Machine Programming-II

元培数科 王恩博

Let's get started →

2025/10/9

Exercises

E1

7.在作业题中，我们考察过 GCC 为一个参数和返回值都是结构体的函数产生的汇编代码，此时参数和返回值都是通过栈传递的，以下有关说法错误的是：

A. 返回的结构体的地址被存放在%rax 寄存器中

B. 虽然参数是通过栈传递的，但这里依旧使用到了%rdi

C. 返回的结构体实际上是存放在被调用者的栈帧当中

D. 如果结构体很小的话，GCC 可能会直接用寄存器来传输参数和返回值

E1

7.在作业题中，我们考察过 GCC 为一个参数和返回值都是结构体的函数产生的汇编代码，此时参数和返回值都是通过栈传递的，以下有关说法错误的是：

A．返回的结构体的地址被存放在%rax 寄存器中

B．虽然参数是通过栈传递的，但这里依旧使用到了%rdi

C．返回的结构体实际上是存放在被调用者的栈帧当中

D．如果结构体很小的话，GCC 可能会直接用寄存器来传输参数和返回值

答案：C。

C．返回的结构体实际上存放在调用者的栈帧当中，而非被调用者

E2

7. 考虑以下 C 语言变量声明：

```
int *(*f[3])();
```

那么在一台 x86-64 机器上，sizeof(f)和 sizeof(*f)的值是多少？

A. 8 24

B. 24 8

C. 8 8

D. 8 不确定

E2

7. 考虑以下 C 语言变量声明：

```
int *(*f[3])();
```

那么在一台 x86-64 机器上，sizeof(f)和 sizeof(*f)的值是多少？

A. 8 24

B. 24 8

C. 8 8

D. 8 不确定

答案：B 本题考察指针和数组的存储方式，答案为 B。f 是一个数组，每个元素都是一个函数指针，指向返回值为 int *的函数。对于 f 而言，sizeof(f)返回整个指针数组的大小，为 3 * 8 = 24；对于*f 而言，它是 f[0]元素，是一个指针变量，大小为 8。

E3

7. 有A的定义：int A[3][2] = {{1,2}, {3,3}, {2,1}};
那么A[2]的值为：

A. &A+16　　B. A+16　　C. *A+4　　D. *A+2

E3

答案：C

解析：参见书 P177 页表格，机器在计算指针与常数的运算时，会将常数乘以指针指向的元素大小。&A 常数扩大的倍数为 sizeof(A[3][2]) = 3*2*4; A 常数扩大的倍数为 sizeof(A[0])=2*4; *A 常数扩大的倍数为 sizeof(int) = 4。正确的答案应为 A+2 或*A+4，故应选择 C。

程序验证如下：

```
int main(){
    int A[3][2] = {{1,2},{3,3},{2,1}};
    cout << "&A+16: " << &A+16 << endl;
    cout << "A+16: " << A+16 << endl;
    cout << "*A+4: " << *A+4 << endl;
    cout << "*A+2: " << *A+2 << endl;
    cout << A[2] << endl;
}
```

```
&A+16: 0xb5fee0
A+16: 0xb5fde0
*A+4: 0xb5fd70
*A+2: 0xb5fd68
0xb5fd70
```

E4

6. 下列关于 C 语言中的结构体(struct)以及联合(union)的说法中，正确的是：

A) 对于任意 struct，将其成员按照其实际占用内存大小从小到大的顺序进行排列不一定会使之内存占用最小

B) 对于任意 struct，将其成员按照其实际占用内存大小从小到大的顺序进行排列一定不会使之内存占用最大

C) 对于任意 union，将其成员按照其实际占用内存大小从小到大的顺序进行排列不一定会使之内存占用最小

D) 对于任意 union，将其成员按照其实际占用内存大小从小到大的顺序进行排列一定不会使之内存占用最大

E4

6. 下列关于 C 语言中的结构体(struct)以及联合(union)的说法中，正确的是：

A) 对于任意 struct，将其成员按照其实际占用内存大小从小到大的顺序进行排列不一定会使之内存占用最小

B) 对于任意 struct，将其成员按照其实际占用内存大小从小到大的顺序进行排列一定不会使之内存占用最大

C) 对于任意 union，将其成员按照其实际占用内存大小从小到大的顺序进行排列不一定会使之内存占用最小

D) 对于任意 union，将其成员按照其实际占用内存大小从小到大的顺序进行排列一定不会使之内存占用最大

答案：A。联合以及只含一个基本数据类型成员的结构体的内存占用与其成员排列方式无关，即任意排列方式都可使得内存占用最小（最大），故 b、c、d 错误。对于 struct，应当将成员按照其存储单元所占内存大小从小到大（或从大到小）的顺序进行排列才能使内存占用最小，故 a 正确。

