

同济大学计算机系

操作系统课程实验报告



学 号 2154312

姓 名 郑博远

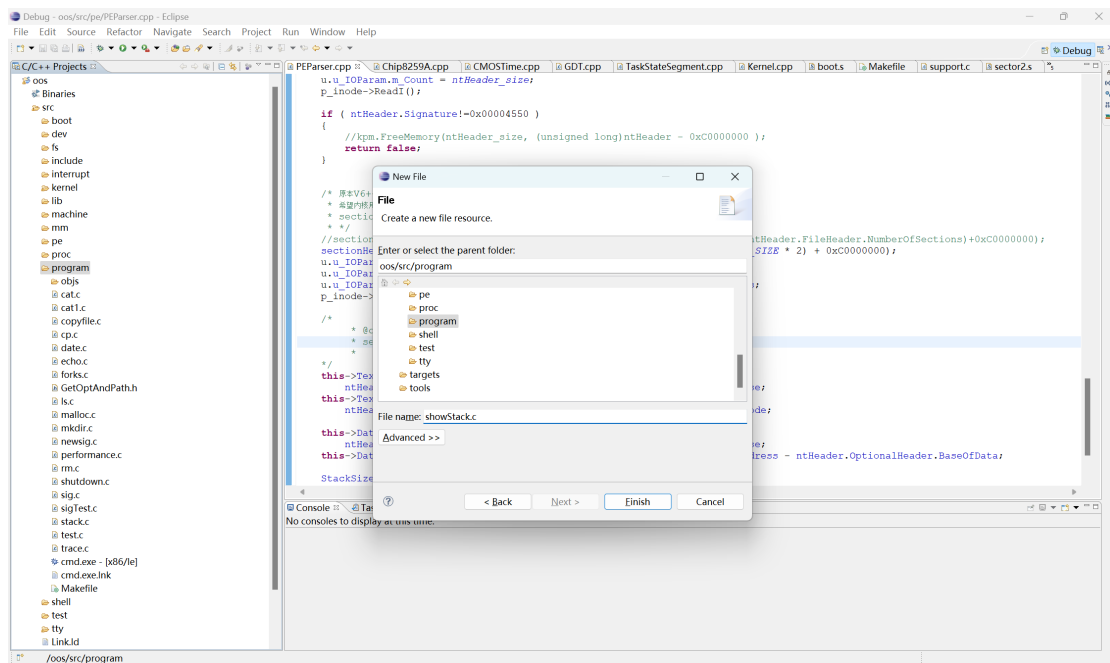
专 业 计算机科学与技术

授课老师 方 钰

P02: UNIX V6++进程的栈帧

一、完成实验 4.1~4.2，掌握在 UNIX V6++中添加自定义程序及编译、链接与运行的全过程，掌握 UNIX V6++中调试运行与观察结果的常规操作，截图说明上述过程。

1. 在 program 下新建 showStack.c 文件：



2. 编写 showStack.c 文件代码：

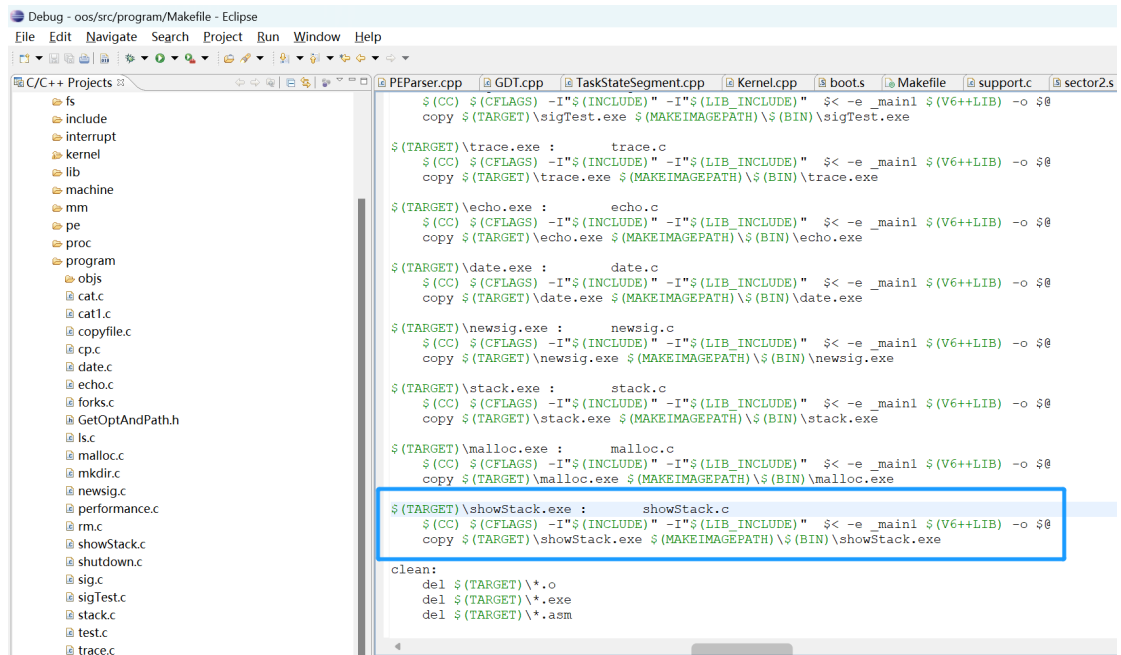
