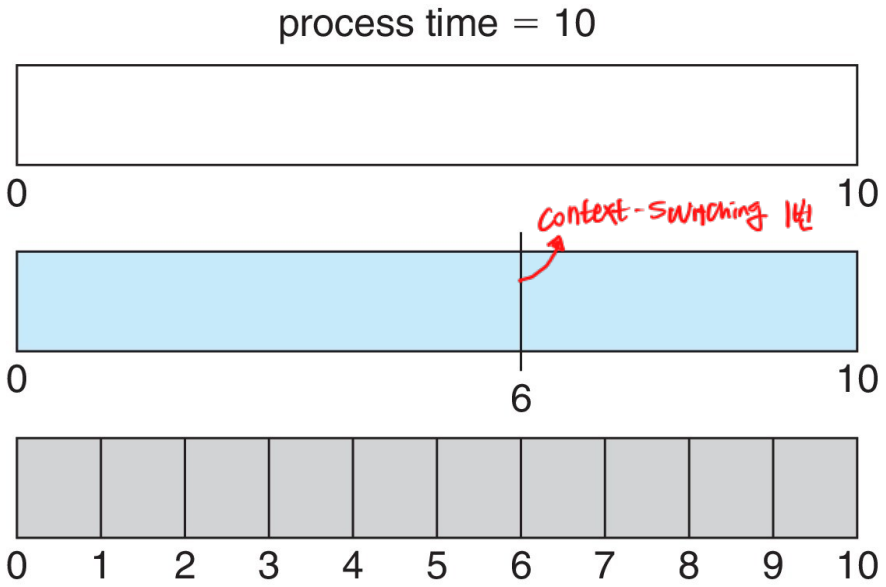


  - Average waiting time = [(10-4)+(4)+(7)]/3 = 17/3 = 5.66 [ms]

- Typically, higher average turnaround than SJF, but better ***response***
- q should be large compared to context switch time 컨텍스트 스위칭이 너무 크면
- q usually 10ms to 100ms, context switch < 10 μsec 약 1ms / 1000

Turnaround를 길게 잡을 수 있지만, Response가 빠른 때문에 유리하다.

Time Quantum and Context Switch Time

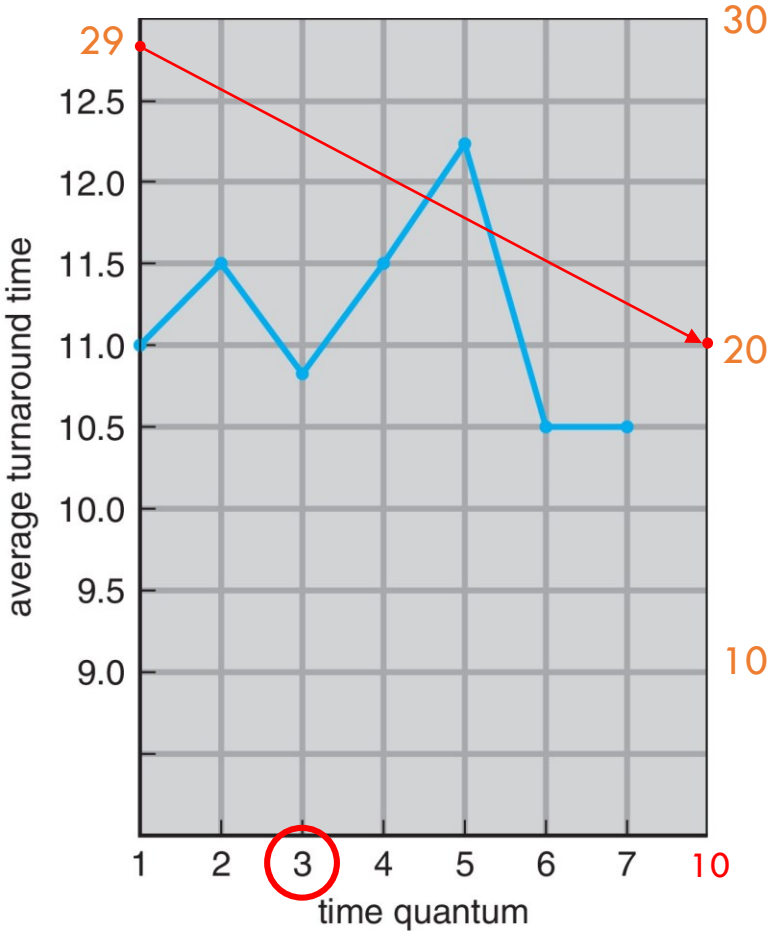


quantum	context switches	
12	0	1 time quantum
6	1	2 time quanta
1	9	10 time quanta

성능저하 크다
context-switching

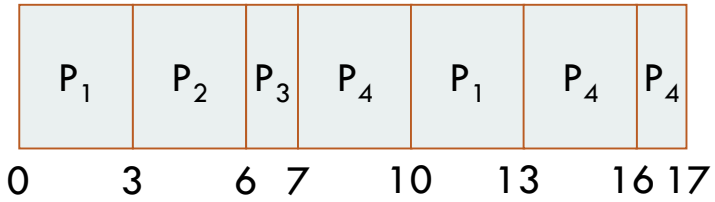
- Each process gets a small unit of CPU time (time quantum q), usually 10~100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- q usually 10*ms* to 100*ms*, context switch < 10 μsec

Turnaround Time Varies With The Time Quantum



process	time
P_1	6
P_2	3
P_3	1
P_4	7

10
10
10

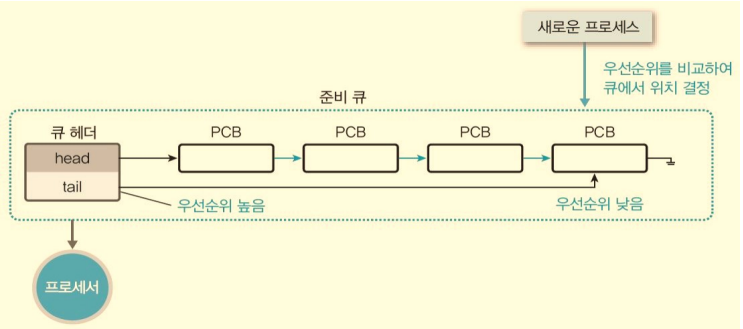


	Turn around Time
P1	13
P2	6
P3	7
P4	17
ATA	**10.75**

- 80% of CPU bursts should be shorter than q

Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority, 3bit or 14bit)
  - Preemptive
  - Nonpreemptive
- SJF is priority scheduling where priority is the inverse of predicted next CPU burst time
- Problem ≡ **Starvation** – low priority processes may never execute
- Solution ≡ **Aging** – as time progresses increase the priority of the process

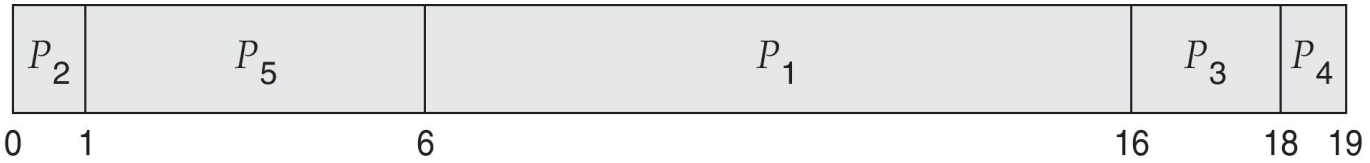


우선순위가 동일한 프로세스들은 FCFS 순서로 스케줄링

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

- Priority scheduling Gantt Chart

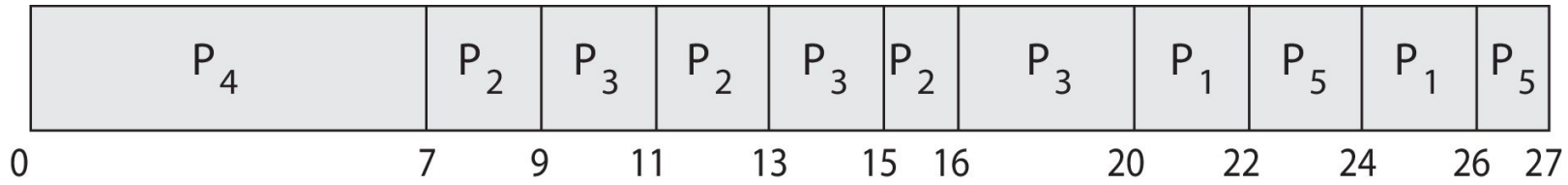


- Waiting time
  - $6+0+16+18+1=41\text{msec}$
- Average waiting time = 8.2 msec

Priority Scheduling w/ Round-Robin

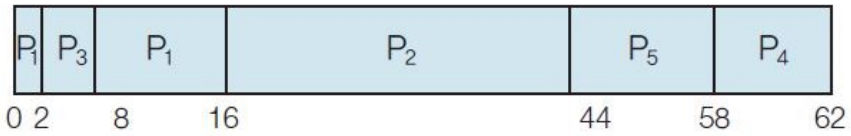
Process	Burst Time	Priority
P_1	4	3
P_2	5	2
P_3	8	2
P_4	7	1
P_5	3	3

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

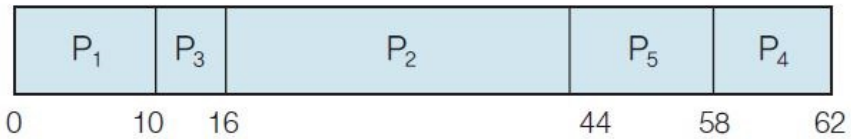


프로세스	도착 시간	실행 시간	우선순위
P_1	0	10	2
P_2	1	28	3
P_3	2	6	1
P_4	3	4	4
P_5	4	14	3

(a) 준비 큐



(b) 선점 우선순위 간트 차트



(c) 비선점 우선순위 간트 차트

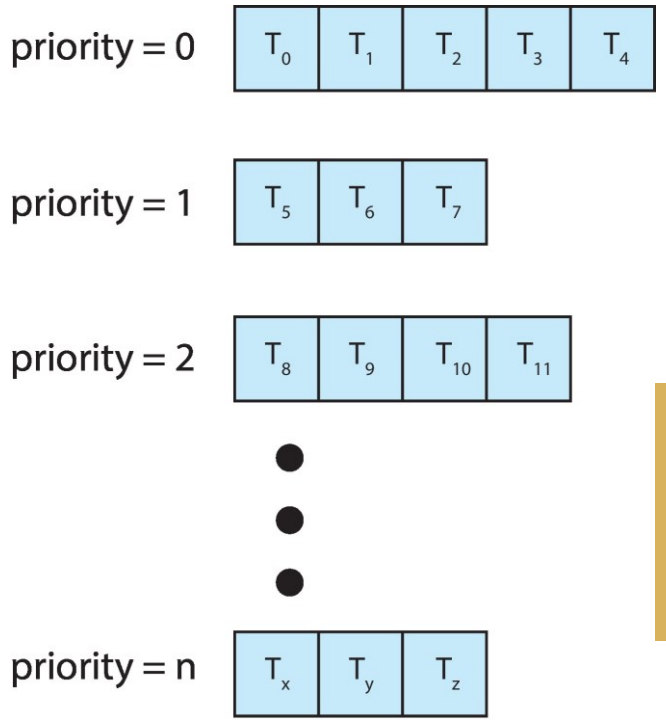
(b) 선점 예의 반환시간과 대기시간

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

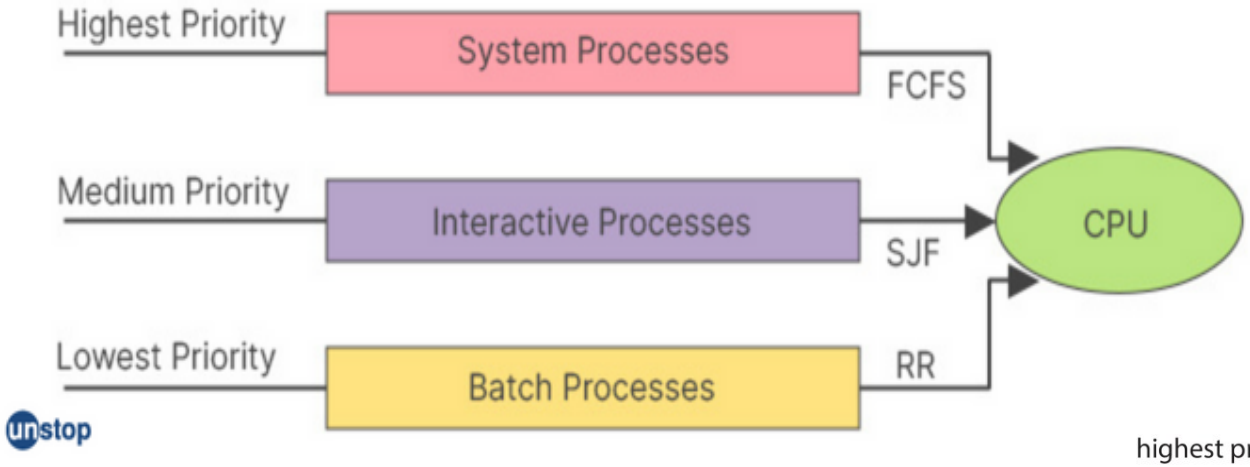
프로세스	반환시간	대기시간
P_1	10	0
P_2	(44 – 1) = 43	(16 – 1) = 15
P_3	(16 – 2) = 14	(10 – 2) = 8
P_4	(62 – 3) = 59	(58 – 3) = 55
P_5	(58 – 4) = 54	(44 – 4) = 40
평균 반환시간 : 36[= (10 + 43 + 14 + 59 + 54)/5]		평균 대기시간 : 23.6[= (0 + 15 + 8 + 55 + 40)/5]

