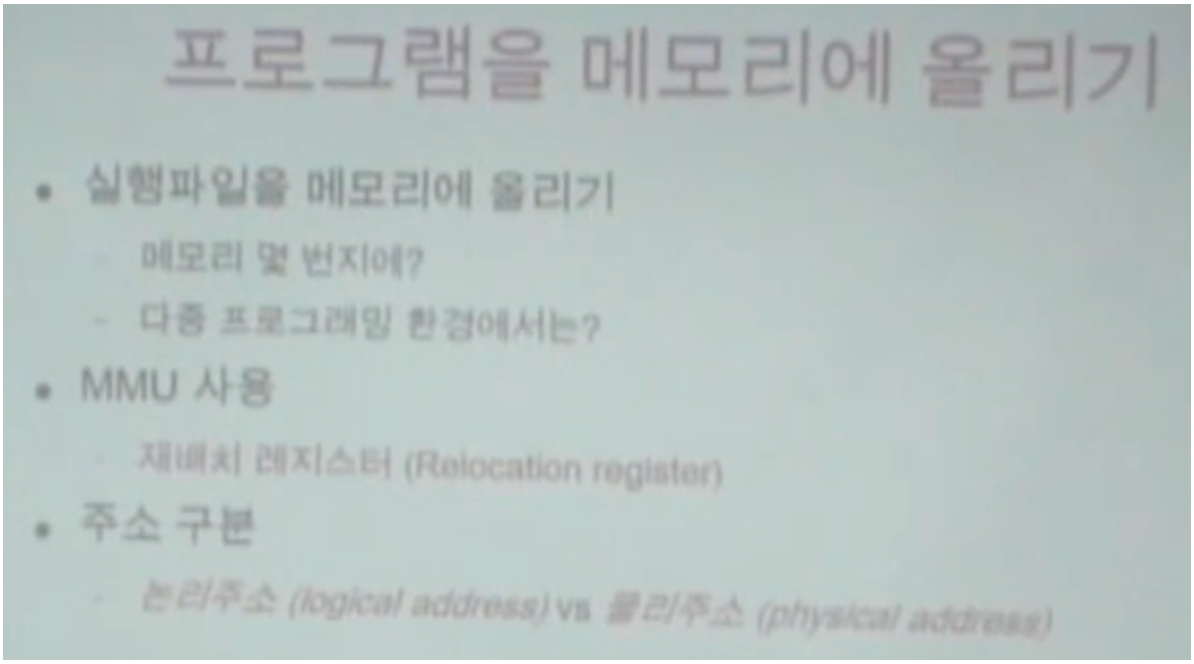
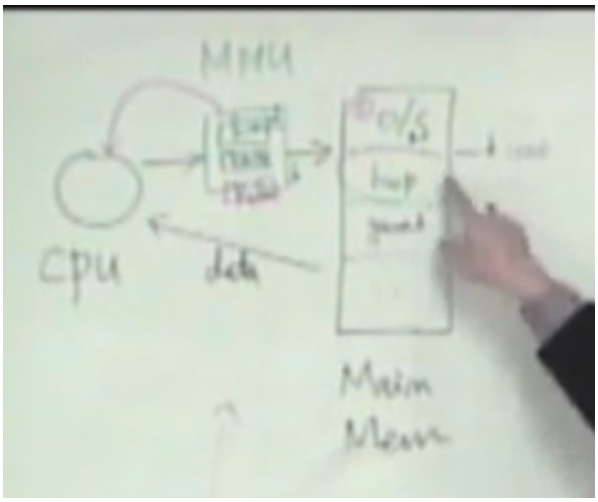
운영체제08_메모리절약

