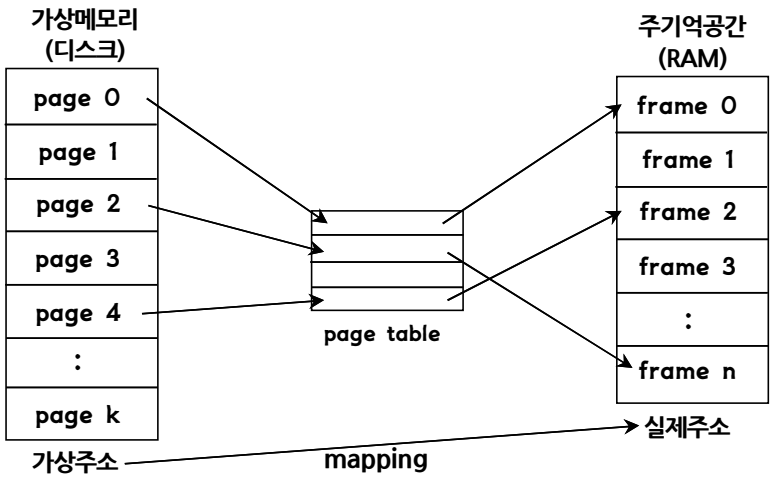
12. 가상메모리(virtual memory)

가상메모리는 하나의 프로세스 전체가 메모리에 적재되지 않아도 실행 가능한 기법이다.
예를 들어, 'Excel'이 전부 메모리에 적재되지 않아도 우리는 이를 사용할 수 있다.

1. 가상메모리 개요



① 페이지와 프레임은 기억공간을 일정한 크기로 나누어 관리하는 단위이다.

페이지(page)	가상메모리를 일정한 같은 크기로 나눈 블록이다.
프레임(frame)	주기억공간(RAM)을 일정한 같은 크기로 나눈 블록이다.(페이지 프레임)

② CPU가 생성하는 주소를 논리주소라 하고, 주기억공간의 주소를 '물리주소'라 한다.

논리주소	실행중인 프로세스가 조회하는 가상메모리 주소이며, '**가상주소**'라고도 한다.
물리주소	주기억공간(RAM)의 주소이며, '**실제주소**'라고도 한다.

③ 프로세스는 CPU가 생성하는 논리주소를 참조하고, 프로세스가 수행되려면 필요한 페이지가 주기억공간에 적재되어야 한다. 논리주소는 물리주소로 'mapping'되어야 한다.

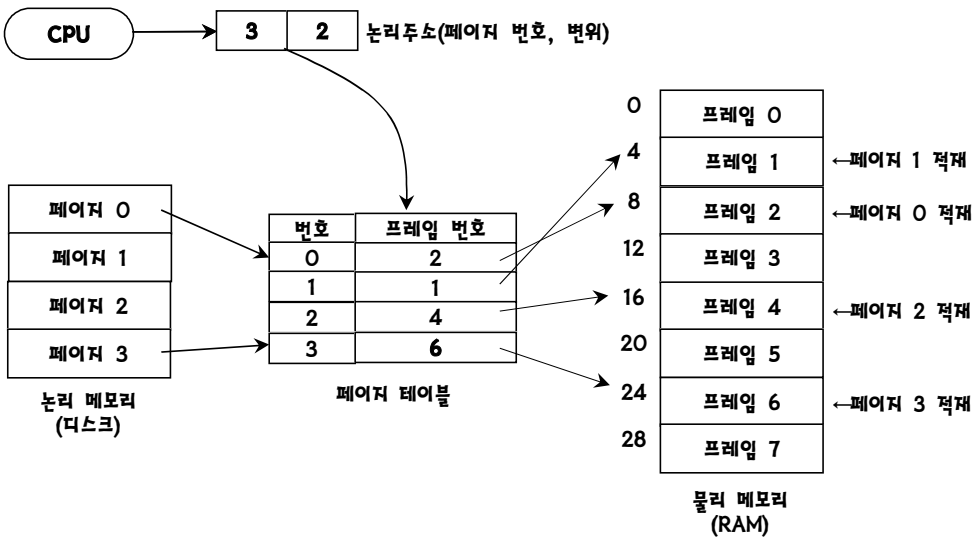
④ 가상메모리에 연속되어 있는 페이지들은 주기억장치에 사상될 때 반드시 연속적으로 사상될 필요는 없다. → 각 페이지는 임의의 페이지 프레임에 적재될 수 있다.

⑤ 동적주소변환(DAT; dynamic address translation)은 프로세스가 수행될 때 가상주소를 실제주소로 변환하는 기법이다.

탐구

페이징 기법의 가상메모리

- 다음은 페이징 기법으로 가상기억장치 구현한 것이다.
- **CPU가 논리주소** = (페이지 번호, 변위) = **(3, 2)**를 생성한 경우이다.



- 주어진 그림은 페이지와 프레임 크기가 각각 4byte인 경우의 예이다.
- 물리 메모리의 각 프레임 앞에 있는 수는 각 프레임의 시작 물리주소이다.
- 페이지 0은 프레임 2에 적재되어 있고, 페이지 3은 프레임 6에 적재되어 있다.

// CPU가 생성한 논리주소에 대한 물리주소는 다음처럼 구할 수 있다.

- 논리주소 = (페이지 번호, 변위) = **(3, 2)**
- 물리주소 = 프레임 번호 × 페이지 크기 + 변위 = 6 × 4 + 2 = 24 + 2 = **26**

  ↓
- 3번 페이지의 변위 2는 물리주소 26이다. 즉, 프로세스는 **26번지를 참조**한다.

[예제] 논리주소 = (페이지 번호, 변위) = (0, 3)에 대응하는 물리주소는?

물리주소 = 프레임 번호 × 페이지 크기 + 변위

= 2 × 4 + 3 = 8 + 3 = **11**

↳, 0번 페이지는 프레임 2에 적재되어 있다.

