제 3 장 주기억장치 관리

3.1 개요

- 계층적인 기억 장치

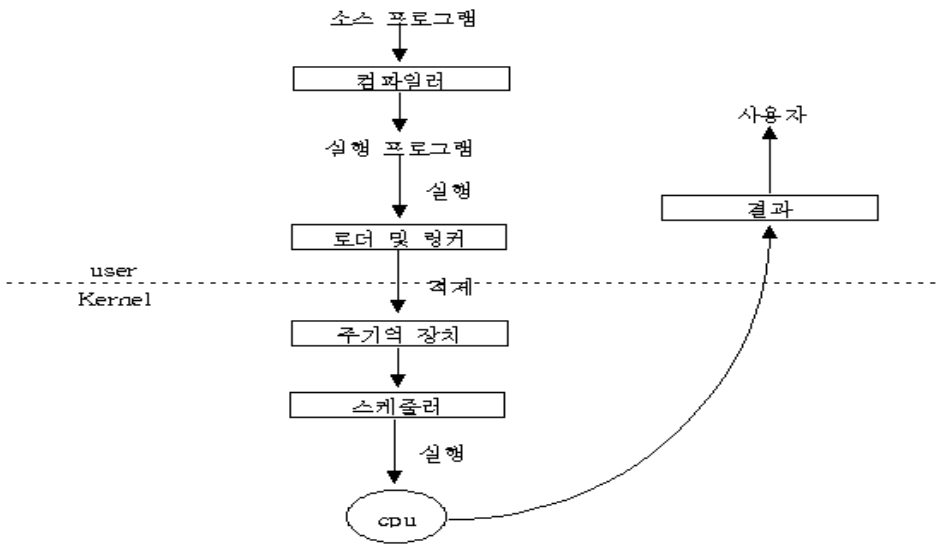
- 캐시 저장 장치(cache storage device)
- 주기억장치(main memory)
- 보조기억장치

3.3 계층적인 저장 장치의 구성

- p109 그림 3.1

3.4 프로그램의 실행 과정

- CPU는 실행할 프로그램을 주 기억 장치로 가져와야(load program into main memory) 실행이 가능
- 주기억 장치로 실행할 프로그램을 가지고 오는 과정(아래 그림)



3.4.1 컴파일(compile)

- 컴파일러(compiler)는 소스 프로그램을 이용하여 목적 코드를 생성하는 역할 수행

3.4.2 주소 연결(Address Binding)

- 프로그램은 일반적으로 상대 주소(relative address)의 개념을 갖는다
- 주소 연결은 왜 필요한 것일까 ?

- 재 배치(relocation)

① 컴파일 시간 바인딩(compile time binding) : 프로세스가 기억 장치 내에 적재될 위치를 **컴파일 시간에 결정하여 컴파일러는 절대 코드를 생성.**

② 적재 시간 바인딩(load time binding) :

③ 수행 시간 바인딩(execution time binding) : 프로세스의 주기억장치 주소 공간 할당(주소 계산)

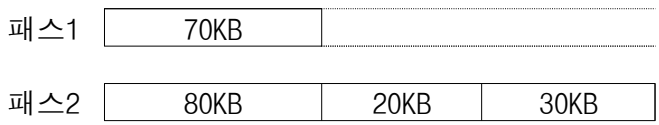
3.4.4 동적 연결(dynamic linking)

➢ 프로그램 연결이 수행 시간까지 지연

3.5 중첩(overlay)

➢ 어셈블러에서 패스1, 패스2는 동시 수행될 필요가 없음

- 패스 1 : 70K, 패스 2: 80K, symbol table : 20K, 공통 루틴 : 30K



3.6 스와핑(swapping)

➢ swap out : 주 기억 장치에서 보조 기억 장치로 이주

➢ swap in : 보조 기억 장치 → 주 기억 장치

3.7 기억 장치 구성 기법

3.7.1 단일 분할 할당(single-partition allocation)

- 가장 간단한 기억 장치 관리 기법
- 사용자가 주기억장치 관리에 대한 모든 권한을 소유
- 단점

- boundary register : 그림 3.2

3.7.2 고정 분할 다중프로그램

다중 프로그래밍(multiprogramming)
여러 개의 프로그램이 동시에 주기억장치에 상주하면서 CPU가 사용 가능할 경우 즉시 실행할 수 있도록 한다. 이 방식은 CPU의 유휴 시간(idle time)을 줄이고, 시스템 자원을 최대한 활용하는 것을 목표.
(장점) CPU 활용 극대화, 시스템 전반적으로 성능이 빨라진다.

- 고정 크기 주기억장치 공간 할당
- 그림 3.3

3.7.3 가변 분할 다중프로그램

- 프로세스가 필요한 만큼의 기억 공간을 동적으로 할당
- 사용을 하고난 공간은 반납(release)

3.7.4 compaction

- 여러 개의 작은 빈 공간을 합쳐서 하나의 큰 공간으로 합치는 작업
  : garbage collection

3.8 가상 기억 장치(Virtual Memory)

- Paging, segmentation 기법

가상기억장치(Virtual Memory)

가상기억장치는 실제 물리적 메모리보다 더 큰 메모리를 사용할 수 있도록 운영체제가 제공하는 기법으로, 프로그램 실행 시 필요한 부분만 주기억장치에 적재하여 효율적인 메모리 사용을 가능. 대용량 프로그램이 수행하는데 필요한 주기억장치의 공간만큼이 있는 것처럼 동작이 되도록 한다.

