운영체제

2019.7.4

컴퓨터공학과 이병문
가천대학교 Gachon University

강의일정

- 7/2 Deadlock
- 3 CPU스케줄링1(온라인수행과제
- 4 CPU스케줄링2
- 5 메모리관리1
- 7/8 중간고사2

스케줄링 알고리즘

- Multi-Level Q 스케줄링
- Multi-Level Feedback Q
- HRN 스케줄링

프로세스 관리

- 쓰레드 스케줄링
- 다중프로세서 스케줄링
- 사례연구
  Windows, Unix/Linux

스케줄링 알고리즘

■ Multilevel Queue Scheduling

☑ **서로 다른 유형별로 구분하여 분리처리**
(**예**, **대화형**(Foreground) job > **일괄처리**(Background) job)

☑ **유형에 맞는 스케줄링 알고리즘을 각각 따로 적용**
(Foreground job -> Round-Robin, Background->FCFS)

레벨별 Ready Queue

Priority Scheduling

Round-Robin Scheduling

FCFS Scheduling

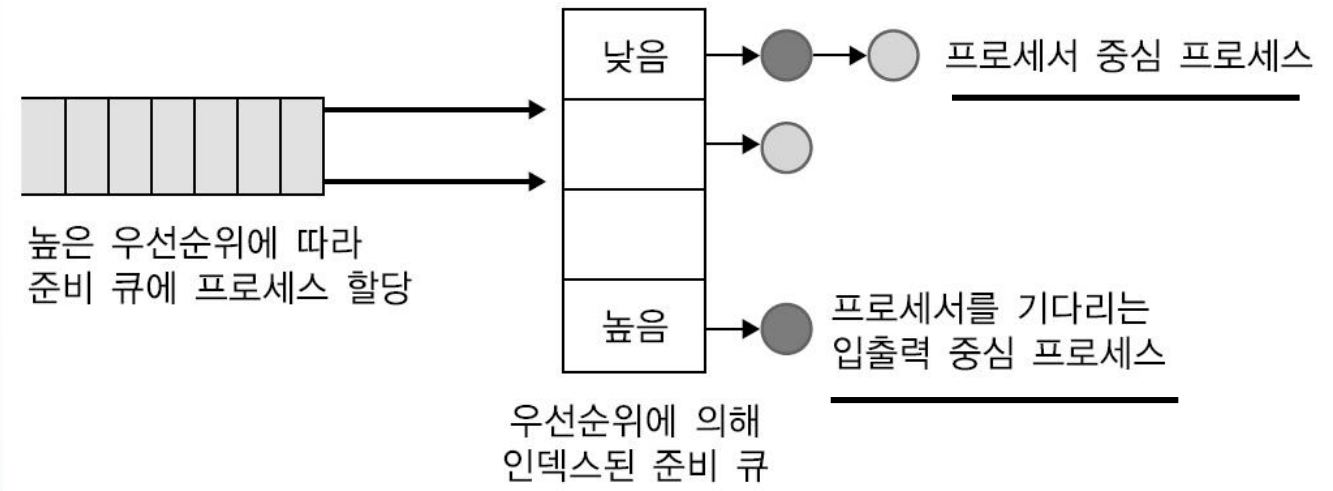
스케줄링 알고리즘

■ Multilevel Feedback Queue Scheduling

① Multilevel Queue Scheduling 은 한번 결정된 프로세스는 Queue를 바꿀 수 없어 신축성 있는 스케줄링이 어렵다.

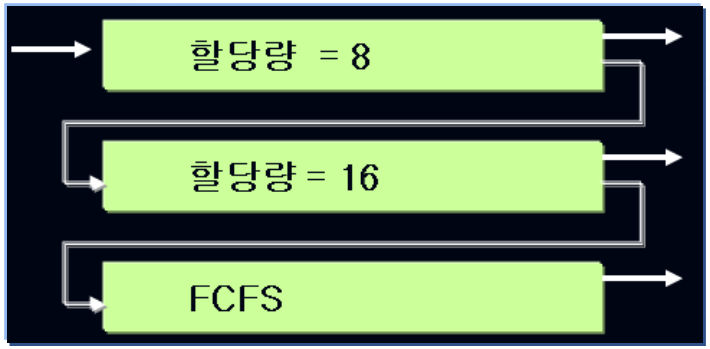
② 다이나믹한 스케줄링상황을 반영하여 프로세스가 Queue 간에 서로 이동하여 알고리즘의 최적을 꾀함

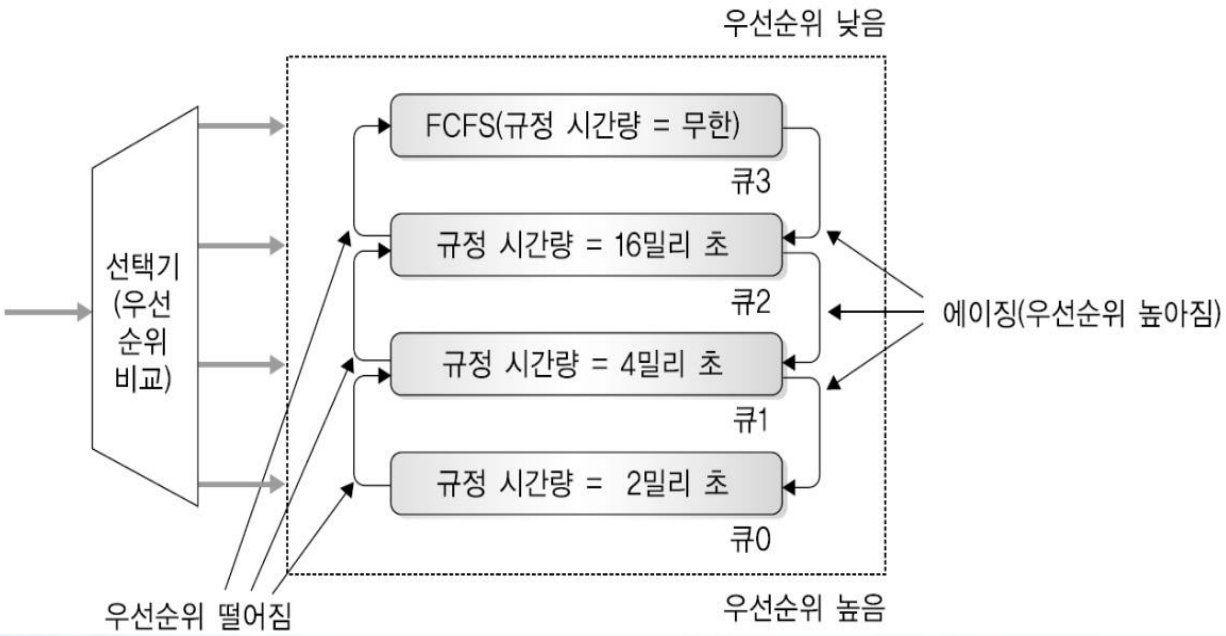
③ 최상의 스케줄러를 정의하기 위한 고려사항이 많아 복잡하다.



