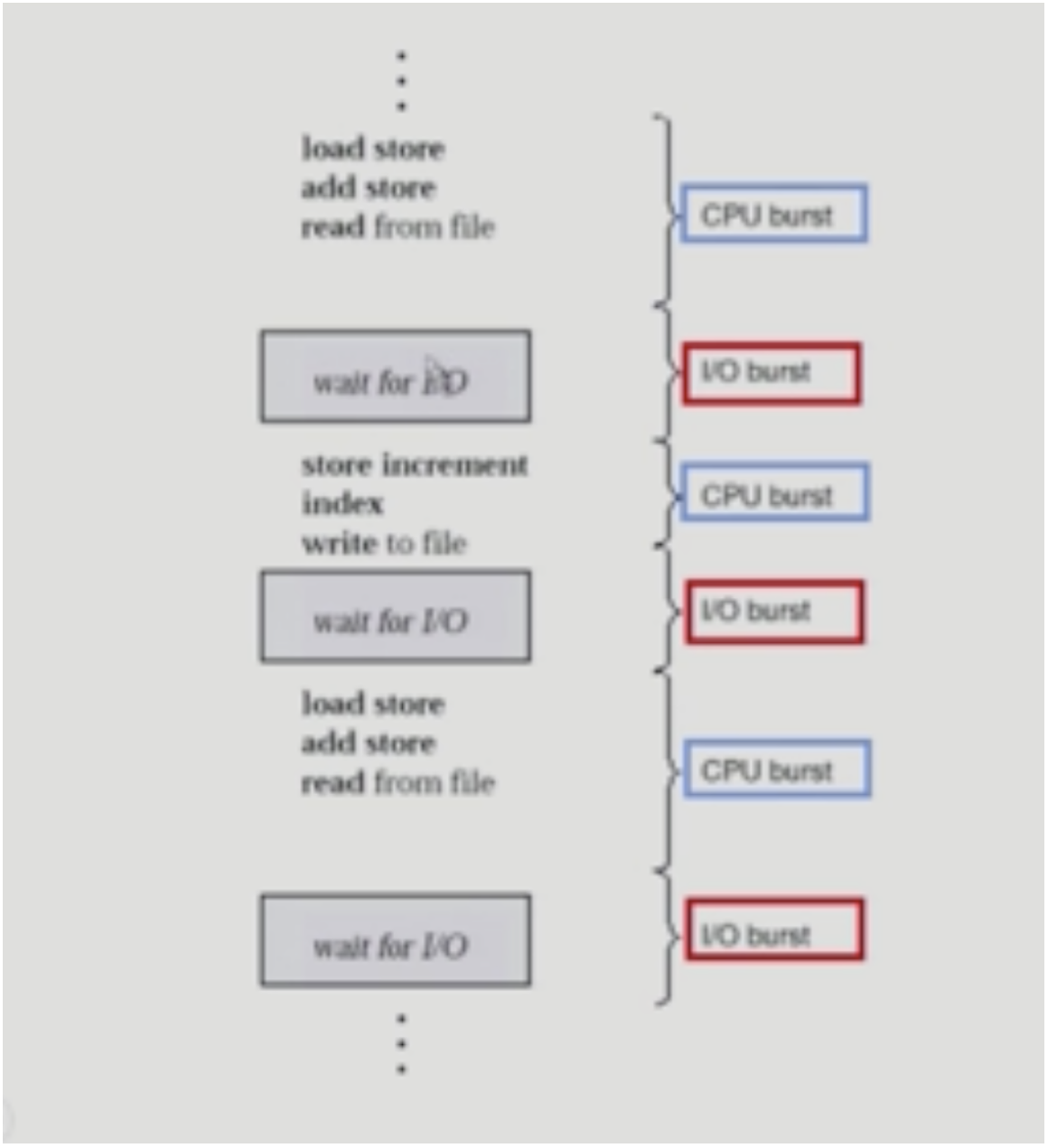
2주차_CPU스케쥴링

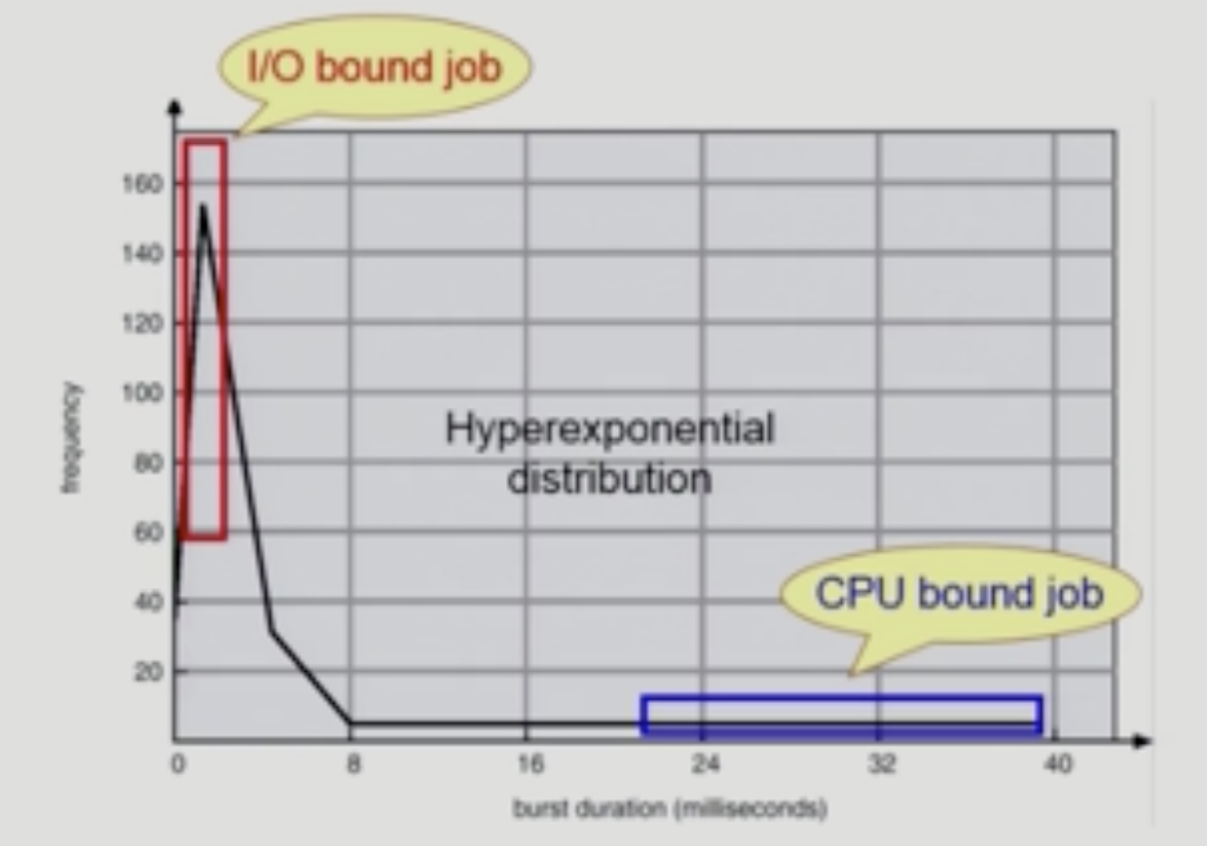
상태	진행 중
강의	CS 스터디
작성일	@2024년 2월 11일

CPU and I/O Bursts in Program Execution



- cpu와 IO를 번갈아가면서 사용

CPU-Burst Time의 분포



** 여러 종류의 job(=process)이 섞여 있기 때문에 CPU스케줄링이 필요하다

- Interactive job에게 적절한 response 제공 요망
- CPU와 I/O 장치 등 시스템 자원을 골고루 효율적으로 사용

- CPU bound job : CPU를 길게 쓰는 프로그램
- I/O bound job : CPU를 짧게 끊어 쓰는 프로그램. I/O 가 중간중간 많이 끼어있다는 뜻.

주로 사람하고 interaction을 많이 하는 프로그램들

** 둘중 누구애게 CPU 스케줄링을 빨리 하는게 좋은가? I/O bound job

프로세스의 특성 분류

- 프로세스는 그 특성에 따라 다음 두가지로 나눔
  - I/O-bound process

        - CPU를 답고 계산하는 시간보다 I/O에 많은 시가닝 필요한 job
        - (many short CPU bursts)
    - CPU-bound process
        - 계산 위주의 job
        - (few very long CPU bursts.)

