**OS 기말고사 정리**

Chapter 8. Memory Management Strategies

**메모리 관리의 필요성**

1. 프로그래밍을 할 때 쉽게 메모리를 사용하기 위해

2. 효율적으로 자원 할당을 하기 위해

3. Memory protection

**요구사항**

1. Protection: 메모리 보호

2. Fast translation: 주소 참조 변환을 빠르게

3. Fast Context switching

**Virtual Memory**

어플리케이션 프로세스가 사용하는 주소와 실제 물리적인 주소가 다르다.

* Execution time (실행할 때 메모리를 바꿈) 사용

실행하려는 프로세스 크기보다 작을 때 관리할 수 있다. → 자원 할당

Dynamic Linking: 라이브러리로 jump했다가 돌아옴.

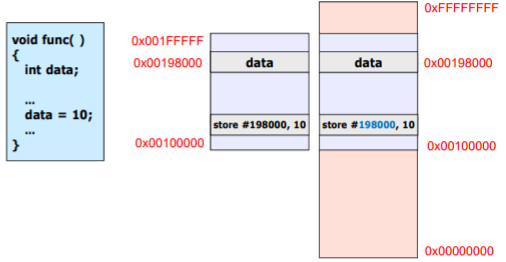
Static Linking: exe파일 안에 라이브러리 코드 포함.

* 같은 명령어가 여러 개 있을 경우 오버헤드는 크지만 메모리 관리는 Dynamic이 효율적.

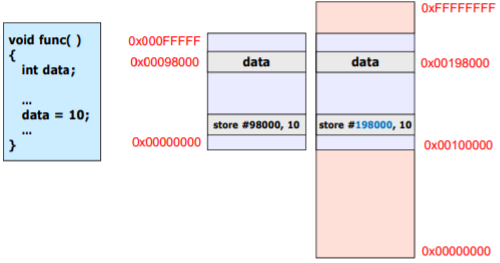
Dynamic Loading: 파일을 필요할 때 메모리에 올림. 현재 사용X

Overlay: 두 개의 파일을 메모리에 번갈아 올림. 현재 사용X

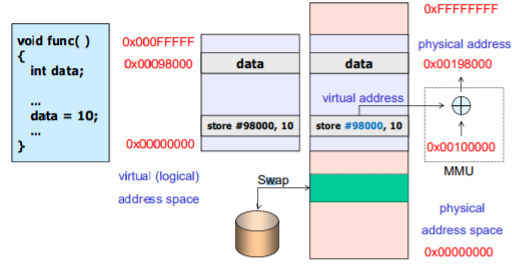
**메모리 번지를 결정하는 시점**

1. Compile time

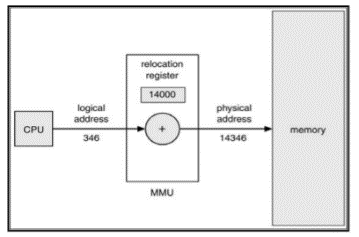
프로그램에서 정해진 주소의 물리적 메모리에 쓰여짐. 즉 프로그램 내의 주소와 물리적 메모리의 주소가 동일하다. OS가 없는 아두이노 프로그래밍 등에서 사용.

2. Load time

프로그램 내부와 물리적 메모리의 주소를 다르게 미리 변환한다. 어디에 로딩되냐에 따라 상대적으로 다르다. 로딩시간이 오래 걸려서 잘 안 쓰인다.



3. **Execution time**

실행할 때 메모리를 바꾼다. 현재 사용하는 방법. 오퍼레이션을 수행하는 하드웨어(MMU)가 필요하다.

**Contiguous Allocation**

Local address가 연속적이면, physical address도 연속적으로 배치한다.

주소 값을 더해주기만 하면 되기 때문에 MMU가 간단하다. 그러나 현재는 사용X

|  |
| --- |
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |

1. 3번에 있는 프로세스 종료 → Hole 생성

2. 3번보다 큰 공간이 필요한 프로세스 A 접근 → 수행 불가

3. 5번 종료 → Hole 생성

4. 역시 프로세스 A 수행 불가

* 메모리가 놀고 있음에도 불구하고 수행하지 못함.
* 메모리 낭비! (External Fragmentation 발생)

**Compaction: 메모리 낭비 해결 방안**

1. First-fit: 들어갈 수 있는 제일 처음 공간에 저장

2. Best-fit: 남은 공간을 적게 만들기 위해 여기저기 비교해서 가장 맞는 공간에 저장

3. Worst-fit: 그냥 가장 많이 남는 공간에 저장. 자투리가 많이 생김

**Fragmentation**

조각조각 Hole이 생겨서 메모리를 낭비하는 현상.

