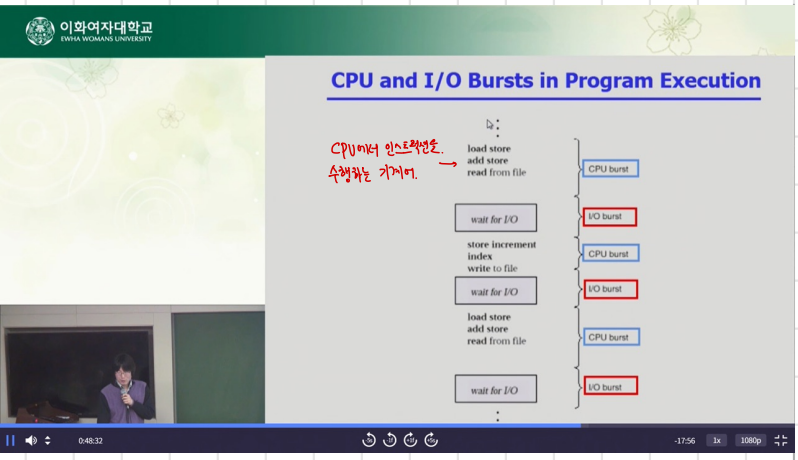
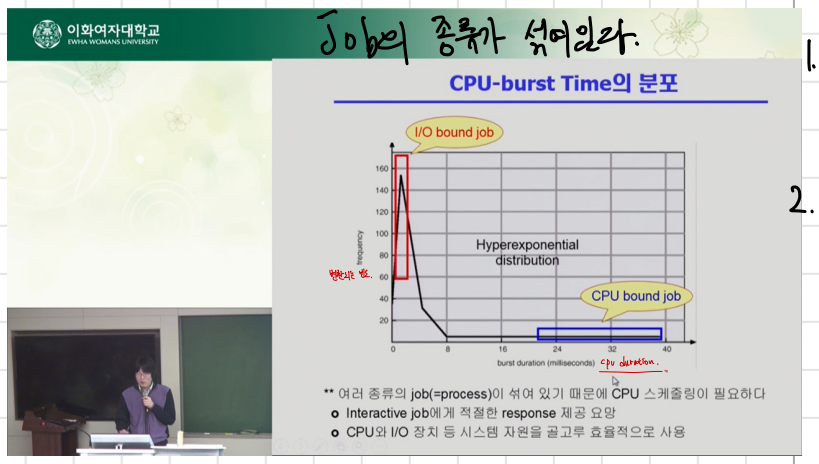


1. CPU burst와 I/O burst가 번갈아 나온다.

2. CPU burst만 나오다가 I/O가 나올수도, 번번하게 두 개가 번갈아 나올 수 있다.



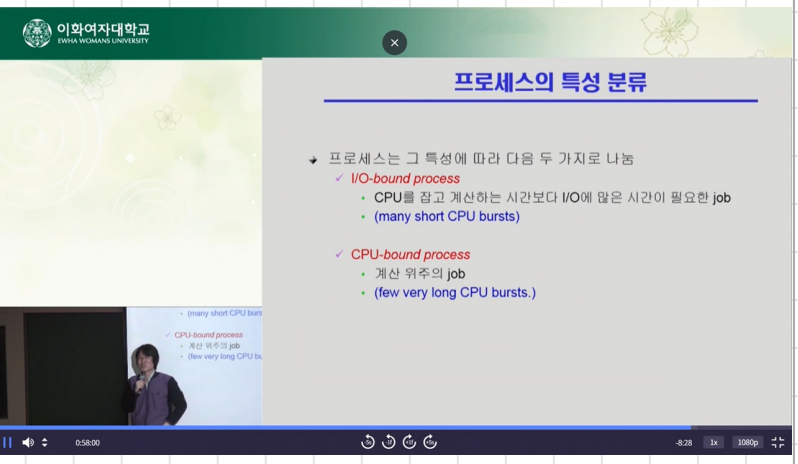
Job의 종류가 섞여있다.

