8️⃣ 메모리 관리1

Logical address (= virtual address)

- 프로세스마다 독립적으로 가지는 주소공간
- *각 프로세스마다 0번지부터 시작*
- **CPU**가 보는 주소는 logical address → 매번 주소 변환해서 메모리에 접근

Physical address

- 메모리에 실제 올라가는 위치

주소 바인딩 (address binding)

주소를 결정하는 것 → 주소 변환은 *하드웨어*가 해줌

symbolic address → logical address → physical address

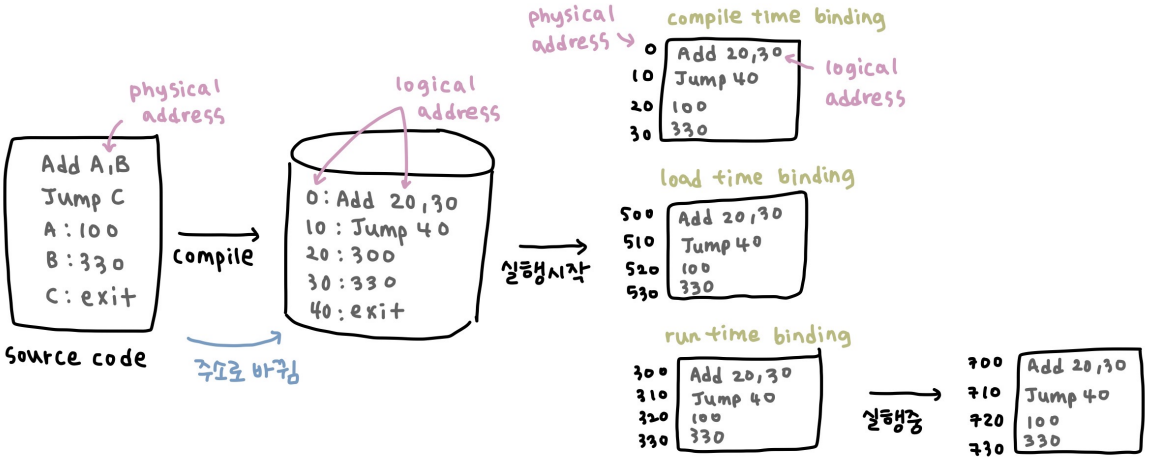
- **compile time binding**
  - 물리적 메모리 주소가 컴파일 시 알려짐 → *컴파일 할 때 이미 주소 결정됨*
  - 시작 위치 변경시 재컴파일
  - 컴파일러는 절대 코드 (absolute code) 생성

    절대 코드 → compil binding 에 의해 만들어진 코드
- **load time binding**
  - *loader의 책임 하에* 물리적 메모리 주소 부여
  - 컴파일러가 재배치 가능 코드 (relocatable code)를 생성한 경우 가능
- **execution time binding** (= run time binding)
  - *수행이 시작된 이후*에도 프로세스의 메모리 상 *위치를 옮길 수 있음*
  - CPU가 메모리 참조할 때마다 binding을 점검 (address mapping table)
  - *하드웨어적인 지원이 필요* (MMU) → 매번 주소변환이 필요

⇒ load time과 execution time 둘 다 실행 시에 물리적 주소로 바뀜

load → 실행 시점에 바뀜 + 실행 도중에 주소가 바뀌지 않음

execution → 실행될 때 주소 바인딩 + 프로그램 실행 도중 주소 바인딩 가능



Memory-Management Unit (MMU)

locigal address를 physical address로 *매핑해주는 hardware device*

- MMU scheme

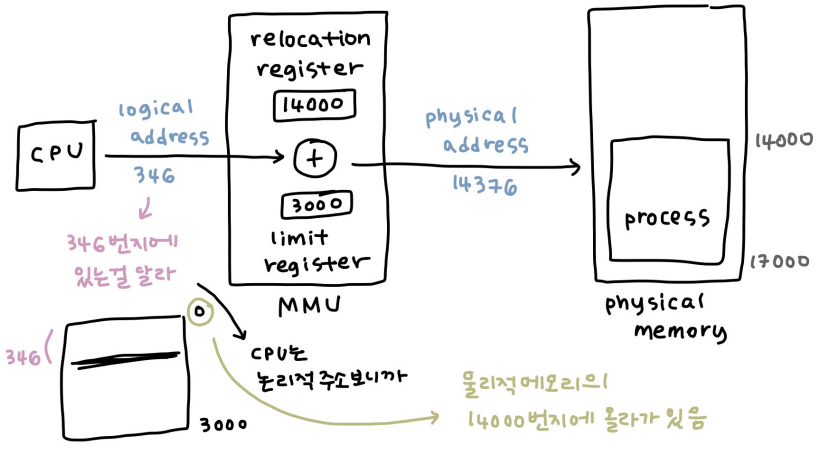
  사용자 프로세스가 CPU에서 수행되며 생성해내는 *모든 주소값에 대해 base register (relocation register)의 값을 더함*

