#왜 관리하는가?

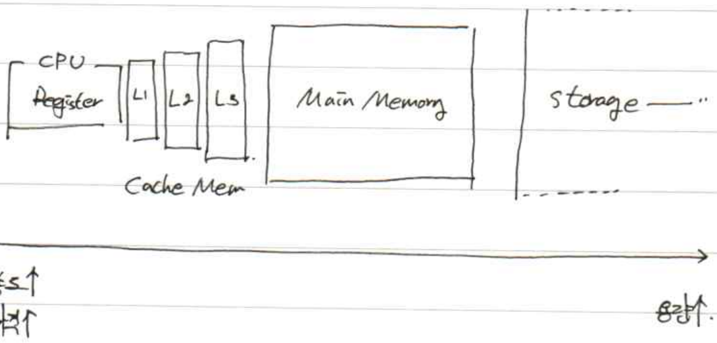
· 메모리: 공유자원.

· 메모리 보호.

· 용량 한계 극복.

· 효율성 증대.

메모리 계층화.



* 지역성.

코드·데이터·자원 등... 아주 짧은 시간 내 다시 사용됨.

⇒ 참조의 지역성으로 계층화 가능

•시간지역성 (loop).

공간지역성 (array).

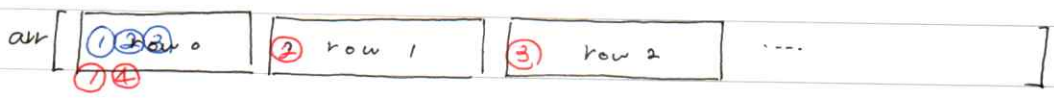
순서대로 체크 없으면 다음단계 참조 → 캐시에 데이터가 없다면? Cache miss.

· 메모리만 사용할때보다 시간↑.

· 지역성은 실존하는가?

```
for i in range(n).
    for j in range(n).
        arr[i][j]=1.  → 총 런타임  0.129.
        arr[j][i]=1.  → ——        0.359.
```

왜?

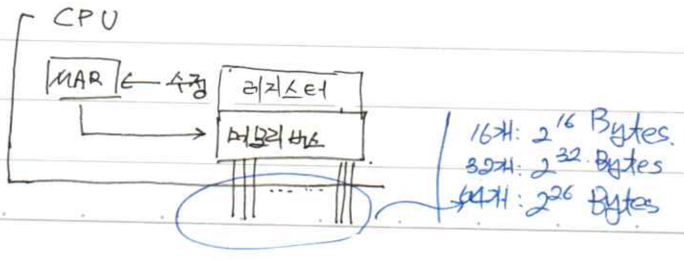


* arr[i][j]
arr[j][i]

#메모리 주소와 CPU 비트.

• 메모리 주소. 메모리를 거대한 1차원 배열로 생각, 각 인덱스 값.
(1 Bytes로 나뉨).

· Memory Address Register.



16개: 2^{16} Bytes.
32개: 2^{32} Bytes
64개: 2^{26} Bytes

NO.

#메모리 관리는 왜 어려운가. · 폰 노이만 구조. … CPU-Mem, CPU의 유일한 작업 공간이 메모리.

⇒ 모든 프로그램, 메모리에.

↓

(프로세스 간 침범 X.
필요한 공간 충분히 제공·할당
가능한 많은 프로세스 로딩)

⇓

#메모리 정책. · 적재 정책 · 배치 정책. · 재배치 정책. (교체)

Fetch Policy. Placement Policy. Replacement Policy.

+) MMU, Memory Management Unit.

#메모리 주소.

#물리 주소와 논리 주소. · Physical Address. (HW) (=절대주소)
→ 하드웨어에 의해 고정.

· Logical Address. (SW). (=상대주소). Virtual Address.
→ CPU가 다루는 주소.

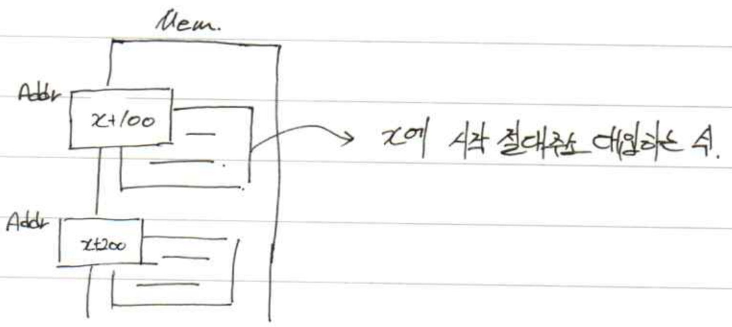
← Address Binding →

* 사용자·프로세스, 절대 물리 주소 확인 X.

# Address Binding. · Compile-time Binding. 물리 주소가 컴파일시 결정.
물리=논리 ∴ 같은 주소에 이미 프로세스 있으면 load 불가.
⇒ 비효율적.

· Load-time Binding. Loader가 프로세스 load 중 물리 주소 결정.

