1. 프로세스 동기화(Process Synchronization)

1) 임계구역 문제(Critical Section Problem)

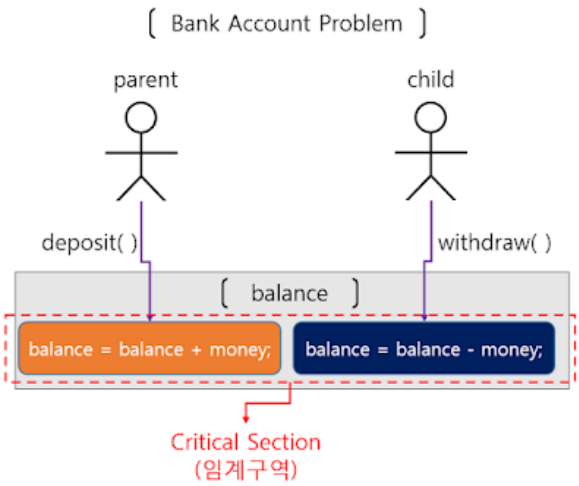
* 독립 프로세스(independent process) & 협력 프로세스(cooperating process)
  : 다른 프로세스와 아무런 연관을 갖지 않는다면 독립 프로세스.
  : 다른 프로세스에 영향을 미치거나 영향을 받게 되면 협력 프로세스.
  (이메일 주고 받기, 데이타 베이스, 온라인 게임, 주식거래 등)

* 프로세스 동기화(Process Synchronization)
  : 상호 협력하는 다수의 프로세스들이 공유 자원(혹은 데아타)을 사용할 경우, 경쟁 조건(Race Condition)으로 인해 그 공유 자원을 신뢰할 수 없게 된다. 따라서 이를 방지하기 위해 프로세스들이 공유 자원을 사용할 때 일련의 규칙을 지키도록 하는 것을 **프로레스 동기화**라고 한다.

* 경쟁 조건(Race Condition)
  : 여러 프로세스가 공유 데이터를 동시에(병행적으로) 접근(읽기나 쓰기)할 때 공유 데이터에 대한 **접근 순서에 따라 실행 결과가 달라지는** 상황.

* 임계 구역(Critical Section)
  : 둘 이상의 프로세스(혹은 쓰레드)가 공유하는 자원(데이타 구조, 장치)이지만 동시에 둘 이상의 프로세스가 접근할 수는 없고 특정 시간에 오직 하나의 프로세스만 접근 가능한 코드 영역.

* 은행 계좌 문제



: 부모(parent)는 은행 계좌(bank account)에 입금한다.
: 자녀(child)는 은행 계좌(bank account)에서 출금한다.
: 은행 계좌(bank account)는 입금과 출금 동기화가 올바로 이루어 져야 은행 시스템이 신뢰받을 수 있다.
: 입금 프로세스가 처리되고 있는 중간에 문맥 교환(Context Switching)이 발생하여 출금 프로세스가 처리되면 안된다.

: 임계 구역 = 은행 계좌 => 변수 balance

***문맥 교환(Context Switching)**

스케줄링에 의해 현재 프로세스를 대기 상태로 만들고, 프로세스 상태나 자원 등을 저장한 뒤 새 프로세스를 실행하는 것

하나의 프로세스가 CPU 를 사용 중인 상태에서 다른 프로세스가 CPU 를 사용하도록 하기 위해, 실행중인 프로세스의 상태(문맥)를 보관하고 새로운 프로세스의 상태를 적재하는 작업.

한 프로세스의 문맥은 그 프로세스의 **프로세스 제어 블록(PCB)**에 저장되어 있다.