CPU Scheduler & Dispatcher

- CPU Scheduler
    - Ready 상태의 프로세스 중에서 이번에 CPU를 줄 프로세스를 고릉다.
- Dispatcher
    - CPU의 제어권을 CPU scheduler에 의해 선책된 프로세스에게 남긴다.
    - 이 과정을 context switch(문맥 교환)라고 한다.
- CPU스케줄링이 필요한 경우는 프로세스에게 다음과 같은 상태 변화가 있는 경우이다.
    1. Running → Blocked(예 : I/O 요청하는 시스템 콜)
    2. Running → Ready(예: 할당시간 만료로 timer interrupt)
    3. BLocked → Ready(예: I/O완료 후 인터럽트)
    4. Terminate

** 1, 4에서의 스케줄링은 nonpreemptive(=강제로 빼앗지 않고자진 반납)

** All other scheduling is preemptivr(=강제로 뺴앗음)

Scheduling Criteria

performance Index(=Performance Measure, 성능 척도)

- CPU utilization(이용률) - 높을수록 좋지
    - keep the CPU as busy as possible
- Throughput(처리량)
    - # of process that complete their execution per time unit
- Turnaround time(소요시간, 반환시간) - CPU 사용한 시간과 기다린 시간의 합
    - amount of time to exectue a particular process

- Waiting time(대기 시간) - 단무지 먹고 자장면 적기 전까지 기다린 시간
  - amount of a time process has been waiting the ready queue
- Responce time(응답 시간)
  - amount of time it takes from when a request was submitted until the first response is produced. **not** output
    (for time-sharing enviornment)

Scheduling Algorithms

- FCFS (First-Come First-Served)
- SJF (Shortest-Job-First)
- SRTF (Shortest-Remaining-Time-First)
- Priority Scheduling
- RR (Round Robin)
- Multilevel Queue
- Multilevel Feedback Queue

FCFS (First-Come First-Served)

- Example:

Process	Burst Time
P_1	24
P_2	3
P_3	3

- 프로세스의 도착 순서 P1, P2, P3
- 스케줄 순서를 Gantt Chart로 나타내면 다음과 같다.

- Waiting time for P1 = 0, P2 = 24, P3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17

프로세스의 도착순서가 다음과 같다고 하자

P2, P3, P1

- the Gantt Chart for the schedule is:

- Waiting time for P1 = 6, P2 = 0, P3 = 3
- Average waiting time: (6 + 0 + 3)/3 = 3
- Much better than previous case.
- **Convoy effect(호위 효과):** short process behind long process (long process가 먼저 도착해서 쓰는 바람에 short process가 시작을 못하는 효과)

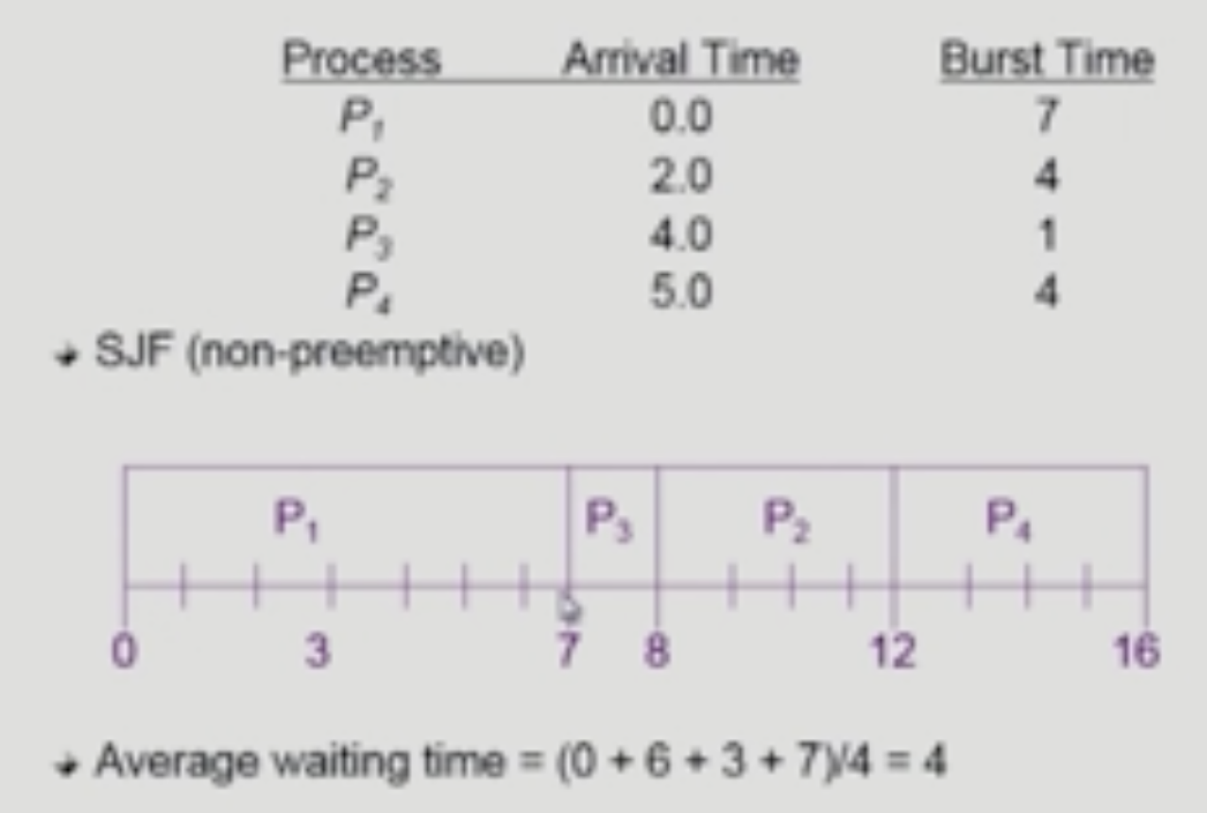
SJF (Shortest-Job-First)

