

ELF File Format

김선호, 조준래

Embedded Systems Lab. of Hansung University

작성자 : 김 선호 (tjsghgogo@naver.com)

조 준래 (wnsfo@hansung.ac.kr)

© 한성대학교 컴퓨터공학과 임베디드 시스템 실험실

© **Embedded Systems Lab. Hansung University**



이 문서는 [크리에이티브 커먼즈 코리아 저작자표시 2.0 대한민국 라이선스](https://creativecommons.org/licenses/by-nc-sa/2.0/ko/)에 따라 이용하실 수 있습니다.

목 차

▶ 머리말

▶ 목적파일

- ELF 헤더
- 섹션 헤더 테이블
- 문자열 테이블
- 심볼 테이블
- 재배치 테이블

▶ 프로그램 적재 및 동적 연결

- 프로그램 헤더
- 프로그램 적재
- Elf 링킹

▶ Elf 파일 추적

머리말

Elf File Format

- ▶ ELF는 원래 UNIX 시스템 연구소에서 Application Binary Interface(ABI)의 한 부분으로 개발되고 공개되었다.
- ▶ ELF의 약자는 “Executable and Linkable Format” 이다.
- ▶ ELF 표준은 다양한 운영체제에 걸쳐서 사용될 수 있는 이진 인터페이스를 프로그래머에게 제공함으로써, 소프트웨어 개발에 연계성을 주기 위해 만들어졌다.
- ※ 따라서 서로 다른 여러 인터페이스의 구현을 방지하고, 그럼으로써 프로그램들을 다시 짜고 재 컴파일 해야 할 필요성을 줄이게 된다.

ELF File Types

▶ 재배치 파일

- 다른 목적파일과 연결됨으로써 실행프로그램이나 공유목적파일을 생성할 수 있는 코드와 데이터들을 가지고 있다.

▶ 실행 파일

- 실행에 적합한 프로그램을 가지고 있다.

▶ 공유목적파일

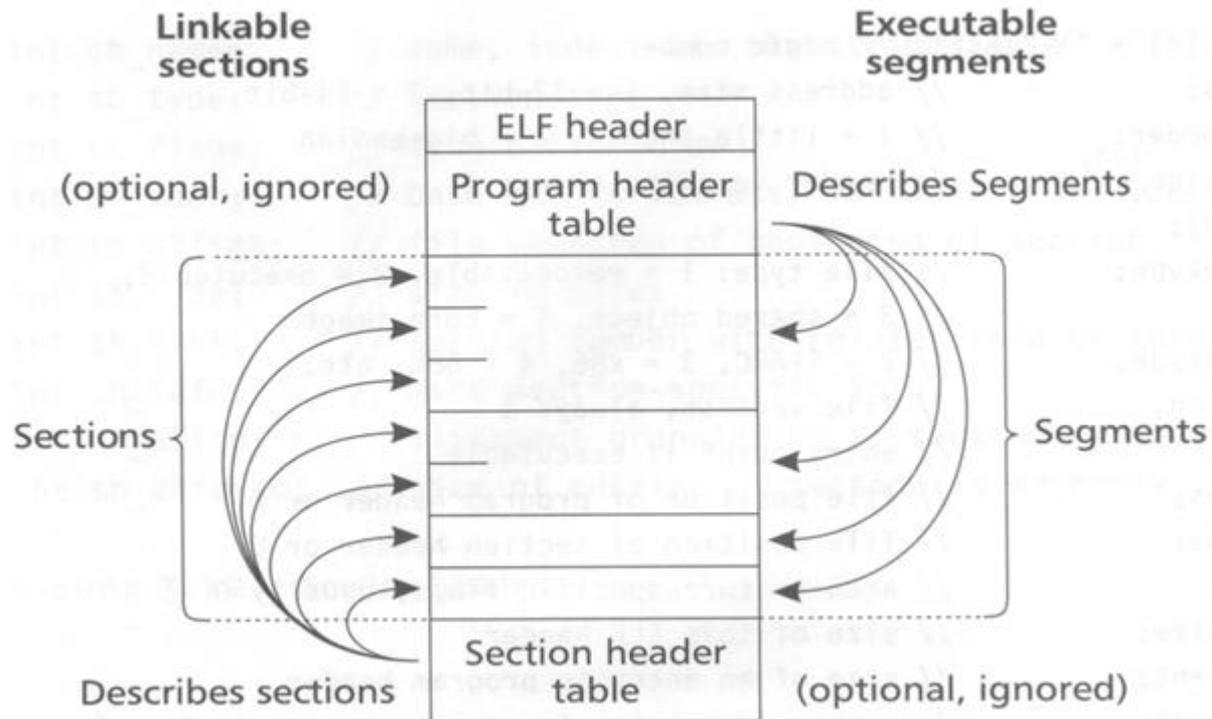
- ld를 이용하여 다른 재배치 목적파일이나 공유목적파일과 연결되어 새로운 목적파일을 만들어내는 것
- 동적 링커가 실행파일과 다른 공유목적파일을 연결하여 프로그램을 실행한다.

ELF Structure

- ▶ **ELF 파일은 두 가지 형태의 파일구조를 가진다.**
 - **컴파일러, 어셈블러, 링커들은 논리적 섹션 테이블로 설명한 논리적 섹션 집합들의 파일을 취급합니다.**
 - **시스템 로더는 프로그램 헤더 테이블로 설명한 세그먼트 집합들의 파일을 취급합니다.**

ELF Structure

- ▶ 목적 파일은 프로그램의 링킹 단계와 실행단계에서 사용된다.



ELF Structure

- ▶ 단일 세그먼트는 보통 몇 가지 섹션으로 구성되어 있습니다
ex) loadable read-only 세그먼트는 실행 코드에 대한 섹션이 포함될 수 있습니다. 또한, 동적 링커에 대한 심볼 섹션이 포함될 수 있습니다.
- ▶ 재배치 파일은 섹션 헤더 테이블을 가진다.
실행파일은 프로그램 헤더 테이블을 가진다.
공유목적파일은 둘 다 가진다.
- ▶ 섹션이 링커에 의해 추가로 처리할 의도가 있다면, 세그먼트는 의도하는 메모리로 매핑합니다.

Data Representation

- ▶ 목적파일 구조는 8-비트 바이트와 32비트 구조를 가진 다양한 CPU들을 지원한다.
- ▶ 기계종속적이 아닌 구조를 지원할 수 있도록 몇 가지의 제어 데이터들을 제공하여, 보다 일반적인 방법으로 목적파일의 내용을 인식하고 해석할 수 있도록 한다.

