

同济大学计算机系

操作系统实验报告



学 号 2152809

姓 名 曾崇然

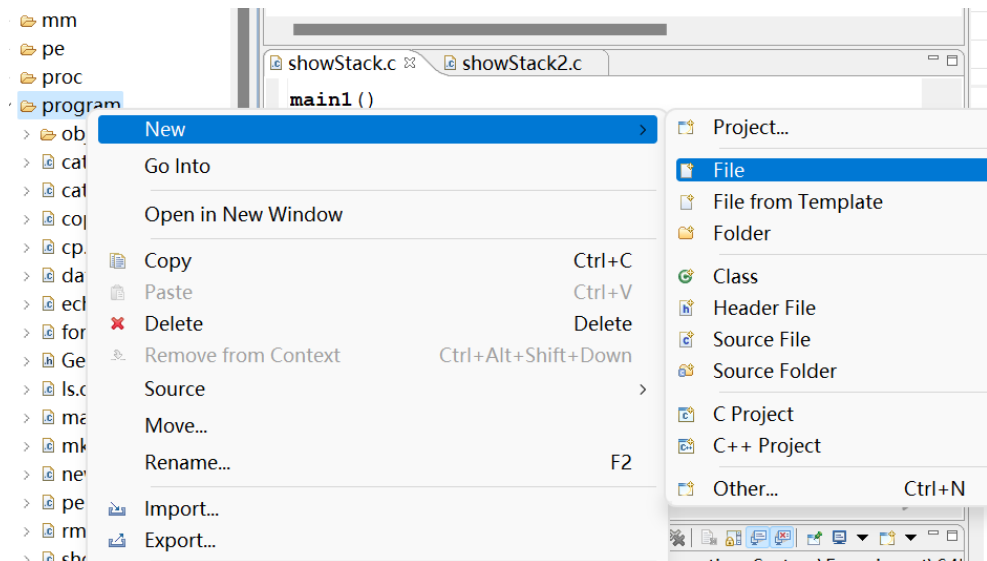
专 业 计算机科学与技术

授课老师 方钰老师

一. UNIX V6++自定义程序的添加、编译、连接和运行

1. 自定义程序的添加

a. 文件的添加与编写:



```
showStack.c x
#include <stdio.h>

int version = 1;

main1 ()
{
    int a,b,result;
    a=1;
    b=2;
    result=sum(a,b);
    printf("result=%d\n",result);
}

int sum(var1, var2)
{
    int count;
    version=2;
    count=var1+var2;
    return(count);
}
```

b. 修改配置文件:

```
$(TARGET)\sigtest.exe \
$(TARGET)\stack.exe \
$(TARGET)\malloc.exe \
$(TARGET)\showStack.exe \
$(TARGET)\showStack2.exe
```

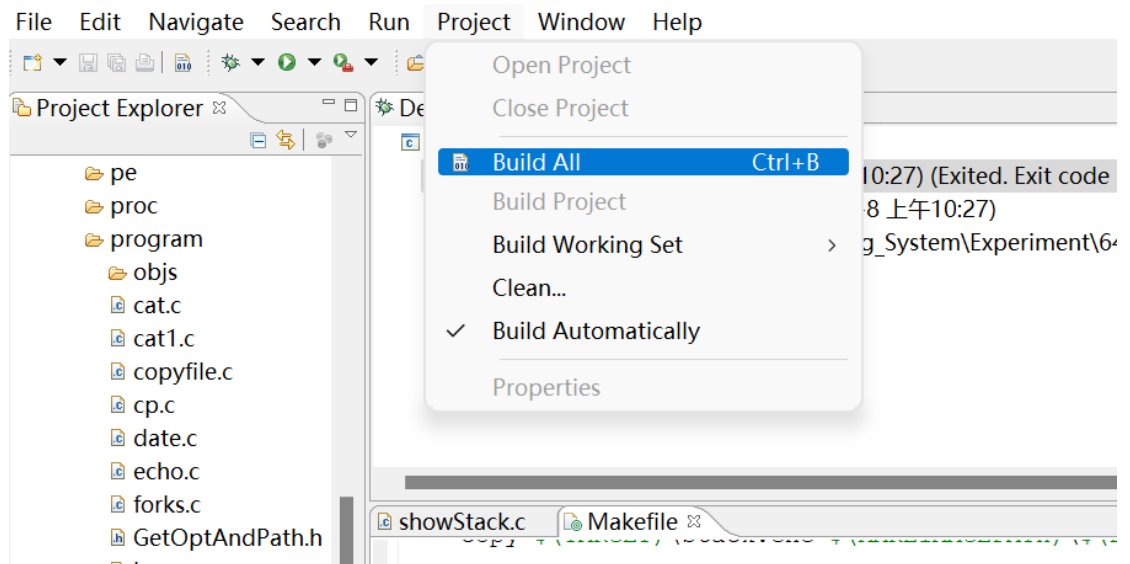
```
$(TARGET)\showStack.exe : showStack.c
$(CC) $(CFLAGS) -I"$(INCLUDE)" -I"$(LIB_INCLUDE)" $< -e _main1 $(V6++LIB) -o $@
copy $(TARGET)\showStack.exe $(MAKEIMAGEPATH)\$(BIN)\showStack.exe

$(TARGET)\showStack2.exe : showStack2.c
$(CC) $(CFLAGS) -I"$(INCLUDE)" -I"$(LIB_INCLUDE)" $< -e _main1 $(V6++LIB) -o $@
copy $(TARGET)\showStack2.exe $(MAKEIMAGEPATH)\$(BIN)\showStack2.exe
```

