메모리 관리1

logical address vs physical address

logical address 논리적 주소 (가상 주소 공간)

- 개발자나 프로세스가 프로세스 내에서 사용하는 주소, 코드나 변수 등에 대한 주소
- 프로세스마다 독립적으로 가지는 주소 공간 → 상대주소
  - 변수 n의 주소가 100번지라는 뜻은 논리 주소가 100이고 물리주소는 모름
  - 실제 메모리의 주소는 아니다.
  - cpu가 다루는 모든 주소는 logical address
- 각 프로세스마다 0번지부터 연속적으로 시작
- 사용자나 프로세스는 결코 물리주소를 알 수 없다.

physical address (물리적 주소)

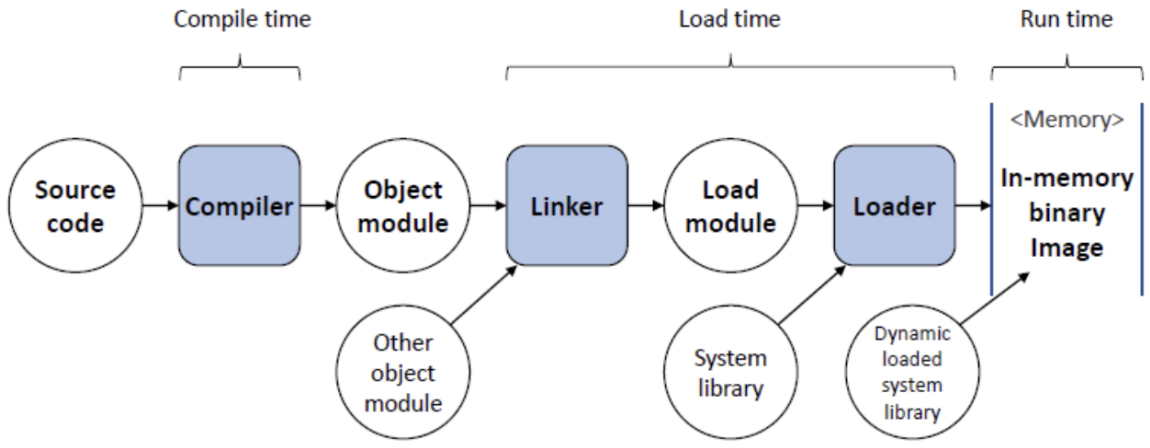
- 물리 메모리(ram) 에 매겨진 주소, 하드웨어에 의해 고정된 메모리 주소
- 주소바인딩 : 주소를 결정하는 것
  - symbol address → logical address → physical address

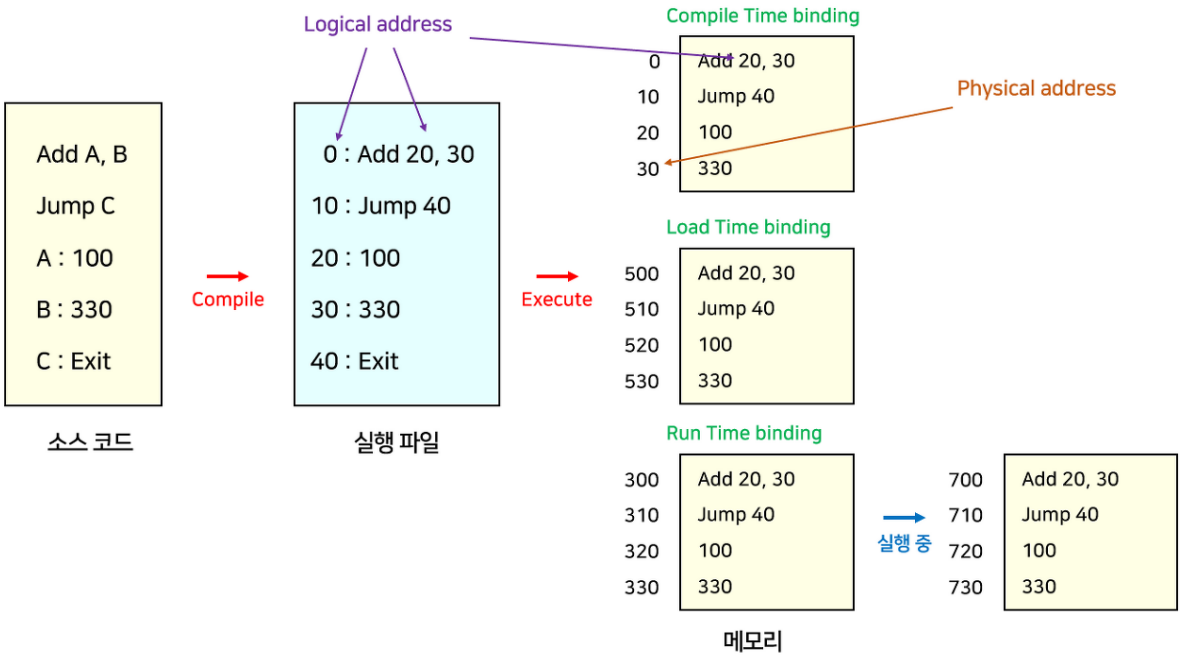
Address binding (주소 바인딩)

프로그램의 논리주소를 실제 메모리의 물리주소로 매핑하는 작업

- compile time binding
  - 물리적 메모리 주소가 컴파일 때 정해진다.
  - 프로세스가 메모리의 어느위치에 들어갈지 미리 알고 있다면 컴파일러는 절대 코드 생성
  - 컴파일 타임의 주소 할당은 논리적 주소와 물리적 주소가 동일하다.

