PPL HW- 子程序实现细节

3170102492 夏豪诚

> Modified Date: 2019.12.7 编译工具：gcc (Ubuntu 9.2.1-9ubuntu2) 9.2.1 20191008 编译环境：Ubuntu 19.10

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 实验步骤

编写测试用c语言源文件person.c.(内容见附录)

而后在同目录下执行如下命令：

```
gcc -O0 -S person.c >  person_O0.asm
gcc -O2 -S person.c >  person_O2.asm
```

而后分析汇编结果,即文件person_o0.asm和person_02.asm(内容见附录)。

2.1 实现细节分析一

2.1.1 对于O0选项

我们需要分析的第一个细节是：

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；

找到对应的汇编码段：

```
create:
.LFB1:
    .cfi_startproc
    endbr64
    pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
```

```
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
    pushq   %rbx
    ; stack空间申请
    subq    $136, %rsp
    .cfi_offset 3, -24
    movq    %rdi, -136(%rbp)
    movl    %esi, -140(%rbp)
    movq    %fs:40, %rax
    movq    %rax, -24(%rbp)
    xorl    %eax, %eax
    ; cmpl和jne语句共同实现了对if(sex==0)的判断
    cmpl    $0, -140(%rbp)
    jne .L5
    ; 根据源码，此处汇编分别代表的语句是
    ; int sameName = 0; int sameNameV = 0; int diffName0 = 0;
    movl    $0, -108(%rbp) ; int sameName = 0;
    movl    $0, -104(%rbp) ; int sameNameV = 0;
    movl    $0, -100(%rbp) ; int diffName0 = 0;
    movl    $20, -64(%rbp)
    movl    $0, -48(%rbp)
    movl    $1, -52(%rbp)
    movb    $84, -60(%rbp)
    movl    $180, -44(%rbp)
    movl    $120, -40(%rbp)
    movq    -64(%rbp), %rax
    movq    -56(%rbp), %rdx
    movq    %rax, -96(%rbp)
    movq    %rdx, -88(%rbp)
    movq    -48(%rbp), %rax
    movq    %rax, -80(%rbp)
    movl    -40(%rbp), %eax
    movl    %eax, -72(%rbp)
    jmp .L6
.L5:
    cmpl    $1, -140(%rbp)
    jne .L6
    ; 根据源码，此处汇编分别代表的语句是
    ; int sameName = 1; int sameNameV = 0; int diffName0 = 1;
    movl    $1, -120(%rbp) ; int sameName = 1;
    movl    $0, -116(%rbp) ; int sameNameV = 0;
    movl    $1, -112(%rbp) ; int diffName0 = 1;
    ; 结构体的位置是相同的
    movl    $20, -64(%rbp)
    movl    $1, -48(%rbp)
    movl    $0, -52(%rbp)
    movb    $76, -60(%rbp)
    movl    $170, -44(%rbp)
    movl    $100, -40(%rbp)
    movq    -64(%rbp), %rax
    movq    -56(%rbp), %rdx
    movq    %rax, -96(%rbp)
    movq    %rdx, -88(%rbp)
```

```
    movq    -48(%rbp), %rax
    movq    %rax, -80(%rbp)
    movl    -40(%rbp), %eax
    movl    %eax, -72(%rbp)
.L6:
    movq    -136(%rbp), %rax
    movq    -96(%rbp), %rcx
    movq    -88(%rbp), %rbx
    movq    %rcx, (%rax)
    movq    %rbx, 8(%rax)
    movq    -80(%rbp), %rdx
    movq    %rdx, 16(%rax)
    movl    -72(%rbp), %edx
    movl    %edx, 24(%rax)
    movq    -24(%rbp), %rax
    xorq    %fs:40, %rax
    je  .L8
    call    __stack_chk_fail@PLT
.L8:
    movq    -136(%rbp), %rax
    addq    $136, %rsp
    popq    %rbx
    popq    %rbp
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
```

