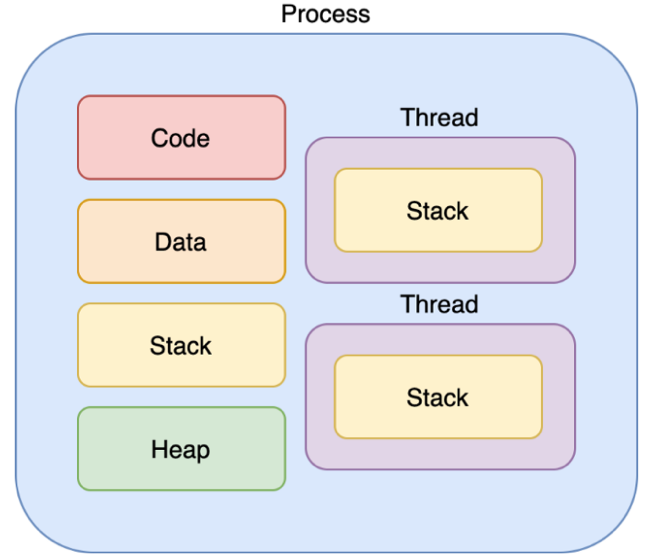
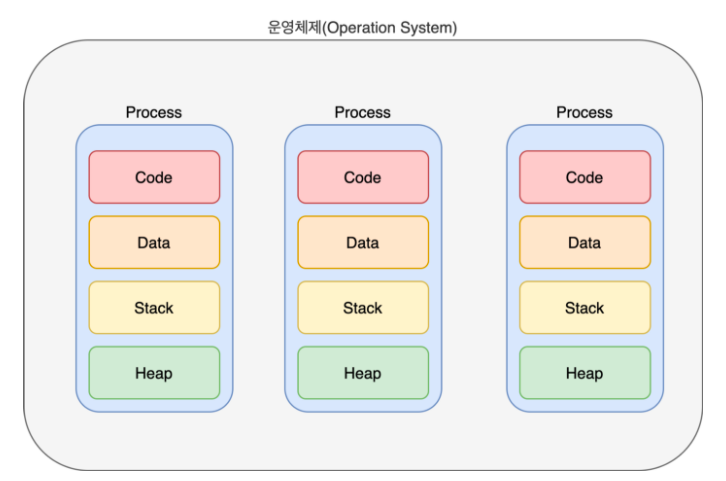
1. **프로세스란?**
   1. **프로세스와 스레드**

* **요약 : 프로세스는** 운영체제로부터 자원을 할당 받는 작업의 단위이고 **스레드는** 프로세스 내의 자원을 이용하는 실행의 단위.
  + 1. 프로세스 :
* 운영체제로부터 메모리에 적재되고 CPU 자원을 할당 받아 프로그램이 실행되고 있는 상태
* 멀티 프로세스 : 하나의 애플리케이션을 여러 개의 프로세스로 구성, 안정적(=하나의 자식 프로세스가 오류나도 다른 자식 프로세스에는 영향이 없기 때문), but 스케쥴링에 따른 context switch가 많다(=성능 저하의 우려).
  + 1. 스레드 :
* 프로세스 내의 메모리를 공유해 실행되는 흐름의 단위
* 운영체제의 일부인 스케줄러에 의해 독립적으로 관리 될 수 있는 프로그래밍 된 명령어의 가장 작은 시퀀스
* 프로세스를 생성하면 기본적으로 하나의 메인 스레드가 생성됨
* 멀티 스레드 : 하나의 애플리케이션을 여러 스레드로 구성, 단일 스레드로 Network 또는 DB 와 같은 긴 작업(Long-running task) 을 수행하는 경우 해당 작업을 처리하는 동안 사용자와 상호작용이 불능인 상태가 되는 것을 방지. 적은 비용 but 구현 어려움, 너무 많은 스레드는 오버헤드 발생됨.



