同济大学

TONGJI UNIVERSITY

操作系统实验报告

学　　院	电子与信息工程学院
专　　业	计算机科学与技术专业
学生姓名	吕博文
学　　号	2151769
指导教师	方钰
日　　期	2023 年 10 月 11 日

目 录

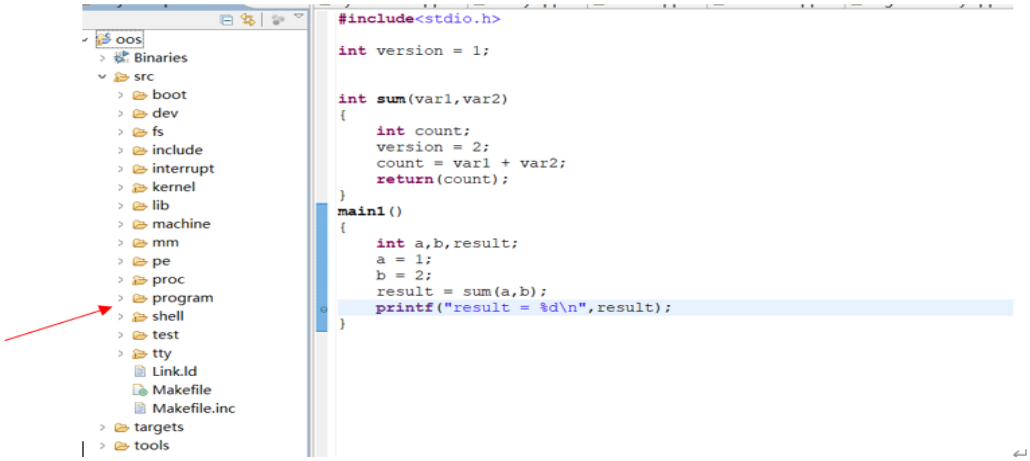
1 Unix V6++中调试运行自定义程序 …… 1
1.1 Unix V6++中编译链接自定义程序 …… 1
1.2 Unix V6++中调试自定义程序 …… 2
2 观察 main1 函数堆栈变化 …… 3
2.1 实验过程 …… 3
3 sum 函数汇编代码及其堆栈分析 …… 4
3.1 汇编代码分析 …… 4
3.2 核心栈绘制 …… 4
4 问题探究 …… 6
4.1 问题一 …… 6
4.2 问题二 …… 6