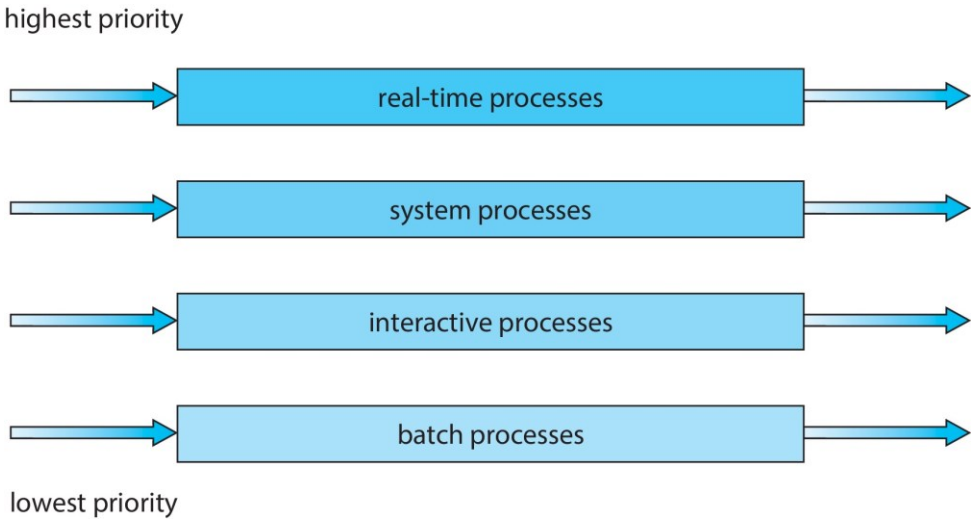
- With priority scheduling, have separate queues for each priority.
- Schedule the process in the highest-priority queue!



- 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



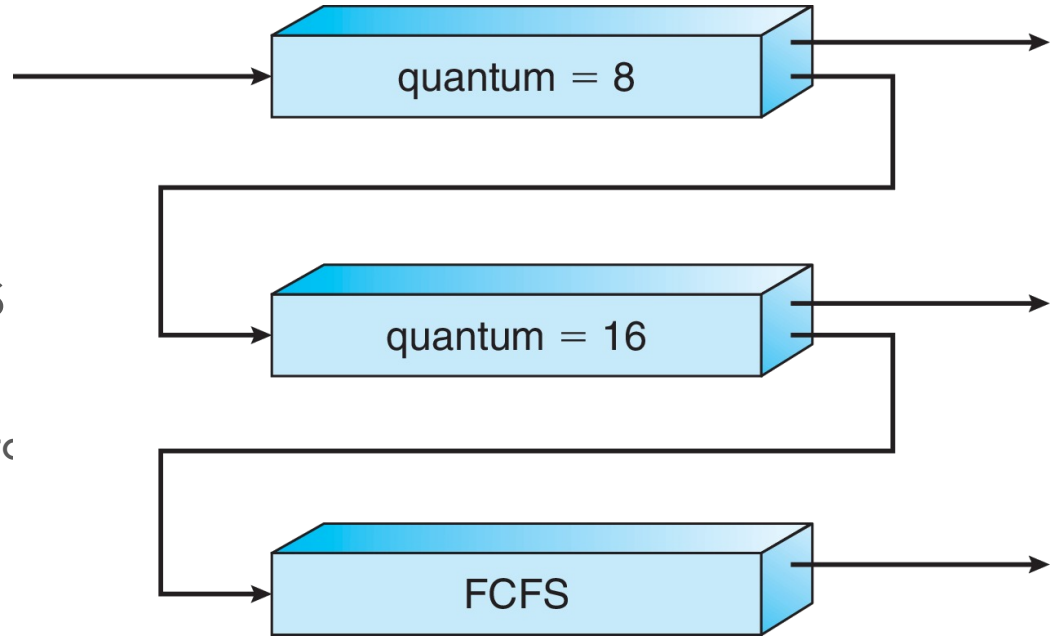


Multilevel Feedback Queue

- 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
  - 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

Example of Multilevel Feedback Queue

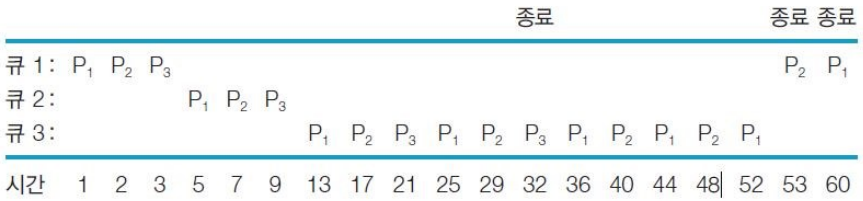
- Three queues:
  - Q0 – RR with time quantum 8 milliseconds
  - Q1 – RR time quantum 16 milliseconds
  - Q2 – FCFS



- 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
  - 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

프로세스	실행 시간
P_1	30
P_2	20
P_3	10

(a) 준비 큐



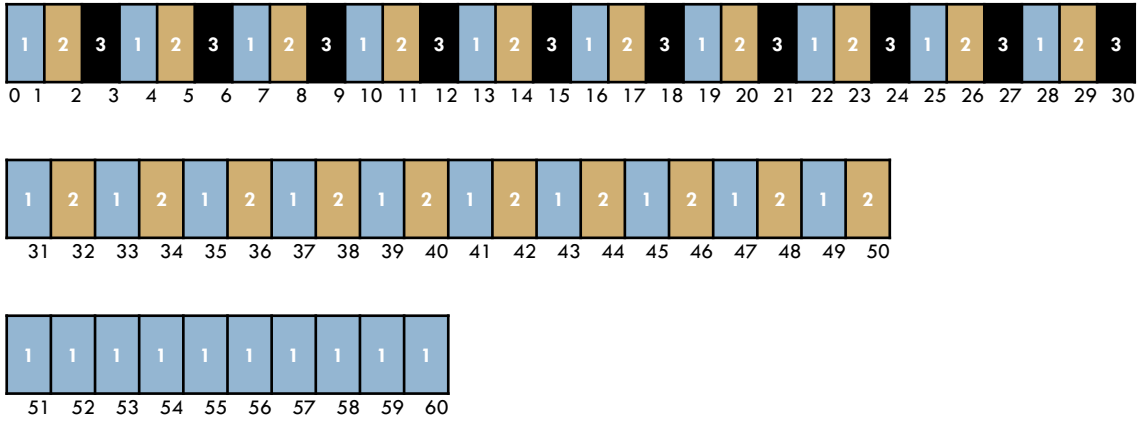
(b) 간트 차트

(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]

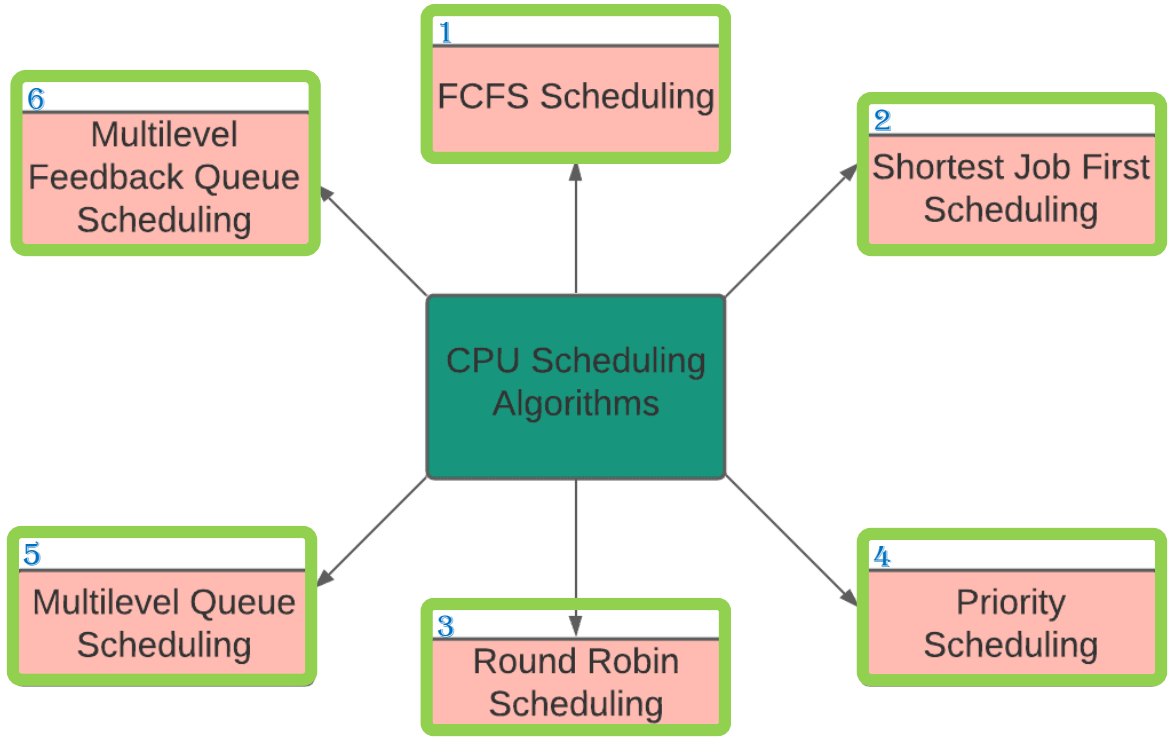


End of Chapter 5

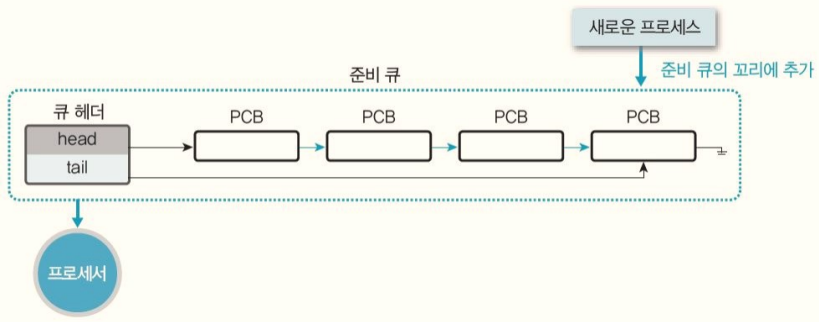
CHAPTER 5
CPU SCHEDULING

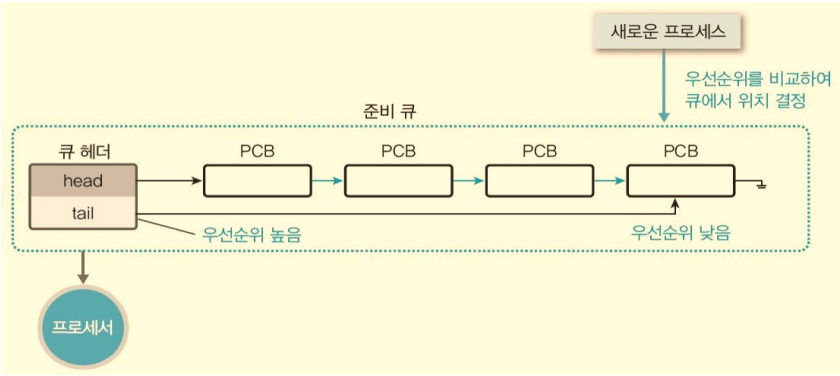
Scheduling Algorithm Optimization Criteria

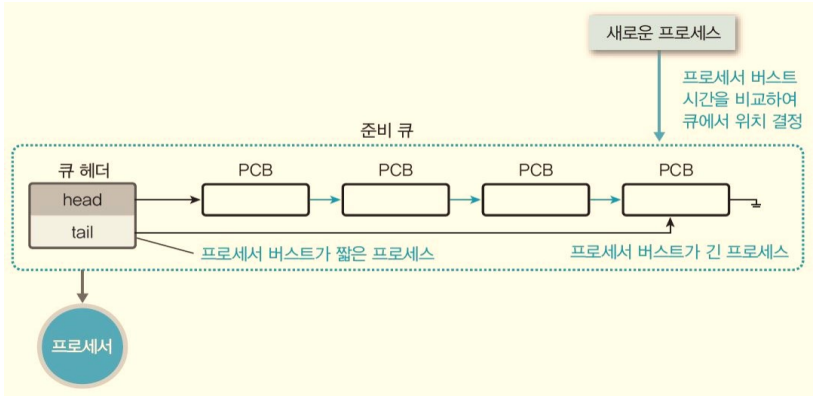
- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