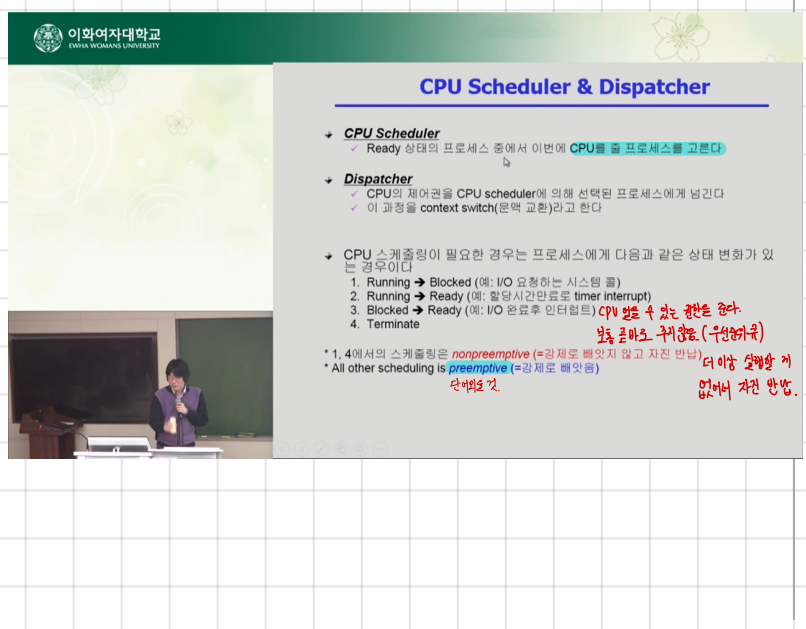
1. I/O bound Job 번갈아 실행되는 빈도가 높은 프로세스

2. CPU bound Job CPU가 집중적 실행되는 프로세스.

CPU 스케줄링시 주의점.
공평한 것이 아닌 효율적 측면.
Interactive 한 잡을 오래 기다리지 않게 하는것.

1. CPU를 줄 것인지.
2. 뺏을 것인지.





세심하게 짚고 넘어가야 할 사항.

1. CPU Scheduler는 하드웨어가 별도로 담당하는지.
   해당 프로그램이 별도로 있는지.
   ⇒ 운영체제에 cpuScheduling하는 코드가 별도로 있음.
   그것을 CPU Scheduler라 한다.

CPU Scheduling Algorithm.

1. Nonpreemptive한 알고리즘.
2. Preemptive한 알고리즘. 요즘 대부분 이방식.

Scheduling Criteria

성능척도.

1. 시스템 입장에서 성능척도.

- **CPU utilization (이용률)**
  keep the CPU as busy as possible
- **Throughput (처리량)**
  ※ # of process that complete their excution per time unit

2. 프로그램 입장에서 성능척도.(프로세스)

- **Turnaround time (소요시간, 반환시간)**
  amount of time to excute a particular process CPU 다쓰고 나갈때까지 걸린 시간. * CPU Scheduling 관점이기 때문에 프로세스가 종료될 때 까지가 아닌 I/O 할때까지 총 시간임.
- **Watting time (대기시간)**
  amount of time a process has been watting in the ready queue 기다리는 시간

Watting time과 Response time의 차이.
프로세스 A가 CPU 뺏겼다 뺐었다 반복.
뺏겨 Ready queue에 서서 쳐 기다리는 시간 이므로.
총합 != watting time., 첫 서비스 = Response time.

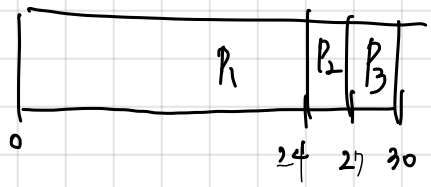
- **Response time (응답시간)** time sharing시 중요.
  amount of time it takes from when a request was submitted until the first response is produced, not output Ready queue에 들어와서 처음으로 뽑힌 시간.
  (for time-sharing environment)

Scheduling Algorithms

1. FCFS (First Come First Served)

- nonpreemptive.