스케줄링 알고리즘

■ Multilevel Feedback Queue Scheduling





고려사항

Queue의 수
각 Queue 에 대한 스케줄링 알고리즘
프로세스의 우선순위 증가시기 결정방법
프로세스의 우선순위 감소시기 결정방법
프로세스가 어느 Queue에 들어갈 것인가 결정
프로세스가 서비스를 받는 시기를 결정하는 방법

스케줄링 알고리즘

■ HRN(Highest Response-Rate Next) 스케줄링

① SJF 스케줄링을 보완, 비선점 스케줄링 기법

② $$\text{가변적 우선순위} = \frac{\text{서비스 대기시간} + \text{서비스받을시간}}{\text{서비스받을시간}}$$

③ 시스템 응답시간 ; 대기한 시간 + 서비스 받을 시간

많이 기다렸다면 우선순위 높다.
서비스 받을 시간이 적다면
우선순위 높다.

알고리즘의 평가

분석적(해석적) 평가

① 최적의 스케줄링 알고리즘을 선택하는 것이 중요
② 알고리즘 선택기준 ; 프로세서 이용률, 응답시간, 처리율 등
③ 상대적인 선택조건 ; 최대응답시간은 1초이내, 반환시간과 전체실행시간의 선형비례 조건

프로세스	버스트시간
P_1	10
p_2	29
p_3	3
p_4	7
p_5	12

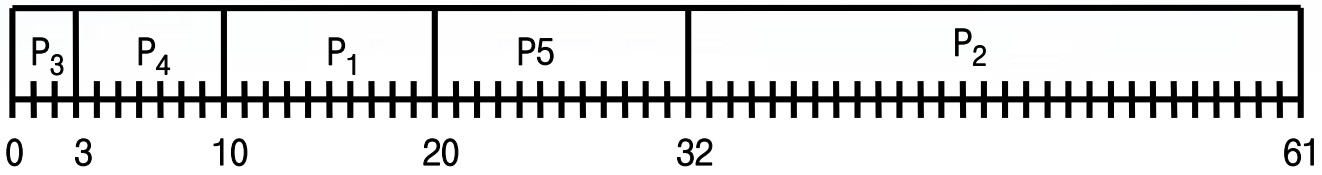
평가시 가정:
모든 프로세스의 도착시간은 0
시간 할당량 = 10

■ FCFS Scheduling **평가**

평균 대기시간 : (0+10+39+42+49)/5=28

알고리즘의 평가

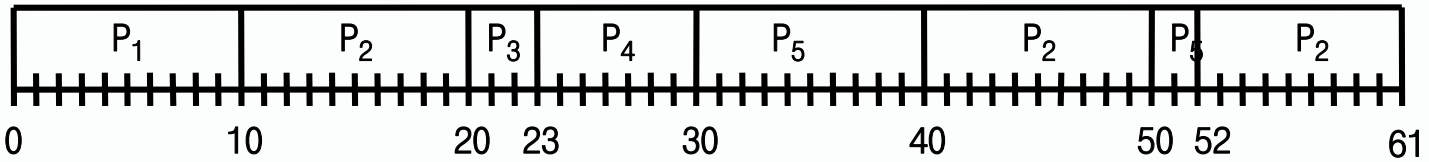
■ **비선점 SJF 평가**



평균 대기시간 : (10+32+0+3+20)/5=13

■ **RR 스케줄링 평가**

시간 할당량 = 10



평균 대기시간 : (0+32+20+23+40)/5=23

평균 대기시간 : FCFS(28) > RR(23) > 비선점SJF(13)

알고리즘의 평가

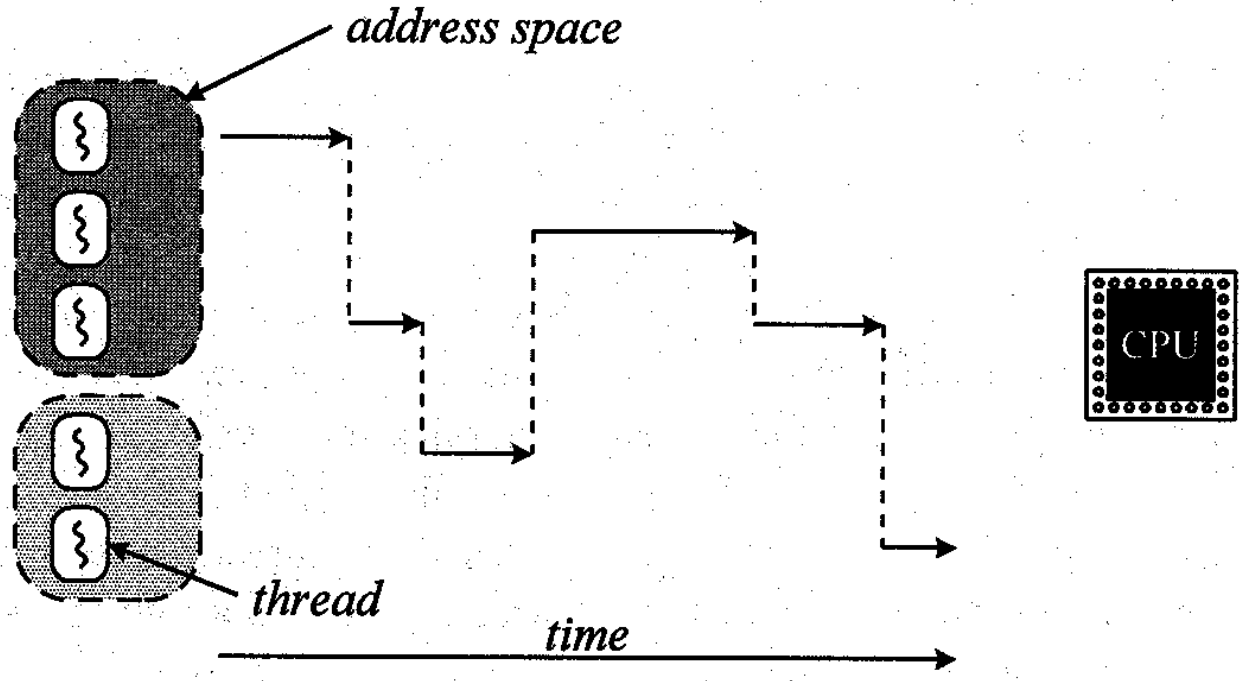
표 3-10 각 스케줄링의 비교

종류	방법	특징	
우선순위 스케줄링	우선순위를 할당해 우선순위가 높은 순서대로 처리하는 방법이다.	• 고정적 우선순위 • 가변적 우선순위 • 구입된 우선순위	비선점
미감 시간 스케줄링	프로세스가 주어진 시간 내에 작업이 끝나게 계획한다.	마감 시간을 계산해야 하므로 막대한 오버헤드와 복잡성이 발생한다.	비선점
FIFO 스케줄링	작업이 시스템에 들어온 순서대로 수행하는 방법이다.	• 대화형에 부적합하다. • 간단하고 공평하다. • 반응 속도를 예측할 수 있다.	비선점
라운드 로빈 스케줄링	FIFO 방식의 변형으로서 일정한 시간을 부여하는 방법이다.	• 시분할 방식에 효과적이다. • 할당 시간이 크면 FIFO와 같다. • 할당 시간이 작으면 문맥 교환이 자주 발생한다.	선점
SJF	수행 시간이 적은 작업을 우선적으로 처리하는 방법이다.	작은 작업에 유리하고 큰 작업은 상당히 시간이 많이 걸린다.	비선점
SRT	수행 도중 나머지 수행 시간이 적은 작업을 우선적으로 처리한다.	작업 처리는 SJF와 같으나 이론적으로 가장 작은 대기 시간이 걸린다.	선점
HRN	SRT의 큰 작업이 시간이 많이 걸리는 점을 보완한 방법이다.	우선순위 = $\frac{(\text{대기 시간} + \text{버스트 시간})}{\text{버스트 시간}}$	비선점
MLQ	서로 다른 작업을 각각의 큐에서 timeslice로 처리한다.	각각의 큐는 독자적인 스케줄링 알고리즘을 사용한다.	선점
MFQ	하나의 준비 상태 큐를 통해서 여러 개의 피드백 큐를 걸쳐 일을 처리한다.	CPU와 I/O 장치의 효율을 높일 수 있다.	선점