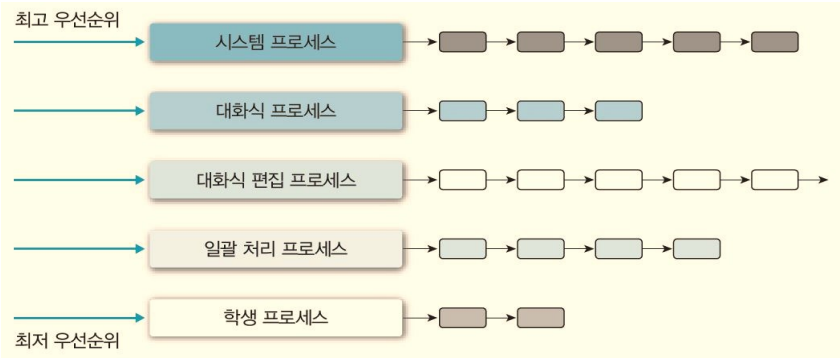


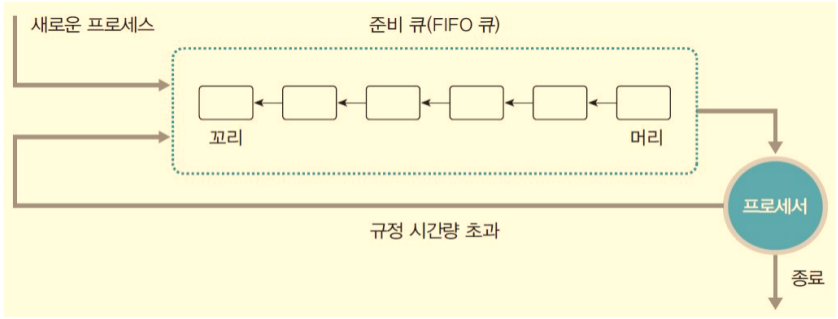
Scheduling Algorithm

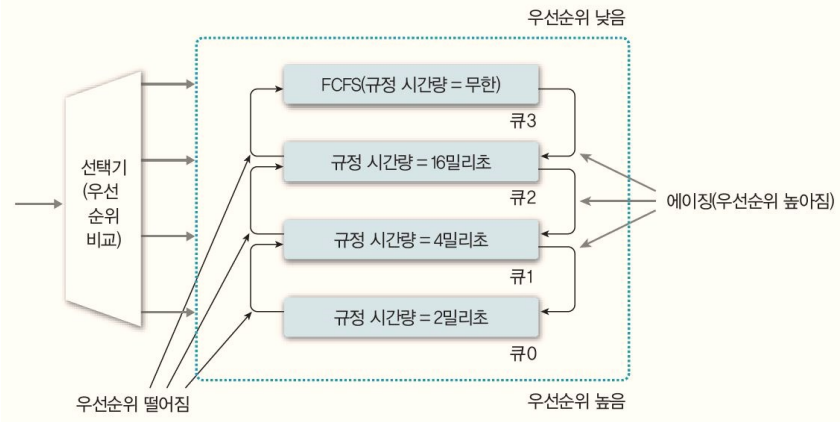












First- Come, First-Served (FCFS) Scheduling

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:

- 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:

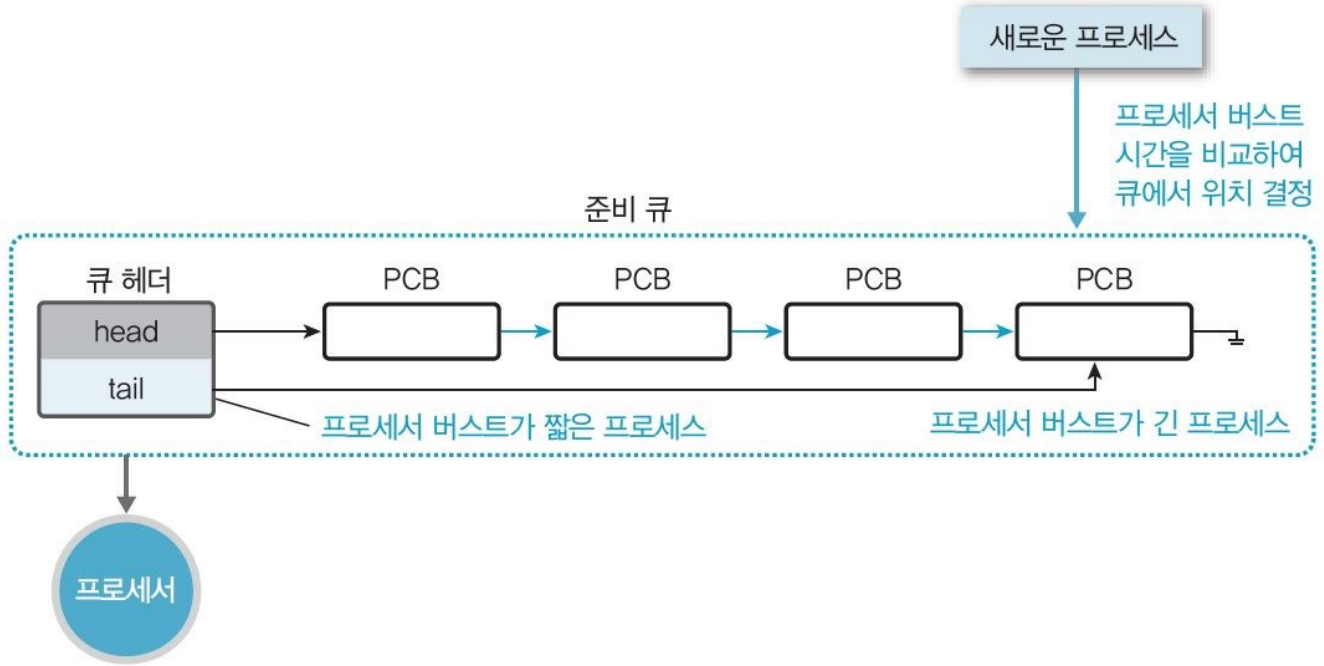
$$P_2\ ,\ P_3\ ,\ P_1$$

- 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

Shortest-Job-First (SJF) Scheduling

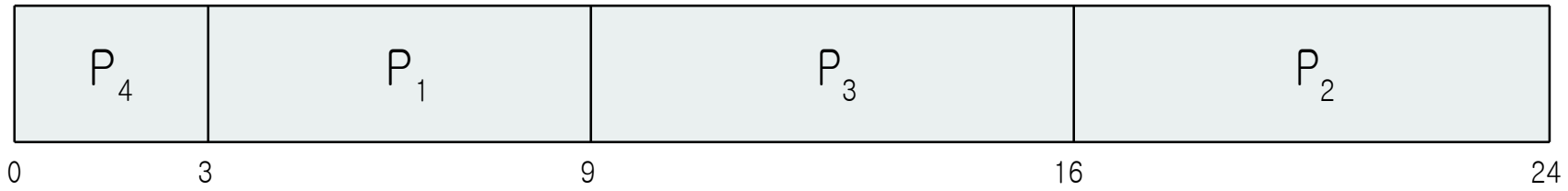
- Associate with each process the length of its next CPU burst
  - Use these lengths to schedule the process with the shortest time
- SJF is optimal – gives minimum average waiting time for a given set of processes
  - The difficulty is knowing the length of the next CPU request
  - Could ask the user



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
- 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. τ_{n+1} = predicted value for the next CPU burst
3. α, $0 \le \alpha \le 1$
4. Define: $\tau_{n+1} = \alpha\, t_n + (1-\alpha)\tau_n$.

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

Prediction of the Length of the Next CPU Burst

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

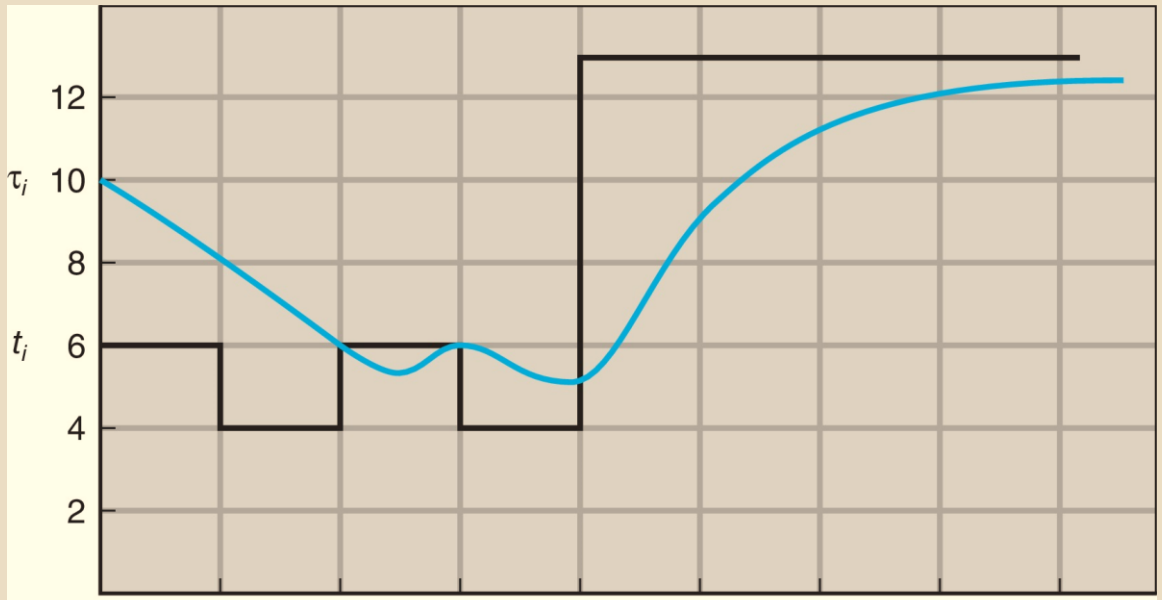
$$\tau_0 = 10$$

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

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

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

$$\vdots$$



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

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

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:

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

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

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

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

$$\vdots$$

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

$$\cdots$$

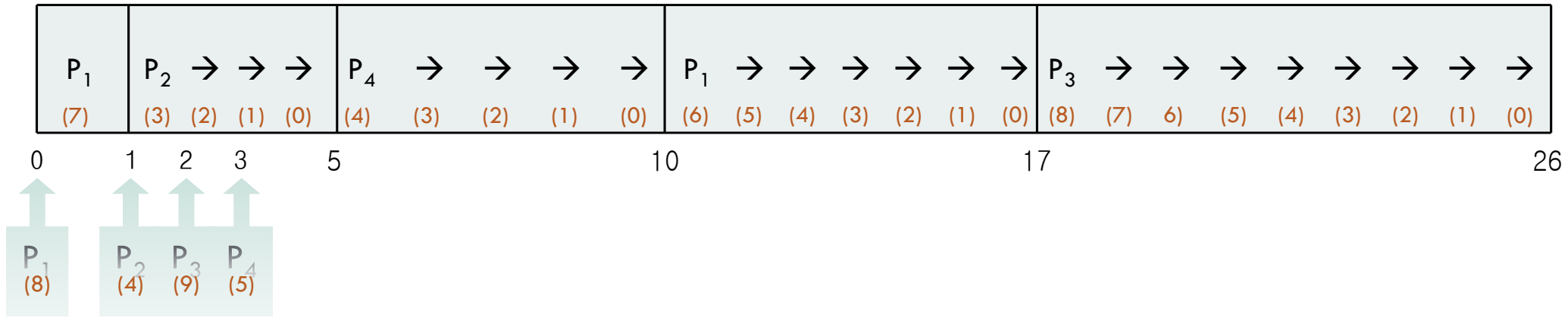
- Since both α and $(1 - \alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor

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

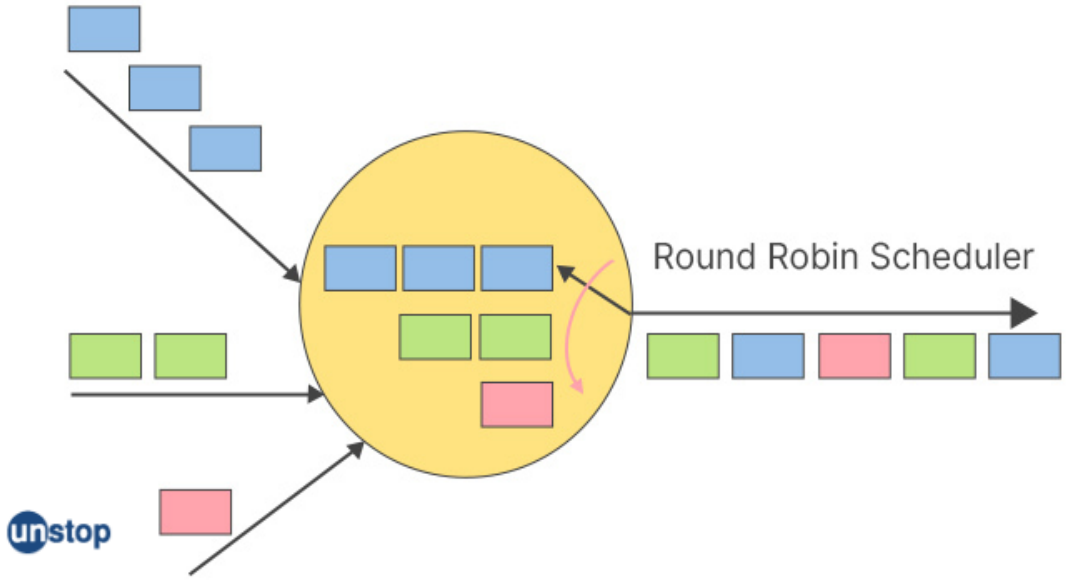
- *Preemptive* SJF Gantt Chart



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

Round Robin (RR)