P_2, P_3가 interactive한 job 이면 대부분 비효율적.

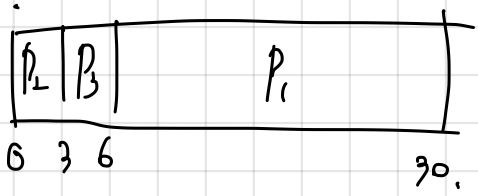


	Waiting Time
P_1	0
P_2	24
P_3	27

$Mean = \frac{0+24+27}{3}$

$= 17$

if.



	Waiting time
P_2	0
P_3	3
P_1	6

$Mean = \frac{0+3+6}{3}$

$= 3.$

Much better than previous case.

Convoy effect : Short process behind long process

2 SJF(Shortest-Job-First)

CPU burst time이 가장 짧은 프로세스를 제일 먼저 스케줄.

Minimum average waiting time 보장.

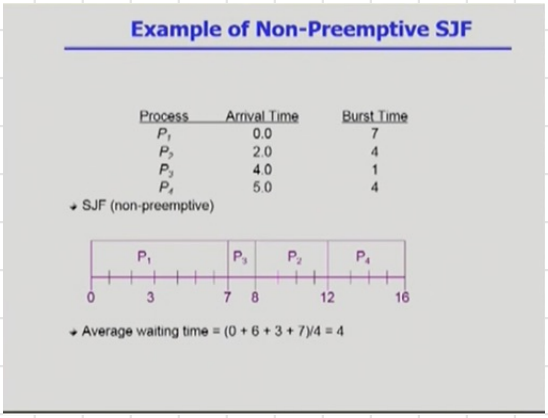
(preemtive한 방식)

두 가지 방식.

1. Nonpreemtive.

2. Preemtive. (=Shortest-Remain-Time-First)

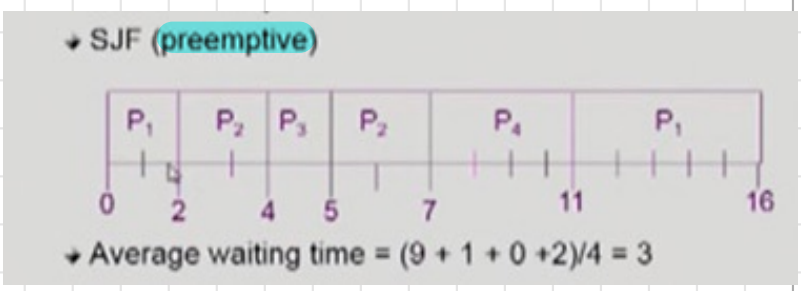
수행중인 프로세스보다 더 짧은 Cpu burst 프로세스가 온면 이를 수행.



0초 시점에 도착프로세스 없음.

P1의 Burst Time이 끝난 시점에 P2~P4 모두 도착.

이 중에서 CPU Burst 타임이 작은것 수행.



스케줄링을 하는데 있어서 도착시 변경가능.

문제점.

1. Starvation 문제.
   CPU burst가 처리X

2. CPU 사용시간을 미리 알기 어렵다.
   추정은 가능 (과거 CPU 사용량을 보고 추정한다.)
   exponential averaging
   1. t_n = actual length of n^{th} CPU burst
   2. τ_{n+1} = predicted value for next CPU burst.
   3. α, $0 \le \alpha \le 1$
   4. Define $\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n$
      ⇒ 일반식, $\tau_{n+1} = \alpha t_n + (1-\alpha)\alpha t_{n-1} + \cdots$
      $+ (1-\alpha)^j \alpha t_{n-j} + \cdots$
      $+ (1-\alpha)^{n+1} \tau_0$

3. Priority Scheduling

highest priority를 가진 프로세스에게 CPU할당.
(Smallest Integer = highest priority)

Preemtive
Nonpreemtive. 두 방법 존재.

SJF : priority = predicted next CPU burst time.

