格。所以-Wl,--wrap,malloc 就把--wrap malloc 传递给链接器，以类似的方式传递-Wl,--wrap,free。

code/link/interpose/mymalloc.c

```
1  #ifdef LINKTIME
2  #include <stdio.h>
3
4  void *__real_malloc(size_t size);
5  void __real_free(void *ptr);
6
7  /* malloc wrapper function */
8  void *__wrap_malloc(size_t size)
9  {
10     void *ptr = __real_malloc(size); /* Call libc malloc */
11     printf("malloc(%d) = %p\n", (int)size, ptr);
12     return ptr;
13 }
14
15 /* free wrapper function */
16 void __wrap_free(void *ptr)
17 {
18     __real_free(ptr); /* Call libc free */
19     printf("free(%p)\n", ptr);
20 }
21 #endif
```

code/link/interpose/mymalloc.c

图 7-21　用--wrap 标志进行链接时打桩

运行该程序会得到如下追踪信息：

```
linux> ./intl
malloc(32) = 0x18cf010
free(0x18cf010)
```

7.13.3　运行时打桩

编译时打桩需要能够访问程序的源代码，链接时打桩需要能够访问程序的可重定位对象文件。不过，有一种机制能够在运行时打桩，它只需要能够访问可执行目标文件。这个很厉害的机制基于动态链接器的 LD_PRELOAD 环境变量。

如果 LD_PRELOAD 环境变量被设置为一个共享库路径名的列表(以空格或分号分隔)，那么当你加载和执行一个程序，需要解析未定义的引用时，动态链接器(LD-LINUX.SO)会先搜索 LD_PRELOAD 库，然后才搜索任何其他的库。有了这个机制，当你加载和执行任意可执行文件时，可以对任何共享库中的任何函数打桩，包括 libc.so。

图 7-22 展示了 malloc 和 free 的包装函数。每个包装函数中，对 dlsym 的调用返回指向目标 libc 函数的指针。然后包装函数调用目标函数，打印追踪记录，再返回。

下面是如何构建包含这些包装函数的共享库的方法：

```
linux> gcc -DRUNTIME -shared -fpic -o mymalloc.so mymalloc.c -ldl
```

这是如何编译主程序：

```
linux> gcc -o intr int.c
```