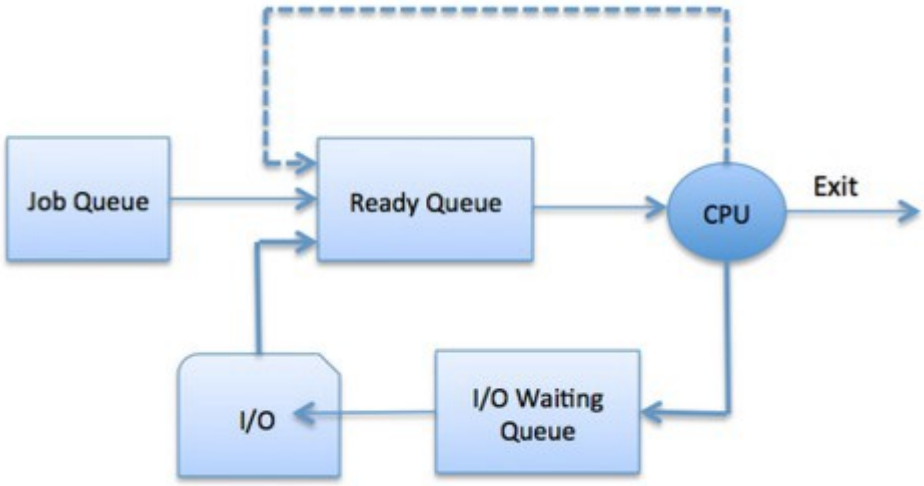
CPU Scheduling

CPU 스케줄링이란?

- 작업을 처리하기 위해서 프로세스들에게 CPU를 할당하기 위한 정책을 계획하는 것
- CPU를 잘 사용하기 위해 프로세스를 잘 배정하기
- CPU 스케줄링 대상 프로세스는, Ready Queue에 있는 프로세스들

  ⇒ 실행준비가 된 프로세스 중에서 하나를 선택해 CPU를 할당하는 것



> **좋은 스케줄링 조건**
>
> - 오버헤드 낮고
> - 사용률 높고(CPU를 최대한 바쁘게)
> - 기아 현상은 낮을때
>   - 기아현상? 특정 프로세스의 우선 순위가 낮아서 원하는 자원을 계속 할당받지 못하는 상태

> **스케줄링의 목표**
> **- CPU 스케줄링의 세부적인 목표는 시스템 마다 다름**

> - Batch System
>   배치 시스템은 한번에 하나의 프로그램만 수행하는 것
>   목표 : 가능하면 많은 일을 수행, 시간보다 처리량이 중요함
> - Interactive System
>   사용자가 컴퓨터 앞에서 대화형으로 동작하는 시스템
>   목표 : 빠른 응답 시간, 적은 대기 시간
> - Real-time System
>   시간 제약 조건이 걸려 있는 시스템
>   목표 : 기한 맞추기

스케줄링 방식

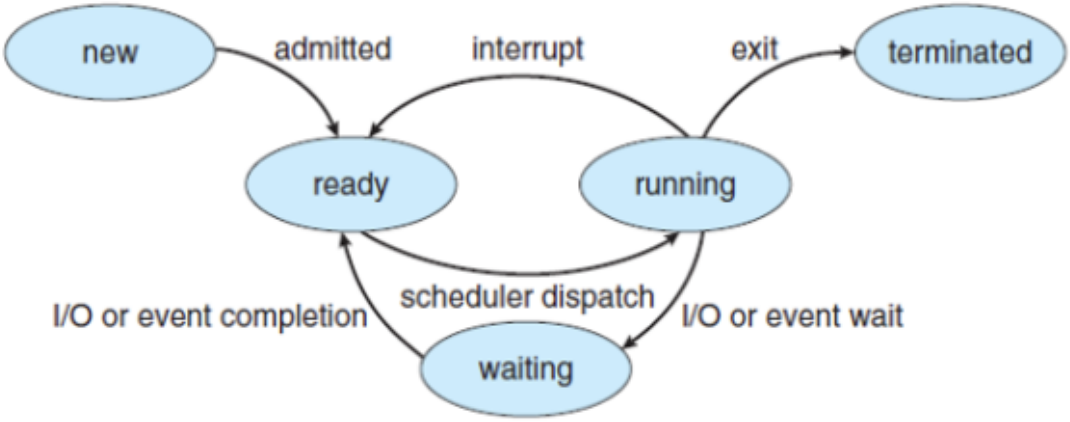
선점형 방식

- 어떤 프로세스가 CPU를 할당받아 실행 중에 있어도, 다른 프로세스가 실행 중인 프로세스를 중지하고 CPU를 강제로 점유할 수 있음
- 처리시간 예측 어려움