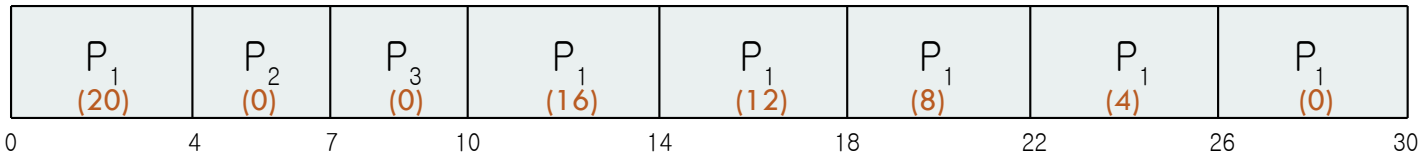
- Each process gets a small unit of CPU time (**time quantum** q), usually 10~100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are n processes in the ready queue and the time quantum is q, then each process gets $1/n$ of the CPU time in chunks of at most q time units at once. No process waits more than $(n-1)q$ time units.
- Timer interrupts every quantum to schedule next process
- Performance
  - q large $\Rightarrow$ FIFO
  - q small $\Rightarrow$ q must be large with respect to context switch, otherwise overhead is too high



Example of RR with Time Quantum = 4

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

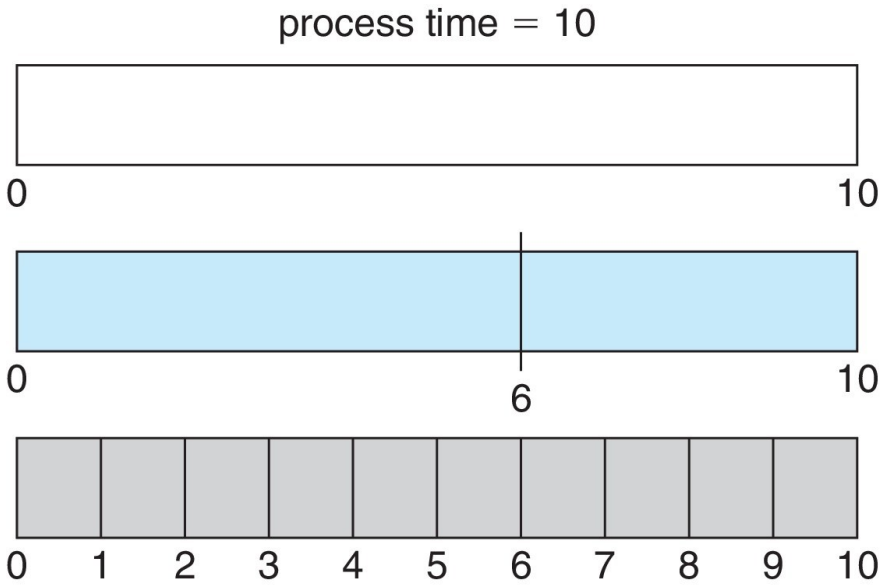
- The Gantt chart is:



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

- Typically, higher average turnaround than SJF, but better ***response***
- q should be large compared to context switch time
- q usually 10ms to 100ms, context switch < 10 μsec

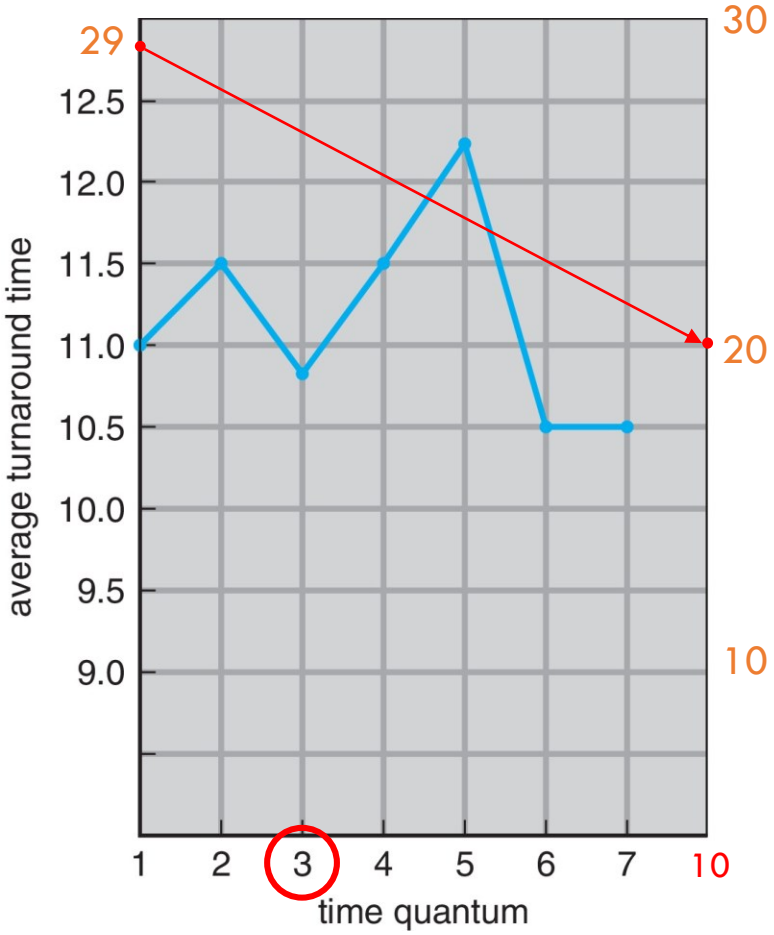
Time Quantum and Context Switch Time



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

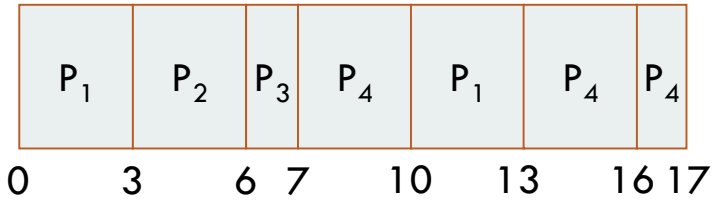
- Each process gets a small unit of CPU time (time quantum q), usually 10~100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- q usually 10*ms* to 100*ms*, context switch < 10 μsec

Turnaround Time Varies With The Time Quantum



process	time
P_1	6
P_2	3
P_3	1
P_4	7

10
10
10

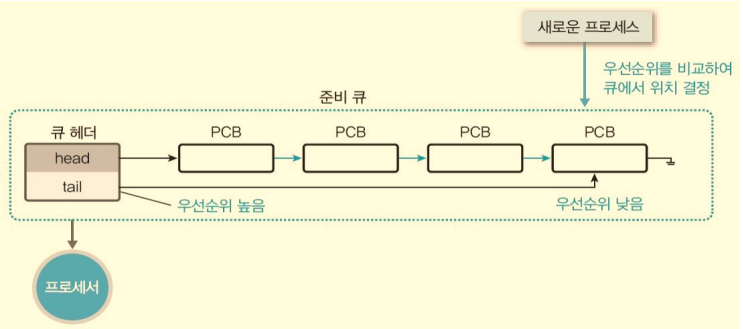


	Turn around Time
P1	13
P2	6
P3	7
P4	17
ATA	**10.75**

- 80% of CPU bursts should be shorter than q

Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority, 3bit or 14bit)
  - Preemptive
  - Nonpreemptive
- SJF is priority scheduling where priority is the inverse of predicted next CPU burst time
- Problem ≡ **Starvation** – low priority processes may never execute
- Solution ≡ **Aging** – as time progresses increase the priority of the process

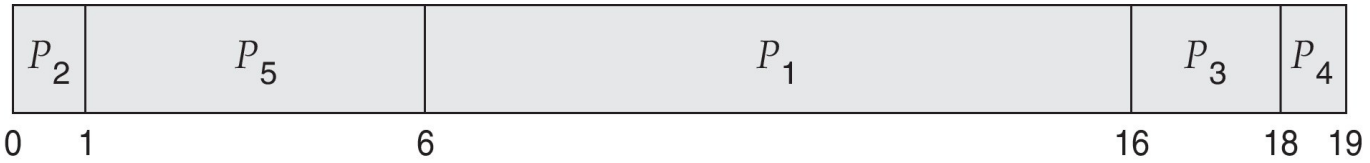


우선순위가 동일한 프로세스들은 FCFS 순서로 스케줄링

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

- Priority scheduling Gantt Chart

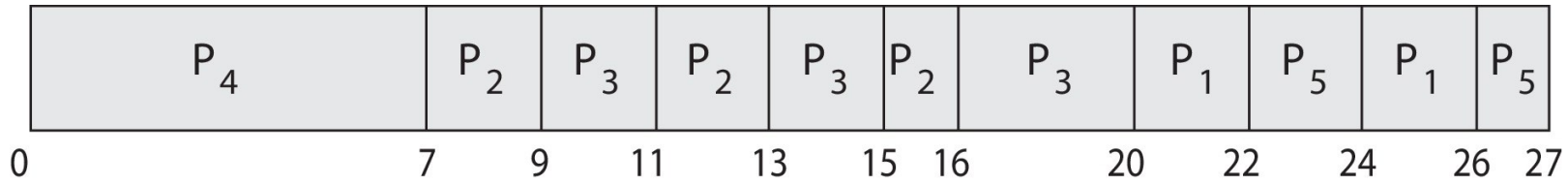


- Waiting time
  - $6+0+16+18+1=41\text{msec}$
- Average waiting time = 8.2 msec

Priority Scheduling w/ Round-Robin

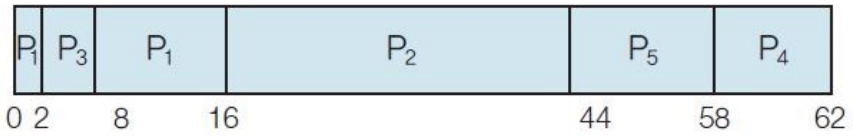
Process	Burst Time	Priority
P_1	4	3
P_2	5	2
P_3	8	2
P_4	7	1
P_5	3	3

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

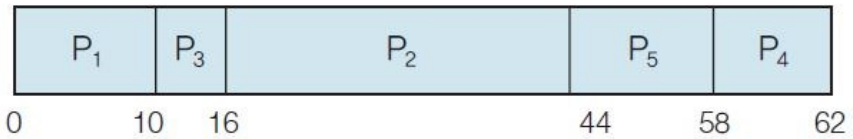


프로세스	도착 시간	실행 시간	우선순위
P_1	0	10	2
P_2	1	28	3
P_3	2	6	1
P_4	3	4	4
P_5	4	14	3

(a) 준비 큐



(b) 선점 우선순위 간트 차트



(c) 비선점 우선순위 간트 차트

표 6-7 그림 6-26에서 (b) 선점 예의 반환시간과 대기시간

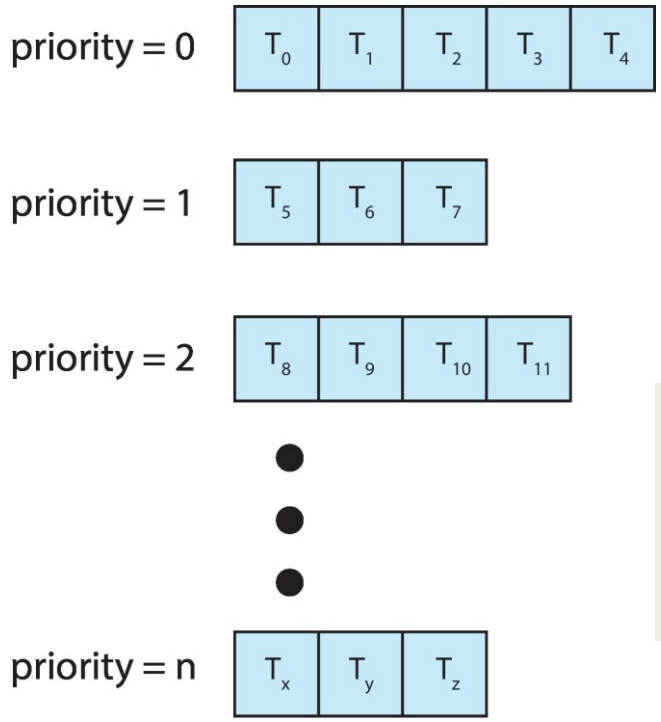
프로세스	반환시간	대기시간
P_1	(16 − 0) = 16	(8 − 2) = 6
P_2	(44 − 1) = 43	(16 − 1) = 15
P_3	(8 − 2) = 6	(2 − 2) = 0
P_4	(62 − 3) = 59	(58 − 3) = 55
P_5	(58 − 4) = 54	(44 − 4) = 40
평균 반환시간 : 35.6[= (16 + 43 + 6 + 59 + 54/5]		평균 대기시간 : 23.2[= (6 + 15 + 0 + 55 + 40)/5]

(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]

