운영체제 Chapter 6

- Cpu Scheduling -

1. Basic Concepts
   - 멀티프로그래밍에서 cpu utilization 최대화하기
   - cpu burst : cpu를 사용하는 구간
   - cpu-bound : cpu burst time이 길다
   - I/O-bound : cpu burst time이 짧음

2. cpu burst time의 분포
   - 대부분의 cpu burst는 8ms 보다 짧음
   - cpu를 약 3ms동안 점유하는 빈도가 가장 높음

3. CPU scheduler
   - short-term scheduler (= cpu scheduler)
     - ready 상태의 프로세스를 대상으로 cpu를 사용할 프로세스를 선정하는 것
   - cpu scheduling은 언제 일어날까?
     1) running -> waiting
     2) running -> ready
     3) waiting -> ready
        ex) 현재 실행중인 프로세스보다 waiting에서 ready로 간 프로세스의 우선순위가 더 높을 때
     4) running -> terminated
   - 위의 1,4는 non-preemptive 방식
     - 현재 실행중인 프로세스가 자발적으로 cpu를 그만 실행하고 내놓는 상황임
   - 2, 3은 preemptive 방식
     - cpu를 점유하기 위해 뺏는 방식

4. Dispatcher 역할
   - 새로운 프로세스를 선택
     - 프로그램을 재시작하기 위해 유저 프로그램에서 적절한 위치로 점프함
   - 문맥 교환
   - 새로운 프로세스에게 cpu 할당
     - 유저 모드로 스위칭
   - Dispatch latency
     - A프로세스에서 B프로세스로 실행전환까지 소요되는 지연시간 (길어서 좋을 것 없음)

5. Scheduling Criteria (스케줄링에 따른 성능척도 기준)

- CPU utilization (활용도)
  - cpu가 놀지 않고 얼마나 많이 사용되는가? 에 대한 기준
  - cpu가 의미 있게 사용되는 시간을 높이는 것이 곧 cpu 활용도를 높이는 방법
- Throughput (처리량, 생산성)
  - 단위시간당 실행이 완료된 프로세스의 개수

-> 위의 둘은 시스템 전체에 대한 성능 척도

- Turnaround time (반환시간)
  - 프로세스가 실행에 돌입하여 완료되기까지의 시간
  - 종료시간에서 도착시간을 뺀 값
- Waiting time (대기시간)
  - wait queue에서 기다린 시간
  - Turnaround time에서 cpu burst을 뺀 값
- Response time (응답시간)
  - 프로세스가 시작해서 첫 번째 출력이 나오는데 까지 걸리는 시간
  - 처음 응답시간에서 도착시간을 뺀 값

-> 위의 셋은 개별 프로세스에 대한 시간성능 척도

∵ **시스템 전체 척도와 개별 프로세스에 대한 척도 둘 다 만족하기엔 어렵다**

6. FCFS 스케줄링 (non - preemptive)

- ready 큐에 먼저 도착한 순서대로 cpu를 사용하게 하는 방식
- convoy effect : cpu 사용이 짧은 작업이 긴 작업을 오래 기다리는 상황
  -> I/O bound 프로세스가, CPU bound 프로세스를 기다리는 상황

7. Shortest-Job-First (SJF) (non - preemptive)

- cpu burst time이 짧은 프로세스 먼저 cpu를 사용하게 하는 방식
- waiting time을 최소화 할 수 있음
- 이상적인 스케줄링 방식이긴 하나, 현실에선 cpu burst time을 파악하기 어렵다.

8. 다음 cpu burst의 길이를 추정하는 방법

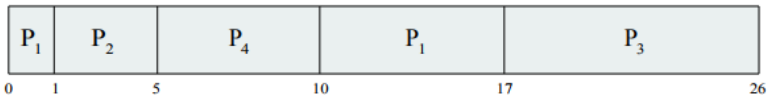
- t[n] = 실제 n번째의 cpu burst의 길이
- estimate[n+1] = 다음번 cpu burst의 예측 값
- 0<= a <=1
- estimate[n+1] = a * t[n] + (1-a) * estimate[n]
  - 이번 실측값 t[n]과 이번 예측값 estimate[n]으로 다음번 예측값을 구할 수 있음
  - a가 0이면, estimate[n+1] = estimate[n]
  - a가 1이면, estimate[n+1] = t[n]
- 일반적으로 알파(a) 는 1/2로 set함

9. Shortest-Remaining-Time-first (SRT) (preemptive)

- 실행이 끝날 때까지 cpu burst가 가장 짧은 프로세스를 먼저 실행
- 반환시간과 대기시간을 구해보자

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

Preemptive SJF Gantt Chart



프로세스	도착시간	cpu burst	종료시간	반환시간 (종료시간 - 도착시간)	대기시간 (반환시간 - cpu burst)	
p1	0	8	17	17	9	
p2	1	4	5	4	0	
p3	2	9	26	24	15	
p4	3	5	10	7	2	
평균				13.00	6.50	