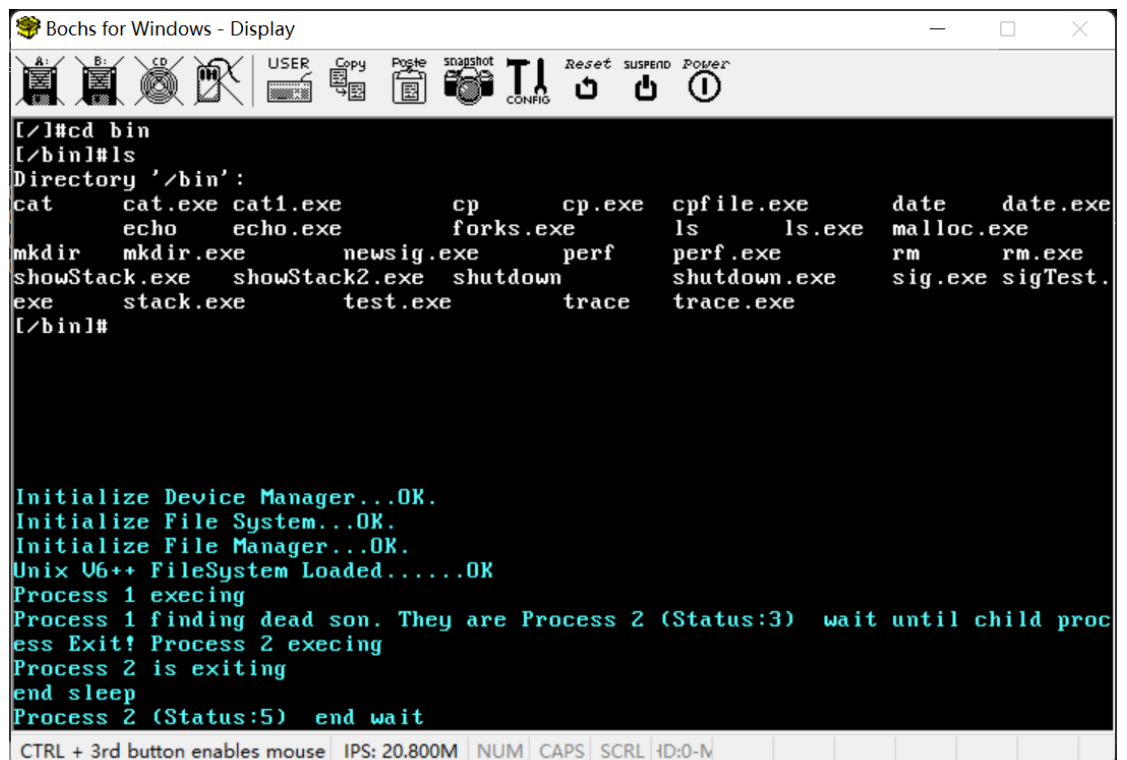
2. 程序的编译

a. 进行编译:

Debug - oos/src/program/Makefile - Eclipse

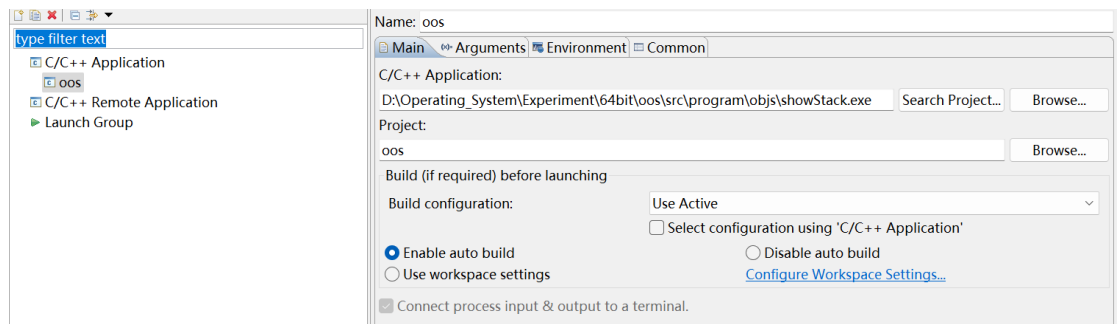


b. 验证执行文件是否生成:

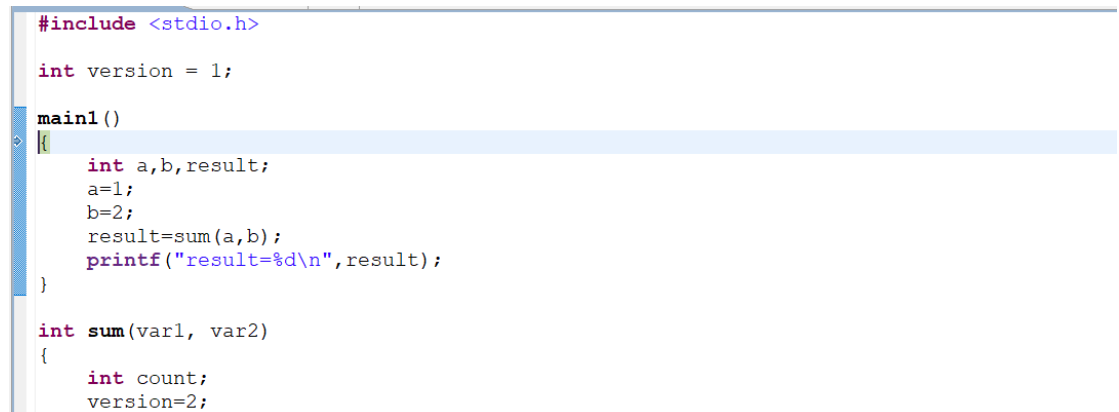


3. 程序的调试和运行

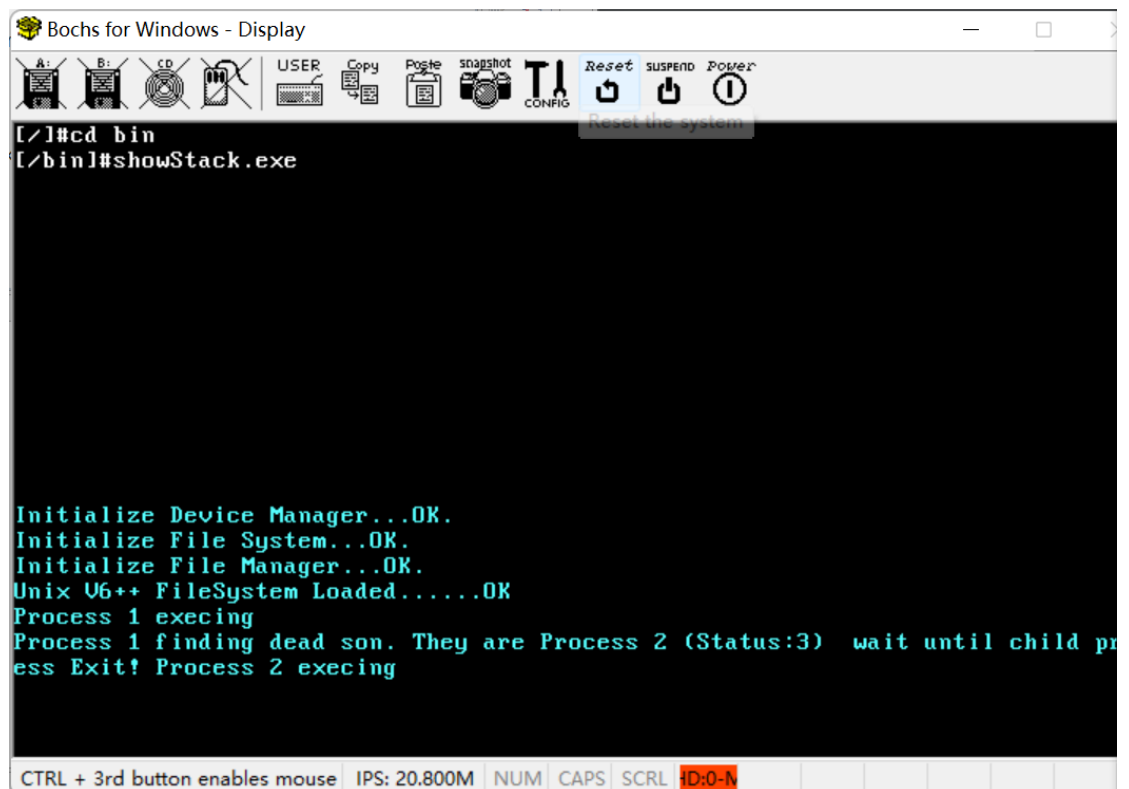
a. 设置调试开始的起点



- b. 开始调试:
此处停在入口处



输入命令开始执行和调试



- c. 调试运行:
设置断点:

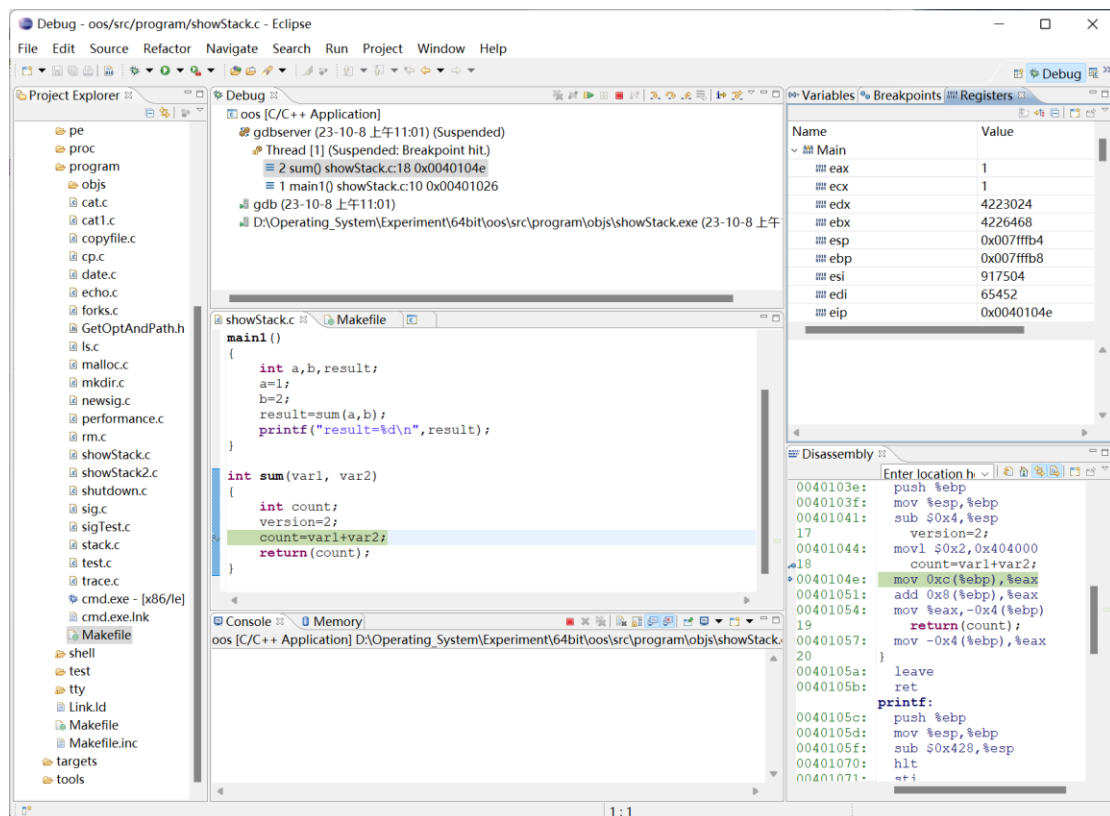
```

}

int sum(var1, var2)
{
    int count;
    version=2;
    count=var1+var2;
    return(count);
}