탐구

페이징(paging)과 세그먼테이션(segmentation)

페이지 기반	① **페이지 기반** 가상메모리 시스템에서는 **외부단편화는 발생되지 않는다.** • 이유는, 페이지 크기와 프레임 크기를 같이 분할하고 • 모든 빈 프레임을 프로세스에게 할당할 수 있기 때문이다. ② 페이지 기반 가상메모리 시스템에서는 **내부단편화는 발생**된다. • 이유는, 마지막 페이지 크기가 프레임 크기와 일치하지 않을 수 있으므로 • 내부단편화는 각 프로세스의 마지막 페이지에서 소량 발생될 수 있다. • 평균적으로 **각 프로세스 당 반 페이지 정도의 내부단편화가 발생**될 수 있다.

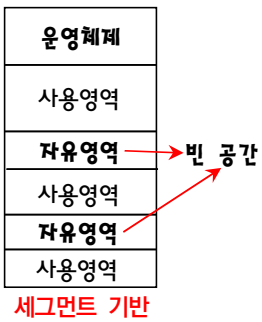
세그먼테이션 기법	① **세그먼테이션** 기법에서는 **외부단편화가 발생할 수 있다.** ② 세그먼테이션 기법에서는 메모리를 **동적으로 분할**한다. • 동적 분할은 반입되는 세그먼트 크기에 맞게 메모리를 분할한다는 것이다. • 세그먼테이션 기법에서 분할된 메모리를 **세그먼트**라고 한다. • 분할된 **세그먼트** 영역들은 서로 크기가 다르다. • 운행 중, 분할된 **세그먼트** 영역들은 **자유영역(공백)**으로 되돌려 질 수 있다. • **자유영역**들이 너무 작을 때, **외부단편화**가 발생할 수 있다.**(적재 불가)** ③ 세그먼테이션 기법에서 외부단편화 해결 방법은 • 자유영역들에 대해 **압축**을 수행하여, 더 큰 공간을 만들면 된다. • 압축은 비어있는 **자유영역을 한 곳으로 합치는 작업**이다. ④ **세그먼트**의 최대 크기는 정해져 있는 것은 아니다.

↓

↓ **핵심** 내용을 요약하면

↓

	내부단편화	**외부단편화**
페이지 기반 가상메모리	**소량** 발생	발생 안함
세그먼트 기반 가상메모리	발생 안함	**발생 가능**



2. 페이지와 페이지 프레임

① 시스템에서 페이지와 페이지 프레임의 크기는 통상적으로 **같은 크기**로 나눈다.
→ 크기가 서로 다르면 엄청난 내부단편화가 발생되어 메모리 낭비가 심하게 된다.
② 하나의 페이지 크기는 통상적으로 512~8,192byte이다. 즉, **2의 멱승**(2^k) 크기이다.
→ 하나의 페이지 크기는 IBM370과 AMD64에서는 4KB, ia64에서는 8KB이었다.

◈ **페이지 크기**

	작은 경우	큰 경우
페이지 테이블 크기	커진다(반비례)	작아진다
페이지 테이블을 위한 공간	많이 필요	적게 필요
프로세스 구역성	좋다(메모리 사용 효율적)	나쁘다(불필요한 정보 포함)
내부단편화 크기	작게 발생	크게 발생(메모리 낭비)
프로세스의 작업 집합 확보	쉽다	어렵다

- **디스크 장치 관점**(디스크 접근시간이 많이 필요함)에서는 **페이지 크기가 클수록** 프로그램 실행 중에 입출력 전송 횟수가 감소되므로 효과적이다.
- 내부단편화가 발생되는 이유는 프레임 크기가 반드시 페이지 크기의 정수배가 아니므로
- 페이징 기법에서는 '**외부단편화**'는 발생되지 않는다.

◈ **프레임 크기가 반드시 페이지 크기의 정수배가 아닌 이유**
- 이유는 마지막 페이지 크기 때문이다.
- 임의 크기의 프로그램을 일정 크기로 분할하면, **마지막 페이지 크기는 다양**하게 된다.
- 마지막 페이지 크기는 프레임 크기에 근접할 수도 있고, 거의 빈 상태가 될 수도 있다.
- 결론은, 프로세스 당 평균 1/2 크기의 내부단편화가 발생될 수 있다.

3. 작업 집합(working set)

① 가상메모리에서 프로세스가 일정시간 동안 참조하는 '**페이지 집합**'이다.
② 프로세스에게 최소한의 프레임을 배정하여 **'스래싱'을 방지**할 수 있는 기법이다.
③ 작업 집합 이론은 페이지 교체 전략에도 이용할 수 있는 기법이다.
④ 작업 집합 이론은 '**구역성**'을 토대로 하고 있다.

4. 구역성(locality)

프로세스는 메모리의 정보를 균일하게 접근하지 않고, 어느 순간에는 특정 부분을 집중적으로 참조한다.
구역성은 캐시메모리, 페이지 크기, working set 이론의 근거가 된다.

시간 구역성 (temporal locality)	• 가까운 시기에 계속 참조될 가능성이 높은 것이다. • 한 번 사용된 데이터는 또 다시 사용될 가능성이 많다. • 예 : 카운팅과 집계에 사용되는 변수, 순환, 서브루틴, **스택**
공간 구역성 (spatial locality)	• 주변의 데이터가 계속 참조될 가능성이 높은 것이다. • CPU가 필요로 하는 다음 데이터는 바로 인접해 있을 경우가 많다. • 예 : 순차코드 실행, 관련된 변수(변수 선언 경향), **배열**

