05. Multi-Level Feedback Queue

▼ Multi-Level Feedback Queue(MLFQ) - 1

▼ review) scheduling metrics

- *turnaround time*

$$T_{turnaround} = T_{completion} - T_{arrival}$$

(SJF, STCF) (But, 현실에서는 OS 작업 실행 시간 모름)

  - 해당 metrics : OS가 각 작업의 실행 시간을 알고 있다는 가정이 들어감
    - OS : 알고 있는 실행 시간을 토대로 scheduler가 적절한 노드를 배치 → OS 내부 scheduler 작동

- *Response time*

$$T_{response} = T_{firstrun} - T_{arrival}$$

  - 해당 metrics : OS는 interactive user에게 반응할 수 있도록 시스템 구성
    - RR : response time은 좋지만 turnaround time이 낮음

⇒ 둘 다 동시에 좋게 만드는 방법?? ① minimize response time for interactive users. ② minimize turn around time

⇒ but, 각 작업의 길이를 알지 못 하면 최적으로 만들 수 없음 ⇒ MLFQ로 해결해보자!

① ▼ MLFQ

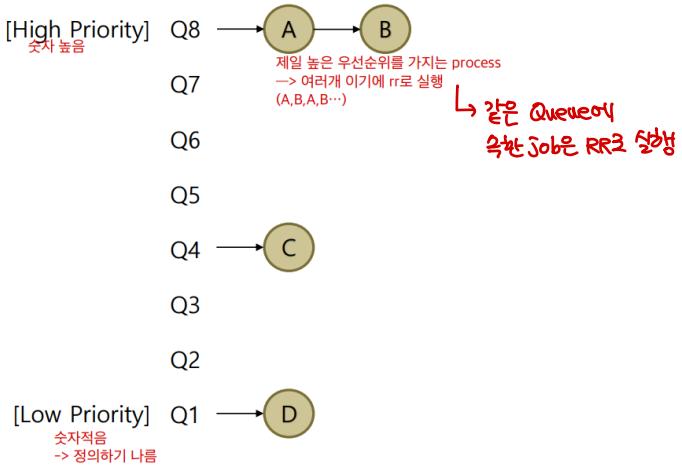
- **multiple queues** : 우선순위에 따라서 queue에 들어감 (스케줄링이 필요)

  ↔ ready queue : cpu core마다 존재

  - 실행할 준비가 되면 언제든 single queue에 들어감.
  - basic rules
    - rule1 : Priority(A) > Priority(B) ⇒ A 실행
    - rule2 : Priority(A) = Priority(B) ⇒ A와 B가 RR 실행

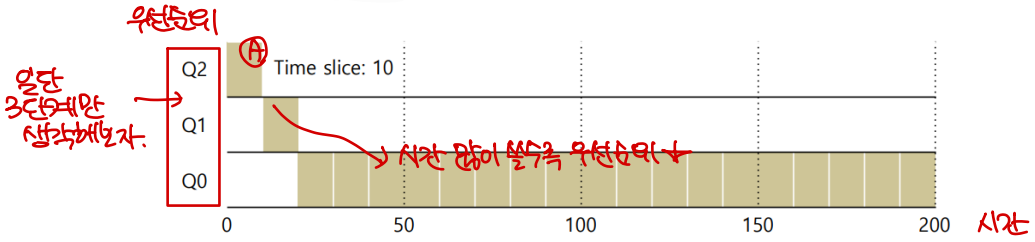
    ⇒ 다양한 rule 추가하면서 해결점 찾기 (위 두 rule은 response, turn around 해결 X)

- example(ready queue)



- priority
  - OS Scheduler : 각 process에 대해 우선순위를 고정
    - process가 어떤 behavior 하느냐에 따라서 priority가 결정됨
  - process workload
    - I/O bound jobs: 짧게 실행
      - cpu 안 쓸수록 priority가 높음 → i/o 작업을 빠르게 실행

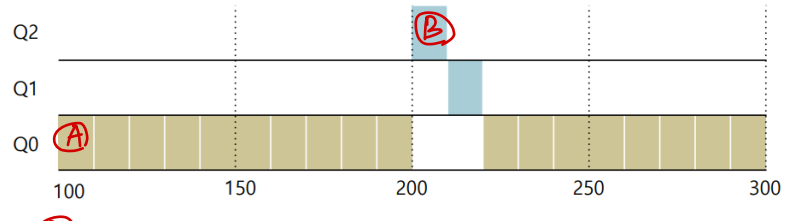
        →response 낮춤
    - CPU-bound jobs: 계산 or 길게 실행
      - cpu 많이 쓸수록 priority가 낮음 → cpu 집중적으로 사용
  - basic rule +
    - rule3 : 어떤 workload를 가지는지 모르기 때문에 일단 높은 priority
    - rule4 : time slice 반복 → 자꾸 running state로 들어감 → priority down
    - rule5 : time slice 다 쓰기 전에 blocked state로 들어감(ex. i/o 작업 실행)
- example → rule3,4
  - 길이가 긴 작업 하나만 도착했을 때