E5

8. 在 x86-64 架构下，有如下变量：

```
union {char c[8], int i;} x;
```

在 x.i=0x41424344 时，x.c[2]的值为多少（提示：'A'=0x41）：

A. 'A'　　B. 'B'　　C. 'C'　　D. 'D'

E5

8. 在 x86-64 架构下，有如下变量：

```
union {char c[8], int i;} x;
```

在 x.i=0x41424344 时，x.c[2]的值为多少（提示：'A'=0x41）：

A. 'A'　　B. 'B'　　C. 'C'　　D. 'D'

答案：B

E6

5. 以下代码的输出结果是

```
union {
    double d;
    struct {
        int i;
        char c[4];
    } s;
} u;
u.d = 1;
printf("%d\n", u.s.c[2]);
```

A) 0　　B) -16　　C) 240　　D) 191

E6

答案：B

c[2]保存的数据是$(11110000)_2$，char 保存的数据是有符号数，所以应该是-16

答案修订：ABC 均正确。

题目未说明大小端，若为大端法，选择 A。若为小端法，u.s.c[2]为$(11110000)_2$，由于 char 在不同的编译器上会被实现成 signed char 或 unsigned char，因此扩展为 int 时-16 和 240 均有可能，B，C 均正确。

E7

5. 已知下面的数据结构，假设在 Linux/IA32 下要求对齐，这个结构的总的大小是多少个字节？如果重新排列其中的字段，最少可以达到多少个字节？

```
struct {
  char a;
  double *b;
  double c;
  short d;
  long long e;
  short f;
};
```

A. 32, 28

B. 36, 32

C. 28, 26　　D. 26, 26

E7

5. 已知下面的数据结构，假设在 Linux/IA32 下要求对齐，这个结构的总的大小是多少个字节？如果重新排列其中的字段，最少可以达到多少个字节？

```
struct {
  char a;
  double *b;
  double c;
  short d;
  long long e;
  short f;
};
```

A. 32, 28

B. 36, 32

C. 28, 26　　D. 26, 26

选择 A，Linux/IA32 要求 double 和 long long 在 4 字节边界上对齐，其中 a 占 1 字节，后面填充 1 字节，b 占 4 字节，填充 2 个字节，c 占 8 字节，d 占 2 字节，填充 2 字节，e 占 8 字节，f 占 2 字节，再填充 2 字节，共 32 字节；重新排列时，可以按字节数从大到小排列，最后，最少需要 8+8+4+2+2+1 再加 3 字节满足 4 字节对齐要求，共 28 个字节。

E8

6、有如下定义的结构，在 x86-64 下，下述结论中错误的是？

```
struct {
    char c;
    union {
        char vc;
        double value;
        int vi;
    } u;
    int i;
} sa;
```

A. sizeof(sa) == 24

B. (&sa.i - &sa.u.vi) == 8

C. (&sa.u.vc - &sa.c) == 8

D. 优化成员变量的顺序，可以做到“sizeof(sa) == 16”

答：(　　)

E8

答案：B

由于对齐的需求，在 x86-64 下，要保证 sa.u.value（double 类型变量）的地址必须 8 字节对齐，而在 ia32 的 Linux 系统中，可 4 字节对齐。故 sizeof(sa) 在 x86-64 下要占用三个 8 字节的空间共 24 字节。sa.u.vc 与 sa.c 之间和 sa.i 与 sa.u.vi 之间的地址值之差均为 8，但即使不知道(&sa.i - &sa.u.vi)表示之间可以放置多少个整数（值为 2），也可通过优化成员变量顺序后有 sizeof(sa) == 16，用排除法得出正确答案。

