子程序的实现细节

3170102668

缪晨露

1. 实验内容

用C语言编译器编译有函数调用的代码，做O0和O2两个选项，查看两次汇编结果，分析两个实现细节：

1. 当函数中有更进一步的代码块，如if语句的子句，并且在这些子句中有变量定义时，这些变量的空间是在什么时候分配的，是在函数开始的时候还是进入子句的时候；不同的子句中的变量，同名与否，是否会共用空间；
2. 当函数返回一个较大的结构时，返回的数据是如何安排空间的。当一个函数调用两个这样的函数的时候，空间是如何安排的。

2. 实验过程及结论

2.1. if else语句O0代码

2.1.A 源代码

```
#include<stdio.h>
void f() {
    int x = 0;

        if(x == 1) {
                int y = 2;
        } else {
                long y = 4;
        }
}
int main() {
    f();
}
```

2.1.B 反汇编代码

```
gcc -S o0.cpp
gcc -c o0.s
objdump -d o0.o
```

```
jones@ubuntu:~/Desktop/PPL_lab2$ objdump -d o0.o

o0.o:     file format elf64-x86-64
```

```
Disassembly of section .text:

0000000000000000 <_Z1fv>:
   0:   55                      push   %rbp
   1:   48 89 e5                mov    %rsp,%rbp
   4:   c7 45 f0 00 00 00 00    movl   $0x0,-0x10(%rbp)
   b:   83 7d f0 01             cmpl   $0x1,-0x10(%rbp)
   f:   75 09                   jne    1a <_Z1fv+0x1a>
  11:   c7 45 f4 02 00 00 00    movl   $0x2,-0xc(%rbp)
  18:   eb 08                   jmp    22 <_Z1fv+0x22>
  1a:   48 c7 45 f8 04 00 00    movq   $0x4,-0x8(%rbp)
  21:   00
  22:   90                      nop
  23:   5d                      pop    %rbp
  24:   c3                      retq

0000000000000025 <main>:
  25:   55                      push   %rbp
  26:   48 89 e5                mov    %rsp,%rbp
  29:   e8 00 00 00 00          callq  2e <main+0x9>
  2e:   b8 00 00 00 00          mov    $0x0,%eax
  33:   5d                      pop    %rbp
  34:   c3                      retq
```

2.1.C 部分汇编详解

```
movl   $0x0,-0x10(%rbp)
```

将x的值设为0，x位于stack中相对于base pointer -0x10的位置

```
cmpl   $0x1,-0x10(%rbp)
```

将1与x值进行比较

```
jne    1a <main+0x1a>
```

如果x不等于1，跳转到main+0x1a

```
1a: 48 c7 45 f8 04 00 00    movq   $0x4,-0x8(%rbp)
```

main+0x1a的代码是将stack中相对于base pointer 0x8位置的变量的值赋为4，对应于-0x8到-0x0的8个bytes

```
movl   $0x2,-0xc(%rbp)
```

如果x等于1,就将stack中相对于base pointer 0xC位置的变量的值赋为2，0xC相对于x的0x8中间差了4个bytes,也就是说，为y分配了32bits的空间，正好符合int的大小

2.1. D 结论

- 变量的空间是在函数开始的时候分配的
- if中的long对应-0x8,-0x0的8个bytes, else中的int对应-0xc,-0x8的4个bytes, 他们不共用空间

2.2. if else 语句O2

2.2.A 源代码

```
#include<stdio.h>

int f() {
        int x = 0;
        scanf("%d", &x);

        if(x == 1) {
                int y = 2;
                scanf("%d", &y);
                if(y == 2) printf("yes");
        } else {
                long y = 4;
                scanf("%ld", &y);
                if(y == 4) printf("no");
        }
}
int main() {
        f();
}
```

