Windows 메모리 레이아웃

프로세스 메모리 구조

섹션

ex) 윈도우 PE 파일 → PE 헤더와 1개 이상의 섹션으로 구성되어 있다.

섹션 : 유사한 용도로 사용되는 데이터가 모여 있는 영역.

섹션에 대한 정보는 PE 헤더에 저장되어 있음.

- 섹션의 이름
- 섹션의 크기
- 섹션이 로드될 주소의 오프셋
- 섹션의 속성과 권한

윈도우는 PE를 실행할 때, 이 정보를 참조하여 PE의 각 섹션들을 가상 메모리의 적절한 세그먼트에 매핑.

섹션의 종류

.text

실행 가능한 기계 코드가 위치하는 영역

읽기/실행 권한이 부여된다.

```
int main() { return 31337; }
```

해당 함수가 컴파일되면 기계 코드로 변환되어 코드 세그먼트(text 섹션)에 위치

.data

컴파일 시점에 값이 정해진 전역 변수들

읽기/쓰기 권한 부여

```
int data_num = 31337;
char data_rwstr[] = "writable_data";        // data

int main() { ... }
```

.rdata

컴파일 시점에 값이 정해진 **전역(const) 상수와 참조할 DLL 및 외부 함수들의 정보**

(*전역 상수 : 프로그램 전체에서 접근 가능하며(전역) 값이 변하지 않음(상수))

읽기 권한만 부여

```
const char data_rostr[] = "readonly_data";
char *str_ptr = "readonly";  // str_ptr은 .data, 문자열은 .rdata

int main() { ... }
```

- str_ptr
  - 문자열 변수 자체는 컴파일 시점에 값이 정해지므로 .text 섹션
  - 변수에 대입될 문자열은 상수 문자열로 취급되어 rdata에 저장

섹션이 아닌 메모리

윈도우의 가상 메모리 공간에는 섹션뿐 아니라 프로그램 실행에 있어 필요한 스택, 힙 역시 적재.

스택

윈도우 프로세스의 각 쓰레드는 자신만의 스택 공간을 가지고 있다.

보통 지역 변수/함수 리턴 주소 저장

스택은 주로 기존 주소보다 낮은 주소로 확장됨. (아래로 자란다)

```
void func() {
  int choice = 0;  // 지역 변수 : 스택으로

  scanf("%d", &choice);

  if (choice)
    call_true();
  else
    call_false();

  return 0;   // 얘는 레지스터 (리턴값 저장하는 레지스터 : rax Return
}
```

힙

프로그램이 여러 용도로 사용하기 위해 할당받는 공간. → 모든 종류의 데이터 저장 가능

스택과 다른 점:

- 스택보다 비교적 큰 데이터 저장 가능
- 전역적으로 접근 가능
- 실행 중 동적으로 할당

보통 읽기/쓰기, 실행은 상황에 따라

```
int main() {
  int *heap_data_ptr =
      malloc(sizeof(*heap_data_ptr));  // 동적 할당한 힙 영역의 주소를 가리킴
  *heap_data_ptr = 31337;              // 힙 영역에 값을 씀
  printf("%d\n", *heap_data_ptr);  // 힙 영역의 값을 사용함
```

```
  return 0;
}
```

강의 요약

섹션	역할	일반적인 권한	사용 예
.text	실행 가능한 코드가 저장된 영역	읽기, 실행	`main()` 등의 함수 코드
.data	초기화된 전역 변수가 위치하는 영역	읽기와 쓰기	초기화된 전역 변수, 전역 상수
.rdata	초기화된 전역 상수나 임포트 데이터가 위치하는 영역	읽기 전용	전역 상수, 임포트 데이터
스택	일시적으로 저장하고 사용하는 임시 영역	읽기, 쓰기	지역 변수, 함수의 인자 등
힙	자유롭게 사용할 수 있는 영역	읽기, 쓰기	malloc(), calloc() 등으로 할당 받은 메모리