Scheduling Levels.

- Low level scheduling. → Processor, CPU.
  Short-term scheduling.

- High level scheduling → Job scheduling.
  Long-term scheduling.

  ↳ 동시 실행 가능한 프로세스의 개수 결정.

- Mid level scheduling → Swapping.
  Mid-term scheduling.

* Refer page 9-10 in material 06.

스케줄링 기준

- CPU burst. vs IO burst.

CPU 필요성↑. CPU 필요성↓.

⇒금방 CPU 돌아옴.

Time Slice.

↓

스케줄링 스리드에 할당되는 CPU 시간.

- ∝ time quantum, time slot.

- Criteria.

CPU 활용률, CPU utilization, Efficiency
→ OS 입장, 전체 시간대비 CPU 사용 시간.

처리율, Throughput.
→ OS 입장, 단위시간당 처리 프로세스 개수.

공평성, Fairness, Load balancing.
→ 유저 입장. 스레드들에게 공평한 시간할당.

↓ (∝ >/< 스레드)

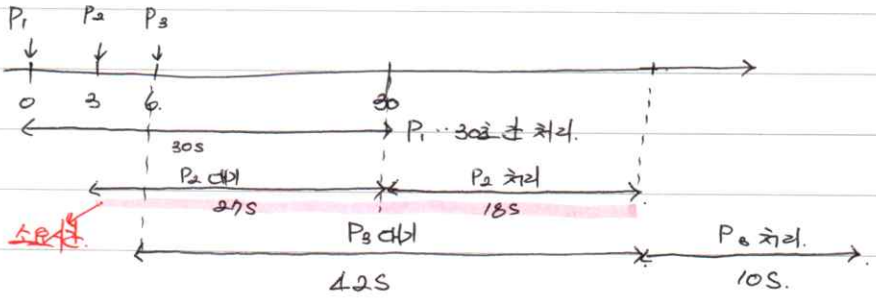
Time Slice. · Critia.

응답시간, Response time.
→ 명령 응답까지 걸리는 시간.

대기시간, Waiting time.
→ 스레드가 전체 큐에 머무는 시간 = CPU의 처리를 대기.

소요시간, 반환시간, Turn-around time.
→ 프로세스가 처리가 끝나는 데 걸리는 시간.
→ Σ응답시간 + Σ대기시간.

· 평균대기시간 · 평균반환시간.



평균 대기시간 = Σ대기시간 / 3.

평균 반환시간 = (Σ대기시간 + Σ처리시간) / 3.

CPU 스케줄링 발생
Page 16, Material 06.

1. 스레드, syscall 후 IO 요청 → 블록.
   CPU 활용률 향상
2. 스레드, 자발적 CPU 반환 (i.e. yield()).
   CPU 자발적 양보.
3. 스레드, 타임 슬라이스 소진 → 인터럽트.
   공평하게 CPU 분배
4. 더 높은 우선순위 스레드 처리.
   우선순위 이행.

scheduler & Dispatcher. (Page 18, Material 06).

- 매우 중요! 커널 내 코드.
  = 별도 실행되는 프로세스, 커널이 아님.

- Scheduler. → 스케줄러 타이머, 주기적 인터럽트.
  → 주기적으로 태스크 검사 … 실행 오래 지난 태스크
  → 스케줄링 필요시, 스케줄링 플래그 바꿈 설정.

  → 인터럽트 종료 (=리턴) ⇒ 플래그 확인, ! then call Dispatcher.

- Dispatcher. → 컨텍스트 스위칭 실행.

  → 스케줄러가 선택한 스레드, CPU가 실행하도록.

  → 커널 모드 ⟷ 사용자 모드.

- 둘 다 실행 시간 짧도록 최적화됨.
  "초경량"

#선점과 비선점.
Page 20 ~
Material 06.

- 선점, Preemptive - 현재 실행중인 스레드 강제종료.
  비선점, Non-preemptive - 강제 종료 X.