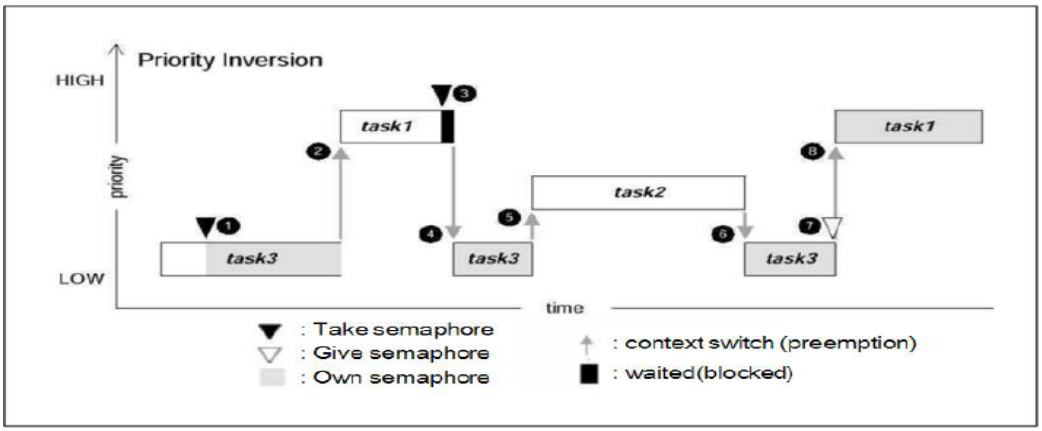
10. 우선순위 스케줄링 (preemptive) (non-preemptive)

• 각 프로세스마다 우선순위를 통해 높은 순서대로 실행

- 기본적으로 우선순위 숫자가 작은 것이 우선순위가 높은 것임

• preemptive 방식에서의 문제점

- starvation : 우선순위가 낮은 프로세스가 계속해서 실행에 밀리는 경우
- priority inversion : 우선순위가 높은 프로세스가 낮은 프로세스를 기다리게 되는 현상



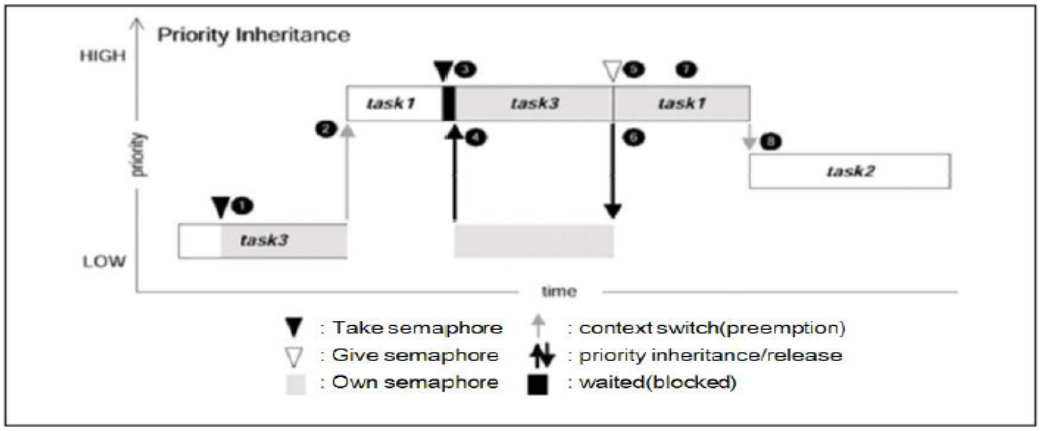
task3이 세마포어를 먼저 얻었고, preemptive 우선순위 스케줄링에 의해 task1이 늦게 도착했음에도 cpu를 점유한다. 그러나 task1은 세마포어를 얻으려고 하는 과정에서 task3이 세마포어를 반환할 때까지, blcok 상태에 돌입하게 된다. 스케줄러에 의해 task3은 세마포어를 반환하도록 실행되던 도중, task2가 task3보다 우선순위가 높기 때문에 cpu 점유를 빼앗기게 된다. task1보다 우선순위가 낮은 task2가 cpu를 점유하는 우선순위 역전 현상이 ⑤번 위치에서 발생한다.

• starvation 해결방법

- aging : ready큐에 머무른 시간을 반영해서 우선순위를 높인다.