- 각 프로세스의 다음번 CPU burst time을 가지고 스케줄링에 활용
- CPU burst time이 가장 짧은 프로세스를 제일 먼저 스케줄
- Two schemes:
  - **Nonpreemptive**
    - 일단 CPU를 잡으면 이번 CPU burst가 완료될 때까지 PCU를 선점(preemption)) 당하지 않음

- **Preemptive**
  - 현재 수행중인 프로세스의 남은 burst time보다 더 짧은 CPU burst time을 가지는 새로운 프로세스가 도착하면 PCU를 빼앗김
  - 이 방법을 Shortest-Remaining-Time-First(SRTF)이라고도 부른다.

- SJF is optimal
  - 주어진 프로세스들에 대해 minimum average waiting time을 보장
  - 그러면 왜 이것만 쓰지 다른 방법들도 존재?
    약점 1. SJF는 Starvation이 발생 할 수 있음
    약점 2. 다음 CPU Burst time을 미리 예측할 수 없음

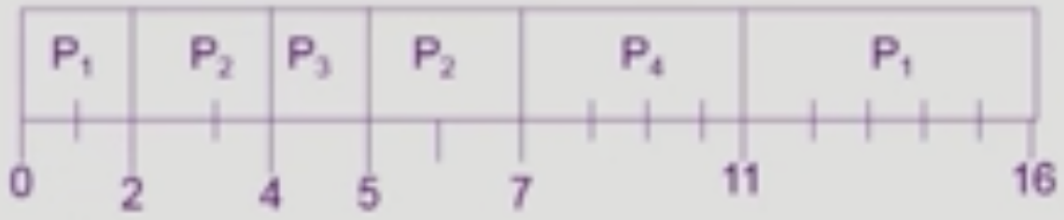
Example of Non-Preemptive SJF



Example of Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (preemptive)



- Average waiting time = (9 + 1 + 0 +2)/4 = 3

다음 CPU Burst Time의 예측

- 다음번 CPU burst time을 어떻게 알 수 있는가?

  (input data, branch, user ...)

- 추정(estimate)만이 가능하다.
- 과거의 CPU burst time을 이용해서 추정
  (exponential averaging)

1. t_n = actual lenght of n^{th} CPU burst
2. τ_{n+1} = predicted value for the next CPU burst
3. $\alpha, 0 \le \alpha \le 1$
4. Define : $\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n$.

$$\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n.$$

Exponential Averaging

- a = 0
  - r n-1 = t n
  - Recent history does not count
- a = 1
  - r n+1 = tn
  - Only the actual last CPU burst counts
- 식을 풀면 다음과 같다.

$$\tau_{n+1} = \alpha t_n + (1-\alpha)\alpha t_{n-1} + \dots$$
$$+(1-\alpha)^j \alpha t_{n-j} + \dots$$
$$+(1-\alpha)^{n+1} \tau_0$$

- a와 (1-a)가 둘다 1 이하이므로 후속 term은 선행 term보다 적은 가중치 값을 가진다.