<선점>	<비선점>
• 타임 슬라이스 소진⇒타이머 인터럽트 더 높은 순위의 스레드 끼어듦.	CPU 뺏기게 됨 (IO … 블록, sleep). yield. 실행 종료.
• (복잡, 오버헤드, 컨텍스트 스위칭↑). (반응성↑)	(간단, 오버헤드↓ 컨텍스트 스위칭↓). (반응성↓).

우선순위.

· 숫자 {낮은수 / 높은수} 우선순위 ↑ ⇒ 시스템마다 다름.

<리눅스>

· Nice ... -20 ~ 20 → (5 / 10000) → 0 ~ 100 , 101 ~

(Realtime Process).

· 정적 스케줄링, 동적 스케줄링.

↳ 실행 중 변경된 우선순위 반영.

↳ 프로세스 종료까지 우선순위 변화 X.

Starvation, Aging.

· Starvation. 스레드, 스케줄링 선택 X ⇒ 오랫동안 장비 리소..

↓ solution

· Aging. 장비 리소에 머무는 시간 비례, 우선순위 ↑..

#스케줄링 ㅅ ㅠ

#스케줄링 알고리즘.

· FCFS, First Come First Served, 선입선출 비선점.

· SJF, Shortest Job First, 최단작업우선. 비선점.

· SRTF, Shortest Remaining Time First. 선점.

· Round-robin 선점.

↳ 들어가면서 할당된 시간을 실행.

· Priority. 선점·비선점

· Multilevel queue. 선점·비선점.

-Multilevel feedback queue. 선점·비선점.

morning glory

스케줄링 알고리즘

· FCFS

처리율 낮음. (먼저 신청한 처리하는 프로세스).

호위 효과 (Convoy Effect) 발생.

↳ 긴 프로세스가 CPU 오래 사용하면, 뒤에 짧은 짧은 프로세스 오래 대기.

· SJF

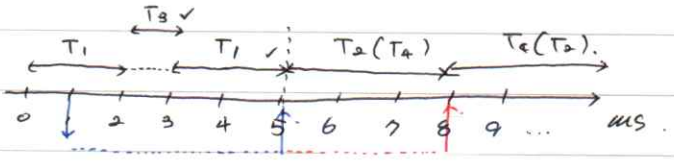
기아 발생 가능.

실행 시간 예측 불가능 ⇒ 사용 x.

평균 대기 시간 최소화.

* SRTF

	invoked	executing
T_1	0	4
T_2	1	3
T_3	2	1
T_4	5	3.



· T_1 대기시간: 1.

T_2 대기시간: 4+(3)

T_3 대기시간: 0.

T_4 대기시간: (3).

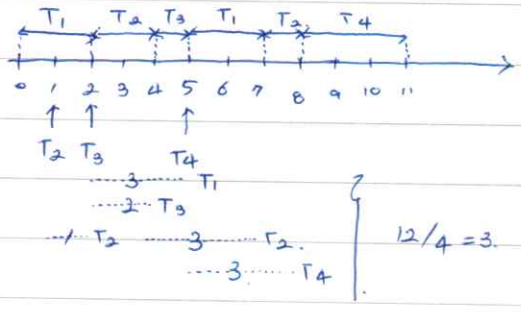
} 평균 2ms.

· 라운드로빈. RR. Round-Robin.

타임 슬라이스가 (너무 짧을 / 너무 길) 수 있음.

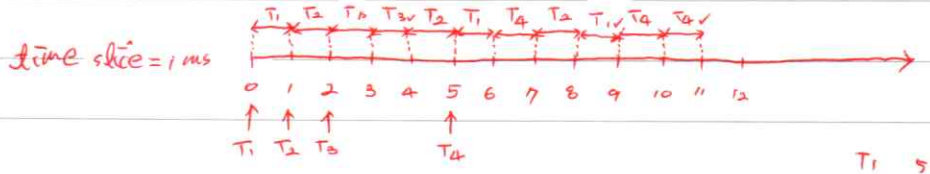
	invoked	executing			
T_1	0	4	T_3	2	1
T_2	1	3	T_4	5	3.

time slice = 2ms.



morning glory

· 라운드 로빈.