Name	Size	Alignment	Purpose
Elf32_Addr	4	4	Unsigned program address
Elf32_Half	2	2	Unsigned medium integer
Elf32_Off	4	4	Unsigned file offset
Elf32_Sword	4	4	Signed large integer
Elf32_Word	4	4	Unsigned large integer
unsigned char	1	1	Unsigned small integer

목적파일

ELF Header (1 / 2)

- ▶ ELF 헤더는 언제나 처음 부분에 위치하고 있다.
- ▶ 프로그램 헤더 테이블과 섹션 헤더 테이블의 파일 위치는 ELF 헤더에서 정의하고 있다.
- ▶ 헤더는 심지어 기계와 파일의 대상 아키텍처에서 다른 바이트 순서를 해독할 수 있다.
 - 클래스와 인코딩 방법은 Elf 헤더에 있는 나머지 필드를 읽은 후에 해독할 수 있다.
 - elf format은 두 개의 다른 주소 사이즈를 제공한다.
 - 32 bits
 - 64 bits

ELF Header (2/2)

```
#define EI_NIDENT      16

typedef struct {
    unsigned char    e_ident[EI_NIDENT];
    Elf32_Half      e_type;
    Elf32_Half      e_machine;
    Elf32_Word      e_version;
    Elf32_Addr      e_entry;
    Elf32_Off       e_phoff;
    Elf32_Off       e_shoff;
    Elf32_Word      e_flags;
    Elf32_Half      e_ehsize;
    Elf32_Half      e_phentsize;
    Elf32_Half      e_phnum;
    Elf32_Half      e_shentsize;
    Elf32_Half      e_shnum;
    Elf32_Half      e_shstrndx;
} Elf32_Ehdr;
```

Elf Header

Section Header (1 / 2)

- ▶ **섹션들은 ELF 헤더와 프로그램 헤더 테이블, 섹션 헤더 테이블을 제외한, 목적파일의 모든 정보를 가지고 있다. 또한 목적파일의 섹션들은 몇 가지 조건들을 만족한다.**
 - **목적파일 안에 있는 모든 섹션들은 자신을 나타내는 섹션 헤더가 정확히 하나 존재한다. 섹션 헤더는 섹션을 가리지 않은 채 존재할 수 있다.**
 - **모든 섹션은 파일내에서 연속적인(비어 있을 수도 있음) 바이트열을 차지하고 있다.**
 - **파일 안의 섹션들은 겹쳐질 수 없다. 파일 안의 어떤 바이트도 하나 이상의 섹션에 포함되지 않는다.**
 - **목적파일은 사용되지 않는 공간을 가질 수 있다. 다수의 헤더들과 섹션들도 목적파일의 모든 바이트를 설명하지는 않는다. 사용되지 않는 데이터들의 내용은 나타나지 않는다.**

Section Header (2/2)

```
typedef struct {
    Elf32_Word    sh_name;
    Elf32_Word    sh_type;
    Elf32_Word    sh_flags;
    Elf32_Addr    sh_addr;
    Elf32_Off     sh_offset;
    Elf32_Word    sh_size;
    Elf32_Word    sh_link;
    Elf32_Word    sh_info;
    Elf32_Word    sh_addralign;
    Elf32_Word    sh_entsize;
} Elf32_Shdr;
```

Types in Section Header

▶ SHT_PROGBITS

- 이 섹션은 프로그램에 의해 정의되는 정보들을 담고 있다.

▶ SHT_NOBITS

- 이 타입을 갖는 섹션은 목적파일속에 아무런 공간도 차지하지 않으며, 그 외는 SHT_PROGBITS 와 유사하다.

▶ SHT_SYMTAB와 SHT_DYNSYM

- 이 섹션들은 심볼 테이블을 가지고 있다.

▶ SHT_STRTAB

- 문자열 테이블을 가지고 있는 섹션이다.

▶ SHT_RELA and SHT_REL

- 재배치 엔트리들을 담고 있는 섹션이다.

▶ SHT_HASH

- 심볼 해시 테이블을 가진 섹션이다. 동적 링크에 관계되는 모든 정보들은 심볼 해시 테이블을 담고 있어야 한다

Flags in Section Header

▶ SHF_WRITE

- 이 섹션은 프로세스가 실행중일 때 쓰여질 수 있는 데이터들을 가지고 있다.

▶ SHF_ALLOC

- 이 섹션은 프로세스가 실행할 때 메모리들을 차지한다. 몇몇의 제어용 섹션들은 목적 파일의 메모리 이미지에 존재하지 않는다. 이 속성은 이러한 섹션들에 대해서는 값을 갖는다.

▶ SHF_EXECINSTR

- 이 섹션은 실행될 수 있는 기계어 명령어들을 담고 있다.

▶ SHF_MASKPROC

- 이 마스크 비트에 포함되는 모든 비트들은 프로세서에 의존적인 의미를 지니는 것으로 예약되어 있다.

Various Sections (1 / 6)

▶ .text

- 프로그램의 “텍스트”, 또는 실행 가능한 명령어들을 담고 있다.
- Type : SHT_PROGBITS
- Flags : SHT_ALLOC + SHT_EXECINSTR

▶ .data

- 프로그램 메모리 영역에 제공되는 초기화된 데이터들을 담고 있다.
- Type : SHT_PROGBITS
- Flags : SHT_ALLOC + SHT_WRITE

Various Sections (2/6)

▶ .rodata

- 프로세스의 이미지에서 쓰기 불가능한 세그먼트에 전형적으로 사용되는 읽기 전용 데이터들을 담고 있다.
- Type : SHT_PROGBITS
- Flags : SHF_ALLOC

▶ .bss

- 프로그램의 메모리 영역에 제공되는 초기화되지 않은 데이터들을 담고 있다. 또한, SHT_NOBITS의 경우와 같이 목적파일 내에 아무런 공간도 차지하지 않는다.
- Type : SHT_NOBITS
- Flags : SHT_ALLOC + SHT_WRITE

Various Sections (3/6)

▶ *.rel.text, .rel.data, and .rel.rodata*

- 재배치 정보를 담고 있다.
- Type : SHT_REL
- Flags : 만일 파일이 재배치를 포함하는 적재 가능한 세그먼트를 가지고 있다면, 섹션의 속성은 SHF_ALLOC 비트를 포함하며, 그렇지 않은 경우 그 비트는 없을 것이다.

▶ *.symtab*

- 심볼 테이블을 담고 있다.

▶ *.strtab*

- 문자열들을 담고 있는데, 대부분은 보통 심볼 테이블과 관련되어 있는 이름을 나타내는 문자열들이다.

Various Sections (4/6)

▶ .init

- 프로세스의 초기화에 사용될 실행 명령어들을 가지고 있다.
- 프로그램이 맨 처음 실행될 때, 시스템은 주 프로그램의 시작점을 호출하기 전에(C에서 main을 호출함) 이 섹션에 있는 명령어들을 수행한다.
- Type : SHT_PROGBITS
- Flags : SHT_ALLOC + SHT_EXECINSTR

▶ .fini

- 프로세스의 종료 코드에 제공될 실행 명령어들을 담고 있다.
- Type : SHT_PROGBITS
- Flags : SHT_ALLOC + SHT_EXECINSTR

- ▶ C에서는 두개의 섹션이 필요하지 않지만, C++에서는 필요하다.

Various Sections (5/6)

▶ .interp

- 프로그램의 분석기의 경로명을 가지고 있다.
- Type : SHT_PROGBITS
- Flags : 만일 파일이 섹션을 가진 적재가능한 세그먼트를 가지고 있다면, 그 섹션의 속성은 SHF_ALLOC 비트를 포함할 것이다. 아닌 경우, 그 비트는 포함되지 않는다.

▶ .debug

- 심볼릭 디버깅에 필요한 정보들을 담고 있다. 내용은 표시되지 않는다.
- Type : SHT_PROGBITS

▶ .line

- 소스 프로그램과 기계어 코드와의 연결을 나타내는 심볼릭 디버깅을 위한 라인 넘버 정보를 가지고 있다.
- Type : SHT_PROGBITS

Various Sections (6/6)

- ▶ **.commnt**
 - 버전 제어 정보를 담고 있다.
- ▶ **.got**
 - 전역 오프셋 테이블을 가지고 있다.
 - Type : SHT_PROGBITS
- ▶ **.plt**
 - 프로시저 연결 테이블을 담고 있다.
 - Type : SHT_PROGBITS
- ▶ **.note**
 - 여분의 정보를 가지고 있다.