· Run-time Binding. 작업 중에 즉석으로 주소 전환.



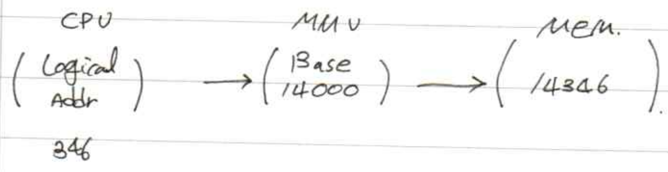
#MMU, Memory Management Unit. · HW주소 → SW주소 변환.
· CPU에 내장.
· 모든 주소참조는 MMU거침: CPU도 MMU 통해서 접근.
CPU; 논리주소만 앎.

Memory Management Unit.

· 논리 주소 → 물리 주소.

· CPU ⟷ MMU ⟷ MEM. … CPU는 물리 주소를 모름.

#상대주소 → 물리주소. 주소변환.



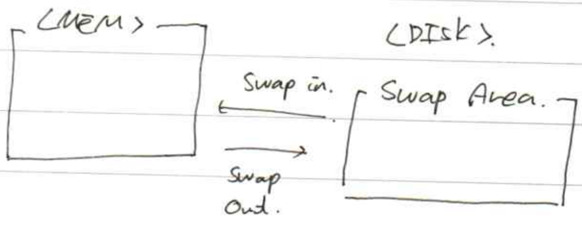
#메모리 오버레이.

· 가상메모리 용도: 각 프로세스, 자신이 모든 메모리 점유하는것처럼 생각.

· 메모리 오버레이 … 프로그램 크기가 실제보다 클 때, 전체에서 일부 잘라서 코드.
필요한 부분만 메모리에 올려서 실행 … "바꿔치면서" (= Swap).

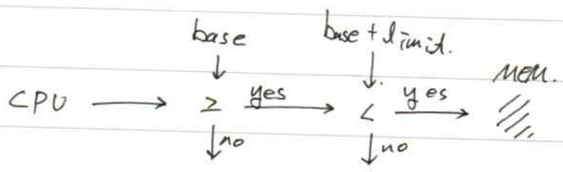
Swap.

· 저장장치: 공간만 대여.
메모리 관리자: 스왑 영역 관리.



* Swap in, out 반복 시 … 단편화. 성능 문제!

#메모리 보호.

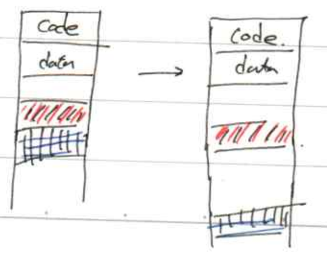


#ASLR. (Address Space Layout Randomization). "주소공간의 랜덤 배치"

· 메모리 주소 유출 .. 위험성.

· 실행할 때마다 논리주소 바뀌게.

* base를 섞음. ((void *) main - func
는 항상 일정.



NO. 24. 05. 27.

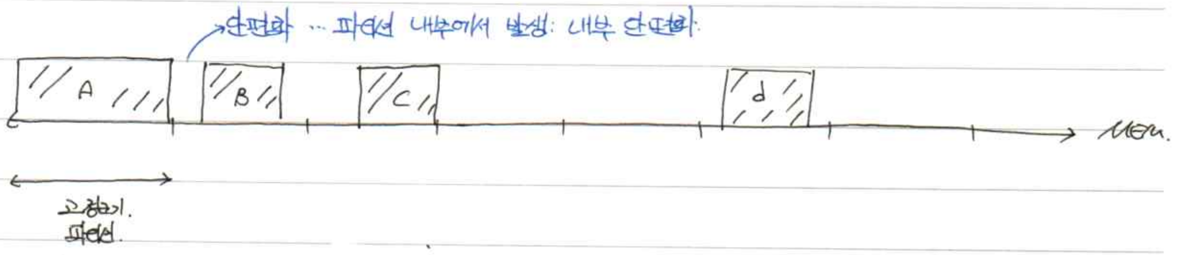
# 메모리 할당

→ 각 프로세스 영역을 연속된 공간에 배치.

	연속	불연속
가변		
고정		현대 OS 채택.

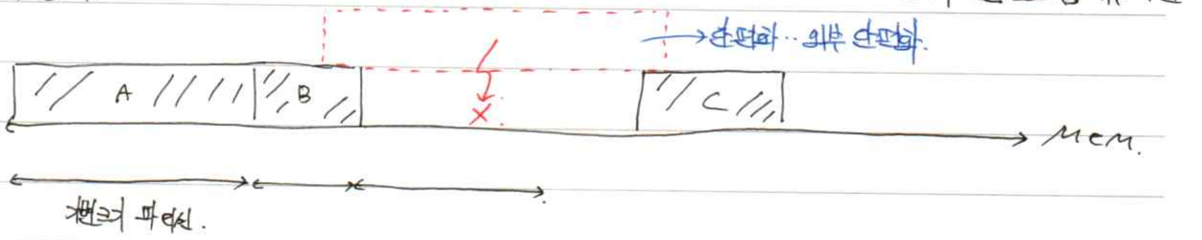
# 연속 메모리 할당. + 고정크기.

