5. CPU Scheduling

CPU scheduler & Dispatcher

CPU Scheduler

Ready 상태의 프로세스 중에서 이번에 CPU를 줄 프로세스를 고른다

→ 운영체제 안에서 CPU Scheduler를 위한 코드가 존재. 그것을 scheduler라고 부름

Dispatcher

CPU의 제어권을 CPU scheduler에 의해 선택된 프로세스에게 넘긴다

이 과정을 **context switch(문맥 교환)**이라고 한다

→ CPU를 넘겨주는 역할을 하는 코드

CPU스케줄링이 필요한 경우는 프로세스에게 다음과 같은 상태 변화가 있는 경우이다

1. Running → Blocked (예: I/O 요청하는 시스템 콜)
2. Running → Ready (예: 할당시간 만료로 timer interrupt)
3. Blocked → Ready (예: I/O 완료후 인터럽트)
4. Terminate

1, 4에서의 스케줄링은 **nonpreemptive (= 강제로 빼앗지 않고 자진 반납)**

All other scheduling is **preemptive (= 강제로 빼앗음)**

Scheduling Criteria

Performance Index (= Performance Measure, 성능 척도)

CPU utilization (이용률)

Keep the **CPU as busy as possible**

Throughput (자리량)

# of process that **complete** their execution per time unit

Turnaround time (소요시간, 반환시간)

amount of time to **execute a particular process**

Waiting time (대기 시간)

amount of time a process has been **waiting in the ready queue**

Response time (응답 시간)

amount of tume it takes **from when a request was submitted until the first response in produced**, not output (for time-sharing environment)

FCFS (First-Come First-Served)

process	burst time
p1	24
p2	3
p3	3

프로세스의 도착 순서 P1, P2, P3

The Gantt chart for the schedule is :

Waiting time for p1=0, p2=24, p3=27

Average waiting time : (0 + 24 + 27) / 3 = 17

Much better than previous case.

Convoy effect : short process behind long process

→ 먼저 들어 온 것을 먼저 처리하게 되므로 처리 시간이 오래 걸리면 다른 프로세스들도 많이 기다리게 되어 평균 waiting time이 길어지게 됨

SJF (Shortest-Job-First)

각 프로세스의 다음번 **CPU burst time**을 가지고 스케줄링에 활용

CPU burst time이 가장 짧은 프로세스를 제일 먼저 스케쥴

Nonpreemptive

일단 CPU를 잡으면 이번 CPU burst가 완료될 때까지 CPU를 선점(preemption) 당하지 않음

Preemptive

현재 수행중인 프로세스의 남은 burst time 보다 더 짧은 CPU burst time을 가지는 새로운 프로세스가 도착하면 CPU를 빼앗김

이 방법을 **Shortest-Remaining-Time-First (SRTF)** 이라고도 부른다

SJF is optimal

주어진 프로세스들에 대해 **minimum average waitin time**을 보장한다.

문제점

burst time이 긴 프로세스는 영원히 CPU를 잡지 못함. → **Starvation**

다음 CPU Burst Time의 예측

다음 CPU burst time을 어떻게 알 수 있는가?

추정(estimate)만이 가능하다.

과거의 CPU burst time을 이용해서 추정

(exponential averaging)

Priority Scheduling

A **priority number** (integer) is associated with each process

highest priority를 가진 프로세스에게 CPU할당

(smallest integer = highest priority)

- Preemptive
- nonpreemptive

SJF는 일종의 priority scheduling이다.

- priority = predicted next CPU burst time

Problem

- **Starvation (기아 현상)** : low priority processes may never execute.

Solution

- **Aging (노화)** : as time progresses increase the priority of the process.

→ 오래 기다리면 우선순위를 높여줌

Round Robin (RR)

각 프로세스는 동일한 크기의 할당 시간(**time quantum**)을 가짐

(dlfqkswjrdmfh 10-100 milliseconds)