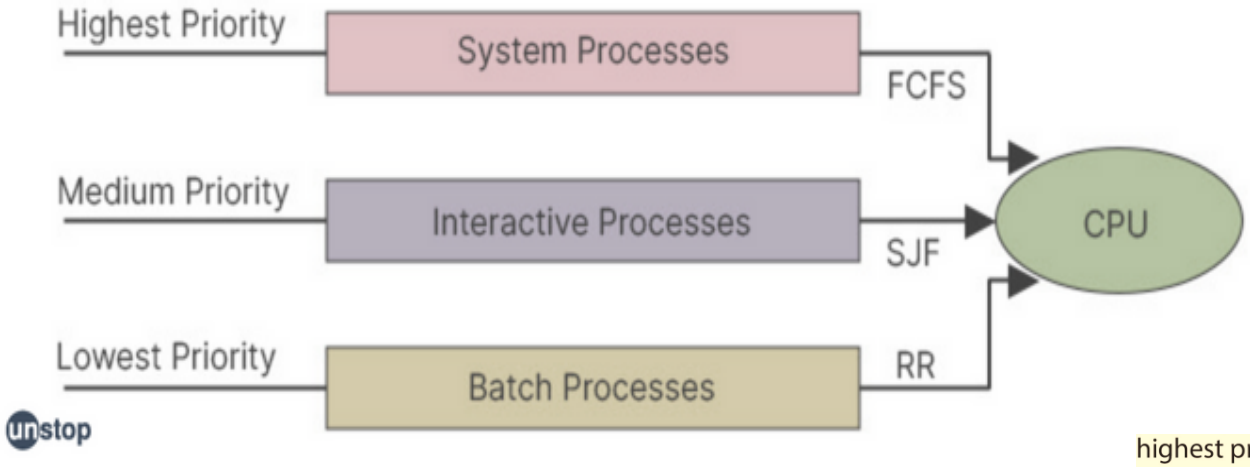
Multilevel Queue

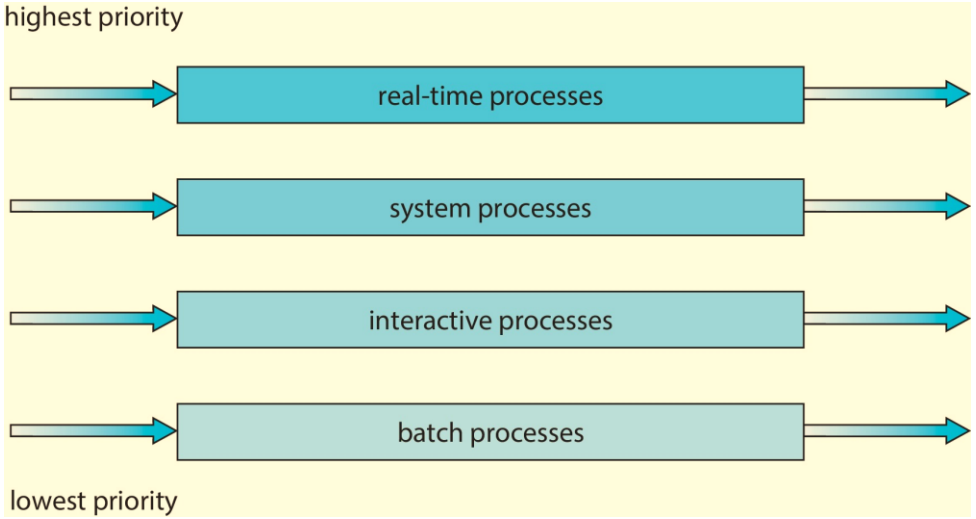
- With priority scheduling, have separate queues for each priority.
- Schedule the process in the highest-priority queue!



- 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



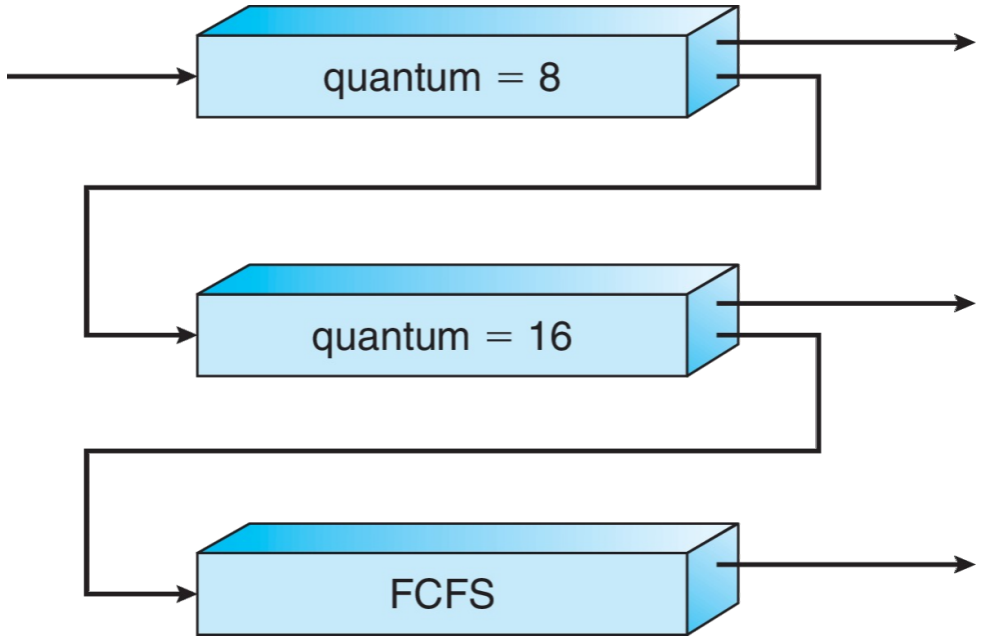


Multilevel Feedback Queue

- 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
  - 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

Example of Multilevel Feedback Queue

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

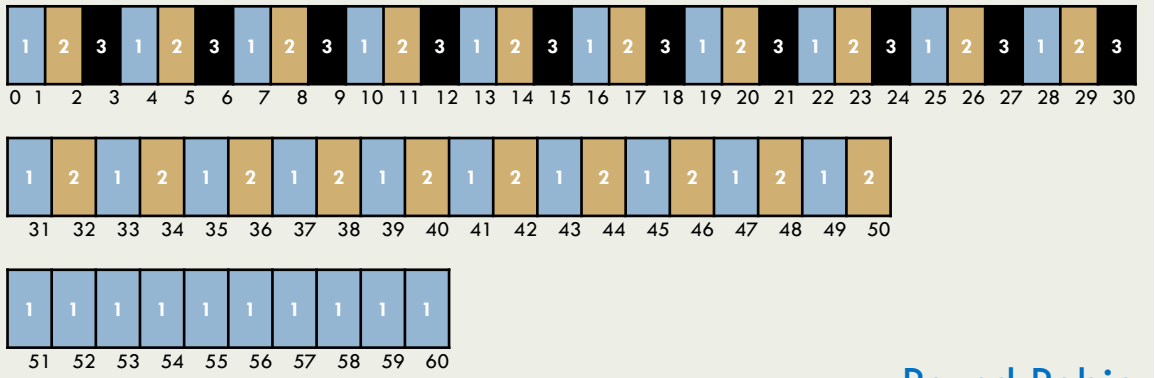


- Scheduling
  - A new job enters queue Q0 which is served FCFS
    - When it gains CPU, job receives 8 milliseconds
    - If it does not finish in 8 milliseconds, job is moved to queue Q1
  - At Q1 job is again served FCFS and receives 16 additional milliseconds
    - If it still does not complete, it is preempted and moved to queue Q2

프로세스	실행 시간
P_1	30
P_2	20
P_3	10

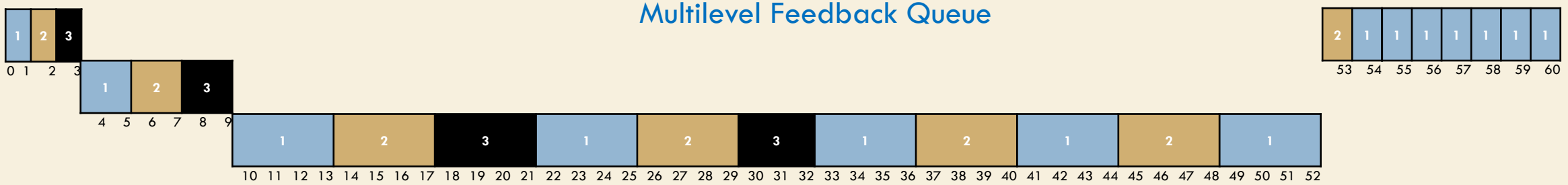
(a) 준비 큐

(b) 간트 차트



Round Robin

Multilevel Feedback Queue



(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

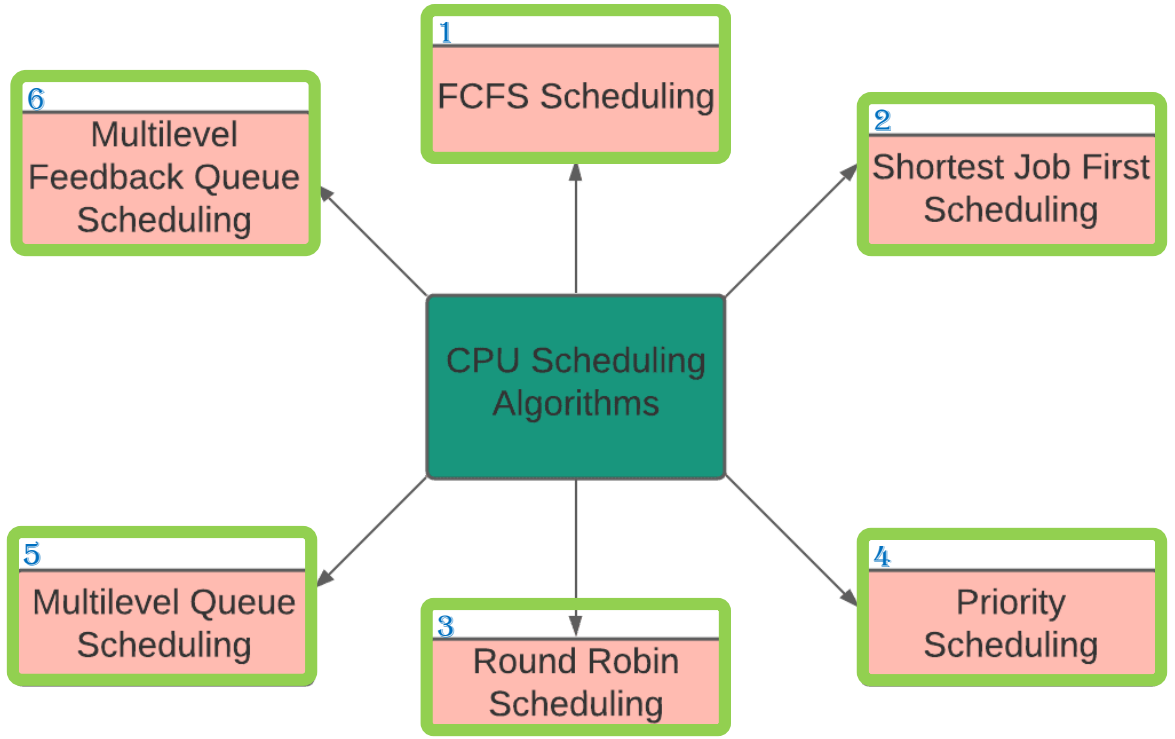
프로세스	라운드 로빈^{RR} 스케줄링	다단계 피드백 큐^{MLFQ} 스케줄링
P_1	30	(53 − 23) = 30
P_2	30	(52 − 19) = 33
P_3	20	(29 − 7) = 22
평균 대기시간 : 30[= (30 + 30 + 30)/3]		평균 대기시간 : $28\frac{1}{3}$[= (30 + 33 + 22)/3]

End of Chapter 5

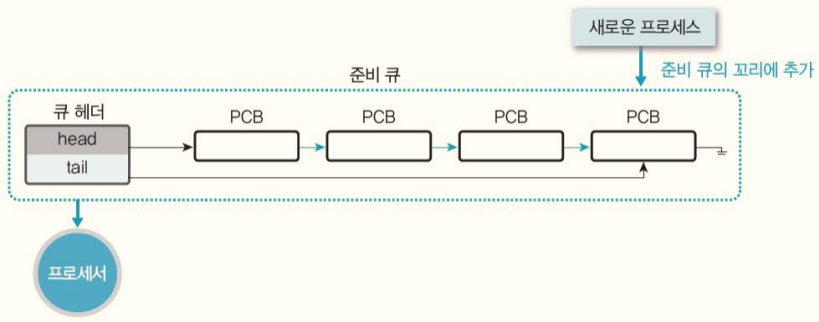
CHAPTER 5
CPU SCHEDULING

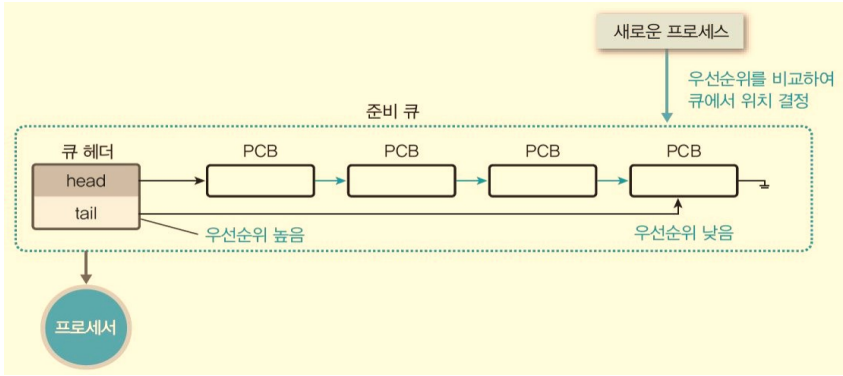
Scheduling Algorithm Optimization Criteria

