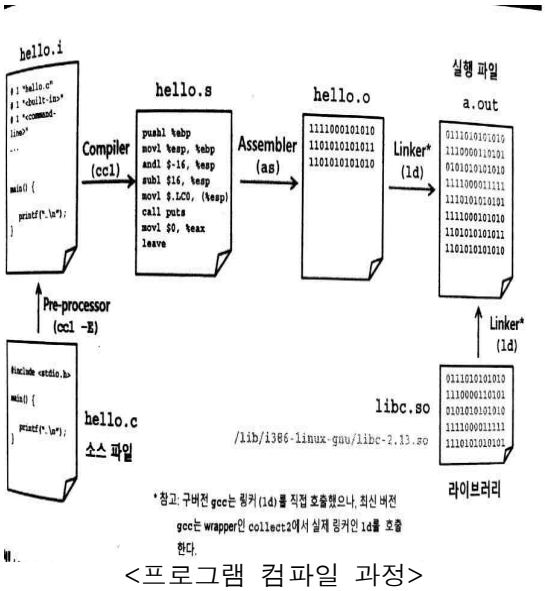
Chapter 2 : 프로그램 동작 원리

2.1 컴파일과 ELF 파일구조



<프로그램 컴파일 과정>

Object File 종류

Relocatable Object File (.O)

- Compile-Time에 생성되는 바이너리 코드와 데이터로 이루어진 파일
- 다른 재배치 가능한 오브젝트 파일과 통합이 가능하다.

Executable Object File (실행파일)

- 메모리에 직접 로딩되어 실행될 수 있는 바이너리 코드와 데이터로 이루어진 파일

Shared Object File (.so)

- Load-time 이나 Run-time에 동적으로 메모리에 로드되고 링크될 수 있는 특수한 타입의 재배치 가능한 오브젝트 파일

Object File 포맷 종류

COFF

System V 계열 초기 UNIX사용

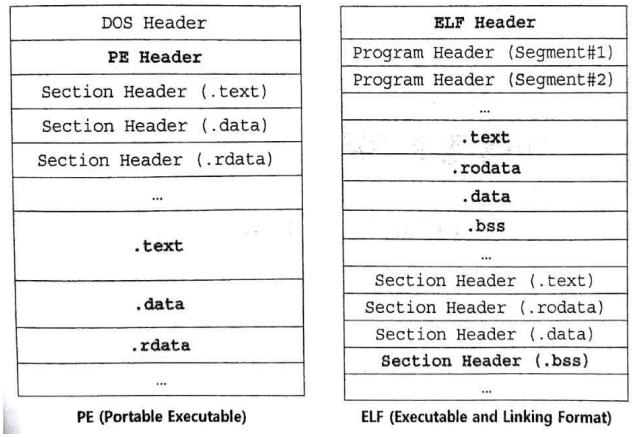
PE(Portable Executable)

Windows NT에서 사용하는 COFF의 변종

ELF(Executable and Linking Format)