비선점형 방식

- 어떤 프로세스가 CPU를 할당받으면, 그 프로세스가 종료되거나 입출력 요구가 발생하여 자발적으로 중지될 떄까지 계속 실행되도록 보장함.
- 처리시간 예측 용이

프로세스 상태



> **New**
>
> - 프로세스가 만들어지는 과정의 상태

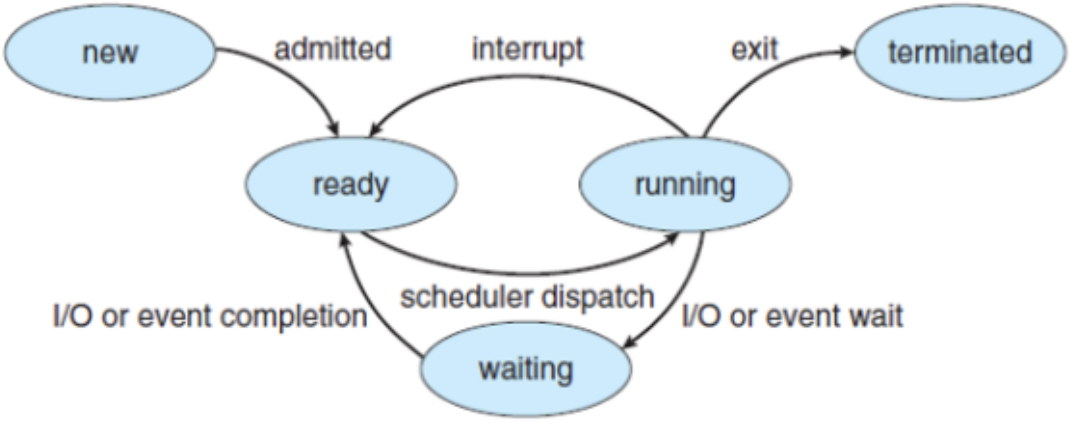
> **Ready**
>
> - Running할 준비가 되어 있는 상태(실행 준비가 다 된 상태에서 기다리는 것)
> - 프로세스가 CPU에 할당되기를 기다리는 상태

> **Running**
>
> - CPU에서 수행이 되고 있는 상태

> **Waiting**
>
> - **Block**이라고도 하며, 입출력이나 다른 이벤트가 발생하기를 기다리는 상태

> **Terminated**
>
> - 프로세스가 다 수행되어 종료될 때 잠시 생기는 상태

프로세스의 상태 전이



> **Admitted(승인)**
>
> - 프로세스가 만들어지는 과정의 상태

> **Scheduler Dispatch(스케줄러 디스패치)**
>
> - 준비 상태에 있는 프로세스 중 하나를 선택해 실행시키는 것

> **Interrupt(인터럽트)**
>
> - 예외, 입출력, 이벤트 등이 발생하여 현재 실행중인 프로세스를 준비 상태로 바꾸고, 해당 작업을먼저 처리하는 것

> **I/O or Event wait(입출력 또는 이벤트 대기)**
>
> - 실행 중인 프로세스가 입출력이나 이벤트를 처리해야하는 경우, 입출력/이벤트가 모두 끝날때까지 대기 상태로 만드는 것

> **I/O or Event Completion(입출력 또는 이벤트 완료)**
>
> - 입출력/이벤트가 끝난 프로세스를 준비 상태로 전환하여 스케줄러에 의해 선택될 수 있도록 만드는 것

헷갈려서 추가로 찾아본 것! **Ready vs Waiting?**

- I/O 작업이나 기타 event로 인한 상태의 변화
  ⇒ **Running → Waiting → Ready → Running** 순서로 상태가 변함
- 운영체제는 프로세스들의 자원 독점을 막기 위해 자원을 사용할 때 일정 시간을 단위로 할당하고 회수를 함. 시간이 초과되면, 다른 작업에게 자원을 할당하는데 이때 발생하는 Interrupt에 의해 변하는 상태
  ⇒ **Running → Ready → Running**