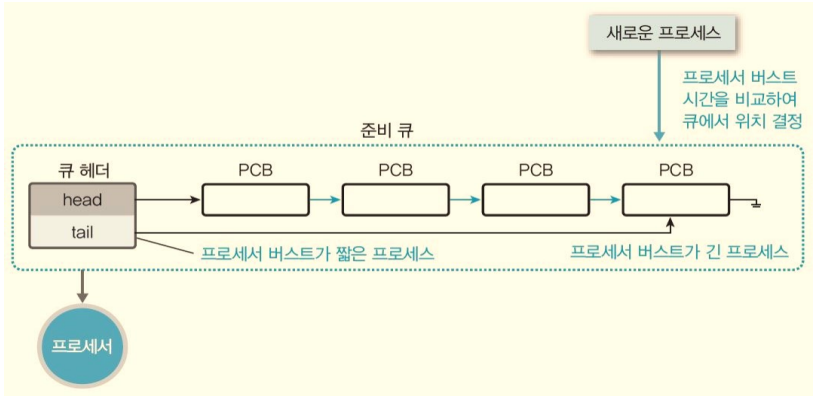
- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

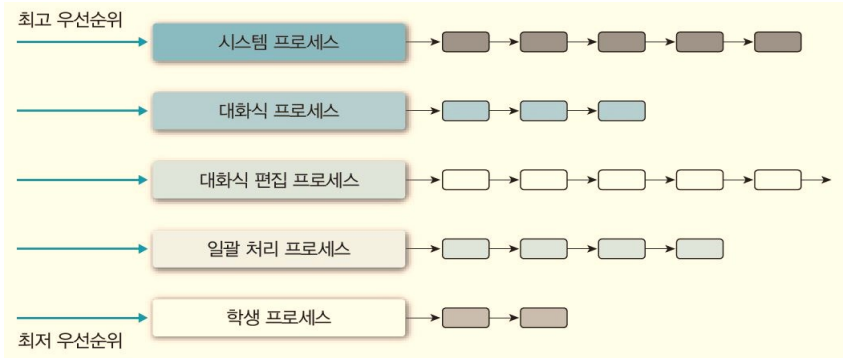


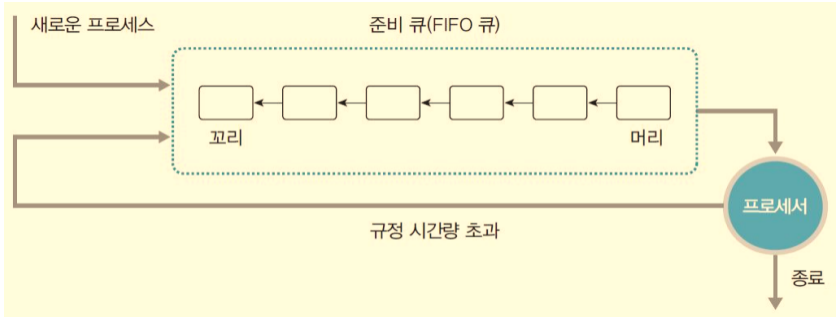
Scheduling Algorithm

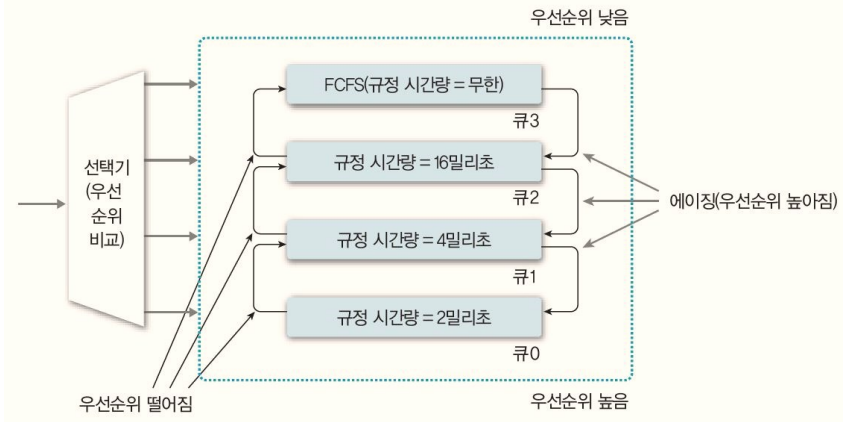












First- Come, First-Served (FCFS) Scheduling

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:

- 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:

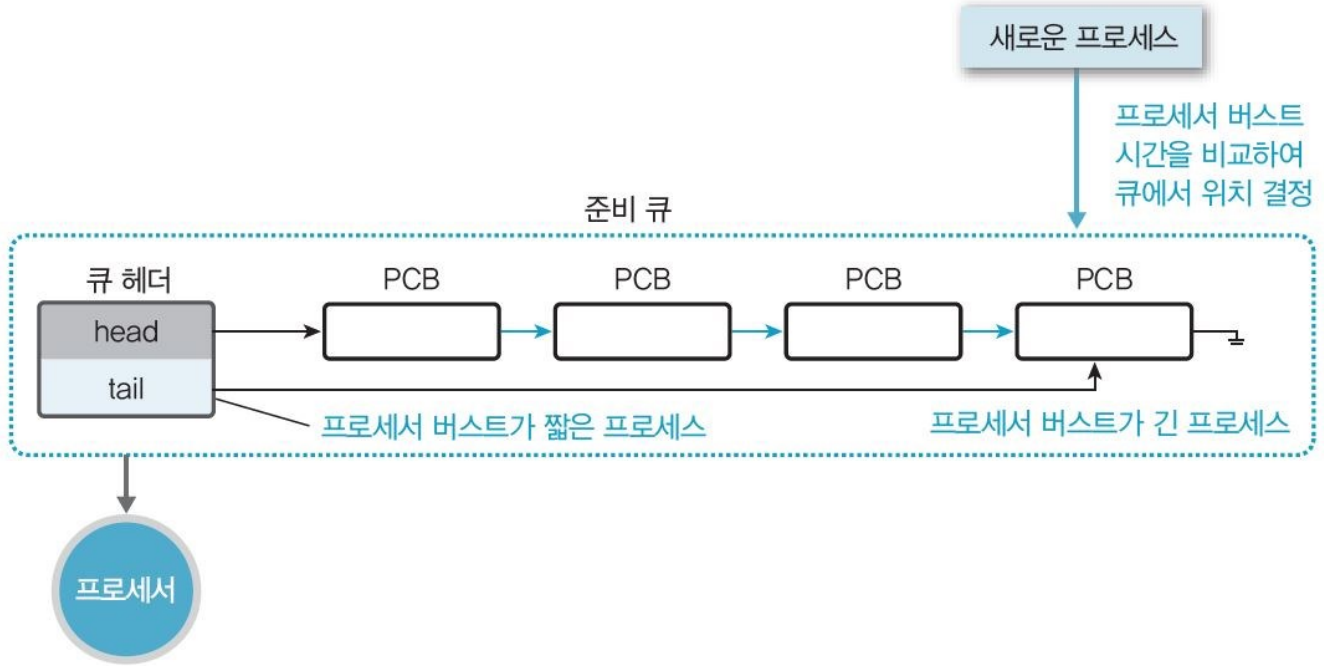
$$P_2\ ,\ P_3\ ,\ P_1$$

- 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

Shortest-Job-First (SJF) Scheduling

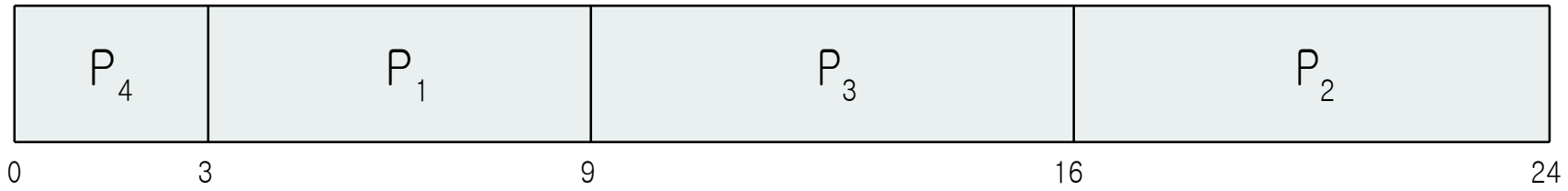
- Associate with each process the length of its next CPU burst
  - Use these lengths to schedule the process with the shortest time
- SJF is optimal – gives minimum average waiting time for a given set of processes
  - The difficulty is knowing the length of the next CPU request
  - Could ask the user



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

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

Prediction of the Length of the Next CPU Burst

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

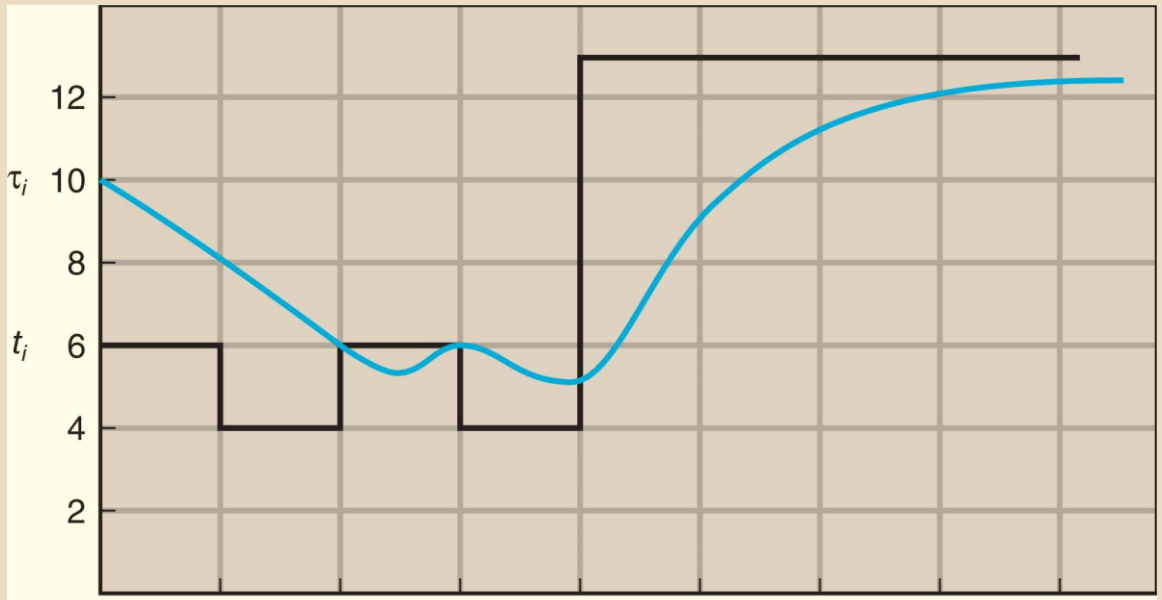
$$\tau_0 = 10$$

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

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

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

$$\vdots$$



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

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

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:

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

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

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

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

$$\vdots$$

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

$$\cdots$$

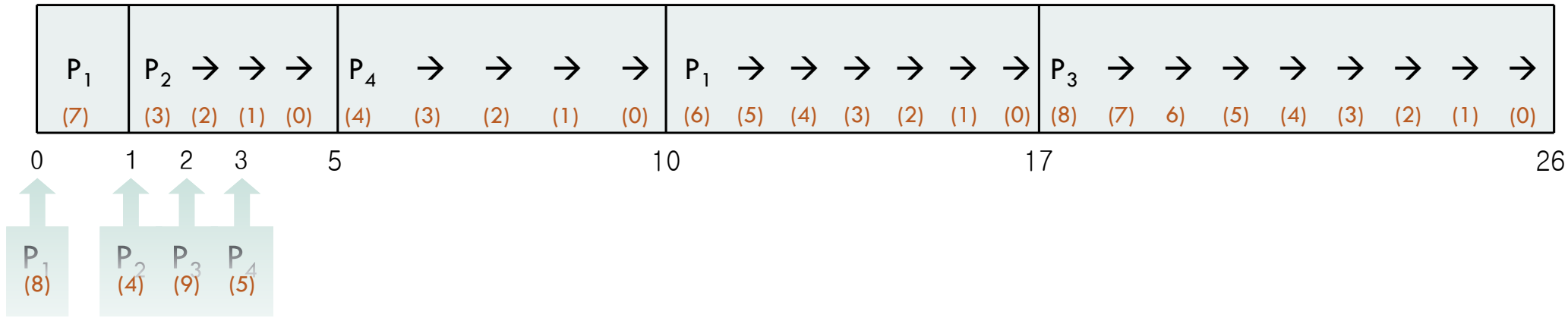
- Since both α and $(1 - \alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor