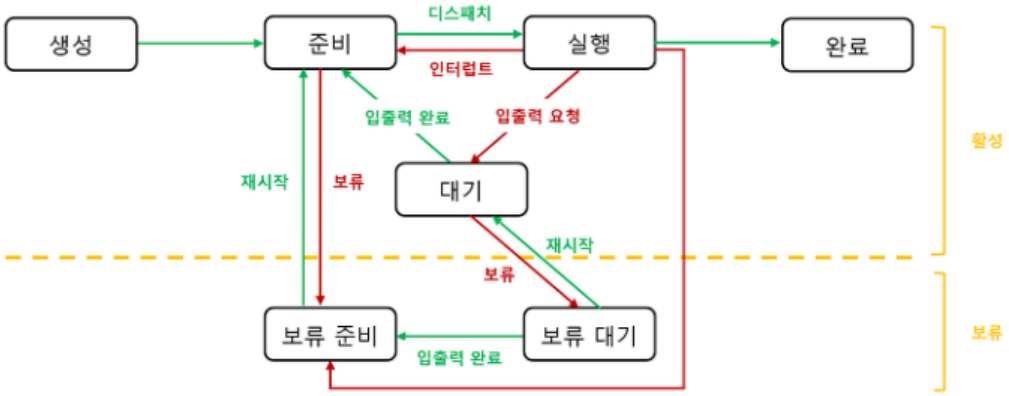
***Context Switch 는 언제 일어나는가?**

Termination of a process (프로세스의 종료)
☞ 에러,예외가 나타나거나 보통의 프로세스의 종료
Expiration of time slice (time slice 만료)
☞ 가능한 CPU 점유 시간이 만료되었을 경우
Blocking system calls (also called supervisor call)
☞ I/O 요청, 파일 오픈과 같은 시스템 콜의 경우
☞ Page fault - 메모리 주소가 가상 메모리에 있으므로 main memory 로 가져와야 한다
I/O 인터럽트
☞ I/O 가 완료되면 Block 상태의 프로세스를 Ready 상태로 바꾼다.

* 임계 구역 문제의 해결방안
: 상호배제(Mutual exclusion)
- 공통 변수에 대해 상호 배타적이어야 한다.
- 오직 한 프로세스(혹은 쓰레드)만 임계 구역에 진입한다.
- 임계 구역이 비어있을 때만 진입할 수 있다.
: 진행(Progress) - 다음에 진입할 프로세스는 유한 시간 내에 결정한다.
: 유한대기(Bounded Waiting) - 어떤 프로세스라도 유한시간내에 진입할 수 있다.
즉, 기다리면 언젠가 진입하게 된다.

2. 프로세스와 스레드

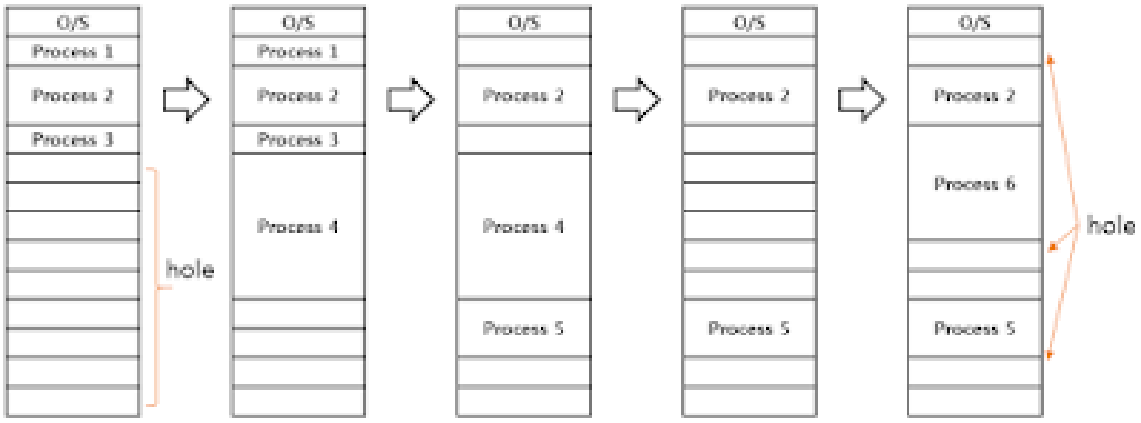
(윤영이가 정리한 부분인데 그림이 잘 나와있어서 첨부합니다)



생성: 메모리에 프로세스가 올라감. PCB 획득
준비: 생성된 프로세스가 CPU 획득을 기다리는 상태
실행
대기: 입출력을 요청한 프로세스가 입출력을 기다리는 상태
완료: 작업이 완료됨. PCB 가 사라진 상태

3. 메모리관리

1) 연속메모리 할당(Contiguous Memory Allocation)

