

作业-2

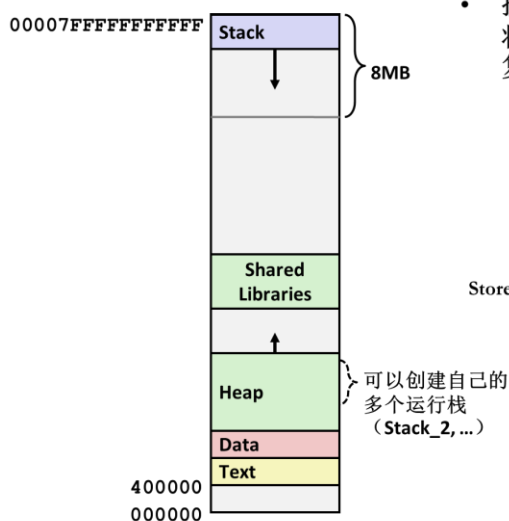
1. 过程调用以及返回的顺序在一般情况下都是“过程返回的顺序恰好与调用顺序相反”，但是我们可以利用汇编以及对运行栈的理解来编写汇编过程打破这一惯例。

有如下汇编代码（x86-32 架构），其中 GET 过程唯一的输入参数是一个用于存储当前处理器以及栈信息的内存块地址（假设该内存块的空间足够大），而 SET 过程则用于恢复被 GET 过程所保存的处理器及栈信息，其唯一的输入参数也是该内存块地址。在理解代码的基础上，回答下列问题：

GET:	SET:
movl 4(%esp), %eax #(A)	movl 4(%esp), %eax
...	...
movl %edi, 20(%eax)	movl 20(%eax), %edi
movl %esi, 24(%eax)	movl 24(%eax), %esi
movl %ebp, 28(%eax)	movl 28(%eax), %ebp
movl %ebx, 36(%eax)	movl 36(%eax), %ebx
movl %edx, 40(%eax)	movl 40(%eax), %edx
movl %ecx, 44(%eax)	movl 44(%eax), %ecx
movl \$1, 48(%eax)	movl _____(%eax), %esp #(D)
movl (%esp), %ecx #(B)	pushl 60(%eax) #(E)
movl %ecx, 60(%eax)	movl 48(%eax), %eax
leal 4(%esp), %ecx #(C)	ret
movl %ecx, 72(%eax)	
movl 44(%eax), %ecx	
movl \$0, %eax	
ret	

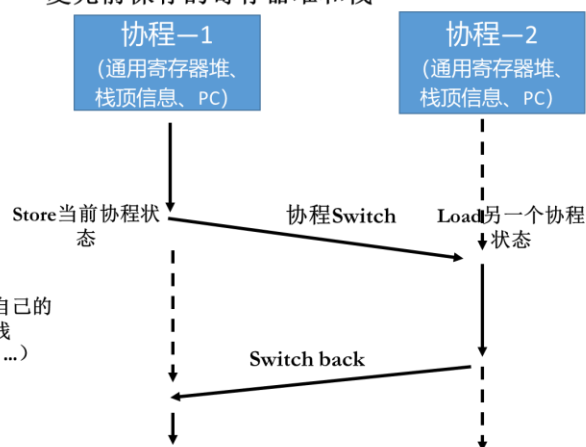
- 1.1 SET 过程的返回地址是什么，其返回值是多少？
 - 1.2 代码段中的 (A) 指令执行后，eax 中存放的是什么？(B) 指令执行后，ecx 中存放的是什么？(C) 指令的作用是什么？(E) 指令的作用是什么？并将 (D) 指令补充完整。
-
2. 上述的思路也可用于“协程”任务切换（上课讲过，PPT 加下图），

Linux进程的内存布局 (x86-64)



协程：轻量级任务（OS不可见）

- 拥有自己的寄存器堆和栈，协程调度切换时，将寄存器堆和栈保存起来，在切换回来时恢复先前保存的寄存器堆和栈



切换协程用的过程的代码如下：

ribs_swapcurcontext:

```

movq    current_ctx, %rsi
/* Save the preserved registers. */
movq    %rsp, 0(%rsi)
movq    %rbx, 8(%rsi)
movq    %rbp, 16(%rsi)
movq    %r12, 24(%rsi)
movq    %r13, 32(%rsi)
movq    %r14, 40(%rsi)
movq    %r15, 48(%rsi)
movq    %rdi, current_ctx
/* Load the new stack pointer and the preserved registers.*/
movq    0(%rdi), %rsp
movq    8(%rdi), %rbx
movq    16(%rdi), %rbp
movq    24(%rdi), %r12
movq    32(%rdi), %r13
movq    40(%rdi), %r14
movq    48(%rdi), %r15
ret