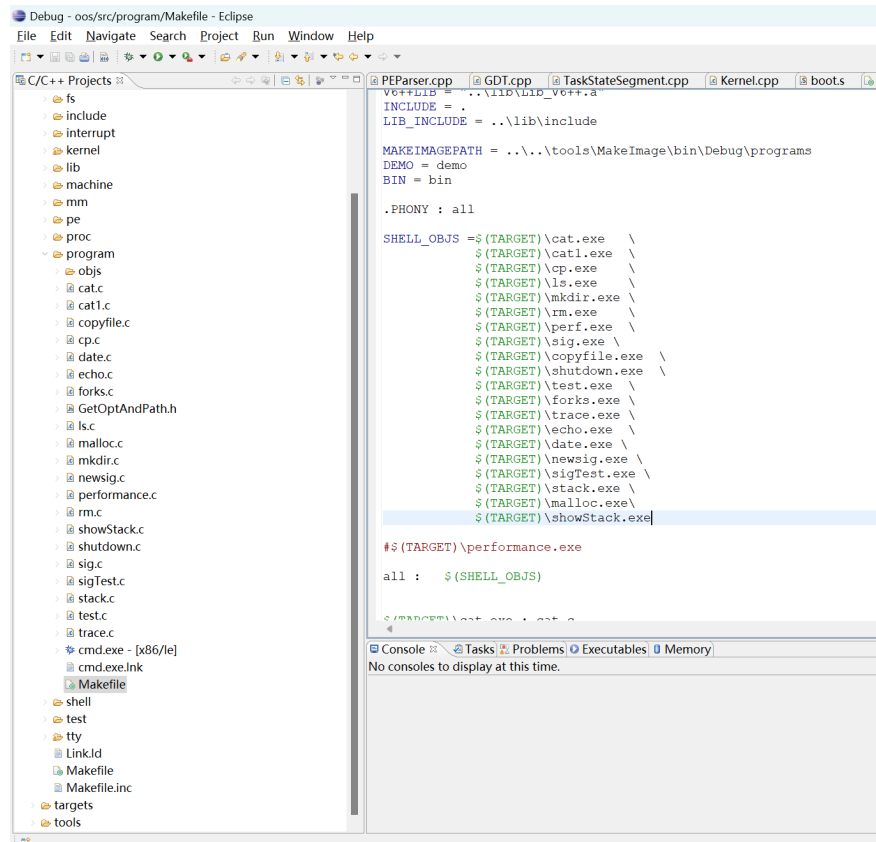
```
#include <stdio.h>

int version = 1;

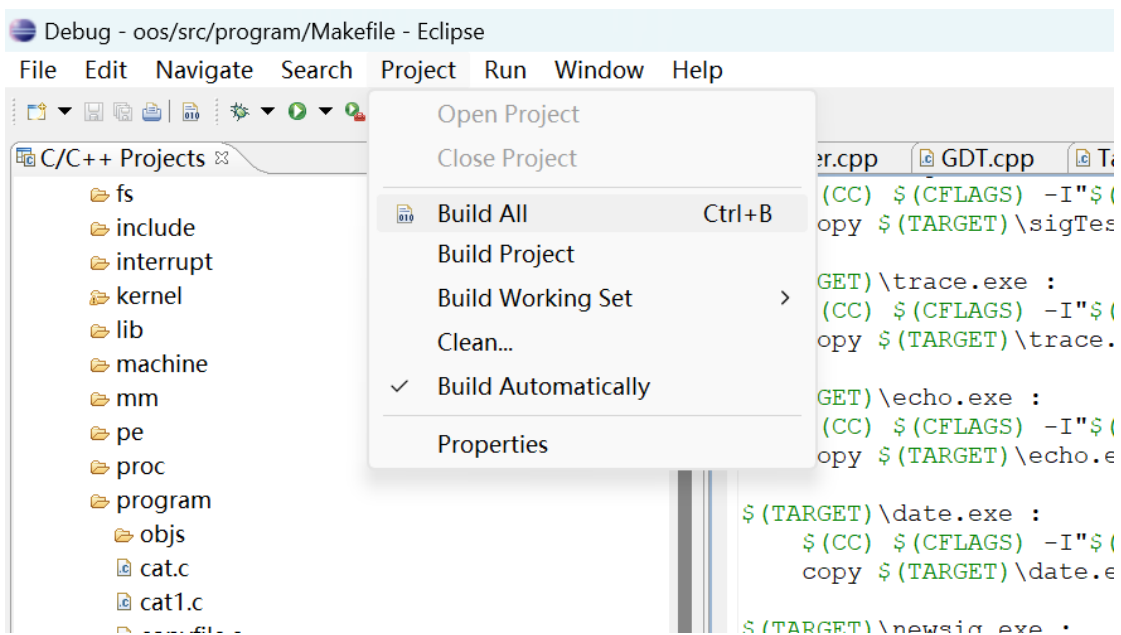
main1() {
    int a, b, result;
    a = 1;
    b = 2;
    result = sum(a, b);
    printf("result=%d\n", result);
}

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

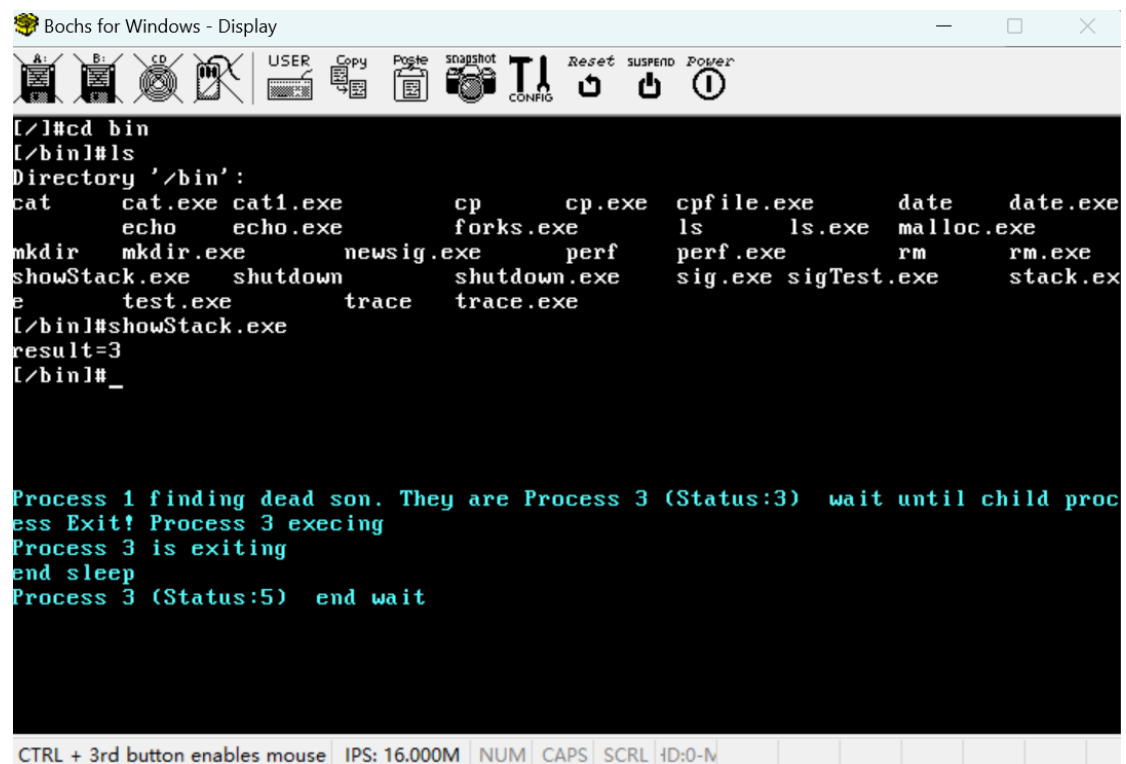
3. 修改 Makefile 文件用于编译:



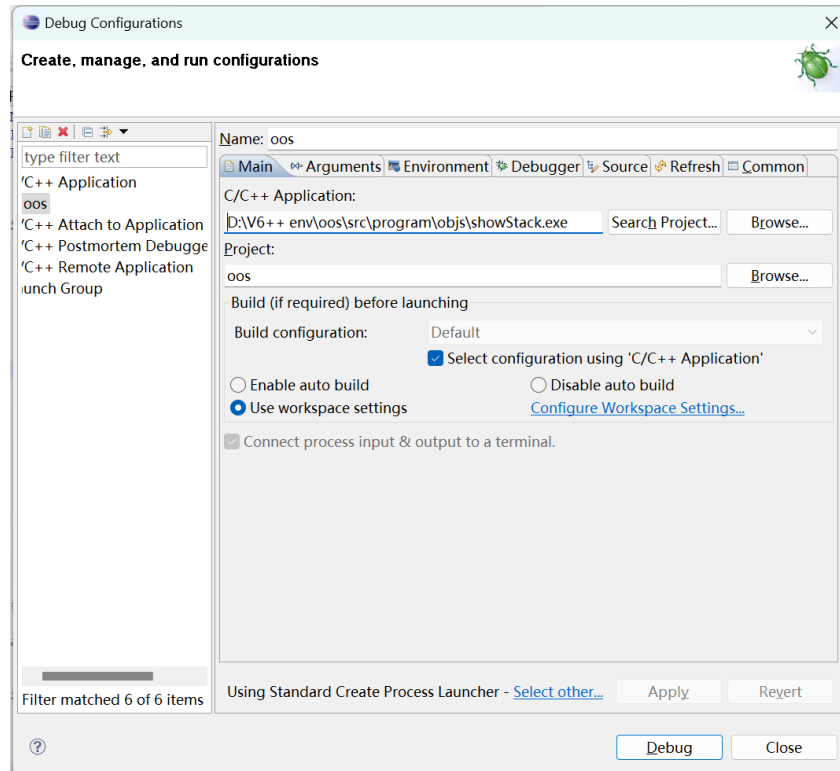
4. 重新编译 UNIX V6++源代码:



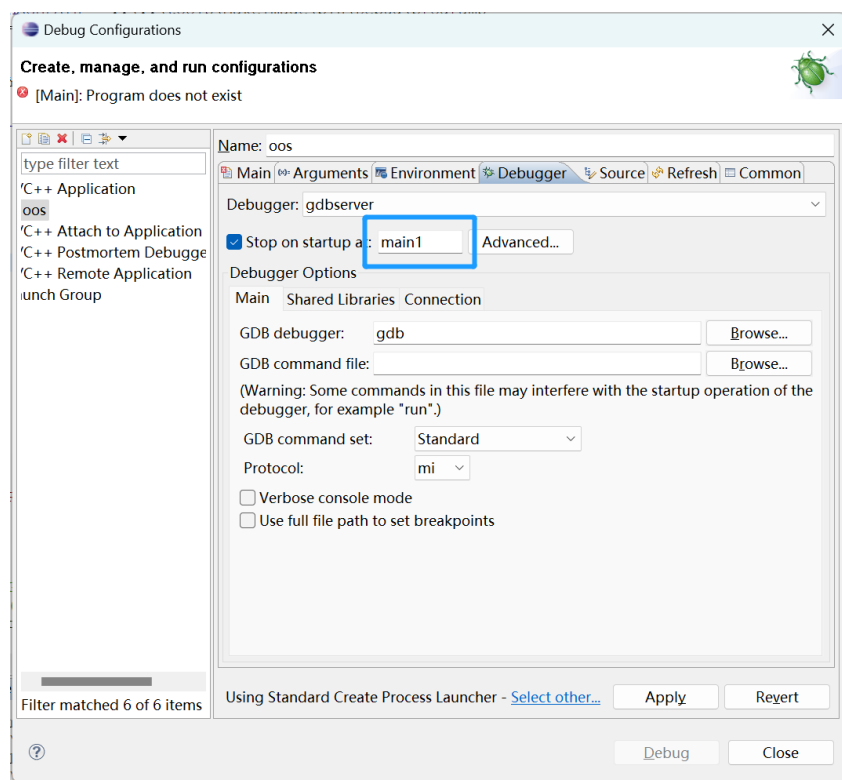
5. showStack.exe 程序运行结果:



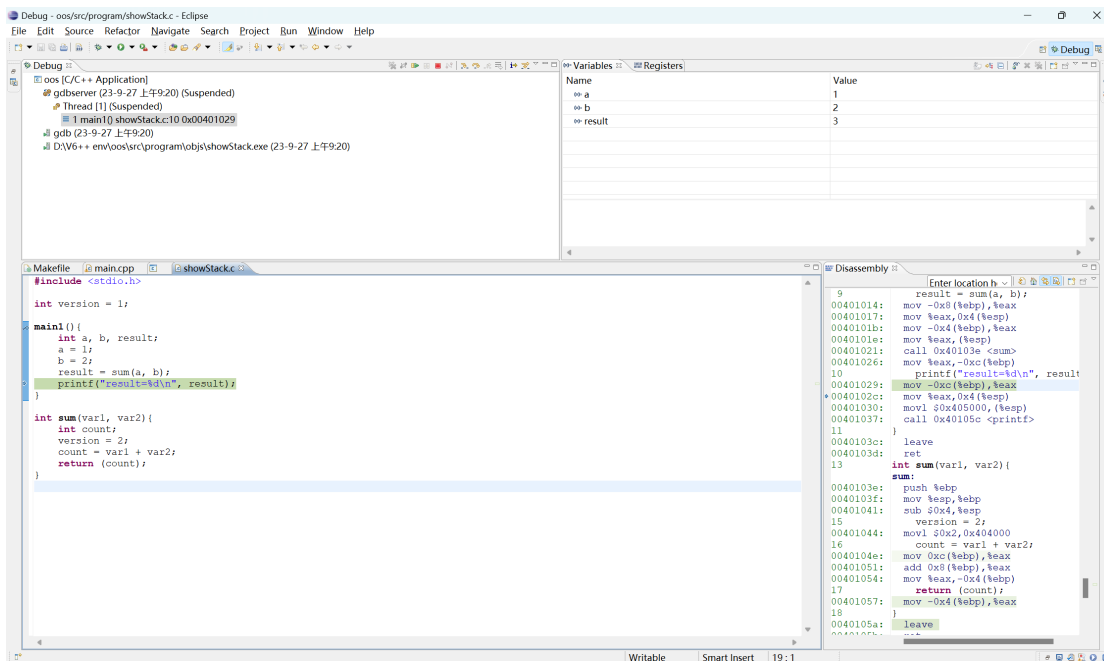
6. 修改调试对象:



7. 修改调试起点:

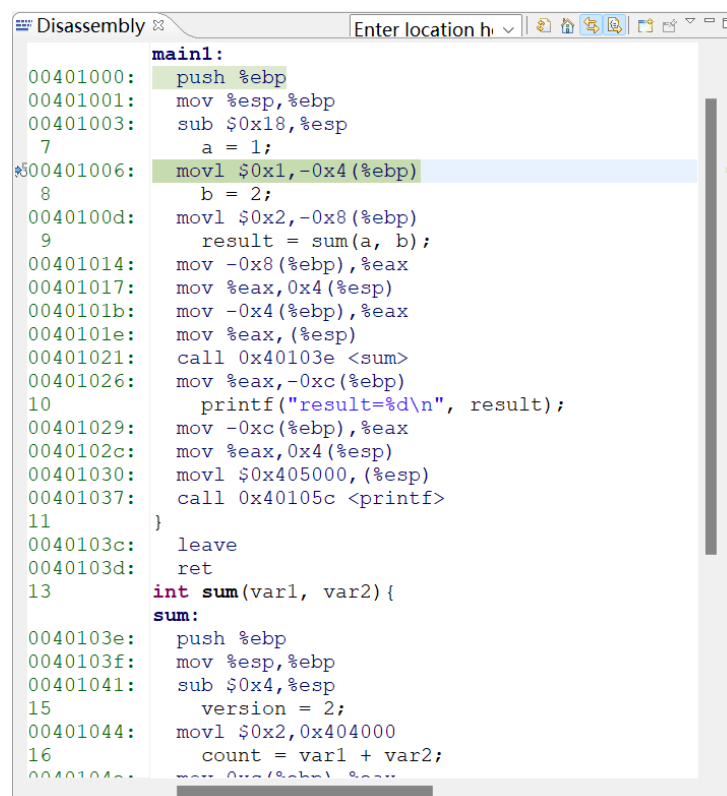


8. 调试过程中的结果查看：

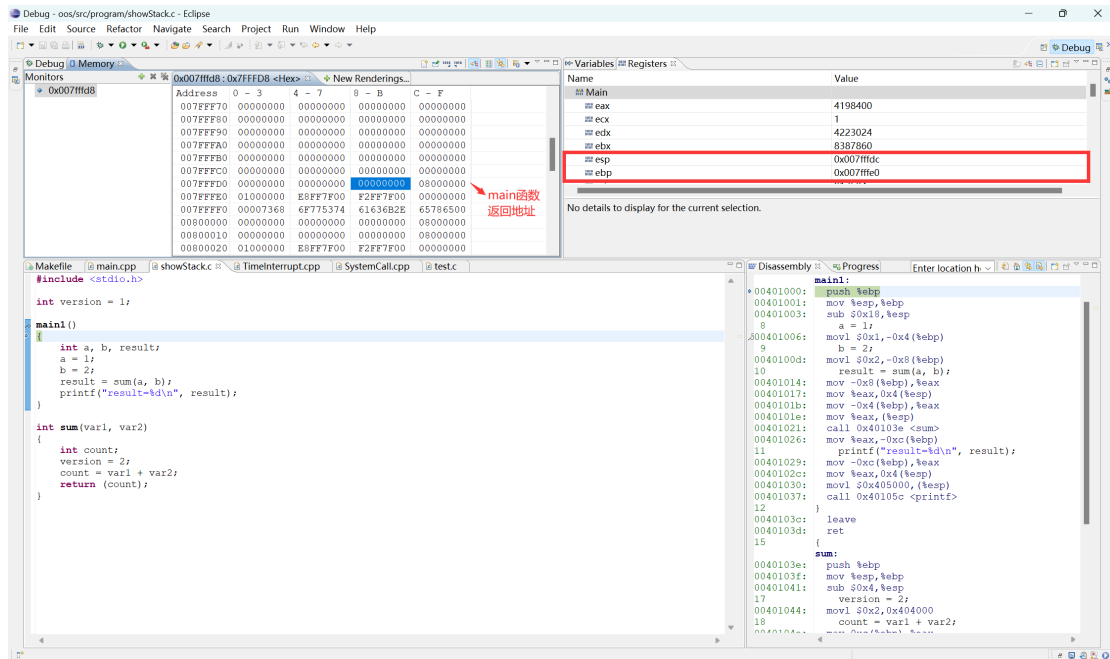