T_1	5
T_2	4
T_3	1
T_4	3

13/4

time slice=10ms then 22.33ms.

· 선점 스케줄링

· 공평. 기아×. 구현 난이도 ↓

· 짧은 스케줄링 → 오버헤드. 타임 슬라이스가 작으면 오버헤드 ↑.

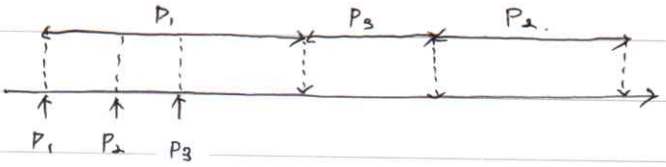
· 문맥의 처리율: Time slice ↑ FCFS / ↓ SJF/SRTF 에 가까움.

· 우선순위. { 스레드 중요 / 높은 우선순위 처럼 동작 } → 가장 높은 순위 스레드 선택.

{ 고정 우선순위 / 변동 } { 선점형 / 비선점형 }

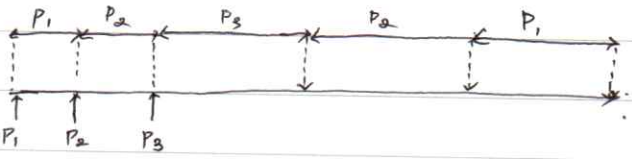
	Invoked	executing	priority
P_1	0	30	3
P_2	3	18	2
P_3	6	9	1

<비선점>



P_1: 0
P_2: 26
P_3: 34
60/3 = 20.

<선점>



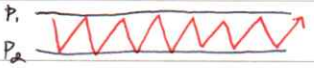
P_1: 27
P_2: 9
P_3: 0
36/3 = 12.

× 기아 발생가능.

morning glory

#스케줄링 알고리즘 · HRN, Highest Response ratio Next.

· SJF + Aging.
· 우선순위 : (대기시간 + CPU 사용시간) / (CPU 사용시간).
· 반드시 바뀐점 … 선점하면 계속 스위칭(핑퐁핑퐁핑퐁…).



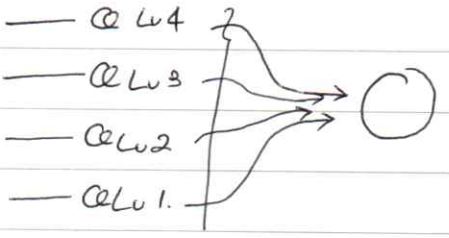
Job	Waiting	Processed	Priority	
A	5	20	(5+20)/20	1.25
B	40	20	(40+20)/20	3 ←
C	15	45	(15+45)/45	1.33…
D	20	20	(20+20)/20	2

* 정적 우선순위 : B 종료 확인도 후 곧 그대로 사용.
동적 우선순위 : B 종료후, B 러닝타임 차감 후 재평가

· 차이 : 발생 가능성 "높음" … 누적에 의해 처리되므로, 수시 조건에만 발생.

Single Queue
↓
Multi Queue

· MLQ, Multi-Level Queue.



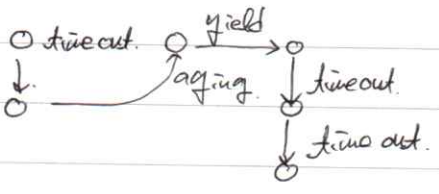
· 스레드, n개의 우선순위 레벨로 구분.
→우선순위 순 처리.
· "우선순위의 그룹화"

· 높은 순위의 큐 빌 때 다음 순위 큐 스케줄링

· MLFQ, Multi-Level Feedback Queue.