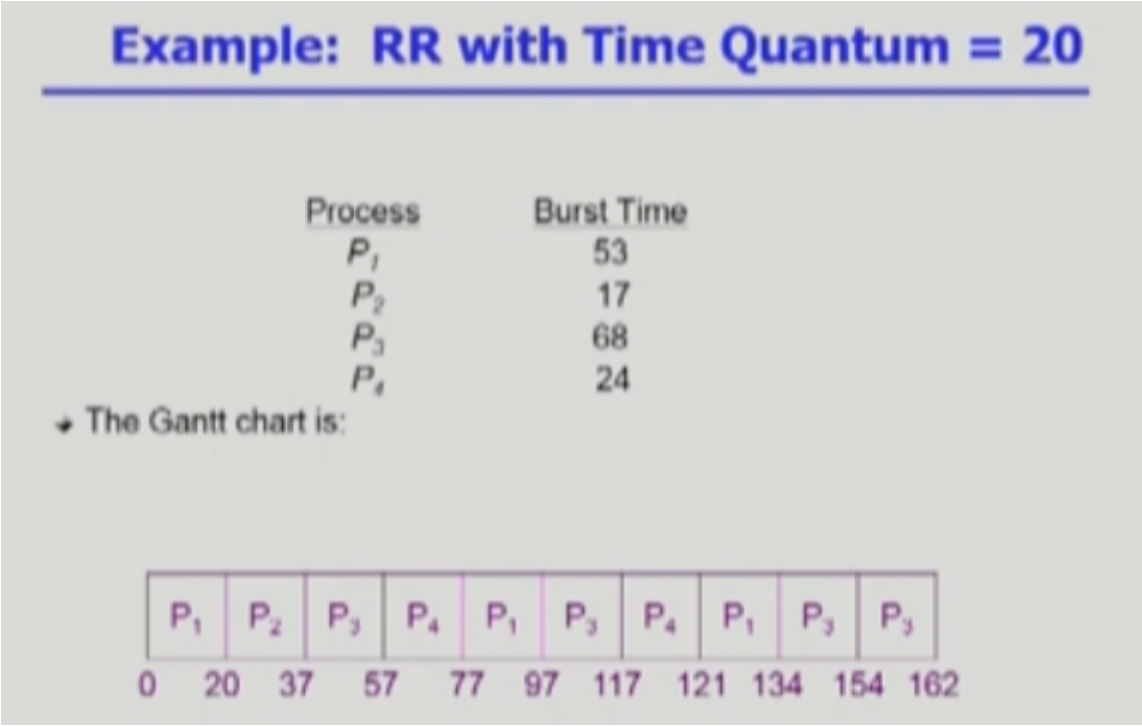
할당 시간이 지나면 프로세스는 선점(preempted)당하고 ready queue의 제일 뒤에 가서 다시 줄을 선다

n개의 프로세스가 ready queue에 있고 할당 시간이 q time인 경우 각 프로세스는 최대 **q time unit** 단위로 CPU시간의 1/n을 얻는다.

⇒ 어떤 프로세스도 (n-1)q time unit 이상 기다리지 않는다.

Performance

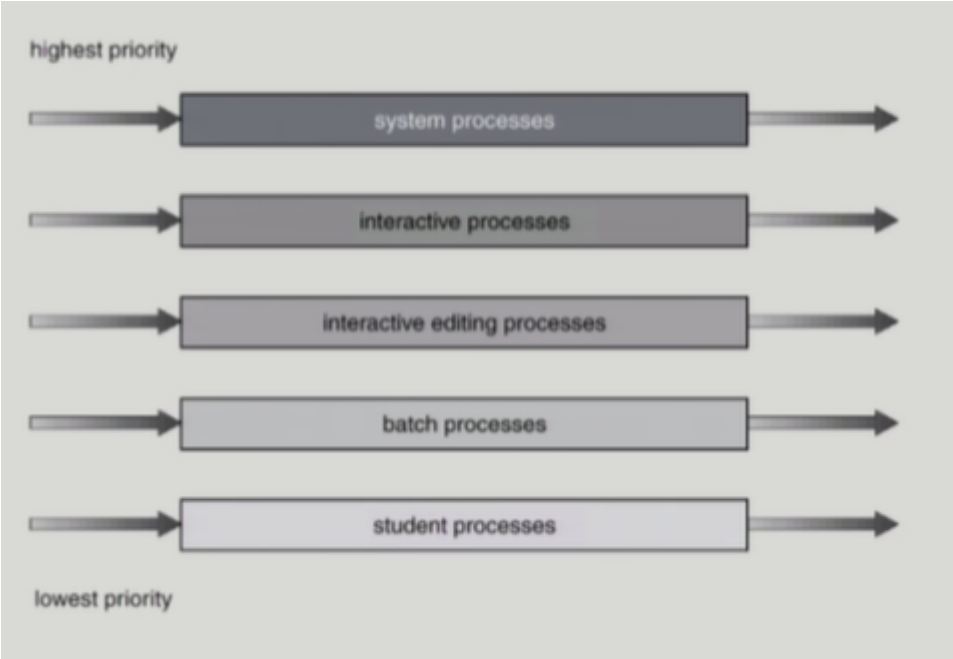
- q large ⇒ FCFS
- q small ⇒ context switch 오버헤드가 커진다