주기억장치 관리





MMU 사용

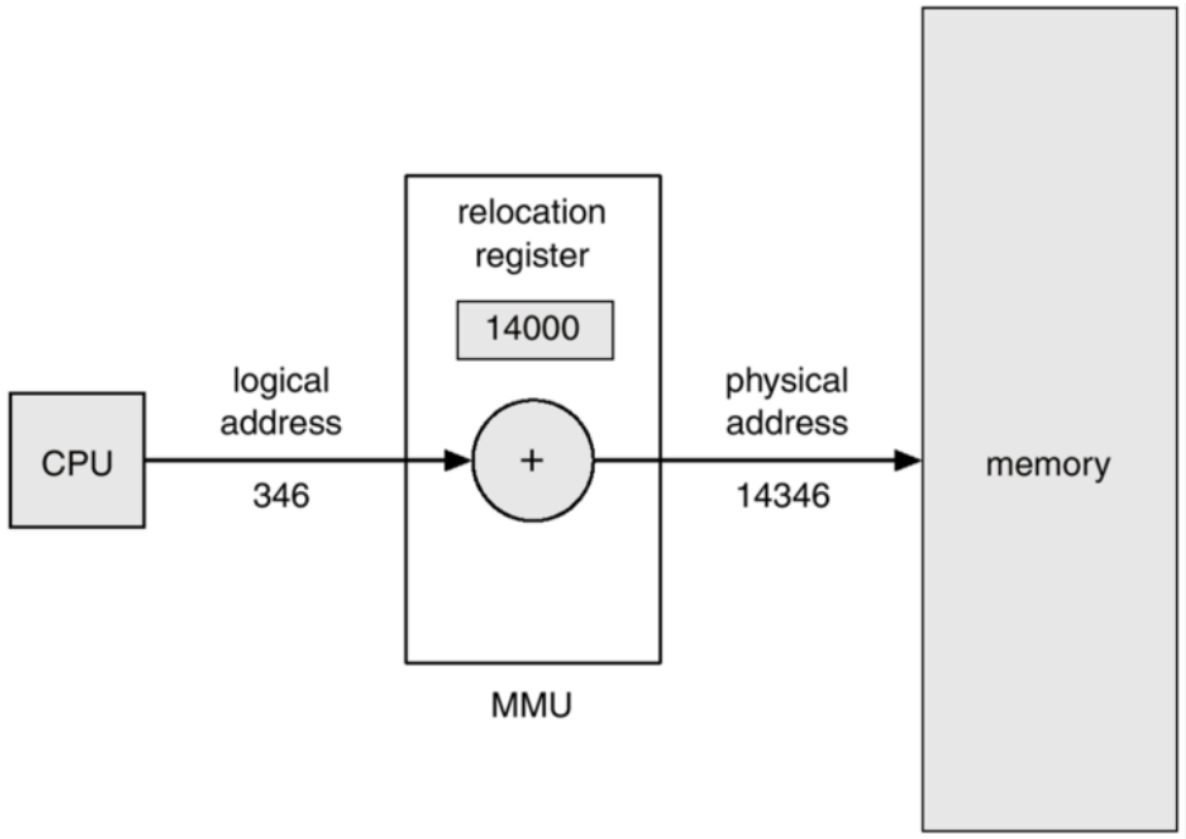
CPU는 메모리의 프로그램에 접근하기 위해 address를 보냄

이 때 다른 영역이 아닌 해당 프로그램의 영역에만 접근해야 하는데 이를 기능하게 하는 것이 MMU의 역할

MMU에는 다음과 같은 register들이 존재

```
- base register
- limit register
- relocation register
```

RELOCATION REGISTER



메모리에 프로그램을 적재할 때 항상 같은 위치에 프로그램을 올릴 수는 없다. 어떤 날에는 0번지에 올리기도 하고 다른 날에는 500번지에 올리기도 한다. 이는 메모리 공간에서 빈 공간을 찾아 프로그램을 올리기 때문에 알 수가 없다. 그래서 이를 맞추어 조절해주는 역할을 하는 것이 바로 MMU의 재배치 레지스터이다.

relocation register의 예시

```
06월 01일에는 hwp 프로그램이 1001번지에 할당되어 있음
CPU는 hwp 프로그램을 향해 1번지 address를 보냄
MMU의 relocation register가 1000이라는 값을 담아서 전송해 1001번에 도달 가능
하지만 cpu는 1번이라고 속고 있다.

06월 02일에는 프로그램 사용이 많아서 5001번지에 할당되어 있음
CPU는 그대로 hwp 프로그램을 향해 1번지 address를 보냄
MMU의 relocation register가 5000이라는 값을 담아서 전송하므로 결국은 5001번에 도달 가능
하지만 CPU는 계속 1번이라고 속고있다.
```