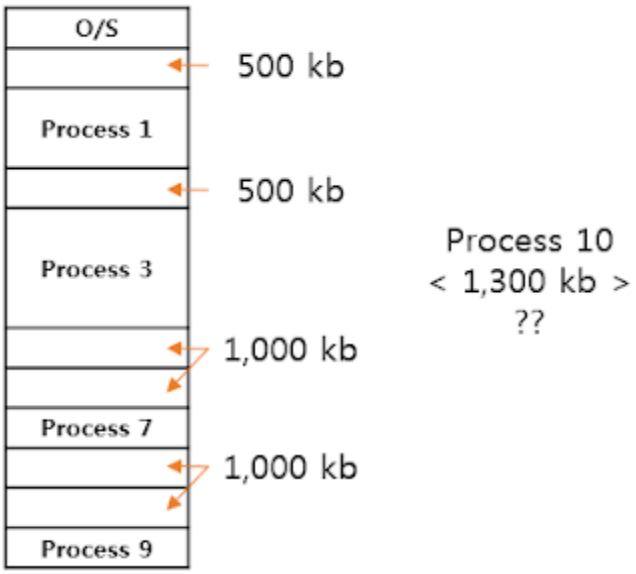


* 하나의 프로세스는 연속된 메모리 공간에 할당 받게 된다.
* 부팅직후에는 메모리홀이 매우 크지만 시간이 지날 수록 프로세스의 생성과 소멸이 반복되면서 불연속적으로 단편화된 작은 홀들이 흩어 지게 된다.

위 그림의 마지막에서 3 칸을 차지하는 Process 는 메모리에 진입할 수 없게 된다.

* 외부 단편화(External Fragmentation)

: 빈 메모리 공간이 있지만 이 공간이 여러개의 불연속적인 작은 공간으로 나뉘어져 있어서 프로세스를 할당할 수 없는 상태일 때 이 조각난 공간들에 대한 현상을 외부 단편화라고 한다. **외부 단편화는 심한 메모리 낭비의 원인**이 된다.

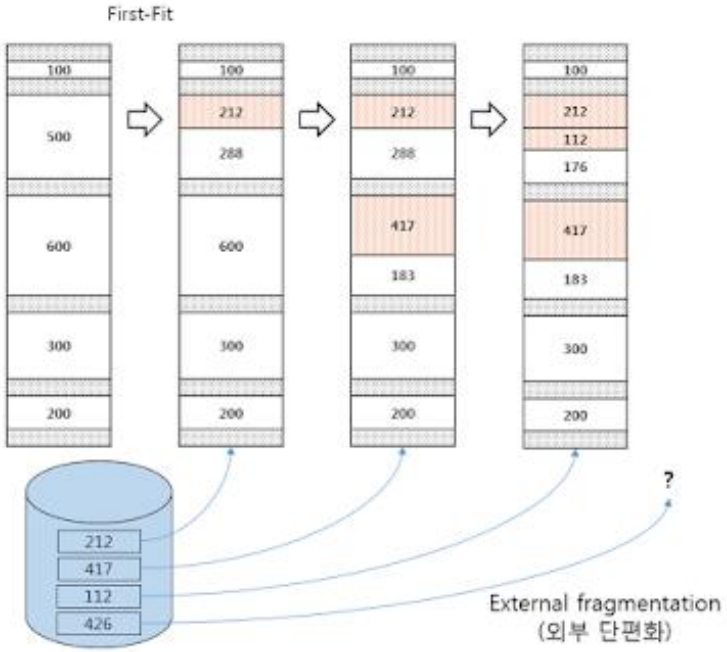


* 메모리 할당 방식

: 메모리를 프로세스에게 어떻게 할당하는 것이 바람직한가?에 대한 문제.