* 1. **스케줄링**

**- 요약 :** 프로세스가 생성 후 실행될 때 필요한 시스템의 여러 자원을 할당하는 작업

* + 1. 비선점/선점 스케줄링
* **비선점 스케줄링**

1. 이미 할당된 CPU를 다른 프로세스가 강제로 빼앗아 사용할 수 없는 스케줄링 기법
2. 프로세스 응답 시간의 예측이 용이하며, 일괄 처리 방식에 적합
3. 중요한 작업(짧은 작업)이 중요하지 않은 작업(긴 작업)을 기다리는 경우가 발생됨
4. **FIFO(=FCFS), SJF, 우선순위, HRN(Highest Response-ratio Next)**, 기한부 등의 알고리즘이 있음

* **선점 스케줄링**

1. 하나의 프로세스가 CPU를 할당받아 실행하고 있을 떄 우선순위가 높은 다른 프로세스가 CPU를 강제로 빼앗아 사용할 수 있는 스케줄링 기법
2. 주로 빠른 응답시간을 요구하는 대화식 시분할 시스템에 사용됩니다.
3. 많은 오버헤드를 초래합니다.
4. **Round-Robin(RR), SRT(Shortest Remaining Time First)**, 선점 우선순위, 다단계 큐, 다단계 피드백 큐 등의 알고리즘이 있습니다.
   1. **동기화**

**- 요약 :** 동기화란 작업들 사이에 실행시기를 맞추는 것. 여러 스레드가 동일한 자원에 접근시 동기화 이슈 발생. 동일 자원을 여러 스레드가 동시수정하면 각 스레드 결과에 영향을 미친다.

* + 1. 동기화 이슈 해결 방안
* **상호 배제(Mutual exclusion)** 
  + 1. 임계영역
* 멀티스레드 프로그램에서 공유 자원이 참조 가능한 코드 영역을 말한다.
* 한 스레드가 사용 중인 자원은 작업이 끝날 때까지 lock을 걸어 다른 스레드가 사용할 수 없도록 한다.
  + 1. 뮤텍스(binary semaphore)
* 임계영역은 한 프로세스 내의 스레드에만 사용 가능하지만 **뮤텍스는 서로 다른 프로세스에 속한 스레드에도 적용 가능**.
  + 1. 세마포어
* 임계구역에 여러 스레드가 들어갈 수 있음
* 다중 스레드 간 공유 자원 접근 순서를 제어하고 공유 자원의 수와 접근 스레드 수에 따라서 보다 유연하게 공유영역으로의 접근을 제어한다.
  1. **데드락**

**- 요약 :** 무한 대기상태로 두 개 이상의 작업이 서로 상대방의 작업이 끝나기만을 기다리고 있어 다음 단계로 진행하지 못하는 상태.

* + 1. 교착상태(Deadlock)
* 교착 상태는 여러 프로세스가 동일 자원 점유를 요청할 때 발생
* 두 개 이상의 작업이 하나씩 자원을 hold(소지)하고 상대방이 가진 자원을 서로 원하고(need)있는 상태
* **교착상태 발생 조건** :
  + 상호 배체
  + 점유 대기
  + 비선점(자원을 강제로 뺏을 수 없음)
  + 환형(순환/Circular wait) 대기

4가지를 모두 만족하면 교착상태 발생

* **교착상태 예방** : 교착상태 발생 조건 중 하나를 제거함으로써 예방
* **교착상태의 회피** : 은행원 알고리즘(필요한 자원의 최대 개수를 미리 신고해 자원 요청을 수락하여도 안전상태에 머무르는지 판단 후 요청 수락), 자원 할당 그래프 알고리즘(교착상태 회피를 위해 예약간선을 추가/시스템에서 자원이 반드시 미리 예약되어 그래프에 표시되어야 함)
* **교착상태의 회복** : 교착 상태가 일어난 프로세스를 종료, 프로세스에 할당된 자원을 선점(우선 순위가 낮은 프로세스 위주로 자원을 선점할당한다)
  + 1. 기아상태(Starvation)
* 부족한 자원을 점유하기 위해 경쟁할 때 발생, 특정 프로세스는 영원히 자원할당이 안되는 경우.
* **기아상태 해결 방안** :
  + 프로세스 우선순위 수시 변경
  + 오래 기다린 프로세스의 우선순위 높이기
  + 우선순위가 아닌 요청 순서대로 처리하는 요청큐

1. **운영체제**
   1. **메모리 관리 전략**

* 메모리 관리가 왜 필요한가?