```

观察寄存器，内存，汇编代码等：



二． 复现并观察核心栈的变化

1. 存前一栈帧的 ebp，修改 ebp 指向当前栈帧，esp 上移
汇编代码：

```

main1:
00401000:  push %ebp
00401001:  mov %esp,%ebp
00401003:  sub $0x18,%esp

```

寄存器值：

eax	4198400
ecx	1
edx	4223024
ebx	4226468
esp	0x007fffc0
ebp	0x007fffd8
esi	917504
edi	65452
eip	0x00401006
eflags	[PF IF]

内存单元的观察：观察可知为小端存储，在 ebp 指向的 0x007fffd8 单元存储着上一栈帧基址 007FFFE0，0x007fffdc 存储着 main 的返回地址 00000008，并空出了局部变量和参数的值。

◆ 0x007ffc0	Address	0 - 3	4 - 7	8 - B	C - F
◆ 0x007ffdc	007FFFC0	00000000	00000000	00000000	00000000
	007FFFD0	00000000	00000000	E0FF7F00	08000000

2. 将 main 的局部变量送入栈中

汇编代码：

```

8          a=1;
00401006:  mov 0x8(%ebp),%eax
00401009:  mov %eax,(%esp)
0040100c:  call 0x402129 <ftoa+115>
9          b=2;
0040100d:  sbb %dl,(%ecx)
0040100f:  add %al,(%eax)
00401011:  mov %eax,-0x4(%ebp)

```

寄存器值：

eax	4198400
ecx	1
edx	4223024
ebx	4226468
esp	0x007ffc0
ebp	0x007fffd8
esi	917504
edi	65452
eip	0x00401014
eflags	[PF IF]
cs	27
ss	35
ds	35
es	35
fs	0

内存单元的观察：可以看到值 2 和 1 已经被放入栈中

◆ 0x007ffc0	Address	0 - 3	4 - 7	8 - B	C - F
◆ 0x007ffdc	007FFFC0	00000000	00000000	00000000	00000000
	007FFFD0	02000000	01000000	E0FF7F00	08000000
	007FFFE0	01000000	E8FF7F00	F2FF7F00	00000000
	007FFFF0	00007368	6F775374	61636B2E	65786500

3. 将参数放入栈中














汇编代码：

```

00401014:  mov -0x4(%ebp),%eax
00401017:  add 0x8(%ebp),%eax
0040101a:  dec %eax
0040101b:  mov %eax,-0x8(%ebp)
0040101e:  mov -0x8(%ebp),%eax

```

寄存器值：

Name	Value
▼  Main	
 eax	1
 ecx	1
 edx	4223024
 ebx	4226468
 esp	0x007fffb4
 ebp	0x007fffb8
 esi	917504
 edi	65452
 eip	0x00401044
 eflags	[PF IF]
 cs	27
 cc	25

内存单元的观察：参数已经被放入栈中

Address	0 - 3	4 - 7	8 - B	C - F
0x007fffc0	01000000	02000000	00000000	00000000
0x007fffd0	02000000	01000000	E0FF7F00	08000000
0x007fffe0	01000000	E8FF7F00	F2FF7F00	00000000
0x007ffff0	00007368	6F775374	61636B2E	65786500
0x00800000	01000000	02000000	00000000	00000000
0x00800010	02000000	01000000	E0FF7F00	08000000

4. 调用 sum 函数并返回

汇编代码：

```

sum:
0040103e:    ret
0040103f:    push %ebp
00401040:    mov %esp,%ebp
00401042:    sub $0x4,%esp
17      version=2;
00401044:    add $0xc7,%al
00401046:    inc %ebp
00401047:    cld
00401048:    add %al,(%eax)
0040104a:    add %al,(%eax)
0040104c:    movl $0x0,0x407004
18      count=var1+var2;
0040104e:    add $0x70,%al
00401050:    inc %eax
00401051:    add %al,(%eax)
00401053:    add %al,(%eax)
00401055:    add %al,%bh
19      return(count);
00401057:    inc %ebp
00401058:    cld
00401059:    cld
20      }
0040105a:    leave
0040105b:    ret

