秘密★启用前

# 2021秋ICS小班练习题8

建议用时：50分钟

Linking I
Linking II

姓名：　　学号：

答卷说明：
a.答卷前,考生务必将自己的姓名填写在答题卡指定位置.
b.答题时,请将答案填写在试卷和答题卡相应位置.如需改动,请用签字笔将原答案划去,再在规定位置填写修正后的答案.未在规定区域作答的答案无效.
c.本卷共4页,卷面分90分.考试结束后,试卷由助教统一收回.

一、判断题（24分） 对请打勾，错请打叉。

/* 编译系统 */

1. （ ）c语言的编译步骤依次是预处理、编译、汇编、链接。其中，预处理阶段主要完成的两件事情是头文件包含和宏展开。
2. （ ）假设当前目录下已有可重定位模块 main.o 和 sum.o，为了链接得到可执行文件 prog，可以使用指令 ld -o prog main.o sum.o

/* 静态链接 */

3. （ ）链接时，链接器会拷贝静态库(.a)中的所有模块(.o)。
4. （ ）链接时，如果所有的输入文件都是.o或.c文件，那么任意交换输入文件的顺序都不会影响链接是否成功。
5. （ ）c程序中的全局变量不会被编译器识别成局部符号。

/* 动态链接 */

6. （ ）动态链接可以在加载时或者运行时完成，并且由于可执行文件中不包含动态链接库的函数代码，使得它比静态库更节省磁盘上的储存空间。
7. （ ）动态库通常被编译成位置无关代码。
8. （ ）通过代码段的全局偏移量表 GOT 和数据段的过程链接表 PLT，动态链接器可以完成延迟绑定 (lazy binding)。

/* 加载 */

9. （ ）_start 函数是程序的入口点。
10. （ ）地址空间布局随机化(ASLR)不会影响代码段和数据段间的相对偏移，这样位置无关代码才能正确使用。

/* static/extern 关键字 */

11. （ ）函数内的被 static 修饰的变量将分配到静态存储区，其跨过程调用值仍然保持。
12. （ ）变量声明默认不带 extern 属性，但对函数原型的声明默认带 extern 属性。

二、选择题（10分） 每题只有一个正确答案

11.（ ）c 源文件 f1.c 和 f2.c 的代码分别如下所示，那么运行结果为?

```
// f1.c
#include <stdio.h>
static int var;
int main() {
  extern void f(void);
  f();
  printf("%d", var);
  return 0;
}
```

```
// f2.c
extern int var;

void f() { var++; }

int var = 100;
```

A. 0　　B. 1　　C. 101　　D. 链接时出错

12.（ ）在 gcc-7 编译系统下，以下的两个文件能够顺利编译并被执行。在 x86-64 机器上，若某次运行时得到输出 0x48\n，请你判断这个 16 进制的 48 产生自？

```
// f1.c
void p2(void);

int main() {
  p2();
  return 0;
}
```

```
// f2.c
#include <stdio.h>
char main;

void p2() {
  printf("0x%x\n", main);
}
```

A. 垃圾值
B. main 函数汇编地址的最低字节按有符号补齐的结果
C. main 函数汇编地址的最高字节按有符号补齐的结果
D. main 函数汇编的第一个字节按有符号补齐的结果

三、非选择题（56分）请将答案填写在答题卡上

13. 有下面两个程序。将它们先分别编译为.o文件，再链接为可执行文件。请注意，本大题内前问信息在后问中均有效。

```
// m.c
#include <stdio.h>
void foo(int *);

int buf[2] = {1,2};

int main(){
  foo(buf);
  printf("%d %d", buf[0],
          buf[1]);
  return 0;
}
```

```
// foo.c
extern int buf[];
int *bufp0 = &buf[0];
int *bufp1;

void foo(){
  static int count = 0;
  int temp;
  bufp1 = &buf[1];
  temp = *bufp0;
  *bufp0 = *bufp1;
  *bufp1 = temp;
  count++;
}
```

**Part A. （20分）**请填写 foo.o 模块的符号表。如果某个变量不在符号表中，那么在名字那一栏打×；如果它在符号表中的名字含有随机数字，那么请用不同的四位数字区分多个不同的符号。对于局部符号，不需要填强符号一栏。

| 变量名 | 符号表中的名字 | 局部符号? | 强符号? | 所在节 |
|---|---|---|---|---|
| buf | | | | |
| bufp0 | | | | |
| bufp1 | | | | |
| temp | | | | |
| count | | | | |