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

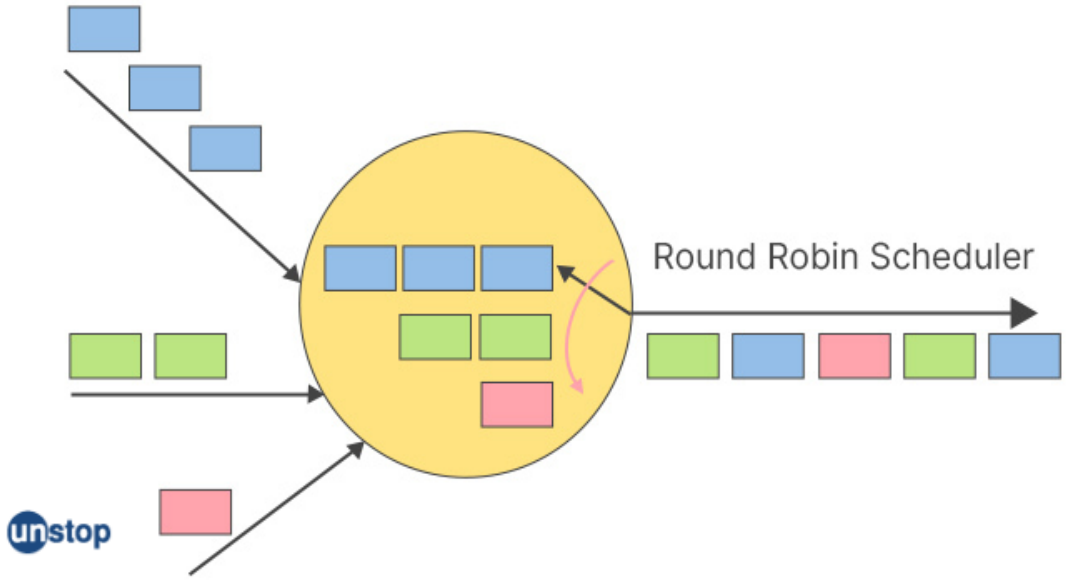
- *Preemptive* SJF Gantt Chart



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

Round Robin (RR)

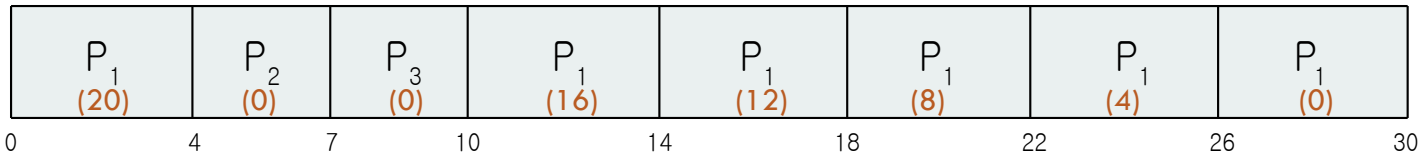
- Each process gets a small unit of CPU time (**time quantum** q), usually 10~100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are n processes in the ready queue and the time quantum is q, then each process gets $1/n$ of the CPU time in chunks of at most q time units at once. No process waits more than $(n-1)q$ time units.
- Timer interrupts every quantum to schedule next process
- Performance
  - q large $\Rightarrow$ FIFO
  - q small $\Rightarrow$ q must be large with respect to context switch, otherwise overhead is too high



Example of RR with Time Quantum = 4

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

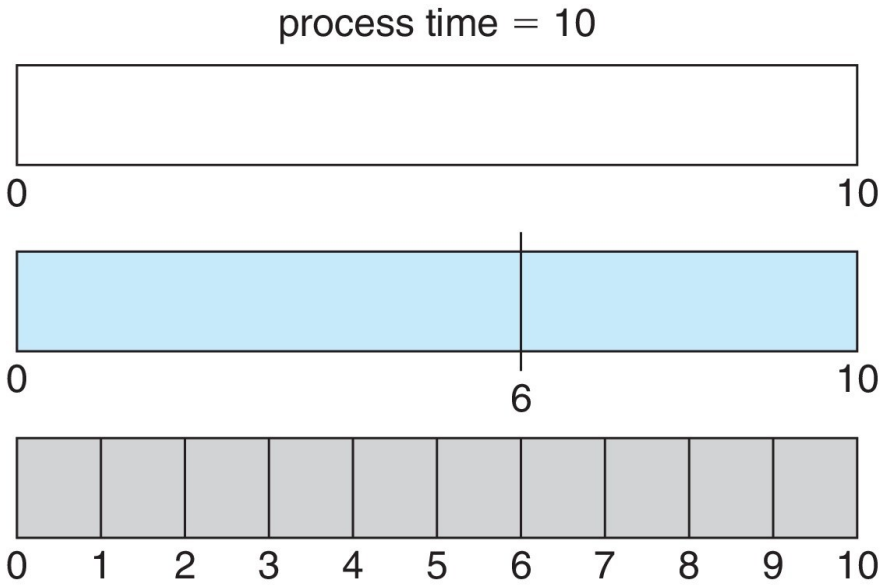
- The Gantt chart is:



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

- Typically, higher average turnaround than SJF, but better ***response***
- q should be large compared to context switch time
- q usually 10ms to 100ms, context switch < 10 μsec

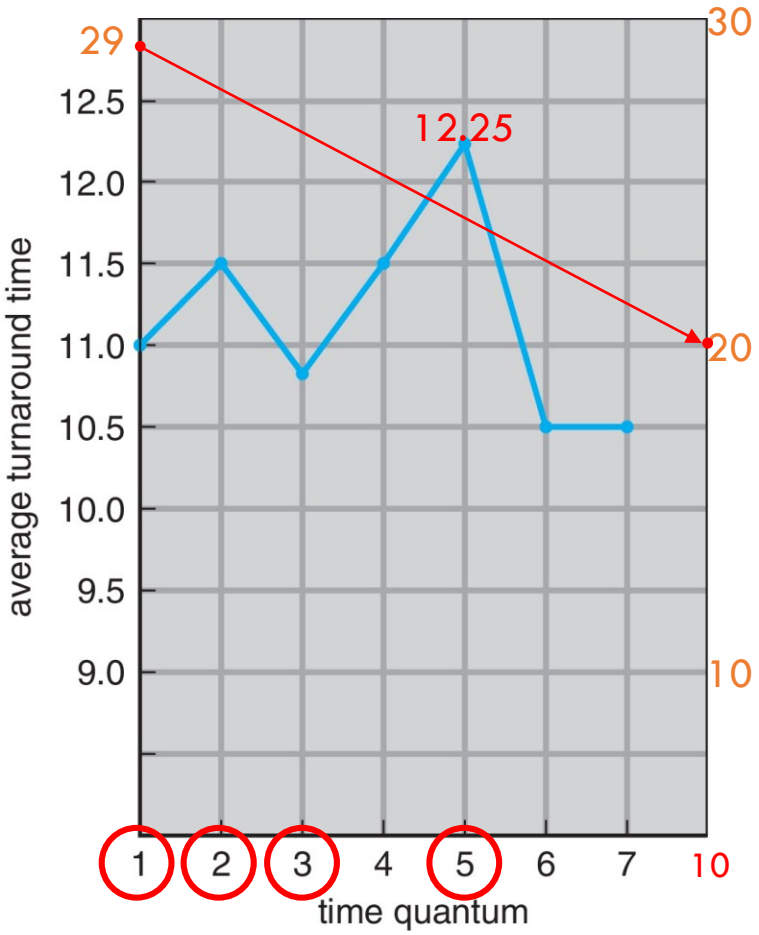
Time Quantum and Context Switch Time



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

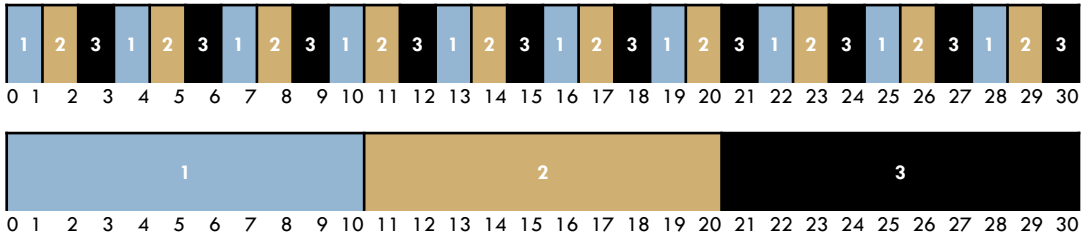
- Each process gets a small unit of CPU time (time quantum q), usually 10~100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- q usually 10*ms* to 100*ms*, context switch < 10 μsec

Turnaround Time Varies With The Time Quantum



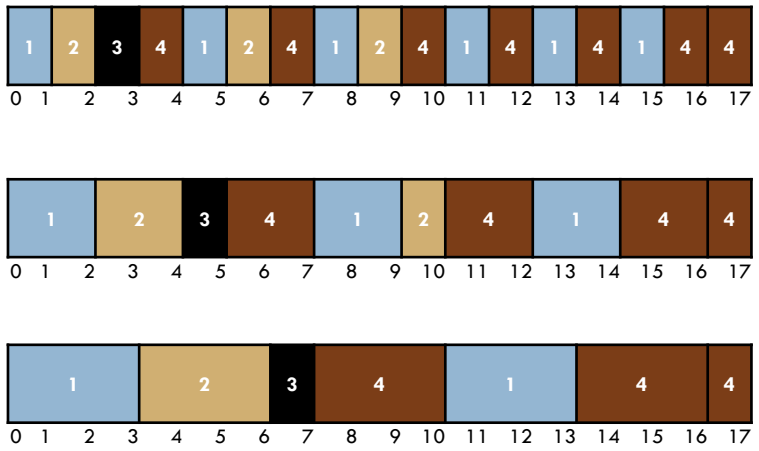
process	time
P_1	6
P_2	3
P_3	1
P_4	7

10
10
10



Q=1	Turn around Time
P1	28
P2	29
P3	30
ATA	29

Q=10	Turn around Time
P1	10
P2	20
P3	30
ATA	20

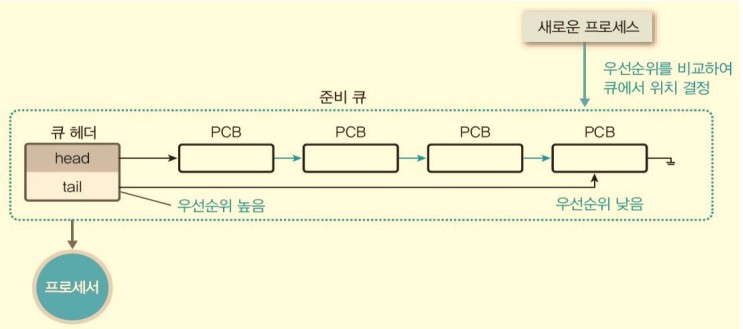


Q=3	Turn around Time
P1	13
P2	6
P3	7
P4	17
ATA	10.75

- 80% of CPU bursts should be shorter than q

Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority, 3bit or 14bit)
  - Preemptive
  - Nonpreemptive
- SJF is priority scheduling where priority is the inverse of predicted next CPU burst time
- Problem ≡ **Starvation** – low priority processes may never execute
- Solution ≡ **Aging** – as time progresses increase the priority of the process

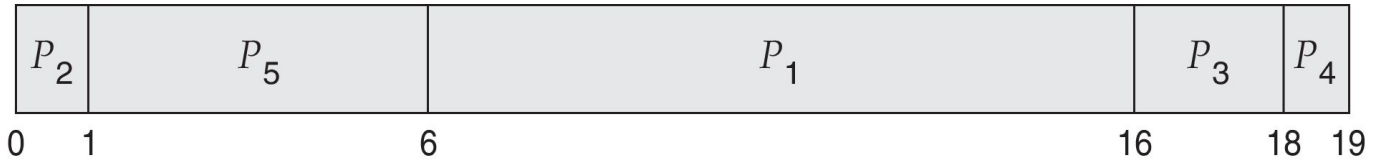


우선순위가 동일한 프로세스들은 FCFS 순서로 스케줄링

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

- Priority scheduling Gantt Chart

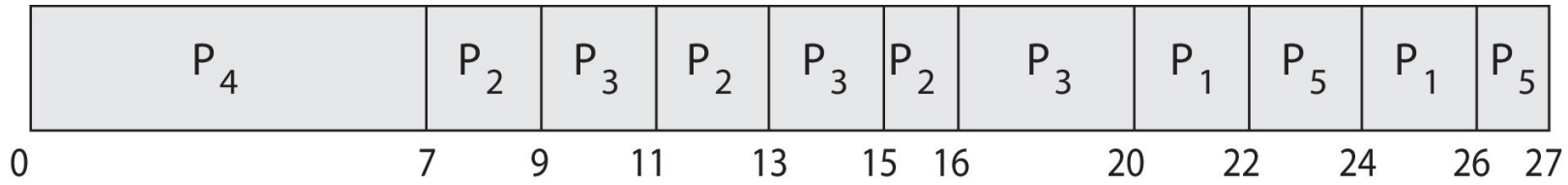


- Waiting time
  - $6+0+16+18+1=41\text{msec}$
- Average waiting time = 8.2 msec

Priority Scheduling w/ Round-Robin

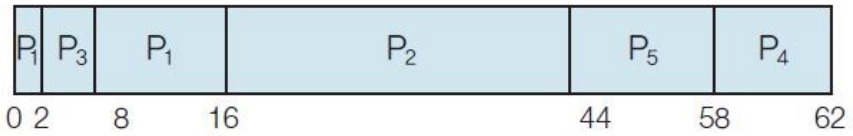
Process	Burst Time	Priority
P_1	4	3
P_2	5	2
P_3	8	2
P_4	7	1
P_5	3	3