쓰레드(Thread) 스케줄링

■ 쓰레드(Thread) 개념

☑ **단일 CPU에서** 멀티쓰레드 처리개념

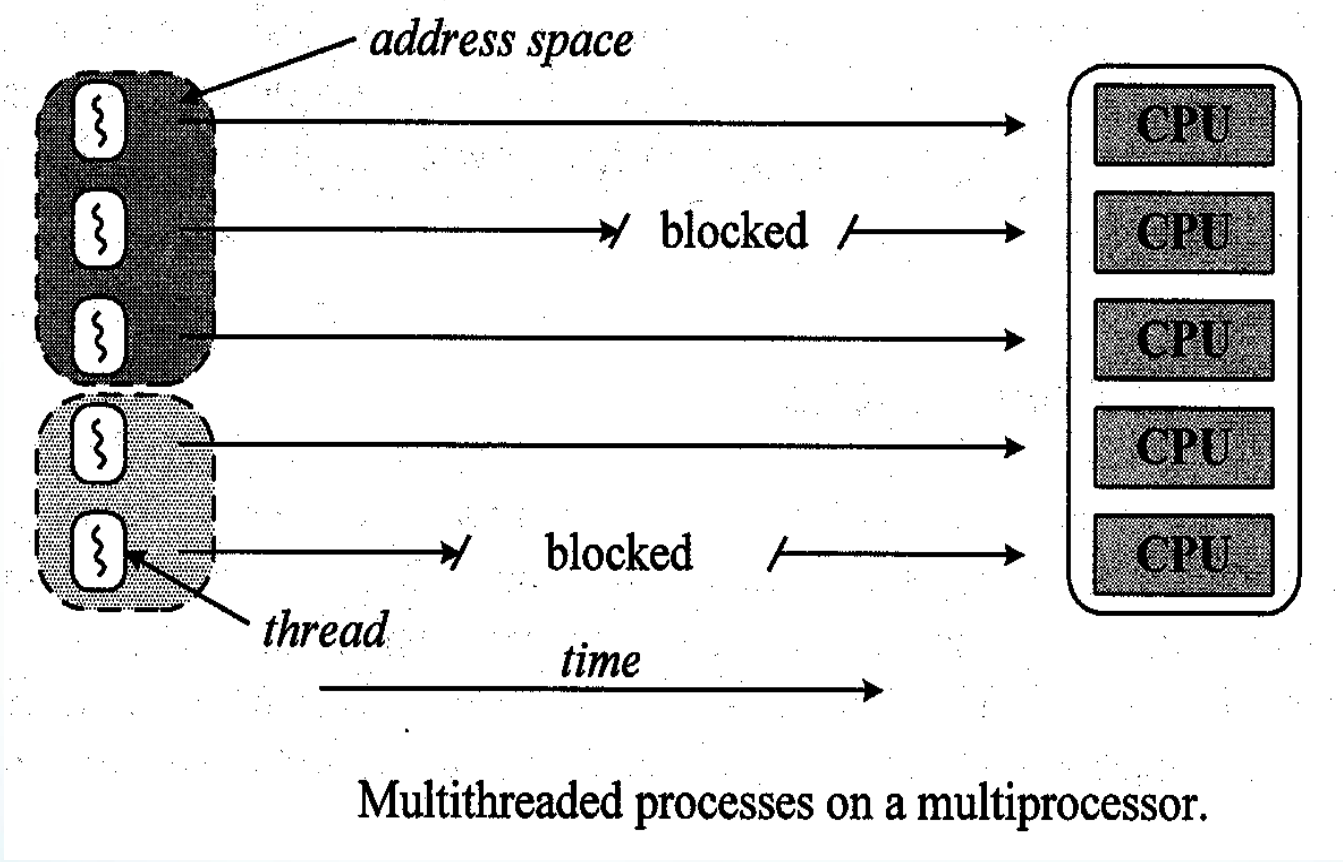


Multithreaded processes in a uniprocessor system.

쓰레드(Thread) 스케줄링

■ 쓰레드(Thread) 개념

☑ **다중 프로세서(CPU)에서** 멀티스레드 처리개념

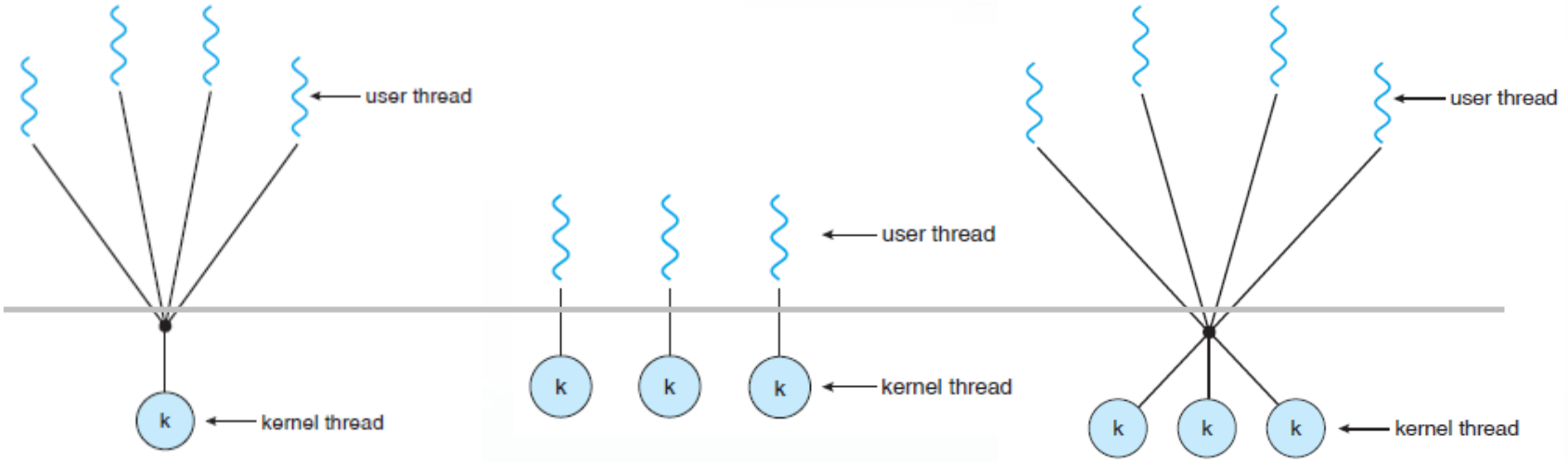


Multithreaded processes on a multiprocessor.

