

template / linker

EUnS

January 12, 2020

① compiler

② linker

Relocatable Object Files

symbol

linker sequence

③ template

④ TMP

⑤ constexpr (c++11)

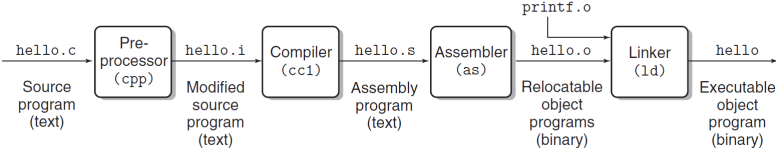


Figure: 컴파일 과정

- 서적 : CSAPP (7장 linker)
- 아마 시프시간에 배울것.(SNU,kaist..)
csapp 강의

Object File

그냥 기계어다.

- Relocatable ob : compiler, assembler output
- Executable ob : linker output
- shared ob : DLL을 위한 파일 생략

ELF

x86-linux : object구조로 ELF(Executable and Linkable Format)사용

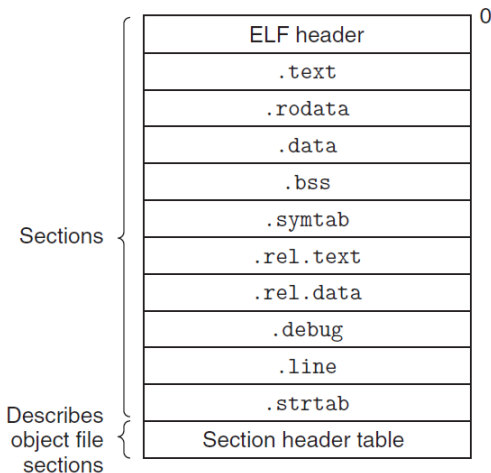


Figure: Typical ELF relocatable object file.

- read write segment
 - .data : 전역, static 변수
 - .bss : 초기화 안된 전역 or static 변수, 0으로 자동 초기화
- read only
 - .text : 머신코드
 - .rodata : 읽기만 가능한 데이터(ex) jump table)
- .symtab : symbol table
- .debug : symbol table for debug
- .line : 생략
- .strtab : 생략
- .rel.text 배치해야할 .text
- .rel.data 배치해야할 .data 영역

symbol

- Global symbols : 모듈 m에 정의되거나 다른 모듈에 참조될 수 있는 심볼 / 전역변수와 static이아닌 함수
- Global symbols : 다른 모듈에 정의된 전역변수 /다른곳에 정의된 (extern)전역변수와 static이아닌 함수
- Local symbols : static으로 선언된 함수, static 전역변수

Q : 지역 변수는요?

A : 런타임에 스택에 쌓습니다.

linker 단계

- ① symbol resolution
- ② Relocation

Symbol Resolution

- strong symbol : 초기화된 전역변수, 함수
- weak symbol : 초기화x 전역변수

Rule

- ① Multiple strong symbols with the same name are not allowed.
- ② Given a strong symbol and multiple weak symbols with the same name, choose the strong symbol.
- ③ Given multiple weak symbols with the same name, choose any of the weak symbols.

Relocation

- 1 Relocating sections and symbol definitions : 여러개의 .data section 합친다. 이후 인스트럭션과 전역변수들이 런타임 메모리 주소를 가진다.
- 2 Relocating symbol references within sections. : 모든 심볼 참조를 수정한다. 이후 모든 심볼들이 런타임 메모리 주소를 가진다. 어셈블러가 만든 .rel.data section에 있는 Relocation Entries 자료구조를 가지고 수행한다
 - R_X86_64_PC32 : 상대주소
 - R_X86_64_32 : 절대주소

Executable Object Files

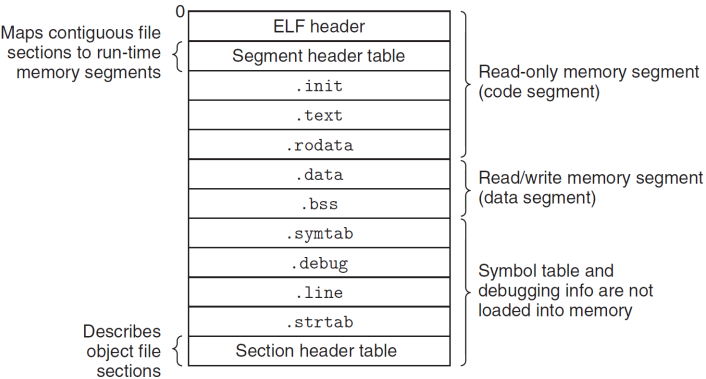


Figure: Typical ELF executable object file.