각 프로세스는 독립된 메모리 공간을 갖고, OS나 다른 프로세스의 메모리 공간에 접근할 수 없기 때문

(단, 운영체제는 운영체제 메모리 영역과 사용자 메모리 영역에 접근에 제약 받지 않는다.)

* + 1. **Swapping** :
* 라운드 로빈과 같은 스케줄링의 다중 프로그래밍 환경에서 **CPU 할당 시간이 끝난 프로세스의 메모리를 보조기억장치로 내보내고** 다른 프로세스의 메모리를 불러들일 수 있음.
* swap-in : 주기억장치(RAM)으로 불러오는 과정
* swap-out : 보조기억장치로 내보내는 과정
* 큰 디스크 전송시간이 요구되어 메모리 공간이 부족할 때만 swapping이 시작됨
  + 1. **단편화** (Fragmentation) :
* 프로세스들이 메모리에 적재/제거되는 일이 반복되다가 프로세스들의 차지한 메모리 틈 사이에 사용하지 못할 정도의 작은 빈 공간이 늘어나게 되는 것
* **외부 단편화** : 메모리 공간 중 사용하지 못하게 되는 일부분. 메모리 사이사이 남는 공간들을 모두 합치면 충분한 공간이 될 때 발생
* **내부 단편화** : 프로세스가 사용하는 메모리 공간에 남은 부분. 프로세스가 요구한 메모리보다 약간 커서 빈 곳이 발생.
* **압축** : 외부 단편화를 해소하기 위해 프로세스가 사용하는 공간들을 한 쪽으로 몰아 공간을 확보. (작업 효울은 bad)

텍스트, 시계이(가) 표시된 사진

자동 생성된 설명

(외부 단편화와 압축 과정)

* + 1. **페이징** (Paging) :
* 하나의 프로세스가 사용하는 메모리 공간이 연속적이어야 한다는 제약을 깨는 관리 방법 = 외부 단편화와 압축 작업 해소 가능
* 물리 메모리는 Frame이라는 고정 크기 블록으로 분리되고 논리(=가상) 메모리(프로세스가 점유)는 page로 불리는 고정 크기 블록으로 분리
* 페이징을 통해 page는 Frame에 저장될 때 연속되어 저장될 필요 없이 남는 프레임에 적절하게 배치됨 = 외부 단편화 해결
* 하나의 프로세스가 사용하는 공간은 여러 페이지로 나뉘어 관리되고 개별 페이지는 순서에 상관없이 프레임에 mapping되어 저장된다.
* **단점** : 내부 단편화 발생 가능성 증가 = page에 여유공간이 남는 경우가 많아짐
  + 1. **세그멘테이션** (Segmentation) – 가변 크기 :
* 페이징처럼 논리 메모리와 물리 메모리를 고정 크기 블록이 아닌, 서로 다른 크기의 **논리적 단위**인 세그먼트로 분할.
* 가변적인 데이터이며, 사용자 관점의 가상 메모리 관리 기법(malloc, calloc…)
* 장점 : 내부 단편화 문제를 해소하기 위한 방법, 메모리 사용 효율 개선 및 동적 분할을 통한 오버헤드가 감소
* **단점** : 외부 단편화 발생 가능성이 있음. (분할된 크기보다 프로그램의 크기가 더 커질 경우)
* **이 단점 해결을 위해 세그먼트를 다시 페이징하는 방법이 있음**