다음 비주얼베이직 코드를 이용하여 구역성에 대해 살펴본다.

```
Private Sub Form_Activate()
  Dim a, b                      '관련된 변수(프로그래머의 변수 선언 경향) → 공간 구역성
  Dim 합, 평균, 인원수          '카운팅과 집계에 사용되는 변수 → 시간 구역성
  Dim 점수(), 석차() As Variant
  점수() = Array(85, 90, 98, 90, 90, 60)          '배열
  석차() = Array(1, 1, 1, 1, 1, 1)                '배열
  For a = 0 To 5
    합 = 합 + 점수(a)                             '집계 → 시간 구역성
    인원수 = 인원수 + 1                           '카운팅 → 시간 구역성
    For b = 0 To 5
      If 점수(a) < 점수(b) Then
        석차(a) = 석차(a) + 1
      End If
    Next
  Next
  평균 = 합 / 인원수
  Print "합 : "; 합, "평균 : "; 평균              '합 : 513, 평균 : 85.5
  Print "석차 : ";
  For a = 0 To 5 : Print 석차(a); : Next          '석차 : 5 2 1 2 2 6
End Sub
```

5. 페이지 부재(page fault)

프로세스가 현재 필요로 하는 페이지가 주기억장치에 없는 경우이다.

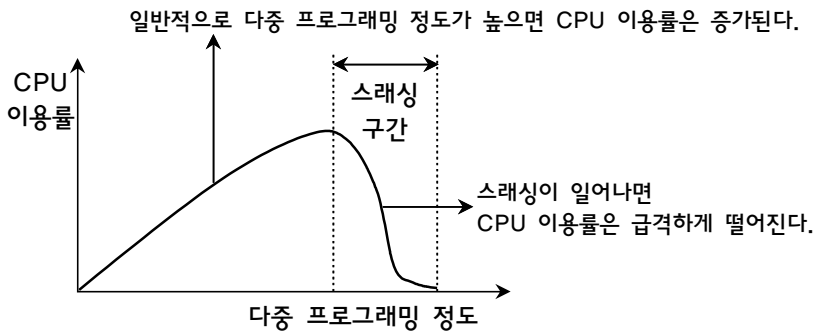
페이지 부재 인터럽트 발생
↓
운영체제가 감지
↓
요구하는 페이지 반입
↓
다시 실행 계속

• 페이지부재율 = $\frac{\text{요구된페이지가없었던횟수}}{\text{프로세스가페이지를요구한횟수}}$

// 가상메모리에서 페이지 부재률은 중요한 요인이다.
• 낮은 경우 : 접근시간이 적어 프로세스 수행 효율이 좋음
• 높은 경우 : 접근시간이 높아 프로세스 수행 효율이 나쁨

6. 스래싱(thrashing)

가상메모리에서 **페이지 부재가 빈번하게 발생**되면 프로세스 수행에 소요되는 시간보다 페이지 교체에 소요되는 시간이 더 커질 수 있다. 이때 **'스래싱'**이 발생했다고 말한다.



스래싱 원인	• 프로세스 실행에 필요한 최소한의 프레임을 갖지 못한 경우 • 다중 프로그래밍 정도가 높아져 프로세스 당 할당되는 프레임 수가 감소하는 경우
스래싱 결과	• 페이지 부재가 발생되면, 페이지 교체 시간 동안 CPU는 공전한다. • 빈번한 페이지 부재는 CPU 사용률을 급격하게 떨어지게 만든다. • 시스템 성능이 급격하게 저하된다.

7. 가상메모리에서 페이지 교체(page replacement)

가상메모리에서 프로세스가 어떤 페이지를 요구할 때 해당 페이지가 주기억장치의 프레임에 없으면 주기억장치에 있는 어느 페이지를 내 보낼 것인가? 를 다루는 것이다.

(1) 최적 페이지 교체(OPT; optimal page replacement) 알고리즘

- 앞으로 **가장 오랫동안 사용되지 않을 페이지를 교체**한다.
- 프로세스가 앞으로 어느 페이지를 필요할 것인지 예측이 불가능하므로 실제 구현은 불가능하다.
- **이론**상 연구 목적을 위해 주로 사용한다.

(2) FIFO(First in First out) 페이지 교체 알고리즘

- 메모리에 들어온 **순서대로 페이지를 교체**한다. → **가장 오래된 페이지 교체**
- 페이지가 적재 시간을 페이지마다 기록하거나 적재된 페이지 순으로 큐를 만들어 두어야 한다.
- 현재 사용 중인 페이지도 오래 동안 머물러 있었다는 이유로 교체될 수 있다.(단점)
- 일명, FCFS(First Come First Serviced)라고도 한다.

〈밸러디 모순(Belady's anomaly)〉

- 어떤 프로세스에게 페이지 프레임을 많이 할당할수록 **'페이지 부재률'은 감소**되어야 하는데,
- 오히려, 거꾸로 **'페이지 부재률'이 증가**하는 비상식적인 현상을 말한다.
- **FIFO** 페이지 교체 알고리즘은 이러한 **비상식적인 현상**이 발생될 수 있다.

(3) SCR(Second Chance Replacement; 2차 기회 교체) 알고리즘

- 참조(조회)비트를 이용하여 FIFO의 단점을 보완한 기법이다.

가장 오래된 페이지의 참조비트가 **0**인 경우	페이지를 교체한다.
가장 오래된 페이지의 참조비트가 **1**인 경우	페이지를 교체하지 않는다**(한 번 더 기회를 줌)**

- SCR 알고리즘은 **원형큐**를 이용하여 구현할 수 있다.

