ICS Seminar Week9 Prep

刘昕垚 杨斯淇 许珈铭

2023.11.20

Rules

```
remainder <- ordinal number in WeChat Group % 4
for all questions do
    if question number % 4 == remainder then
        you should work on it
    end
end
```

4. 下列关于链接技术的描述，错误的是（　）

A. 在 Linux 系统中，对程序中全局符号的不恰当定义，会在链接时刻进行报告。

B. 在使用 Linux 的默认链接器时，如果有多个弱符号同名，那么会从这些弱符号中任意选择一个占用空间最大的符号。

C. 编译时打桩（interpositioning）需要能够访问程序的源代码，链接时打桩需要能够访问程序的可重定位对象文件，运行时打桩只需要能够访问可执行目标文件。

D. 链接器的两个主要任务是符号解析和重定位。符号解析将目标文件中的全局符号都绑定到唯一的定义，重定位确定每个符号的最终内存地址，并修改对那些目标的引用。

A

12.下列说法中哪一个是错误的？（　　）

A. 中断一定是异步发生的

B. 异常处理程序一定运行在内核模式下

C. 故障处理一定返回到当前指令

D. 陷阱一定是同步发生的

C

7. 学完本课程后，几位同学聚在一起讨论有关异常的话题，请问你认为他们中谁学习的结果有错误？

A. 发生异常和异常处理意味着控制流的突变。

B. 与异常相关的处理是由硬件和操作系统共同完成的。

C. 异常是由于计算机系统发生了不可恢复的错误导致的。

D. 异常的发生可能是异步的，也可能是同步的。

C

7. 关于 x86-64 系统中的异常，下面那个判断是正确的：

A．除法错误是异步异常，Unix 会终止程序；

B．键盘输入中断是异步异常，异常服务后会返回当前指令执行；

C．缺页是同步异常，异常服务后会返回当前指令执行；

D．时间片到时中断是同步异常，异常服务后会返回下一条指令执行；

C

10. 当一个网络数据包到达一台主机时，会触发以下哪种异常：

A. 系统调用

B. 信号

C. 中断

D. 缺页异常

C

11.在系统调用成功的情况下，下列代码会输出几个 hello? (　　)

```
void doit()
{
    if ( fork() == 0 ) {
       printf("hello\n");
       fork();
    }
    return ;
}

int main()
{
    doit();
    printf("hello\n");
    exit(0) ;
}
```

B

A. 3　　B. 4　　C. 5　　D. 6

9. **在系统调用成功的情况下，下面哪个输出是可能的？**

```
int main() {
    int pid = fork();
    if (pid == 0) {
        printf("A");
    } else {
        pid = fork();
        if (pid == 0) {
            printf("A");
        } else {
            printf("B");
        }
    }
    exit(0);
}
```

A. AAB
B. AAA
C. AABB
D. AA

A

9. 下列程序输出的数字顺序可能是：

```
int count = 1;
if (fork() == 0) {
    if (fork() == 0) {
        printf("%d\n", ++count);
    }
    else {
        printf("%d\n", --count);
    }
}
printf("%d\n", ++count);
```

A. 0 1 3 2 2　　B. 0 3 2 2 1

C. 2 0 1 3 2　　D. 2 1 0 2 3

C

9. C 语言中的代码如下：

```
fork() && fork();
printf("-");
fork() || fork();
printf("-");
```

这段代码一共输出（ ）个“-”字符。

A. 12

B. 18

C. 20

D. 32

B

第四题（10 分）

考虑以下三个文件：

polygon.h

```
struct Node {
    float pos[2];
    int marked;
    struct Node* next;
    struct Node* prev;
};
typedef struct Node node;

node* alloc();
void init();
void gc();
```

main.c（函数中部分内容折叠）

```
#include "polygon.h"
node* root_ptr ;
int main(){
    node* p;
    init();
    p=alloc();
    root_ptr =p;
    ...
    gc();
    ...
    return 0;
}
```

gc.c（函数体被折叠）

```
#include "polygon.h"
#define N (1<<20)
static node polygon [N];
static node* free_ptr ;
static node* root_ptr ;
void mark(node* v){...}
void sweep() {...}
void gc() {...}
void init() {...}
node* alloc() {...}
```

使用命令 gcc –o polygon main.c gc.c 得到可执行文件 polygon。

1．对于每个程序中的相应符号，给出它的属性（局部或全局，强符号或弱符号）
提示：如果某表项中的内容无法确定，请画 X。

main.c

	局部或全局？	强或弱？
root_ptr		
init		
main		

gc.c

	局部或全局？	强或弱？
N		
polygon		
alloc		

2．解释为何其中一些符号被定义了多次，链接器仍然可以成功创建可执行文件。

3．gc.c 的功能是实现一个垃圾收集器。解释为何前面的命令能够编译、链接成功，但得到的执行文件中却存在潜在错误。并试提出如何修复这一 bug。

第四题（10 分）

在 x86_64 **环境下，考虑如下** 2 **个文件：**main.c **和** foo.c **：**

```
/* main.c */
#include <stdio.h>

long long ____________________________;
const char* foo(int);

int main(int argc, char **argv){
    int n = 0;
    sscanf(argv[1], "%d", &n);
    printf(foo(n));
    printf("%llx\n", a);
}
```

```
/* foo.c */
#include <stdio.h>

int a[2];

static void swapper(int num){
    int swapper;
    if (num % 2){
        swapper = a[0];
        a[0] = a[1];
        a[1] = swapper;
    }
}

const char* foo(int num){
    static char out_buf[50];
    swapper(num);
    sprintf(out_buf,                                    "%x\n",
____________________________);
    return out_buf;
}
```

