메모리 & 캐시

메모리 구조

코드 영역

- 실행할 프로그램의 코드

데이터 영역

- 전역 변수, 정적 변수 등 데이터

힙 영역

- 사용자가 동적으로 할당하는 공간
- 프로그램 실행 중 할당되기 때문에 런 타임시 크기 결정

스택 영역

- 지역 변수와 매개 변수 저장되는 공간
- 컴파일 타임에 따라 크기가 결정된다

메모리 계층 구조

레지스터

- CPU가 요청을 처리하는데 필요한 데이터를 일시적으로 저장하는 기억장치

캐시

- 데이터를 미리 복사해 놓는 임시 저장소이자 빠른 장치와 느린 장치에서 속도 차이에 따른 병목 현상을 줄이기 위한 메모리
- 실제로 메모리와 CPU간의 속도 차이가 너무 크기 때문에 중간에 레지스터 계층을 둬서 속도차이를 해결한다.

- 이와 같이 계층과 계층 사이에 있는 계층을 캐싱 계층이라고 한다.
- L1 캐시 : 코어 안에 내장된 캐시로 속도가 빠르다
- L2 캐시 : 프로세서 안에 내장된 캐시로 상대적으로 속도가 느리고 많은 용량 저장 가능
- L3 캐시 : 자주 사용 X

메인 메모리

- 주기억장치로 컴퓨터에서 수치, 명령, 자료 등을 기억하는 컴퓨터 하드웨어 장치이다.

RAM(Random Access memory)

- 빠른 접근을 위해 데이터를 단기간 저장하는 구성 요소
- 사용자가 요청하는 프로그램이나 문서를 스토리지 디스크에서 메모리로 로드하여 각각의 정보에 접근
- 휘발성 기억 장치 (전원 종료 시 기억된 내용 삭제)
- 어느 위치에서든 똑같은 속도로 접근하여 읽고 쓰는 것이 가능
- 전원이 유지되는 동안 CPU의 연산 및 동작에 필요한 모든 내용 저장

ROM (Read Only Memory)

- 컴퓨터에 지시사항을 영구히 저장하는 비휘발성 메모리(고정 기억 장치)
- 변경 가능성이 희박한 기능 및 부품에 사용

하드 디스크 드라이브

- 보조 기억 장치로 비휘발성 데이터 장소
- 대중적이며 용량 대비 가격이 가장 저렴하다.

캐시 메모리

> 속도가 빠른 장치와 느린 장치 간의 속도차에 따른 병목 현상을 줄이기 위한 메모리

캐시 적중률 (Hit Rate)

캐시 적중

- 캐시 적중이란 캐시에서 파일이 요청되고 캐시가 해당 요청을 이행할 수 있을 때 발생

캐시 미스

> CPU가 참조하려는 데이터가 캐시 메모리에 없을 때 발생

- Compulsory Miss : 특정 데이터에 처음 접근할 때 발생
- Capacity Miss : 캐시 메모리의 공간이 부족해서 발생
- Conflict Miss : 캐시 메모리에 A와 B 데이터를 저장해야 하는데, A와 B가 같은 캐시 메모리 주소에 할당되어 있어서 발생

캐시 적중률 = (캐시 적중) / (캐시 적중 + 캐시 미스)

캐시 지역성

> 데이터에 대한 접근이 시간적 혹은 공간적으로 가깝게 발생하는 것

- 캐시 적중률을 극대화하여 캐시가 효율적으로 동작하기 위해 사용되는 성질

  ### 공간지역성 (Spatial Locality)

  - 최근에 사용했던 데이터와 인접한 데이터가 참조될 가능성이 높다는 특징

  ### 시간지역성 (Temporal Locality)

  - 최근에 사용했던 데이터가 재참조될 가능성이 높은 특징

메모리 단편화

> 메모리 공간이 작은 조각으로 나눠져서 사용 가능한 메모리가 있으나 할당이 불가능한 상태

내부 단편화 (Internal Fragmentation)

- 프로세스가 필요한 양보다 더 큰 메모리가 할당되어서 프로세스에서 사용하는 메모리 공간이 낭비되는 상황

외부 단편화 (External Fragmentation)

- 메모리 할당 및 해제 작업 시 작은 메모리가 중간중간 존재하는데 이 메모리들로 인해 총 메모리 공간은 충분하지만 실제로 할당을 할 수가 없는 상황