Section Header

String Table

- ▶ 문자열 테이블 섹션은 널문자로 끝나는데, 보통 스트링(string)이라고 부르는 문자열들을 담고 있다.
- ▶ 목적파일은 이러한 문자열들을 심볼과 섹션 이름을 표현하기 위해서 사용한다. 경우에 따라 문자열은 문자열 테이블 섹션으로의 인덱스로 참조한다.
- ▶ 첫 번째 바이트는 인덱스 0으로서, 널문자를 갖도록 정의된다.

Index	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9
0	\0	n	a	m	e	.	\0	v	a	r
10	i	a	b	l	e	\0	a	b	l	e
20	\0	\0	x	x	\0					

Index	String
0	<i>none</i>
1	<i>name.</i>
7	<i>Variable</i>
11	<i>able</i>
16	<i>able</i>
24	<i>null string</i>

String Table

Symbol Table

- ▶ 목적파일의 심볼 테이블은 프로그램 심볼의 정의와 참조에 대한 배치와 재배치 정보를 담고 있다.
- ▶ 심볼 테이블 인덱스는 이 배열의 첨자이다.
- ▶ 인덱스 0은 테이블의 첫 번째 엔트리를 가리키고 미정의된 심볼 인덱스를 표현한다.

```
typedef struct {  
    Elf32_Word      st_name;  
    Elf32_Addr     st_value;  
    Elf32_Word      st_size;  
    unsigned char   st_info;  
    unsigned char   st_other;  
    Elf32_Half      st_shndx;  
} Elf32_Sym;
```

Symbol Table

Relocation Table

- ▶ 재배치는 심볼에 대한 참조를 심볼의 실제 값과 연결시키는 과정이다.
- ▶ 재배치 파일은 어떻게 그들의 섹션 내용을 수정할 것인지 나타내는 정보를 가지고 있어야 하며, 따라서 실행파일이나 공유목적파일로 하여금 프로세스의 프로그램 이미지에 대해서 올바른 정보를 갖도록 해 주어야 한다.

```
typedef struct {  
    Elf32_Addr    r_offset;  
    Elf32_Word    r_info;  
} Elf32_Rel;  
  
typedef struct {  
    Elf32_Addr    r_offset;  
    Elf32_Word    r_info;  
    Elf32_Sword   r_addend;  
} Elf32_Rela;
```

Relocation Table

프로그램 적재 및 동적 연결

Program Header

- ▶ 실행 가능 또는 공유 가능한 목적파일의 프로그램 헤더는 여러 구조체의 배열이며, 각 구조체는 세그먼트 또는 프로그램의 실행을 준비하는데 필요한 다른 정보들을 기술한다.
- ▶ 목적파일 세그먼트는 하나 또는 다수의 섹션을 포함하며, 세그먼트 내용에서 기술된 바와 같이 저장된다.
- ▶ 프로그램 헤더는 실행 가능하거나 공유 가능한 목적파일에만 의미가 있다.

```
typedef struct {  
    Elf32_Word    p_type;  
    Elf32_Off     p_offset;  
    Elf32_Addr    p_vaddr;  
    Elf32_Addr    p_paddr;  
    Elf32_Word    p_filesz;  
    Elf32_Word    p_memsz;  
    Elf32_Word    p_flags;  
    Elf32_Word    p_align;  
} Elf32_Phdr;
```

The Types in Program Header

- ▶ 정의된 타입 값들은 아래와 같으며, 다른 값들은 미래의 사용을 위해 예약되어 있다.
- PT_LOAD
 - 이 배열 원소는 p_filesz와 p_memsz에 표시되는 적재가능한 세그먼트를 나타낸다.
 - 적재 가능한 세그먼트 엔트리들은 p_vaddr 멤버의 의해 정렬되어 오름차순으로 프로그램 헤더 테이블에 나타난다.
- PT_DYNAMIC
 - 이 배열 원소는 동적 연결 정보를 담고 있다.
- PT_INTERP
 - 이 배열 원소는 인터프리터로서 호출하기 위한 0종류 경로명의 위치와 크기를 담고 있다.
 - 이 세그먼트 타입은 실행가능한 파일에만 적용된다.

Program Header

Program Loading (1 / 3)

- ▶ 시스템이 프로세스 이미지를 생성하거나 증대시킬 때, 그것은 논리적으로 파일의 세그먼트를 가상 메모리 세그먼트로 복사한다.
- ▶ 시스템이 물리적으로 읽을 때, 파일은 프로그램의 실행 행위와 시스템 부하 등에 의존한다.
- ▶ 실제로 이러한 효과를 얻기 위해서는, 실행 가능 및 공유 가능한 목적파일들은 파일 오프셋과 가상 주소들이 페이지 크기의 나머지와 부합하는 세그먼트 이미지들을 가져야 한다.

Program Loading (2/3)

▶ 실행파일과 프로그램 헤더 세그먼트

File Offset	File	Virtual Address
0	ELF header	
Program header table		
	Other information	
0x100	Text segment	0x8048100
	...	
	0x2be00 bytes	0x8073eff
0x2bf00	Data segment	0x8074f00
	...	
	0x4e00 bytes	0x8079cff
0x30d00	Other information	
	...	

실행파일

Member	Text	Data
p_type	PT_LOAD	PT_LOAD
p_offset	0x100	0x2bf00
p_vaddr	0x8048100	0x8074f00
p_paddr	unspecified	unspecified
p_filesz	0x2be00	0x4e00
p_memsz	0x2be00	0x5e24
p_flags	PF_R + PF_X	PF_R + PF_W + PF_X
p_align	0x1000	0x1000

프로그램 헤더 세그먼트

Program Loading (3/3)