Starvation Solution.
Aging : as time progresses increase the priority of the process

4. Round Robin (RR)

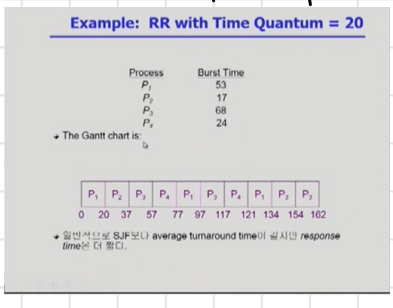
현대컴퓨터에서 기본이 되는 스케줄.

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

- n개의 프로세스가 ready queue에 있고, 할당 시간이 q time unit인 경우
  각 프로세스는 최대 q time unit 단위로 CPU 시간의 1/n을 얻는다.
  ⇒ 어떤 프로세스도 (n-1)q time unit 이상 기다리지 않는다.
  → 반응성↑

- Waiting time ∝ CPU burst.

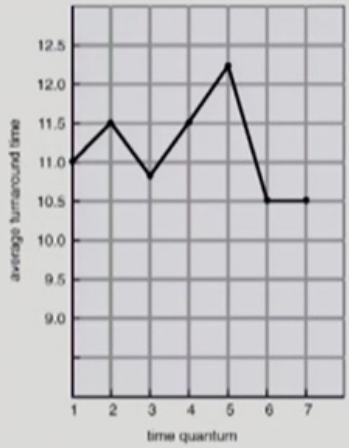
- Performance
  q large ⇒ FCFS
  q small ⇒ context switch 오버헤드가 커진다.
  ⇒ 적당한 크기의 q를 유지하자. (10~100 milsecond)



- CPU 사용시간이 모두 동일한 프로그램에서 n개 일 경우,
  (ex: 100초 4개)
  400초일 때 4프로세스 모두 종료.
  1개씩 돌리면 100초마다 1개씩 빠져나가므로 Waiting time↓.

라운드 로빈은 무작위 CPU burst일 경우 효율적으로 적용된다.

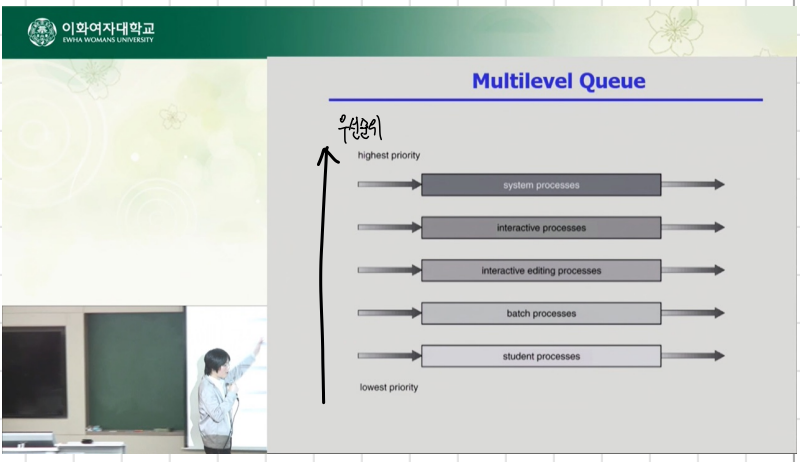
Turnaround Time Varies With Time Quantum