• priority inversion(우선순위 역전) 해결방법

- priority inheritance(donation) 우선순위 상속



task1이 세마포어를 얻으려는 지점에 task3의 우선순위를 task1의 우선순위만큼 높인다. task3이 최대한 빨리 세마포어를 반환하도록 하기 위함이며, task3이 세마포어를 반환하는 순간 우선순위를 원래대로 낮춘다. block상태에 있던 task1은 세마포어를 얻어 cpu실행에 옮긴다.

11. Round Robin Scheduling

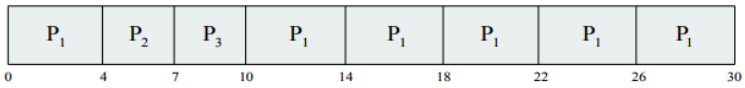
- 준비 큐에 도착한 순서에 따라 cpu를 점유하지만 time quantum에 의해 실행을 제한하며 time quantum 안에 완료되지 못한 프로세스는 ready 큐의 맨 뒤에 배치된다.
- response time이 짧아 첫 번째 I/O 결과가 사용자의 눈에 들어오기 까지 시간이 짧다.
- time quantum이 크면 cpu 독점시간이 길어져 FCFS와 다를게 없음..
- time quantum이 대부분의 cpu burst를 커버하지 못할 만큼 작다면, 불필요한 문맥교환이 자주 발생한다. -> overhead 발생

∵ **적절한 time quantum을 설정하는 것이 중요함, 대부분의 IO bound의 cpu burst를 커버할 수 있을 만큼은 돼야 한다.**

Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

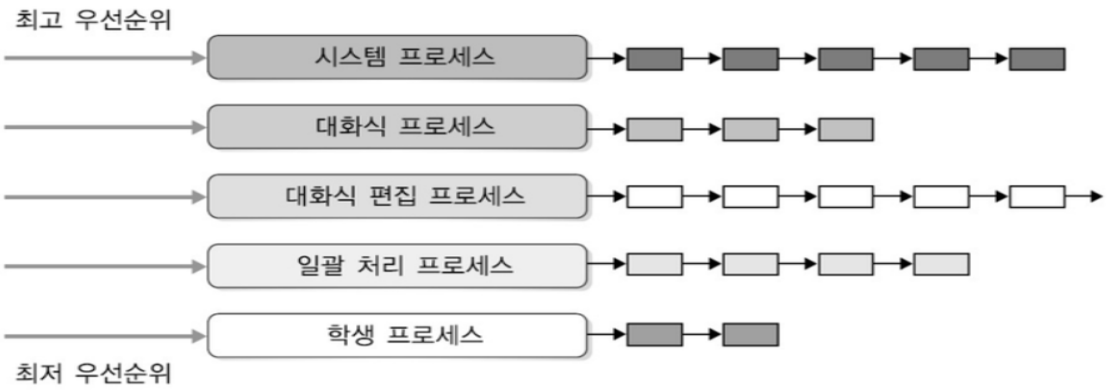
- The Gantt chart is:



프로세스	도착시간	cpu burst	종료시간	반환시간 (종료시간 - 도착시간)	대기시간 (반환시간 - cpu burst)	응답시간 (첫번째 응답시간 - 도착시간)
p1	0	24	30	30	6	4 - 0 = 4
p2	0	3	7	7	4	7 - 0 = 7
p3	0	3	10	10	7	10 - 0 = 10
평균				15.67	5.67	

12. Multilevel Queue

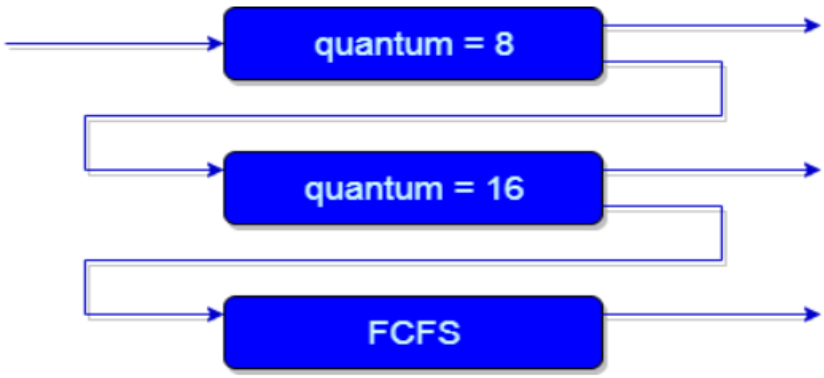
• 프로세스 종류에 따라 그룹으로 나누어 여러 개의 큐에 다양한 알고리즘을 적용하는 스케줄링 기법
  - foreground 즉, interactive(상호작용방식, 대화식) 으로 작업을 처리해야 하는 전면 작업의 우선순위가 높음(response time이 짧아야하기 때문에)
  - background 즉, batch(일괄처리) 방식으로 작업을 처리하는 후면 작업은 우선순위가 낮은 쪽에 속함
  - foreground는 Round Robin 스케줄링을 수행하기 때문에 I/O bound 프로세스를 우선순위가 높은 위쪽 큐에 집어넣음
  - background는 FCFS 기법을 사용하므로 CPU bound 프로세스가 주로 삽입됨
• 프로세스는 오로지 한 큐에만 삽입된다. (프로세스들이 큐를 이동할 수 없음)