**Part B.（15分）**使用 gcc foo.c m.c 生成 a.out。 其节头部表部分信息如下。已知 main 和 foo 的汇编代码相邻，且 Ndx 和 Nr 都是指节索引。请补充空缺的内容。

Section Headers:

| [Nr] Name | Type | Address | Offset | Size |
|---|---|---|---|---|
| [ 1] .interp | PROGBITS | 00000000000002a8 | 000002a8 | 000000000000001c |
| [14] .text | PROGBITS | 0000000000001050 | 00001050 | 0000000000000205 |
| [16] .rodata | PROGBITS | 0000000000002000 | 00002000 | 0000000000000027 |
| [23] .data | PROGBITS | 0000000000004000 | 00003000 | 0000000000000020 |
| [24] .bss | NOBITS | 0000000000004020 | 00003020 | 0000000000000010 |

Symbol Table:

| Num: | Value | Size | Type | Bind | Ndx | Name |
|---|---|---|---|---|---|---|
| 35: | 0000000000004024 | _____ | _____ | _____ | ___ | count.1797 |
| 54: | 0000000000004010 | 8 | OBJECT | _____ | ___ | bufp0 |
| 59: | 000000000000115a | _____ | FUNC | GLOBAL | ___ | foo |
| 62: | _____ | _____ | OBJECT | GLOBAL | ___ | buf |
| 64: | 00000000000011a8 | 54 | _____ | GLOBAL | 14 | main |
| 68: | _____ | 8 | OBJECT | GLOBAL | ___ | bufp1 |
| 51: | 0000000000000000 | 0 | FUNC | _____ | UND | printf@@GLIBC_2.2.5 |

**Part C.** （4分）接 Part B. 回答以下问题。

a) 读取 .interp 节，发现是一个可读字符串 /lib64/____-linux-x86-64.____.2。

b) .bss节存储时占用的空间为_____字节，运行时占用的空间为_____字节。

**Part D.** （7分）接 Part B. 通过 objdump -dx foo.o 我们看到如下重定位信息。

```
0000000000000000 <main>:
 0:   55                      push %rbp
...
10:   8b 15 00 00 00 00      mov  0x0(%rip),%edx   # 16 <main+0x16>
                     12: R_X86_64_PC32        buf
...
1e:   48 8d 3d 00 00 00 00   lea  0x0(%rip),%rdi   # 25 <main+0x25>
                     21: R_X86_64_PC32        .rodata-0x4
...
2a:   e8 00 00 00 00         callq  2f <main+0x2f>
                     2b: R_X86_64_PLT32       printf-0x4
...
```

假设链接器生成 a.out 时已经确定：foo.o 的 .text 节在 a.out 中的起始地址为 ADDR(.text)=0x11a8。请写出重定位后的对应于 main+0x10 位置的代码。

```
____: 8b 15 __ __ __ __        mov  0x____(%rip),%edx
```

而 main+0x1e 处的指令变成

```
11c6: 48 8d 3d 54 0e 00 00    lea  0xe54(%rip),%rdi
```

可见字符串“%d %d”在 a.out 中的起始地址是 0x_____。

**Part E.** （10分）使用 objdump -d a.out 可以看到如下 .plt 节的代码。

```
Disassembly of section .plt:

0000000000001020 <.plt>:
    1020:  ff 35 9a 2f 00 00      pushq  0x2f9a(%rip)
           # 3fc0 <_GLOBAL_OFFSET_TABLE_+0x8>
    1026:  ff 25 9c 2f 00 00      jmpq   *0x2f9c(%rip)
           # 3fc8 <_GLOBAL_OFFSET_TABLE_+0x10>
    102c:  0f 1f 40 00            nopl   0x0(%rax)

0000000000001030 <printf@plt>:
    1030:  ff 25 9a 2f 00 00      jmpq   *0x2f9a(%rip)
           # 3fd0 <printf@GLIBC_2.2.5>
    1036:  68 00 00 00 00         pushq  $0x0
    103b:  e9 e0 ff ff ff         jmpq   1020 <.plt>
```

a) 完成 main+0x2a 处的重定位。

```
_____:        e8 __ __ __ __          callq <printf@plt>
```

a) printf 的 PLT 表条目是 PLT[____]，GOT 表条目是 GOT[____] (均填写数字)。

b) 使用 gdb 对 a.out 进行调试。某次运行时 main 的起始地址为 0x5555555551a8，那么当加载器载入内容后而尚未重定位 printf 地址前，其 GOT 的内容是 0x_____。你填写的这个值是 _____ (填 静态/动态) 链接器设置的。而重定位后可以使用 disas _____ 读出 printf 动态链接进来的代码。