E9

7、关于如何避免缓冲区溢出带来的程序风险，下述错误的做法为？

A．编程时定义大的缓冲区数组

B．编程时避免使用 gets，而采用 fgets

C．程序运行时随机化栈的偏移地址

D．在硬件级别引入不可执行代码段的机制

答：（　　）

E9

7、关于如何避免缓冲区溢出带来的程序风险，下述错误的做法为？

A．编程时定义大的缓冲区数组

B．编程时避免使用 gets，而采用 fgets

C．程序运行时随机化栈的偏移地址

D．在硬件级别引入不可执行代码段的机制

答：（　　）

答案：A

B、C、D 均为讲义中所提及的解决缓冲区溢出风险的方法。A 策略则无法从根本上解决缓冲区溢出问题，只要输入足够长数据就仍然可以实现缓冲区溢出攻击。

E10

8. 大多数过程的栈帧是_____的，其长度在_____时确定。（注：此处的编译指从高级语言转化为汇编语言的过程）

A. 定长，编译

B. 定长，汇编

C. 可变长，汇编

D. 可变长，运行

E10

8. 大多数过程的栈帧是_____的，其长度在_____时确定。（注：此处的编译指从高级语言转化为汇编语言的过程）

A. 定长，编译

B. 定长，汇编

C. 可变长，汇编

D. 可变长，运行

答案：A。

说明：大多数过程的栈帧是定长的，在过程开始时通过减小栈指针的方式分配，减小的大小由编译器在编译时计算。大家常常在汇编代码中过程的开头看到“subq $24, %rsp”，就是编译器计算出了栈帧大小并写在了汇编代码里。（书P165）

E11

9. `pushq %rbp` 的行为等价于以下（）中的两条指令。

A. `subq $8,%rsp  movq %rbp, (%rdx)`

B. `subq $8,%rsp  movq %rbp, (%rsp)`

C. `subq $8,%rsp  movq %rax, (%rsp)`

D. `subq $8,%rax  movq %rbp, (%rdx)`

E11

9. pushq %rbp 的行为等价于以下（）中的两条指令。

```
A.  subq   $8, %rsp    movq  %rbp,  (%rdx)
B.  subq   $8, %rsp    movq  %rbp,  (%rsp)
C.  subq   $8, %rsp    movq  %rax,  (%rsp)
D.  subq   $8, %rax    movq  %rbp,  (%rdx)
```

答案：B。

说明：x86-64 系统中，pushq %rbp 指令将栈指针减 8，并向其中存入%rbp 寄存器的值（书 P127）

E12

5. 已知函数 func 的参数超过 6 个。当 x86-64 机器执行完指令 call func 之后，%rsp 的值为 S。那么 func 的第 k(k > 6) 个参数的存储地址是？

A. S + 8 * (k - 6)

B. S + 8 * (k - 7)

C. S - 8 * (k - 6)

D. S - 8 * (k - 7)

E12

5. 已知函数 func 的参数超过 6 个。当 x86-64 机器执行完指令 call func 之后，%rsp 的值为 S。那么 func 的第 k(k > 6)个参数的存储地址是？

   A. S + 8 * (k - 6)

   B. S + 8 * (k - 7)

   C. S - 8 * (k - 6)

   D. S - 8 * (k - 7)

答案：A

本题考察 x86-64 运行时栈帧结构，答案为 A。当执行完 call 指令后，S 处存储的是函数的返回地址；再往上依次是第 7、第 8 个函数参数...故第 k 个函数参数存储在 S + 8 * (k - 6)的地址处。

E13

9. x86 体系结构中，下面哪个说法是正确的？

A. leal 指令只能够用来计算内存地址

B. x86_64 机器可以使用栈来给函数传递参数

C. 在一个函数内，改变任一寄存器的值之前必须先将其原始数据保存在栈内

D. 判断两个寄存器中值大小关系，只需要 SF 和 ZF 两个条件码

E13

9. x86 体系结构中，下面哪个说法是正确的？

A. leal 指令只能够用来计算内存地址

B. x86_64 机器可以使用栈来给函数传递参数

C. 在一个函数内，改变任一寄存器的值之前必须先将其原始数据保存在栈内

D. 判断两个寄存器中值大小关系，只需要 SF 和 ZF 两个条件码

答案：B

A.leal 指令做普通算术运算；C.caller saved 寄存器才需要；D.还需要 OF

E14

第三题 机器级编程（15 分，每空 1 分）

下面的 C 程序包含 `main()`, `caller()`, `callee()`三个函数。本题给出了该程序的部分 C 代码和 X86-64 汇编与机器代码。请分析给出的代码，补全空白处的内容，并回答问题。