프로세스 상태와 선점/비선점 스케줄링

- **선점 스케줄링**
  - CPU 이용을 우선순위 높은 프로세스가 강제로 차지 가능

    `Interrupt` , `I/O or Event Completion`, `I/O or Event Wait`, `Exit` 시점에서 스케줄링이 일어날 수 있음
- **비선점 스케줄링**
  - 프로세스 종료 또는 I/O등의 이벤트가 있을 때까지 실행 보장

    `I/O or Event Wait`, `Exit` 에만 스케줄링

CPU 스케줄링의 종류

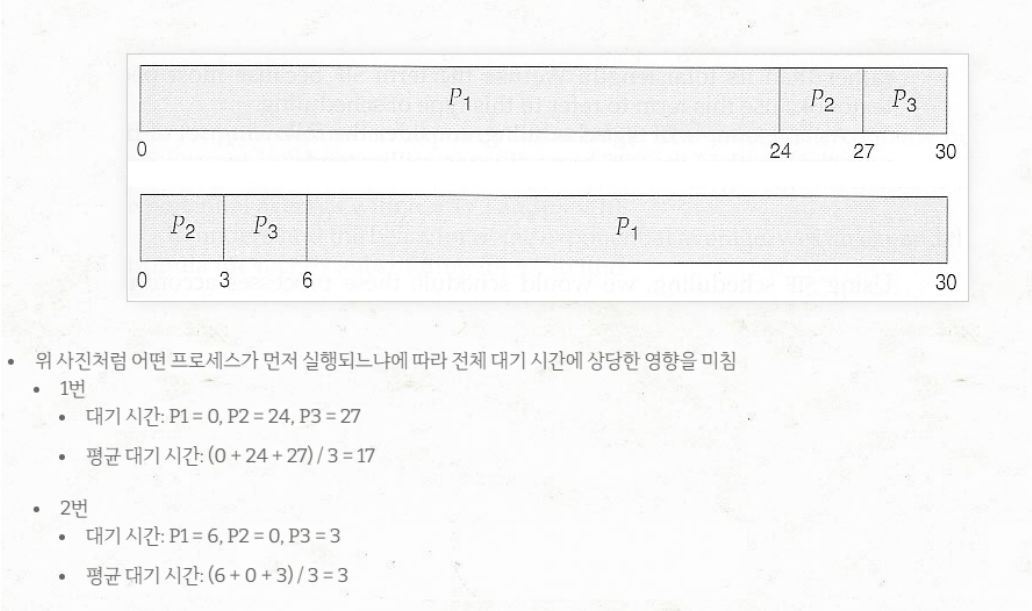
간단 요약

- First-Come, First-Served (FCFS) : 먼저 온걸 먼저 서비스함
- Shortest-Job-First (SJF) : 작업시간이 짧은걸 먼저 처리
- Highest Response-ratio Next(HRN) : 짧으면서도 오래 대기한 것을 우선순위
- Priority : 우선순위가 높은걸 먼저처리함.
- Round-Robin (RR) : 짧은 시간동안 계쏙 뱅뱅돌면서 하는 것
- Multilevel Queue : 우선순위에 따라 큐를 분리한것
- Multilevel Feedback Queue : 큐를 옮겨탈 수 있는 것

비선점 스케줄링

- **FCFS(First Come First Served)**
  - 큐에 도착한 순서대로 CPU를 할당
  - 장점 : 쉽고 공평함
  - 단점 : 실행 시간이 짧은게 뒤로 가면, 평균 대기 시간이 길어짐

    ▼ 예시



- 위 사진처럼 어떤 프로세스가 먼저 실행되느냐에 따라 전체 대기 시간에 상당한 영향을 미침
  - 1번
    - 대기 시간: P1 = 0, P2 = 24, P3 = 27
    - 평균 대기 시간: (0 + 24 + 27) / 3 = 17
  - 2번
    - 대기 시간: P1 = 6, P2 = 0, P3 = 3
    - 평균 대기 시간: (6 + 0 + 3) / 3 = 3

