<GOT와 PLT>

1. GOT, PLT

- PLT (Procedure Linkage Table) : 외부 프로시저를 연결해주는 테이블, PLT를 통해 다른 라이브러리에 있는 프로시저를 호출해 사용할 수 있음

- GOT (Global Offset Table) : PLT가 참조하는 테이블, 프로시저들의 주소가 들어있음

2. 왜 GOT와 PLT를 사용하는가

- "함수 호출 시, GOT로 점프하는데 GOT에는 함수의 실제 주소가 쓰여있음, 첫 번째 호출이라면 GOT는 함수의 주소를 가지고 있지 않고 어떤 과정을 거쳐 주소를 알아냄. 두 번째 호출 땐 첫 번째 호출 때 알아낸 주소로 바로 점프

-->왜??

1) 링커(Linker)

예를 들어 printf함수를 호출하는 코드를 작성 시에, include한 헤더파일에는 printf의 선언이 존재

소스파일을 실행파일로 만들기 위해서 컴파일 과정을 거침

컴파일을 통해 오브젝트 파일이 생성됨

(하지만 오브젝트 파일 자체로는 실행이 가능하지 않음, printf의 구현 코드를 모름)

오브젝트 파일을 실행 가능하게 만들기 위해서는 printf의 실행 코드를 찾아서 오브젝트 파일과 연결시켜야 함

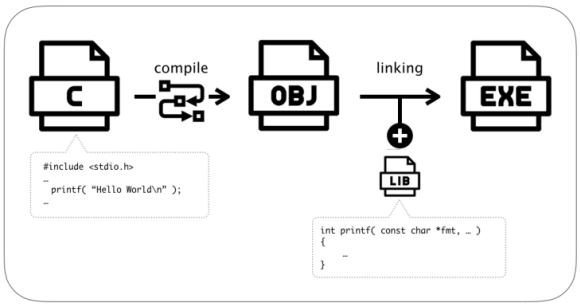
(printf의 실행코드 = printf의 구현코드를 컴파일한 오브젝트 파일)

-->이러한 오브젝트 파일들(각각의 실행코드들)이 모여있는 곳이 바로 라이브러리

라이브러리 등 필요한 오브젝트 파일들의 집합과 연결시키는

작업이 바로 링킹

링킹까지 마치면 최종적인 실행파일이 생김



2) 링크를 하는 방법

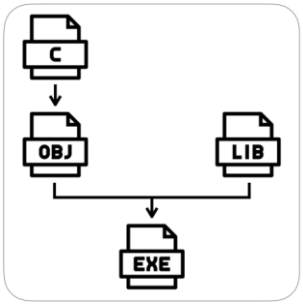
Static과 Dynamic방식

- Static Link : 파일 생성시 라이브러리 내용을 포함한 실행 파일을 생성

< Static compile 하기 >

```
$ gcc -o testfile_static testfile.c -static
```

```
$ file testfile_static
testfile_static: ELF 32-bit LSB  executable, Intel 80386, version 1 (GNU/Linux), statically linked, for GNU/Linux 2.6.24, BuildID[sha1]=b41e882489b2eca44b30c1868fc212f8b6edbf4d, not stripped
```



실행 파일 안에 모든 코드가 포함되기 때문에 라이브러리 연동 과정이 따로 필요 없고, 한 번 생성한 파일에 대해서 필요한 라이브러리를 따로 관리하지 않아도 되기 때문에 편하다는 장점

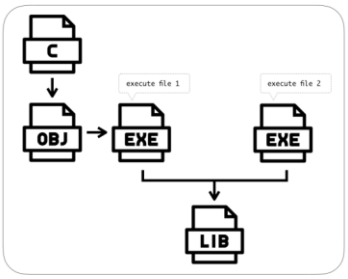
하지만 파일 크기가 커지고, 동일한 라이브러리를 사용하더라도 해당 라이브러리를 사용하는 모든 프로그램들은 각자 라이브러리의 내용을 메모리에 매핑 시켜야 함

- Dynamic Link : 라이브러리를 하나의 메모리 공간에 매핑하고 여러 프로그램에서 공유하여 사용

< Dynamic compile 하기 >

$ gcc -o testfile_dynamic testfile.c

```
$ file testfile_dynamic
testfile_dynamic: ELF 32-bit LSB  executable, Intel 80386, version 1 (
SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.24, Bu
ildID[sha1]=7b0e483669cc38f3c1fc35e3e413458b61e71a5a, not stripped
```