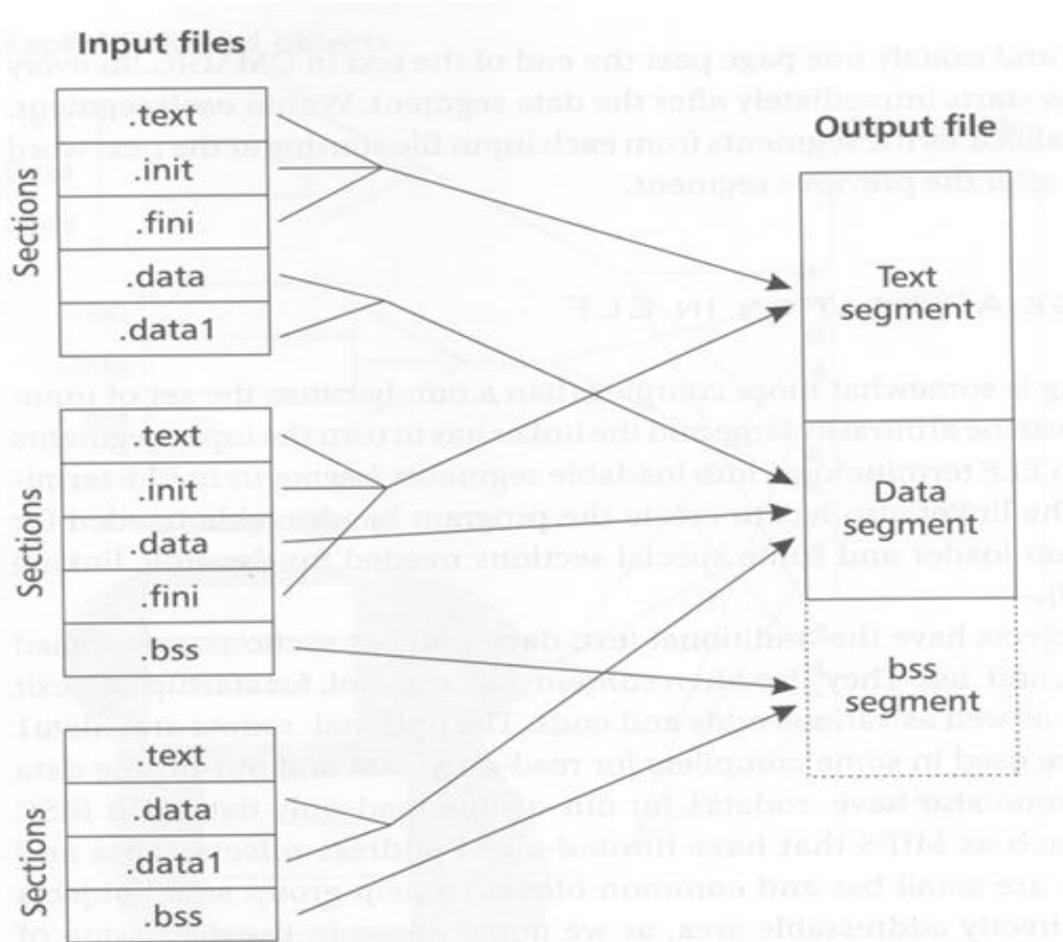
▶ 프로세스 이미지

Virtual Address	Contents	Segment
0x8048000	<i>Header padding</i> 0x100 bytes	Text
0x8048100	Text segment ...	
0x8073f00	0x2be00 bytes <i>Data padding</i> 0x100 bytes	
0x8074000	<i>Text padding</i> 0xf00 bytes	Data
0x8074f00	Data segment ...	
	0x4e00 bytes	
0x8079d00	Uninitialized data 0x1024 zero bytes	
0x807ad24	<i>Page padding</i> 0x2dc zero bytes	

▶ 공유 목적파일의 가상주소 할당 예

Source	Text	Data	Base Address
File	0x200	0x2a400	0x0
Process 1	0x80000200	0x8002a400	0x80000000
Process 2	0x80081200	0x800ab400	0x80081000
Process 3	0x900c0200	0x900ea400	0x900c0000
Process 4	0x900c6200	0x900f0400	0x900c6000

Elf Linking



Elf File Trace

(We can use the objdump and readelf command)

readelf

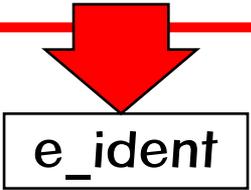
- ▶ readelf 는 BFD 라이브러리를 이용하지 않고 직접 ELF를 읽기 위한 툴이다.
- ▶ BFD (Binary File Descriptor Library : 다양한 형식의 오브젝트 파일의 호환성을 위한 GNU 프로젝트의 주 메커니즘.)
- ▶ objdump 보다 상세한 정보를 얻을 수 있다.

Objdump

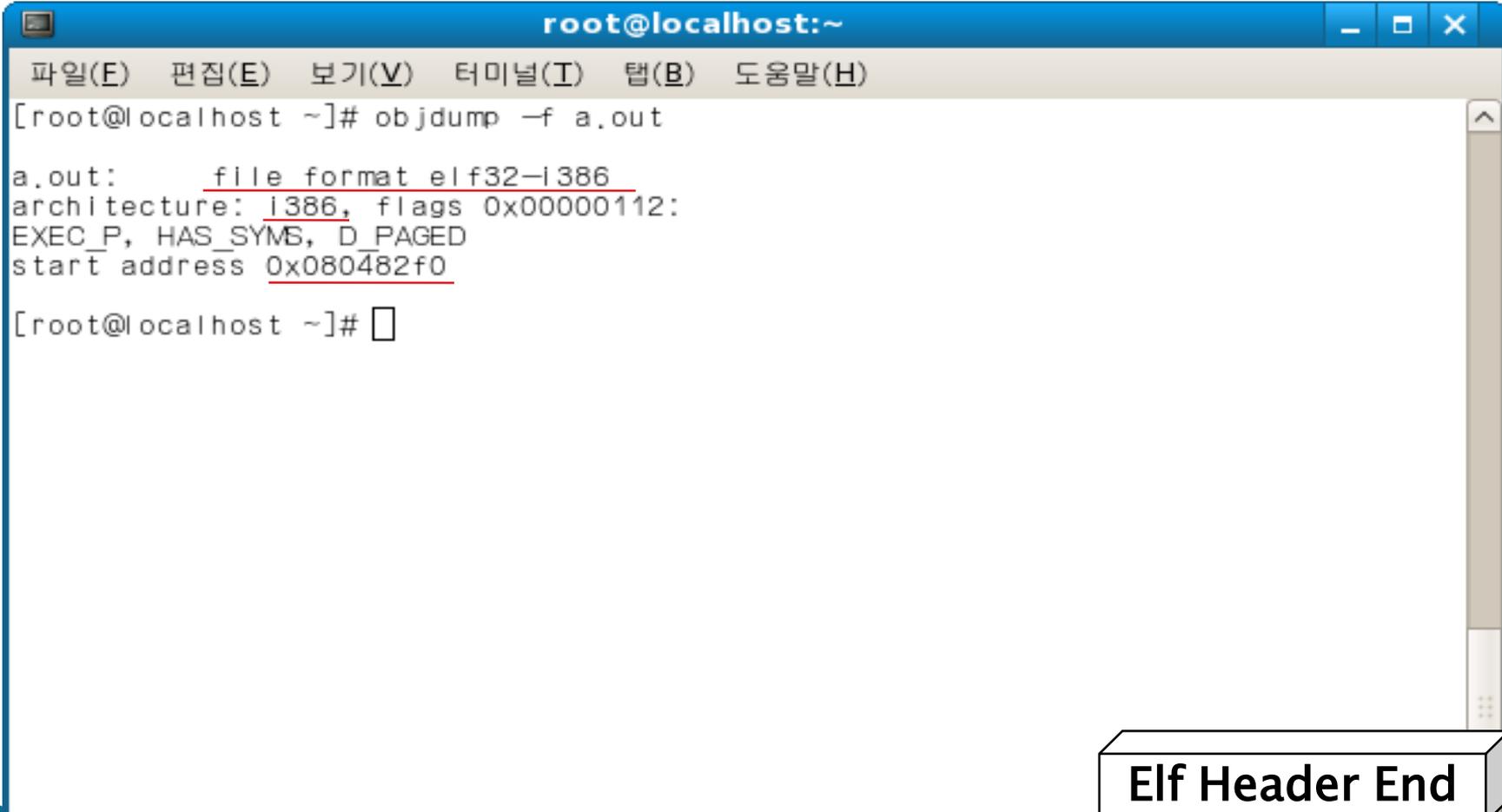
- ▶ Objdump는 자신의 소유이거나 또는 내부 라이브러리에 있는 하나 이상의 목적파일들에 대한 정보를 보여준다.
- ▶ 각 옵션들은 어떠한 정보들을 출력할지를 제어한다.
- ※ 특히 이 정보들은 주로 일반적인 프로그래머가 아닌 컴파일 툴을 개발하는 프로그래머들에게 유용할 것이다.