쓰레드(Thread) 스케줄링

■ 쓰레드(Thread) 스케줄링

☑ 3가지 모델에 따라 스케줄링 방식이 달라짐



Many to One Model

- 스레드관리: Thread Library
- user-thread 는 kernel thread 를 공유하므로 효율적
- Kernel 하나 이므로 multi 효과가 없음
- blocking 경우 문제가 생김
- Sun Solaris (UNIX)

One to One Model

- 스레드관리 : OS가 직접
- 병행성 효과는 매우 좋으나 user가 kernel thread 을 과다하게 생성하는 문제

- Linux (Kernel 2.60이전)
- Windows OS

Many to Many Model

- user thread는 kernel thread로 multiplexing 하여 처리
- kernel thread 갯수제한을 둠
- 두 모델의 하이브리드 모델

- HP-UX, IRIS, Tru64 UNIX
- Linux (최근, Kernel 2.6 이후)

쓰레드(Thread) 스케줄링

■ 쓰레드(Thread) 스케줄링

☑ Many to One Model, Many to Many Model

- user-level thread

  스케줄러: Thread Library, Priority

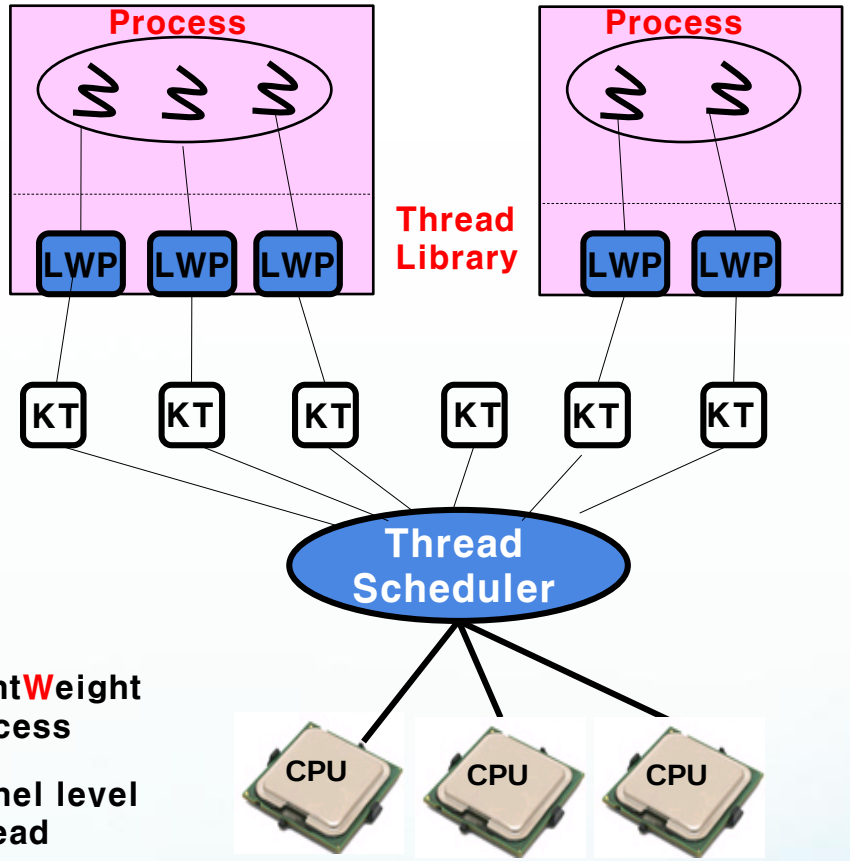
  (스케줄러는 프로그래머가 부여한 우선순위에 따라 스케줄링 함.)

- kernel-level thread

  스케줄러: Thread Scheduler

☑ One to One model

- Scheduler : Thread Scheduler



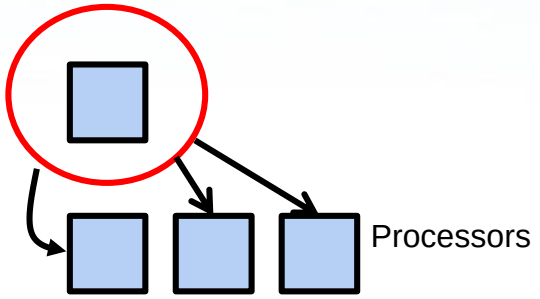
다중 프로세서 스케줄링

■ 다중프로세서 스케줄링

■ Asymmetric Multi Processing(AMP)

☑ Master Processor is a scheduler

☑ Other processors are executing user codes

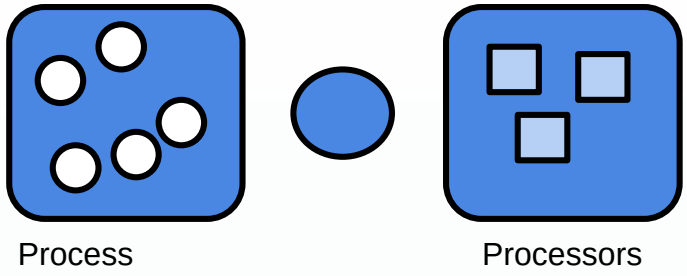


■ Symmetric Multi Processing(SMP)

☑ Each processor is a self-scheduler.

☑ Given a set of runnable threads
set of CPUs, assign threads to CPUs

☑ Linux, Windows OS, Solaris, Mac OS



☑ **스케줄링 기준**

공정성▲, 효율성▲, 처리율▲, 응답시간▼ …

☑ **스케줄링 알고리즘**

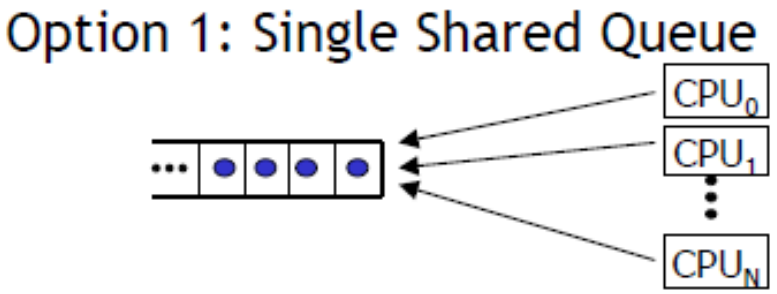
- **부하공유스케줄링, 전용프로세서할당스케줄링, 갱 스케줄링, 동적 스케줄링**