- User program

  logical address만을 다룸 → *사용자는 논리적 주소만 봄*

  실제 physical address를 볼 수 없으며 알 필요가 없음

Dynamic relocation



- relocation register : 프로그램의 *물리적 메모리의 첫 시작*을 담음
- limit register : *프로그램의 크기* → 다른 프로그램에 접근하려는 악의적인 신호를 막기 위해
- MMU ⇒ 두 개의 register만 있으면 됨
- 운영체제 및 사용자 프로세스 간의 메모리 보호를 위해 사용하는 레지스터

- relocation register (=base register) : 접근할 수 있는 물리적 메모리 주소의 최소값
- limit register : 논리적 주소의 범위

Some terminologies

- dynamic loading
- dynamic linking
- overlays
- swapping

Dynamic loading

- 프로세스 전체를 메모리에 미리 다 올리는 것이 아니라 *해당 루틴이 불려질 때 메모리에 load*하는 것
- *메모리 utilization의 향상* → 실행되는 부분만 올리면 낭비 없음
- 가끔씩 사용되는 많은 양의 코드의 경우 유용
- 운영체제의 특별한 지원 없이 *프로그램 자체에서 구현 가능* (OS는 라이브러리를 통해 지원 가능)

Overlays

- 메모리에 프로세스의 부분 중 *실제 필요한 정보만을 올림*
- 프로세스의 크기가 메모리보다 클 때 유용
- 운영체제의 지원없이 *사용자에 의해 구현*
- 작업공간의 메모리를 사용하던 초창기 시스템에서 수작업으로 프로그래머가 구현
- “manual overlay” → 프로그래밍이 아주 복잡

Swapping

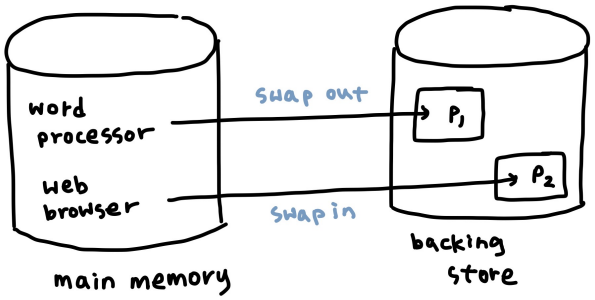
프로세스를 일시적으로 메모리에서 **backing store**로 쫓아내는 것 → 메모리에서 통째로 쫓아냄

backing area = *swap area* → 많은 사용자의 프로세스 이미지를 담을 만큼 충분히 빠르고 큰 저장 공간

swap in / swap out

- 일반적으로 중기 스케줄러(swapper)에 의해 swap out 시킬 프로세스 선정 → 쫓겨나면 *suspend*상태
- priority -based CPU scheduling algorithm
  - priority가 낮은 프로세스를 swapped out 시킴
  - priority가 높은 프로세스를 메모리에 올려 놓음
- compile time 혹은 load time binding에서는 *원래 메모리 위치*로 swap in 해야 함
  → 프로그램 종료될 때까지 주소가 유지되기 때문에 주소가 바뀌면 안됨

- execution time binding에서는 추후 빈 메모리 영역 *아무 곳에나* 올릴 수 있음 → 효율적
- swap time은 대부분 transfer time(swap되는 양에 비례하는 시간)임