    - 시작 위치 변경 시 재 컴파일이 발생한다.
- load time binding
    - 프로세스가 메모리의 어느 위치에 들어갈지 미리 알 수 없다면 컴파일러는 Relocatable code를 생성해야한다.
        - Relocatable code (재배치 가능코드) : 메모리의 어느 위치에서나 수행될 수 있는 기계 언어 코드
    - loader 책임하에 프로세스를 메모리에 load하는 시점에 물리적 메모리 주소를 부여한다.
    - 컴파일러가 재배치 가능 코드를 생성한 경우 가능
    - 로드 타임 주소 할당은 논리적 주소와 물리적 주소가 다르다.
- exectution time binding (runtime binding)
    - 수행 시작 이후 프로세스가 실행될 때 프로세스의 메모리 상 위치를 옮길 수 있음
    - runtime 상황에서 물리적 주소가 결정되며, CPU가 주소를 참조할 때 마다 binding을 점검 (address mapping table)
    - 하드웨어적인 지원 필요
        - base and limit registers
            - 접근 할 수 있는 물리적 주소의 최솟값을 나타냄
            - 잘못된 메모리를 참조하지 않도록 막아줌
        - mmu (Memory Management Unit) : 논리적 주소를 물리적 주소로 바꿈
- 그림





Memory management unit

- mmu
  - 논리주소를 물리주소로 바꿔주는(매핑해주는) 하드웨어 장치
  - cpu가 발생시킨 논리 주소는 mmu에 의해 물리주소로 바뀌어 메모리에 도달
  - 오늘날에는 cpu에 내장
- mmu scheme
  - 사용자 프로세스가 CPU에서 수행되며 생성해내는 모든 주소값에 대해 base register의 값을 더한다.
- user program
  - 논리주소만을 다룬다
  - 실제 물리주소를 볼 수 없으며 알 필요가 없다

> 💡 변수의 주소는 논리주소? 물리주소?
>
> 프로그램 내에서 변수의 주소는 논리 주소이다.

hardware support for address translation

- 운영체제 및 사용자 프로세스간의 메모리 보호를 위해 사용하는 레지스터
  - relocation register (base register) : 물리적 메모리 주소의 최소값
  - limit register : 논리적 주소 범위

Dynamic loading

- 프로세스 전체를 메모리에 올리는 것이 아니고 필요할 때 메모리에 올리는 것

overlays

- 메모리에 프로세스 부분 중 실제 필요한 정보만 올리는 것
- 프로세스의 크기가 메모리보다 클때 사용
- 운영체제 지원 없이 사용자에 의해 구현

swapping

- 프로세스를 임시로 디스크에 보냈다가 다시 메모리에 올리는 상황에서 사용
- swapping : 프로세스를 일시적으로 메모리에서 backing store에 보내는 것
  - swap in : 메모리로 들여보내는 것
  - swap out : 디스크로 내보내는 것

dynamic linking

- linking을 실행시간까지 미루는 기법
- static linking : 라이브러리가 실행 파일 코드에 포함되어 메모리가 낭비된다.