이 때 CPU가 보는 주소가 `logical address, 논리주소`

실제로 MMU를 통과하고 오는 주소 `physical address 물리주소`

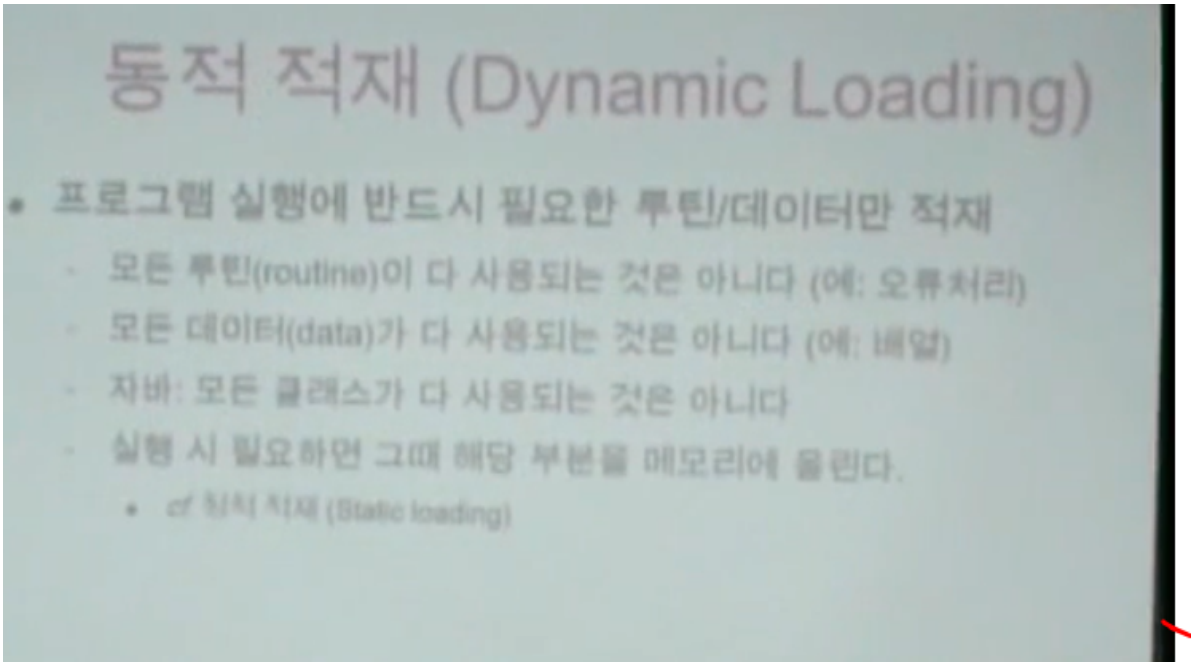
결론

MMU 덕분에 어떤 주소에 프로그램이 와도 되게 되는 것

프로그램이 메모리의 몇번지에 위치하는지가 문제가 되지 않음

메모리 낭비 방지

1. 동적 적재