Loading

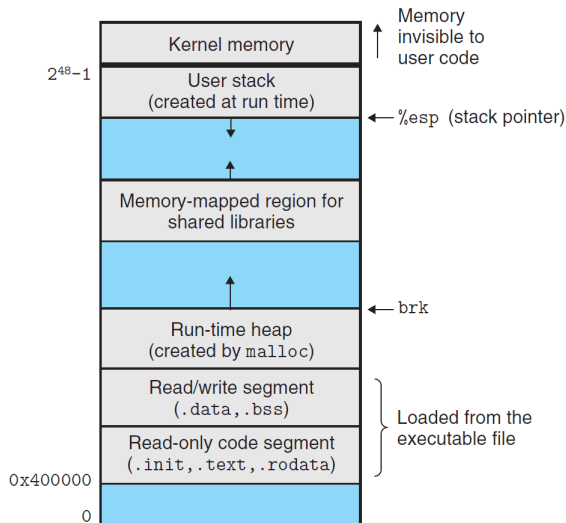


Figure: Linux x86-64 run-time memory image.

template function

- 타입에 얽매이지 않고 코드를 작성할 수 있게 해줌
- container와 기타 라이브러리를 구현한 방법

```
1 void swap(int &a, int &b)
2 void swap(double &a, double &b)
3 void swap(name &a, name &b)
4 // replace
5 template<typename T>
6 void swap(T &a, T& b);
7
```

template

- 타입(class) 아무거나 넣어도 가능, 템플릿 인자 여러개 사용가능.
- template을 실제로 인자를 넣어서 사용할때 직접 인자를 넣은 실제 코드를 생성함 짜게되는 template 코드는 실제 실행 코드가 아님
- 실행 파일크기가 생각보다 커질 수 있음
- 템플릿의 링커단계 처리는 컴파일러마다 차이가 있음. (GNU에서는 weak symbol로 처리해서 생성되는 template중 아무거나 택한다.)[참고](#)


```

1  template<typename T>
2  class name
3  {
4      T item;
5      T function(const T &a, const T &b);
6      int a;
7  };
8  template<typename T>
9  T name<T>::function(const T &a, const T &b) {;}
10

```

template linker error

- template을 파일 분할을 해보자.
- 링커 에러가 뜰것이다.
- 왜?

header file

- 그냥 텍스트파일과 같다.
- 선언을 위한것...
- 실제 구현은 어느 cpp파일에서...
- `#include`는 **전처리 지시자**.
- `#include`로 cpp파일에 그대로 붙여넣는다.

template specialization

```
1  #include<iostream>
2  using namespace std;
3  template<typename T>
4  T function(T a) {
5      cout << "Template" << endl;
6  }
7  template<> // specialization
8  char function<char>(char a) {
9      cout << "specialization " << endl;
10 }
11 char function(char a){
12     cout << "overloading" << endl;
13 }
14 int main(){
15     function(1);
16     function('s');
17 }
18
```

- 권장되는 방법은 아님 [참고1](#) [참고2](#)

TMP

```
1  template <int N>
2  struct Factorial {
3      static const int result = N * Factorial<N - 1>::
   result;
4  };
5  template <
6  struct Factorial<1> {
7      static const int result = 1;
8  };
9  cout << Factorial<4>::result;
10
```

- template의 특성을 이용해서 반복되는 계산을 코드를 생성하여 계산 해놓은다음 그 값을 $O(1)$ 에 부르는 흑마법
- 이런것도 가능

constexpr

참고

- 변수에 사용할때
 - `#define` 상수 대체가능
 - `const` 상수 완전히 대체가능
 - 컴파일타임에 상수로 대체
- 함수에 사용할때
 - 어느부분 TMP 대체가능
 - 컴파일타임에 계산할수도있고 안할수도있음

constexpr 활용

```
1      constexpr int facto(int i){
2          int a = 1;
3          for (int j = 2 ; j <= i; ++j) {
4              a *= j;
5          }
6          return a;
7      }
8      constexpr int Factorial[13] = {
9          1,
10         facto(1) ,
11         facto(2) ,
12         facto(3) ,
13         facto(4) ,
14         facto(5) ,
15         facto(6) ,
16         facto(7) ,
17         facto(8) ,
18         facto(9) ,
19         facto(10) ,
20         facto(11) ,
21         facto(12)
22     };
23
```

과제

과제 3-2 : 링크드리스트