Linux, Solaris와 같은 대부분의 UNIX 사용

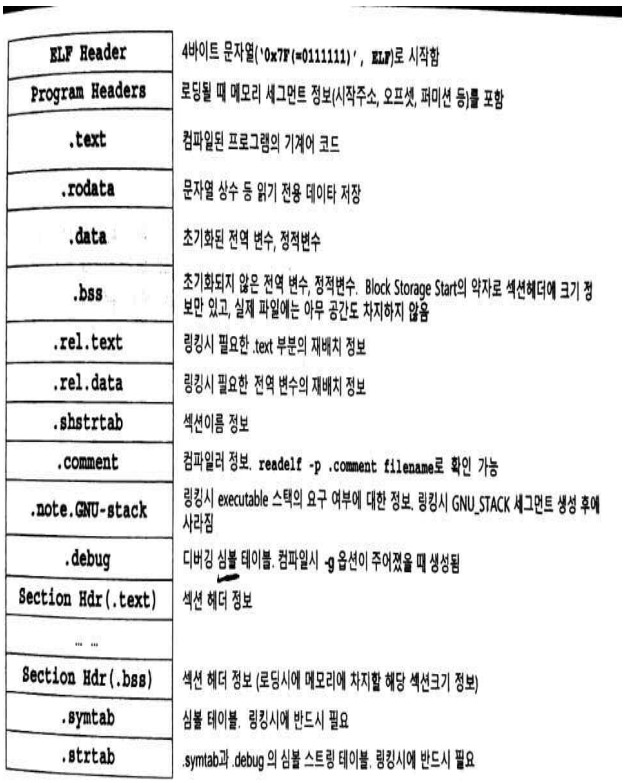
Object File 포맷 구조



Object File 분석 도구

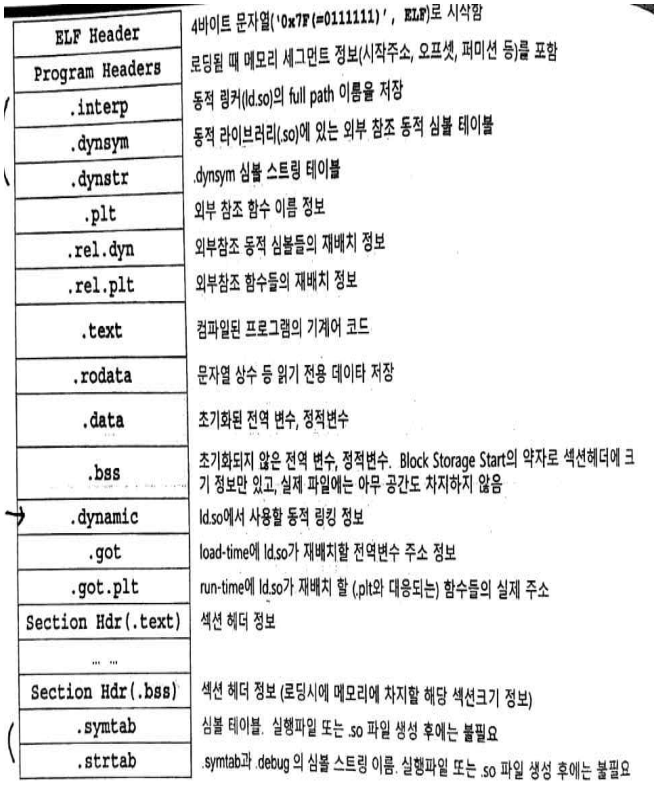
- readelf : ELF 포맷 구조 출력
- objdump : ELF 파일 정보를 출력
  disassemble 가능
- ar : 정적 라이브러리 생성
- strip : 심볼 테이블 정보 삭제
- strings : 파일내의 printable string 출력
- ldd : 공유 라이브러리 목록 출력

오브젝트 파일(.O) 구조



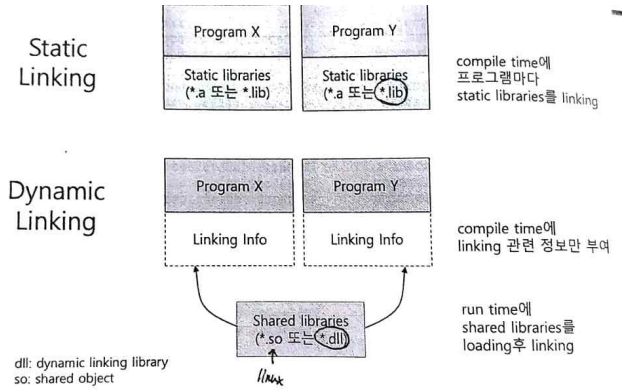
ELF Header	4바이트 문자열('0x7F(=0111111)', ELF)로 시작함
Program Headers	로딩될 때 메모리 세그먼트 정보(시작주소, 오프셋, 퍼미션 등)를 포함
.text	컴파일된 프로그램의 기계어 코드
.rodata	문자열 상수 등 읽기 전용 데이타 저장
.data	초기화된 전역 변수, 정적변수
.bss	초기화되지 않은 전역 변수, 정적변수. Block Storage Start의 약자로 섹션헤더에 크기 정보만 있고, 실제 파일에는 아무 공간도 차지하지 않음
.rel.text	링킹시 필요한 .text 부분의 재배치 정보
.rel.data	링킹시 필요한 전역 변수의 재배치 정보
.shstrtab	섹션이름 정보
.comment	컴파일러 정보. readelf -p .comment filename로 확인 가능
.note.GNU-stack	링킹시 executable 스택의 요구 여부에 대한 정보. 링킹시 GNU_STACK 세그먼트 생성 후에 사라짐
.debug	디버깅 심볼 테이블. 컴파일시 -g 옵션이 주어졌을 때 생성됨
Section Hdr(.text)	섹션 헤더 정보
... ...	
Section Hdr(.bss)	섹션 헤더 정보 (로딩시에 메모리에 차지할 해당 섹션크기 정보)
.symtab	심볼 테이블. 링킹시에 반드시 필요
.strtab	.symtab과 .debug 의 심볼 스트링 테이블. 링킹시에 반드시 필요

실행 파일(.out) 구조



2.2 심볼과 링킹

Static VS Dynamic Linking



Symbol 종류

모든 오브젝트 파일은 심볼 테이블을 가지고 있다.

