信息安全基础综合设计实验

Lecture 05

李经纬

电子科技大学

课程回顾

内容回顾

- 伪随机数：与随机数不可区分，**可重现**
- 线性同余伪随机数生成器：
  - 迭代式：$X_{i+1} = aX_i + c \bmod m$
  - 特征：**一次产生一个伪随机数**；不具备可证明安全性
- BBS伪随机数生成器：
  - 参数选择：p和q为素数，且$p \bmod 4 = q \bmod 4 = 3$
  - 迭代式：$X_{i+1} = X_i^2 \bmod N$
  - 特征：**一次产生一个伪随机比特**；安全性可规约到大数难分解困难问题

线性同余算法

➢思路：设定基础种子后，通过`lcg_rand()`更新迭代后的值

```
// 设定种子_seed
void lcg_srand(unsigned int seed) {
    _seed = seed;
}

// 线性同余算法基本步骤
unsigned int lcg_rand() {
    _seed = (_a * _seed + _c) % _m;
    return _seed;
}
```

BBS伪随机数生成算法

- 思路：迭代后通过flag选择最重要比特位

```
// 迭代32轮，选择重要比特位后保存在bit_stream中
for (int i = 0; i < 32; ++i) {
    _bbs_seed = (_bbs_seed * _bbs_seed) % _n;
    switch (flag) {
        case 0: bit_stream += last_bit(_bbs_seed);
        case 1: bit_stream += parity_odd(_bbs_seed));
        case 2: bit_stream += parity_even(_bbs_seed));
    }
}
// 最后将二进制流转换成十进制数；
```

BBS伪随机数生成算法：最低位

- 思路：与1求AND，获取最后一位比特值

```
// 与0x0001进行并运算获得最后一位的比特值
int last_bit(unsigned int x) {
    return x & 0x0001;
}
```

BBS伪随机数生成算法：奇偶校验位

- 思路：通过统计数字二进制的“1”的个数返回相应值

```
int parity_even(unsigned int x) {
    int count = 0;
    while (x) { // 判断是否为“1”并计数
        if (x & 0x1) { ++count;}
        x >>= 1; // 移位进行下一位判断
    }
    return count % 2 == 0 ? 0 : 1; // 奇数则返回1
}
// 奇校验为偶校验的相反值
int parity_odd(unsigned int x) {
    return 1 - parity_even(x);
}
```

编译过程

编译过程

- **gcc/g++ [-o filename] source.c/cpp**
  - 功能：将源程序文件转化为可执行文件
  - **过程：预处理→编译→汇编→链接**

- **DEMO程序：**
  - 打印字符串及长度

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    char *name = "hello"; // define name of user
    // print the name and the length of the name
    printf("%s      %ld\n", name, strlen(name));
    return 0;
}
```

预处理

- **gcc -E demo.c -o demo.i**
  - 处理宏定义指令
  - 处理条件编译指令
  - 扩展头文件包含指令
- DEMO程序的预处理代码中：
  - 注释、空行被删除
  - **printf、strlen**函数声明被加入
    - **未定义声明的函数，在预处理过程不会报错**

```
…
void run();
#define RUN 1
int main(int argc, char *argv[])
{
#ifndef RUN
    run();
#endif
    …
}
```

预处理后，不会包含调用**run**函数

编译

- `gcc -S demo.i -o demo.s`
  - 将预处理代码转化为**汇编代码**
- ***汇编语言入门***