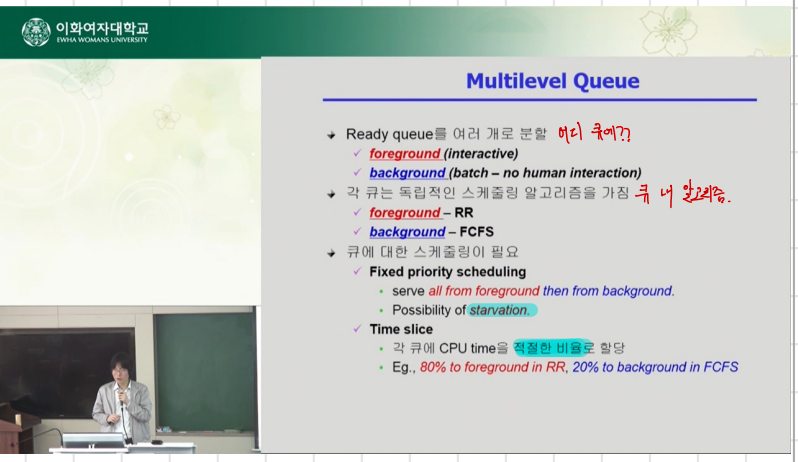


process	time
P_1	6
P_2	3
P_3	1
P_4	7

6
3
1
7

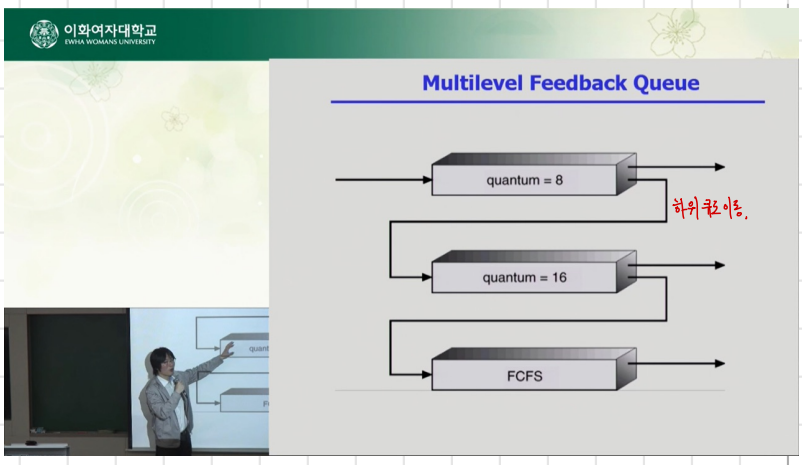
average turnaround time





줄이 여러 개인 CPU는 하나기 때문에 고려해야 할 사항이 존재.

1. 프로세스를 어느 줄에 세울 것인가?
2. 낮은 우선순위 큐의 Starvation



보통

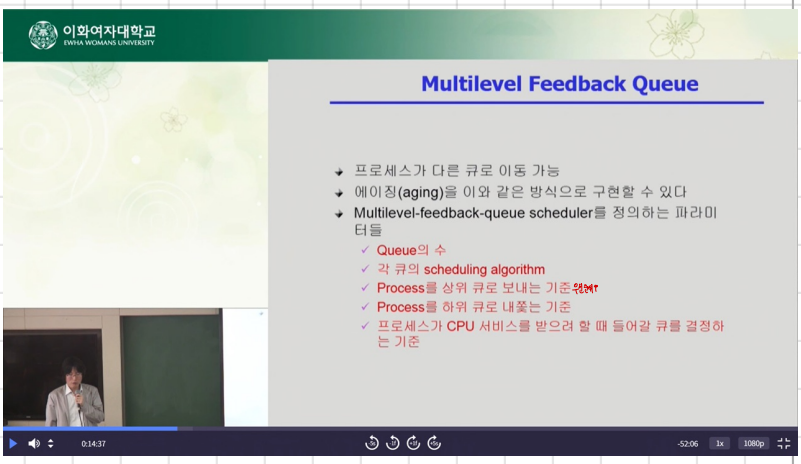
1. 첫번째 프로세스는 상위 큐,
2. 두번째 프로세스는 하위 큐 ...

상위큐에서 모두 작동되지 않으면

하위큐로 이동해서 처리.

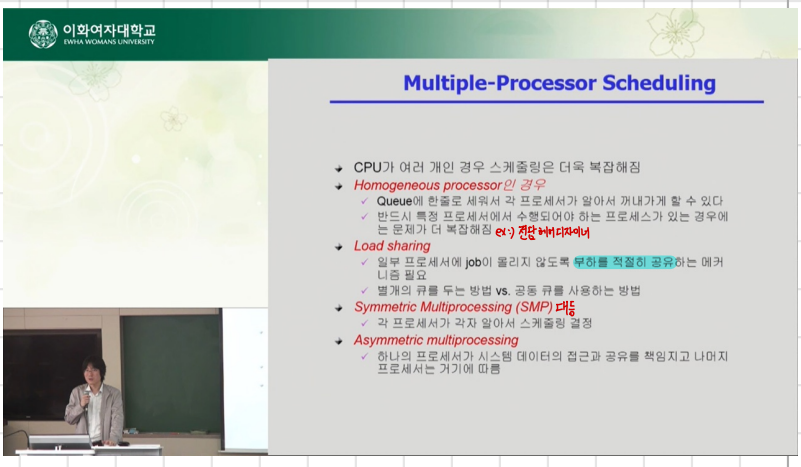
즉, CPU 사용시간이 짧은 프로세스에게 우선순위가 높다.