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

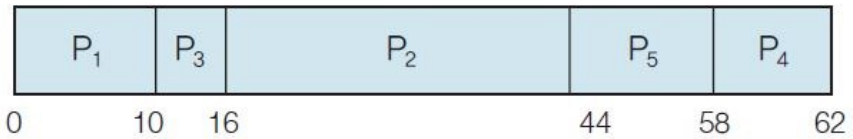


프로세스	도착 시간	실행 시간	우선순위
P_1	0	10	2
P_2	1	28	3
P_3	2	6	1
P_4	3	4	4
P_5	4	14	3

(a) 준비 큐



(b) 선점 우선순위 간트 차트



(c) 비선점 우선순위 간트 차트

표 6-7 그림 6-26에서 (b) 선점 예의 반환시간과 대기시간

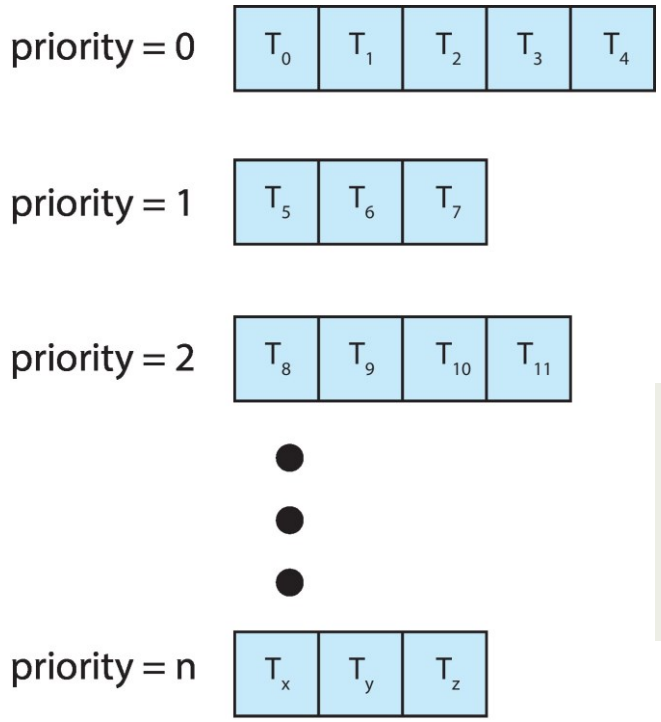
프로세스	반환시간	대기시간
P_1	(16 − 0) = 16	(8 − 2) = 6
P_2	(44 − 1) = 43	(16 − 1) = 15
P_3	(8 − 2) = 6	(2 − 2) = 0
P_4	(62 − 3) = 59	(58 − 3) = 55
P_5	(58 − 4) = 54	(44 − 4) = 40
평균 반환시간 : 35.6[= (16 + 43 + 6 + 59 + 54/5]		평균 대기시간 : 23.2[= (6 + 15 + 0 + 55 + 40)/5]

(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]

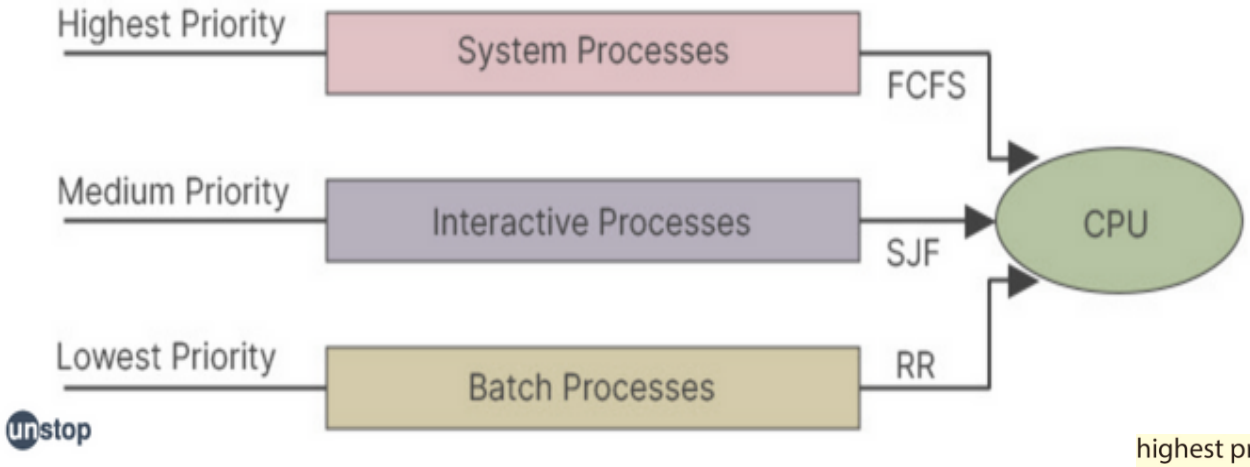
Multilevel Queue

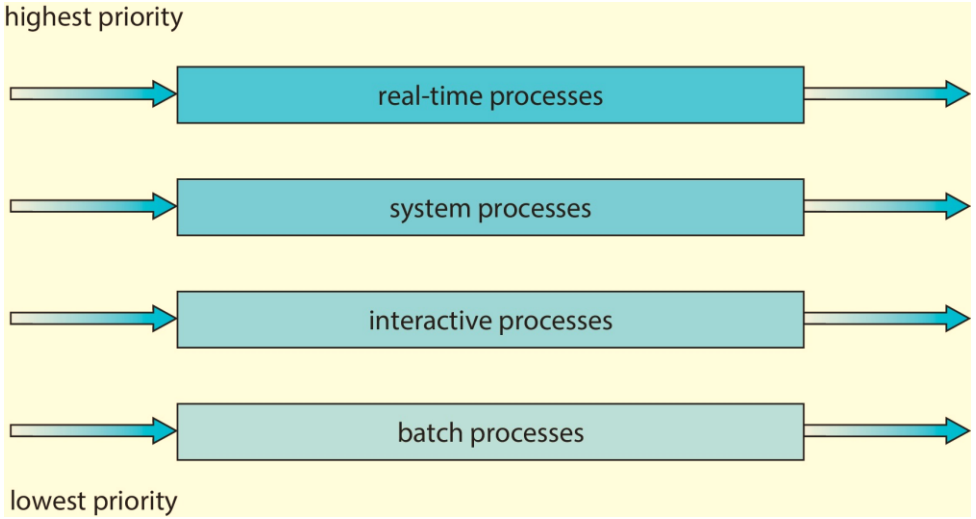
- With priority scheduling, have separate queues for each priority.
- Schedule the process in the highest-priority queue!



- 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

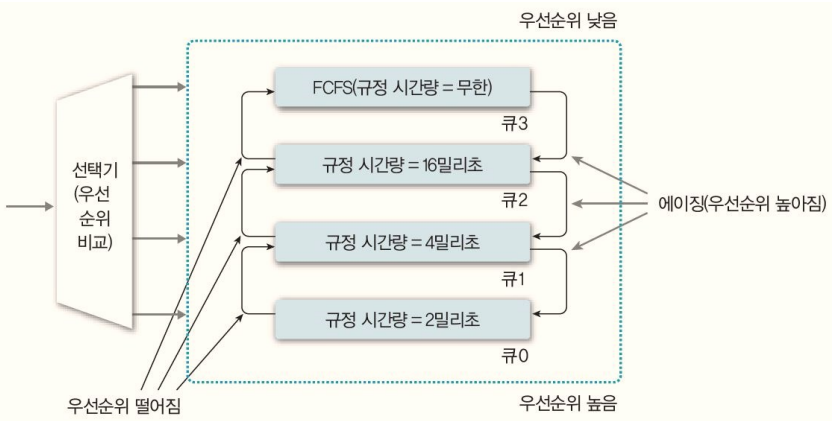
- Prioritization based upon process type





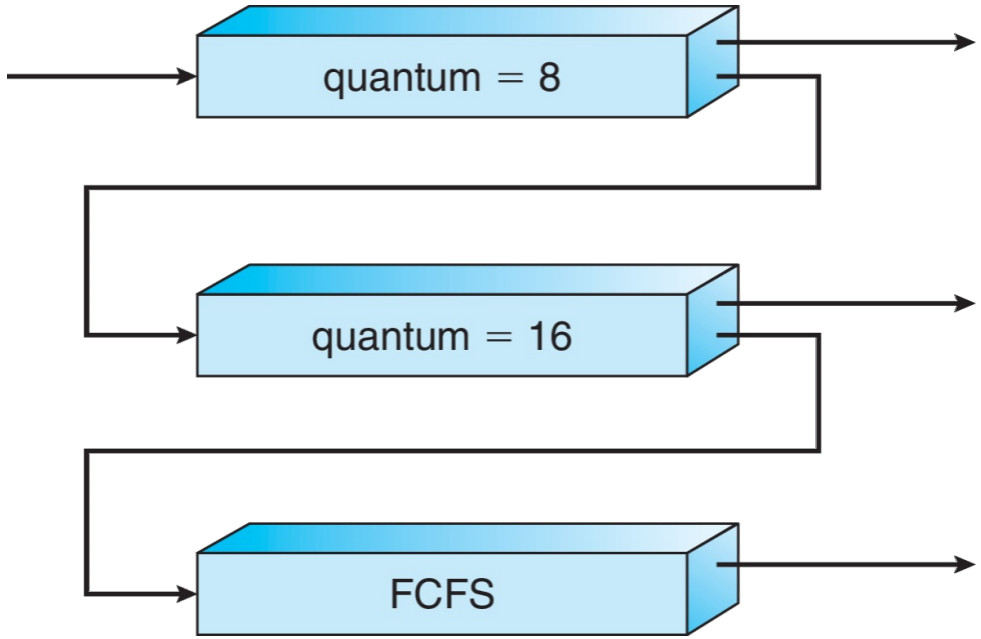
Multilevel Feedback Queue

- 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
  - 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



Example of Multilevel Feedback Queue

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



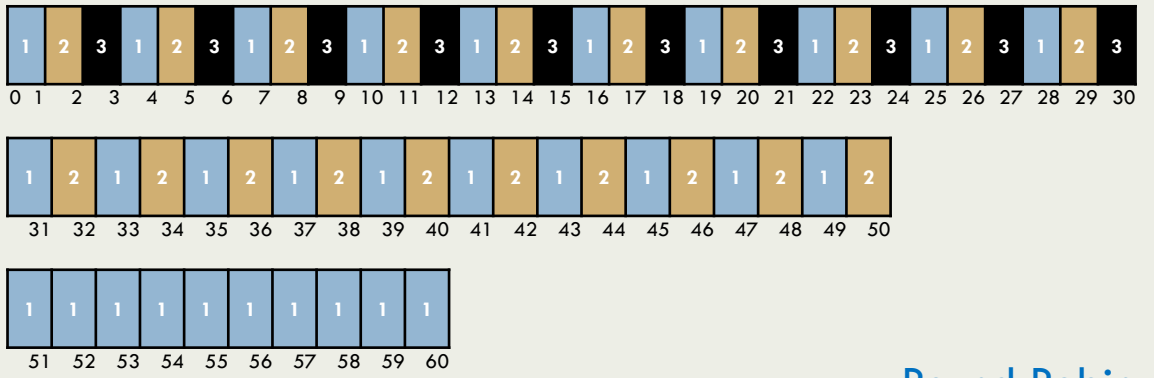
- Scheduling
  - A new job enters queue Q0 which is served FCFS
    - When it gains CPU, job receives 8 milliseconds
    - If it does not finish in 8 milliseconds, job is moved to queue Q1
  - At Q1 job is again served FCFS and receives 16 additional milliseconds
    - If it still does not complete, it is preempted and moved to queue Q2

프로세스	실행 시간
P_1	30
P_2	20
P_3	10

(a) 준비 큐

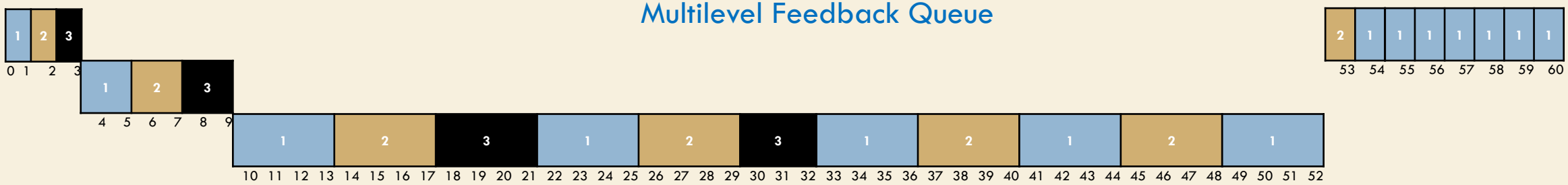
												종료							종료	종료
큐 1:	P_1	P_2	P_3																P_2	P_1
큐 2:				P_1	P_2	P_3														
큐 3:							P_1	P_2	P_3	P_1	P_2	P_3	P_1	P_2	P_1	P_2	P_1			
시간	1	2	3	5	7	9	13	17	21	25	29	32	36	40	44	48	52	53	60	

(b) 간트 차트



Round Robin

Multilevel Feedback Queue



(a) 반환시간

프로세스	라운드 로빈^{RR} 스케줄링	다단계 피드백 큐^{MLFQ} 스케줄링
P_1	60	60
P_2	50	53
P_3	30	32
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]

End of Chapter 5