装 订 线

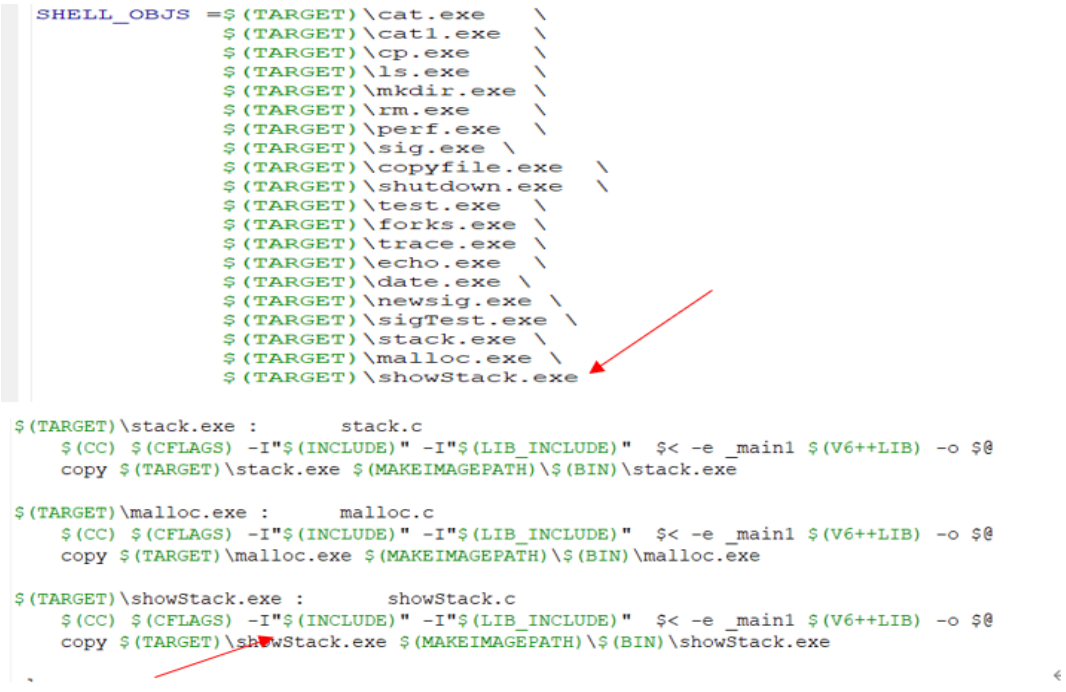
1 Unix V6++中调试运行自定义程序

1.1 Unix V6++中编译链接自定义程序

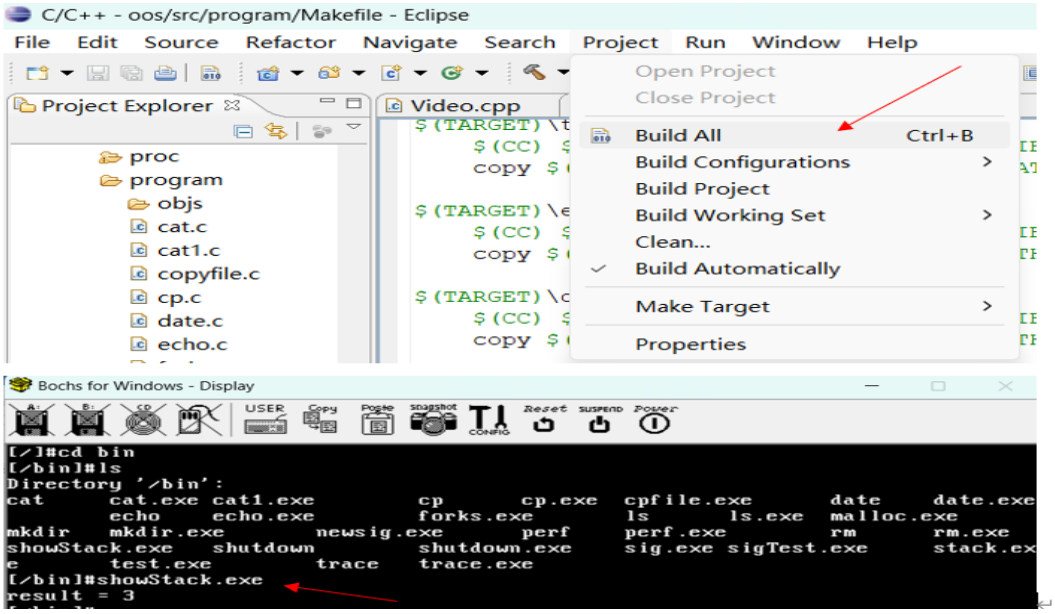
（1）自定义程序的添加位置均为/oos/src/program 目录下，通过 new 一个新文件，我们可以再 ecilpse 中完成自定义程序的编写，在本实验中就以模板为例，编写了一个简单的主函数和 sum 函数方便观察函数调用过程中栈帧的变化。



（2）需要注意的是，不同于常见的编译器可以直接点击运行，我们每增加一个自定义文件，都需要在/oos/src/program 目录下的 makefile 批处理文件中做出改动使得再重新编译运行时涵盖掉新增的文件，具体处理如下：