다중 프로세서 스케줄링

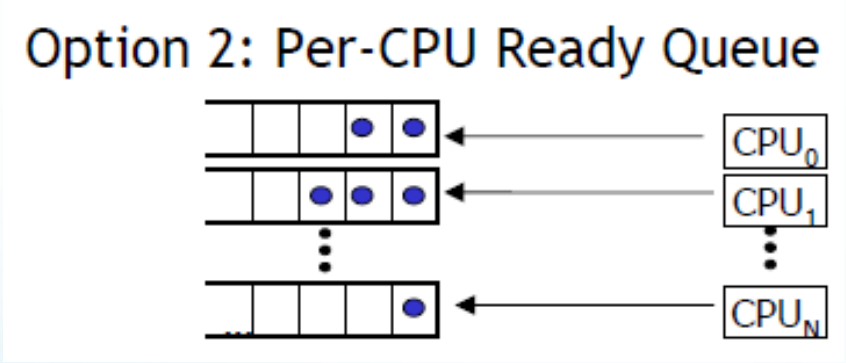
■ 부하공유(Load Sharing) 스케줄링

☑ CPU당 스케줄링 이벤트가 발생
-> 현재 실행중인 thread 가 일시적으로 중단될수 있음

☑ 스케줄러는 어떤 CPU 에서든지 실행될 수 있으며 준비큐를 사용할 수 있음



■ 전용프로세서 할당 스케줄링



☑ 각 CPU당 스케줄러가 실행
-> CPU 전용 준비 큐를 사용할 수 있음

☑ thread 생성후 어떤 준비 큐에 삽입

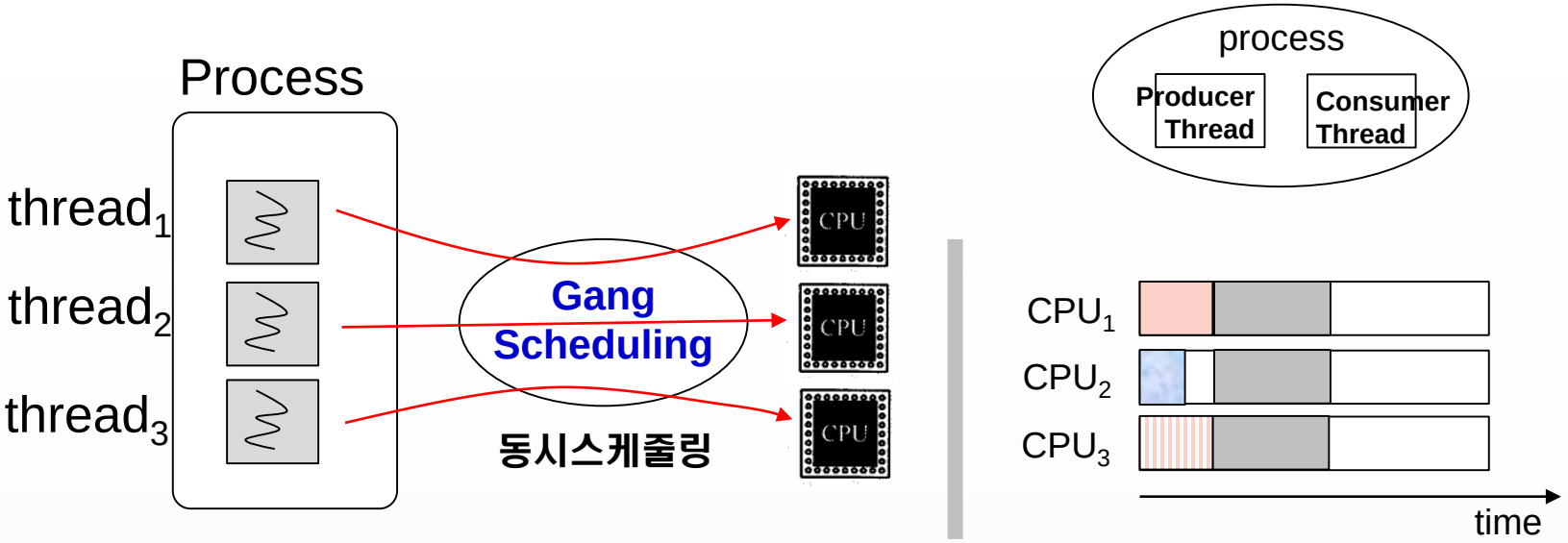
☑ Load Balancing 이 가능함

다중 프로세서 스케줄링

■ 갱(Gang) 스케줄링

☑ 한 프로세스에 속한 여러 쓰레드들을 동시에 스케줄링하는 기법

예) Producer/Consumer threads in a process



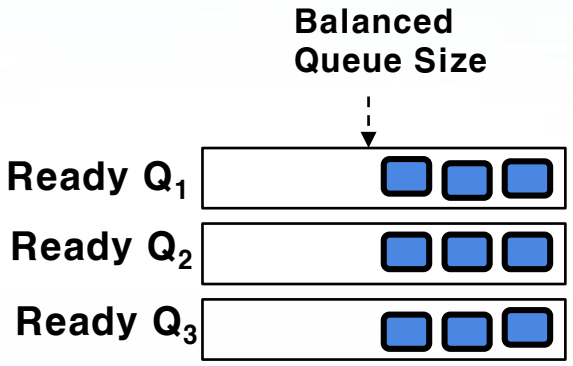
☑ 장단점

- 쓰레드간의 문맥교환 횟수를 줄일 수 있어 효율적 (스케줄링 오버헤드 줄여줌)

다중 프로세서 스케줄링

■ 동적 스케줄링(Load Balancing(부하 균등))

☑ CPU 당 준비 큐의 크기가 같도록 관리되어야 함



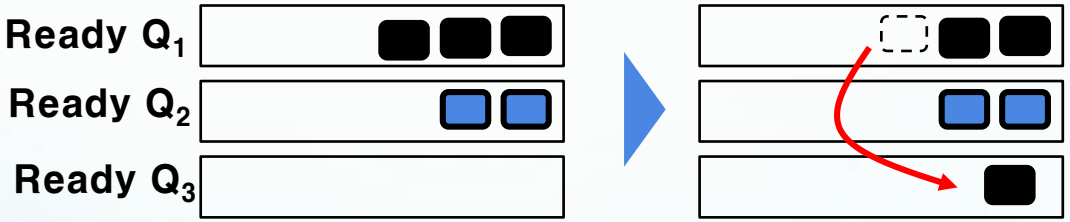
☑ 균등을 위한 2가지 기능

- Push Migration 기능

  커널은 주기적으로 Queue Size 를 점검하여
  크기가 작은 Q로 thread 를 이동시킴

- Pull Migration 기능

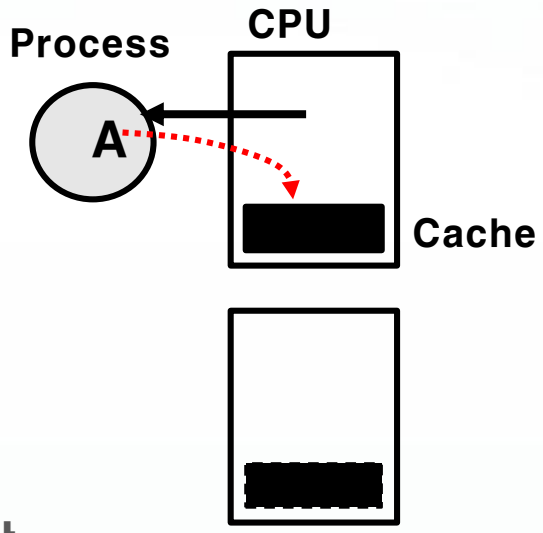
  CPU 는 Queue 가 Empty 가 되면 다른 Queue에 있는 thread 를 가져옴