    - 가장 먼저 들어온 놈 → 제일 높은 priority(Q2)

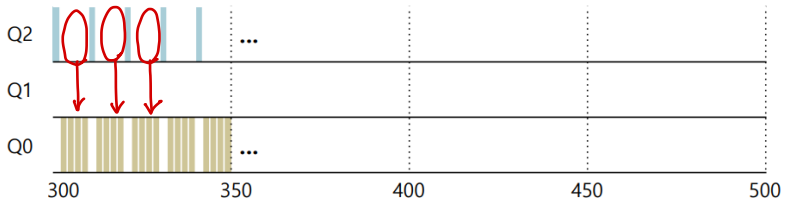
- 만약 cpu-intensive한다면 priority 낮아짐
  - → CPU 많이 쓰면

- 길이가 짧은 놈이 도착했을 때



  - 파란 프로세스 : 들어오면 가장 높은 priority(Q2)
    - time slice 소진 → 비교적 노란 프로세스에 비해 길이 짧음
  - 노란 프로세스 : 너무 긴 시간 → 중단되어도 turn around time 차이 x ⇒ 파란놈이 너무 자주 중단!

- running with a I/O job

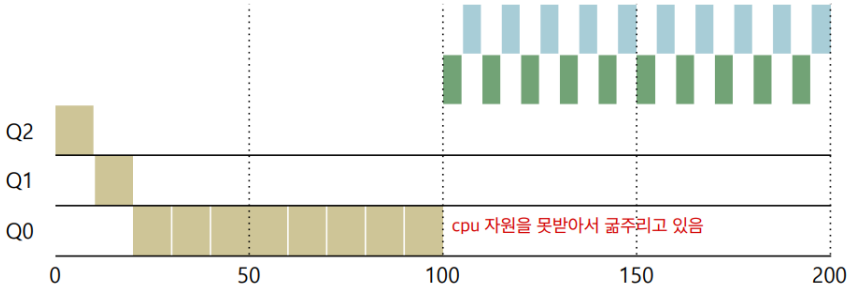


  - 파란 프로세스 : 잦은 I/O 작업 → time slice 다 안 쓰고 I/O 작업 실행 → Blocked state
    - 우선순위 그대로
  - 노란 프로세스 : 파란 프로세스보다 우선순위 낮춰서 실행

▼ Multi-Level Feedback Queue - 2

▼ Problems

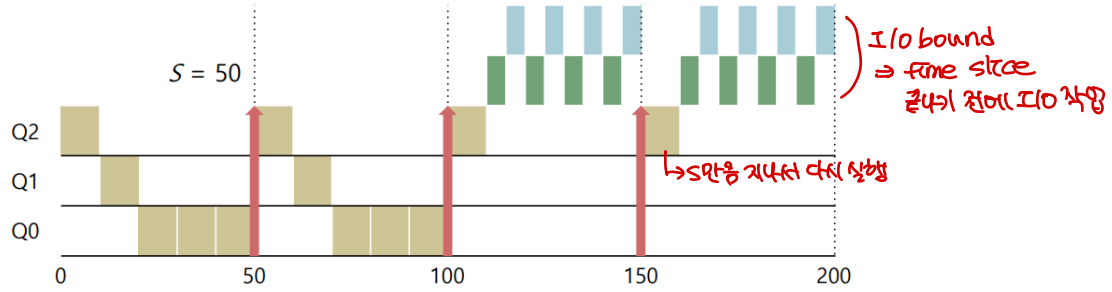
1. ***starvation*** : system에 너무 많은 interactive job 존재 → job이.. 굶주림



- Priority Boost : 여러 상황에 대처해 모두의 우선순위를 up해서 reset된 뒤에 다시 판단

- basic rule+
  - rule 5 : S 정도의 time 지나면 system 안에 있는 모든 job을 top priority로 옮겨줌
    - i/o workload job : 상대적으로 높은 쪽에 오래 머물게 됨
    - cpu workload job : 상대적으로 금방 낮아지고 중간에 위로 올라감 → 좋은 점은 x

    ⇒ but, starving + changing-behaivor problem 발생하지 않는다!
- example



- s=50 → 우선순위 계속 올려줌
- 파란, 초록 프로세스 : 새로 도착했을 때 실행할 수 있는 기회 주어짐 (I/O bound)
  - 노란 프로세스 : 우선순위가 낮아서 실행할 수 없음

  (끝나는 시점에 Q1으로 내려감)

time slice

(time slice가 끝나기 직전에 I/O 작업) ⇒ 양쪽 같음.

OS 특징!!