```

寄存器值：eax 为 3，esp 重新变回 0x007fffc0（和 pdf 文档中不同？）

Main	
eax	3
ecx	1
edx	4223024
ebx	4226468
esp	0x007fffc0
ebp	0x007fffd8
esi	917504
edi	65452
eip	0x00401029
eflags	PF IF 1

内存单元的观察：可以看到为 result 预留出来的值变为了 3

Address	0 - 3	4 - 7	8 - B	C - F
007FFFC0	01000000	02000000	00000000	03000000
007FFFD0	02000000	01000000	E0FF7F00	08000000
007FFFE0	01000000	E8FF7F00	F2FF7F00	00000000

5. 打印，结果如下

```
[/bin]#showStack.exe
result=3
```

三 . sum 代码分析和堆栈的绘制

1. 代码分析：

sum:

0040103e: push %ebp //将 main 函数栈帧的 ebp 存入当前栈

0040103f: mov %esp,%ebp //修改 ebp 指向当前栈帧

00401041: sub \$0x4, %esp //esp 上移 1 个字，空出 sum 函数局部变量 count 的位置

17 version=2;

00401044: movl \$0x2,0x404000 //将 2 送入全局变量 version 中

18 count=var1+var2;

0040104e: mov 0xc(%ebp), %eax //将参数 b 送入 eax 中

00401051: add 0x8(%ebp), %eax //将参数 a 和 eax 中的参数 b 相加送入 eax 中

00401054: mov %eax, -0x4(%ebp) //将相加的结果送入局部变量 result 中

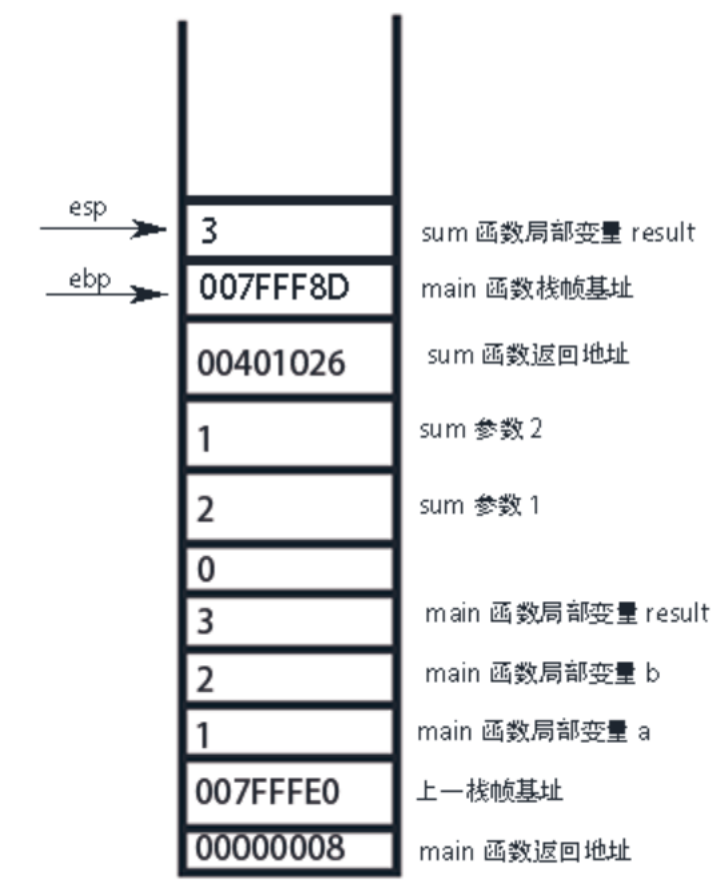
19 return(count);

00401057: mov -0x4(%ebp), %eax //将返回值送入 eax 中

0040105a: leave //撤销当前栈帧

0040105b: ret//返回调用函数前存储的指令位置

2. 完整的栈帧绘制：



四 . 问题回答

问题一：在 main1 的汇编代码中出现了“sub \$0x18,%esp”语句来预留出 6 个字，请查阅资料，解释这么做的原因是什么。

- ① 这是为 main 函数的局部变量，调用函数的参数分配的预留空间
- ② 注意到该分配的空间大于实际使用的空间，这可能是由于以下的原因：
 - a. 出于安全性的考虑，避免局部变量覆盖返回地址等重要数据的可能性
 - b. 出于可维护性的考虑，可以轻松添加更多的局部变量而无需调整栈分配的代码
 - c. 可能是出于内存单元对齐的要求，多分配了一个字