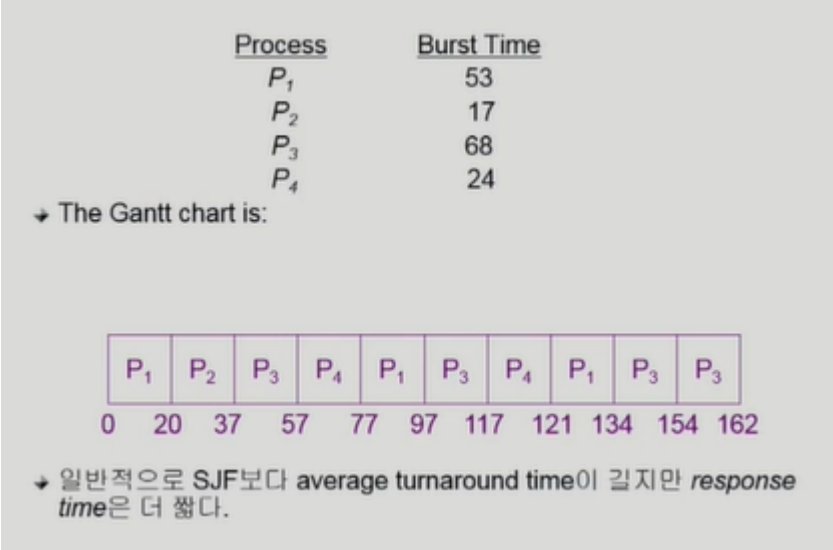
- **SJF(Shortest Job First)**
  - FCFS를 보완하기 위해 만들어짐.
  - 수행시간이 가장 짧다고 판단되는 작업을 먼저 수행
  - 장점 : FCFS보다 평균 대기 시간 감소, 짧은 작업에 유리함
  - 단점: 사용 시간이 긴 프로세스는 영원히 CPU를 할당 받지 못할 수 있음 (Starvation)
- **HRN(Highest Response-ratio Next)**
  - SJF 알고리즘을 보완하기 위해 만들어짐
  - 우선순위를 계산하여, 점유 불평등을 보완한 방법
  - 우선순위 = (대기시간 + 실행시간) / 실행시간

선점 스케줄링

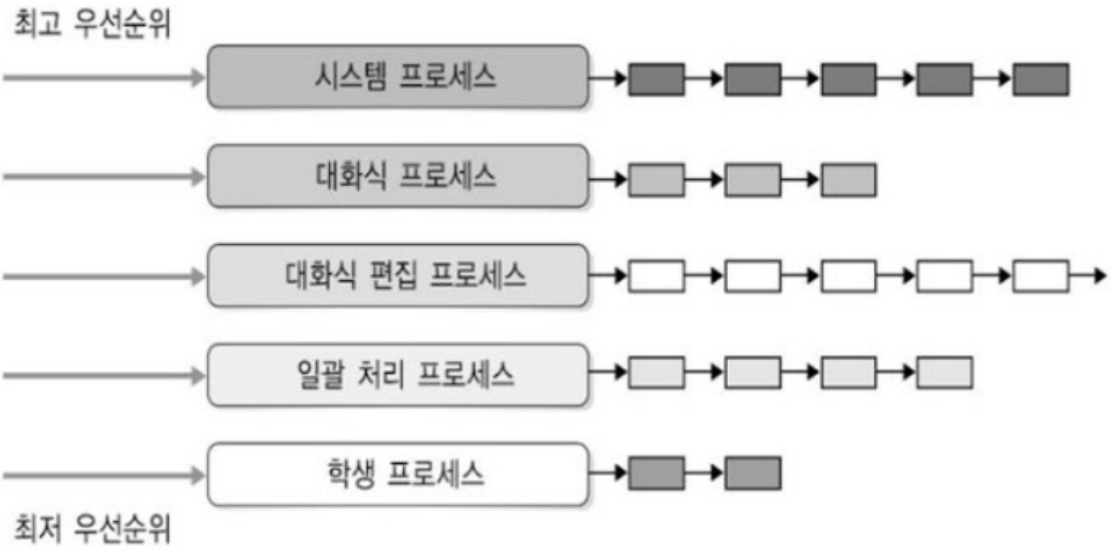
- **Priority Scheduling**
  - 정적/동적으로 우선 순위를 부여하여 우선 순위가 높은 순서대로 처리함
  - 우선 순위가 낮은 프로세스가 무한정 기다리는 Starvation, Indefinite Blocking이 생길 수 있음
  - Aging 기법으로 문제 해결 가능
    - **Aging 기법이란?** 자원이 할당되기를 기다린 시간에 비례하여, 높은 우선순위를 부여하는 것(아무리 우선 순위가 낮은 프로세스라 하더라도 시간이오래 지나면 우선순위를 높여주는 기법)
- **Round Robin**
  - FCFS에 의해 프로세스를 받아, 각 프로세스에 동일한 시간의 **Time Quantum(실행의 최소 단위 시간)** 만큼 CPU를 할당함
  - 할당 시간이 지나면, 프로세스는 CPU를 빼앗기고, Ready Queue 맨 뒤에 가서 줄을 서게 됨
  - 짧은 응담 시간 보장
  - 할당 시간이 크면 FCFS와 같게 되고,
  - 할당 시간이 작으면 문맥 교환이 잦아져서 오버헤드가 증가함