• 우선순위가 높은 ready큐 먼저 스케줄링을 하고, 우선순위가 낮은 쪽으로 내려가면서 스케줄링 함

13. Multilevel Feedback Queue (preemptive)

• 프로세스의 행동패턴에 따라 매번 ready큐에 줄을 설 때 마다 다른 큐에 줄을 설 수 있다.



• cpu를 매번 time quantum 8이하만큼 사용한다면 첫 번째 큐에 머물겠지만 time quantum 내에 작업을 처리하지 못했다면, 다음 큐로 내려가는 방식이다.
• 아래 큐로 내려갈수록 cpu를 점유할 기회가 줄어들지만 받으면 오래 사용함
• 아래의 큐에서 위로 올라가는 건 일정시간동안 수행되지 못하고 아래에 오래 남아있을 경우에 상위 큐로 올려 실행함

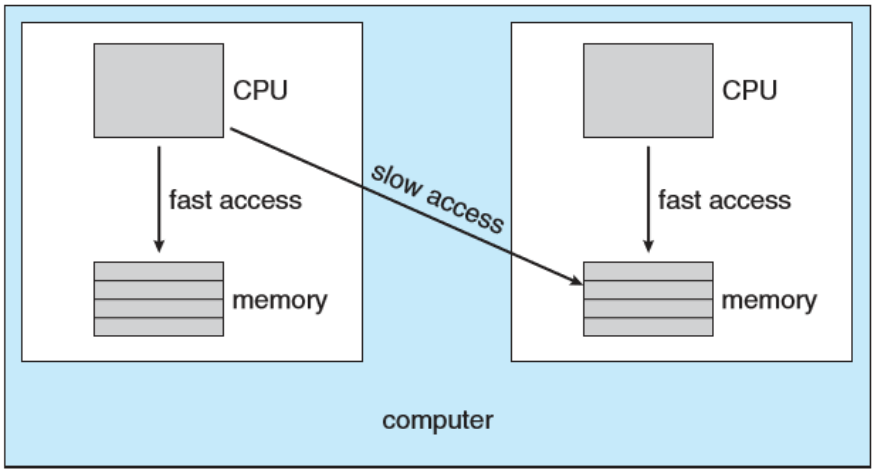
14. Multiple Processor Scheduling (별로 안중요..)

• 다중 처리기 시스템의 형태에 따라 크게 두 가지로 분류

- Asymmetric multiprocessing
- Symmetric multiprocessing

둘을 나누는 기준은 여러 개의 cpu가 있을 때 메모리 접근에 있어서 일관성이 있느냐 없느냐가 기준!
cpu별 메모리 접근속도가 모두 동일하다면 SMP 그렇지 않으면 ASMP

• ASMP의 예



cpu별로 main memory가 따로 있고, 가끔 서로의 메모리를 접근할 수도 있는데, 접근 속도가 한결같지 않다! non-uniform하다!
즉, 다른 memory를 접근할 때는 속도가 느리기 때문에 대칭적이지 않다!

• Processor affinity

- 프로세서 친화도(대충 친화성 그런 정도로 해석)
- 처리기에 친숙한 프로세스를 그 처리기에 다시 할당하는 방법으로 cache로 인해 성능 상 효율을 노릴 수 있음

15. Real-Time CPU Scheduling

• 여태까지 다뤘던 스케줄링기법은 Best 효율을 위한 스케줄링 이었으나, Real-time cpu 스케줄링은 이벤트에 대응하기 위한 최선의 스케줄링기법

- 이벤트는 마감기한을 뜻함
- 프로세스마다 마감기한이 있을 때, 그것을 만족시켜주기 위한 스케줄링

• soft real-time systems은 마감기한을 꼭 지켜야하는 것은 아님

• Hard real-time systems은 마감기한을 필수적으로 지켜야함

16. Rate Montonic Scheduling

• 수행 주기가 가장 짧은 프로세스에 가장 높은 우선순위를 부여하는 방식

17. Earliest Deadline First Scheduling

• 우선순위를 동적으로 변경하여 deadline이 가장 빠른 process에게 가장 높은 우선순위를 부여하는 방식