注：汇编与机器码中的数字用 16 进制数填写

X86-64 汇编与机器代码：

答案填写处：

```
0000000000400cd <caller>:
  4006cd: 55                    push   %rbp
  4006ce: 48 89 e5              mov    %rsp,    %rbp
  4006d1: 48 83 ec 50           sub    $0x50,   %rsp
  4006d5: 48 89 7d b8           mov    %rdi,    -0x48(%rbp)
  4006d9: 64 48 8b 04 25 28 00  mov    %fs:0x28,  %rax
  4006e0: 00 00
  4006e2: 48 89 45 f8           mov    %rax,    -0x8(%rbp)
  4006e6: 31 c0                 xor    %eax,    %eax
  4006e8: c6 45 d0 00           movb   $0x0,    -0x30(%rbp)
  4006ec: c6 45 e0 00           movb   $0x0,    (1)          (1) _____
  4006f0: 48 8b 45 b8           mov    (2),     %rax         (2) _____
  4006f4: 48 89 c7              mov    %rax,    %rdi
  4006f7:                       callq  400510 <strlen@plt>
  4006fc:  89 45 cc             mov    (3),     -0x34(%rbp)  (3) _____
  4006ff:  83 7d cc 0e          cmpl   $0xe,    -0x34(%rbp)
  400703: 7f (4)                jg     400752 <caller+0x85>  (4) _____
  400705: 83 7d cc 09           cmpl   $0x9,    -0x34(%rbp)
  400709:                       jg     400720 <caller+0x53>
  40070b: 48 8b 55 b8           mov    -0x48(%rbp), %rdx
  40070f:  48 8d 45 d0          lea    (5),     %rax         (5) _____
  400713: 48 89 d6              mov    %rdx,    %rsi
  400716: 48 89 c7              mov    %rax,    %rdi
  400719:                       callq  400500 <strcpy@plt>
  40071e:                       jmp    40073b <caller+0x6e>
  400720: 48 8b 45 b8           mov    -0x48(%rbp), %rax
  400724: 48 8d 50 0a           lea    0xa(%rax),  %rdx
  400728: 48 8d 45 d0           lea    -0x30(%rbp), %rax
  40072c: 48 83 c0 10           add    (6),     %rax         (6) _____
  400730: 48 89 d6              mov    %rdx,    %rsi
  400733: 48 89 c7              mov    %rax,    %rdi
  400736:                       callq  400500 <strcpy@plt>
  40073b: ff 75 e8              pushq  -0x18(%rbp)
  40073e: ff 75 e0              pushq  -0x20(%rbp)
  400741: ff 75 d8              pushq  -0x28(%rbp)
  400744: ff 75 d0              pushq  -0x30(%rbp)
  400747: e8 (7)                callq  400666 <callee>      (7) _____
  40074c: 48 83 c4 20           add    $0x20,   %rsp
  400750:                       jmp    400753 <caller+0x86>
  400752: 90                    nop
  400753: 48 8b 45 f8           mov    (8),     %rax        (8) _____
  400757: 64 48 33 04 25 28 00  xor    %fs:0x28,  %rax
  40075e: 00 00
  400760:                       je     400767 <caller+0x9a>
  400762:                       callq  400520 <__stack_chk_fail@plt>
  400767: c9                    leaveq
  400768: c3                    retq
```

C 代码：

答案填写处：

```
#include <stdio.h>
#include "string.h"
#define N    (9)                                  (9) _____
#define M    (10)                                 (10)_____

typedef union {char str_u[N]; long l;} union_e;
typedef struct {char str_s[M]; union_e u; long c;} struct_e;

void callee(struct_e s){
  char buf[M+N];
  strcpy(buf, s.str_s);
  strcat(buf, s.u.str_u);
  printf("%s \n",buf);
}

void caller(char *str){
  struct_e s;
  s.str_s[0]='\0';
  s.u.str_u[0]='\0';
  int len = strlen(str);
  if(len>=  M+N)
    (11) ;                                        (11)_____
  else if(len<N){
    strcpy(s.str_s, (12) );                       (12)_____
  }
  else{
    strcpy(s.u.str_u, (13) );                     (13)_____
  }
  callee(s);
}

int main(int argc, char *argv[]){
  caller("0123456789abcd");
  return 0;
}
```

caller 函数中，变量 s 所占的内存空间为：(14)_____

该程序运行后，printf 函数是否有输出？输出结果为：(15)_____

E14

答案：

机器级编程（15 分，每空 1 分）

下面的 C 程序包含 main(), caller(), callee()三个函数。本题给出了该程序的部分 C 代码和 X86-64 汇编与机器代码。请分析给出的代码，补全空白处的内容，并回答问题。