- Global Symbol(다른 모듈에서 참조 가능)
  일반함수,전역변수,외부함수,외부변수
  - Strong Symbol : 함수,초기화된 전역변수
  - Weak Symbol : 초기화되지 않은 전역변수
- Local Symbol(해당 모듈에서만 참조)
  정적함수,정적변수,파일명

전역변수 sum앞에 static붙이면 Global Symbol에서 Local Symbol로 변경되어 Linking Error가 발생한다.

Symbol Resolution(심볼 해석 규칙)

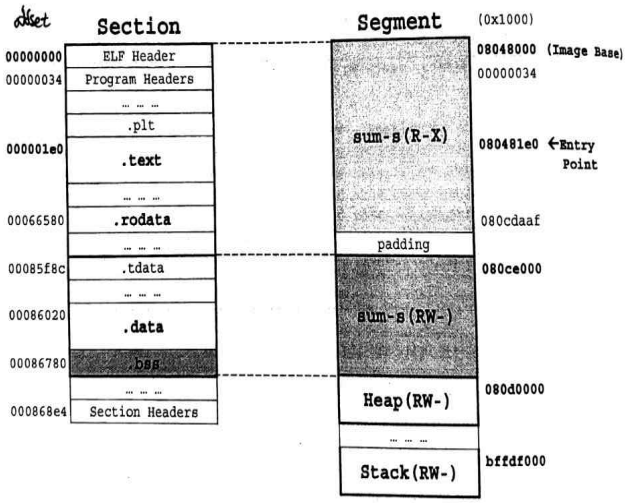
- 여러개의 Strong Symbol은 허락되지 않는다.
- Strong Symbol과 Weak Symbol이 동시에 주어질 때, Strong Symbol을 선택한다.
- 여러 개의 Weak Symbol이 있을 경우 아무거나 선택한다.

2.3 정적 링킹과 로딩

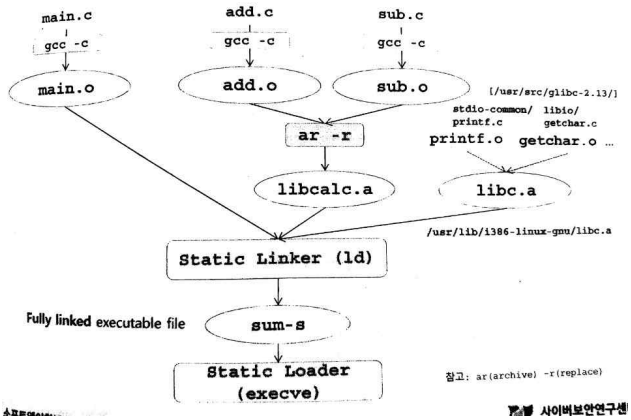
Static Loader의 역할

로딩(Loading) + (Relocation)