Priority Scheduling

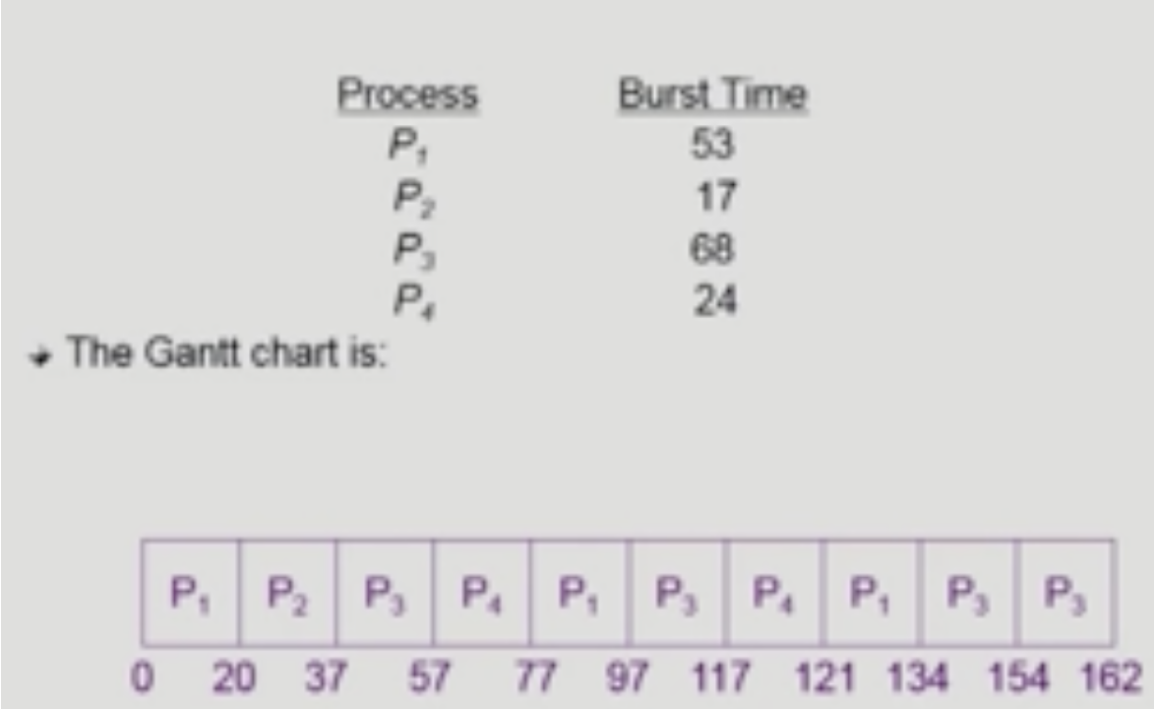
- A priority number (integer) is associated with each process
- highest priority를 가진 프로세스레게 CPU 할당
  (smallest integer = highest priority)
  - Preemptive
  - nonpreemptive
- SJF는 일종의 Priority scheduling이다.
  - priority = predected next CPU burst time
- Problem
  - **Starvation** : low priority process may never execute
- Solution
  - **Aging :** as time progresses increase the priority of the process.

Rond Robin(RR)

- 각 프로세스는 동일한 크기의 할당 시간(time quantum)을 가짐
  (일반적으로 10-100 milliseconds).
- 할당시간이 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** 단위고 CPU시간의 1/n을 얻는다.