1. 对于每个程序中的相应符号，给出它的属性（局部或全局，强符号或弱符号）（提示：如果某表项中的内容无法确定，请画 X。）

main.c

	局部或全局？	强或弱？
a		
foo		

foo.c

	局部或全局？	强或弱？
a		
foo		
out_buf		

2. 根据如下的程序运行结果，补全程序【在程序空白处填空即可】。

```
$ gcc -o test main.c foo.c
$ ./test 1
bffedead
cafebffedeadbeef

$ ./test 2
beefcafe
deadbeefcafebffe
```

3. 现在有一位程序员要为这个程序编写头文件。假设新的头文件名称为 foo.h，内容如下：

```
extern long long a;
extern char *foo(int);
```

然后在 main.c 和 foo.c 中分别引用该头文件，请问编译链接能通过吗？请说明理由。

main.c

	局部或全局？	强或弱？
A	全局	强
foo	全局	弱

foo.c

	局部或全局？	强或弱？
A	全局	弱
foo	全局	强
out_buf	局部	X

```
long longa = 0xdeadbeefcafebffe; （1 分）
*(int*)((unsigned long long)a + 2) （2 分）
```

不能。无论如何声明 a 的类型都会造成在至少一个文件内引起声明和定义冲突。
结论 1 分，理由 2 分。（结论错不得分）

3. 有下面两个程序。将他们先分别编译为.o 文件，再链接为可执行文件。

main.c

```
#include <stdio.h>
        A
int foo(int n) {
    static int ans = 0;
    ans = ans + x;
    return n + ans;
}
int bar(int n);
void op(void) {
    x = x + 1;
}
int main() {
    for (int i = 0; i < 3; i++) {
        int a1 = foo(0);
        int a2 = bar(0);
        op();
        printf("%d %d ", a1, a2);
    }
    return 0;
}
```

count.c

```
        B
int bar(int n) {
    static int ans = 0;
    ans = ans + x;
    return n + ans;
}
```

(1) 当 A 处为 `int x = 1;`，B 处为 `int x;` 时，完成下表。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，不需要填最后一栏。

文件名	变量名	在符号表中的名字	是局部符号吗?	是强符号吗?
main.o	x			
	bar			
	ans			
count.o	x			
	bar			
	ans			

程序能够链接成功吗？如果可以，程序的运行结果是什么？如果不可以，链接器报什么错？

(2) 当 A 处为 `static int x = 1;`，B 处为 `static int x = 1;` 时，完成下表。

文件名	变量名	在符号表中的名字	是局部符号吗?	是强符号吗?
main.o	x			
	bar			
	ans			
count.o	x			
	bar			
	ans			

程序能够链接成功吗？如果可以，程序的运行结果是什么？如果不可以，链接器报什么错？

(3) 当 A 处为 `int x = 1;`，B 处为 `int x = 1;` 时。程序能够链接成功吗？如果可以，程序的运行结果是什么？如果不可以，链接器报什么错？

文件名	变量名	在符号表中的名字	是局部符号吗?	是强符号吗?
main.o	x	x	×	√
	bar	bar	×	×
	ans	ans.1597	√	
count.o	x	x	×	×
	bar	bar	×	√
	ans	ans.0344	√	

1 1 3 3 6 6

文件名	变量名	在符号表中的名字	是局部符号吗?	是强符号吗?
main.o	x	x	√	
	bar	bar	×	×
	ans	ans.1597	√	
count.o	x	x	√	
	bar	bar	×	√
	ans	ans.0344	√	

1 1 3 2 6 3。两个 x 在各自的.o 文件中的名字都为 x，因为它们不是过程中的静态变量。思考：对于非过程间的静态变量，为什么**编译器**不需要作这样的区分？

链接错误，x 被定义多次。

5. 下面对指令系统的描述中，错误的是：（ ）

A. 通常 CISC 指令集中的指令数目较多，有些指令的执行周期很长；而 RISC 指令集中指令数目较少，指令的执行周期较短。

B. 通常 CISC 指令集中的指令长度不固定；RISC 指令集中的指令长度固定。

C. 通常 CISC 指令集支持多种寻址方式，RISC 指令集支持的寻址方式较少。

D. 通常 CISC 指令集处理器的寄存器数目较多，RISC 指令集处理器的寄存器数目较少。

D

6. Y86 指令 rmmovl 的 SEQ 实现如下图所示，其中①和②分别为：

	rmmovl rA, D(rB)
Fetch	icode:ifun ← M_1[PC] rA:rB ← M_1[PC+1] valC ← M_4[PC+2] valP ← ①
Decode	valA ← R[rA] valB ← R[rB]
Execute	valE ← ②
Memory	M_4[valE] ← valA
Write back	
PC update	PC ← valP