(4) LFU(Least Frequently Used) 페이지 교체 알고리즘

- **사용 횟수(빈도)**가 가장 적은 페이지를 교체한다.
- 가장 최근에 메모리로 들어온 페이지는 최소 빈도수를 가지므로 곧 바로 교체될 수 있다.
- 막 들어온 페이지의 빈도수는 한번이지만 앞으로 사용하기 위한 것인데 교체될 수 있다.**(문제점)**

(5) LRU(Least Recently Used) 페이지 교체 알고리즘

- LRU 알고리즘은 **가장 오랫동안(최근에)** 사용되지 않고 있는 페이지를 교체한다.
- LRU 알고리즘은 각 페이지의 마지막 사용시간을 보관한다.**(시간적 집약성에 기반)**
- LRU는 **과거 시간에 대한 최적 교체 정책**에 속한다.
- LRU 구현은 하드웨어 지원이 필요하다. 소프트웨어 처리는 속도가 매우 느려진다.
- LRU 알고리즘은 밸러디 모순을 발생시키지 않는다.
  → 밸러디 모순을 발생시키지 않는 페이지 교체 기법을 '스택 알고리즘'라 한다.
- LRU 알고리즘은 **스택** 또는 **계수기**(참조시간 기록)를 이용하여 구현할 수 있다.
- 스택 사용 구현(최근 사용된 페이지는 Stack 꼭대기에 위치)
  스택 꼭대기 : 가장 최근에 사용된 페이지 번호 위치
  스택 바닥 : 가장 오래 전에 사용된 페이지 번호 위치
- 스택구조는 **이중연결리스트로 구현**하는 것이 좋다,

(6) NUR(Not Used Recently) 페이지 교체 알고리즘

- 최근에 사용되지 않은 페이지를 교체한다.
- 2개의 비트를 추가하여 **LRU를 보완한 기법**이다.

교체순서	참조비트	수정비트	설명
①	0	0	페이지가 참조되지 않았고, 수정도 되지 않았음
②	0	1	페이지가 참조되지 않았고, 수정은 되었음
③	1	0	페이지가 참조되었고, 수정은 되지 않았음
④	1	1	페이지가 참조되었고, 수정도 되었음

[예제] 프로세스에게 3개의 페이지 프레임이 할당된 경우(초기에는 빈 상태)

	페이지 요구	1	2	3	1	5	2	1	3	4
FIFO	페이지 프레임	1	1	1	1	5	5	5	5	5
			2	2	2	2	2	1	1	1
				3	3	3	3	3	3	4
	페이지 부재	f	f	f		f		f		f
LRU	페이지 프레임	1	1	1	1	1	1	1	1	1
			2	2	2	5	5	5	3	3
				3	3	3	2	2	2	4
	페이지 부재	f	f	f		f	f		f	f

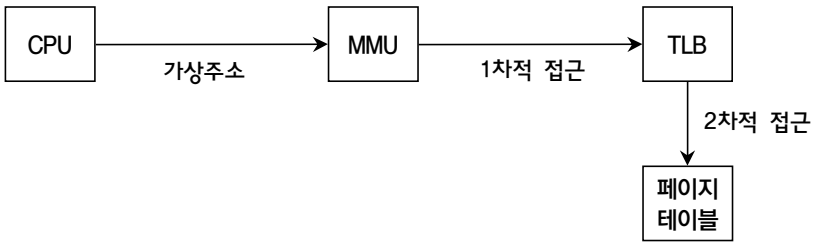
- 하단에 표시된 f는 페이지 부재가 발생된 것을 의미한다.

8. MMU와 TLB

① 메모리 관리 장치(Memory Management Unit, MMU)

- MMU는 CPU가 메모리에 접근하는 것을 관리하는 컴퓨터 하드웨어 장치이다.
- 가상메모리 시스템에서 MMU는 **가상주소**를 **물리주소로 변환**한다.
- MMU는 메모리 보호, 캐시 관리, 버스 중재 등의 역할도 담당한다.

〈가상메모리 시스템에서 가상주소를 물리주소로 변환〉



- 가상메모리 시스템에서 CPU가 가상주소를 MMU에 넘겨주면
  → MMU는 가상주소와 물리주소 사이의 변환을 위해 우선적으로 TLB를 참조한다.
  → TLB는 변환색인버퍼(Translation Lookaside Buffer)라는 고속의 보조기억장치이다.
- 가상메모리 시스템에서 TLB에 원하는 주소 변환 정보가 없을 때는
  → 주소 변환을 위해 MMU는 페이지 테이블(page table)을 참조한다.

// 다시 정리하면,

- **MMU**는 **1차적**으로 **TLB**에 접근하여, 원하는 페이지가 존재하는지 탐색하고
- **TLB**에 원하는 페이지가 존재하지 않으면, **2차적**으로 페이지 테이블을 참조한다.

② 변환색인버퍼(Translation Lookaside Buffer, TLB)

- TLB는 고속의 보조기억장치이다.**(주소 변환 캐시)**
- TLB는 가상주소를 물리주소로 변환하는 속도를 높이기 위해 사용되는 캐시이다.
  → 명령어가 실행되려면, CPU가 생성하는 가상주소가 물리주소로 변환되어야 한다.
  → TLB는 주소 변환 속도를 높이기 위해 사용한다.
- TLB에는 최근에 발생된 가상주소와 물리주소의 변환 테이블이 저장되어 있다.
- TLB는 단지 가상주소와 물리주소를 1:1로 저장해 둔 하나의 기억장치이다.
- TLB는 CPU와 CPU 캐시, CPU 캐시와 메모리 사이 등에서 주소 변환에 사용할 수 있다.
- 현재, 모든 데스크탑이나 서버용 프로세서는 하나 이상의 TLB를 가지고 있다.

기출문제 분석

1. ㉠에 들어갈 용어로 옳은 것은? [2018년 계리직]

주기억장치의 물리적 크기의 한계를 해결하기 위한 기법으로 주기억장치의 크기에 상관없이 프로그램이 메모리의 주소를 논리적 관점에서 참조할 수 있도록 하는 것을 (㉠)라고 한다.

---

① 레지스터(register)
② 정적메모리(static memory)
③ 가상메모리(virtual memory)
④ 플래시메모리(flash memory)

♤ **가상메모리**

• 가상메모리는 주기억장치의 물리적 크기의 한계를 해결하기 위한 기법이다.
• 가상메모리는 하나의 프로세스 전체가 메모리에 적재되지 않아도 실행 가능한 기법이다.

