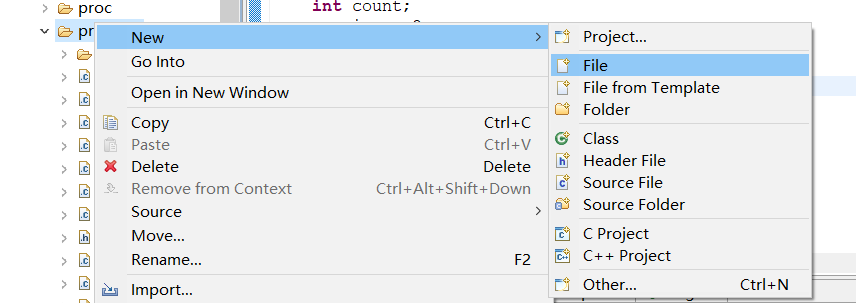
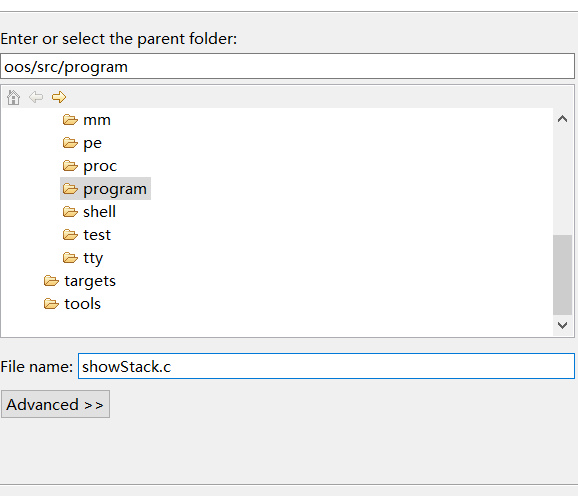
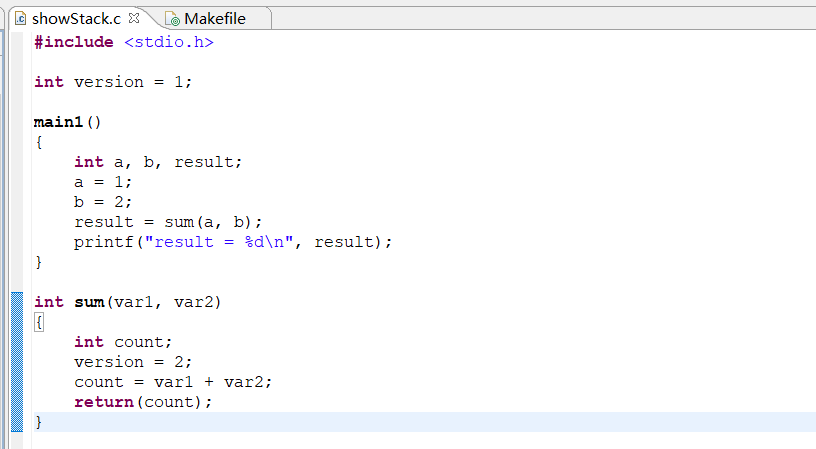
**P02：UNIX V6++进程的栈帧**

**2152118 史君宝**

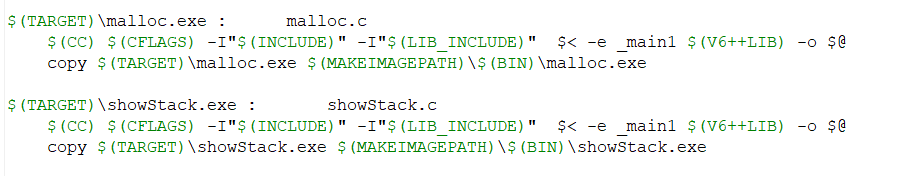
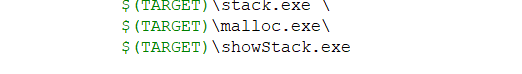
1. **UNIX V6++中编译链接运行一个 C 语言程序**
2. 在 program 文件加入一个新的 c 语言文件