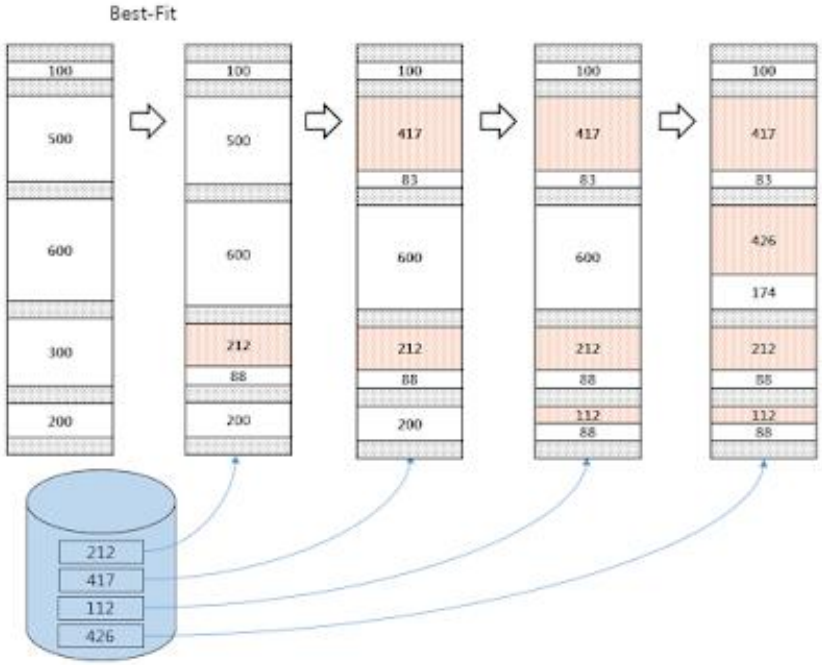
(1) First-fit (최초 적합)

: 최초로 할당받을 수 있는 크기의 공간에 무조건 할당된다.(위 또는 아래 부터)



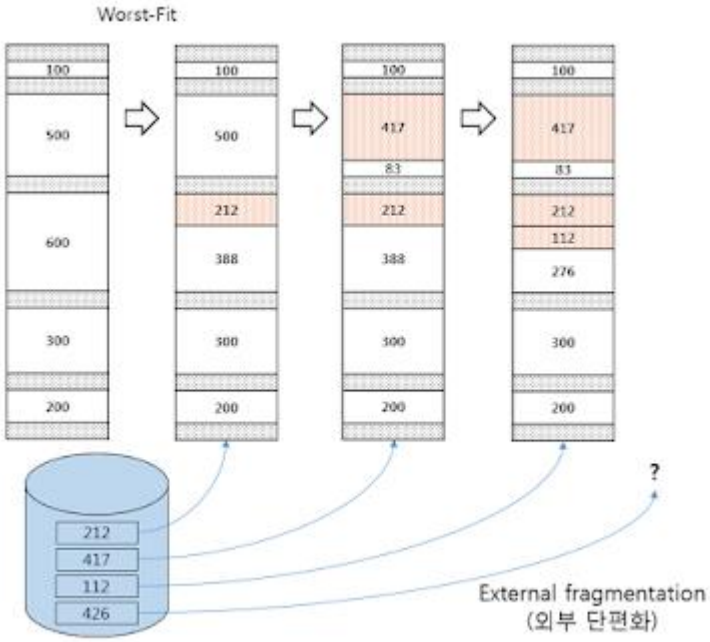
(2) Best-fit (최적 적합)

: 크기가 가장 근접한 공간에 할당된다.



(3) Worst-fit (최악 적합)

: 크기가 가장 많이 차이가 나는 공간에 할당된다.



*** 속도면에서는 "best-fit"이 최고이다.**

*** 이용률 면에서는 "first-fit"과 "best-fit"이 좋은 편이고 "worst-fit"은 좋지 못함.**

*** 어떤 경우이든 외부 단편화는 존재하기 마련이다.**

* 압축(Compaction)

: 외부 단편화의 해결방안으로 흩어져 있는 홀들을 하나의 커다란 홀로 통합하여 사용한다.

: 문제는 어떤 방식으로 압축을 할 것인가가 큰 문제가 된다. 상대적으로 커다란 프로세스를 이동하여 압축을 하게 된다면 압축을 위한 비용이 많이 들어 비효율적이게 된다.

4. 가상 메모리

* 출현 계기

: **물리 메모리(Physical Memory)의 크기에 따른 한계를 극복하기 위한 기술**

즉, 물리 메모리(Physical Memory)보다 큰 프로세스 실행이 가능하다.

예) 100MB 메인 메모리에서 200MB 크기의 프로세스 실행가능

원래는 100MB 인 메인 메모리보다 큰 프로세스는 실행이 불가.

