libmylib.a libmylib.so

printf函数是由谁提供的？

/usr/lib/libc.a

/usr/lib/libc.a中的printf.o

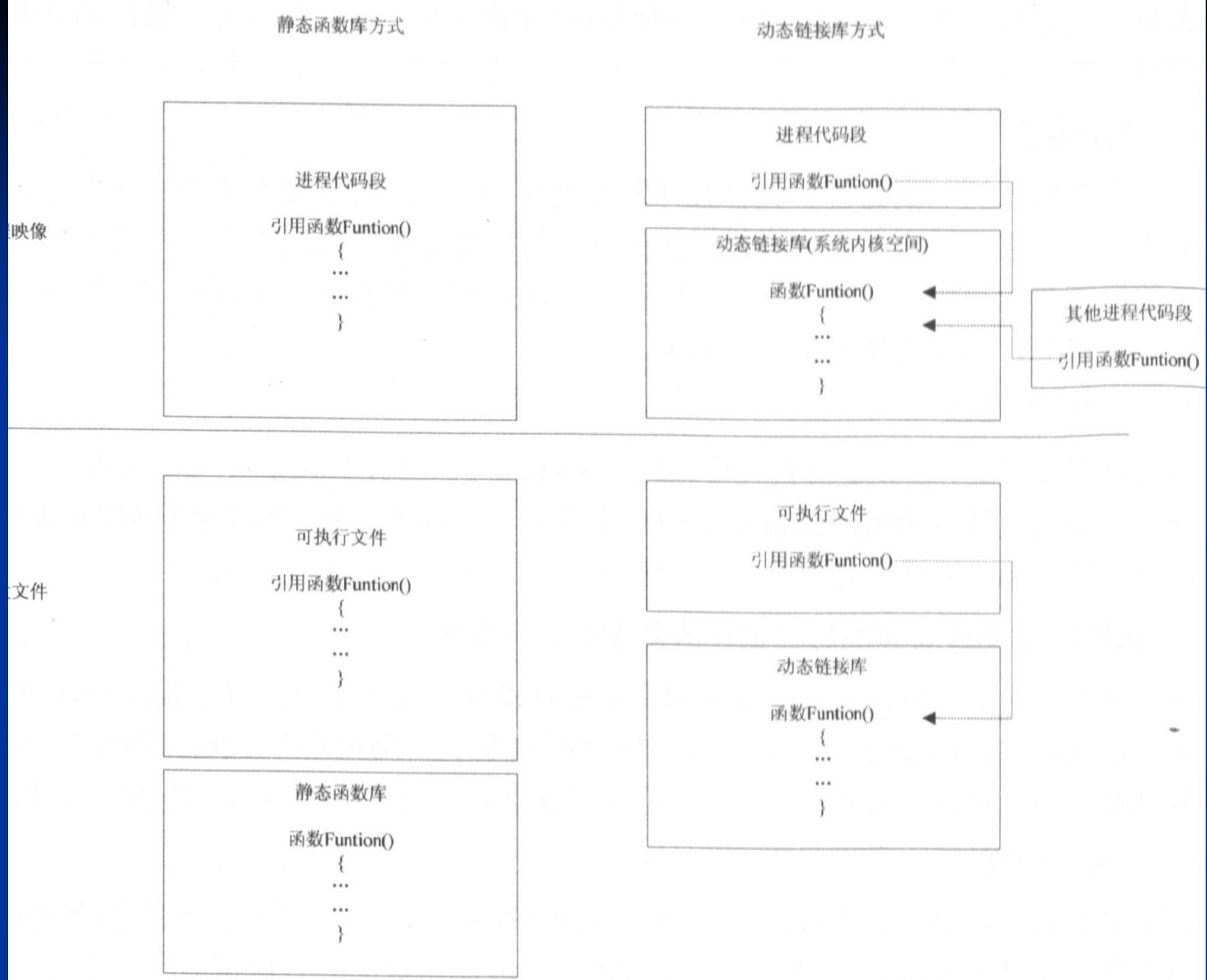
printf.o是怎么得到的？

系统函数库的常见位置

/usr/lib

/lib

/usr/local/lib


静态函数库方式
动态链接库方式
进程代码段
引用函数Funtion()
{
…
…
}
进程代码段
引用函数Funtion()
动态链接库(系统内核空间)
函数Funtion()
{
…
…
}
其他进程代码段
引用函数Funtion()
可执行文件
引用函数Funtion()
{
…
…
}
静态函数库
函数Funtion()
{
…
…
}
可执行文件
引用函数Funtion()
动态链接库
函数Funtion()
{
…
…
}

创建函数库（静态）的步骤

将相对独立的功能形成函数，写入若干.c源文件中

将.c源文件编译成.o文件（gcc -c）

将.o文件打包成libfilename.a文件（ar rv）

使用函数库（静态）的步骤

在源文件中include函数的原型

在源文件中调用函数

在链接时将libfilename.a作为输入，或是使用-L -l选项

系统调用

一些有着特定功能的函数，用法没什么特别的不同。

由Linux内核提供，在核心态下执行

不用在链接的时候指明-l选项，因为根本就不是由函数库实现的

函数库中的函数可能使用了系统调用的函数来实现功能

动态函数库（so）

静态库的二进制代码和.o文件链接后放在最后的可执行文件中，一起被载入内存，为此进程独享。

动态链接库和.o文件链接后，其代码并不进入最后的可执行文件中，需要时载入内存中，可被多个进程共享。

动态函数库（so）

可执行文件尺寸小

升级动态函数库后，不用重新编译使用它的程序

将使用它的程序拷贝到别的系统里，会因为缺少相应的动态函数库而报错

创建函数库（动态）的步骤

将相对独立的功能形成函数，写入若干.c源文件中

gcc –fPIC –c 将.c源文件编译成.o文件

gcc –shared 将.o文件生成libfilename.so文件

使用函数库（动态）的步骤

在源文件中include函数的原型

在源文件中调用函数

在链接时使用，-L -l选项

动态函数库（so）的搜索路径

/etc/ld.so.conf

ldconfig（仅限root)

LD_LIBRARY_PATH

另一种使用动态函数库的方式

dladdr, dlclose, dlerror, dlopen, dlsym, dlvsym

链接时使用-ldl选项

如果dlopen中使用了函数库的相对路径，可以避免路径问题