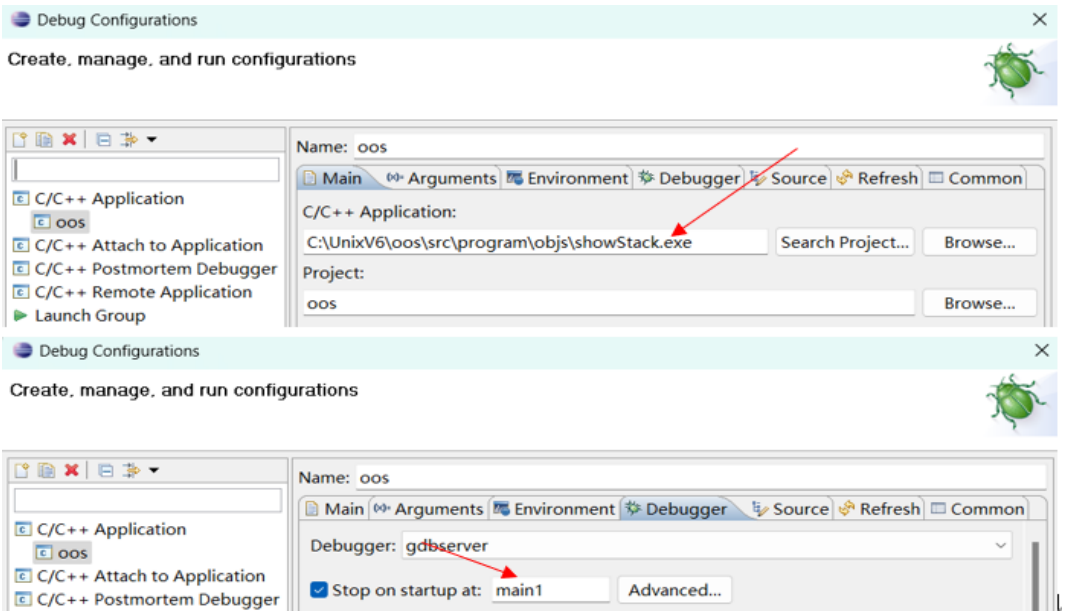


（3）完成批处理文件改动后，重新编译整个项目文件，之后在“运行”模式下运行 Unix V6++ 命令控制行，在 bin 文件夹中就可找到新增的可执行文件，输入对应命令即可得到执行结果。

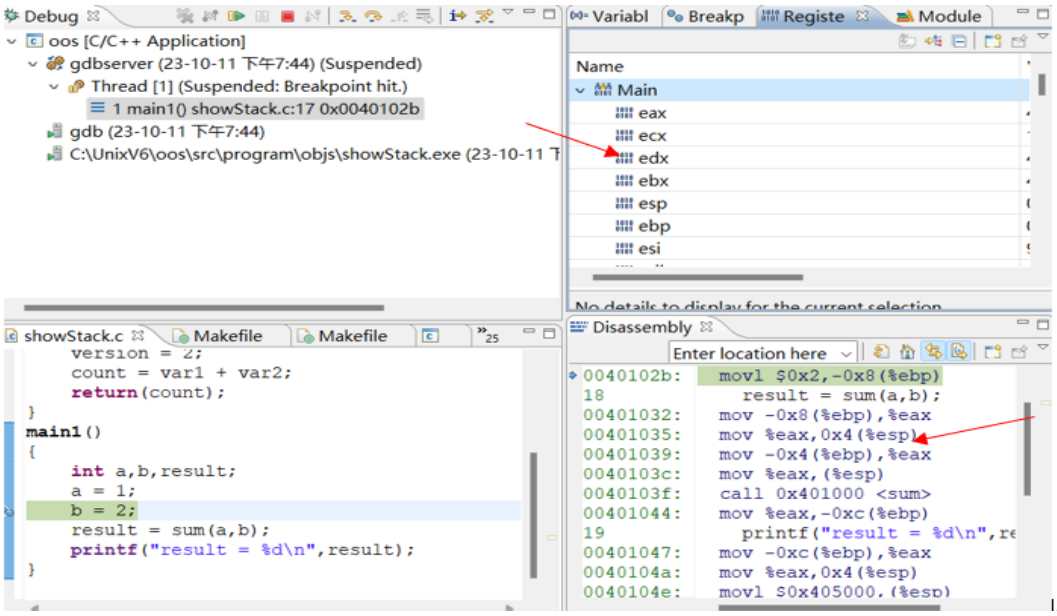


1.2 Unix V6++中调试自定义程序

（1）调试自定义程序时，需要首先将调试对象从 Kerner.exe 改为所选择的可执行文件，并将调试起始点改为 main1。



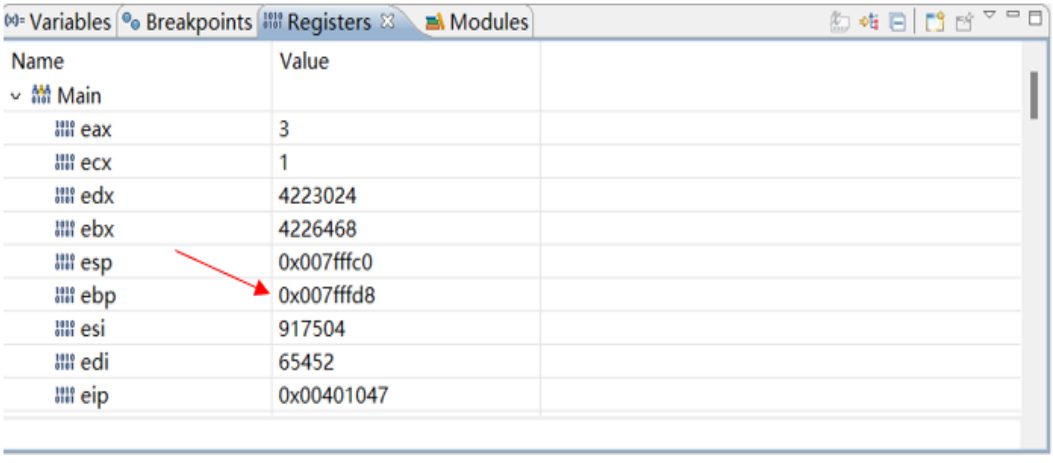
（2）之后改为“调试”模式，在对应文件中添加合适的断点，调试运行，进入调试模式（注意可能需要多次点击 resume 按钮使程序正确停在断点上），可以观察寄存器，内存，汇编代码等等。