2.2.B 反汇编代码

```
jones@ubuntu:~/Desktop/PPL_lab2$ objdump -d longcode.o

longcode.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <_Z1fv>:
   0:   48 83 ec 28             sub    $0x28,%rsp
   4:   bf 00 00 00 00          mov    $0x0,%edi
   9:   64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
  10:   00 00
  12:   48 89 44 24 18          mov    %rax,0x18(%rsp)
  17:   31 c0                   xor    %eax,%eax
  19:   48 8d 74 24 0c          lea    0xc(%rsp),%rsi
  1e:   c7 44 24 0c 00 00 00    movl   $0x0,0xc(%rsp)
  25:   00
```

```
  26:   e8 00 00 00 00          callq  2b <_Z1fv+0x2b>
  2b:   83 7c 24 0c 01          cmpl   $0x1,0xc(%rsp)
  30:   74 3e                   je     70 <_Z1fv+0x70>
  32:   48 8d 74 24 10          lea    0x10(%rsp),%rsi
  37:   31 c0                   xor    %eax,%eax
  39:   bf 00 00 00 00          mov    $0x0,%edi
  3e:   48 c7 44 24 10 04 00    movq   $0x4,0x10(%rsp)
  45:   00 00
  47:   e8 00 00 00 00          callq  4c <_Z1fv+0x4c>
  4c:   48 83 7c 24 10 04       cmpq   $0x4,0x10(%rsp)
  52:   74 54                   je     a8 <_Z1fv+0xa8>
  54:   48 8b 54 24 18          mov    0x18(%rsp),%rdx
  59:   64 48 33 14 25 28 00    xor    %fs:0x28,%rdx
  60:   00 00
  62:   75 57                   jne    bb <_Z1fv+0xbb>
  64:   48 83 c4 28             add    $0x28,%rsp
  68:   c3                      retq
  69:   0f 1f 80 00 00 00 00    nopl   0x0(%rax)
  70:   48 8d 74 24 10          lea    0x10(%rsp),%rsi
  75:   31 c0                   xor    %eax,%eax
  77:   bf 00 00 00 00          mov    $0x0,%edi
  7c:   c7 44 24 10 02 00 00    movl   $0x2,0x10(%rsp)
  83:   00
  84:   e8 00 00 00 00          callq  89 <_Z1fv+0x89>
  89:   83 7c 24 10 02          cmpl   $0x2,0x10(%rsp)
  8e:   75 c4                   jne    54 <_Z1fv+0x54>
  90:   be 00 00 00 00          mov    $0x0,%esi
  95:   bf 01 00 00 00          mov    $0x1,%edi
  9a:   31 c0                   xor    %eax,%eax
  9c:   e8 00 00 00 00          callq  a1 <_Z1fv+0xa1>
  a1:   eb b1                   jmp    54 <_Z1fv+0x54>
  a3:   0f 1f 44 00 00          nopl   0x0(%rax,%rax,1)
  a8:   be 00 00 00 00          mov    $0x0,%esi
  ad:   bf 01 00 00 00          mov    $0x1,%edi
  b2:   31 c0                   xor    %eax,%eax
  b4:   e8 00 00 00 00          callq  b9 <_Z1fv+0xb9>
  b9:   eb 99                   jmp    54 <_Z1fv+0x54>
  bb:   e8 00 00 00 00          callq  c0 <_Z1fv+0xc0>

Disassembly of section .text.startup:

0000000000000000 <main>:
   0:   48 83 ec 08             sub    $0x8,%rsp
   4:   e8 00 00 00 00          callq  9 <main+0x9>
   9:   31 c0                   xor    %eax,%eax
   b:   48 83 c4 08             add    $0x8,%rsp
   f:   c3                      retq
```

2.2.C 部分汇编代码详解

1. 进入f()函数后, 在栈中就分配了空间

```
   0:   48 83 ec 28             sub    $0x28,%rsp
```

2. 在if中

```
if(x == 1) {
    int y = 2;
    scanf("%d", &y);
    if(y == 2) printf("yes");
}
```

第一句`int y = 2;`可以发现y放在了相对rsp 0x10的位置

```
  7c:   c7 44 24 10 02 00 00    movl   $0x2,0x10(%rsp)
```

3. 在else 中

```
else {
    long y = 4;
    scanf("%ld", &y);
    if(y == 4) printf("no");
}
```

第一句`long y = 4;`, y也放在了相对rsp 0x10的位置

```
  3e:   48 c7 44 24 10 04 00    movq   $0x4,0x10(%rsp)
```

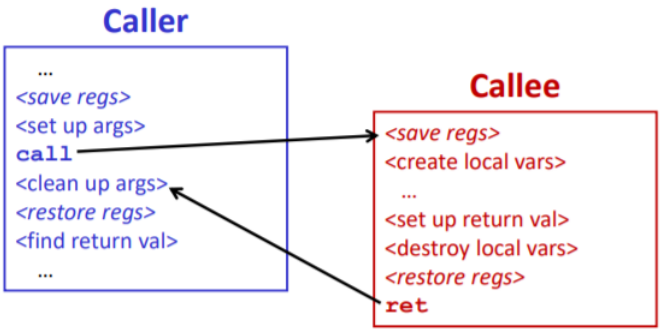
发现将y的变量名改成z依然是同样的结果

2.3.D 结论

在O2优化的情况下，在进入函数时，就会分配空间，为if else语句分配二者中较大的变量需要的空间。

if else语句中的变量会共用一部分空间。

2.3. 返回大结构体O0



Returning a larger object of class type may require more expensive copying from one memory location to another. To avoid this, an implementation may create a hidden object in the caller's stack frame, and pass the address of this object to the function. The function's return value is then copied into the hidden object

```
struct Data {
  char bytes[16];
};

Data F() {
  Data result = {};
  // generate result
  return result;
}

int main() {
  Data d = F();
}
```