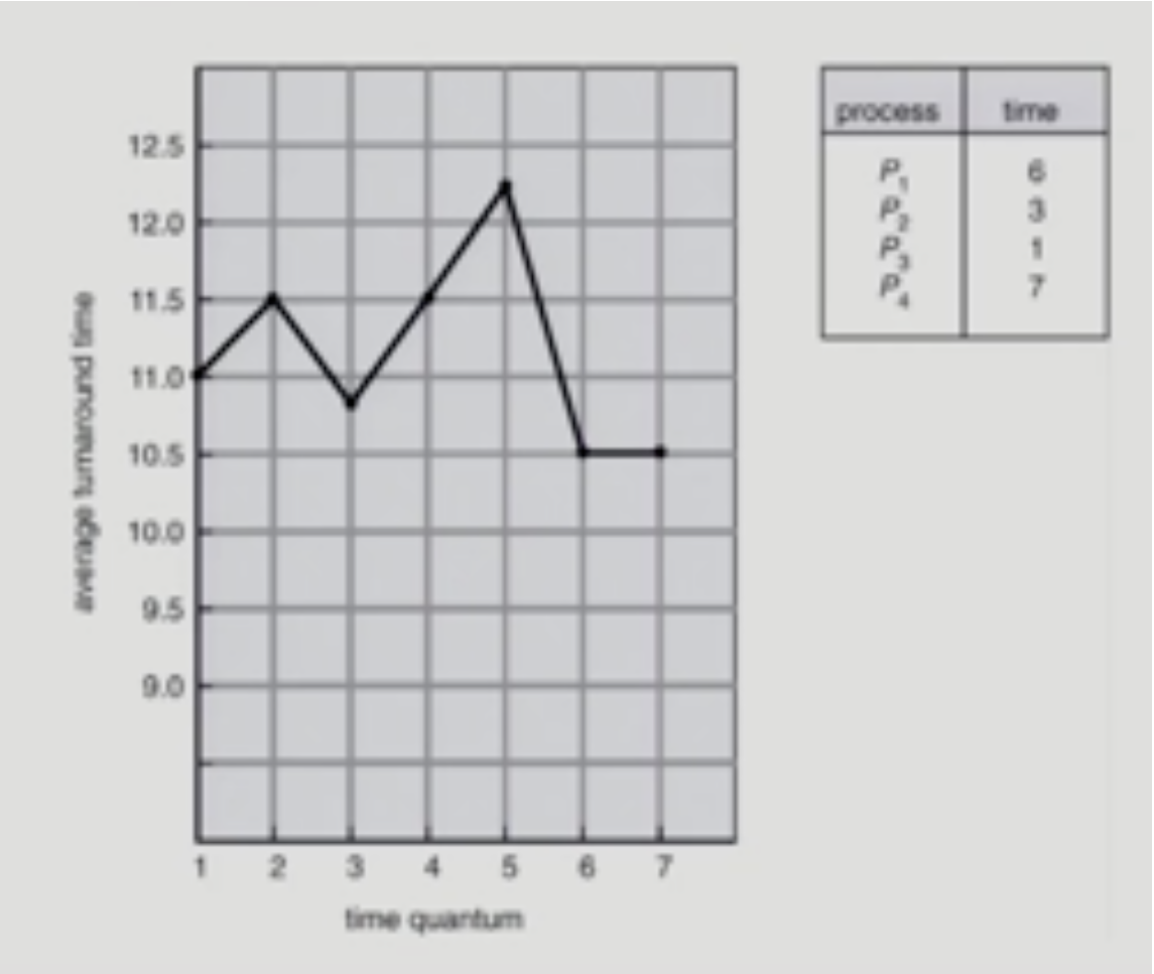
    ⇒ 어떤 프로세스도 (n-1) q tume unit 이상 기다리지 않는다.
- Performance
  - q large ⇒ FCFS
  - q small ⇒ context switch 오버헤드가 커진다.

Example: RR with Time Quantum = 20



- 일반적으로 SJF보다 average turnaround time이 길지만 response time은 더 짧다.
- homogeneous한 job들에서는 별로 좋지 않아.