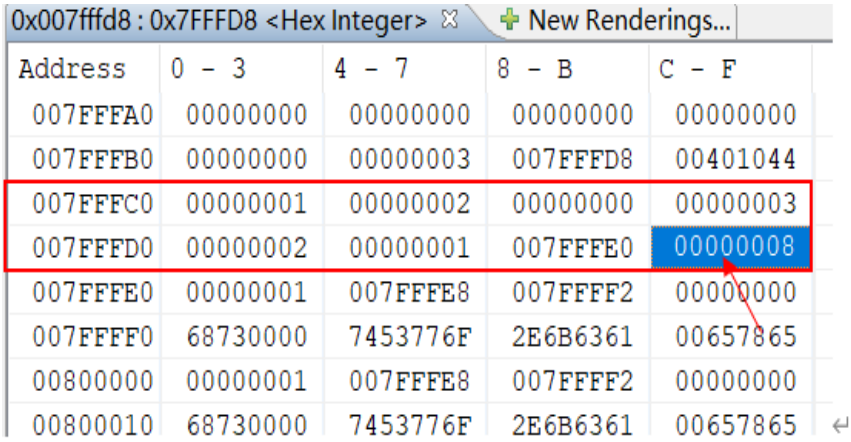
2 观察 main1 函数堆栈变化

2.1 实验过程

（1）我们将断点设置在 sum 函数完成后的下一条语句，进入调试状态运行后可以发现 ebp 的值为 0x007fffd8，通过 ebp 的值，我们在 Mermory 空间中可以观察堆栈中的值。



（2）分析堆栈的值，此时自地址 0x007ffdc 开始向上看，分别为 main 函数返回地址 00000008，上一栈帧地址 007FFFE0，main 局部变量 a 的值 1，main 局部变量 b 的值 2，main 局部变量 result 的值 3，空闲值，sum 函数的参数 var2，sum 的参数 var1，sum 函数的返回地址 00401044，与汇编代码核查后核心栈的解释无误。



0x007fffd8 : 0x7FFFD8 <Hex Integer> | New Renderings...

Address	0 - 3	4 - 7	8 - B	C - F
007FFFA0	00000000	00000000	00000000	00000000
007FFFB0	00000000	00000003	007FFFD8	00401044
007FFFC0	00000001	00000002	00000000	00000003
007FFFD0	00000002	00000001	007FFFE0	00000008
007FFFE0	00000001	007FFFE8	007FFFF2	00000000
007FFFF0	68730000	7453776F	2E6B6361	00657865
00800000	00000001	007FFFE8	007FFFF2	00000000
00800010	68730000	7453776F	2E6B6361	00657865