다중 프로세서 스케줄링

■ 동적스케줄링에서의 문제점(프로세서 연관성(Affinity))

☑ 프로세스(또는 Thread)가 다른 준비 큐에 Migration 되다면, 이전 CPU 내의 캐쉬에 저장해 놓았던 데이터들을 사용할 수 없게 되는 문제
-> 프로세서 연관성 문제



☑ Soft Affinity

Migration 을 일부 허용하는 경우를 'Soft Affinity' 하다라 함

☑ Hard Affinity

Migration 을 강력하게 불허하는 경우를 'Hard Affinity' 하다라 함
예) Linux OS

사례연구 – 프로세스 스케줄링(Windows)

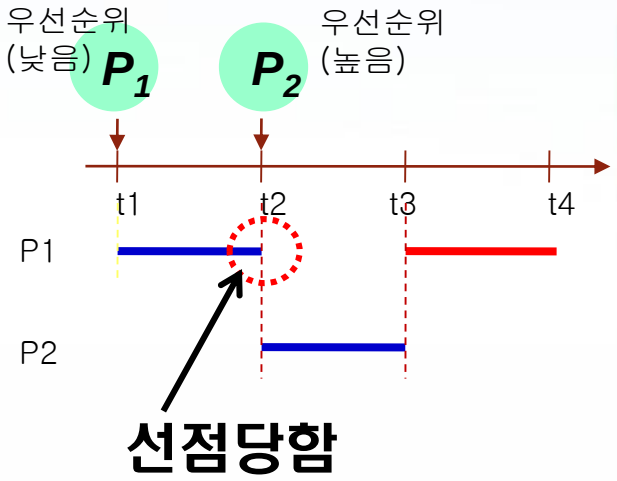
■ Windows XP/7/8/10 의 스케줄링

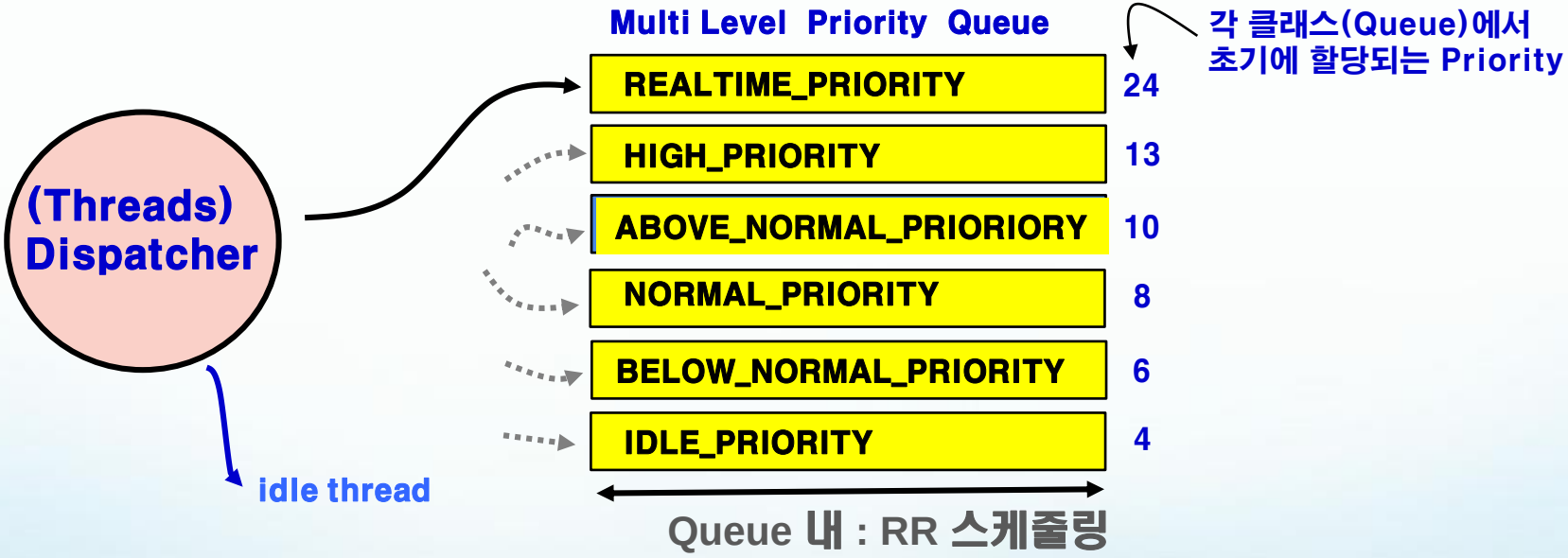
☑ Windosw OS 의 스케줄링 특징

- 스레드(Threads)단위로 스케줄링
- **Multilevel Feedback Queue + Priority 스케줄링**
- 선점방식 알고리즘을 채택
- Software Real-time 스케줄링 운영체제

☑ 32-level priority value

- 메모리 관리용(0), Variable Class(1~15), Real-time Class(16~31)





사례연구 – 프로세스 스케줄링(Windows)

■ Windows XP/7/8/10 의 스케줄링

☑ Windosw XP 의 Base Priority

Multilevel Queue (CLASS)

Level within each class \ CLASS	real-time	high	above normal	normal	below normal	idle priority
time-critical	31	15	15	15	15	15
highest	26	15	12	10	8	6
above normal	25	14	11	9	7	5
normal	24	13	10	8	6	4
below normal	23	12	9	7	5	3
lowest	22	11	8	6	4	2
idle	16	1	1	1	1	1

기본값
(초기설정값)

시간할당초과시마다
Priority**값 감소**
(**단**, Base Priority
이하로 감소안됨)