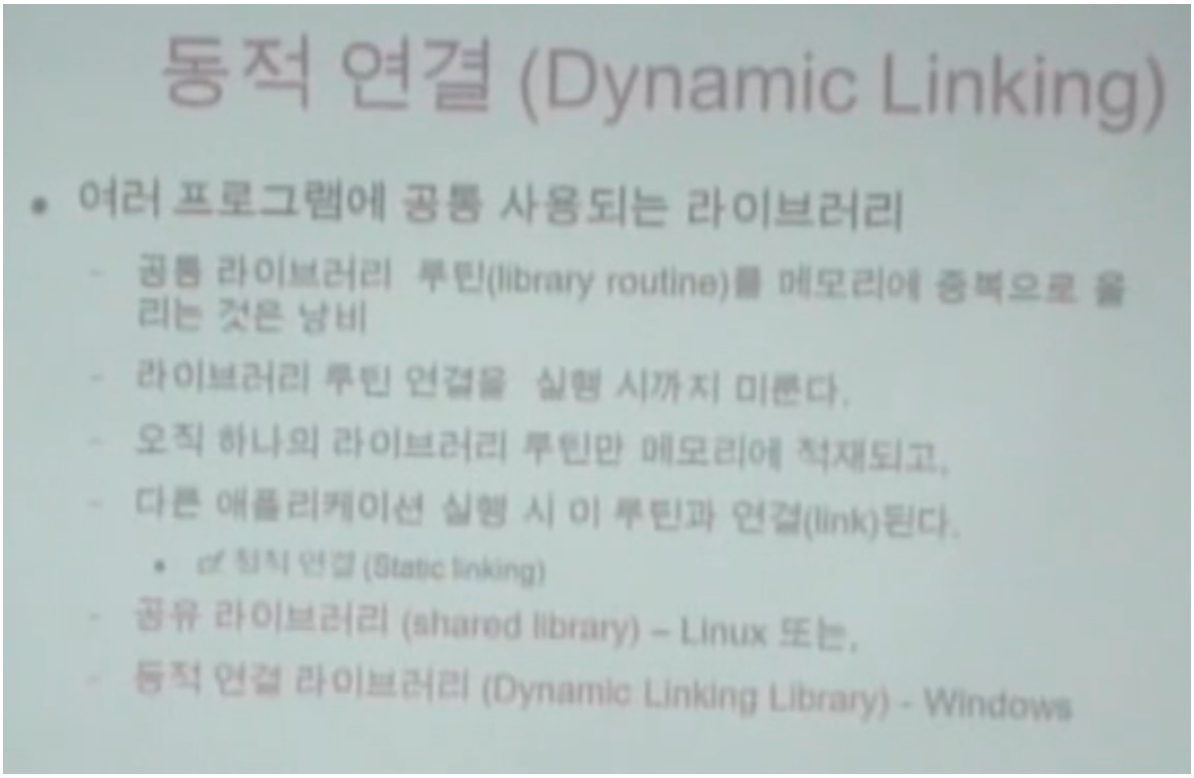


상황 예시

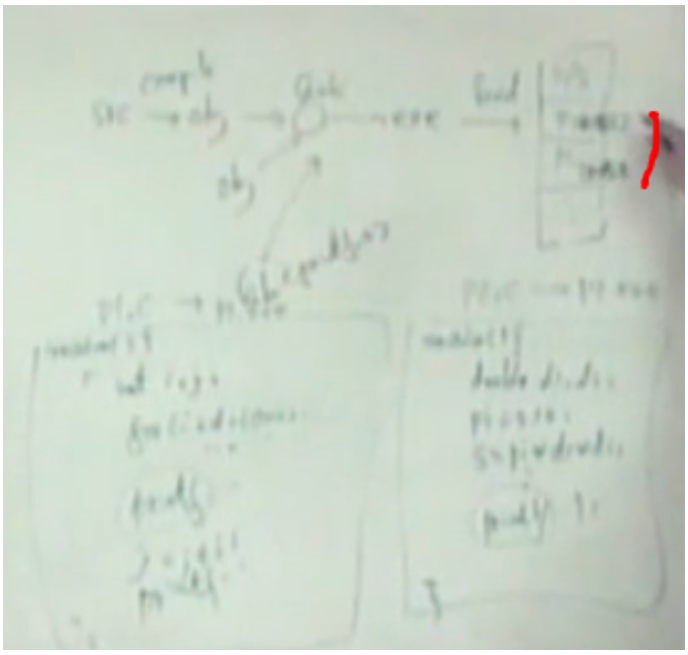
```
- 프로그램에서 오류가 발생했을 때 ~한 것을 실행한다고 함수를 짰다고 가정
- 이를 포함해 메모리에 올림
- 근데 사실 오류가 안일나지 않음
- 메모리에 올릴 필요가 없었는데 올려버린 그 자리가 너무 아까움
- 실제로 오류가 일어났을때만 올리고 싶어짐
```