ELF Header Information (readelf)

```
root@localhost:~  
파일(E) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)  
[root@localhost ~]# readelf -h a.out  
ELF Header:  
Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00  
Class:           ELF32  
Data:           2's complement, little endian  
Version:        1 (current)  
OS/ABI:         UNIX - System V  
ABI Version:    0  
Type:           EXEC (Executable file)  
Machine:        Intel 80386  
Version:        0x1  
Entry point address: 0x80482f0  
Start of program headers: 52 (bytes into file)  
Start of section headers: 2744 (bytes into file)  
Flags:          0x0  
Size of this header: 52 (bytes)  
Size of program headers: 32 (bytes)  
Number of program headers: 8  
Size of section headers: 40 (bytes)  
Number of section headers: 37  
Section header string table index: 34  
[root@localhost ~]#
```



ELF Header Information (objdump)



```
root@localhost:~  
파일(E) 편집(E) 보기(V) 터미널(I) 탭(B) 도움말(H)  
[root@localhost ~]# objdump -f a.out  
a.out:      file format elf32-i386  
architecture: i386, flags 0x00000112:  
EXEC_P, HAS_SYMS, D_PAGED  
start address 0x080482f0  
[root@localhost ~]#
```

Elf Header End

Section Header - readelf (1 / 2)

```
root@localhost:~  
파일(E) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)  
[root@localhost ~]# readelf -S a.out  
There are 37 section headers, starting at offset 0xab8:  
  
Section Headers:  
[Nr] Name                Type           Addr           Off           Size          ES Flg Lk  Inf Al  
[ 0]                      NULL          00000000      000000      000000       00      0 0  0  0  
[ 1] .interp                PROGBITS      08048134      000134      000013       00      A 0  0  1  
[ 2] .note.ABI-tag          NOTE          08048148      000148      000020       00      A 0  0  4  
[ 3] .note.gnu.build-id     NOTE          08048168      000168      000024       00      A 0  0  4  
[ 4] .gnu.hash              GNU_HASH      0804818c      00018c      000020       04      A 5  0  4  
[ 5] .dynsym                DYNAMIC       080481ac      0001ac      000050       10      A 6  1  4  
[ 6] .dynstr                STRTAB        080481fc      0001fc      00004c       00      A 0  0  1  
[ 7] .gnu.version           VERSYM        08048248      000248      00000a       02      A 5  0  2  
[ 8] .gnu.version_r         VERNEED       08048254      000254      000020       00      A 6  1  4  
[ 9] .rel.dyn               REL           08048274      000274      000008       08      A 5  0  4  
[10] .rel.plt               REL           0804827c      00027c      000018       08      A 5 12  4  
[11] .init                  PROGBITS      08048294      000294      000017       00     AX 0  0  4  
[12] .plt                   PROGBITS      080482ac      0002ac      000040       04     AX 0  0  4  
[13] .text                  PROGBITS      080482f0      0002f0      0001d8       00     AX 0  0 16  
[14] .fini                  PROGBITS      080484c8      0004c8      00001c       00     AX 0  0  4  
[15] .rodata                PROGBITS      080484e4      0004e4      000019       00      A 0  0  4  
[16] .eh_frame_hdr          PROGBITS      08048500      000500      00001c       00      A 0  0  4  
[17] .eh_frame              PROGBITS      0804851c      00051c      000058       00      A 0  0  4  
[18] .ctors                 PROGBITS      08049574      000574      000008       00     WA 0  0  4
```

Section Header - readelf (2/2)

```
root@localhost:~  
파일(E) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)  
[18] .ctors          PROGBITS          08049574 000574 000008 00 WA 0 0 4  
[19] .dtors          PROGBITS          0804957c 00057c 000008 00 WA 0 0 4  
[20] .jcr           PROGBITS          08049584 000584 000004 00 WA 0 0 4  
[21] .dynamic       DYNAMIC          08049588 000588 0000c8 08 WA 6 0 4  
[22] .got           PROGBITS          08049650 000650 000004 04 WA 0 0 4  
[23] .got.plt       PROGBITS          08049654 000654 000018 04 WA 0 0 4  
[24] .data          PROGBITS          0804966c 00066c 000004 00 WA 0 0 4  
[25] .bss           NOBITS           08049670 000670 000010 00 WA 0 0 4  
[26] .comment       PROGBITS          00000000 000670 000114 00 0 0 1  
[27] .debug_aranges PROGBITS          00000000 000784 000020 00 0 0 1  
[28] .debug_pubnames PROGBITS          00000000 0007a4 000029 00 0 0 1  
[29] .debug_info     PROGBITS          00000000 0007cd 000091 00 0 0 1  
[30] .debug_abbrev   PROGBITS          00000000 00085e 000045 00 0 0 1  
[31] .debug_line     PROGBITS          00000000 0008a3 00003a 00 0 0 1  
[32] .debug_frame    PROGBITS          00000000 0008e0 00003c 00 0 0 4  
[33] .debug_loc      PROGBITS          00000000 00091c 000043 00 0 0 1  
[34] .shstrtab       STRTAB           00000000 00095f 000159 00 0 0 1  
[35] .symtab         SYMTAB           00000000 001080 0004b0 10 36 53 4  
[36] .strtab         STRTAB           00000000 001530 000212 00 0 0 1  
Key to Flags:  
W (write), A (alloc), X (execute), M (merge), S (strings)  
I (info), L (link order), G (group), x (unknown)  
O (extra OS processing required) o (OS specific), p (processor specific)  
[root@localhost ~]#
```

Section Header - objdump (1 / 3)

```
root@localhost:~  
파일(E) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)  
[root@localhost ~]# objdump -h a.out  
a.out:      file format elf32-i386  
  
Sections:  
Idx Name          Size      VMA      LMA      File off  Algn  
  0 .interp          00000013  08048134  08048134  00000134  2**0  
          CONTENTS, ALLOC, LOAD, READONLY, DATA  
  1 .note.ABI-tag    00000020  08048148  08048148  00000148  2**2  
          CONTENTS, ALLOC, LOAD, READONLY, DATA  
  2 .note.gnu.build-id 00000024  08048168  08048168  00000168  2**2  
          CONTENTS, ALLOC, LOAD, READONLY, DATA  
  3 .gnu.hash        00000020  0804818c  0804818c  0000018c  2**2  
          CONTENTS, ALLOC, LOAD, READONLY, DATA  
  4 .dynsym          00000050  080481ac  080481ac  000001ac  2**2  
          CONTENTS, ALLOC, LOAD, READONLY, DATA  
  5 .dynstr          0000004c  080481fc  080481fc  000001fc  2**0  
          CONTENTS, ALLOC, LOAD, READONLY, DATA  
  6 .gnu.version     0000000a  08048248  08048248  00000248  2**1  
          CONTENTS, ALLOC, LOAD, READONLY, DATA  
  7 .gnu.version_r   00000020  08048254  08048254  00000254  2**2  
          CONTENTS, ALLOC, LOAD, READONLY, DATA  
  8 .rel.dyn         00000008  08048274  08048274  00000274  2**2  
          CONTENTS, ALLOC, LOAD, READONLY, DATA
```