```
 1   .file "demo.c"
 2   .text
 3   .section  .rodata
 4 .LC0:
 5   .string "hello"
 6 .LC1:
 7   .string "%s      %ld\n"
 8   .text
 9   .globl  main
10   .type main, @function
11 main:
12 .LFB6:
13   .cfi_startproc
14   endbr64
15   pushq %rbp
16   .cfi_def_cfa_offset 16
17   .cfi_offset 6, -16
18   movq  %rsp, %rbp
19   .cfi_def_cfa_register 6
20   subq  $32, %rsp
21   movl  %edi, -20(%rbp)
22   movq  %rsi, -32(%rbp)
23   leaq  .LC0(%rip), %rax
24   movq  %rax, -8(%rbp)
25   movq  -8(%rbp), %rax
26   movq  %rax, %rdi
27   call  strlen@PLT
```

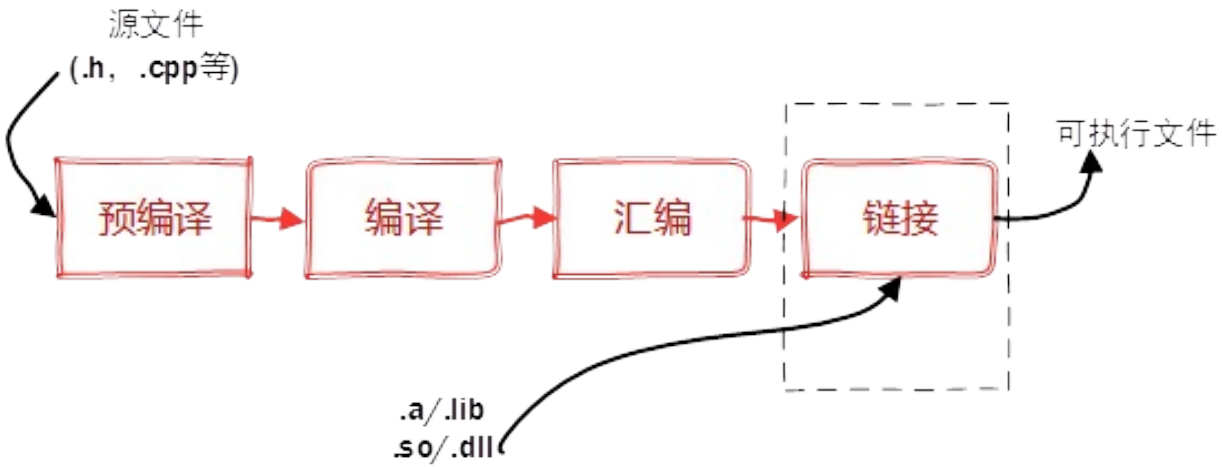
汇编

- **gcc -c demo.s -o demo.o**
  - 将汇编代码转化为目标代码（Linux下为ELF文件；Windows下为PE文件）
- **objdump -s -d demo.o查看*ELF文件结构***
- **注：目标代码还不能执行**

```
demo.o:     file format elf64-x86-64

Contents of section .text:
 0000 f30f1efa 554889e5 4883ec20 897dec48  ....UH..H.. .}.H
 0010 8975e048 8d050000 00004889 45f8488b  .u.H......H.E.H.
 0020 45f84889 c7e80000 00004889 c2488b45  E.H.......H..H.E
 0030 f84889c6 488d3d00 000000b8 00000000  .H..H•=.........
 0040 e8000000 00b80000 0000c9c3           ............
Contents of section .rodata:
 0000 68656c6c 6f002573 20202020 2020256c  hello.%s      %l
 0010 640a00                               d..
Contents of section .comment:
 0000 00474343 3a202855 62756e74 7520392e  .GCC: (Ubuntu 9.
 0010 332e302d 31307562 756e7475 32292039  3.0-10ubuntu2) 9
 0020 2e332e30 00                          .3.0.
Contents of section .note.gnu.property:
 0000 04000000 10000000 05000000 474e5500  ............GNU.
 0010 020000c0 04000000 03000000 00000000  ................
Contents of section .eh_frame:
 0000 14000000 00000000 017a5200 01781001  .........zR..x..
 0010 1b0c0708 90010000 1c000000 1c000000  ................
 0020 00000000 4c000000 00450e10 8602430d  ....L....E....C.
 0030 0602430c 07080000                    ..C.....

Disassembly of section .text:

0000000000000000 <main>:
   0:   f3 0f 1e fa             endbr64
--More--
```

链接

- **`gcc demo.o -o demo`**
  - 纳入函数/变量定义实现
  - 静态链接：将实现一起打包生成可执行文件
  - 动态链接：运行时动态加载



静态库、动态库区别来自【链接阶段】如何处理库，链接成可执行程序。分别称为静态链接方式、动态链接方式。

静态库 & 动态库

函数库

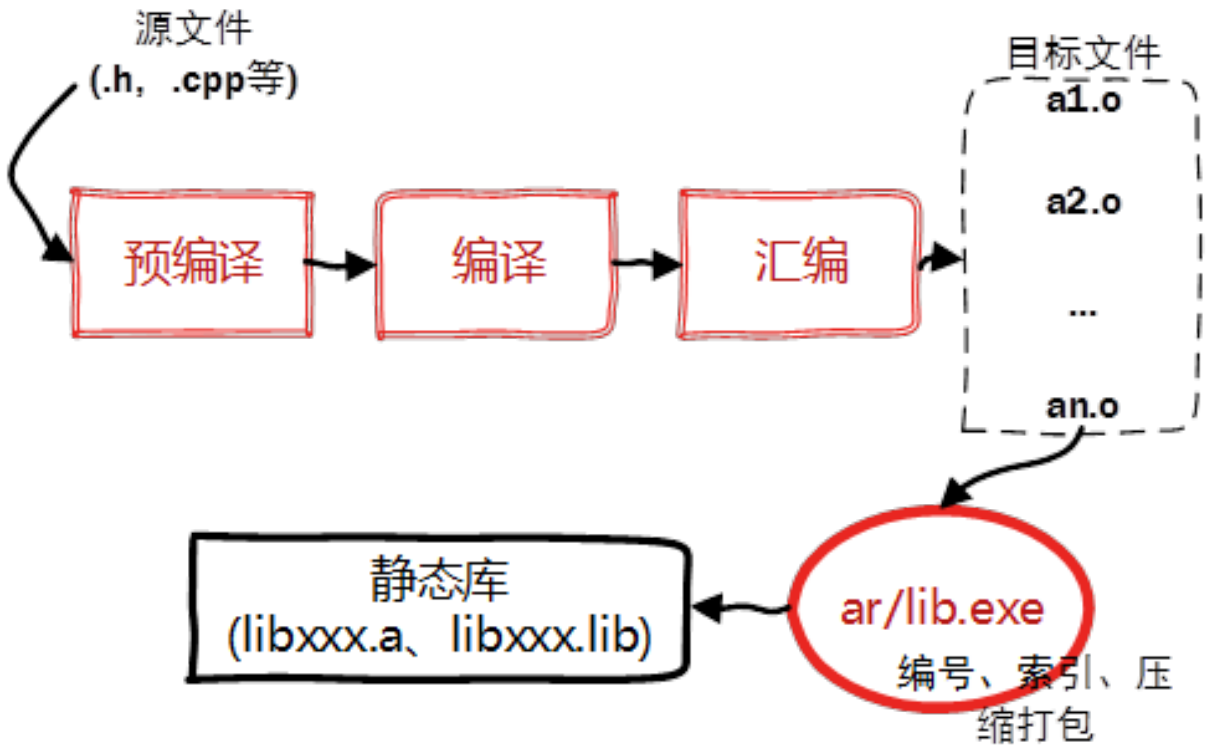
- 现有的，成熟的，可复用的代码
- 根据链接方式，可分为：
  - 静态库：`.a`（Linux）、`.lib`（Windows）
  - 动态库：`.so`（Linux）、`.dll`（Windows）
- Linux下命名规则：**lib<函数库名称>.<a/so>**

静态库

- 一组目标文件（.o/.obj文件）的集合
  - 与汇编生成的目标文件.o一起链接打包到可执行文件中
- 特点：
  - 链接是在编译阶段完成
  - 程序运行时与函数库再无瓜葛，移植方便
  - 可执行文件巨大，包含所有相关目标文件与涉及函数库

静态库创建过程

- 工具：
  - Linux：**ar**
  - Windows：**lib.exe**



Linux下静态库创建过程

```
// add.h
int add(int a, int b);
```

```
// add.c
#include<add.h>
int add(int a, int b) {
    return a+b;
}
```

```
// main.c
#include<add.h>
#include<stdio.h>
int main() {
    int a = 10, b = 90;
    printf("a+b=%d", add(a, b));
    return 0;
}
```

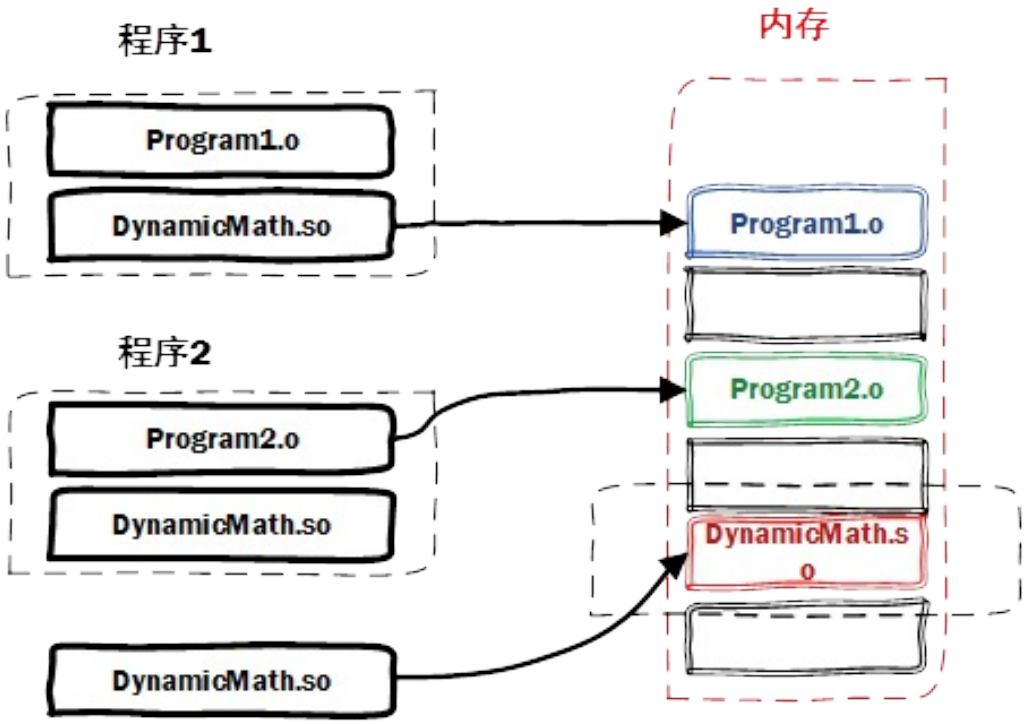
```
# 命令流程

