c 语言程序动态加载动态库

参考来源

动态加载动态库，一般用于模块扩展或者更换动态库。

1. dlopen 用法

1. 包含头文件 `#include<dlfcn.h>`
2. 函数定义 `void * dlopen(const char* pathName, int mode);`
   `pathName` 指的是 `db` 文件或 `listDB.so` 文件在实机环境中的位置， `mode` 指的是打开数据库的模式。
   `mode` 在 `linux` 下，按功能有以下几种
   - 解析方式：
     - `RTLD_LAZY`： 暂缓决定，在 `dlopen` 返回前，对于动态库中的未定义的符号不执行解析(只对函数引用有效，对于变量引用总是立即解析)
     - `RTLD_NOW`： 立即决定，在 `dlopen` 返回前，解析出所有未定义的符号，如果解析不出来，在 `dlopen` 会返回 `NULL` ， 错误为 `undefined symbol: XXX...`
   - 作用范围：
     - `RTLD_GLOBAL`：动态库中定义的符号可被其后打开的其他库重定位
     - `RTLD_LOCAL`：与 `RTLD_GLOBAL` 作用相反，动态库中定义的符号不能被其后打开的其他库重定位。如果没有指明是 `RTLD_GLOBAL` 还是 `RTLD_LOCAL` ，那么默认是 `RTLD_LOCAL` 。
   - 作用方式：
     - `RTLD_NODELETE`:在 `dlclose()` 期间不卸载库，并且在以后使用 `dlopen()` 重新加载库时不初始化库中的静态变量。这个 `flag` 不是 `POSIX-2001` 标准。
     - `RTLD_NOLOAD`：不加载库，可用于测试库是否已经加载(`dlopen()` 返回 `NULL` 说明未加载，否则说明加载)，也可用于改变已加载库的 `flag` ， 如：先前加载库的 `flag` 为 `RTLD_LOCAL` ，用 `dlopen(RTLD_NOLOAD|RTLD_GLOBAL)` 后 `flag` 将变成 `RTLD_GLOBAL` .这个 `flag` 不是 `POSIX-2001` 标准
     - `RTLD_DEEPBIND`:在搜索全局符号前先搜索库内的符号，避免同名符号的冲突，这个 `flag` 不是 `POSIX-2001` 标准
3. 返回值 `void*` ,如果成功则返回引用这个数据库的句柄，如果失败则返回 `NULL` ， 编译时要加入 `-ldl` （指定 `dl` 库）：

```
ey: gcc test.c -o test -ldl
```

2. dlsym 用法

1. 包含头文件 `#include<dlfcn.h>`
2. 函数定义 `void *dlsym(void *handle, const char* symbol);`
   `handle` 是使用 `dlopen` 函数之后返回的句柄， `symbol` 是要求获取的函数的名称，函数，返回值是 `void*` ，指向函数的地址，供调用使用。

`dlsym` 与 `dlopen` 的以如下例子解释:

```
#include<dlfcn.h>

void * handle = dlopen("./testListDB.so", RTLD_LAZY);
```

如果 `createListDB` 函数定义为:

```
int32_t createListDB(std::string);
```

那么 `dlsym` 的用法则为;

```
int32_t  (*create_listDB)(std::string) = reinterpret_cast<int32_t (*)(std::string)>(dlsym(handle
```

`createListDB` 库函数的定义要用 `extern` 来声明，这样在主函数中才能通过 `createListDB` 来查找函数。

3. dlerror和dlclose用法

1. `dlclose()`：
   - 包含头文件 `#include<dlfcn.h>`
   - 函数定义 `int dlclose(void *handle)`
     `dlclose` 用于关闭指定句柄的动态链接库，只有当此动态链接库的使用计数为0时，才会真正被系统卸载。
2. `dlerror()`：
   - 包含头文件 `#include<dlfcn.h>`
   - 函数定义 `const char* dlerror(void);`
     当动态链接库操作函数执行失败时， `dlerror` 可以返回出错信息，返回值为 `NULL` 时表示操作函数执行成功。

4. 例子

动态库 `testlib.so`：

```
#include <stdio.h>

extern "C" void testlib(void);

void testlib(void)
{
        printf("printf : testlib.\n");
}


gcc testlib.c -o testlib.o -c -fPIC
gcc testlib.o -shared -o libtestlib.so
```

程序 `test.c`：

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

typedef void (*ex_fn)(void);

int main(void)
{
        void* ex_lib = dlopen("./libtestlib.so", RTLD_LAZY);

        if (!ex_lib)
        {
                printf("error : %s\n", dlerror());
                return -1;
        }

        ex_fn ex_test = (ex_fn)dlsym(ex_lib, "testlib");

        if (!ex_test)
        {
                printf("error : %s\n", dlerror());
                dlclose(ex_lib);
                return -1;
        }

        ex_test();

        dlclose(ex_lib);

        return 0;
}


gcc test.c -o test
```

运行:

```
./test
printf : testlib.
```