Section Header - objdump (2/3)

```
root@localhost:~  
파일(E) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)  
 9 .rel.plt      CONTENTS, ALLOC, LOAD, READONLY, DATA  
    00000018 0804827c 0804827c 0000027c 2**2  
10 .init        CONTENTS, ALLOC, LOAD, READONLY, DATA  
    00000017 08048294 08048294 00000294 2**2  
11 .plt         CONTENTS, ALLOC, LOAD, READONLY, CODE  
    00000040 080482ac 080482ac 000002ac 2**2  
12 .text       CONTENTS, ALLOC, LOAD, READONLY, CODE  
    000001d8 080482f0 080482f0 000002f0 2**4  
13 .fini       CONTENTS, ALLOC, LOAD, READONLY, CODE  
    0000001c 080484c8 080484c8 000004c8 2**2  
14 .rodata     CONTENTS, ALLOC, LOAD, READONLY, CODE  
    00000019 080484e4 080484e4 000004e4 2**2  
15 .eh_frame_hdr CONTENTS, ALLOC, LOAD, READONLY, DATA  
    0000001c 08048500 08048500 00000500 2**2  
16 .eh_frame   CONTENTS, ALLOC, LOAD, READONLY, DATA  
    00000058 0804851c 0804851c 0000051c 2**2  
17 .ctors     CONTENTS, ALLOC, LOAD, READONLY, DATA  
    00000008 08049574 08049574 00000574 2**2  
18 .dtors     CONTENTS, ALLOC, LOAD, DATA  
    00000008 0804957c 0804957c 0000057c 2**2  
19 .jcr       CONTENTS, ALLOC, LOAD, DATA  
    00000004 08049584 08049584 00000584 2**2  
20 .dynamic   CONTENTS, ALLOC, LOAD, DATA  
    000000c8 08049588 08049588 00000588 2**2
```

Section Header - objdump (3/3)

```
root@localhost:~  
파일(E) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)  
15 .eh_frame_hdr CONTENTS, ALLOC, LOAD, READONLY, DATA  
0000001c 08048500 08048500 00000500 2**2  
16 .eh_frame CONTENTS, ALLOC, LOAD, READONLY, DATA  
00000058 0804851c 0804851c 0000051c 2**2  
17 .ctors CONTENTS, ALLOC, LOAD, READONLY, DATA  
00000008 08049574 08049574 00000574 2**2  
18 .dtors CONTENTS, ALLOC, LOAD, DATA  
00000008 0804957c 0804957c 0000057c 2**2  
19 .jcr CONTENTS, ALLOC, LOAD, DATA  
00000004 08049584 08049584 00000584 2**2  
20 .dynamic CONTENTS, ALLOC, LOAD, DATA  
000000c8 08049588 08049588 00000588 2**2  
21 .got CONTENTS, ALLOC, LOAD, DATA  
00000004 08049650 08049650 00000650 2**2  
22 .got.plt CONTENTS, ALLOC, LOAD, DATA  
00000018 08049654 08049654 00000654 2**2  
23 .data CONTENTS, ALLOC, LOAD, DATA  
00000004 0804966c 0804966c 0000066c 2**2  
24 .bss CONTENTS, ALLOC, LOAD, DATA  
00000010 08049670 08049670 00000670 2**2  
25 .comment ALLOC  
00000114 00000000 00000000 00000670  
CONTENTS, READONLY  
[root@localhost ~]#
```

Section Header End

String Table - readelf

```
[35] .symtab          SYMTAB          00000000 001080 0004b0 10      36  53  4
[36] .strtab          STRTAB          00000000 001530 000212 00      0   0  1
```



-x 옵션을 이용해 지정한 섹션의 내용으로 덤프

```
root@localhost:~
파일(E) 편집(E) 보기(V) 터미널(T) 램(B) 도움말(H)
[root@localhost ~]# readelf -x36 a.out

Hex dump of section '.strtab':
0x00000000 0063616c 6c5f676d 6f6e5f73 74617274 .call_gmon_start
0x00000010 00637274 73747566 662e6300 5f5f4354 .crtstuff.c._CT
0x00000020 4f525f4c 4953545f 5f005f5f 44544f52 OR_LIST._DTOR
0x00000030 5f4c4953 545f5f00 5f5f4a43 525f4c49 LIST._JCR_LI
0x00000040 53545f5f 0064746f 725f6964 782e3536 ST._dtor_idx.56
0x00000050 34370063 6f6d706c 65746564 2e353634 47.Completed.564
0x00000060 35005f5f 646f5f67 6c6f6261 6c5f6474 5._do_global_dt
0x00000070 6f72735f 61757800 6672616d 655f6475 ors_aux.frame du
0x00000080 6d6d7900 5f5f4354 4f525f45 4e445f5f mmy._CTOR_END_
0x00000090 005f5f46 52414d45 5f454e44 5f5f005f .FRAME_END_
0x000000a0 5f4a4352 5f454e44 5f5f005f 5f646f5f _JCR_END_._do_
0x000000b0 676c6f62 616c5f63 746f7273 5f617578 global_ctors_aux
0x000000c0 00746573 742e6300 5f474c4f 42414c5f .test.c._GLOBAL_
0x000000d0 4f464653 45545f54 41424c45 5f005f5f OFFSET_TABLE_
0x000000e0 696e6974 5f617272 61795f65 6e64005f init_array_end_
0x000000f0 5f696e69 745f6172 7261795f 73746172 init_array_star
0x00000100 74005f44 594e414d 49430064 6174615f t._DYNAMIC.data
0x00000110 73746172 74005f5f 6c696263 5f637375 start._libc_csu_
0x00000120 5f66696e 69005f73 74617274 005f5f67 _fini._start._g
0x00000130 6d6f6e5f 73746172 745f5f00 5f4a765f mon_start._Jv_
0x00000140 52656769 73746572 436c6173 73657300 RegisterClasses_
0x00000150 5f66705f 6877005f 66696e69 005f5f6c fp_hw._fini._i
0x00000160 6962635f 73746172 745f6d61 696e4040 libc_start_main@@
0x00000170 474c4942 435f322e 30005f49 4f5f7374 GLIBC_2.0._IO_st
0x00000180 64696e5f 75736564 005f5f64 6174615f din_used._data
0x00000190 73746172 74005f5f 64736f5f 68616e64 start._dso_hand
0x000001a0 6c65005f 5f44544f 525f454e 445f5f00 le._DTOR_END_
0x000001b0 5f5f6c69 62635f63 73755f69 6e697400 _libc_csu_init_
0x000001c0 7072696e 74664040 474c4942 435f322e printf@@GLIBC_2_
0x000001d0 30007979 005f5f62 73735f73 74617274 0.yy._bss_start
0x000001e0 005f656e 64005f65 64617461 005f5f69 .end._data
0x000001f0 3638362e 6765745f 70635f74 68756e6b 686.get_data
0x00000200 2e627800 6d61696e 005f696e 69740078 .bx.ma
0x00000210 7800 x.
```