Dynamic linking

linking을 *실행시간 (execution time)까지 미루는 기법* / linking : 라이브러리를 연결하는 작업

- **static linking** → static library
  - 라이브러리가 *프로그램의 실행 파일 코드에 포함*됨
  - 실행 파일의 크기가 커짐
  - 동일한 라이브러리를 각각의 프로세스가 메모리에 올리므로 메모리 낭비
- **dynamic linking** → dynamic library
  - 라이브러리가 *실행시 연결*됨 → 실행 파일에 존재하지 않고 *별도로 존재*
  - 라이브러리 호출 부분에 라이브러리 루틴의 위치를 찾기 위한 stub이라는 작은 코드를 둠
  - 라이브러리가 이미 메모리에 있으면 그 루틴의 주소로 가고 없으면 디스크에서 읽어옴
  - 운영체제의 도움이 필요

Allocation of Physical Memory

메모리는 일반적으로 두 영역로 나뉘어 사용

- *OS 상주 영역* → interrupt vector와 함께 낮은 주소 영역 사용
- *사용자 프로세스 영역* → 높은 주소 영역 사용

사용자 프로세스 영역의 할당 방법

- **contiguous allocation** (연속 할당)

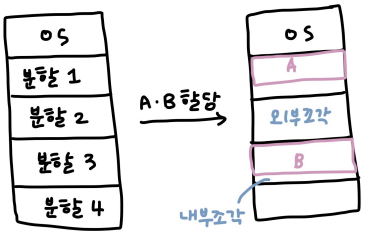
  각각의 프로세스가 메모리의 연속적인 공간에 적재되도록 하는 것 → 프로그램이 통째로 들어감 (register 2개면 가능)
  - fixed partition allocation / variable partition allocation
- **noncontiguous allocation** (불연속 할당) → 프로그램이 쪼개져서 올라갈 수 있음

  하나의 프로세스가 메모리의 여러 영역에 분산되어 올라갈 수 있음

- paging / segmentation / paged segmentation

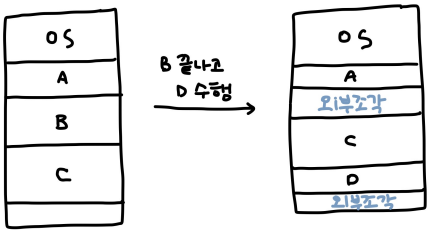
Contiguous allocation

- 고정 분할 방식



→ 미리 분할해 메모리 나눠놓은 것

- 가변 분할 방식



→ 비어있는 공간이어도 크기에 따라 낭비될 수 있음

- **외부조각** (external fragmentation)
  - 프로그램 크기보다 분할의 크기가 *작은 경우*
  - 아무 프로그램에도 배정되지 않은 빈 곳인데도 프로그램이 올라갈 수 없는 작은 분할
- **내부조각** (internal fragmentation)
  - 프로그램 크기보다 분할의 크기가 *큰 경우*
  - 하나의 분할 내부에서 발생하는 사용되지 않는 메모리 조각
  - 특정 프로그램에 배정되었지만 사용되지 않는 공간

Hole

가용 메모리 공간

- 다양한 크기의 hole들이 메모리 여러 곳에 흩어져 있음
- 프로세스가 도착하면 *수용가능한 hole을 할당*
- 운영체제는 다음의 정보를 유지 → 할당 공간, 가용공간 (hole)

Dynamic storage - allocation problem

가변 분할 방식에서 size n인 요청을 만족하는 *가장 적절한 hole*을 찾는 문제

- **first-fit**

  size가 n 이상인 것 중 최초로 찾아지는 hole에 할당
- **best-fit**

  size가 n 이상인 것 중 가장 작은 hole을 찾아서 할당
  - hole들이 리스트가 크기 순으로 정렬되지 않은 경우 *모든 hole의 리스트를 탐색*해야 함
  - 많은 수의 아주 작은 hole들이 생성됨

- **worst-fit**

  가장 큰 hole에 할당

  - 역시 *모든 리스트를 탐색*해야 함
  - 상대적으로 아주 큰 hole들이 생성됨

Contiguous allocation

compaction

- external fragmentation 문제를 해결하는 한 가지 방법
- *사용 중인 메모리 영역을 한군데*로 몰고 *hole들을 다른 한 곳*으로 몰아 더 큰 block을 만드는 것
- 매우 비용이 많이 드는 방법
- 최소한의 메모리 이동으로 compaction하는 방법은 매우 복잡한 문제
- compaction은 프로세스의 주소가 실행시간에 동적으로 재배치 가능한 경우에만 수행될 수 있음