A．PC + 4，valB + 4　　B．PC + 4，valB + valC

C．PC + 6，valB + 4　　D．PC + 6，valB + valC

D

2. 在本课程的 PIPE 流水线中，下列情况会出现数据冒险的是：

A．当前指令会改变下一条指令的目的操作数

B．当前指令会改变下一条指令的源操作数

C．下一条指令会改变当前指令的目的操作数

D．下一条指令会改变当前指令的源操作数

B

4. 下述关于 RISC 和 CISC 的讨论，哪个是错误的

A．RISC 指令集包含的指令数量通常比 CISC 的少

B．RISC 的寻址方式通常比 CISC 的寻址方式少

C．RISC 的指令长度通常短于 CISC 的指令长度

D．手机处理器通常采用 RISC，而 PC 采用 CISC

C

2. 有如下结构定义和程序片段

```
struct A
{
    char c;
    int i;
    double d;
    int array[10];
};

struct B
{
    int array[10];
    double d;
    char c;
    int i;
};

void foo(struct A *pa, struct B *pb, int index)
{
    pb->i = pa->array[index];
}
```

在 Linux 下使用 GCC 编译器，仅采用-O2 选项，上述代码对应的汇编语言是：（将选项依次填入空格内）

```
movslq %edx, %rdx
movl __(%rdi,%rdx,__), %eax
movl %eax, __(%rsi)
```

A. (16, 4, 52)　B. (24, 4, 52)　C. (16, 4, 49)　D. (24, 4, 49)

A

3. 下面说法正确的是：

A. 不同指令的机器码长度是相同的

B. `test %rax, %rax` 恒等于 `cmp $0, %rax`

C. switch 编译后总是会产生跳转表

D. 以上都不对

4. 分析下图的指令执行步骤，请问这是 Y86 指令系统的哪条指令？

Fetch	icode:ifun ← M_1[PC]
Decode	valA ← R[%esp] valB ← R[%esp]
Execute	valE ← valB + 4
Memory	valM ← M_4[valA]
Write back	R[%esp] ← valE
PC update	PC ← valM

A. call

B. ret

C. pushl

D. popl

3． 缓冲区溢出会带来程序风险，下列避免方法中错误的是：

A．在栈中存放特殊字段用于检测是否发生缓冲区溢出

B．避免使用有风险的库函数，如 gets 等

C．随机设置栈的偏移地址

D．分配尽可能大的缓冲区数组

D

4．现有四级指令流水线，分别完成取指、取数、运算、传送结果 4 步操作。若完成上述操作的时间依次为 9ns、10ns、6ns、8ns，则流水线的操作周期应设计为 _____ ns。

A. 6　　B. 8　　C. 9　　D. 10

D

3. 下面哪条指令不是 X86 正确的寻址方式

A. movl $34, (%eax)

B. movl (%eax), %eax

C. movl $23, 10(%edx, %eax)

D. movl (%eax), 8(%ebx)

D

3. 左边的 C 函数中，在 x86_64 服务器上采用 GCC 编译产生的汇编语言如右边所示。那么（1）和（2）的内容分别是：（　　）

```
int arith(int x, int y) {
   return (x < y) ? ( 1 ) : ( 2 );
}
```

```
<arith>:
  lea    (%rsi,%rdi,1),%eax
  mov    %esi,%edx
  sub    %edi,%edx
  cmp    %esi,%edi
  cmovge %edx,%eax
  retq
```

（提示：第一个参数放在 rdi 寄存器中，第二个参数放在 rsi 寄存器中）

A. x-y，x+y　B. x+y，x-y　C. x+y，y-x　D. y-x，x+y

C

4. 假定 struct P {int i; char c; int j; char d;}; 在 x86_64 服务器的 Linux 操作系统上，下面哪个结构体的大小与其它三个不同：答：（　　）

A. struct P1 {struct P a[3]};
B. struct P2 {int i[3]; char c[3]; int j[3]; char d[3]};
C. struct P3 {struct P *a[3]; char *c[3];};
D. struct P4 {struct P *a[3]; int *f[3];};

B

2、按照教材描述的原则，对于 x86_64 程序，在 callq 指令执行后，函数的第一个参数一般存放在哪里？答：（　　）

A. 8(%rsp)　　B. 4(%rsp)　　C. %rax　　D. %rdi

D

3、已知变量 x 的值已经存放在寄存器 eax 中，现在想把 5x+7 的值计算出来并存放到寄存器 ebx 中，如果不允许用乘法和除法指令，则至少需要多少条 IA-32 指令完成该任务？答：（　　）

A．1 条　　B．3 条　　C．2 条　　D．4 条

A