```

2.1 请简要介绍其工作原理；

2.2 为何 save/load 的通用寄存器个数这么少（x86-64 有 16 个通用寄存器）？

3. 请对照下列的 C 代码与对应的汇编代码, 解释下 C 函数返回 struct 类型是如何实现的? 可以通过画出 call return_struct 时栈的 layout 以及传参情况, 并辅以说明来解释。

```
typedef struct{
int age; int bye; int coo; int ddd; int eee;
} TEST_Struct;
int i = 2;
TEST_Struct __cdecl return_struct(int n)
{
    TEST_Struct local_struct;
    local_struct.age = n;
    local_struct.bye = n;
    local_struct.coo = 2*n;
    local_struct.ddd = n;
    local_struct.eee = n;
    i = local_struct.eee + local_struct.age *2 ;
    return local_struct;
}
int function1()
{
    TEST_Struct main_struct = return_struct(i);
    return 0;
}
```

```

return_struct:
    movq    %rdi, %rax
    movl    %esi, (%rdi)
    movl    %esi, 4(%rdi)
    leal    (%rsi,%rsi), %edx
    movl    %edx, 8(%rdi)
    movl    %esi, 12(%rdi)
    movl    %esi, 16(%rdi)
    addl    %edx, %esi
    movl    %esi, i(%rip)
    ret
function1:
    subq    $32, %rsp
    movl    i(%rip), %esi
    movq    %rsp, %rdi
    call    return_struct
    movl    $0, %eax
    addq    $32, %rsp
    ret

```

4. 请分别对照下列的 C 代码与对应的汇编代码，解释下 C 函数是如何传入 struct 类型参数的？可以通过画出 call input_struct 时栈的 layout，并辅以说明来解释。

4.1 gcc -Og ...

```

typedef struct{
    int age; int bye; int coo; int ddd; int eee;
} TEST_Struct;
int i = 2;
int input_struct(TEST_Struct in_struct)
{
    return in_struct.eee + in_struct.age*2 ;
}
int function2()
{
    TEST_Struct main_struct;
    main_struct.age = i;
    main_struct.bye = i;
    main_struct.coo = 2*i;
    main_struct.ddd = i;
    main_struct.eee = i;
    return input_struct(main_struct);
}

```

```

input_struct:
    movl    8(%rsp), %eax    #age
    addl    %eax, %eax
    addl    24(%rsp), %eax   #eee
    ret

function2:
    subq    $56, %rsp
    movl    i(%rip), %eax
    movl    %eax, 24(%rsp)   #age
    movl    %eax, 28(%rsp)   #bye
    leal    (%rax,%rax), %edx
    movl    %edx, 32(%rsp)   #coo
    movl    %eax, 36(%rsp)   #ddd
    movq    24(%rsp), %rdx
    movq    %rdx, (%rsp)     #age/bye
    movq    32(%rsp), %rdx
    movq    %rdx, 8(%rsp)    #coo/ddd
    movl    %eax, 16(%rsp)   #eee
    call    input_struct
    addq    $56, %rsp
    ret

```

4.2 gcc -O1/2 ...

C 代码不变。汇编如下：

```

input_struct:
    movl    24(%rsp), %eax
    movl    8(%rsp), %edx
    leal    (%rax,%rdx,2), %eax
    ret

function2:
    movl    i(%rip), %eax
    leal    (%rax,%rax,2), %eax
    ret

```

请分析这段代码，编译器做了什么优化工作。

4.3 如果在上面的 C 代码的 `int input_struct (...)` 声明前加上 `static`, gcc -O1/2 ... 编译后的代码如下：

```

function2:
    movl    i(%rip), %eax
    leal    (%rax,%rax,2), %eax
    ret

```

请分析这段代码，编译器做了什么优化工作。