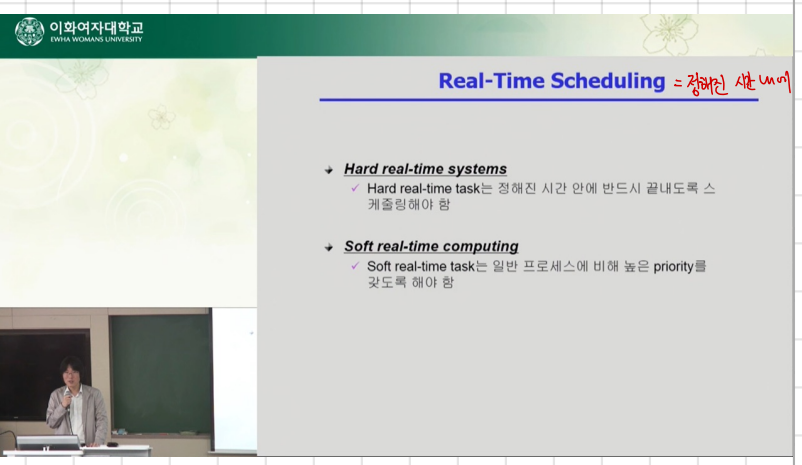
CPU 사용시간 예측값이 짧은 프로세스가 우선시된다.



↑ 단일 프로세서

↓ 멀티프로세서 or 시간 dead line 조건 or 다중 Thread.





= 정해진 시간 내에 끝내야 하는 Job.

. Real-Time Job
  보통 주기성을 가진다.

. Soft real-time Job
  . 영화같은것. 데드라인은 꼭 엄격하지 못함.

이화여자대학교 EWHA WOMANS UNIVERSITY

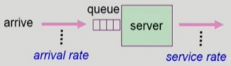
Thread Scheduling

- ***Local Scheduling***
  - ✓ User level thread의 경우 사용자 수준의 thread library에 의해 어떤 thread를 스케줄할지 결정
- ***Global Scheduling***
  - ✓ Kernel level thread의 경우 일반 프로세스와 마찬 가지로 커널의 단기 스케줄러가 어떤 thread를 스케줄할지 결정

프로세스 내부에서 결정.

이화여자대학교 EWHA WOMANS UNIVERSITY

Algorithm Evaluation



- ***Queueing models***
  - ✓ 확률 분포로 주어지는 *arrival rate*와 *service rate* 등을 통해 각종 *performance index* 값을 계산
- ***Implementation (구현) & Measurement (성능 측정)***
  - ✓ 실제 시스템에 알고리즘을 구현하여 실제 작업(*workload*)에 대해서 성능을 측정 비교
- ***Simulation (모의 실험)***
  - ✓ 알고리즘을 모의 프로그램으로 작성후 *trace*를 입력으로 하여 결과 비교

실제 프로그램에서 추출한 data.

· 알고리즘 평가 방법.

1. Queueing models

광장히 이론적인 방법.

server = CPU

2.

만약 linux CPU Scheduling과 새로만든 CPUScheduling을 비교한다면, linux kernel source code를 수정하여 새로운 알고리즘을 구현한 커널을 컴파일하여 bin code 생성.

실제 프로그램을 돌려 어떤 스케줄러가 좋은 지 평가.

but, kernel 수정은 매우 힘든 작업.

3. Simulation.

복잡한 프로그램에 대해 모의프로그램을 작성하여 계산.