"동시에 올릴 프로그램 개수고정"



# 연속 메모리 할당 + 가변크기.

"동시에 올릴 프로그램 개수 가변"



# 연속 메모리 할당.
· base 레지스터 하나만 있으면 됨 ⇒ 단순.
CPU의 MEM 액세스 ↑.
/ 유연성 ↓.
단편화.. (외부단편화).

# 연속 할당의 배치 전략. ."홀 선택 알고리즘. 동적 메모리 할당"
· Allocation List 유지.

(할당가능한 크기의 홀중).

· first fit. ✓	· best fit ✓	· worst fit.
제일 첫 홀	가장 작은 홀.	가장큰 홀.

NO.

단편화. fragmentation.

· 프로세스 할당 불가한 틈 발생.

· External	Internal.
할당된 메모리 사이에 틈	할당된 메모리 내부에 틈.

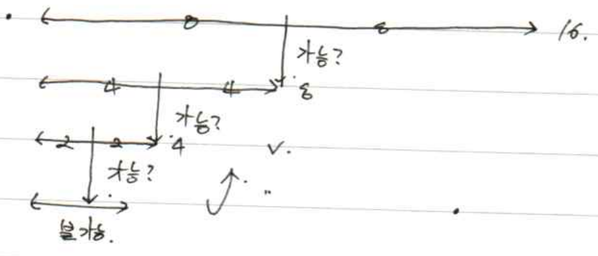
Area Size	Task Size.	
50	60	Allocation Impossible, external fragmentation.
150	160	Impossible, external.
200	100	Possible, Internal (100).
250	150.	Possible, Internal (100).

조각모음. De-fragmentation.

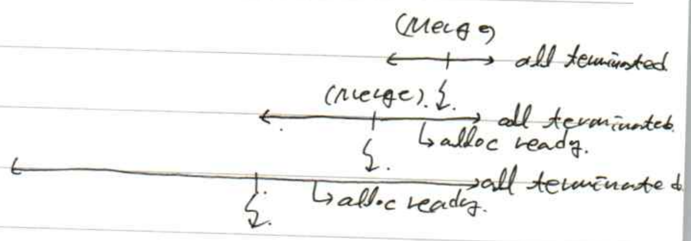
1. 조각모음 대상의 동작 정지.
2. 적당한 위치로 이동.
3. 작업 후 다시 시작.

} 비선 작업.

Buddy System. - Binary Search of Memory Allocation.



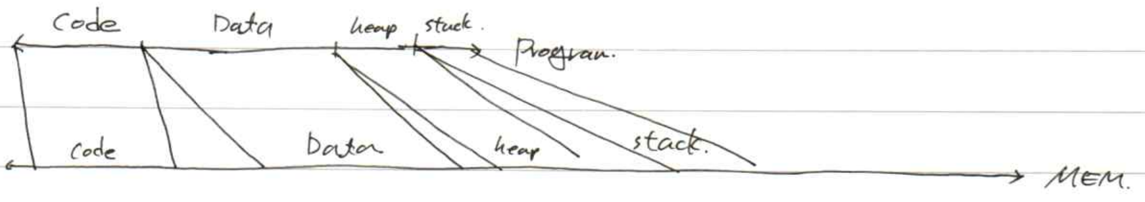
3 allocating.



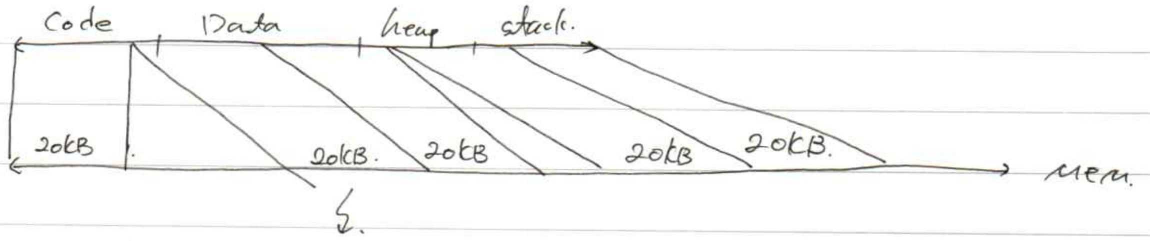
morning glory

#Non-Contiguous Allocation.

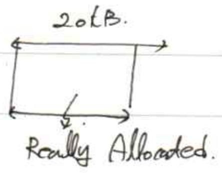
Segmentation.



Paging.



=> 내부단편화.



#Segmentation.

· Segment: 개발자 관점의 논리적 구성 단위. (세그먼트마다 크기 상이).

· 각 세그먼트마다 Base 존재… Base 4개 존재.

* Using 세그먼트 테이블.

컴파일러·링커·로더·OS에 의해. 논리 세그먼트 분할..

· 외부 단편화 발생.