may generate code equivalent to this:

```
struct Data {
  char bytes[16];
};

Data* F(Data* _hiddenAddress) {
  Data result = {};
  // copy result into hidden object
  *_hiddenAddress = result;
  return _hiddenAddress;
}

int main() {
  Data _hidden;           // create hidden object
  Data d = *F(&_hidden);  // copy the result into d
}
```

2.3.A 源代码

```
#include<stdio.h>

typedef struct _bigStruct {
    int a[1000];
} bigStruct;

bigStruct f1() {
    bigStruct bS;
    bS.a[999] = 100;
    return bS;
}
void f() {
    bigStruct bs1 = f1();
    bigStruct bs2 = f1();
    bs1.a[0] = 1;
    bs1.a[999] = 2;
    bs2.a[0] = 2;

}
int main() {
    f();
}
```

2.3.B 反汇编代码

```
jones@ubuntu:~/Desktop/PPL_lab2/struct$ objdump -d structO0.o

structO0.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <_Z2f1v>:
   0:   55                      push   %rbp
   1:   48 89 e5                mov    %rsp,%rbp
   4:   48 83 ec 20             sub    $0x20,%rsp
   8:   48 89 7d e8             mov    %rdi,-0x18(%rbp)
   c:   64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
  13:   00 00
  15:   48 89 45 f8             mov    %rax,-0x8(%rbp)
  19:   31 c0                   xor    %eax,%eax
  1b:   90                      nop
  1c:   48 8b 45 e8             mov    -0x18(%rbp),%rax
  20:   48 8b 55 f8             mov    -0x8(%rbp),%rdx
  24:   64 48 33 14 25 28 00    xor    %fs:0x28,%rdx
  2b:   00 00
  2d:   74 05                   je     34 <_Z2f1v+0x34>
  2f:   e8 00 00 00 00          callq  34 <_Z2f1v+0x34>
  34:   c9                      leaveq
  35:   c3                      retq

0000000000000036 <_Z1fv>:
  36:   55                      push   %rbp
  37:   48 89 e5                mov    %rsp,%rbp
  3a:   48 81 ec 50 1f 00 00    sub    $0x1f50,%rsp
  41:   64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
  48:   00 00
  4a:   48 89 45 f8             mov    %rax,-0x8(%rbp)
  4e:   31 c0                   xor    %eax,%eax
  50:   48 8d 85 b0 e0 ff ff    lea    -0x1f50(%rbp),%rax
  57:   48 89 c7                mov    %rax,%rdi
  5a:   e8 00 00 00 00          callq  5f <_Z1fv+0x29>
  5f:   48 8d 85 50 f0 ff ff    lea    -0xfb0(%rbp),%rax
  66:   48 89 c7                mov    %rax,%rdi
  69:   e8 00 00 00 00          callq  6e <_Z1fv+0x38>
  6e:   90                      nop
  6f:   48 8b 45 f8             mov    -0x8(%rbp),%rax
  73:   64 48 33 04 25 28 00    xor    %fs:0x28,%rax
  7a:   00 00
  7c:   74 05                   je     83 <_Z1fv+0x4d>
  7e:   e8 00 00 00 00          callq  83 <_Z1fv+0x4d>
  83:   c9                      leaveq
  84:   c3                      retq

0000000000000085 <main>:
```

```
  85:   55                      push   %rbp
  86:   48 89 e5                mov    %rsp,%rbp
  89:   e8 00 00 00 00          callq  8e <main+0x9>
  8e:   b8 00 00 00 00          mov    $0x0,%eax
  93:   5d                      pop    %rbp
  94:   c3                      retq
```

2.3.C 部分代码详解

看上述代码，发现在返回big struct时确实存在一个隐藏的指针，表明了big struct存放的位置

从f()函数开始看起

1. 进入f()函数之后，空间就被分配了

```
sub    $0x1f50,%rsp
```

2. 接着f去调用f1()函数，来获得一个大结构体

```
bigStruct bs1 = f1();
```

汇编代码中将其想将big struct存放的位置放入`rdi`寄存器，作为调用f1()函数的隐藏的指针参数

```
  50:   48 8d 85 b0 e0 ff ff    lea    -0x1f50(%rbp),%rax
  57:   48 89 c7                mov    %rax,%rdi
  5a:   e8 00 00 00 00          callq  5f <_Z1fv+0x29>
```