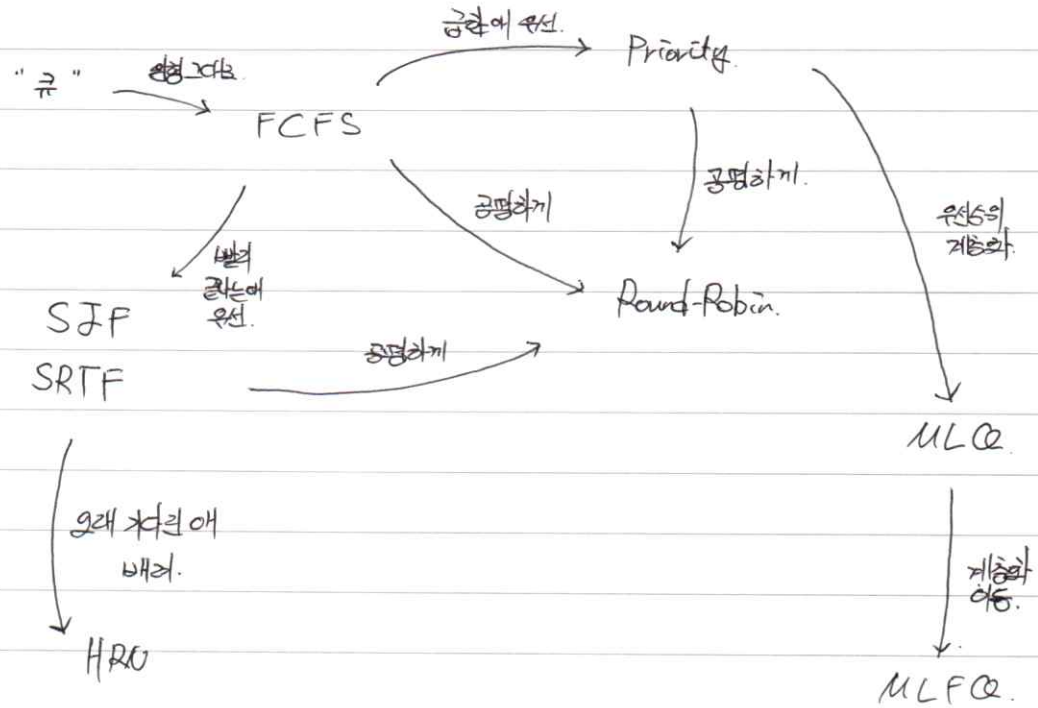
· MLQ, 차이 발생 가능 ⇒ 레벨 변경 가능한 스펙.
+) 정책에 따라 메커니즘 상이.

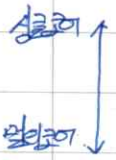
i.e. { Q Lv4 (time slice = 4ms)
Q Lv3 (= 6ms)
Q Lv2. (= 8ms).



morning glory

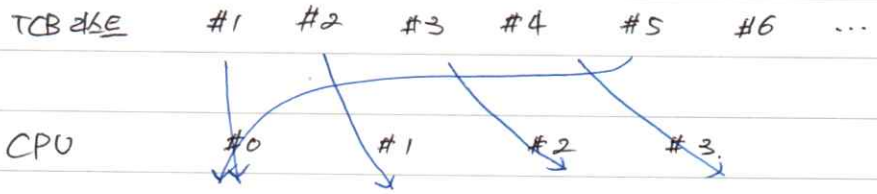
스케줄링 전략의 관계들.





멀티코어 CPU 스케줄링

· 멀티코어 시스템에서의 멀티스레딩.



· "한 처리에 넣기."

문제점 P1
· 쉽지 않다 ... "컨텍스트 스위칭". → 새 다른 코어로 스위칭은 오버헤드 ↑.

P2
· CPU 부하 불균형 CPU의 작업량 ↑.

· S1 코어는 스위칭 문제: 코어 친화성, CPU 피닝(CPU Pinning).
↳동일 코어에서만 실행하도록 스케줄링.

morning glory

멀티코어 CPU 스케줄링.

· 멀티코어 시스템에서의 멀티프로세싱.

- 부하 불균형 문제 → 균등화…
  - 푸시 마이그레이션. 남한테 작업 밀어냄.
  - 풀 내한테 작업 당겨옴.

… Windows <선호도 설정>
Linux `taskset`

* 멀티프로세싱 … 공유데이터 문제. 캐시..!
  → 서로 다른 스레드-합 맞춰 필요 "동기화" (짧은 스케줄링).