정답 : ③

2. 가상기억장치(virtual memory)에 대한 설명으로 가장 옳은 것은? [2015년 서울 9급]

① 가상기억장치를 사용하면 메모리 단편화가 발생하지 않는다.
② 가상기억장치는 실기억장치로의 주소변환 기법이 필요하다.
③ 가상기억장치의 참조는 실기억장치의 참조보다 빠르다.
④ 페이징 기법은 가변적 크기의 페이지 공간을 사용한다.

♤ **가상기억장치(virtual memory)**

---

① 가상기억장치를 사용하면 메모리 단편화가 발생하지 않는다.**(×)**
→ 페이징 기법에서 내부단편화가 소량 발생될 수도 있다.
→ 이유는 프레임 크기가 반드시 페이지 크기의 정수배가 아니기 때문이다.
② 가상기억장치는 실기억장치로의 주소변환 기법이 필요하다.(○)
→ 가상주소를 실제주소로 변환하는 것을 Mapping이라 한다.
③ 가상기억장치의 참조는 실기억장치의 참조보다 빠르다.**(×)**
→ 가상기억장치의 참조는 실기억장치의 참조보다 느리다.
④ 페이징 기법은 가변적 크기의 페이지 공간을 사용한다.**(×)**
→ 페이징 기법은 블록의 크기를 동일하게 분할한다.

정답 : ②

3. 가상메모리(virtual memory) 관리 기법에 대한 설명으로 옳지 않은 것은? [2017년 국회 9급]

① 프로그램의 크기가 실제 메모리의 크기보다 커도 실행이 가능하다.
② 페이지 방식은 페이지 사상 작업으로 인하여 비용이 증가하고 수행속도가 감소한다.
③ 페이지 방식은 페이지를 작게 설계하여 내부단편화를 줄일 수 있다.
④ 세그먼트 방식은 동적 메모리 할당 방법으로 외부단편화가 발생하지 않는다.
⑤ 세그먼트 방식은 세그먼트 번호와 변위(offset)로 구성하는 가상주소를 만든다.

♤ **가상메모리 – 단편화**

	내부단편화	외부단편화
페이지 기반 가상메모리	**소량** 발생	발생 안함
세그먼트 기반 가상메모리	발생 안함	**발생 가능**

- **세그먼테이션** 기법에서는 **외부단편화가 발생할 수 있다.**
- 분할된 **세그먼트** 영역들은 서로 크기가 다르다.
- 운행 중, 분할된 **세그먼트** 영역들은 **자유영역(공백)**으로 되돌려 질 수 있다.
- **자유영역**들이 너무 작을 때, **외부단편화**가 발생할 수 있다.**(적재 불가)**

정답 : ④

4. 가상기억장치 기술에 대한 설명으로 옳지 않은 것은? [2017년 법무 9급]

① 가상주소(virtual address)에서 물리주소(physical address)로의 주소 변환(address translation)이 이루어진다.
② 가상주소와 물리주소의 비트 수가 서로 다를 수 있다.
③ 다중 프로그래밍 정도(degree of multiprogramming)가 높아짐에 따라 CPU 이용률(utilization)은 계속 높아진다.
④ 서로 다른 프로세스가 동일한 물리 기억장치 영역을 공유할 수 있다.

♤ **가상기억장치 – 스래싱**

- 다중 프로그래밍 정도가 높아짐에 따라 CPU 이용률이 계속 높아지는 것은 아니다.
- 다중 프로그래밍 정도가 높아짐에 따라 스래싱이 발생되면 CPU 이용률은 급격히 떨어진다.

정답 : ③

5. 다중 프로그래밍 환경에서 연속 메모리 할당 방법에 대한 설명으로 옳지 않은 것은? [2022년 지방 9급]

① 가변분할 메모리 할당은 프로세스의 크기에 따라 메모리를 나누는 것으로 단편화 문제가 발생하지 않는다.
② 가변분할 메모리 할당의 메모리 배치방법으로는 최초 적합, 최적 적합, 최악 적합 방법이 있다.
③ 고정분할 메모리 할당은 프로세스의 크기와 상관없이 메모리를 같은 크기로 나누는 것이다.
④ 고정분할 메모리 할당에서는 쓸모없는 공간으로 인해 메모리 낭비가 발생할 수 있다.

♤ **다중 프로그래밍 환경 – 연속 메모리 할당**

• **가변분할** 메모리 할당은 프로세스의 크기에 따라 메모리를 나누는 것으로 **단편화 문제가 발생하지 않는다.(×)**
→ **가변분할** 메모리 할당은 **외부단편화 문제가 발생한다.**
→ **세그먼테이션 기법**은 **가변분할** 메모리 할당이다.

// 페이지와 세그먼트

페이지	블록의 크기를 **동일하게 분할**한 경우에 '페이지'라 한다.**(고정분할)**
세그먼트	블록의 크기를 **서로 다르게 분할**한 경우에 '세그먼트'라 한다.**(가변분할)**

세그먼테이션 기법 | (가변분할)	① **세그먼테이션** 기법에서는 **외부단편화가 발생할 수 있다.** ② 세그먼테이션 기법에서는 메모리를 **동적으로 분할**한다. • 동적 분할은 반입되는 세그먼트 크기에 맞게 메모리를 분할한다는 것이다. • 세그먼테이션 기법에서 분할된 메모리를 **세그먼트**라고 한다. • 분할된 **세그먼트** 영역들은 서로 크기가 다르다. • 운행 중, 분할된 **세그먼트** 영역들은 **자유영역(공백)**으로 되돌려 질 수 있다. • **자유영역**들이 너무 작을 때, **외부단편화**가 발생할 수 있다.**(적재 불가)** ③ 세그먼테이션 기법에서 외부단편화 해결 방법은 • 자유영역들에 대해 **압축**을 수행하여, 더 큰 공간을 만들면 된다. • 압축은 비어있는 **자유영역을 한 곳으로 합치는 작업**이다. ④ **세그먼트**의 최대 크기는 정해져 있는 것은 아니다.

