CS 스터디(메모리 관리)

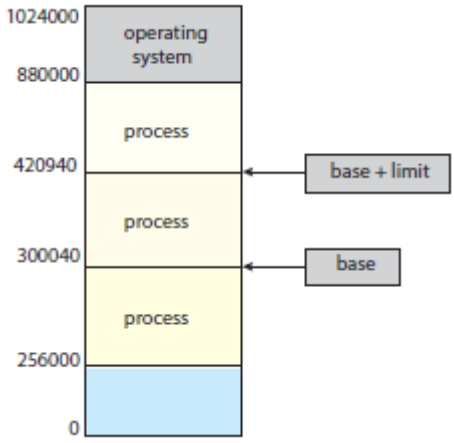
메모리 관리 전략

Background

프로그램이 실행되기 위해서는 디스크에서 메모리로 옮겨져야 한다.

→ 프로세스가 메인 메모리 상에서 독립된 공간을 확보할 수 있어야 함

- **Base & Limit registers**



두 개의 register를 사용해 한 프로세스의 range of legal addresses를 설정해줌.

Base register = legal physical memory address의 가장 작은 값

Limit register = range의 사이즈

base ≤ 프로세스 구역 < limit

cpu가 모든 memory access마다 base와 limit 사이에 있는지 확인해서 독립된 공간을 확보할 수 있도록 해준다.

메모리에 어떻게 배치시킬 것인가?

→ Address Binding

언제 일어나는가?

- Compile Time

  컴파일러가 실행프로그램을 만들때 이미 함수나 변수가 적재될 위치가 정해짐 → 절대 코드(↔ 상대 코드)가 생성됨(실행 가능한 기계어 코드)

  주소 변환을 하려면 다시 컴파일 해야 함

- Load Time

  컴파일 시 프로세스가 메모리 내의 어디로 올라올지 모르면 컴파일러는 일단 relocatable code로 생성 → 언제 바인딩? 프로그램이 주 메모리로 실제로 적재되는 시간에 이루어짐

- Execution Time

  프로세스가 execution 중에 메모리 내의 세그먼트에서 다른 세그먼트로 이동할 수 있다면 바인딩이 런타임까지 허용된다고 한다. 즉 프로그램이 실행 후에도 주소를 변경할 수 있다 → 요즘 컴퓨터가 사용하는 방식

  하드웨어가 지원을 해야 함(Base & Limit registers)

여기까지는 프로세스가 실행되기 위해 프로그램 전부와 프로세스의 모든 자료가 미리 메모리에 올라와있어야 한다.

physical memory는 한정적이기 때문에 효율적으로 사용해야 한다.

- Dynamic Loading

  사용되지 않는 함수는 로딩하지 않는다

  실제 호출되기 전까진 메모리에 올라오지 않고 재배치 가능한 상태로 디스크에서 대기

  OS의 지원이 필요하지 않음

- Dynamic Linking

  linking → 컴파일된 obj파일을 라이브러리와 링킹하여 실행파일로 만드는것

  실행시까지 링킹을 지연시킴

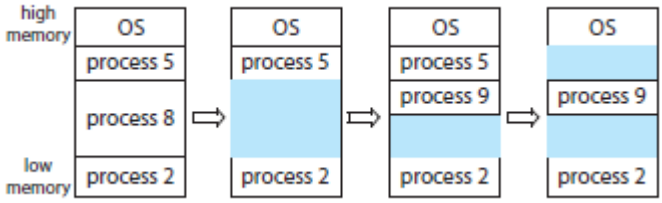
Contiguous Allocation

base & limit register를 사용해 연속적으로 할당을 한다

→ 프로세스가 인접해 있는 상황에서 사이즈가 변한다면?

→ 메모리 침범에 따른 많은 에러 발생

- Multiple Partition Allocation



메모리를 여러 partition으로 나눠 프로세스를 할당

프로세스가 도착하면 수용가능한 크기를 가진 메모리에 할당

os는 allocated partition과 free partition(hole)에 대한 정보를 유지해야 한다.

- First fit

  보통 많이 사용(처리 속도 향상)

- Best fit
- Worst fit

Fragmentation

- External Fragmentation

  전체 메모리의 공간은 충분한 공간을 가지고 있지만 그것들이 작은 공간들로 여러곳에 분산되어 있는 상태. 연속적이지 않음

  해결 방법 - compaction

  빈공간(hole)을 이동시켜 큰 공간을 다시 만든다.