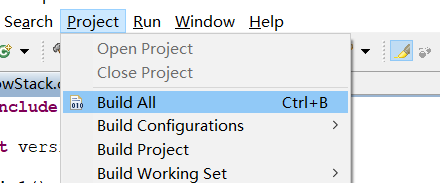


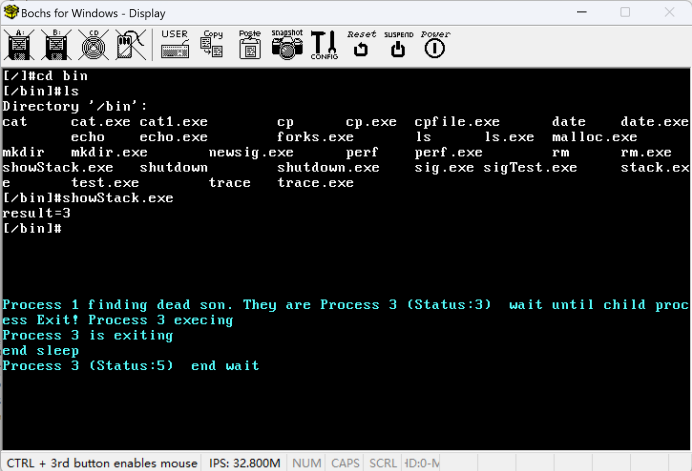


1. 修改编译需要使用的 Makefile 文件

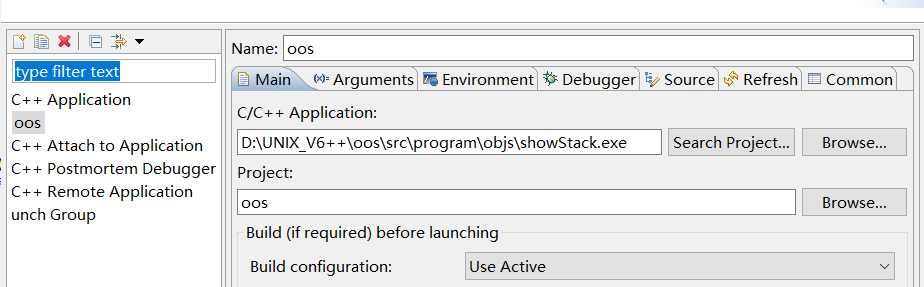


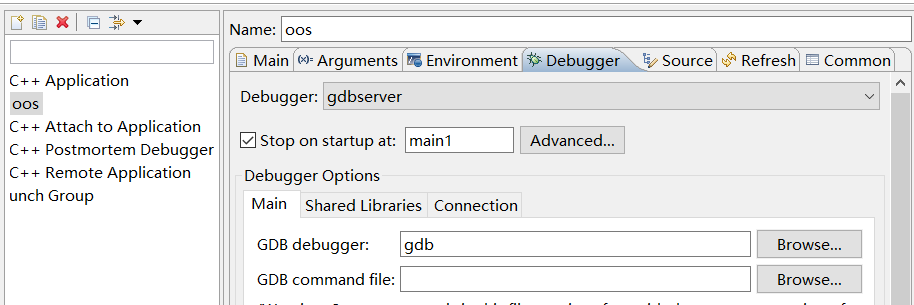
1. 重新编译运行 UNIX V6++代码



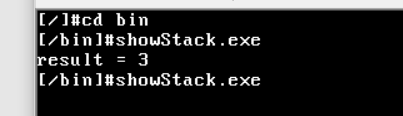


1. 设置调试对象和调试起点



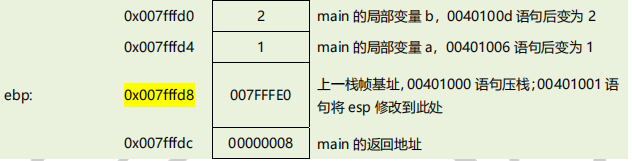


1. 设置并开始调试



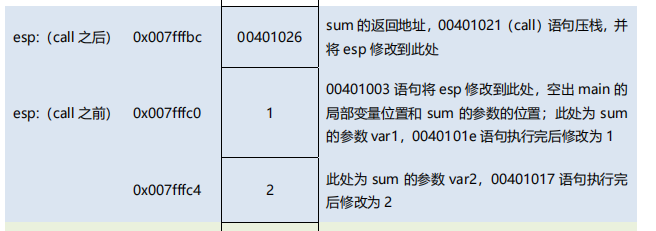
1. **复现main1 函数核心栈的变化**

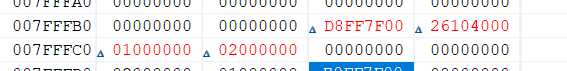
1. 未进入sum中的核心栈





1. 进入sum函数的核心栈





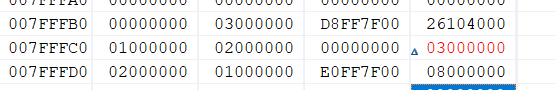
1. Sum函数中的count = var1 + var2



其中0X007FFFD8 是上一栈帧的地址，即对应EBP的上一地址

而0X00000003 是count的值

1. 返回main1函数



返回之后会将栈帧地址以上的第三个参数result赋值为0X00000003

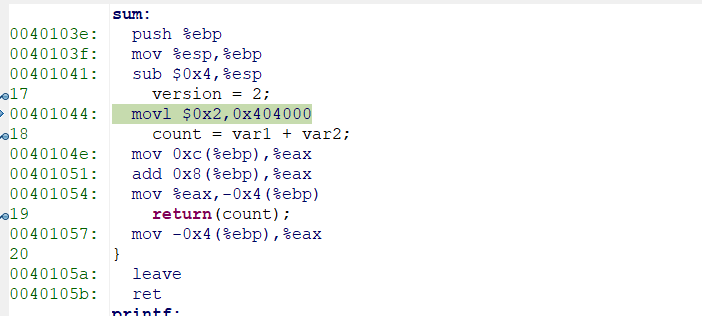
综上核心栈为：

地址 数值 说明

|  |  |  |
| --- | --- | --- |
| 0X007fffb4 | 3 | 局部变量count |
| 0X007fffb8 | 0X007fffd8 | 上一栈帧main1的基址 |
| 0X007fffbc | 00401026 | sum函数的返回地址 |
| 0X007fffc0 | 1 | 调用sum的参数var1 |
| 0X007fffc4 | 2 | 调用sum的参数var2 |
| 0X007fffc8 | 0 |  |
| 0X007fffcc | 3 | 局部变量result |
| 0X007fffd0 | 2 | 局部变量b |
| 0X007fffd4 | 1 | 局部变量a |
| 0X007fffd8 | 0X007fffe0 | 上一栈帧基址 |
| 0X007fffdc | 00000008 | main1的返回地址 |

**三、分析 sum 的汇编代码，完善栈帧**

（1）sum函数的汇编代码



我们依次分析：