在create函数中，我们可以看到`sub $0x88,%rsp`一句即为对栈空间的申请，并且需要注意的是在整个create函数中只进行了这一次申请。因此我们可以回答 ***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*** 这一问题，答案是变量空间为在函数开始的时候就已经被分配完成了。

同时我们看到在之后对于同名获知不同名的变量进行操作的时候，其指向的空间都是不相同的，因而可以回答 ***不同的子句中的变量，同名与否，是否会共用空间*** 这一问题，答案是对于整数等变量（在测试代码中的对应于sameName等变量）来说不会；然而对于结构体来书其空间是共用的。

2.1.2 对于O2选项

同样在开启O2编译选项后的带的汇编码中找到对应`create`函数的部分：

```
0000000000001140 <create>:
    1140:   f3 0f 1e fa             endbr64
    1144:   48 89 f8                mov    %rdi,%rax
    1147:   85 f6                   test   %esi,%esi
    1149:   74 4d                   je     1198 <create+0x58>
    114b:   83 fe 01                cmp    $0x1,%esi
    114e:   b9 64 00 00 00          mov    $0x64,%ecx
    1153:   ba 00 00 00 00          mov    $0x0,%edx
    1158:   bf aa 00 00 00          mov    $0xaa,%edi
    115d:   0f 44 d1                cmove  %ecx,%edx
    1160:   b9 00 00 00 00          mov    $0x0,%ecx
    1165:   41 b9 4c 00 00 00       mov    $0x4c,%r9d
    116b:   41 b8 00 00 00 00       mov    $0x0,%r8d
```

```
    1171:   0f 44 cf                cmove  %edi,%ecx
    1174:   bf 00 00 00 00          mov    $0x0,%edi
    1179:   41 0f 44 f9             cmove  %r9d,%edi
    117d:   c7 00 14 00 00 00       movl   $0x14,(%rax)
    1183:   40 88 78 04             mov    %dil,0x4(%rax)
    1187:   44 89 40 0c             mov    %r8d,0xc(%rax)
    118b:   89 70 10                mov    %esi,0x10(%rax)
    118e:   89 48 14                mov    %ecx,0x14(%rax)
    1191:   89 50 18                mov    %edx,0x18(%rax)
    1194:   c3                      retq
    1195:   0f 1f 00                nopl   (%rax)
    1198:   ba 78 00 00 00          mov    $0x78,%edx
    119d:   b9 b4 00 00 00          mov    $0xb4,%ecx
    11a2:   41 b8 01 00 00 00       mov    $0x1,%r8d
    11a8:   bf 54 00 00 00          mov    $0x54,%edi
    11ad:   eb ce                   jmp    117d <create+0x3d>
    11af:   90                      nop
```

可以看到的是变量的空间同样在进入时就被定义了。而在此时结构体仍然会共用空间。

2.2 实现细节分析二

2.2.1 对于O0选项

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?

根据main函数部分在开头为局部变量sex0和sex1进行赋值操作时所使用的地址，我们可以知道，返回结构的空间实际上已经由调用者为被调用者开好了．为了印证这一点，我在main函数中添加了如下语句:

```
printf("%d",worker0.age);
```

用以检查调用printf@PLT时的参数来源及其位置，于是在汇编中看到了如下代码:

```
    movl    -80(%rbp), %eax
    movl    %eax, %esi
    leaq    .LC0(%rip), %rdi
    movl    $0, %eax
    call    printf@PLT
```

足以证明，main函数在开始时的`subq $96, %rsp`语句已经为返回的结构体开辟好了空间．

接下来我们观察两个结构的相对位置:

```
movl    -80(%rbp), %eax
movl    %eax, %esi
leaq    .LC0(%rip), %rdi
movl    $0, %eax
call    printf@PLT
; 添加了  printf("%d",worker1.age);  语句
movl    -48(%rbp), %eax
movl    %eax, %esi
leaq    .LC0(%rip), %rdi
movl    $0, %eax
call    printf@PLT
```