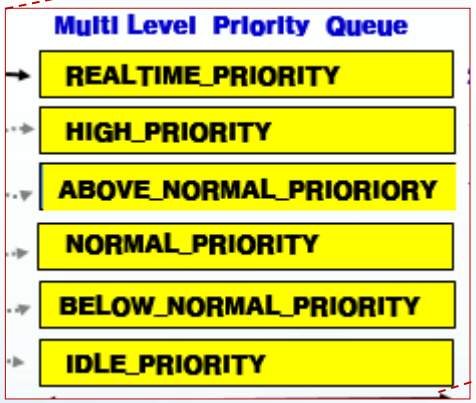
Base
Priority

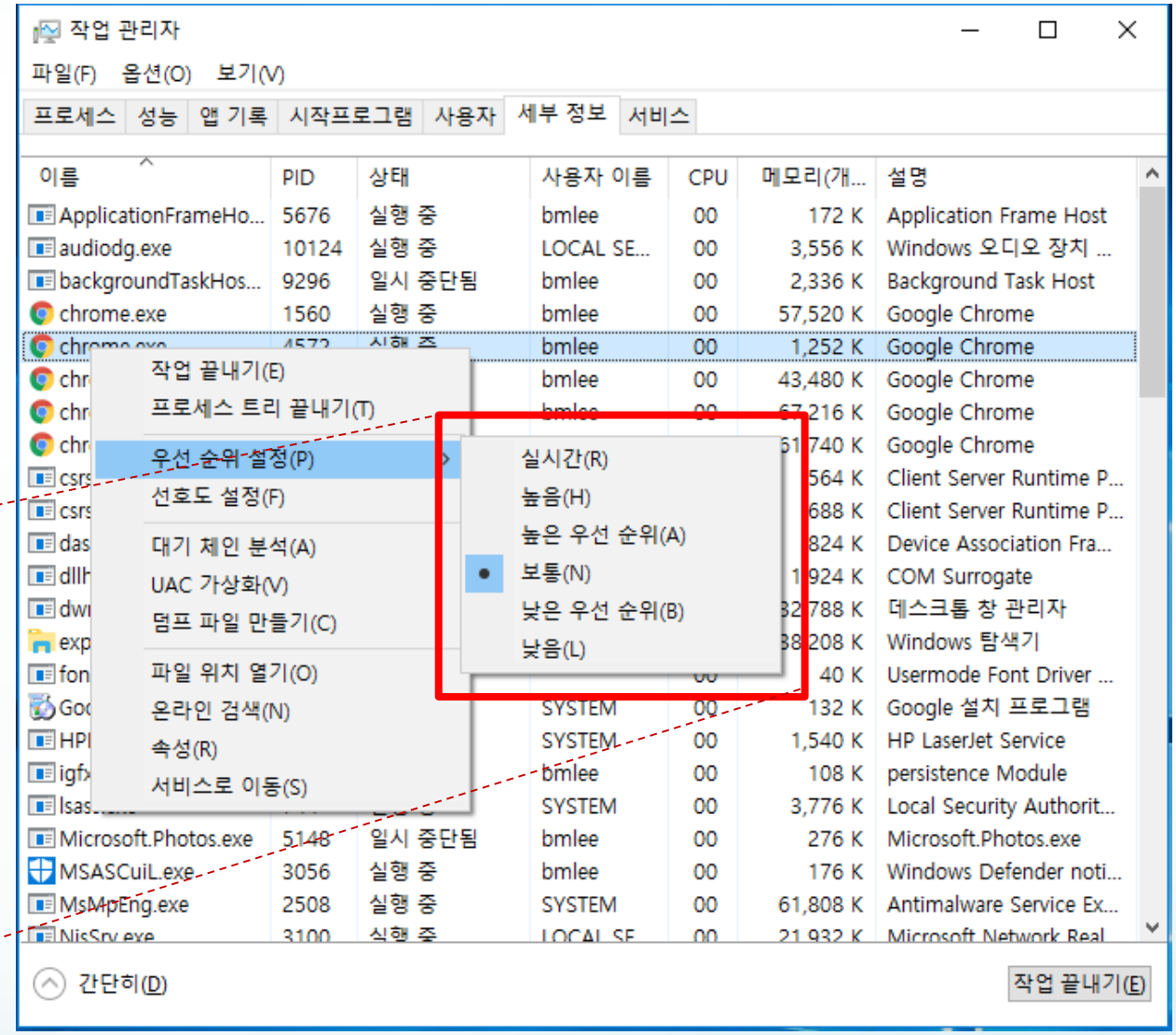
사례연구 – 프로세스 스케줄링(Windows)

■ Windows 10

☑ **작업관리자 프로세스**

- **우선순위** Level **변경가능**

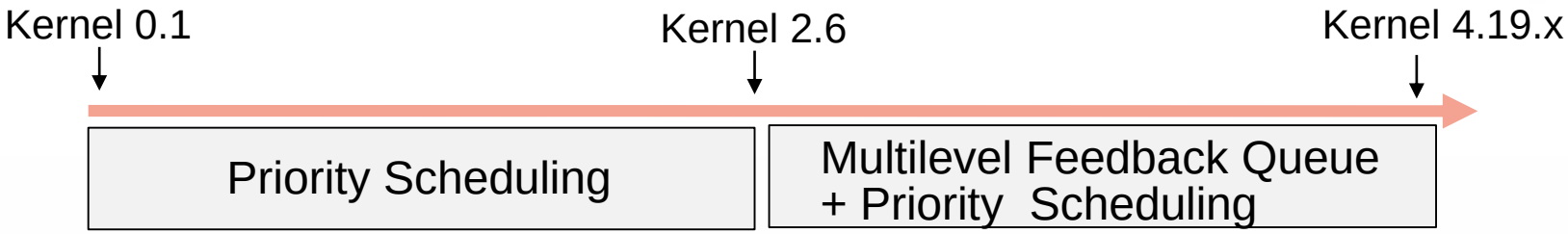




사례연구 – 프로세스 스케줄링 (Linux)

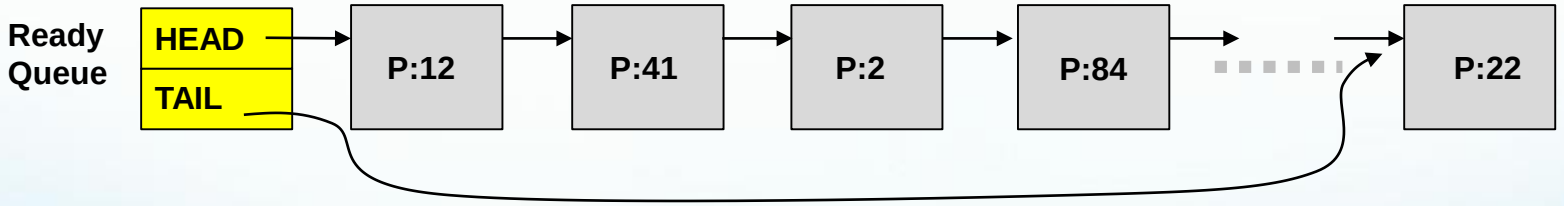
■ Linux 의 스케줄링

☑ Linux Kernel 2.6 이전/이후 스케줄링



☑ Priority Scheduling (Linux Kernel 2.6 이전)

- Scanning priority number at a list of ready queue
- Select a highest priority process
- **단순하고, 명료하지만, 매번 스캐닝 & 스캐닝 시간소요로 인한** overhead **발생**

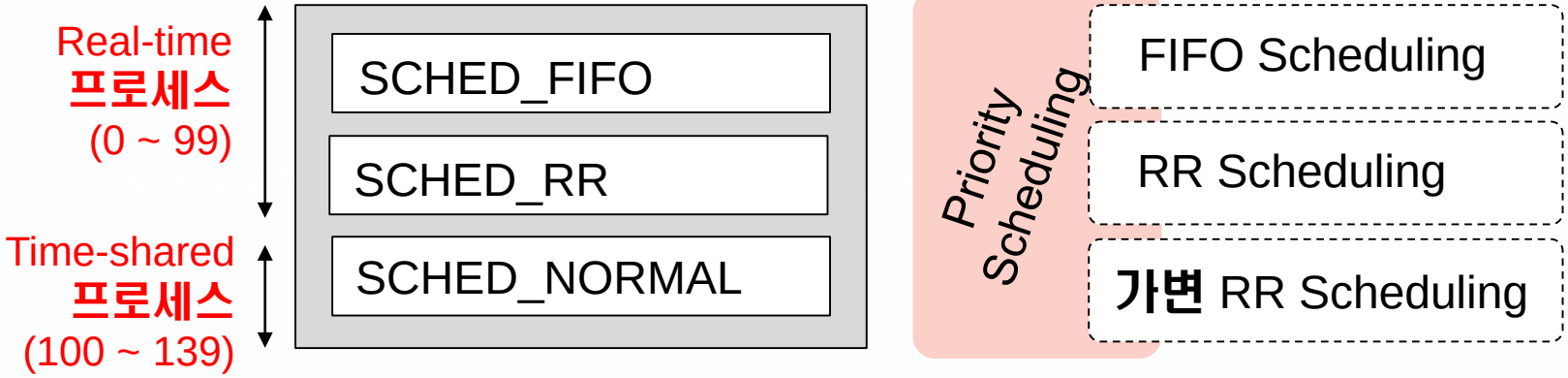


사례연구 – 프로세스 스케줄링 (Linux)