실행파일 안에 라이브러리 코드를 포함하지 않으므로, 파일 크기가 Static에 비해 훨씬 작아짐

실행 시에도 상대적으로 적은 메모리를 차지, 또한 라이브러리를 따로 업데이트 할 수 있기 때문에 유연한 방법

하지만 실행파일이 라이브러리에 의존해야 하기 대문에 라이브러리가 없으면 실행할 수 없음

*Dynamic compile시에 아무런 옵션도 주지 않는다면, 자동으로 Dynamic방식으로 컴파일

3) PLT와 GOT

Dynamic Link방식으로 컴파일 했을 때 PLT와 GOT를 사용

Static Link방식으로 컴파일 하면 라이브러리가 프로그램 내부에 있기 때문에 함수의 주소를 알아오는 과정이 필요하지 않지만, Dynamic Link방식으로 컴파일 하면 라이브러리가 프로그램 외부에 있기 때문에 함수의 주소를 알아오는 과정이 필요함

Dynamic Link방식으로 프로그램이 만들어지면 함수 호출 시 PLT를 참조하게 됨, PLT에서는 GOT로 점프를 하는데, GOT에 라이브러리에 존재하는 실제 함수의 주소가 쓰여있어서 이 함수를 호출하게 됨

이 때, 첫 호출이냐 아니냐에 따라 동작 과정이 조금 달라짐

두 번째 호출이라면 GOT에 실제 함수의 주소가 쓰여있지만, 첫 번째 호출이라면 GOT에 실제 함수의 주소가 쓰여있지 않음

그래서 첫 호출 시에는 Linker가 dl_resolve라는 함수를 사용해 필요한 함수의 주소를 알아오고, GOT에 주소를 써준 후 해당 함수를 호출 함

4)

< Static Compile 된 실행파일의 함수 호출과정 >

```
(gdb) set disassembly-flavor intel
(gdb) disas main
Dump of assembler code for function main:
   0x08048e24 <+0>:     push   ebp
   0x08048e25 <+1>:     mov    ebp,esp
   0x08048e27 <+3>:     and    esp,0xfffffff0
   0x08048e2a <+6>:     sub    esp,0x10
   0x08048e2d <+9>:     mov    DWORD PTR [esp],0x61
   0x08048e34 <+16>:    call   0x804f5e0 <putchar>
   0x08048e39 <+21>:    mov    eax,0x0
   0x08048e3e <+26>:    leave
   0x08048e3f <+27>:    ret
End of assembler dump.
(gdb) p putchar
$1 = {<text variable, no debug info>} 0x804f5e0 <putchar>
(gdb) r
Starting program: /home/arch/aaa/testfile_static
a[Inferior 1 (process 4358) exited normally]
(gdb) p putchar
$2 = {<text variable, no debug info>} 0x804f5e0 <putchar>
```

〈Static Link〉

: 함수 호출 전과 후의 주소가 같고, 프로그램 내의 주소

< Dynamic Compile 된 실행파일의 함수 호출과정 >

```
(gdb) set disassembly-flavor intel
(gdb) disas main
Dump of assembler code for function main:
   0x0804841d <+0>:     push   ebp
   0x0804841e <+1>:     mov    ebp,esp
   0x08048420 <+3>:     and    esp,0xfffffff0
   0x08048423 <+6>:     sub    esp,0x10
   0x08048426 <+9>:     mov    DWORD PTR [esp],0x61
   0x0804842d <+16>:    call   0x8048310 <putchar@plt>
   0x08048432 <+21>:    mov    eax,0x0
   0x08048437 <+26>:    leave
   0x08048438 <+27>:    ret
End of assembler dump.
(gdb) p putchar
$1 = {<text variable, no debug info>} 0x8048310 <putchar@plt>
(gdb) r
Starting program: /home/arch/aaa/testfile_dynamic
a[Inferior 1 (process 4380) exited normally]
(gdb) p putchar
$2 = {<text variable, no debug info>} 0xf7e6e480 <putchar>
```

〈Dynamic Compile〉

: 함수 호출 전과 후의 주소가 다르고, 프로그램 외부 주소