2. ***gaming the scheduler*** : i/o bound인척 scheduler 속일 수 있음 → 느리게 낮아짐

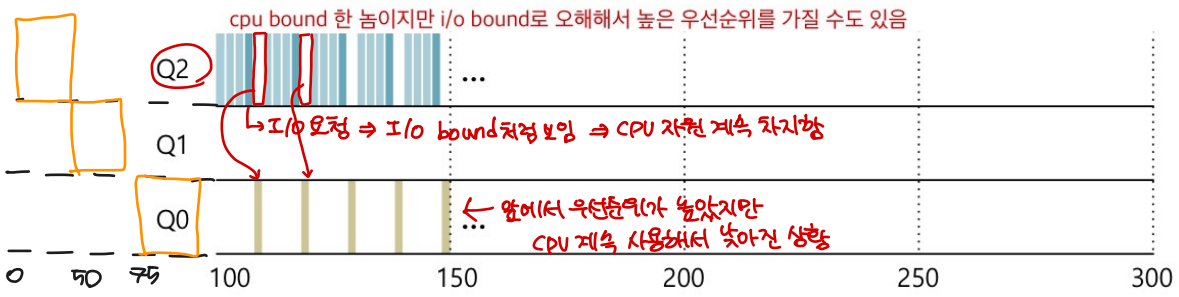
- Better Accounting
  - basic rule+
    - rule 4a : 실행하는 동안 모든 time slice 이용했음 → 우선순위 낮아짐
    - rule 4b : time slice 끝나기 전에 실행 종료 → 우선순위 유지 (CPU 포기)

    악용가능

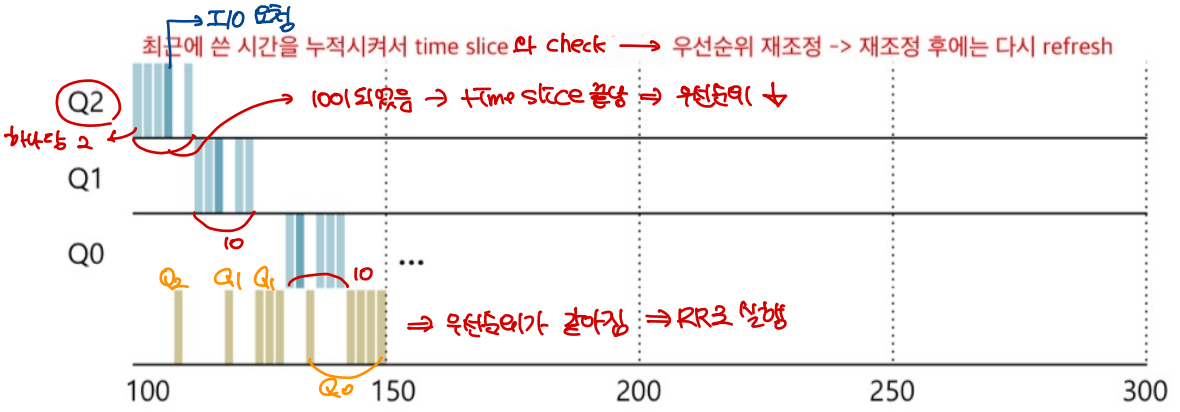
    ⇒ Gaming tolerance(↔ Gaming the scheduler) ⇒ 버전 수정!!
    - rule 4 : CPU 포기한 횟수와 관련 없이 그동안 사용한 시간이 time slice만큼 한번이라도 다 사용했음 → 우선순위 낮아짐

- example → Gaming the scheduler (rule 4 수정 전 ver)

  (악용하는 예시)



- example → Gaming tolerance (rule 4 수정후 ver) , time slice = 10



↳우선순위 낮아진상태로 I/O 작업 하게 됨 ⇒ 잘 처리할 수 있을까?

3. ***changing behavior*** : cpu-bound ⇒ i/o bound로 job이 변화했을 때 어떻게 반영?
   - priority boost로 자연스럽게 해결됨. ⇒ S라는 주기로 우선순위 변경해주기에 괜찮음.

(우선순위 변화 간격) S값은 어떻게 설정?

   - s라는 간격 대신 굉장히 큰 간격 잡음 → 해결 x
   - s가 되게 작음 → 너무 많은 boost로 OS의 overhead 발생
   - s의 default value : 1s(solaris) → MFLQ 중 하나
   - time slice는 얼마나 설정?
     - **short** time slices for **high** priority queues : 20ms
     - **longer** time slices for **low** priority queues : few hundred ms

▼ Multi-Level Feedback Queue

- Rule 1: If Priority(A) > Priority(B), A runs (B doesn't) ->기본적인 rule
- Rule 2: If Priority(A) = Priority(B), A & B run in RR
- Rule 3: When a job enters the system, it is placed at the highest priority
- Rule 4: Once a job uses up its time allotment at a given level (regardless of how many times it gives up the CPU), its priority is reduced
  Round robin, 배정된 시간 -> 모두 time slice와 같음
  time slice만큼 쓰면 우선순위 낮아짐
  하나씩 낮춰 줌
- Rule 5: After some time period *S*, move all the jobs in the system to the topmost queue
  높은 priority로 boost
  주기 S마다 priority boost