■ Linux 의 스케줄링

☑ Multilevel Queue + Priority **스케줄링 알고리즘** (Kernel 2.6 **이후**)

- 3개 클래스로 분류 : SCHED_FIFO, SCHED_RR, SCHED_NORMAL
- 각 클래스마다 다른 스케줄링 방법 적용



- Real time **프로세스** …… **우선순위** (0 ~ 99)

> Interactive Job … **마우스, 키보드 등 실시간 반영이 필요한 프로세스**
> Real-time Job ….. **센싱데이타 처리등 즉시 처리가 필요한 프로세스**
> (**더 높은 우선순위가 온다면**, Time Quantum**이 초과되면**, I/O Block**경우** => **선점됨**

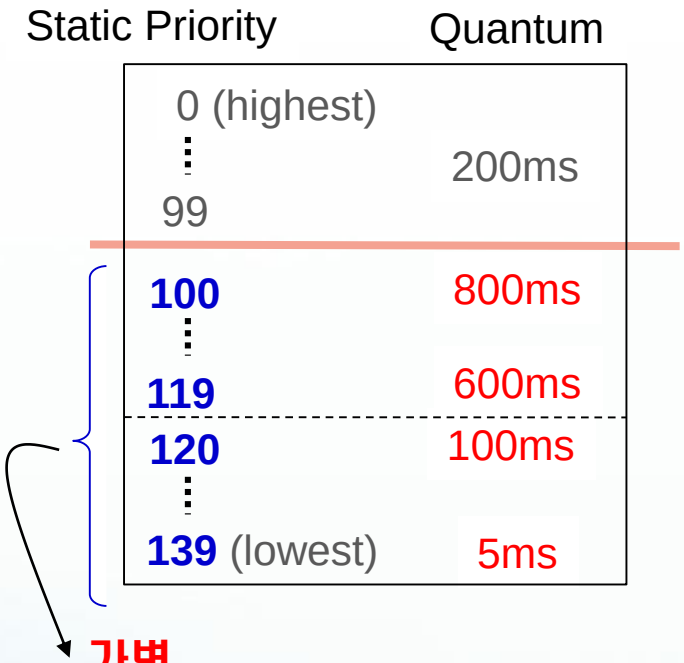
사례연구 – 프로세스 스케줄링 (Linux)

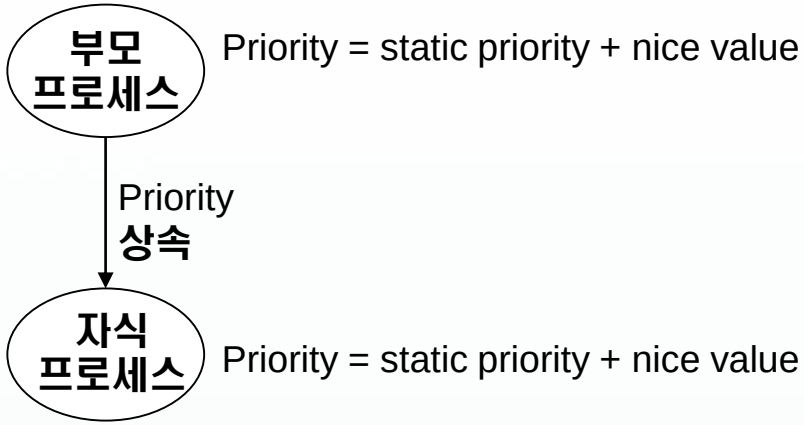
■ Linux 의 스케줄링

- Time shared **프로세스**

  > **일반적인 프로세스** … 100 ~ 140 priority

  > **우선순위에 따라** (**가변** Time Quantum) **할당**





= (140 – numeric priority) x 20, if priority < 120
= (140 – numeric priority) x 5, if priority >= 120

사례연구 – 프로세스 스케줄링 (Linux)

■ Linux 의 스케줄링

- CentOS 7 Linux 경우 (Kernel 4.5)

```
[leebyungmun@prjsvr ~]$ ps -eo pid,ppid,class,rtprio,ni,pri,fname | more
  PID  PPID CLS  RTPRIO  NI PRI COMMAND
    1     0 TS       -   0  19 systemd
    2     0 TS       -   0  19 kthreadd
    3     2 TS       -   0  19 ksoftirq
    5     2 TS       - -20  39 kworker/
    7     2 TS       -   0  19 rcu_sche
    8     2 TS       -   0  19 rcu_bh
    9     2 TS       -   0  19 rcuos/0
   10     2 TS       -   0  19 rcuob/0
   11     2 FF      99   - 139 migratio
   12     2 TS       - -20  39 lru-add-
   13     2 FF      99   - 139 watchdog
   14     2 TS       -   0  19 cpuhp/0
   15     2 TS       -   0  19 cpuhp/1
   16     2 FF      99   - 139 watchdog
   17     2 FF      99   - 139 migratio
   18     2 TS       -   0  19 ksoftirq
   20     2 TS       - -20  39 kworker/
   21     2 TS       -   0  19 rcuos/1
   22     2 TS       -   0  19 rcuob/1
   24     2 TS       -   0  19 kdevtmpf
   25     2 TS       - -20  39 netns
```

PID ..... process ID
PPID ..... Parent PID
CLS ..... Class queue
- TS (Time shared) class
- FF (FIFO) class

RTPRIO … realtime priority
NI ..... Nice value
PRI ..... static priority

사례연구 – 프로세스 스케줄링 (Unix)

■ Solaris 의 CPU 스케줄링

☑ Priority based thread 스케줄링
- 4 classes of scheduling
- different priority, different scheduling within each class

☑ 시분할(time sharing): default class
- multi-level feedback queue scheduling
- 우선순위 높을수록 적은 시간량,
- 우선순위 낮을수록 많은 시간양
- CPU burst 프로세스는 우선순위 낮아짐
- sleep에서 복귀한 쓰레드는 높은 우선순위

☑ 대화형(interactive)
- multi-level feedback queue scheduling
- windowing application에 높은 우선순위

☑ 실시간(real time): 최상위 우선순위

☑ 시스템(system): 우선순위 결정된 후 불변

우선순위 높다

우선순위	시간양
0	200ms
5	200ms
10	160ms
15	160ms
20	120ms
25	120ms
30	80ms
35	80ms
40	40ms
45	40ms
50	40ms
55	40ms
59	20ms

Q&A