静态链接、动态链接、动态加载实验

- Static Library is loaded memory at start time, and always reside in memory during a program runs to the end. - Static Link
- Dynamic Library is loaded memory at run time when a program needs. - Dynamic Link

> - The suffix of a static library file is .a extension.
> - The suffix of a dynamic library file is .so or .sa extension.
> - For example: libm.a, libm.so

1. 实验目的

了解和掌握静态链接、动态链接以及动态加载之间的区别

2. 实验内容

> - 利用例子代码，生成静态库文件
> - 利用例子代码，生成动态库文件
> - 利用 VIM 编辑器，编写例子代码，查看运行结果，并加以确认分析
> - 利用动态加载方式，重新编写程序
> - 确认和查看静态链接、动态链接、动态加载时的内存使用

Tips

> - $ ldd (list dynamic dependencies) ：列出动态库依赖关系，比较静态链接、动态链接、动态加载
> - $ dlopen：打开动态库。函数原型 void * dlopen (const char *filename, int flag); dlopen用于打开指定名字的动态库，并返回操作句
> - $ dlsym：取函数执行地址。函数原型为: void * dlsym(void * handle, char * symbol); dlsym根据动态库操作句柄(handle)与符号(symbol)，返回符号对应的函数的执行代码地址。
> - $ dlclose：关闭动态库。函数原型为: int dlclose (void * handle); dlclose用于关闭指定句柄的动态库，只有当此动态库的使用计数为0时,才会真正被系统卸载。
> - $ ranlib : 使用生成动态库索引的工具， man ranlib 命令查看使用说明
> - $ ar: 生成库的工具，使用 man ar 命令查看使用说明
> - $ gcc 编译选项说明：-f 后面跟一些编译选项，PIC(Position Indepentdent Code) 表示生成位置无关代码

3. 例子代码

3.1. fred.c

```c
#include <stdio.h>
void fred(int arg)
{
    printf("fred: you passed %d\n", arg);
 }
```

3.2. bill.c

```c
#include <stdio.h>
void bill(char* arg)
{
   printf("bill:you passed %s\n", arg);
}
```

3.3. lib.h

```c
/* This is lib.h. It declares the functions fred and bill for users */
void bill(char *);
void fred(int);
```

3.4. main.c

```c
#include "lib.h"

int main()
{
     bill("Hello World!");
     exit(0);
 }
```

3.5. libtest.c

```c
#include <stdio.h>
void printHello()
{
    printf("After load dynamic library\n");
    printf("Please enter to continue:");
    getchar();
}
```

3.6. dloading.c

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

void main(void)
{
    void * plib;
    typedef void (*FUN_HELLO)();
    FUN_HELLO funHello = NULL;
    printf("Before loading dynamic library\n");
    printf("Please enter to continue:");
    getchar();

    plib = dlopen("./libtest.so", RTLD_NOW| RTLD_GLOBAL);
    if (plib == NULL)
        printf("error\n");
    funHello = dlsym(plib, "printHello");
    funHello();
    dlclose(plib);
    printf("After release dynmic library\n");
    printf("Please enter to continue:");
    getchar();
}
```

4. 静态库

- 编译生成目标文件，并把目标文件打包成静态库

```
gcc -c fred.c bill.c
ls *.o
ar crv libfoo.a bill.o fred.o
ranlib libfoo.a
```

- 通过链接静态库生成可执行文件-静态链接

```
gcc -c main.c
gcc -o slmain main.o libfoo.a
./slmain
```

- 运行可执行文件，并确认生成的可运行文件的文件大小
- 利用目标文件生成可运行文件

```
gcc -c main.c
gcc -o main  main.o bill.o
./main
```

- 运行可执行文件，并确认生成的可执行文件的文件大小
- 比较 slmain 文件和 main 文件的大写哦

5. 动态库

- 编译生成目标文件，并把目标文件打包成动态库

```
/* 生成目标文件 */
gcc -c fPIC bill.c fred.c
/* 生成动态库 */
gcc -shared -o libfoo.so bill.o fred.o
/* 编译时连接 libfoo.so 动态库 */
gcc main.c -o dlmain -L ./ -lfoo
./dlmain
```

- 如运行可执行文件会出现一下错误提示
  error while loading shared libraries: libfoo.so:cannot open shared object file: No such file or directory

- 把生成的 libfoo.so 文件复制到 /usr/lib 目录下，重新运行。同时确认生成的可执行文件的文件大小，与静态库连接生成的可执行文件进行比较。

```
sudo cp libfoo.so /usr/lib
./dlmain
```

6. 动态加载

- 利用 libtest.c 源码，生成动态库，命名为 libtest.so，并动态加载方式运行 dloading.c.
- 利用所提供的工具，观察对动态加载 libtest 库之前、之后，以及 release 动态库之后的内存使用情况
- 对 slmain, dlmain，dloading 文件大小进行比较

```
gcc libtest.c -shared -fPIC -o libtest.so
gcc dloading.c -o dloading -ldl
./dloading
```