# （1）编译生成目标文件
$ gcc -c add.c -I. -o add.o
$ gcc -c main.c -I. -o main.o

# （2）生成静态库
$ ar -crv libadd.a add.o
# -c：创建静态库
# -r：将目标文件加入静态库
# -v：显示处理信息
# 注：函数库名前须加lib，后缀名为.a

# （3）静态链接
$ gcc main.o -L. -ladd -o main
# -l<函数库名>：指明需要链接的函数库
```

将当前目录纳入系统头文件搜索路径

将当前目录纳入系统函数库搜索路径

动态库

- 静态库不足：巨大的可执行文件，全量更新
- 动态库：编译时不会链接到目标代码，**在程序运行时动态加载**



动态库在内存中只存在一份拷贝，避免了静态库浪费空间的问题。

Linux下动态库创建过程

```
// add.h
int add(int a, int b);
```

```
// add.c
#include<add.h>
int add(int a, int b) {
    return a+b;
}
```

```
// main.c
#include<add.h>
#include<stdio.h>
int main() {
    int a = 10, b = 90;
    printf("a+b=%d", add(a, b));
    return 0;
}
```

```
# 命令流程

# (1)编译生成动态库
$ gcc -fPIC -shared add.c -I. -o libadd.so
# -fPIC：创建与地址无关的程序
# -shared：生成动态链接库

# (2)动态链接
$ gcc main.c -L. -ladd -o main

# 注：须增加动态链接库搜索路径
$ sudo vim /etc/ld.so.conf #增加动态库路径
$ sudo ldconfig
```

基于OpenSSL的大数运算

大数运算

- 基于RSA的非对称密码实现基础
  - 基本数据类型表示的数值范围有限，难以满足大规模数值计算
- ***实现大数运算***

OpenSSL

- 开源密码学库
  - 安装：`sudo apt-get install libssl-dev`
- 编译指令：`gcc/g++ <源文件> -o <可执行文件> -lcrypto`
  - 大数运算库名称：***crypto***
  - 头文件和库文件已在环境变量**LD_LIBRARY_PATH**和**CPLUS_INCLUDE_PATH**（**C_INCLUDE_PATH**）搜索路径中

大数运算库——基础

- 头文件：**#include<openssl/bn.h>**
- BIGNUM基础：初始化和回收
  - 初始化：**BN_new**
  - 回收：**BN_free**

```
#include <openssl/bn.h>
int main () {
    BIGNUM *bn;
    bn = BN_new();
    BN_free(bn);
    return 0;
}
```

大数运算库——赋值和输出

- 赋值函数：**BN_zero、BN_one、BN_set_word…**
- 输出函数：**BN_print_fp**
  - 注：**BN_print_fp**按16进制输出

```
#include <openssl/bn.h>
int main () {
    BIGNUM *bn;
    bn = BN_new();
    BN_zero(bn);
    BN_print_fp(stdout, bn);
    BN_one(bn);
    BN_print_fp(stdout, bn);
    BN_set_word(1024, bn);
    BN_print_fp(stdout, bn);
    BN_free(bn);
    return 0;
}
```

课堂作业

OpenSSL模指数运算

- 计算大数模指数

- 函数头：

```
string mod_exp(string a, string e, string m)
// 该函数用于进行大数模指数运算
// 参数：string类型，求解a^e mod m，表示为10进制数字符串
// 返回值：string类型，返回计算的结果，表示为10进制数字符串
```

OpenSSL乘法逆元运算

- 计算大数乘法逆元

- 函数头：

```
string mod_inverse(string a, string m)
// 该函数用于进行大数求乘法逆元
// 参数：string类型，求解a关于m的乘法逆元，表示为10进制数字符串
// 返回值：string类型，返回计算的结果，表示为10进制数字符串
```