* 적용 방법

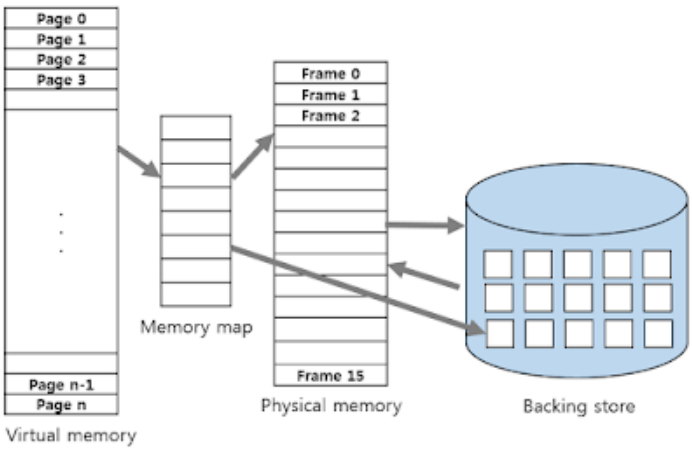
: **현재 실행에 필요한 부분만 메모리에 올린다!**

즉, 프로세스 이미지를 모두 메모리에 올릴 필요는 없다.

오류 처리, 배열의 일부, 워드프로세스에서 정렬, 표 기능 등은 제외가능하다.

동적 적재 (dynamic loading)와 비슷한 개념이다.

1) 요구 페이징(Demand Paging)

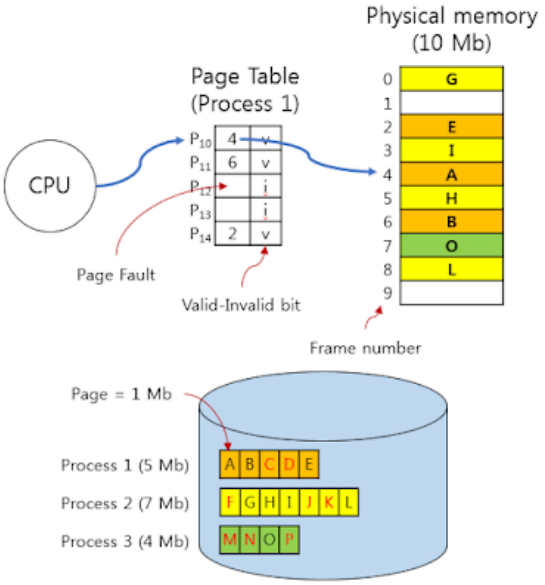


* 대체로 "가상 메모리 = 요구 페이징"을 의미함.

* 프로세스를 페이지 단위로 나누고 CPU 가 처리하는데 필요한 페이지만 즉, 요구되는 페이지만 메모리에 적재하는 것이어서 "요구 페이징(demand paging)"이라고 한다.

* 프로세스는 페이지의 집합이다.

* 메모리에 적재되지 못한 프로세스의 나머지 페이지들은 backing store 에 저장되어져 있다.



* 위 그림의 경우 10 Mb 메인 메모리에 각각 5 Mb, 7 Mb, 4 Mb 인 프로세스 3 개를 모두 적재하였다. 단, 각 프로세스에서 필요한 페이지(3 개, 4 개, 1 개)만 적재되어져 있다.

* 페이지 부재(Page Fault)

: CPU 가 필요로 하는 Page 가 메인 메모리에 적재되어져 있지 않은 경우를 의미한다.

: 페이지 부재(Page Fault)가 발생하면 해당 페이지를 메모리에 적재해주어야 한다.

* 순수 요구 페이징(Pure Demand Paging)

: 순수하게 현재 시점에 필요한 페이지만 적재한다.

: 장점 - 메모리 절약을 최대화 할 수 있다.

: 단점 - 시작하자 마자 페이지 부재가 일어난다. 속도가 느려진다.

* 프리페이징(Prepaging)

: 당연히 사용될 페이지라고 예측되어지는 페이지는 미리 적재한다.

: 장점 - 미리 적재되어져 있기 때문에 속도가 빠르다.

: 단점 - 페이지 부재가 적다. 만약 사용되지 않으면 메모리 낭비가 된다.

* Swapping vs Demand Paging

: Swapping - 프로세스 단위로 이동

: Demand Paging - 페이지 단위로 이동