可以看到在添加了另一条输出另一个结构体信息的语句后，函数同样从已开辟的堆栈中取出数据．并且两者的offset为32，先定义先返回的结构体所在地址数值较小，而第二个所在的地址数值更大．

为了印证这一点可以看到靠上位置的如下汇编：

```
; 为调用函数设置栈桢的位置  -48(%rbp) -> %rax -> %rdi
leaq    -80(%rbp), %rax
; 通过  %esi  传入参数
movl    -88(%rbp), %edx
movl    %edx, %esi
movq    %rax, %rdi
call    create
; 为调用函数设置栈桢的位置  -48(%rbp) -> %rax -> %rdi
leaq    -48(%rbp), %rax
; 通过  %esi  传入参数
movl    -84(%rbp), %edx
movl    %edx, %esi
movq    %rax, %rdi
call    create
```

2.2.2 对于O2选项

而开启了O2选项之后，被调用函数返回结果被完全展开了，本应通过返回值的获取，再从返回值中取出年龄相关的信息，而在此情况下，直接被设置成为给printf@PLT传入对应的常数．

如下所示：

```
movl    $20, %edx
movl    $1, %edi
xorl    %eax, %eax
leaq    .LC0(%rip), %rsi
call    __printf_chk@PLT
movl    $20, %edx
leaq    .LC0(%rip), %rsi
xorl    %eax, %eax
movl    $1, %edi
call    __printf_chk@PLT
```

因此也就没有了返回数据空间安排的问题.

但是我们可以对函数进行如下声明:

```
struct person create(int sex) __attribute__((noinline));
```

控制其展开，由此得到的结果和对于O0选项的分析相同,都是由调用者为被调用者开好空间，先定义先返回的结构体所在地址数值较小，而第二个所在的地址数值更大:

```
    movq    %rax, 72(%rsp)
    xorl    %eax, %eax
    movq    %rsp, %rdi
    call    create
    leaq    32(%rsp), %rdi
    movl    $1, %esi
    call    create
    movl    (%rsp), %edx
    movl    $1, %edi
    xorl    %eax, %eax
    leaq    .LC0(%rip), %rsi
    call    __printf_chk@PLT
    movl    32(%rsp), %edx
    xorl    %eax, %eax
    movl    $1, %edi
    leaq    .LC0(%rip), %rsi
    call    __printf_chk@PLT
```

3 分析结论

对于这两个实现细节我们在函数声明是添加`__attribute__((noinline));`后得到的结论是相同的:

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函数开始的时候就分配好的，而不同的子句中的结构体变量，同名与否，都会共用空间;
2. 当函数返回一个较大的结构时，返回的数据空间由函数调用者在函数调用前已经分配完成。当一个函数调用两个这样的函数的时候，空间也是由函数调用者在函数调用前已经分配完成，同时先定义先返回的结构体所在地址数值较小，而第二个所在的地址数值更大。

除了上述结论外，我在实验中发现，在O0选项时定义在子句内部的整数等类型的变量不会共用空间，而在O2时这些变量被直接优化去除了。

附录:

person_o0.asm

```
    .file   "person.c"
    .text
    .section    .rodata
.LC0:
    .string "%d"
```

```
    .text
    .globl  main
    .type   main, @function
main:
.LFB0:
    .cfi_startproc
    endbr64
    pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
    subq    $96, %rsp
    movq    %fs:40, %rax
    movq    %rax, -8(%rbp)
    xorl    %eax, %eax
    movl    $0, -88(%rbp)
    movl    $1, -84(%rbp)
    leaq    -80(%rbp), %rax
    movl    -88(%rbp), %edx
    movl    %edx, %esi
    movq    %rax, %rdi
    call    create
    leaq    -48(%rbp), %rax
    movl    -84(%rbp), %edx
    movl    %edx, %esi
    movq    %rax, %rdi
    call    create
    movl    -80(%rbp), %eax
    movl    %eax, %esi
    leaq    .LC0(%rip), %rdi
    movl    $0, %eax
    call    printf@PLT
    movl    -48(%rbp), %eax
    movl    %eax, %esi
    leaq    .LC0(%rip), %rdi
    movl    $0, %eax
    call    printf@PLT
    movl    $0, %eax
    movq    -8(%rbp), %rcx
    xorq    %fs:40, %rcx
    je  .L3
    call    __stack_chk_fail@PLT
.L3:
    leave
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
.LFE0:
    .size   main, .-main
    .globl  create
    .type   create, @function
create:
```

```
.LFB1:
    .cfi_startproc
    endbr64
    pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
    pushq   %rbx
    subq    $136, %rsp
    .cfi_offset 3, -24
    movq    %rdi, -136(%rbp)
    movl    %esi, -140(%rbp)
    movq    %fs:40, %rax
    movq    %rax, -24(%rbp)
    xorl    %eax, %eax
    cmpl    $0, -140(%rbp)
    jne .L5
    movl    $0, -108(%rbp)
    movl    $0, -104(%rbp)
    movl    $0, -100(%rbp)
    movl    $20, -64(%rbp)
    movl    $0, -48(%rbp)
    movl    $1, -52(%rbp)
    movb    $84, -60(%rbp)
    movl    $180, -44(%rbp)
    movl    $120, -40(%rbp)
    movq    -64(%rbp), %rax
    movq    -56(%rbp), %rdx
    movq    %rax, -96(%rbp)
    movq    %rdx, -88(%rbp)
    movq    -48(%rbp), %rax
    movq    %rax, -80(%rbp)
    movl    -40(%rbp), %eax
    movl    %eax, -72(%rbp)
    jmp .L6
.L5:
    cmpl    $1, -140(%rbp)
    jne .L6
    movl    $1, -120(%rbp)
    movl    $0, -116(%rbp)
    movl    $1, -112(%rbp)
    movl    $20, -64(%rbp)
    movl    $1, -48(%rbp)
    movl    $0, -52(%rbp)
    movb    $76, -60(%rbp)
    movl    $170, -44(%rbp)
    movl    $100, -40(%rbp)
    movq    -64(%rbp), %rax
    movq    -56(%rbp), %rdx
    movq    %rax, -96(%rbp)
    movq    %rdx, -88(%rbp)
    movq    -48(%rbp), %rax
```

```
    movq    %rax, -80(%rbp)
    movl    -40(%rbp), %eax
    movl    %eax, -72(%rbp)
.L6:
    movq    -136(%rbp), %rax
    movq    -96(%rbp), %rcx
    movq    -88(%rbp), %rbx
    movq    %rcx, (%rax)
    movq    %rbx, 8(%rax)
    movq    -80(%rbp), %rdx
    movq    %rdx, 16(%rax)
    movl    -72(%rbp), %edx
    movl    %edx, 24(%rax)
    movq    -24(%rbp), %rax
    xorq    %fs:40, %rax
    je  .L8
    call    __stack_chk_fail@PLT
.L8:
    movq    -136(%rbp), %rax
    addq    $136, %rsp
    popq    %rbx
    popq    %rbp
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
.LFE1:
    .size   create, .-create
    .ident  "GCC: (Ubuntu 9.2.1-9ubuntu2) 9.2.1 20191008"
    .section    .note.GNU-stack,"",@progbits
    .section    .note.gnu.property,"a"
    .align 8
    .long    1f - 0f
    .long    4f - 1f
    .long    5
0:
    .string  "GNU"
1:
    .align 8
    .long    0xc0000002
    .long    3f - 2f
2:
    .long    0x3
3:
    .align 8
4:
```