정답 : ①

6. 스레싱(thrashing)에 대한 설명으로 옳지 않은 것은? [2018년 국가 9급]

① 프로세스의 작업 집합(working set)이 새로운 작업 집합으로 전이 시 페이지 부재율이 높아질 수 있다.

② 작업 집합 기법과 페이지 부재 빈도(page fault frequency) 기법은 한 프로세스를 중단(suspend)시킴으로써 다른 프로세스들의 스레싱을 감소시킬 수 있다.

③ 각 프로세스에 설정된 작업 집합 크기와 페이지 프레임 수가 매우 큰 경우 다중 프로그래밍 정도(degree of multiprogramming)를 증가시킨다.

④ 페이지 부재 빈도 기법은 프로세스의 할당받은 현재 페이지 프레임 수가 설정한 페이지 부재율의 하한보다 낮아지면 보유한 프레임 수를 감소시킨다.

♤ **스레싱**

---

① 프로세스의 작업 집합(working set)이 새로운 작업 집합으로 전이 시 페이지 부재율이 높아질 수 있다.(○)
→ 작업 집합 크기를 크게 하면, 페이지 부재율은 낮아질 것이고**(너무나 당연)**
→ 작업 집합 크기를 작게 하면, 페이지 부재율은 높아질 것이고**(너무나 당연)**

② 작업 집합 기법과 페이지 부재 빈도(page fault frequency) 기법은 한 프로세스를 중단(suspend)시킴으로써 다른 프로세스들의 스레싱을 감소시킬 수 있다.(○)
→ 한 프로세스를 중단시키면 다른 프로세스들에게 프레임을 더 많이 할당할 수 있으므로
→ 그런데, 한 프로세스를 중단시킨다고 얼마나 많은 프레임을 더 할당할 수 있을까요?
→ 아무튼, 문제가 그저 그렇습니다.

③ 각 프로세스에 설정된 작업 집합 크기와 페이지 프레임 수가 매우 큰 경우 다중 프로그래밍 정도(degree of multiprogramming)를 **증가**시킨다.**(×)**
→ 프로세스에게 설정된 프레임 수가 클수록 다중 프로그래밍 정도는 **감소**된다.

④ 페이지 부재 빈도 기법은 프로세스의 할당받은 현재 페이지 프레임 수가 설정한 페이지 부재률의 하한보다 낮아지면 보유한 프레임 수를 감소시킨다.(○)
→ 부재률의 하한보다 낮아진다는 것은 프로세스에게 할당한 프레임 수가 많다는 것이므로

◈ **페이지 부재 빈도(PFF, Page Fault Frequency)에 의한 스래싱 발생 조절**

- 스래싱 발생은 페이지 부재율이 높은 것이므로, 목적은 부재률을 적절히 조절하는 것이다.
- 부재률이 높을수록 더 많은 프로세스가 더 많은 프레임을 필요로 하는 것이고
- 부재률이 낮을수록 더 많은 프로세스가 너무 많은 프레임을 가지고 있는 것이다.
- 해서, 페이지 부재률 상하한을 정해 놓고
  페이지 부재률이 상한보다 높으면 그 프로세스에게는 프레임을 더 할당해 주고
  페이지 부재률이 하한보다 낮으면 그 프로세스에게는 프레임을 줄인다.

정답 : ③

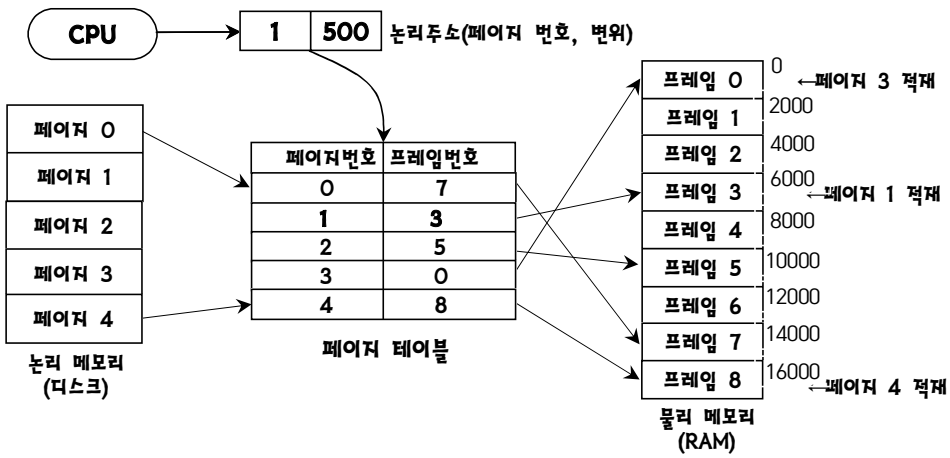
7. 페이지 크기가 2,000byte인 페이징 시스템에서 페이지 테이블이 다음과 같을 때 논리주소에 대한 물리주소가 옳게 짝지어진 것은?(단, 논리주소와 물리주소는 각각 0에서 시작되고, 1byte 단위로 주소가 부여된다) [2017년 국가 9급]

페이지번호(논리)	프레임번호(물리)
0	7
1	3
2	5
3	0
4	8

	논리주소	물리주소
①	4,300	2,300
②	3,600	4,600
③	2,500	6,500
④	900	7,900

♤ 페이징 기법의 가상메모리

• 논리주소 2,500 = (페이지번호, 변위) = (1, 500)을 의미, 페이지 크기가 2,000이므로



• 주어진 그림은 페이지와 프레임 크기가 각각 2,000 byte인 경우이다.
→ 페이지 1은 프레임 3에 적재되어 있다.

// CPU가 생성한 논리주소에 대한 물리주소는 다음처럼 구할 수 있다.