注：汇编与机器码中的数字用 16 进制数填写

X86-64 汇编与机器代码：　　答案填写处：

```
00000000004006cd <caller>:
  4006cd: 55                      push    %rbp
  4006ce: 48 89 e5                mov     %rsp,    %rbp
  4006d1: 48 83 ec 50             sub     $0x50,   %rsp
  4006d5: 48 89 7d b8             mov     %rdi,    -0x48(%rbp)
  4006d9: 64 48 8b 04 25 28 00    mov     %fs:0x28, %rax
```

E14

```
4006e0: 00 00
4006e2: 48 89 45 f8     mov     %rax,     -0x8(%rbp)
4006e6: 31 c0           xor     %eax,     %eax
4006e8: c6 45 d0 00     movb    $0x0,     -0x30(%rbp)
4006ec: c6 45 e0 00     movb    $0x0,     (1)          (1) -0x28(%rbp)
4006f0: 48 8b 45 b8     mov     (2),      %rax         (2) -0x48(%rbp)
4006f4: 48 89 c7        mov     %rax,     %rdi
4006f7:                 callq   400510 <strlen@plt>
4006fc: 89 45 cc        mov     (3),      -0x34(%rbp)  (3) ___%eax_____
4006ff: 83 7d cc 0e     cmpl    $0xe,     -0x34(%rbp)
400703: 7f (4)          jg      400752 <caller+0x85>  (4)_____4d_____
400705: 83 7d cc 09     cmpl    $0x9,     -0x34(%rbp)
400709:                 jg      400720 <caller+0x53>
40070b: 48 8b 55 b8     mov     -0x48(%rbp), %rdx
40070f: 48 8d 45 d0     lea     (5),      %rax         (5) -0x30(%rbp)
400713: 48 89 d6        mov     %rdx,     %rsi
400716: 48 89 c7        mov     %rax,     %rdi
400719:                 callq   400500 <strcpy@plt>
40071e:                 jmp     40073b <caller+0x6e>
400720: 48 8b 45 b8     mov     -0x48(%rbp), %rax
400724: 48 8d 50 0a     lea     0xa(%rax),  %rdx
400728: 48 8d 45 d0     lea     -0x30(%rbp), %rax
40072c: 48 83 c0 10     add     (6),      %rax         (6)___0x8_____
400730: 48 89 d6        mov     %rdx,     %rsi
400733: 48 89 c7        mov     %rax,     %rdi
400736:                 callq   400500 <strcpy@plt>
40073b: ff 75 e8        pushq   -0x18(%rbp)
40073e: ff 75 e0        pushq   -0x20(%rbp)
400741: ff 75 d8        pushq   -0x28(%rbp)
400744: ff 75 d0        pushq   -0x30(%rbp)
400747: e8 (7)          callq   400666 <callee>      (7) 1a ff ff ff
40074c: 48 83 c4 20     add     $0x20,    %rsp
400750:                 jmp     400753 <caller+0x86>
400752: 90              nop
400753: 48 8b 45 f8     mov     (8),      %rax         (8) -0x8(%rbp)
400757: 64 48 33 04 25 28 00   xor     %fs:0x28,  %rax
40075e: 00 00
400760:                 je      400767 <caller+0x9a>
400762:                 callq   400520 <__stack_chk_fail@plt>
400767: c9              leaveq
400768: c3              retq
```

C 代码：

```
#include <stdio.h>
#include "string.h"
#define N   (9)
#define M   (10)

typedef union {char str_u[N]; long l;} union_e;
typedef struct {char str_s[M]; union_e u; long c;} struct_e;

void callee(struct_e s){
 char buf[M+N];
 strcpy(buf, s.str_s);
 strcat(buf, s.u.str_u);
 printf("%s \n",buf);
}

void caller(char *str){
 struct_e s;
 s.str_s[0]='\0';
 s.u.str_u[0]='\0';
 int len = strlen(str);
 if(len>=  M+N)
  (11) ;                          (11)___return____
 else if(len<N){
  strcpy(s.str_s, (12) );         (12)___str_____
 }
 else{
  strcpy(s.u.str_u, (13) );       (13)___str+M_____
 }
 callee(s);
}

int main(int argc, char *argv[]){
 caller("0123456789abcd");
 return 0;
}
```

答案填写处：

caller 函数中，变量 s 所占的内存空间为：(14)____32 字节__

该程序运行后，printf 函数是否有输出？输出结果为：(15)_____abcd____