External Fragmentation: 충분한 공간이 존재하지만, 작은 조각들로 분산되어 있을 때 발생.

Internal Fragmentation: 메모리를 먼저 분할하고 할당하였을 때,

할당된 공간과 요구된 공간의 차이로 발생.

\* 낭비되는 메모리는 Internal < External

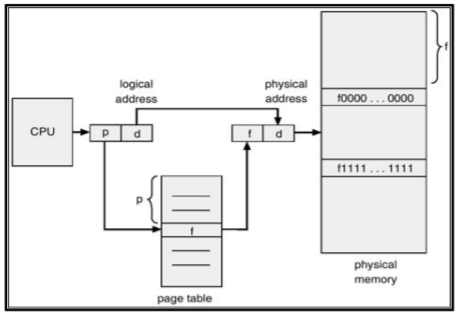
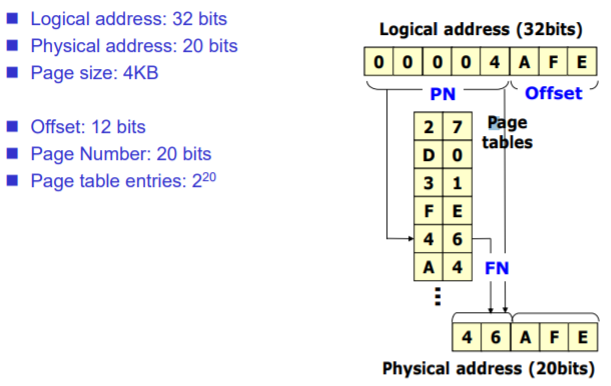
시험 출제

**Paging**

동일한 크기로 메모리를 자르고 불연속적으로 배치한다. External fragmentation 발생X

할당된 크기에 맞지 않는 프로세스가 존재하기 때문에 Internal fragmentation 발생!

\* 메모리를 100% 가까이 다 쓸 수 있는 최적의 방법. 현재 사용 중

**Frame**: physical memory를 고정된 크기로 나눈 블록. 보통 크기는 2의 제곱.

**Page**: logical memory를 프레임과 같은 크기로 나눈 블록

**Page table**: page number와 frame number를 매핑. 프로세스마다 하나씩 존재.

**Page number (p)**: page table에 접근하기 위한 인덱스

**Page offset (d)**: page table에 존재하는 **frame number (f)**와 결합하여 실제 메모리 주소를 만듦.

**TLB (translation look-aside buffer)**

자주 쓰이는 것들을 저장해 놓은 소형 하드웨어 캐시. 더 빨리 처리 가능.

Cache hit: cache에 접근해서 원하는 정보 발견 → 많이 발생 시 hot

Cache miss: cache에서 원하는 정보 발견 못함 → 많이 발생 시 cold

**Memory Protection: Valid or Invalid**

Valid (v): 실제 존재하는 번호, 참조 가능

Invalid (i): 실제 존재하지 않는 번호 또는 하드디스크에 존재, 참조 불가능

\*메인 메모리의 테이블 외에도 TLB에도 존재하나, 자주 쓰이기 때문에 대부분 v이다.

**Page Table 구조**

Hierarchical Paging: 페이지 테이블을 여러 개의 작은 조각으로 나누는 방법

→ two-level paging scheme: 페이지 테이블 자체가 다시 페이징되는 방법.

Hashed Page Tables: 연결 리스트를 가지고 있어서 일치 할 때까지 검색하는 방법.

Inverted Page Tables: 프레임마다 한 항목씩 할당하고, 위치, 해당 프로세스 정보 포함하는 방법.

**Segmentation**

Contiguous allocation의 세그먼트 단위 → external fragmentation 발생

|  |  |
| --- | --- |
| Segment-number | offset |

\*인강 듣기