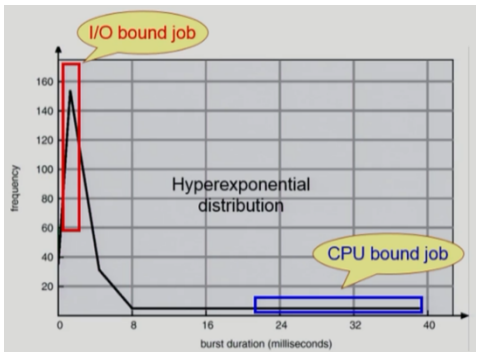
CPU 스케줄링

생성일	@2022년 4월 2일 오후 1:27
태그	

프로세스의 특성 분류



- 프로세스는 그 특성에 따라 다음 두 가지로 나눈다.
  - IO-bound process
    - CPU를 잡고 계산하는 시간보다 IO에 많은 시간이 필요한 job
    - many short CPU bursts
  - CPU-bound process
    - 계산 위주의 job
    - few very long CPU bursts
- 여러 종류의 job(=process)가 섞여 있기 대문에 CPU 스케줄링이 필요하다
  - interactive한 job에게 적절한 response 제공
  - CPU와 IO 장치 등 시스템 자원을 골고루 효율적으로 사용

CPU Scheduler & Dispatcher

- CPU Scheduler
  - Ready 상태의 프로세스 중에서 이번에 CPU를 줄 프로세스를 고른다.

- Dispatcher
  - CPU의 제어권을 CPU scheduler에 의해 선택된 프로세스에게 넘긴다
  - 이 과정을 context switch라 한다
- CPU 스케줄링이 필요한 경우는 프로세스에게 다음과 같은 상태 변화가 있는 경우다
  - 1) Running → Blocked (ex. IO 요청하는 시스템 콜)
  - 2) Running → Ready (ex. 할당시간만료로 timer interrupt)
  - 3) Blocked → Ready (ex IO 완료 후 인터셉트)
  - 4) Terminated

- 1) 4)는 non-preemptive (= 강제로 빼앗지 않고 자진 반납), 나머지는 preemptive (= 강제로 빼앗음)

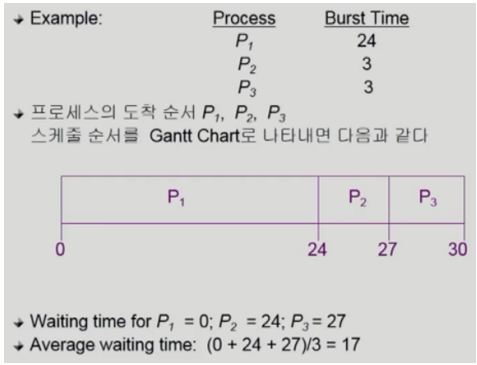
Scheduling Criteria (Performance Index)

- 성능척도 : 스케줄링 알고리즘 중 무엇이 더 좋은지 판단하는 지표
- CPU Utilization (이용률)
- Throughput (처리량)
- Turnaround Time (소요시간, 반환시간)
- Waiting Time (대기시간)
- Response Time (응답시간)

Scheduling Algorithms

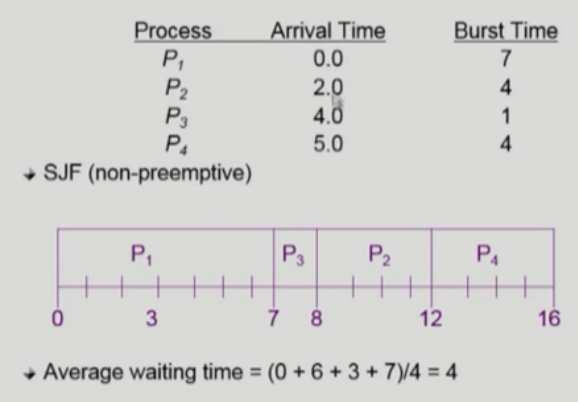
FCFS (First-come First-served)