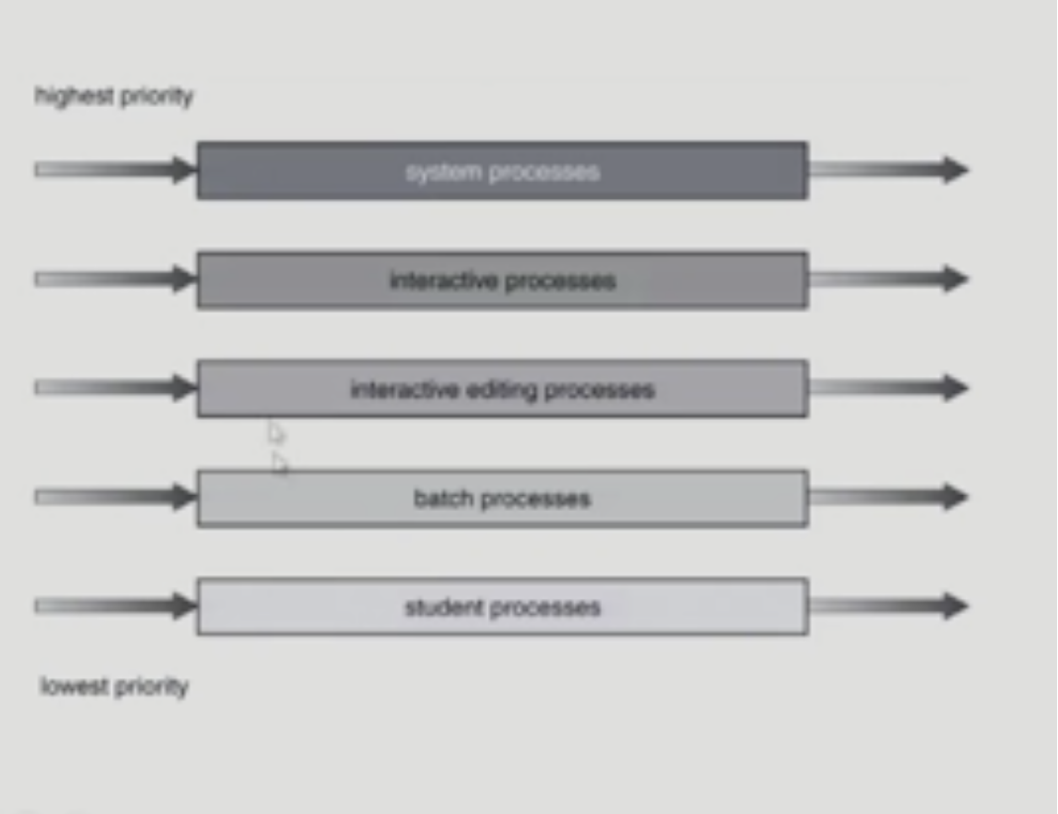
Turnaraound Time Varies with Time Quantum



Multilevel Queue

- Ready queue를 여러 개로 분할
  - foreground (interactice)
  - background(batch - no human interaction)
- 각 큐는 독립적인 스케줄링 알고리즘을 가짐
  - foreground - RR
  - background - FCFS
- 큐에 대한 스케줄링이 필요
  - Fixed priority scheduling
    - serve all from foreground the from background

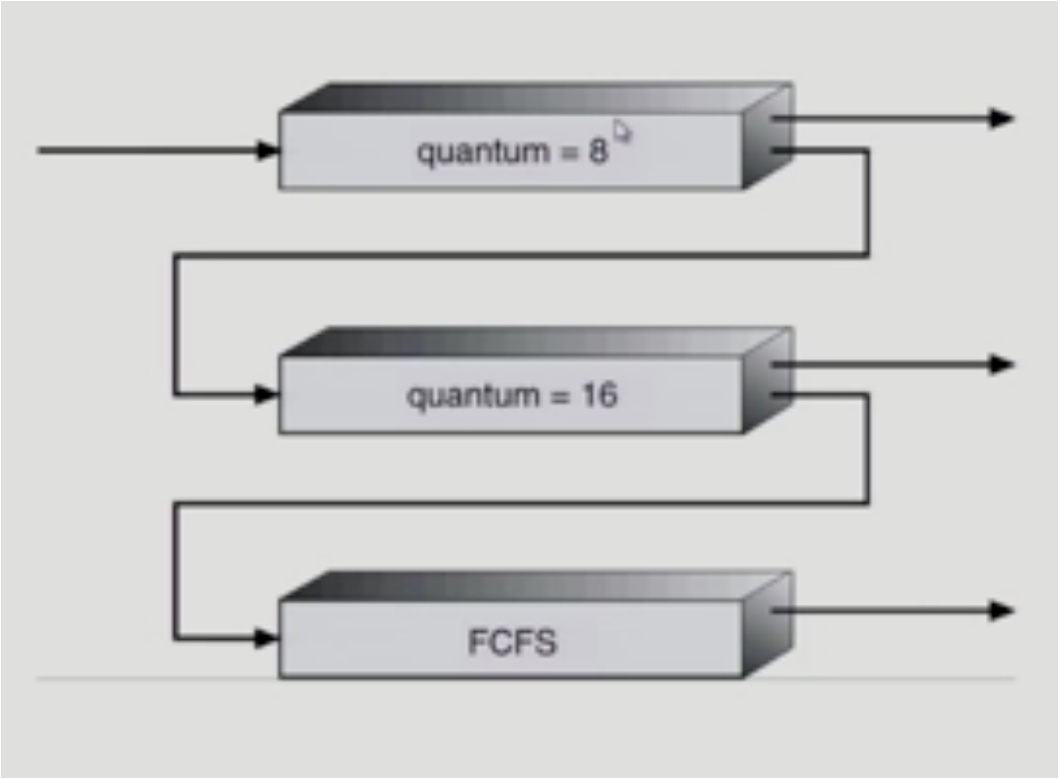
        - Possibility of starvation
    - Time slice
        - 각 큐에 CPU time을 적절한 비율로 할당
        - Eg. 80% to foreground in RR, 20% to background in FCFS