**提示：** disas 是 gdb 中用于反汇编的指令。gdb 如果通过立即数直接访问内存地址，直接使用该数即可。如果需要从一个地址中读值并以此间接访问内存，可以使用 *(long *)0xImm 的格式，其中 Imm 表示该立即数。

**Part F.** (14分) (a)已知x86-64汇编指令 ret 的十六进制机器码为 0xc3 。如果在一台现代Intel x86机器上使用 gcc 编译 foo.c 和 bar.c 得到可执行文件 a.out，再执行它，则会在_____个步骤中出错(填"编译"，"链接"，"执行"之一)?

```c
void foo(void);
int main(){
      foo();
      return 0;
}
```
*foo.c*

```c
int foo = 0xc3;
```
*bar.c*

(b) C 源文件 m1.c 和 m2.c 的代码分别如下所示，编译链接生成可执行文件后执行，结果最可能为：____.若交换链接时m2.c和m1.c的顺序，则答案为____.

```c
// m1.c
#include <stdio.h>

int a1 ;
int a2 = 2 ;
extern int a4 ;

void hello()
{
    printf("%p ", &a1);
    printf("%p ", &a2);
    printf("%p\n", &a4);
}
```

```c
//m2.c
int a4 = 10 ;

int main()
{
    extern void hello() ;

    hello() ;
    return 0 ;
}
```

```
$ gcc -o a.out m2.c m1.c ; ./a.out
0x1083020 _____________ _____________
A. 0x1083018, 0x108301c        B. 0x1083028, 0x1083024
C. 0x1083024, 0x1083028        D. 0x108301c, 0x1083018
```

秘密★启用前

2021秋ICS小班18班第(8)次考试

# 参考答案

命题人：丁睿

！试卷和答案请勿外传或上网！

一、判断题（24分） 对请打勾，错请打叉。

/* 编译系统 */

1. （对）c语言的编译步骤依次是预处理、编译、汇编、链接。其中，预处理阶段主要完成的两件事情是头文件包含和宏展开。
2. （对）假设当前目录下已有可重定位模块 main.o 和 sum.o，为了链接得到可执行文件 prog，可以使用指令 ld -o prog main.o sum.o

/* 静态链接 */

3. （错）链接时，链接器会拷贝静态库(.a)中的~~所有模块~~(.o)。
4. （对）链接时，如果所有的输入文件都是.o或.c文件，那么任意交换输入文件的顺序都不会影响链接是否成功。
5. （错）c程序中的全局变量不会被编译器识别成局部符号。static

/* 动态链接 */

6. （对）动态链接可以在加载时或者运行时完成，并且由于可执行文件中不包含动态链接库的函数代码，使得它比静态库更节省磁盘上的储存空间。
7. （对）动态库通常被编译成位置无关代码。
8. （错）通过~~代码段~~的全局偏移量表 GOT 和~~数据段~~的过程链接表 PLT，动态链接器可以完成延迟绑定 (lazy binding)。

/* 加载 */

9. （对）_start 函数是程序的入口点。
10. （对）地址空间布局随机化(ASLR)不会影响代码段和数据段间的相对偏移，这样位置无关代码才能正确使用。

/* static/extern 关键字 */

11. （对）函数内的被 static 修饰的变量将分配到静态存储区，其跨过程调用值仍然保持。
12. （对）变量声明默认不带 extern 属性，但对函数原型的声明默认带 extern 属性。

二、选择题（10分） 每题只有一个正确答案

11.（ A ）c 源文件 f1.c 和 f2.c 的代码分别如下所示，那么运行结果为?

```
// f1.c
#include <stdio.h>
static int var;
int main() {
  extern void f(void);
  f();
  printf("%d", var);
  return 0;
}
```

```
// f2.c
extern int var;

void f() { var++; }

int var = 100;
```