- **일반적으로 SJF 보다 average turnaround time이 길지만 response time은 더 짧다.**

→ 만약 CPU 사용 시간이 모두 동일한 경우에는 100초짜리 프로세스 4개 → 400초가 돼야 모두 끝남. 그럴땐 그냥 순서대로 처리하면 적어도 하나는 100초에 끝낼 수 있음. 그런 경우에는 RR을 사용하면 안좋을 수도 있음. 그러나 일반적으로는 짧은 프로세스와 긴 프로세스가 섞여있기 때문에 좋음!

Multilevel Queue



Ready queue를 여러 개로 분할

- foreground (**interactive**)
- background (**batch-no human interaction**)

각 큐는 독립적인 스케줄링 알고리즘을 가짐

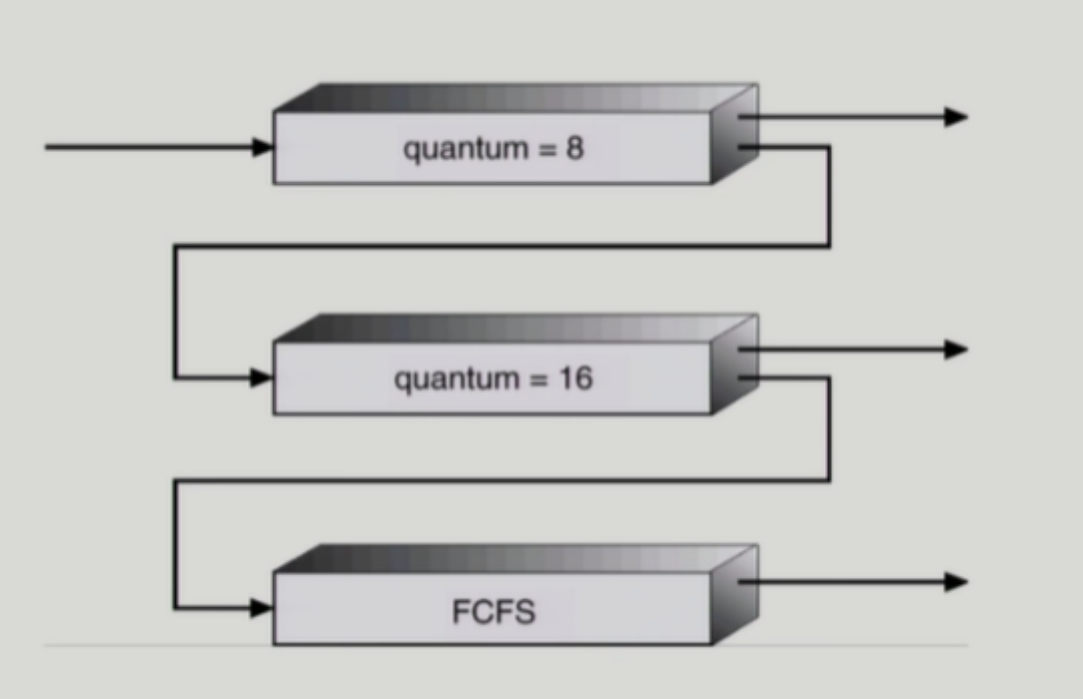
- forground - **RR**
- background - **FCFS**

큐에 대한 스케줄링이 필요

- Fixed priority scheduling
    - serve all from foreground then from background

    - Possibility of starvation.
- Time slice
    - 각 큐에 CPU time을 적절한 비율로 할당
    - Eg. 80% to foreground in RR, 20% to background in FCFS