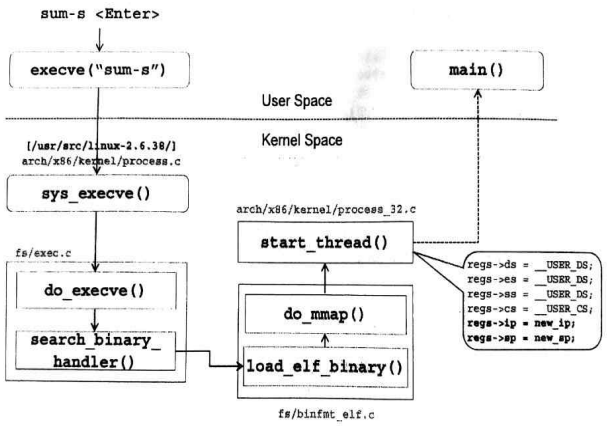
Static Loading



Static Library 생성 후 링킹

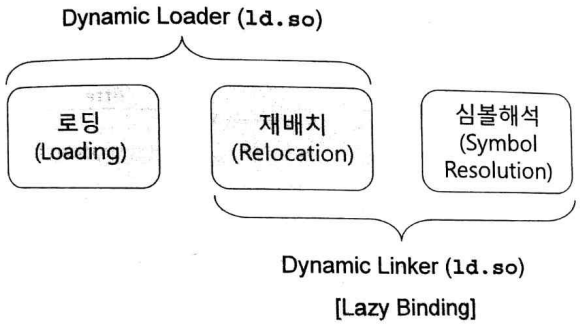


실제 Static Loading 과정



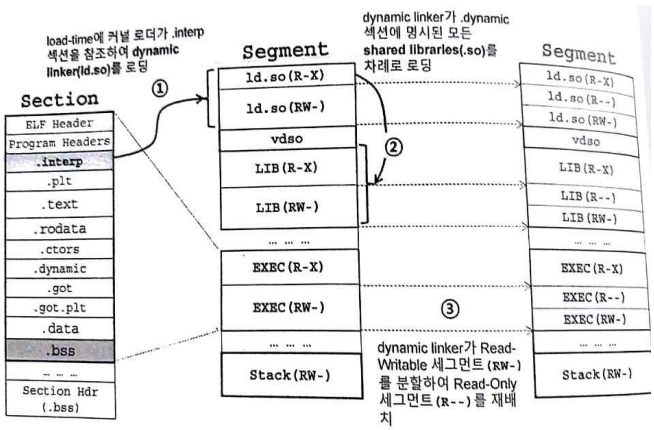
2.4 동적 로딩

Dynamic Loading and Linking

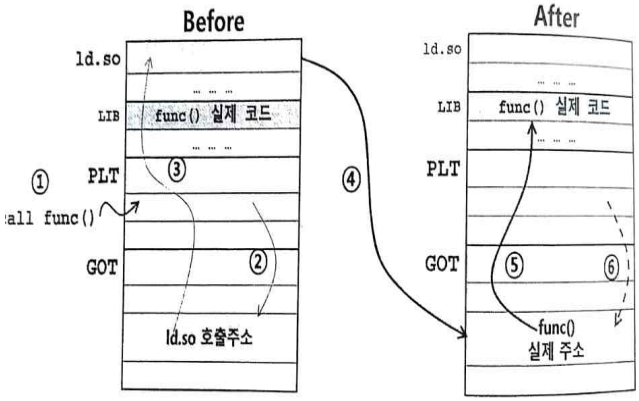


Dynamic은 Loader와 Linker의 일의 차이가 모호하다.

Dynamic Loading



Dynamic Linking(Lazy Binding)

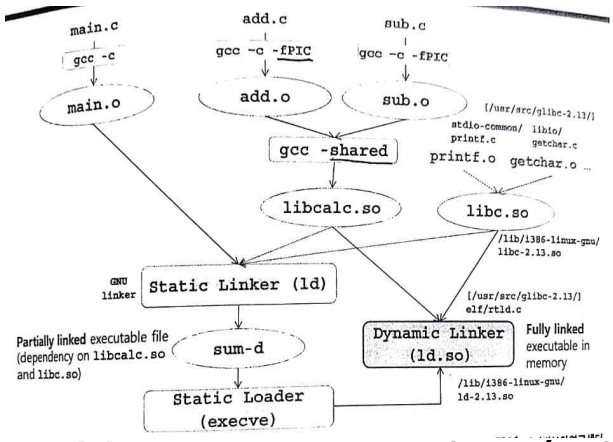


함수 호출시 함수 이름만 존재하는 PLT를 참조하면 해당 함수의 실제 주소가 들어 있는 GOT로 연결 되어 해당 함수 주소를 찾아 가게 된다.

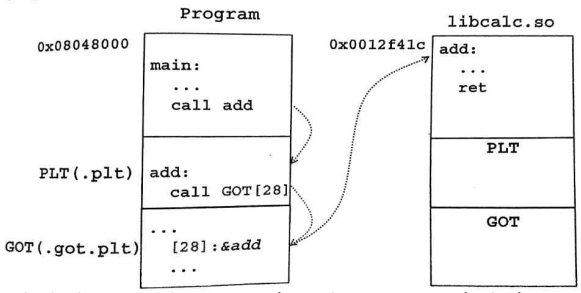
Lazy Binding

정적 링킹 시에 GOT에는 dynamic linker(ld.so) 호출 주소가 저장되어 있고, 해당 함수가 처음 호출 될 때 ld.so가 실제 함수 주소를 찾아서 GOT 값을 업데이트함