3. 在f1函数中

```
bigStruct bS;
return bS;
```

对应汇编代码，在caller指定的地址创建了结构体

```
   8:   48 89 7d e8             mov    %rdi,-0x18(%rbp)
   c:   64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
  13:   00 00
  15:   48 89 45 f8             mov    %rax,-0x8(%rbp)
  19:   31 c0                   xor    %eax,%eax
  1b:   90                      nop
  1c:   48 8b 45 e8             mov    -0x18(%rbp),%rax
```

4. 如果在f1函数中创建两个结构体，一个作为返回值，一个不返回

```
bigStruct f1() {
    bigStruct bS;
    bigStruct bStest;
    return bS;
}
```

对应汇编代码如下，在f1()中，为bStest分配了空间，而bS依然放在caller给的空间

```
   4:   48 81 ec c0 0f 00 00    sub    $0xfc0,%rsp
   b:   48 89 bd 48 f0 ff ff    mov    %rdi,-0xfb8(%rbp)
  12:   64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
  19:   00 00
  1b:   48 89 45 f8             mov    %rax,-0x8(%rbp)
  1f:   31 c0                   xor    %eax,%eax
  21:   90                      nop
  22:   48 8b 85 48 f0 ff ff    mov    -0xfb8(%rbp),%rax
  29:   48 8b 55 f8             mov    -0x8(%rbp),%rdx
  2d:   64 48 33 14 25 28 00    xor    %fs:0x28,%rdx
```

2.3.D 结论

因此，可知，调用返回大结构体的函数时，caller会传递一个隐藏的参数，指明这个大结构体要放的位置，callee会把大结构体放到对应的位置上

2.4. 返回大结构体O2

2.4.1 源代码

```
#include<stdio.h>

typedef struct _bigStruct {
    int a[1000];
} bigStruct;

bigStruct f1() {
    int i = 1;
    scanf("%d", &i);
    if(i == 1) printf("yes\n");
    bigStruct bS;
    bS.a[999] = 100;
    return bS;
}
void f() {
    bigStruct bs1 = f1();
    bigStruct bs2 = f1();

    scanf("%d", &bs1.a[0]);
    scanf("%d", &bs2.a[999]);
    if(bs1.a[0] == 1) {
        printf("yes");
    }
    if(bs2.a[999] == 1) {
        printf("yes");
    }
```

```
}
int main() {
    f();
}
```