二、复现实验 4.3 中 main 函数核心栈的变化，通过在 Memory 窗口中观察地址单元的值验证核心栈的变化。

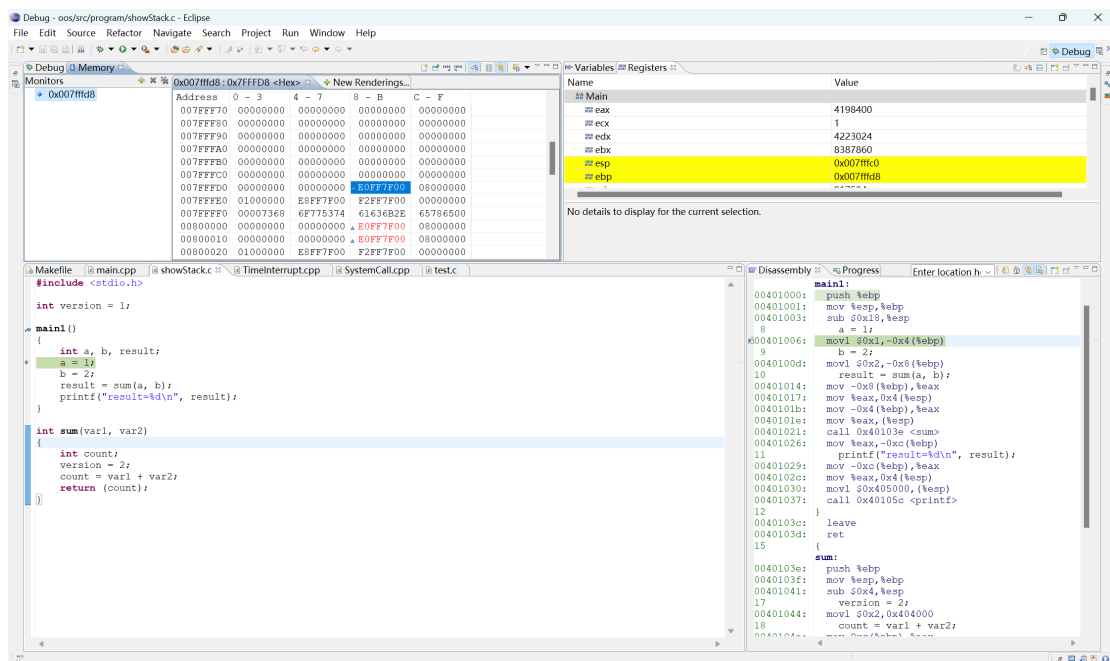
1. 主函数 main1 的汇编代码：



2. 下面通过 memory 窗口观察核心栈变化。在进入 main1 函数前，ebp 还未入栈，ebp 寄存器显示原来栈底值 0x007ffe0:



3. 进入 main1 函数后，ebp 压栈，esp 为局部变量以及 sum 函数的参数预留出足够空间。栈空间变化如下：



esp→	0x007fffc0		此处为sum的参数var1
	0x007fffc4		此处为sum的参数var2
	0x007fffc8		
	0x007fffcc		main 的局部变量 result