  재배치가 동적으로 일어날 수 있을 경우에만 가능하다

  런타임 시에 가능해야 되므로 비용이 크고 어렵다.

- Internal Fragmentation

  할당된 메모리 크기가 프로세스 크기보다 큰 경우. 남은 공간은 사용할 수 없음

여기까지 다 옛날 얘기고

Paging

메모리를 페이지 단위로 관리, 하드웨어 의존적임

- 장점

- external fragmentation을 방지한다.
- 논리 주소 공간이 연속적이여야한다는 제약을 없앤다.

- 단점
  - internal fragmentation

frame - physical memory가 일정 사이즈 블록으로 나뉜것

보통 2kb or 4kb

pages - logical memory가 일정 사이즈 블록으로 나뉜것

CPU에서 나오는 모든 주소는 2가지로 구성된다

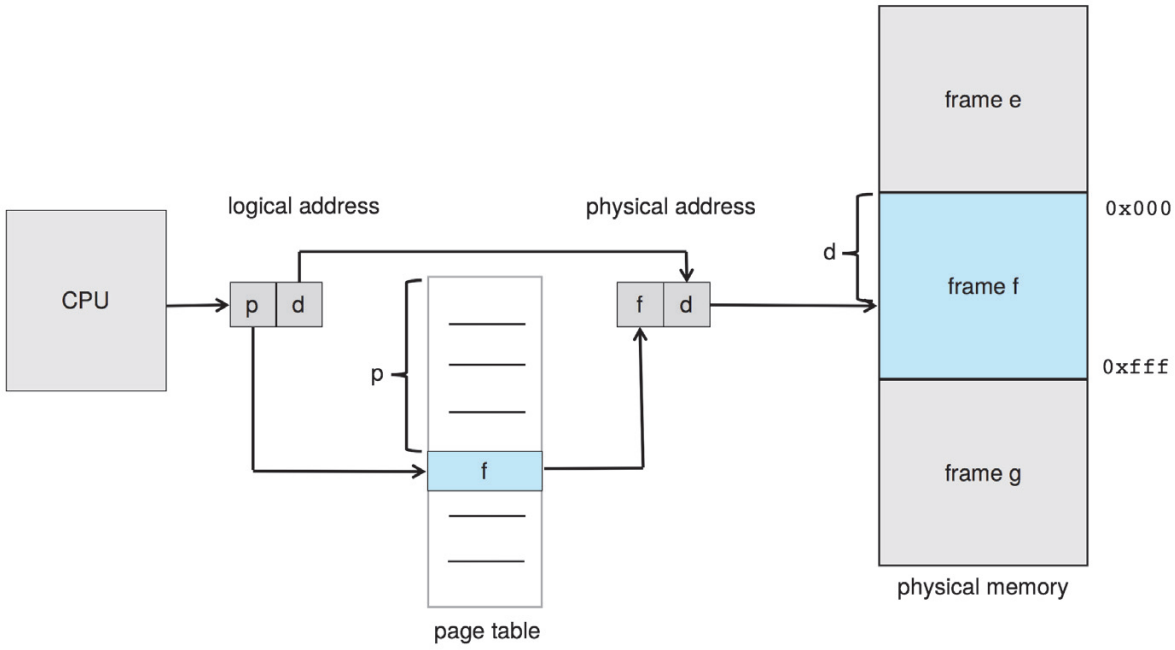
- page number

  page table에서 page의 인덱스로 사용

  물리 메모리의 시작 주소를 가지고 있다

- page offset

  주소로 부터 얼마나 떨어져 있는지를 나타냄



페이지 테이블은 메인 메모리에 저장돼있다.

Page-table base register(PTBR)은 페이지 테이블 시작 위치를 가르킴

Page-table length register(PTLR)은 페이지 테이블의 사이즈를 저장

→ 두번 접근(테이블 접근에 한 번, 데이터/명령어 접근에 한 번)

해결하기 위해 **translation look-aside buffers**(TLBs)

> 일부 페이지 넘버와 프레임 넘버를 캐싱하고 있어서 빠르게 메모리를 읽어오는 하드웨어 캐시

page table의 구조

- hierarchical page table
  - two level
  - three level
- hashed page table
- inverted page table

Segmentation

페이징은 같은 크기의 블록으로 나누지만 세그멘테이션은 가상 메모리를 서로 다른 크기로 나누는 기법

크기가 서로 다르기 때문에 페이징을 할 때처럼 메모리를 미리 분할해 둘 수 없다.

→ external fragmentation 발생 가능

paging → internal

segmentation → external