2.4.2 反汇编代码

```
jones@ubuntu:~/Desktop/PPL_lab2/struct$ objdump -d structo2.o

structo2.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <_Z2f1v>:
   0:   53                      push   %rbx
   1:   48 89 fb                mov    %rdi,%rbx
   4:   bf 00 00 00 00          mov    $0x0,%edi
   9:   48 83 ec 10             sub    $0x10,%rsp
   d:   64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
  14:   00 00
  16:   48 89 44 24 08          mov    %rax,0x8(%rsp)
  1b:   31 c0                   xor    %eax,%eax
  1d:   48 8d 74 24 04          lea    0x4(%rsp),%rsi
  22:   c7 44 24 04 01 00 00    movl   $0x1,0x4(%rsp)
  29:   00
  2a:   e8 00 00 00 00          callq  2f <_Z2f1v+0x2f>
  2f:   83 7c 24 04 01          cmpl   $0x1,0x4(%rsp)
  34:   74 2a                   je     60 <_Z2f1v+0x60>
  36:   48 8b 54 24 08          mov    0x8(%rsp),%rdx
  3b:   64 48 33 14 25 28 00    xor    %fs:0x28,%rdx
  42:   00 00
  44:   c7 83 9c 0f 00 00 64    movl   $0x64,0xf9c(%rbx)
  4b:   00 00 00
  4e:   48 89 d8                mov    %rbx,%rax
  51:   75 19                   jne    6c <_Z2f1v+0x6c>
  53:   48 83 c4 10             add    $0x10,%rsp
  57:   5b                      pop    %rbx
  58:   c3                      retq
  59:   0f 1f 80 00 00 00 00    nopl   0x0(%rax)
  60:   bf 00 00 00 00          mov    $0x0,%edi
  65:   e8 00 00 00 00          callq  6a <_Z2f1v+0x6a>
  6a:   eb ca                   jmp    36 <_Z2f1v+0x36>
  6c:   e8 00 00 00 00          callq  71 <_Z2f1v+0x71>
  71:   0f 1f 44 00 00          nopl   0x0(%rax,%rax,1)
  76:   66 2e 0f 1f 84 00 00    nopw   %cs:0x0(%rax,%rax,1)
  7d:   00 00 00

0000000000000080 <_Z1fv>:
  80:   48 81 ec 58 1f 00 00    sub    $0x1f58,%rsp
  87:   48 89 e7                mov    %rsp,%rdi
  8a:   64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
```

```
  91:   00 00
  93:   48 89 84 24 48 1f 00    mov    %rax,0x1f48(%rsp)
  9a:   00
  9b:   31 c0                   xor    %eax,%eax
  9d:   e8 00 00 00 00          callq  a2 <_Z1fv+0x22>
  a2:   48 8d bc 24 a0 0f 00    lea    0xfa0(%rsp),%rdi
  a9:   00
  aa:   e8 00 00 00 00          callq  af <_Z1fv+0x2f>
  af:   48 89 e6                mov    %rsp,%rsi
  b2:   bf 00 00 00 00          mov    $0x0,%edi
  b7:   31 c0                   xor    %eax,%eax
  b9:   e8 00 00 00 00          callq  be <_Z1fv+0x3e>
  be:   48 8d b4 24 3c 1f 00    lea    0x1f3c(%rsp),%rsi
  c5:   00
  c6:   31 c0                   xor    %eax,%eax
  c8:   bf 00 00 00 00          mov    $0x0,%edi
  cd:   e8 00 00 00 00          callq  d2 <_Z1fv+0x52>
  d2:   83 3c 24 01             cmpl   $0x1,(%rsp)
  d6:   74 40                   je     118 <_Z1fv+0x98>
  d8:   83 bc 24 3c 1f 00 00    cmpl   $0x1,0x1f3c(%rsp)
  df:   01
  e0:   74 1e                   je     100 <_Z1fv+0x80>
  e2:   48 8b 84 24 48 1f 00    mov    0x1f48(%rsp),%rax
  e9:   00
  ea:   64 48 33 04 25 28 00    xor    %fs:0x28,%rax
  f1:   00 00
  f3:   75 36                   jne    12b <_Z1fv+0xab>
  f5:   48 81 c4 58 1f 00 00    add    $0x1f58,%rsp
  fc:   c3                      retq
  fd:   0f 1f 00                nopl   (%rax)
 100:   be 00 00 00 00          mov    $0x0,%esi
 105:   bf 01 00 00 00          mov    $0x1,%edi
 10a:   31 c0                   xor    %eax,%eax
 10c:   e8 00 00 00 00          callq  111 <_Z1fv+0x91>
 111:   eb cf                   jmp    e2 <_Z1fv+0x62>
 113:   0f 1f 44 00 00          nopl   0x0(%rax,%rax,1)
 118:   be 00 00 00 00          mov    $0x0,%esi
 11d:   bf 01 00 00 00          mov    $0x1,%edi
 122:   31 c0                   xor    %eax,%eax
 124:   e8 00 00 00 00          callq  129 <_Z1fv+0xa9>
 129:   eb ad                   jmp    d8 <_Z1fv+0x58>
 12b:   e8 00 00 00 00          callq  130 <_Z1fv+0xb0>

Disassembly of section .text.startup:

0000000000000000 <main>:
   0:   48 83 ec 08             sub    $0x8,%rsp
   4:   e8 00 00 00 00          callq  9 <main+0x9>
   9:   31 c0                   xor    %eax,%eax
   b:   48 83 c4 08             add    $0x8,%rsp
   f:   c3                      retq
```

2.4.3 部分汇编代码详解

总体来看区别不大，O2优化后倾向于将代码内联

但是如果在f1()函数中创建两个结构体，一个作为返回值，一个不返回

```
bigStruct f1() {
    int i = 1;

    scanf("%d", &i);
    if(i == 1) printf("yes\n");

    bigStruct bS;
    bigStruct bStest;
    bS.a[999] = 100;
    return bS;
}
```

则对应汇编中,并不会为这个结构体分配空间

```
_Z2f1v:
    pushq   %rbx
    movq    %rdi, %rbx
    movl    $.LC0, %edi
    subq    $16, %rsp
    .cfi_def_cfa_offset 32
    movq    %fs:40, %rax
    movq    %rax, 8(%rsp)
    xorl    %eax, %eax
    leaq    4(%rsp), %rsi
    movl    $1, 4(%rsp)
    call    scanf
    cmpl    $1, 4(%rsp)
    je  .L6
    ...
```