3.8.2 가상 저장 장치

- 가상 주소 공간과 실 주소 공간 사이의 맵핑이 필요

- map table

3.8.4 블록 사상(block mapping)

- map table에 저장되는 정보 양을 줄여야 한다.

- 블록의 크기를 가능한 크게
- 보조 기억 장치에서 주 기억 장치로 이동하는데 걸리는 시간이 많이 소모

- 블록 크기가 다른 경우 : 세그먼트(segment)기법
- 가상 주소 변환

3.8.5 페이지 기법

페이지 기법(paging)
주 기억장치 공간을 일정한 크기로 나갠다. 이 일정하게 나눈 크기를 페이지(page)라고 한다. 메모리 할당을 페이지 단위로 수행하는 것이 페이지 기법이다.

<페이징 시스템에서 주소 매핑 기법>

(1) 직접 사상(direct mapping)에 의한 페이지 주소 변환

- 그림 3.8(pp. 129)

(2) Associative mapping에 의한 페이지 주소 변환

- associative memory에 페이지 map table 정보를 둔다

(3) Associative/직접 사상을 결합한 페이지 주소 변환

- 가격이 비싸지 않으면서 빠른 사상 방법

- 그림 3.10(pp. 135)

3.8.6 세그먼트 기법(segmentation)

- 가변크기로 주기억장치 할당
- 저장 장치 보호하는 것이 복잡(보안에 취약)

세그먼트 기법
가변적인 크기의 주기억장치 공간을 할당
(장점) 효율적인 메모리 사용을 보장

<세그먼트 시스템에서 주소 매핑 기법>

- 직접 사상에 의한 세그먼트 주소 변환
  V=(s,d) --> R=s' +d

- 페이지/세그먼트 혼용 시스템에서의 주소 변환 기법
  - Associative/직접 사상 기법

3.9 페이지 교체 알고리즘(Page Replacement Algorithm)

- 주기억 장치에 빈 공간이 없을 경우, 커널은 공간 확보를 어떻게 할것인가 ?
  ---> 커널은 기존 주기억장치를 차지하고 있는 페이지를 대체시켜서 공간 확보
  ===> 페이지 교체 알고리즘

3.9.3 FIFO 페이지 교체 알고리즘

- 주 기억 장치에 가장 오래 있었던 페이지 교체

(1) FIFO 기법의 모순

3.9.4 LRU(Least Recently Used) 페이지 교체 알고리즘

- 가장 오랫동안 사용되지 않은 페이지를 교체

3.9.5 LFU(Least Frequently Used) 페이지 교체 알고리즘

- 호출 빈도가 가장 적은 페이지가 교체
- 페이지들의 사용 빈도수 정보를 관리

3.9.7 재기회(second chance) 페이지 교체 알고리즘

-
- 페이지가 참조되면 R 비트를 조사

3.10 구역성(locality)

- 프로그램의 동작 특성은 프로그램 내에서 일부 지역 만을 집중적으로 참조한다.

  ---> 이런 이유 ? 사용자 프로그램이 순차 수행되는 특성을 가짐

- 시간 구역성(temporal locality)

- 오후 2시30분에 날씨가 맑았다면 오후 3시에도 맑을 가능성이 높다

- 공간 구역성(spatial locality)

구역성(locality)
프로그램의 동작은 확률적으로 특정한 시점에 특정한 프로그램 코드를 집중적으로 참조하는 특성을 가짐.

3.11 워킹 셋(Working set)

- 하나의 프로세스가 자주 참조하는 페이지들의 집합
- working set은 계속적으로 주기억 장치에 유지, 관리시켜주는 것이 좋다.

  ---> 이유는 ?

working set
프로세스가 실행될 때 일정한 시간 동안 자주 참조하는 페이지들의 집합(window size, W)을 의미한다. 페이지 교체를 효율적으로 관리하기 위해 사용되며, 운영체제의 메모리 관리에서 중요한 역할을 수행

3.12 요구 페이징(demand paging)

- 프로그램 실행시 페이지가 명백히 참조될 때에만 보조기억장치에서 주기억장치로 이송

요구불 페이징(demanding paging)
수행중인 프로그램에서 필요한 데이터나 코드(페이지)를 실제 필요한 시점에 주 기억장치로 읽어들이는 것
(장점) 주기억장치의 효율적인 사용이 가능(비용 절감).

3.13 예상 페이징(Anticipatory Paging)

- prepaging

예상 페이징(Anticipatory Paging, Prepaging)
미리 예측하여 프로세스가 필요로 할 페이지를 주기억장치에 로드하는 방식의 가상 메모리 관리 기법
(장점) CPU가 필요한 페이지를 사용하고자 할 경우 입출력 등의 부담이 없이 바로 사용이 가능(성능 우수).