▼ 예시

- 일정 할당 시간(20)을 공평하게 할당 받아 CPU 사용



- **Multi level-Queue(다단계 큐)**
  - 작업들을 여러 종류의 그룹으로 나누어 여러 개의 Queue를 사용함
  - Ready Queue를 우선 순위에 따라 여러 개로 분할
    - Single queue에서는 모든 프로세서의 우선순위를 찾아야 하기에, 시간복잡도가 O(n)
    - Multi level Queue를 도입하면 이 시간 복잡도를 1로 줄일 수 있음



- 우선 순위가 낮은 큐들이 실행 못하는 것을 방지하고자, 큐마다 다른 Time Qunatum을 설정함
- 우선 순위가 높은 큐에는 작은 Time Quantum을, 낮은 큐에는 큰 Time Quantum을 할당함

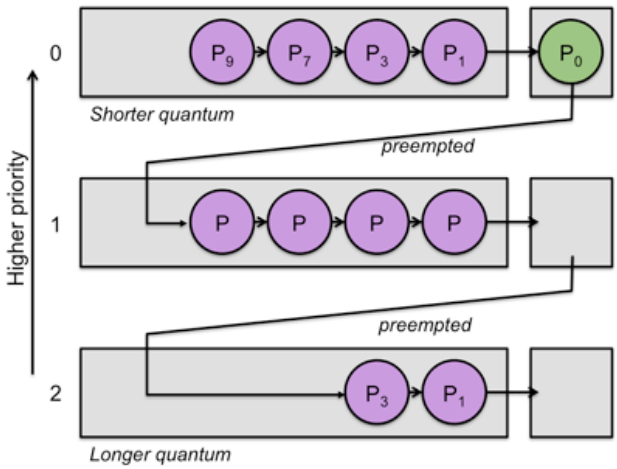
  ⇒ 이해가 잘 안됨 ㅜ 우선 순위가 높은데 왜 시간 할당을 적게 하는지..?

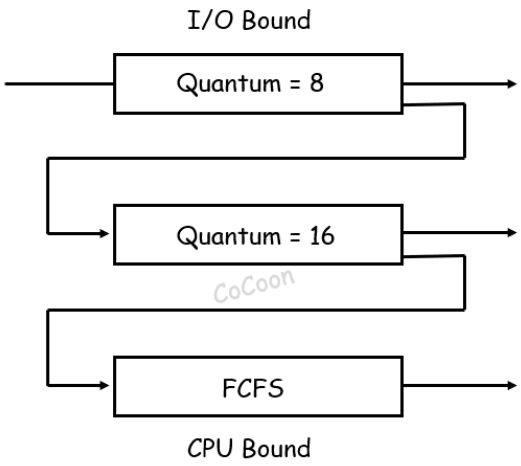
- Multi level Queue를 도입하면 무엇이 좋은가?
  - 프로세스별로 특징이 다름
    - 응답이 빨리와야 하는 프로세서가 있고, 계속 돌아가야 하는 프로그램도 있고,, 비교적 처리를 빨리해도 되는 프로세서가 있음
    - ex) 카톡 : 켜놓기만하고 메시지 올때만 확인하면 됨. But, 게임 : 상태를 계속 업데이트해야함

  ⇒ 이런 경우에 특징에 따라 다른 알고리즘사용이 가능함. 높은 우선순위를 줘야하는 프로세스와 낮은 우선순위를 갖는 프로세스를 따로 관리가 가능함.

  ex) 높은 우선순위를 가지는 애들은 RR 알고리즘을 사용하고, 낮은 우선순위를 가지는 애들은 FCFS알고리즘을 사용하는 등

- **Multi level-Feedback-Queue(다단계 피드백 큐)**





- 프로세스가 여러 개로 분할된 Ready Queue 내에서 **다른 큐로 이동이 가능**
- 프로세스 생성 시 가장 높은 우선 순위 준비 큐에 등록되며 등록된 프로세스는 FCFS 순서로 CPU를 할당받아 실행된다. **해당 큐의 CPU 시간 할당량(Time Quantum)이 끝나면 한 단계 아래의 준비 큐에 들어**간다.