0040103e: push %ebp :

将上一栈帧基址入栈，将main1栈帧的基址存放到%esp的位置。

0040103f: mov %esp, %ebp :

将esp寄存器值赋值给ebp，这样就将ebp指向新栈帧的基址。

00401041: sub &0X4, %esp :

将esp指针上移，为sum函数中的局部变量申请空间。

00401044: mov1 &0X2, 0X404000 :

将0X2值赋值给地址为0X404000位置的变量，即为version变量。

0040104e: mov 0Xc(%ebp), %eax :

将ebp指针下面三个位置的变量，即为变量var1移到寄存器eax。

00401051: add 0X8(%ebp), %eax :

将ebp指针下面两个位置的变量，即变量var2与寄存器eax相加。

00401054: mov %eax, -0X4(%ebp) :

将寄存器eax赋值给ebp指针上面一个位置的变量，即变量count。

00401057: mov -0X4(%ebp), %eax :

将ebp指针上面一个位置的变量，即变量count赋值给寄存器eax。

0040105a: leave :

0040105b: ret :

离开并退出函数，返回函数地址。

（2）完善的栈帧

下面是完整的栈帧，在原图上直接进行修改。

其中上面的第一格应为sum函数中的局部变量count，刚开始申请的时候值为0，在经过count = var1 + var2 计算之后，会变为3。

第二格应为上一栈帧的基址，也就是main1函数的栈帧基址，即为0X007fffd8



地址 数值 说明

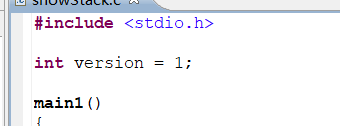
|  |  |  |
| --- | --- | --- |
| 0X007fffb4 | 3 | 局部变量count |
| 0X007fffb8 | 0X007fffd8 | 上一栈帧main1的基址 |
| 0X007fffbc | 00401026 | sum函数的返回地址 |
| 0X007fffc0 | 1 | 调用sum的参数var1 |
| 0X007fffc4 | 2 | 调用sum的参数var2 |
| 0X007fffc8 | 0 |  |
| 0X007fffcc | 3 | 局部变量result |
| 0X007fffd0 | 2 | 局部变量b |
| 0X007fffd4 | 1 | 局部变量a |
| 0X007fffd8 | 0X007fffe0 | 上一栈帧基址 |
| 0X007fffdc | 00000008 | main1的返回地址 |