Shared Library 생성 후 링킹

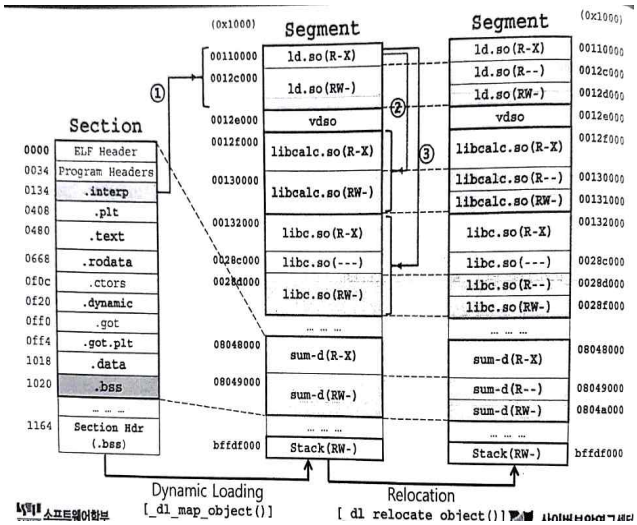


Position Independent Code(PIC)



임의의 주소에 로드 가능한 코드를 말한다.
데이터 접근이나 점프는 상대 주소로 수행한다.
대표적인 예 : Shared Library(*.so, *.dll)

Dynamic Loading



Dynamic Linking : PLT와 GOT

PLT(Procedure Linkage Table) : .plt 섹션

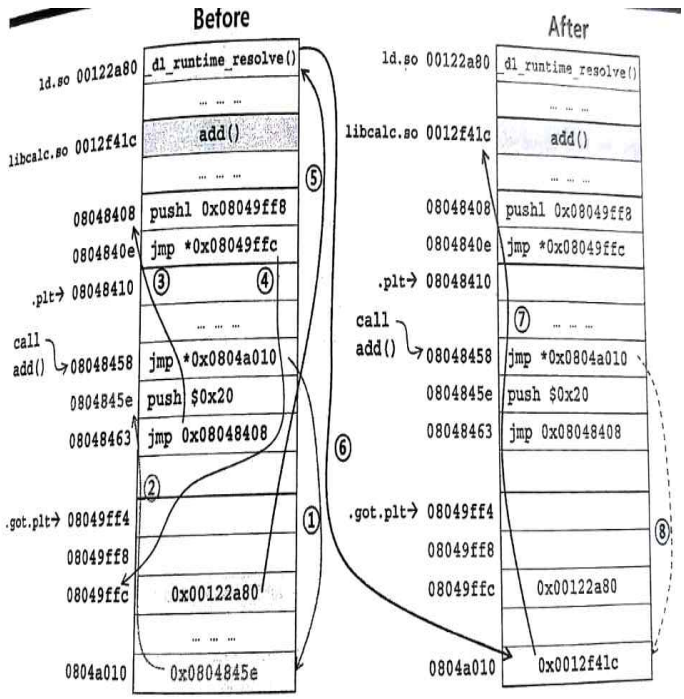
- 참조할 외부 함수들의 이름을 가지고 있음
- 다른 외부 라이브러리에 위치한 함수를 호출 할 경우 PLT를 사용함

GOT(Global Offset Table) :.got.plt 섹션

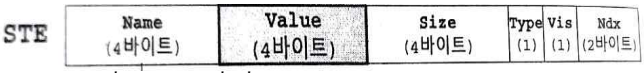
- PLT에 있는 함수들의 실제 주소를 가지고 있음
- PLT가 어떤 외부 함수를 호출할 때 이 GOT를 참조해서 해당 주소로 점프함

* got 섹션은 재배치할 전역변수들의 주소들로 load-time에 결정됨

Dynamic Linking - Lazy Binding



Symbol Table Entry 구조체



Entry 크기 : 16 바이트

Name : .dynstr의 인덱스이지 Human Readable한 String이 아니다

Symbol Resolution

```
csec@syssec:~/source/chap2$ readelf -x .dynsym libcalc.so

Hex dump of section '.dynsym':
0:   0x0000015c 00000000 00000000 00000000 00000000 ................
5:   0x000001ac 55000000 0c200000 00000000 1000f1ff U.... ..........
6:   0x000001bc 3f000000 14200000 04000000 11001500 ?.... ..........
7:   0x000001cc 43000000 1c040000 0e000000 12000b00 C...............
```

00000043	0000041c	0000000e	00	0b	0012

```
csec@syssec:~/source/chap2$ readelf -p .dynstr libcalc.so

String dump of section '.dynstr':
  [     3f]  sum
→ [     43]  add
  [     47]  sub
```

Little Endian 이기 때문에 거꾸로 읽어주어야 한다.
1c040000 -> 1c 04 00 00 -> 00 00 04 1c

공유 라이브러리의 .dynsym에서 add심볼을 찾고 strcmp를 통해 비교하는 방식으로 진행된다.