String Table End

Symbol Table - readelf (1 / 2)

```
root@localhost:~  
파일(E) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)  
[root@localhost ~]# readelf -s a.out  
Symbol table '.dynsym' contains 5 entries:  
  Num:      Value      Size  Type      Bind      Vis      Ndx  Name  
  0: 00000000          0 NOTYPE    LOCAL     DEFAULT   UND  
  1: 00000000          0 NOTYPE    WEAK      DEFAULT   UND __gmon_start__  
  2: 00000000        438 FUNC      GLOBAL    DEFAULT   UND __libc_start_main@GLIBC_2.0 (2)  
  3: 00000000         57 FUNC      GLOBAL    DEFAULT   UND printf@GLIBC_2.0 (2)  
  4: 080484e8          4 OBJECT    GLOBAL    DEFAULT   15  _IO_stdin_used  
Symbol table '.symtab' contains 75 entries:  
  Num:      Value      Size  Type      Bind      Vis      Ndx  Name  
  0: 00000000          0 NOTYPE    LOCAL     DEFAULT   UND  
  1: 08048134          0 SECTION  LOCAL     DEFAULT    1  
  2: 08048148          0 SECTION  LOCAL     DEFAULT    2  
  3: 08048168          0 SECTION  LOCAL     DEFAULT    3  
  4: 0804818c          0 SECTION  LOCAL     DEFAULT    4  
  5: 080481ac          0 SECTION  LOCAL     DEFAULT    5  
  6: 080481fc          0 SECTION  LOCAL     DEFAULT    6  
  7: 08048248          0 SECTION  LOCAL     DEFAULT    7  
  8: 08048254          0 SECTION  LOCAL     DEFAULT    8  
  9: 08048274          0 SECTION  LOCAL     DEFAULT    9  
 10: 0804827c          0 SECTION  LOCAL     DEFAULT   10  
 11: 08048294          0 SECTION  LOCAL     DEFAULT   11  
 12: 080482ac          0 SECTION  LOCAL     DEFAULT   12  
 13: 080482f0          0 SECTION  LOCAL     DEFAULT   13  
 14: 080484c8          0 SECTION  LOCAL     DEFAULT   14  
 15: 080484e4          0 SECTION  LOCAL     DEFAULT   15  
 16: 08048500          0 SECTION  LOCAL     DEFAULT   16  
 17: 0804851c          0 SECTION  LOCAL     DEFAULT   17  
 18: 08049574          0 SECTION  LOCAL     DEFAULT   18  
 19: 0804957c          0 SECTION  LOCAL     DEFAULT   19  
 20: 08049584          0 SECTION  LOCAL     DEFAULT   20  
 21: 08049588          0 SECTION  LOCAL     DEFAULT   21  
 22: 08049650          0 SECTION  LOCAL     DEFAULT   22  
 23: 08049654          0 SECTION  LOCAL     DEFAULT   23  
 24: 0804966c          0 SECTION  LOCAL     DEFAULT   24  
 25: 08049670          0 SECTION  LOCAL     DEFAULT   25  
 26: 00000000          0 SECTION  LOCAL     DEFAULT   26
```

Symbol Table - readelf (2/2)

```
root@localhost:~  
파일(E) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)  
37: 0804957c 0 OBJECT LOCAL DEFAULT 19 __DTOR_LIST__  
38: 08049584 0 OBJECT LOCAL DEFAULT 20 __JCR_LIST__  
39: 08049670 4 OBJECT LOCAL DEFAULT 25 dtor_idx.5647  
40: 08049674 1 OBJECT LOCAL DEFAULT 25 completed.5645  
41: 08048340 0 FUNC LOCAL DEFAULT 13 __do_global_dtors_aux  
42: 080483a0 0 FUNC LOCAL DEFAULT 13 frame_dummy  
43: 00000000 0 FILE LOCAL DEFAULT ABS crtstuff.c  
44: 08049578 0 OBJECT LOCAL DEFAULT 18 __CTOR_END__  
45: 08048570 0 OBJECT LOCAL DEFAULT 17 __FRAME_END__  
46: 08049584 0 OBJECT LOCAL DEFAULT 20 __JCR_END__  
47: 080484a0 0 FUNC LOCAL DEFAULT 13 __do_global_ctors_aux  
48: 00000000 0 FILE LOCAL DEFAULT ABS test.c  
49: 08049654 0 OBJECT LOCAL HIDDEN 23 __GLOBAL_OFFSET_TABLE__  
50: 08049574 0 NOTYPE LOCAL HIDDEN 18 __init_array_end  
51: 08049574 0 NOTYPE LOCAL HIDDEN 18 __init_array_start  
52: 08049588 0 OBJECT LOCAL HIDDEN 21 __DYNAMIC  
53: 0804966c 0 NOTYPE WEAK DEFAULT 24 data_start  
54: 08048420 5 FUNC GLOBAL DEFAULT 13 __libc_csu_fini  
55: 080482f0 0 FUNC GLOBAL DEFAULT 13 __start__  
56: 00000000 0 NOTYPE WEAK DEFAULT UND __gmon_start__  
57: 00000000 0 NOTYPE WEAK DEFAULT UND __Jv_RegisterClasses  
58: 080484e4 4 OBJECT GLOBAL DEFAULT 15 __fp_hw  
59: 080484c8 0 FUNC GLOBAL DEFAULT 14 __fini  
60: 00000000 438 FUNC GLOBAL DEFAULT UND __libc_start_main@@GLIBC_  
61: 080484e8 4 OBJECT GLOBAL DEFAULT 15 __IO_stdin_used  
62: 0804966c 0 NOTYPE GLOBAL DEFAULT 24 data_start  
63: 080484ec 0 OBJECT GLOBAL HIDDEN 15 __dso_handle  
64: 08049580 0 OBJECT GLOBAL HIDDEN 19 __DTOR_END__  
65: 08048430 105 FUNC GLOBAL DEFAULT 13 __libc_csu_init  
66: 00000000 57 FUNC GLOBAL DEFAULT UND printf@@GLIBC_2.0  
67: 08049678 4 OBJECT GLOBAL DEFAULT 25 yy  
68: 08049670 0 NOTYPE GLOBAL DEFAULT ABS __bss_start  
69: 08049680 0 NOTYPE GLOBAL DEFAULT ABS __end__  
70: 08049670 0 NOTYPE GLOBAL DEFAULT ABS __edata__  
71: 08048499 0 FUNC GLOBAL HIDDEN 13 __i686.get_pc_thunk.bx  
72: 080483c4 77 FUNC GLOBAL DEFAULT 13 main  
73: 08048294 0 FUNC GLOBAL DEFAULT 11 __init__  
74: 0804967c 4 OBJECT GLOBAL DEFAULT 25 xx  
[root@localhost ~]#
```

Symbol Table - objdump (1 / 3)

```
root@localhost:~  
파일(E) 편집(E) 보기(V) 터미널(I) 탭(B) 도움말(H)  
[root@localhost ~]# objdump -t a.out  
a.out:      file format elf32-i386  
  
SYMBOL TABLE:  
08048134 | d  .interp      00000000      .interp  
08048148 | d  .note.ABI-tag 00000000      .note.ABI-tag  
08048168 | d  .note.gnu.build-id 00000000      .note.gnu.build-id  
0804818c | d  .gnu.hash     00000000      .gnu.hash  
080481ac | d  .dynsym       00000000      .dynsym  
080481fc | d  .dynstr       00000000      .dynstr  
08048248 | d  .gnu.version  00000000      .gnu.version  
08048254 | d  .gnu.version_r 00000000      .gnu.version_r  
08048274 | d  .rel.dyn      00000000      .rel.dyn  
0804827c | d  .rel.plt     00000000      .rel.plt  
08048294 | d  .init         00000000      .init  
080482ac | d  .plt         00000000      .plt  
080482f0 | d  .text        00000000      .text  
080484c8 | d  .fini        00000000      .fini  
080484e4 | d  .rodata      00000000      .rodata  
08048500 | d  .eh_frame_hdr 00000000      .eh_frame_hdr  
0804851c | d  .eh_frame     00000000      .eh_frame  
08049574 | d  .ctors       00000000      .ctors  
0804957c | d  .dtors       00000000      .dtors
```