  (모든 프로세스는 가장 위의 큐에서 CPU 점유를 대기하고 이상태로 진행하다가 시간이 너무 오래 걸린다면 아래의 큐로 프로세스를 옮겨 대기시간을 조정하는 방식)
- 단계가 내려갈수록 시간 할당량(Time Quantum)이 증가
- 큐 사이의 프로세스 이동 가능
- 짧은 작업에 유리, **입출력 위주(Interrupt가 잦은) 작업에 우선권**을 줌

응용 프로그램은 대부분 계산 위주 단계와 입/출력 위주 단계를 반복하므로 프로세스 진행 과정에서 그 특성이 변하면 이를 인지하여 해당 프로세스를 적절한 준비 큐로 이동시켜주는 전략이 MFQ 스케줄링이다.

- 실행 중인 프로세스가 해당 큐의 타임 퀀텀을 소진하지 못하고 입출력 등으로 CPU를 자진 반납하면, 이 프로세스는 입/출력 성향이 강해진 것으로 인식하여 입/출력 쪽으로 한 단계 높은 준비 큐로 이동시킨다.
- 반대로, 실행 중인 프로세스가 주어진 타임 퀀텀을 모두 소진한 후 CPU를 강제로 회수당하면, 이 프로세스는 계산 성향이 강해진 것으로 인식하여 이 프로세스를 입/출력 성향 기준 한 단계 낮은 준비 큐로 이동시킨다.

- CPU Burst는 낮은 우선순위의 큐, I/O Burst는 높은 우선순위의 큐에 배치
- Time Quantum을 다 채운 프로세스는 CPU burst 프로세스로 판단하기 때문
- 처리 시간이 짧은 프로세스를 먼저 처리하기 때문에 Turnaround 평균 시간을 줄여줌

- 가장 하위 큐는 FCFS 스케줄링
- 맨 아래 큐에서 너무 오래 대기하면 다시 상위 큐로 이동 (에이징 기법을 통한 기아상태 예방)

💡 **Burst란..?**

- 특정 기준에 따라 한 단위로서 취급되는 연속된 신호나 데이터의 모임
- CPU 버스트 : CPU가 할당되어 실행 중인 주기
- I/O 버스트 : 입/출력이 이루어지는 주기
- 입/출력이 완료될 때까지 그 프로세스는 CPU 할당을 받지 못하는 I/O 버스트고, 그 외 부분은 CPU 버스트

Scheduling Criteria(스케줄링 척도)

> 최선의 알고리즘을 결정하는 데 사용되는 기준

- **CPU Utilization (CPU 이용률)**
  - 가능한한 CPU를 최다한 바쁘게 유지하기를 원함
- **Throughput (처리량)**
  - 단위시간당 완료된 프로세스의 개수
- **Turnaround time (총처리 시간)**
  - 한 프로세스가 New 상태에서 Terminated 상태에 도달할 때까지의 시간
- **Waiting time (대기 시간)**
  - Ready 큐에서 대기하면서 보낸 시간의 합
- **Response Time (응답 시간)**
  - 작업이 처음 실행되기까지 걸린 시간

⇒ CPU 이용률, 처리량이 높을 수록, 대기시간, 응답시간 낮을 수록 바람직함

나올만한 질문

- **운영체제에서 '기아현상(Starvation)'과 '노화기법(Aging)'이 무엇인지 각각 설명해주세요.**

> ### 운영체제에서 기아현상(Starvation)이란 무엇입니까?
>
> - 기아현상은 자원관리 문제입니다.
> - 특정 프로세스의 우선순위가 낮아서 원하는 자원을 계속 할당받지 못하는 현상을 말합니다.
> - 대기중인 프로세스는 리소스가 다른 프로세스에 할당되어 있기 때문에 필요한 리소스를 얻기 위해 무한정 기다리게 됩니다.
>
> ### 운영체제에서 노화기법(Aging)는 무엇입니까?
>
> - 노화기법은 기아현상을 방지하는 기법입니다.
> - 특정 프로세스의 우선순위가 낮아 무한정 기다리게 된다면, 한번 양보하거나 기다린 시간에 비례하여 우선순위를 높여 자원을 할당받을 수 있도록 해주는 기법입니다.

- **CPU 스케줄링은 언제 발생하는가?**
- **CPU 스케줄링의 종류를 설명하시오.**
- **선점 스케줄링과 비선점 스케줄링의 차이점?**