	0x007fffd0		main 的局部变量 b, 0040100d 语句后变为 2
	0x007fffd4		main 的局部变量 a, 00401006 语句后变为 1
ebp→	0x007fffd8	007FFFE0	上一栈帧基址, 00401000 语句压栈; 00401001 语句将 ebp 修改到此处
	0x007fffdc	00000008	main 的返回地址

4. 执行了 main 函数局部变量 a、b 的赋值语句后, 0x07fffd4 与 0x07fffd0 两个单元（分别对应变量 a 与 b）被赋值, 栈内变化如下:

The screenshot shows a debugger interface with the following components:

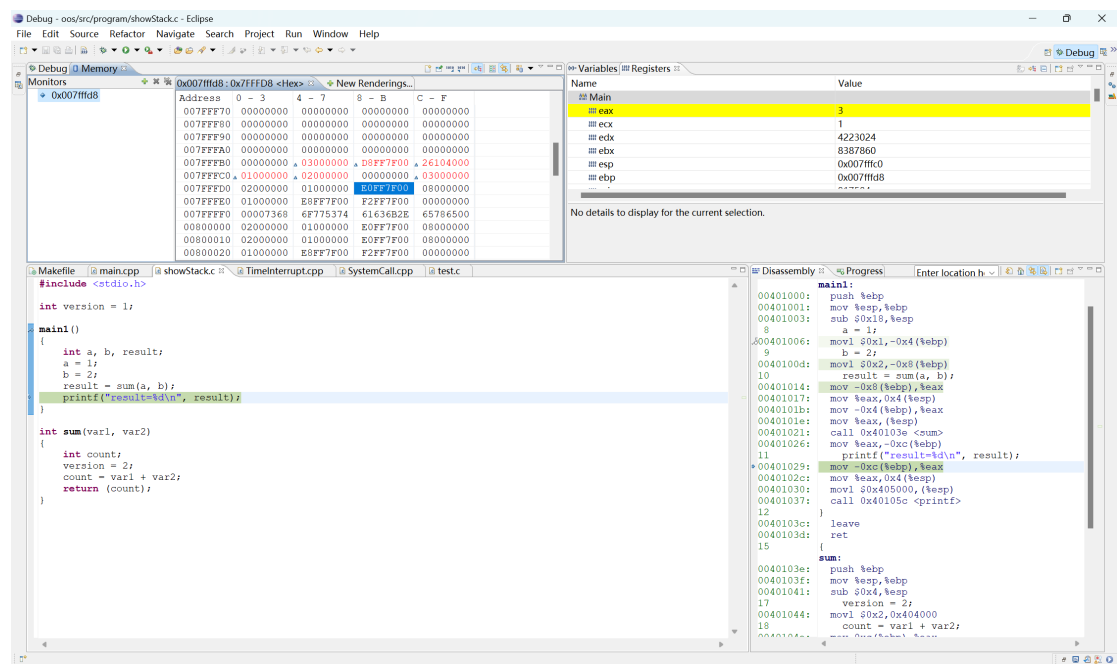
- Monitors:** Displays memory at address 0x007fffd8. The table shows addresses from 007FFF70 to 00800020 and their corresponding values in hex, decimal, and ASCII.
- Variables/Registers:** Shows the state of registers. Notably, `esp` is at 0x007ffdc and `ebp` is at 0x007fffd8.