Symbol Table - objdump (2/3)

```
root@localhost:~  
파일(E) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)  
0804957c | d .dtors 00000000 .dtors  
08049584 | d .jcr 00000000 .jcr  
08049588 | d .dynamic 00000000 .dynamic  
08049650 | d .got 00000000 .got  
08049654 | d .got.plt 00000000 .got.plt  
0804966c | d .data 00000000 .data  
08049670 | d .bss 00000000 .bss  
00000000 | d .comment 00000000 .comment  
08048314 | F .text 00000000 call_gmon_start  
00000000 | df *ABS* 00000000 crtstuff.c  
08049574 | O .ctors 00000000 __CTOR_LIST__  
0804957c | O .dtors 00000000 __DTOR_LIST__  
08049584 | O .jcr 00000000 __JCR_LIST__  
08049670 | O .bss 00000004 dtor_idx.5647  
08049674 | O .bss 00000001 completed.5645  
08048340 | F .text 00000000 __do_global_dtors_aux  
080483a0 | F .text 00000000 frame_dummy  
00000000 | df *ABS* 00000000 crtstuff.c  
08049578 | O .ctors 00000000 __CTOR_END__  
08048570 | O .eh_frame 00000000 __FRAME_END__  
08049584 | O .jcr 00000000 __JCR_END__  
080484a0 | F .text 00000000 __do_global_ctors_aux  
00000000 | df *ABS* 00000000 test.c  
08049654 | O .got.plt 00000000 .hidden GLOBAL_OFFSET_TAB
```

Symbol Table - objdump (3/3)

```
root@localhost:~  
파일(E) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)  
08048420 g F .text 00000005 __libc_csu_fini  
080482f0 g F .text 00000000 _start  
00000000 w *UND* 00000000 __gmon_start__  
00000000 w *UND* 00000000 _Jv_RegisterClasses  
080484e4 g O .rodata 00000004 __fp_hw  
080484c8 g F .fini 00000000 _fini  
00000000 F *UND* 000001b6 __libc_start_main@@GLIBC_2.0  
080484e8 g O .rodata 00000004 _IO_stdin_used  
0804966c g .data 00000000 __data_start  
080484ec g O .rodata 00000000 .hidden __dso_handle  
08049580 g O .dtors 00000000 .hidden __DTOR_END__  
08048430 g F .text 00000069 __libc_csu_init  
00000000 F *UND* 00000039 printf@@GLIBC_2.0  
08049678 g O .bss 00000004 yy  
08049670 g *ABS* 00000000 __bss_start  
08049680 g *ABS* 00000000 _end  
08049670 g *ABS* 00000000 _edata  
08048499 g F .text 00000000 .hidden __l686.get_pc_thunk.bx  
080483c4 g F .text 0000004d main  
08048294 g F .init 00000000 _init  
0804967c g O .bss 00000004 xx  
  
[root@localhost ~]#
```

Symbol Table End

Relocation Table – readelf

```
root@localhost:~  
파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)  
[root@localhost ~]# readelf -r a.out  
  
Relocation section '.rel.dyn' at offset 0x274 contains 1 entries:  
  Offset      Info      Type           Sym.Value    Sym. Name  
08049650  00000106 R_386_GLOB_DAT 00000000    __gmon_start__  
  
Relocation section '.rel.plt' at offset 0x27c contains 3 entries:  
  Offset      Info      Type           Sym.Value    Sym. Name  
08049660  00000107 R_386_JUMP_SLOT 00000000    __gmon_start__  
08049664  00000207 R_386_JUMP_SLOT 00000000    __libc_start_main  
08049668  00000307 R_386_JUMP_SLOT 00000000    printf  
[root@localhost ~]#
```

Relocation Table End

Program Header - readelf

```
root@localhost:~  
파일(E) 편집(E) 보기(V) 터미널(I) 탭(B) 도움말(H)  
[root@localhost ~]# readelf -i a.out  
Elf file type is EXEC (Executable file)  
Entry point 0x80482f0  
There are 8 program headers, starting at offset 52  
Program Headers:  
Type           Offset           VirtAddr         PhysAddr         FileSiz MemSiz  Flg Align  
PHDR           0x000034         0x08048034      0x08048034      0x00100 0x00100 R E  0x4  
INTERP        0x000134         0x08048134      0x08048134      0x00013 0x00013 R    0x1  
              [Requesting program interpreter: /lib/ld-linux.so.2]  
LOAD          0x000000         0x08048000      0x08048000      0x00574 0x00574 R E  0x1000  
LOAD          0x000574         0x08049574      0x08049574      0x000fc 0x0010c RW  0x1000  
DYNAMIC       0x000588         0x08049588      0x08049588      0x000c8 0x000c8 RW  0x4  
NOTE          0x000148         0x08048148      0x08048148      0x00044 0x00044 R    0x4  
GNU_EH_FRAME  0x000500         0x08048500      0x08048500      0x0001c 0x0001c R    0x4  
GNU_STACK     0x000000         0x00000000      0x00000000      0x00000 0x00000 RW  0x4  
  
Section to Segment mapping:  
Segment Sections...  
00  
01      .interp  
02      .interp .note.ABI-tag .note.gnu.build-id .gnu.hash .dynsym .dynst  
r .gnu.version .gnu.version_r .rel.dyn .rel.plt .init .plt .text .fini .rod  
ata .eh_frame_hdr .eh_frame  
03      .ctors .dtors .jcr .dynamic .got .got.plt .data .bss  
04      .dynamic  
05      .note.ABI-tag .note.gnu.build-id  
06      .eh_frame_hdr  
07  
[root@localhost ~]# █
```

Program Header - objdump

```
root@localhost:~  
파일(E) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)  
[root@localhost ~]# objdump -p a.out  
a.out:      file format elf32-i386  
  
Program Header:  
  PHDR off      0x00000034 vaddr 0x08048034 paddr 0x08048034 align 2**2  
      filesz 0x00000100 memsz 0x00000100 flags r-x  
  INTERP off     0x00000134 vaddr 0x08048134 paddr 0x08048134 align 2**0  
      filesz 0x00000013 memsz 0x00000013 flags r--  
  LOAD  off     0x00000000 vaddr 0x08048000 paddr 0x08048000 align 2**12  
      filesz 0x00000574 memsz 0x00000574 flags r-x  
  LOAD  off     0x00000574 vaddr 0x08049574 paddr 0x08049574 align 2**12  
      filesz 0x000000fc memsz 0x0000010c flags rw-  
  DYNAMIC off    0x00000588 vaddr 0x08049588 paddr 0x08049588 align 2**2  
      filesz 0x000000c8 memsz 0x000000c8 flags rw-  
  NOTE  off     0x00000148 vaddr 0x08048148 paddr 0x08048148 align 2**2  
      filesz 0x00000044 memsz 0x00000044 flags r--  
  EH_FRAME off    0x00000500 vaddr 0x08048500 paddr 0x08048500 align 2**2  
      filesz 0x0000001c memsz 0x0000001c flags r--  
  STACK off     0x00000000 vaddr 0x00000000 paddr 0x00000000 align 2**2  
      filesz 0x00000000 memsz 0x00000000 flags rw-  
  
Dynamic Section:  
  NEEDED      libc.so.6
```

Program Table End

Q & A