3 sum 函数汇编代码及其堆栈分析

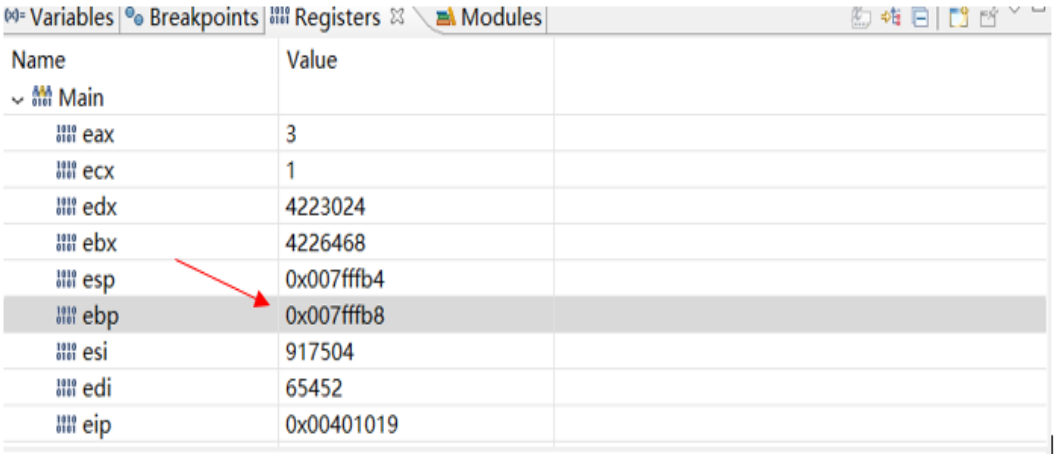
3.1 汇编代码分析

通过对 sum 函数汇编代码的分析，写出注释如下：

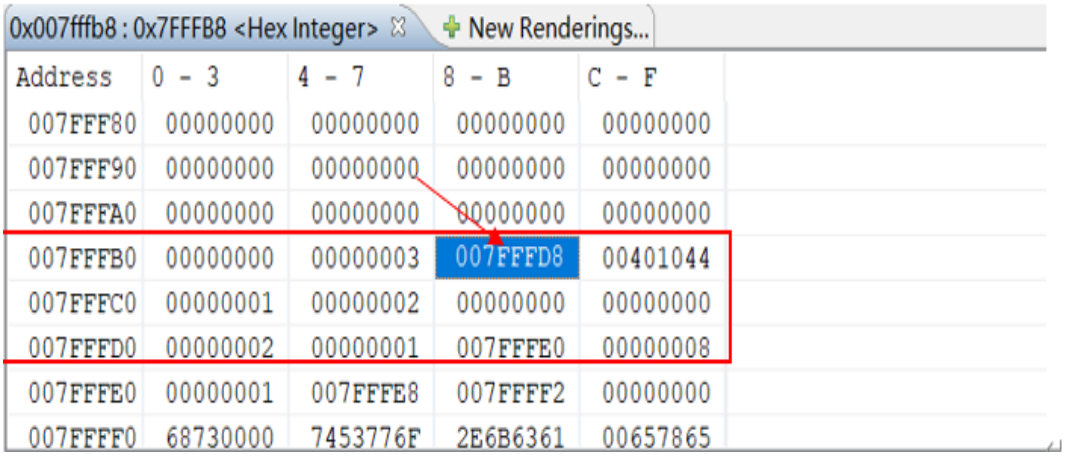
```
          sum:
00401000:   push %ebp            //保存old ebp的值
00401001:   mov %esp,%ebp        //ebp指向esp的值，成为sum函数栈帧的栈底
00401003:   sub $0x4,%esp        //esp向上移动一个字
00401006:   movl $0x2,0x404000   //全局变量version赋值为2
00401010:   mov 0xc(%ebp),%eax   //将var2的值付给寄存器eax
00401013:   add 0x8(%ebp),%eax   //寄存器eax的值加上var1的值
00401016:   mov %eax,-0x4(%ebp) //将寄存器eax的值赋给临时变量count
00401019:   mov -0x4(%ebp),%eax //将count的值赋给eax作为返回值
0040101c:   leave
0040101d:   ret
```

3.2 核心栈绘制

再次进入调试状态，将断点设置在 sum 函数中的 return 语句，通过观察此时 ebp 的值定位堆栈中的情况观察如下：



可以观察到此时 ebp 的值为 0x007fffb8, 该地址中存储的为 main 函数的 ebp 值即 007FFFD8，之后上方存储为 sum 函数局部变量 count 的值 3（此时已经经过计算）。



所以我们可以补全核心栈的表示如下：

4 问题探究

4.1 问题一

经查阅资料得到，在 main1 函数开头，编译器自动调用语句 00401021：*sub*0x18, 预留出 6 个字的空间是为了提供给程序的局部变量一些空间，将定义的局部变量存储在该预留空间中，同时函数的执行过程中可能需要一些临时数据存储，这个额外的栈空间可以用于这些用途。

4.2 问题二

该地址为全局变量 version 的地址，实验如下：

首先从对汇编代码中这个地址出现的位置我们大致可以判断这即为全局变量 version 的地址，为了确认，我们可以“version = 2”该语句后再添加一条语句"version = 3", 通过观察添加语句前后汇编代码的变化来确定是否为 version 的地址。

装 订 线

```
sum:
00401000:   push %ebp
00401001:   mov %esp,%ebp
00401003:   sub $0x4,%esp
00401006:   movl $0x2,0x404000
00401010:   movl $0x3,0x404000
0040101a:   mov 0xc(%ebp),%eax
0040101d:   add 0x8(%ebp),%eax
00401020:   mov %eax,-0x4(%ebp)
00401023:   mov -0x4(%ebp),%eax
00401026:   leave
00401027:   ret
```

```
int sum(var1,var2)
{
    int count;
    version = 2;
    version = 3;
    count = var1 + var2;
    return(count);
}
```