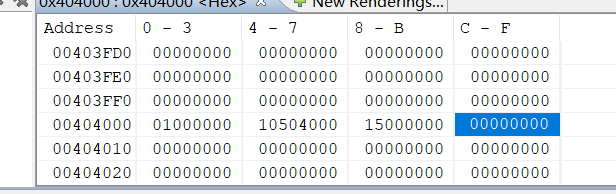
**四、回答问题。**

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。

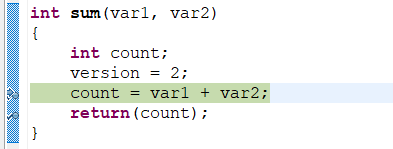
解答：我们通过分析代码可以知道，这个0x404000应该是在刚开始申请的全局变量version的地址，我们可以通过Memory找到具体的位置，观察其在进入sum函数前后的变化，来验证。

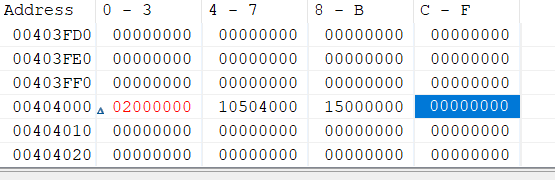
进入sum函数前：



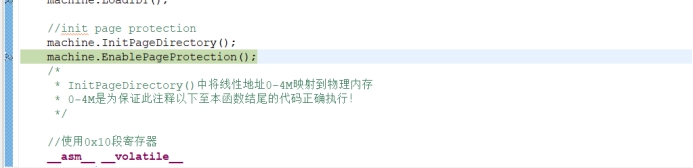


进入sum函数后：

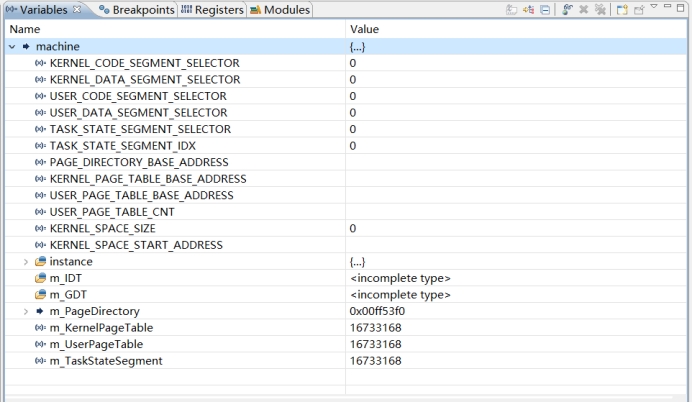




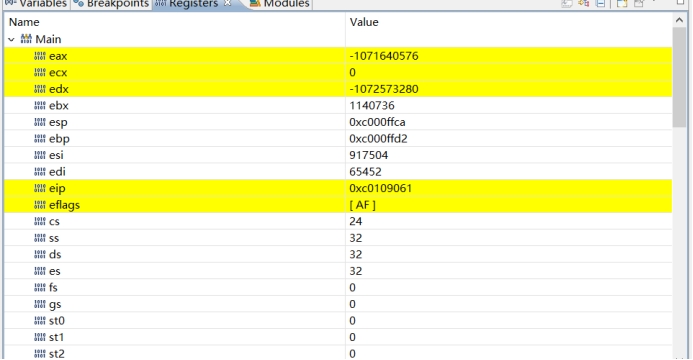
（2）设置断点二：



变量值：



寄存器值：



1. **UNIX V6++源代码目录结构**

boot/：应该是包含引导加载程序的源代码和配置文件。

dev/：应该是包含设备驱动程序的源代码，用于与硬件设备进行交互。

fs/：应该是包含文件系统相关的源代码，包括文件系统的实现和文件操作。

include/：包含系统的头文件，就像我们在C++ 中使用的include <iostream> 一样定义了各种常量、数据结构和函数原型等。

kernel/：包含内核的核心代码，整个操作系统最重要的部分，负责处理系统调用、进程管理、内存管理等核心功能。

lib/：包含通用的库函数和工具函数的源代码，就像我们在使用eclipse中导入的库一样。

mm/：应该与存储相关，包含内存管理相关的源代码，比如内存分配、虚拟内存管理等等。

sys/：应该包含有关系统调用的相关信息，比如包含系统调用相关的源代码，定义了系统调用的接口和实现。

tools/：包含一些辅助工具和实用程序的源代码，用于构建和调试系统。