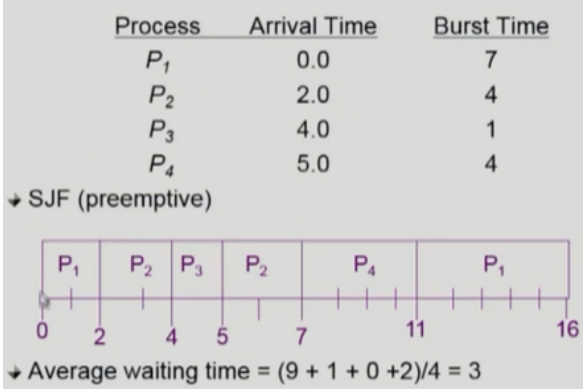
- non-preemptive
- CPU 스케줄링에서는 매우 비효율적인 방법
- Convoy Effect : short process behind long process → average waiting time이 너무 길어짐



SJF (Shortest-Job-First)

- 각 프로세스의 다음번 CPU burst time을 가지고 스케줄링에 활용
- CPU burst time이 가장 짧은 프로세스를 제일 먼저 스케줄
- Two schemes
  - Non-preemptive : 일단 CPU를 잡으면 이번 CPU burst가 완료될 때까지 CPU를 선점(preemption)당하지 않는다.
  - Preemptive : 현재 수행중인 프로세스의 남은 burst time보다 더 짧은 CPU burst time을 가지는 새로운 프로세스가 도착하면 CPU를 빼앗긴다. (Short-Remaining-Time-First, SRTF)
- SJF is optimal : 주어진 프로세스에 대해 minimum average waiting time을 보장한다!!
  - 하지만 치명적인 약점, starvation을 발생 : 너무 효율성만 생각해서 long job은 영원히 cpu를 못 얻을수도 있음.





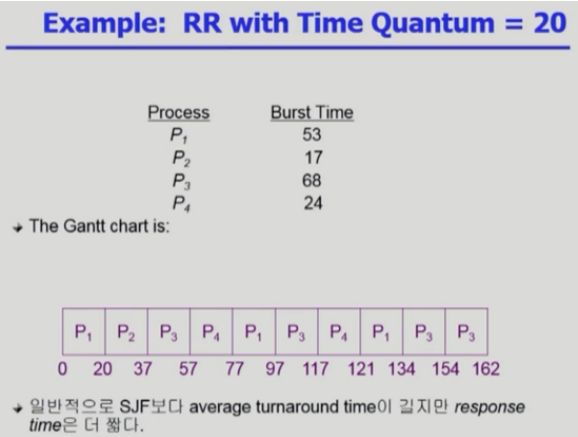
- 다음 CPU Burst Time의 예측
  - 추정만 가능, 과거의 CPU burst time을 이용해서 추정 (exponential averaging)

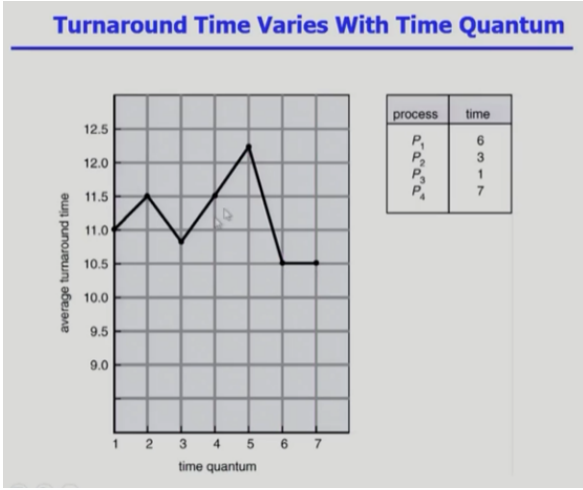
Priority Scheduling

- higher priority를 가진 프로세스에게 CPU를 할당
- SJF도 일종의 priority scheduling
  - priority = predicted next CPU burst time
- 문제점 : low priority process may never execute (starvation)
- 해결책 : as time progresses increase the priority of the process (aging)