Multilevel Feedback Queue



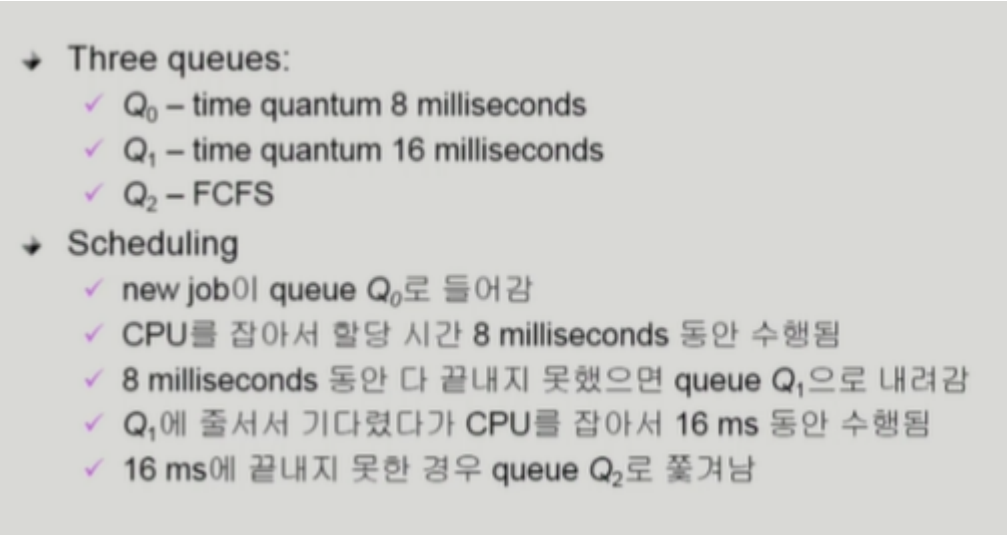
프로세스가 다른 큐로 이동 가능

에이징(aging)을 이와 같은 방식으로 구현할 수 있다

Multi-feedback-queue scheduler를 정의하는 파라미터들

- **Queue의 수**
- **각 큐의 scheduling algorithm**
- **Process를 상위 큐로 보내는 기준**
- **Process를 하위 큐로 내쫓는 기준**
- **프로세스가 CPU 서비스를 받으려 할 때 들어갈 큐를 결정하는 기준**

Example



Multiple-Processor Scheduling

CPU가 여러 개인 경우 스케줄링은 더욱 복잡해짐

Homogeneous processor인 경우

- Queue에 한줄로 세워서 각 프로세서가 알아서 꺼내가게 할 수 있다
- 반드시 특정 프로세서에서 수행되어야 하는 프로세스가 있는 경우에는 문제가 더 복잡해진다.

Load sharing

- 일부 프로세서에 job이 몰리지 않도록 부하를 적절히 공유하는 메커니즘 필요
- 별대의 큐를 두는 방법 vs 공동 큐를 사용하는 방법

Symmetric Multiprocessing (SMP)

- 각 프로세서가 각자 알아서 스케줄링 결정

Asymmetric multiprocessing

- 하나의 프로세서가 시스템 데이터의 접근과 공유를 책임지고 나머지 프로세서는 거기에 따름

Real-Time Scheduling

Hard real-time systems

- Hard real-time task는 정해진 시간 안에 반드시 끝내도록 스케줄링 해야 함

Soft real-time computing

- Soft real-time task는 일반 프로세스에 비해 높은 priority를 갖도록 해야 함

Thread Scheduling

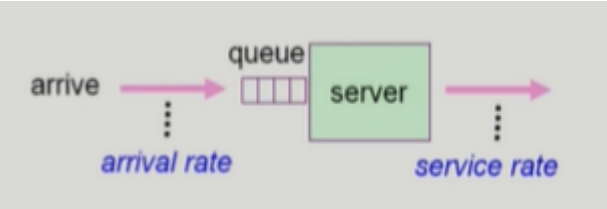
Local Scheduling

- User level thread의 경우 사용자 수준의 thread library에 의해 어떤 thread를 스케줄할지 결정

Global Scheduling

- Kernel level thread의 경우 일반 프로세스와 마찬가지로 커널의 단기 스케줄러가 어떤 thread를 스케줄할지 결정

Algorithm Evaluation



Queueing models

확률 분포로 주어지는 **arrival rate**와 **service rate**등을 통해 각종 **performance index 값**을 계산

Implementation (구현) & Measurement (성능 측정)

실제 시스템에 알고리즘을 **구현**하여 실제 작업(**workload**)에 대해서 성능을 **측정** 비교

Simulation(모의 실험)

알고리즘을 **모의 프로그램**으로 작성 후 **trace**를 입력으로 하여 결과 비교

출저 : kocw 운영체제 - 반효경 교수님 (이화여자대학교)

http://www.kocw.net/home/search/kemView.do?kemId=1046323