Keep fvcking!

zooyo.blog.chinaunix.net

暂无签名

zooyo

博客访问： 606217
博文数量： 284
博客积分： 5544
博客等级： 大校
技术积分： 3931
用 户 组： 普通用户
注册时间： 2011-03-08 00:48

加关注 短消息

dlopen函数详解

2013-03-05 10:19:52

分类： LINUX

Linux提供了一套API来动态装载库。下面列出了这些API：

- dlopen，打开一个库，并为使用该库做些准备。
- dlsym，在打开的库中查找符号的值。
- dlclose，关闭库。
- dlerror，返回一个描述最后一次调用dlopen、dlsym，或dlclose的错误信息的字符串。

C语言用户需要包含头文件dlfcn.h才能使用上述API。glibc还增加了两个POSIX标准中没有的API：
- dladdr，从函数指针解析符号名称和所在的文件。
- dlvsym，与dlsym类似，只是多了一个版本字符串参数。

在Linux上，使用动态链接的应用程序需要和库libdl.so一起链接，也就是使用选项-ldl。但是，编译时不需

要和动态装载的库一起链接。程序3-1是一个在Linux上使用dl*例程的简单示例。

延迟重定位(Lazy Relocation)

延迟重定位/装载是一个允许符号只在需要时才重定位的特性。这常在各UNIX系统上解析函数调用时用到。当一个和共享库一起链接的应用程序几乎不会用到该共享库中的函数时，该特性被证明是非常有用的。这种情况下，只有库中的函数被应用程序调用时，共享库才会被装载，否则不会装载，因此会节约一些系统资源。但是如果把环境变量LD_BIND_NOW设置成一个非空值，所有的重定位操作都会在程序启动时进行。也可以在链接器命令行通过使用-z now链接器选项使延迟绑定对某个特定的共享库失效。需要注意的是，除非重新链接该共享库，否则对该共享库的这种设置会一直有效。

初始化(initializing)和终止化(finalizing)函数

有时候，以前的代码可能用到了两个特殊的函数：_init和_fini。_init和_fini函数用在装载和卸载某个模块(注释14)时分别控制该模块的构造器和析构器(或构造函数和析构函数)。他们的C语言原型如下：

```
void _init(void);
void _fini(void);
```

当一个库通过dlopen()动态打开或以共享库的形式打开时，如果_init在该库中存在且被输出出来，则_init函数会被调用。如果一个库通过dlclose()动态关闭或因为没有应用程序引用其符号而被卸载时，_fini函数会在库卸载前被调用。当使用你自己的_init和_fini函数时，需要注意不要与系统启动文件一起链接。可以使用GCC选项 -nostartfiles 做到这一点。

但是，使用上面的函数或GCC的-nostartfiles选项并不是很好的习惯，因为这可能会产生一些意外的结果。相反，库应该使用__attribute__((constructor))和__attribute__((destructor))函数属性来输出它的构造函数和析构函数。如下所示：

```
void __attribute__((constructor)) x_init(void)
```

void __attribute__((destructor)) x_fini(void)
构造函数会在dlopen()返回前或库被装载时调用。析构函数会在这样几种情况下被调用：dlclose()返回前，或main()返回后，或装载库过程中exit()被调用时。

我们通过一个例子来讲解dlopen系列函数的使用和操作:

主程序:

```
1.  #include <stdlib.h>
2.  #include <dlfcn.h>
3.  #include <stdio.h>
4.
5.  //申明结构体
6.  typedef struct __test {
7.      int i;
8.      void (* echo_fun)(struct __test *p);
9.  }Test;
10.
11. //供动态库使用的注册函数
12. void __register(Test *p) {
13.     p->i = 1;
14.     p->echo_fun(p);
15. }
16.
17. int main(void) {
18.
19.     void *handle = NULL;
20.     char *myso = "./mylib.so";
21.
22.     if((handle = dlopen(myso, RTLD_NOW)) == NULL) {
23.         printf("dlopen - %sn", dlerror());
24.         exit(-1);
25.     }
26.
27.     return 0;
28. }
```

动态库:

```
1.  #include <stdio.h>
```

```
2.  #include <stdlib.h>
3.
4.  //申明结构体类型
5.  typedef struct __test {
6.      int i;
7.      void (*echo_fun)(struct __test *p);
8.  }Test;
9.
10. //申明注册函数原型
11. void __register(Test *p);
12.
13. static void __printf(Test *p) {
14.     printf("i = %dn", p->i);
15. }
16.
17. //动态库申请一个全局变量空间
18. //这种 ".成员"的赋值方式为c99标准
19. static Test config = {
20.     .i = 0,
21.     .echo_fun = __printf,
22. };
23.
24. //加载动态库的自动初始化函数
25. void _init(void) {
26.     printf("initn");
27.     //调用主程序的注册函数
28.     __register(&config);
29. }
```

主程序编译: gcc test.c -ldl -rdynamic

动态库编译: gcc -shared -fPIC -nostartfiles -o mylib.so mylib.c

主程序通过dlopen()加载一个.so的动态库文件, 然后动态库会自动运行 _init() 初始化函数, 初始化函数打印一个提示信息, 然后调用主程序的注册函数给结构体重新赋值, 然后调用结构体的函数指针, 打印该结构体的值. 这样就充分的达到了主程序和动态库的函数相互调用和指针的相互传递.

gcc参数 -rdynamic 用来通知链接器将所有符号添加到动态符号表中（目的是能够通过使用 dlopen 来实现向后跟踪）.

gcc参数 -fPIC 作用: 当使用.so等类的库时,当遇到多个可执行文件共用这一个库时, 在内存中,这个库就不会被复制多份,让每个可执行文件一对一的使用,而是让多个可执行文件指向一个库文件,达到共用. 宗旨:节

省了内存空间,提高了空间利用率.

上一篇：syslog函数详解
下一篇：循环队列的基本操作

相关热门文章

ftok返回值生成键值相同问题...
linux 常见服务端口
kernel 报错l701.exel[16922]:...
[nginX-1.7.2] ngx_cycle_t的...
【ROOTFS搭建】busybox的httpd...
C语言 如何在一个整型左边补0...
kernel 3.10代码分析--KVM相关...
xmanager 2.0 for linux配置
python无法爬取阿里巴巴的数据...
C++ 拷贝构造函数 赋值构造函...
什么是shell
linux-2.6.28 和linux-2.6.32....
linux的poll机制
linux socket的bug??
linux su - username -c 命...

给主人留下些什么吧！~~

评论热议

请登录后评论。

登录 注册