• 논리주소 = 2,500 = (페이지 번호, 변위) = (1, 500)
• 물리주소 = 프레임 번호 × 페이지 크기 + 변위 = 3 × 2,000 + 500 = **6,500**

정답 : ③

8. 메모리 크기가 200KB인 시스템에서 요구 페이징(demand paging)으로 가상메모리(virtual memory)를 구현한다고 하자. 페이지 크기가 2KB이고 페이지 테이블(page table)의 각 항목이 3바이트라고 하면, 25KB 크기의 프로세스를 위한 최소 페이지 테이블의 크기는 어떻게 되는가? [2016년 서울 9급]

① 25바이트 ② 39바이트
③ 60바이트 ④ 75바이트

♤ **최소 페이지 테이블 크기**

• 페이지 수 = $\frac{\text{프로세스 크기}}{\text{페이지 크기}} = \frac{25KB}{2KB}$ = 12.5 → 페이지 수는 13개로 분할된다.
• 최소 페이지 테이블의 크기 = 13 × 3 = **39**(바이트)

정답 : ②

9. 가상메모리(virtual memory)를 효과적으로 제공하기 위해 Core i7과 같은 프로세서 내부에 있는 장치는 무엇인가? [2016년 서울 9급]

① TLB(translation lookaside buffer) ② 캐시(cache)
③ 페이지 테이블(page table) ④ 스왑 스페이스(swap space)

♤ **TLB(Translation Lookaside Buffer) – 변환색인버퍼**

• TLB는 고속의 보조기억장치이다.**(주소 변환 캐시)**
• TLB는 최근에 일어난 가상주소와 물리주소의 변환 테이블을 저장한다.
• TLB는 페이지 테이블 또는 세그먼트 테이블의 항목들 중 일부를 미리 저장한다.
• TLB는 가상메모리 시스템에서 주소 변환 속도를 높이기 위해 사용한다.
• TLB는 CPU와 CPU 캐시, CPU 캐시와 메모리 사이 등에서 주소 변환에 사용할 수 있다.

◈ **메모리 관리 장치(Memory Management Unit, MMU)**
• MMU는 CPU가 메모리에 접근하는 것을 관리하는 컴퓨터 하드웨어 장치이다.
• MMU는 가상주소를 물리주소로 변환하며, 캐시 관리, 버스 중재 등의 역할을 담당한다.
• CPU가 가상주소를 MMU에 넘겨주면
→ MMU는 가상주소와 물리주소 사이의 변환을 위해 우선적으로 TLB를 참조한다.
→ TLB에 주소 변환 정보가 없으면, MMU는 페이지 테이블을 참조한다.

정답 : ①

10. 주기억장치의 페이지 교체 기법에 대한 설명으로 가장 옳은 것은? [2018년 서울 9급]

① FIFO(First In First Out)는 가장 오래된 페이지를 교체한다.
② MRU(Most Recently Used)는 최근에 적게 사용된 페이지를 교체한다.
③ LRU(Least Recently Used)는 가장 최근에 사용한 페이지를 교체한다.
④ LFU(Least Frequently Used)는 최근에 사용 빈도가 가장 많은 페이지를 교체한다.

♤ **페이지 교체 기법**

① FIFO(First In First Out)는 가장 오래된 페이지를 교체한다.(○)
② MRU(Most Recently Used)는 최근에 적게 사용된 페이지를 교체한다.**(×)**
→ MRU(Most Recently Used)는 최근에 가장 많이 사용된 페이지를 교체한다.
③ LRU(Least Recently Used)는 가장 최근에 사용한 페이지를 교체한다.**(×)**
→ LRU(Least Recently Used)는 가장 오랫동안 사용되지 않은 페이지를 교체한다.
④ LFU(Least Frequently Used)는 최근에 사용 빈도가 가장 많은 페이지를 교체한다.**(×)**
→ LFU(Least Frequently Used)는 사용 빈도가 가장 적은 페이지를 교체한다.

정답 : ①

11. 운영체제에서 가상메모리의 페이지 교체 기법에 대한 설명으로 가장 옳지 않은 것은? [2019년 서울 9급]

① FIFO 기법에서는 아무리 참조가 많이 된 페이지라도 교체될 수 있다.
② LRU 기법을 위해서는 적재된 페이지들의 참조된 시간 또는 순서에 대한 정보가 필요하다.
③ Second-chance 기법에서는 참조비트가 0인 페이지는 교체되지 않는다.
④ LFU 기법은 많이 참조된 페이지는 앞으로도 참조될 확률이 높을 것이란 판단에 근거한 기법이다.

♤ **SCR(Second Chance Replacement; 2차 기회 교체) 알고리즘**

• SCR은 참조(조회)비트를 이용하여 FIFO의 단점을 보완한 기법이다.
• SCR은 가장 오래된 페이지의 참조비트가 0이면 페이지를 교체한다.
• SCR은 참조비트가 1이면 교체하지 않는다(한 번 더 기회를 줌)
• SCR 알고리즘은 원형큐를 이용하여 구현할 수 있다.

정답 : ③

12. 가상메모리 시스템에서 메모리 부족 시의 페이지 교체 기법(page replacement algorithm)들에 대한 설명으로 옳지 않은 것은? [2018년 국회 9급]

① LRU(Least Recently Used) 기법은 메모리에 적재된 페이지 중 가장 오랫동안 참조되지 않았던 페이지를 교체하는 기법이다.
② LRU 기법은 실제 그 구현 오버헤드가 커서, 일반적으로 오버헤드를 줄인 여러 유형의 LRU 근사 알고리즘(LRU approximation algorithm)들이 사용되는 것이 보통이다.
③ LRU 기법은 물리적 페이지의 개수를 확장했음에도 페이지 폴트가 늘어나는 경우가 발생할 수도 있는데, 이를 Belady's anomaly라 한다.
④ 이론적으로는 최적의 페이지 교체 기법은 메모리에 적재된 페이지들 중에서 앞으로 가장 오랫동안 참조되지 않을 페이지를 교체 하는 것이다.
⑤ FIFO(First In First Out) 기법은 메모리에 적재된 페이지들 중 가장 먼저 메모리에 적재된 페이지를 교체하는 방법이다.