Multilevel Feedback Queue

- 프로세스가 다른 큐로 이동 가능
- 에이징(aging)을 이와 같은 방식으로 구현할 수 있다.
- Multilevel-feedback-queue scheduler를 정의하는 파라미터들
    - Queue의 수
    - 각 큐의 scheduling algorithm
    - Process를 상위 큐로 보내는 기준
    - Process를 하위 큐로 내쫓는 기준

- 프로세스가 CPU 서비스를 받으려 할 때 들어갈 큐를 결정하는 기준



Example of Multilevel Feedback Queue

- Three queues:
  - Q1 - time quantum 8 milliseconds
  - Q2- time quantum 16 milliseconds
  - Q3 - FCFS
- Scheduling
  - new job이 queue Q0로 들어감
  - CPU를 잡아서 할당 시간 8 milliseconds 동안 수행됨
  - 8 milliseconds동안 다 끝내지 못했으면 queue Q1으로 내려감
  - Q1에 줄서서 기다렸다가 CPU를 잡아서 16ms 동안 수행됨
  - 16ms에 끝내지 못한 경우 queue Q2로 쫓겨남

Multiple-Processor Scheduling

- CPU가 여러개인 경우 스케줄링은 더욱 복잡해짐
- Homogeneous processor인 경우
  - Queue에 한줄로 세워서 각 프로세서가 알아서 꺼내가게 할 . 수있다.
  - 반드시 특정 프로세서에서 수행되어야 하는 프로세스가 있는 경우에는 문제가 더 복잡해짐
- Load Sharing
  - 일부 프로세서에 job이 몰리지 않도록 부하를 적저절히 공유하는 메커니즘 필요
  - 별개의 큐를 두는 방법 vs. 공동 큐를 사용하는 방법
- Symmetric Multiprocessing(SMP)
  - 각 프로세서가 각자 알아서 스케줄링 결정
- Asymmetric multiprocessing
  - 하나의 프로세서가 시스템 데이터의 접근과 공유를 책임지고 나머지 프로세서는 거기에 따름

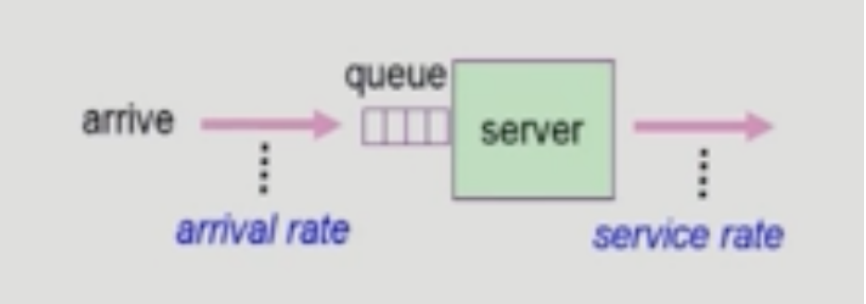
Real-Time Scheduling

- Hard real-time systems
  - Hard real-time task는 정해진 시간 안에 반드시 끝내도록 스케줄링해야 함
- Soft real-time computing
  - soft real-time task는 일반 프로세스에 비해 높은 priority를 갖도록 해야 함
  - ex) 동영상 재생

Thread Scheduling

- Local Scheduling
  - User level thread의 경우 사요앚 수준의 thread library에 의해 어떤 thread를 스케줄할지 결정
- Global Scheduling
- Kernel level thread의 경우 일반 프로세스와 마찬가지로 커널의 단기 스케줄러가 어껀 thread를 스케줄할지 결정

Algorithm Evaluation



- **Queueing models**
  - **확률 분포**로 주어지는 **arrival rate**와 **service rate**등을 통해 각종 **performance index** 값을 계산
- **Implementation(구현) & Measurement(성능 측정)**
  - **실제 시스템**에 알고리즘을 **구현**하여 실제 작업(**workload**)에 대해서 성능을 **측정** 비교
- **Simulation (모의 실험)**
  - 알고리즘을 **모의 프로그램**으로 작성 수 **trace**를 입력으로 하여 결과 비교