페이징

> 가상 메모리를 같은 크기의 블록으로 나눈 것을 페이지라고 하며 RAM을 페이지와 같은 크기로 나눈 것을 프레임이라고 할 때

페이징 기법이란 사용하지 않는 프레임을 페이지에 옮기고 필요한 메모리를 페이지 단위로 프레임에 옮기는 기법

- 연속적이지 않은 공간도 활용할 수 있기 때문에 외부 단편화 해결
- 페이지가 필요한 메모리보다 큰 경우에는 여전히 내부 단편화 문제 발생
- 페이지 단위를 작게 한다면 내부 단편화는 해결되지만 page mapping 과정이 많아져서 오히려 비효율적

세그멘테이션

> 가상 메모리를 서로 크기가 다른 논리적 단위인 세그먼트로 분할해서 메모리에 할당하는 기법

- 프로세스가 필요한 메모리 만큼 할당하기 때문에 내부 단편화 해결
- 중간에 프로세스가 메모리를 해제하게 될 경우 외부 단편화 문제 발생
- mapping을 위한 세그먼트 테이블 필요

동적 메모리 할당 문제

최초 적합 (First-Fit)

- 메모리를 할당할 수 있는 공간 중 가장 처음으로 발견하는 공간에 프로그램을 할당함
- 시간적인 측면에서 효율적

최적 적합 (Best-fit)

- 메모리를 할당할 수 있는 공간 중 가장 작은 가용 공간을 찾아 프로그램을 할당함
- 공간적인 측면에서 효율적
- 공간을 모두 탐색해야하기 때문에 오버헤드가 발생한다.

최악 적합 (Worst-fit)

- 가장 가용 공간이 큰 공간에 프로그램을 할당함

페이지 교체 알고리즘

FIFO (First in First Out)

> 가장 오래된 페이지를 교체

- 오랜된 페이지를 자주 참조하는 경우가 있음 ex) 배열 순회

OPT (Optimal)

> 앞으로 사용하지 않을 페이지를 교체

- 앞으로 사용하지 않을 페이지를 미리 아는 것이 불가능하여 구현 X

LRU (Last Recently Used)

> 가장 오랫동안 사용되지 않은 페이지 교체

- OPT 보다는 부족하지만 가장 근접한 성능을 보임

LFU (Last Frequently Used)

> 가장 적게 참조된 페이지 교체

- 페이지가 참조된 횟수를 추가로 사용해야하기 때문에 메모리가 낭비됨
- 구현이 복잡함

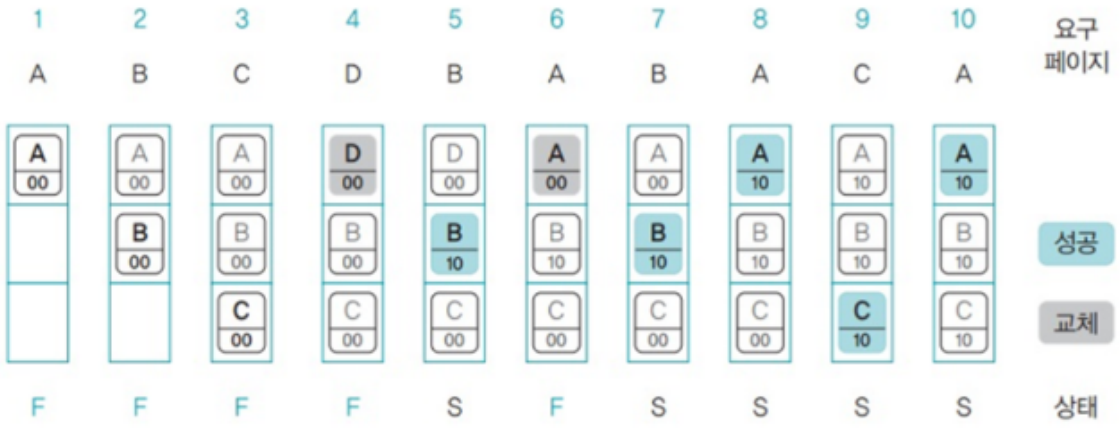
MFU (Most Frequently Used)

> LFU와 반대로 가장 참조가 많이 된 페이지를 교체

NUR (Not Used Recently)

> 추가 비트 2개를 사용하여 최근에 사용되지 않은 페이지를 교체

- 참조 비트 : 페이지의 접근 비트
- 변경 비트 : 페이지의 변경 비트
- 모든 페이지가 1,1이 되면 모든 페이지 비트를 0,0 으로 초기화



쓰레싱(Thrashing)

> 메모리 영역에 접근하게 될 때, 메모리에 페이지 부재율이 높은 것을 의미

- 심각한 성능 저하를 초래합니다.

Working set

Page Fault Frequency