A. 0　　B. 1　　C. 101　　D. 链接时出错

12. ( D )在 gcc-7 编译系统下，以下的两个文件能够顺利编译并被执行。在 x86-64 机器上，若某次运行时得到输出 0x48\n，请你判断这个 16 进制的 48 产生自？

```
// f1.c
void p2(void);

int main() {
  p2();
  return 0;
}
```

```
// f2.c
#include <stdio.h>
char main;

void p2() {
  printf("0x%x\n", main);
}
```

A. 垃圾值
B. main 函数汇编地址的最低字节按有符号补齐的结果
C. main 函数汇编地址的最高字节按有符号补齐的结果
D. main 函数汇编的第一个字节按有符号补齐的结果

三、非选择题（56分）请将答案填写在答题卡上

13. 有下面两个程序。将它们先分别编译为.o文件，再链接为可执行文件。请注意，本大题内前问信息在后问中均有效。

```
// m.c
#include <stdio.h>
void foo(int *);

int buf[2] = {1,2};

int main(){
  foo(buf);
  printf("%d %d", buf[0],
         buf[1]);
  return 0;
}
```

```
// foo.c
extern int buf[];
int *bufp0 = &buf[0];
int *bufp1;

void foo(){
  static int count = 0;
  int temp;
  bufp1 = &buf[1];
  temp = *bufp0;
  *bufp0 = *bufp1;
  *bufp1 = temp;
  count++;
}
```

**Part A. （20分）**请填写 foo.o 模块的符号表。如果某个变量不在符号表中，那么在名字那一栏打×；如果它在符号表中的名字含有随机数字，那么请用不同的四位数字区分多个不同的符号。对于局部符号，不需要填强符号一栏。

| 变量名 | 符号表中的名字 | 局部符号? | 强符号? | 所在节 |
|---|---|---|---|---|
| buf | buf | No | No | UND/UNDEF |
| bufp0 | bufp0 | No | Yes | .data/.rel.data |
| bufp1 | bufp1 | No | No | COM/COMMON |
| temp | × | | | |
| count | count.1797 | Yes | | .bss |

**Part B.** （15分）使用 gcc foo.c m.c 生成 a.out。 其节头部表部分信息如下。已知 main 和 foo 的汇编代码相邻，且 Ndx 和 Nr 都是指节索引。请补充空缺的内容。

Section Headers:

| [Nr] Name | Type | Address | Offset | Size |
|---|---|---|---|---|
| [ 1] .interp | PROGBITS | 00000000000002a8 | 000002a8 | 000000000000001c |
| [14] .text | PROGBITS | 0000000000001050 | 00001050 | 0000000000000205 |
| [16] .rodata | PROGBITS | 0000000000002000 | 00002000 | 0000000000000027 |
| [23] .data | PROGBITS | 0000000000004000 | 00003000 | 0000000000000020 |
| [24] .bss | NOBITS | 0000000000004020 | 00003020 | 0000000000000010 |

Symbol Table:

| Num: | Value | Size | Type | Bind | Ndx | Name |
|---|---|---|---|---|---|---|
| 35: | 0000000000004024 | 4 | OBJECT | LOCAL | 24 | count.1797 |
| 54: | 0000000000004010 | 8 | OBJECT | GLOBAL | 23 | bufp0 |
| 59: | 000000000000115a | 78 | FUNC | GLOBAL | 14 | foo |
| 62: | 0000000000004018 | 8 | OBJECT | GLOBAL | 23 | buf |
| 64: | 00000000000011a8 | 54 | FUNC | GLOBAL | 14 | main |
| 68: | 0000000000004028 | 8 | OBJECT | GLOBAL | 24 | bufp1 |
| 51: | 0000000000000000 | 0 | FUNC | GLOBAL | UND | printf@@GLIBC_2.2.5 |

**Part C.** （4分）接 Part B. 回答以下问题。

a) 读取 .interp 节，发现是一个可读字符串 /lib64/ld-linux-x86-64.so.2。
由 .interp 大小可知要填 4 个字符。而这是动态链接器的绝对路径。

b) .bss节存储时占用的空间为 0 字节，运行时占用的空间为 16 字节。

**Part D.** （7分）现在通过 objdump -dx foo.o 我们看到如下重定位信息。

```
0000000000000000 <main>:
 0:   55                      push %rbp
...
10:   8b 15 00 00 00 00      mov  0x0(%rip),%edx   # 16 <main+0x16>
                     12: R_X86_64_PC32        buf
...
1e:   48 8d 3d 00 00 00 00   lea  0x0(%rip),%rdi   # 25 <main+0x25>
                     21: R_X86_64_PC32        .rodata-0x4
...
2a:   e8 00 00 00 00         callq  2f <main+0x2f>
                     2b: R_X86_64_PLT32       printf-0x4
...
```