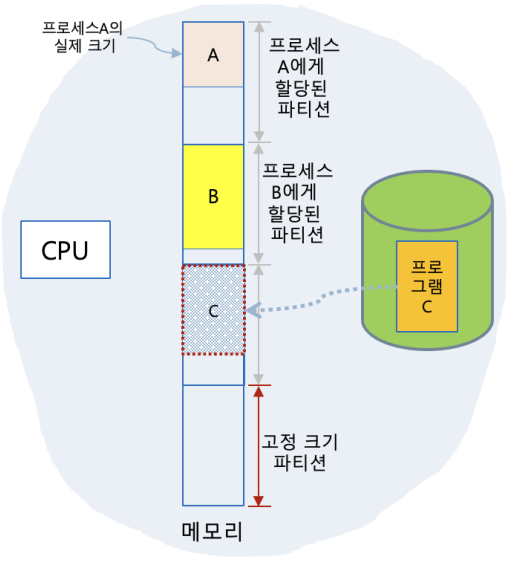
Allocation of Physical Memory

- 메모리는 두 영역으로 나뉨
  - OS 상주영역

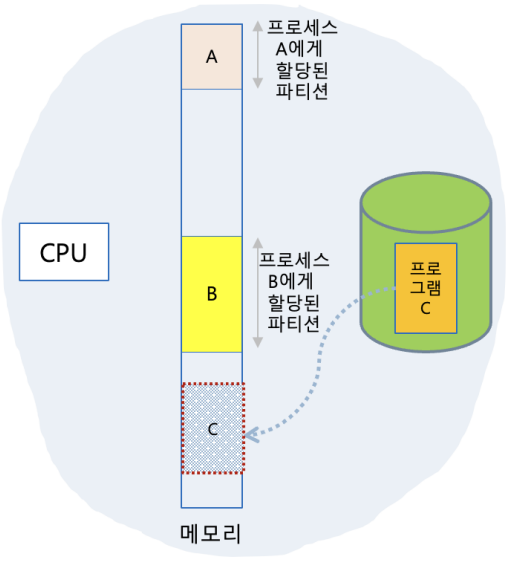
    - 사용자 프로세스 영역
- 사용자 프로세스 영역 할당
    - contignous allocation (연속 할당) : 프로세스별로 연속된 한 덩어리의 메모리 할당
        - fixed partition allocation (고정 크기 할당)
        - veriable partition allocation (가변 크기 할당)
    - noncontignous allocation (분할 메모리 할당)
        - fixed partition allocation (고정 크기 할당) : segmentation 기법
        - veriable partition allocation (가변 크기 할당) : paging 기법

Contiguous allocation (연속 할당)

- 고정 분할 방식
    - 분할의 크키가 모두 동일하거나 서로 다를 수 있다.
    - 분할 당 하나의 프로세스가 적재되기에 메모리에 load되는 프로세스의 수가 정해짐
    - 수행가능한 프로세스의 최대 크기도 제한됨
    - 관리가 편함
- 가변 분할 방식
    - 프로세스의 크기를 고려하여 할당하기에 분할의 크기나 개수가 동적으로 변함
    - 기술적인 관리 기법 필요



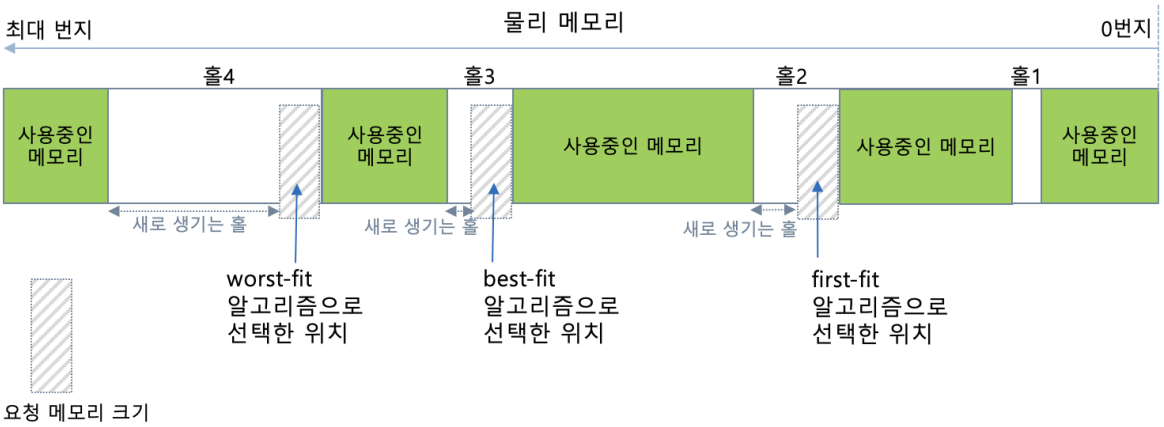
메모리를 고정 크기의 파티션으로 나누고
각 프로세스를 하나의 파티션에 배치



각 프로세스를 자신의 크기만한 가변 크기 파티션 할당

- hole
  - 메모리를 분할하는 각 단위는 block
  - 프로세스가 사용할 수 있는 block이 hole임
  - 프로세스가 들어갈 가장 적절한 hole을 찾는 방식을 dynamic storage-allocation problem
    - ffirst-fit : 최초 적합
      - 비어있는 파티션 중 맨 앞 요청 크기보다 큰 파티션 선택
      - 할당 속도 빠르고 단편화 발생 가능성
    - best-fit : 최적 적합
      - 비어있는 파티션 중 요청을 수용하는 가장 작은 파티션 선택
      - 파티션이 크기별로 정렬되어 있지 않으면 전부검색한다.
      - 가장 작은 홀 생성
    - worst-fit : 최악 적합
      - 비어있는 파티션중 요청을 수용하는가장 큰 파티션 선택
      - 파티션이 크기별로 정렬되어 있지 않으면 전부검색한다.
      - 가장 큰 홀 생성

⇒ first-fit과 best-fit이 worst-fit 보다 속도, 공간 이용률 측면에서 효과적



- compaction
  - external fragmentation 문제를 해결하는 방법
  - 사용중인 메모리 영역 한군데로 몰고 다른 hole들을 다른 한 곳으로 몰아서 큰 block을 만드는 것
  - 프로세스의 주소가 실행시간에 동적으로 재배치 가능한 경우에만 수행가능