♤ **가상메모리 시스템에서 메모리 부족 시의 페이지 교체 기법**

- LRU 기법은 물리적 페이지의 개수를 확장했음에도 페이지 폴트가 늘어나는 경우가 발생할 수도 있는데, 이를 Belady's anomaly라 한다.**(×)**
  → LRU 기법은 밸러디 모순을 발생시키지 않는다.

정답 : ③

13. 3개의 페이지 프레임으로 구성된 기억장치에서 다음과 같은 순서대로 페이지 요청이 일어날 때, 페이지 교체 알고리즘으로 LFU(Least Frequently Used)를 사용한다면 몇 번의 페이지 부재가 발생하는가? (단, 초기 페이지 프레임은 비어있다고 가정한다) [2014년 국가 9급]

요청된 페이지 번호의 순서 : 2, 3, 1, 2, 1, 2, 4, 2, 1, 3, 2

① 4번 ② 5번 ③ 6번 ④ 7번

♤ LFU(Least Frequently Used) – LFU는 사용 빈도수가 가장 적은 페이지를 교체한다.

	페이지 요청	2	3	1	2	1	2	4	2	1	3	2
LFU	페이지 프레임	2	2	2	2	2	2	2	2	2	2	2
			3	3	3	3	3	4	4	4	3	3
				1	1	1	1	1	1	1	1	1
	페이지 부재	f	f	f				f			f	

- 하단에 표시된 f는 페이지 부재가 발생된 것을 의미한다. → **5번 발생**된다.

정답 : ②

14. FIFO 페이지 교체 알고리즘을 사용하는 가상메모리에서 프로세스 P가 다음과 같은 페이지 번호 순서대로 페이지에 접근할 때, 페이지 부재(page-fault) 발생 횟수는? (단, 프로세스 P가 사용하는 페이지 프레임은 총 4개이고, 빈 상태에서 시작한다) [2019년 국가 9급]

---

1 2 3 4 5 2 1 1 6 7 5

---

① 6회 ② 7회 ③ 8회 ④ 9회

♤ 가상메모리 – FIFO 페이지 교체 알고리즘

	페이지 요구	1	2	3	4	5	2	1	1	6	7	5
FIFO	페이지 프레임	1	1	1	1	5	5	5	5	5	5	5
			2	2	2	2	2	1	1	1	1	1
				3	3	3	3	3	3	6	6	6
					4	4	4	4	4	4	7	7
	페이지 부재	f	f	f	f	f		f		f	f		=> 8회

정답 : ③

15. 3개의 page를 수용할 수 있는 메모리가 있으며, 현재 완전히 비어 있다. 어느 프로그램이 <보기>와 같이 page 번호를 요청했을 때, LRU(Least-Recently-Used)를 사용할 경우 몇 번의 page-fault가 발생하는가? [2016년 서울 9급]

---<보기>---

요청하는 번호순서 : 2 3 2 1 5 2 4 5

---

① 6번 ② 5번 ③ 4번 ④ 3번

♤ LRU – 가장 오래동안 사용되지 않은 페이지 교체

페이지 참조	2	3	2	1	5	2	4	5
페이지 프레임	2	2	2	2	2	2	2	2
		3	3	3	5	5	5	5
				1	1	1	4	4
페이지 결함	f	f		f	f		f	

정답 : ②

16. 다음 〈조건〉에 따라 페이지 기반 메모리 관리시스템에서 LRU(Least Recently Used) 페이지 교체 알고리즘을 구현하였다. 주어진 참조열의 모든 참조가 끝났을 경우 최종 스택(stack)의 내용으로 옳은 것은? [2014년 계리직]

- LRU 구현 시 스택 사용한다.
- 프로세스에 할당된 페이지 프레임은 4개이다.
- 메모리 참조열 : 1 2 3 4 5 3 4 2 5 4 6 7 2 4

---

①

스택 top	7
	6
	4
스택 bottom	5

②

스택 top	2
	7
	6
스택 bottom	4

③

스택 top	5
	4
	6
스택 bottom	2

④

스택 top	4
	2
	7
스택 bottom	6

♤ LRU 페이지 교체 : 현시점에서 가장 오랫동안(최근에) 사용되지 않은 페이지 교체

// 일반적인 풀이 과정은 다음과 같다.(원리는 간단하지만 설명은 길다)

bottom ... top

1	2	3	4	5	3	4	2	5	4	6	7	2	4			
×	×	×	×	×	×	×	×	×	×							
1	4	2	3	5	7	6	8	9	10							

- 스택 그림에서 아래의 작은 수는 페이지가 참조될 때마다 필요시 페이지 번호가 스택의 바닥에서 제거되는 순서를 의미한다.(제거되므로 아래에 표시 ×를 해 두었다)
- 스택의 바닥에서 제거된 페이지 번호 중에서 다시 사용되는 페이지 번호는 스택 꼭대기(top)에 다시 위치한다. → 예를 들면, 페이지 번호 3번이 이에 해당한다.

↓ 주어진 문제를 약식으로 풀면, 다음과 같다.

- 메모리 참조열 순서 뒤에서 서로 다른 4개를 택하면 된다.(서로 다른 4개 : 4, 2, 7, 6)
- 이유는, LRU는 가장 오랫동안 사용되지 않은 페이지를 교체하므로

↓ LRU에서 스택의 내용

- 스택 꼭대기 : 가장 최근에 사용된 페이지 번호가 위치
- 스택 바닥 : 가장 오래 동안 사용되지 않은 페이지 번호가 위치

☞ LRU에서 스택 구조 : 스택 구조는 이중연결리스트로 구현하는 것이 좋다,

정답 : ④