- Disassembly:** Shows the assembly code for the `main` function. Key instructions include:
 - `00401000: push %ebp`
 - `00401001: mov %esp, %ebp`
 - `00401003: sub $0x18, %esp`
 - `00401006: movl $0x1, -0x4(%ebp)` (Assigns 1 to variable a)
 - `00401008: movl $0x2, -0x8(%ebp)` (Assigns 2 to variable b)
 - `0040100d: movl -0x8(%ebp), %eax` (Loads value of a into eax)
 - `00401017: mov %eax, 0x4(%esp)` (Pushes a onto the stack)
 - `0040101b: mov -0x4(%ebp), %eax` (Loads value of b into eax)
 - `0040101e: mov %eax, (%esp)` (Pushes b onto the stack)
 - `00401021: call 0x40103e <sum>` (Calls the sum function)
 - `00401026: mov %eax, -0xc(%ebp)` (Stores the result of sum in the stack)
 - `00401029: printf("result=%d\n", result);`
 - `0040102c: mov %eax, 0x4(%esp)` (Pushes return address)
 - `00401030: movl $0x405000, (%esp)` (Pushes 405000)
 - `00401037: call 0x40105c <printf>`
 - `0040103c: leave`
 - `0040103d: ret`

esp→	0x007fffc0		此处为sum的参数var1
	0x007fffc4		此处为sum的参数var2
ebp→	0x007fffc8		main 的局部变量 result
	0x007fffcc		
	0x007fffd0	2	main 的局部变量 b, 0040100d 语句后变为 2
	0x007fffd4	1	main 的局部变量 a, 00401006 语句后变为 1
	0x007fffd8	007FFFE0	上一栈帧基址, 00401000 语句压栈; 00401001 语句将 ebp 修改到此处
	0x007fffdc	00000008	main 的返回地址

5. 执行了 sum()函数的调用语句后, 局部变量对应的地址 0x007fffc4 与 0x007fffc0 被赋值, ebp 与 esp 指向 sum 函数栈帧的对应位置。

esp:	0x007fffb4		此处为sum的局部变量count
ebp:	0x007fffb8	007FFFD8	main函数栈帧基址,0040103e语句压栈; 0040103f语句将ebp修改到此处
	0x007fffc0	1	00401003 语句将 esp 修改到此处, 空出 main 的局部变量位置和 sum 的参数位置; 此处为 sum 的参数 var1, 0040101e 语句执行完后修改为 1
	0x007fffc4	2	此处为 sum 的参数 var2, 00401017 语句执行完后修改为 2
	0x007fffc8		main 的局部变量 result
	0x007fffcc		
	0x007fffd0	2	main 的局部变量 b, 0040100d 语句后变为 2
	0x007fffd4	1	main 的局部变量 a, 00401006 语句后变为 1
	0x007fffd8	007FFFE0	上一栈帧基址, 00401000 语句压栈; 00401001 语句将 ebp 修改到此处
	0x007fffdc	00000008	main 的返回地址

6. 执行了 sum()函数返回后, sum 的栈帧被撤销。ebp、esp 寄存器恢复 main 函数栈帧的原先位置。main 函数的局部变量 result 被 eax 寄存器赋值。



ebp→	0x007fffc0	1	00401003 语句将 esp 修改到此处, 空出 main 的局部变量位置和 sum 的参数位置; 此处为 sum 的参数 var1, 0040101e 语句执行完后修改为 1
	0x007fffc4	2	此处为 sum 的参数 var2, 00401017 语句执行完后修改为 2
esp→	0x007fffc8		main 的局部变量 result, 00401026 语句后变为 3 main 的局部变量 b, 0040100d 语句后变为 2 main 的局部变量 a, 00401006 语句后变为 1