RR (Round Robin)

- 각 프로세스는 동일한 크기의 할당 시간(time quantum)을 가진다. (일반적으로 10-100ms)
- 할당 시간이 지나면 프로세스는 선점(preempted)당하고 ready queue의 제일 뒤에 가서 다시 줄을 선다.
- 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!
- 성능
  - q large : FCFS
  - q small : context switch 오버헤드가 커진다

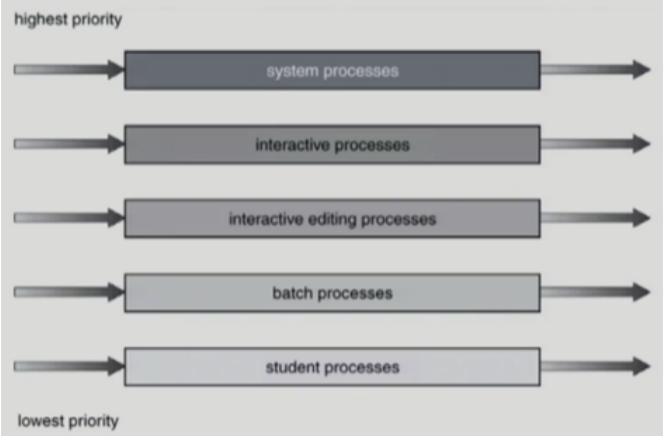




Multilevel Queue

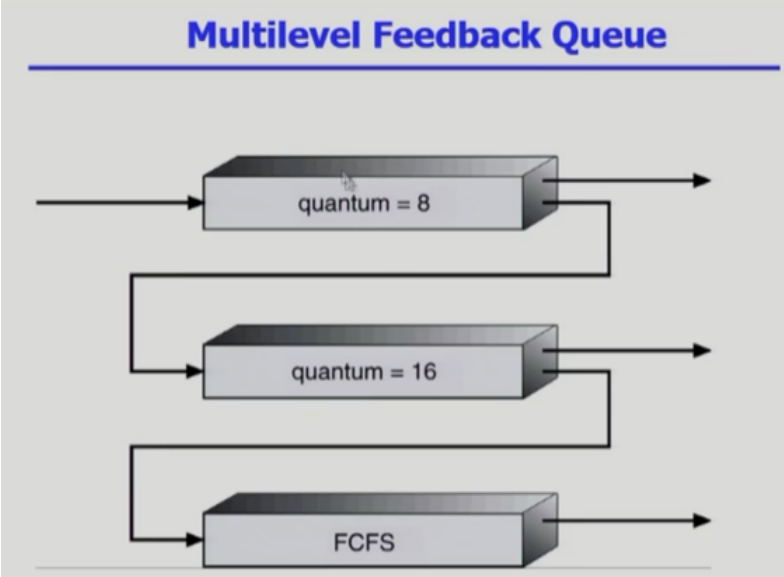
- Ready Queue를 여러 개로 분할
  - foreground(interactive)
  - background(batch - no human interaction)
- 각 queue는 독립적인 스케줄링 알고리즘을 갖느다.
  - foreground - RR
  - background - FCFS
- 각 queue에 대한 스케줄링이 필요하다
  - Fixed priority scheduling
    - serve all from foreground then from background
    - possibility of starvation
  - Time slice
    - 각 큐에 CPU time을 적절한 비율로 할당

    ex) 80% to foreground in RR, 20% to background in FCFS



Multilevel Feedback Queue

- 프로세스가 다른 queue로 이동 가능
- aging을 이와 같은 방식으로 구현할 수 있다.
- Multilevel Feedback Queue sceheduler를 정의하는 파라미터들
  - queue의 수
  - 각 queue의 scheduling algorithm
  - process를 상위 큐로 보내는 기준
  - process를 하위 큐로 보내는 기준
  - process가 cpu 서비스를 받으려 할 때 들어갈 queue를 결정하는 기준


Multilevel Feedback Queue
quantum = 8
quantum = 16
FCFS