假设链接器生成 a.out 时已经确定：foo.o 的 .text 节在 a.out 中的起始地址为 ADDR(.text)=0x11a8。请写出重定位后的对应于原本 main+0x10 位置的代码。

```
11b8: 8b 15 5e 2e 00 00      mov  0x2e5e(%rip),%edx
```
(注意addend=0，或者从程序代码知访问的是buf后一个元素，不要填成 5e2a)

而原本 main+0x1e 处的指令变成

```
11c6: 48 8d 3d 54 0e 00 00   lea  0xe54(%rip),%rdi
```

可见字符串“%d %d”在 a.out 中的起始地址是 0x2021。

**Part E.** （10分）使用 objdump -d a.out 可以看到如下 .plt 节的代码。

```
Disassembly of section .plt:

0000000000001020 <.plt>:
    1020:  ff 35 9a 2f 00 00      pushq  0x2f9a(%rip)
           # 3fc0 <_GLOBAL_OFFSET_TABLE_+0x8>
    1026:  ff 25 9c 2f 00 00      jmpq   *0x2f9c(%rip)
           # 3fc8 <_GLOBAL_OFFSET_TABLE_+0x10>
    102c:  0f 1f 40 00            nopl   0x0(%rax)

0000000000001030 <printf@plt>:
    1030:  ff 25 9a 2f 00 00      jmpq   *0x2f9a(%rip)
           # 3fd0 <printf@GLIBC_2.2.5>
    1036:  68 00 00 00 00         pushq  $0x0
    103b:  e9 e0 ff ff ff         jmpq   1020 <.plt>
```

a) 完成 main+0x2a 处的重定位。

```
11d2:       e8 59 fe ff ff          callq  1030 <printf@plt>
```

b) printf 的 PLT 表条目是 PLT[1]，GOT 表条目是 GOT[3]（均填写数字）。

c) 使用 gdb 对 a.out 进行调试。某次运行时 main 的起始地址为 0x5555555551a8，那么当加载器载入内容后而尚未重定位 printf 地址前，其 GOT 的内容是 0x555555555036。你填写的这个值是 动态（填 静态/动态）链接器设置的。而重定位后可以使用 disas *(long *)0x555555557fd0 读出 printf 动态链接进来的代码。

**提示：** disas 是 gdb 中用于反汇编的指令。gdb 如果通过立即数直接访问内存地址，直接使用该数即可。如果需要从一个地址中读值并以此间接访问内存，可以使用 *(long *)0xImm 的格式，其中 Imm 表示该立即数。

**Part F.** （14分）(a)已知x86-64汇编指令ret的十六进制机器码为0xc3。如果在一台现代Intel x86机器上使用gcc编译foo.c和bar.c得到可执行文件a.out，再执行它，则会在____执行____个步骤中出错(填“编译”，“链接”，“执行”之一)?

```
void foo(void);
int main(){
      foo();
      return 0;
}
```
*foo.c*

```
int foo = 0xc3;
```
*bar.c*

(b) C 源文件 m1.c 和 m2.c 的代码分别如下所示，编译链接生成可执行文件后执行，结果最可能为:D.若交换链接时m2.c和m1.c的顺序，则答案为A.

```
// m1.c
#include <stdio.h>

int a1 ;
int a2 = 2 ;
extern int a4 ;

void hello()
{
    printf("%p ", &a1);
    printf("%p ", &a2);
    printf("%p\n", &a4);
}
```

```
//m2.c
int a4 = 10 ;

int main()
{
    extern void hello() ;

    hello() ;
    return 0 ;
}
```

```
$ gcc -o a.out m2.c m1.c ; ./a.out
0x1083020 _____________ _____________
```

A. 0x1083018, 0x108301c　　B. 0x1083028, 0x1083024

C. 0x1083024, 0x1083028　　D. 0x108301c, 0x1083018