person_o2.asm

```
    .file   "person.c"
    .text
```

```
    .p2align 4
    .globl  create
    .type   create, @function
create:
.LFB24:
    .cfi_startproc
    endbr64
    movq    %rdi, %rax
    testl   %esi, %esi
    je  .L3
    cmpl    $1, %esi
    movl    $100, %ecx
    movl    $0, %edx
    movl    $170, %edi
    cmove   %ecx, %edx
    movl    $0, %ecx
    movl    $76, %r9d
    movl    $0, %r8d
    cmove   %edi, %ecx
    movl    $0, %edi
    cmove   %r9d, %edi
.L2:
    movl    $20, (%rax)
    movb    %dil, 4(%rax)
    movl    %r8d, 12(%rax)
    movl    %esi, 16(%rax)
    movl    %ecx, 20(%rax)
    movl    %edx, 24(%rax)
    ret
    .p2align 4,,10
    .p2align 3
.L3:
    movl    $120, %edx
    movl    $180, %ecx
    movl    $1, %r8d
    movl    $84, %edi
    jmp .L2
    .cfi_endproc
.LFE24:
    .size   create, .-create
    .section    .rodata.str1.1,"aMS",@progbits,1
.LC0:
    .string "%d"
    .section    .text.startup,"ax",@progbits
    .p2align 4
    .globl  main
    .type   main, @function
main:
.LFB23:
    .cfi_startproc
    endbr64
    subq    $88, %rsp
    .cfi_def_cfa_offset 96
```

```
    xorl    %esi, %esi
    movq    %fs:40, %rax
    movq    %rax, 72(%rsp)
    xorl    %eax, %eax
    movq    %rsp, %rdi
    call    create
    leaq    32(%rsp), %rdi
    movl    $1, %esi
    call    create
    movl    (%rsp), %edx
    movl    $1, %edi
    xorl    %eax, %eax
    leaq    .LC0(%rip), %rsi
    call    __printf_chk@PLT
    movl    32(%rsp), %edx
    xorl    %eax, %eax
    movl    $1, %edi
    leaq    .LC0(%rip), %rsi
    call    __printf_chk@PLT
    movq    72(%rsp), %rax
    xorq    %fs:40, %rax
    jne .L8
    xorl    %eax, %eax
    addq    $88, %rsp
    .cfi_remember_state
    .cfi_def_cfa_offset 8
    ret
.L8:
    .cfi_restore_state
    call    __stack_chk_fail@PLT
    .cfi_endproc
.LFE23:
    .size   main, .-main
    .ident  "GCC: (Ubuntu 9.2.1-9ubuntu2) 9.2.1 20191008"
    .section    .note.GNU-stack,"",@progbits
    .section    .note.gnu.property,"a"
    .align 8
    .long    1f - 0f
    .long    4f - 1f
    .long    5
0:
    .string  "GNU"
1:
    .align 8
    .long    0xc0000002
    .long    3f - 2f
2:
    .long    0x3
3:
    .align 8
4:
```

person.c

```
// file: person.c
#include<stdio.h>
struct person{
    int age;
    char name[8];
    int male;   // 0 - > not male & 1 -> male
    int female; // 0 - > not female & 1 -> female
    int height;
    int weight;
};

struct person create(int sex) __attribute__((noinline));

int main(){
    int sex0 = 0;
    int sex1 = 1;
    struct person worker0, worker1;
    worker0 = create(sex0);
    worker1 = create(sex1);
    printf("%d",worker0.age);
    printf("%d",worker1.age);
    return 0;
}
struct person create(int sex){
    struct person worker;
    if (sex == 0){
        struct person male;
        // check whehter the same name vari will use the same space
        int sameName = 0;
        int sameNameV = 0;
        int diffName0 = 0;
        male.age = 20;
        male.female = 0;
        male.male = 1;
        male.name[0] = 'T';
        male.height = 180;
        male.weight = 120;
        worker = male;
    }
    else if (sex == 1){
        // check whehter the same name vari will use the same space
        struct person female;
        int sameName = 1;
        int sameNameV = 0;
        int diffName1 = 1;
        female.age = 20;
        female.female = 1;
        female.male = 0;
        female.name[0] = 'L';
```

```
        female.height = 170;
        female.weight = 100;
        worker = female;
    }
    return worker;
}
```