	0x007ffcc	3	
	0x007fffd0	2	
	0x007fffd4	1	
	0x007fffd8	007FFFE0	上一栈帧基址, 00401000 语句压栈; 00401001 语句将 ebp 修改到此处
	0x007fffdc	00000008	main 的返回地址

三、完整分析 sum 的汇编代码，添加注释，并仿照图 16，绘制完整的堆栈。

1. 对 sum 的汇编代码进行分析并添加注释：

```
sum:
0040103e:  push %ebp                # 前一栈帧的 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，空出局部变量 count 的位置
17      version = 2;
00401044:  movl $0x2,0x404000        # 把 2 送入地址 0x404000，即全局变量 version
18      count = var1 + var2;
0040104e:  mov 0xc(%ebp),%eax        # 把%ebp+12（即参数 var2）送入寄存器 eax 中
00401051:  add 0x8(%ebp),%eax        # 把%ebp+8（即参数 var1）与寄存器 eax 中的值相加
00401054:  mov %eax,-0x4(%ebp)        # 把寄存器 eax 中的值移入%ebp-4，即 sum 的局部变量 count
19      return (count);
00401057:  mov -0x4(%ebp),%eax        # 将%ebp-4 中的值（即局部变量 count）移入 eax 以返回 main
20      }
0040105a:  leave                    # 将%ebp 赋给%esp，并弹出 main 函数的旧%ebp 值
0040105b:  ret                      # 返回 main 函数
```

2. 在图 16 的基础上绘制包含 sum 函数栈帧在内的完整堆栈：

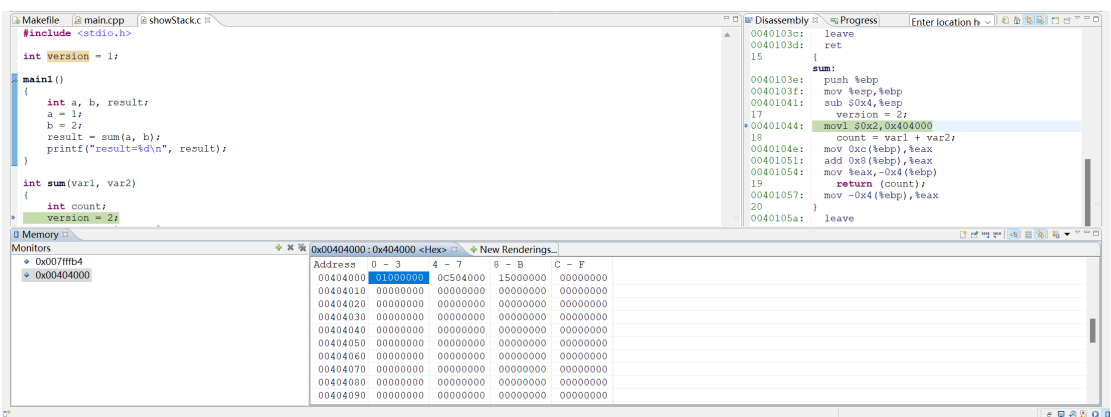
esp:	0x007fffb4	3	此处为sum的局部变量count，00401054语句执行完后修改为3
ebp:	0x007fffb8	007FFFD8	main函数栈帧基址,0040103e语句压栈；0040103f语句将ebp修改到此处
esp: (call 之后)	0x007fffb4	00401026	sum 的返回地址，00401021 (call) 语句压栈，并将 esp 修改到此处
esp: (call 之前)	0x007fffc0	1	00401003 语句将 esp 修改到此处，空出 main 的局部变量位置和 sum 的参数位置；此处为 sum 的参数 var1，0040101e 语句执行完后修改为 1
	0x007fffc4	2	此处为 sum 的参数 var2，00401017 语句执行完后修改为 2
	0x007fffc8	0	
	0x007fffcc	3	main 的局部变量 result，00401026 语句后变为 3
	0x007fffd0	2	main 的局部变量 b，0040100d 语句后变为 2
	0x007fffd4	1	main 的局部变量 a，00401006 语句后变为 1
ebp:	0x007fffd8	007FFFE0	上一栈帧基址,00401000 语句压栈;00401001 语句将 ebp 修改到此处
	0x007fffdc	00000008	main 的返回地址

四、选择以下两个问题其一回答：

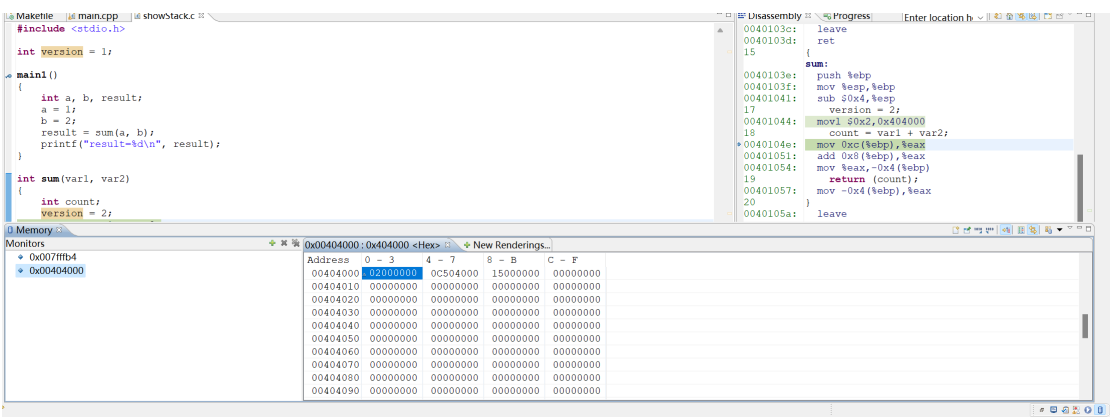
问题二：在 sum 的汇编代码中出现了 0x404000 这个地址，请先通过分析代码回答这个地址是什么，然后尝试通过调试验证你的答案。

答：0x404000 对应的是全局变量 version 的地址，其处在该进程的数据段当中。由于数据段中仅有 version 一个全局变量，因此其地址就是数据段的首地址。

1. 执行 00401044 处的 mov 语句前，该地址（version）处对应的值为 1。



2. 执行 00401044 处的 mov 语句后，该地址（version）处对应的值被赋值为 2。



3. 通过“objdump”查看数据段的首地址确实为 0x00404000。

```
2884 Disassembly of section .data:
2885
2886 00404000 <__data_start__>:
2887 404000: 01 00 add %eax, (%eax)
2888 ...
```