즉, 필요하면 해당 부분을 올리자! (= 동적 적재)

2. 동적 연결



상황 예시



```
1) 숫자의 합을 출력하는 함수
2) 원의 넓이를 출력하는 함수

- 두개의 함수를 실행해 메모리에 올림
- 둘다 PRINT(함수)를 공통적으로 사용중
- 동일한 코드를 올리는 것은 낭비
- 메모리에 하나만 올려두고 LINK 해서 사용하자
```

원래는 실행되기 전에 라이브러리를 link (`정적 연결`)

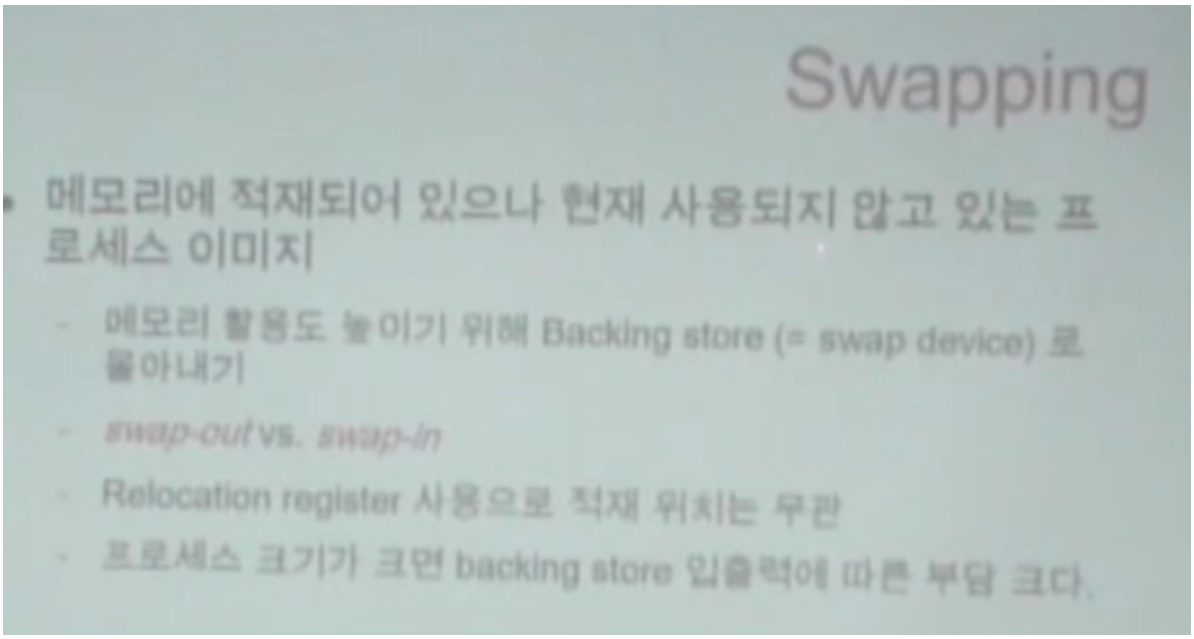
이제는 하나만 올려놓고 필요할때 link (`동적 연결`)

이를 linux에서는 공유 라이브러리라고 부름, 윈도우는 동적 라이브러리라고 부름

결론

연결은 프로그램이 실행될 때

SWAPPING



상황 예시

```
< swap-out >
- 화장실에 다녀오게 됨
- 메인메모리에는 올라와있지만 사용되지 않는 프로그램이 생김
- 메인메모리에서 하드디스크의 backing store로 쓸어냄

< swap-in >
- 화장실에서 돌아옴
- 그 memory에 다른 process가 이미 사용중
- 그 자리에는 못가고 memory의 다른 주소로 들어감
- (이 때 relocation register 덕분에 잘 찾아가게 해줌)
```

결론

이런 입출력 과정속에서 메모리의 효율은 높아질 수 있으나

하드디스크의 backing store에 대한 부담이 큼