

P02 UNIX V6++进程的栈帧

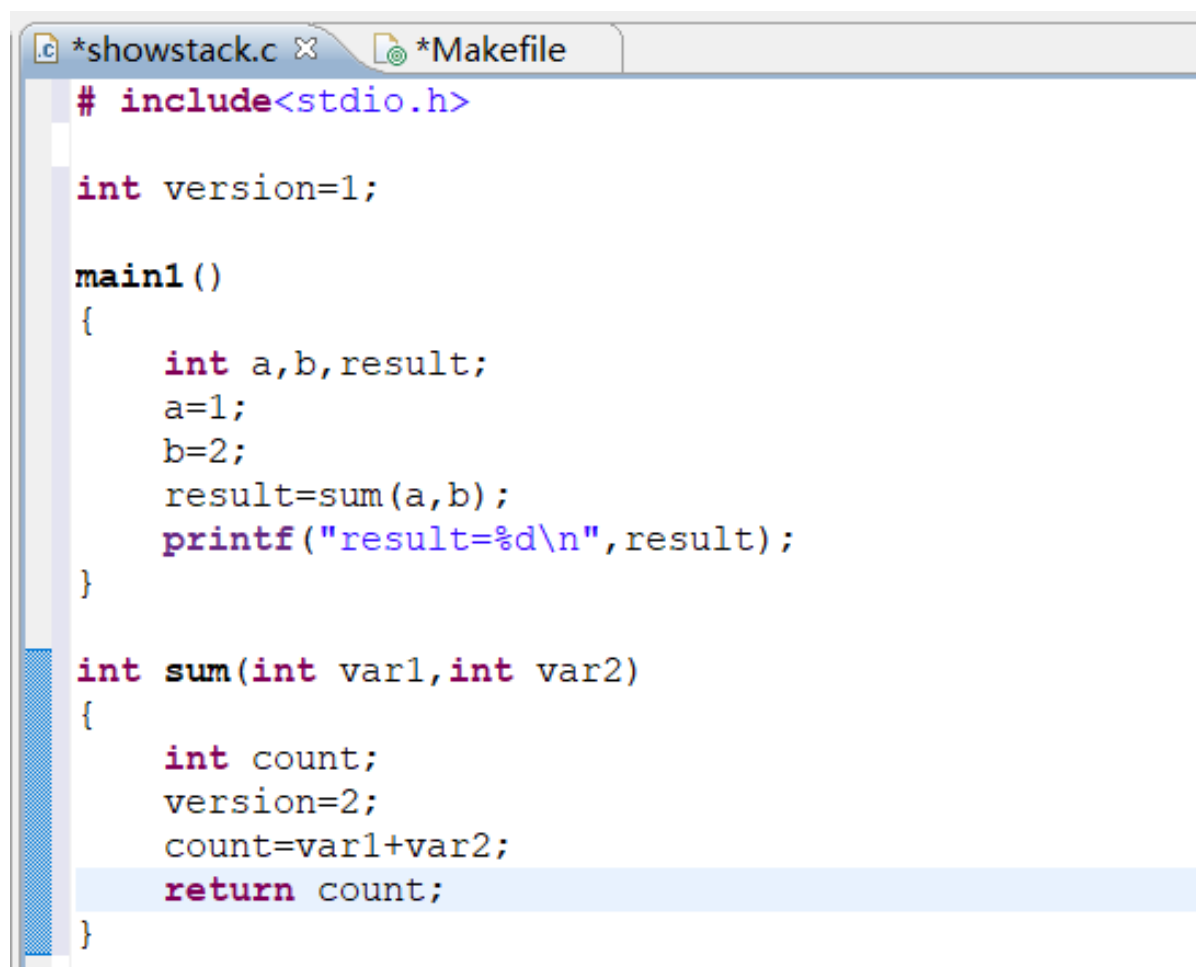
一、实验目的

通过编写一个简单的 C++ 代码，并在 UNIX V6++ 中编译和运行调试，观察程序运行时核心栈的变化。通过实践，进一步掌握 UNIX V6++ 重新编译及运行调试的方法。

二、实验内容

2.1 在UNIX V6++中编译链接一个c语言程序

在program中添加一个新的c语言文件并修改Makefile文件



```
*showstack.c x *Makefile
# 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(int var1,int var2)
{
    int count;
    version=2;
    count=var1+var2;
    return count;
}
```

```
*showstack.c  *Makefile x
SHELL_OBJS =$(TARGET)\cat.exe \
            $(TARGET)\cat1.exe \
            $(TARGET)\cp.exe \
            $(TARGET)\ls.exe \
            $(TARGET)\mkdir.exe \
            $(TARGET)\rm.exe \
            $(TARGET)\perf.exe \
            $(TARGET)\sig.exe \
            $(TARGET)\copyfile.exe \
            $(TARGET)\shutdown.exe \
            $(TARGET)\test.exe \
            $(TARGET)\forks.exe \
            $(TARGET)\trace.exe \
            $(TARGET)\echo.exe \
            $(TARGET)\date.exe \
            $(TARGET)\newsig.exe \
            $(TARGET)\sigTest.exe \
            $(TARGET)\stack.exe \
            $(TARGET)\malloc.exe\
            $(TARGET)\showstack.exe

#$(TARGET)\performance.exe
```

在project———build all将项目重新编译成功后，运行unixv6++，进入bin文件夹，通过ls命令查看所有可执行文件，运行showstack.exe,可以得到result=3的结果

```
Bochs for Windows - Display
A: B: CD USER Copy Paste Snapshot CONFIG Reset SUSPEND Power
[/bin]#cd bin
Invalid path!
[/bin]#ls
Directory '/bin':
cat      cat.exe cat1.exe      cp      cp.exe  cpfile.exe  date  date.e
echo     echo.exe      forks.exe  ls      ls.exe  malloc.exe
mkdir   mkdir.exe      newsig.exe perf    perf.exe  rm      rm.exe
showstack.exe shutdown shutdown.exe sig.exe sigTest.exe stack.
e      test.exe      trace     trace.exe

[/bin]#showstack.exe
result=3
[/bin]#_

ess Exit! Process 6 execing
Process 6 is exiting
end sleep
Process 6 (Status:5) end wait
Process 1 finding dead son. They are Process 7 (Status:3) wait until child pr
ess Exit! Process 7 execing
Process 7 is exiting
end sleep
Process 7 (Status:5) end wait
```

2.2 程序的调试运行

将调试对象设置为showstack.exe，调试入口为main1

Name: oos

Main Arguments Environment Debugger Source Refresh Common

C/C++ Application:
 D:\study_files\operating system\unix_tools\64\oos\src\program\objs\showsta Search Project... Browse...

Project:
 oos Browse...

Build (if required) before launching

Build configuration: Default

☒ Select configuration using 'C/C++ Application'

☐ Enable auto build ☐ Disable auto build

☒ Use workspace settings [Configure Workspace Settings...](#)

☒ Connect process input & output to a terminal.

Using Standard Create Process Launcher - [Select other...](#) Apply Revert

Main Arguments Environment Debugger Source

Debugger: gdbserver

☒ Stop on startup at: main1 Advanced...

Debugger Options

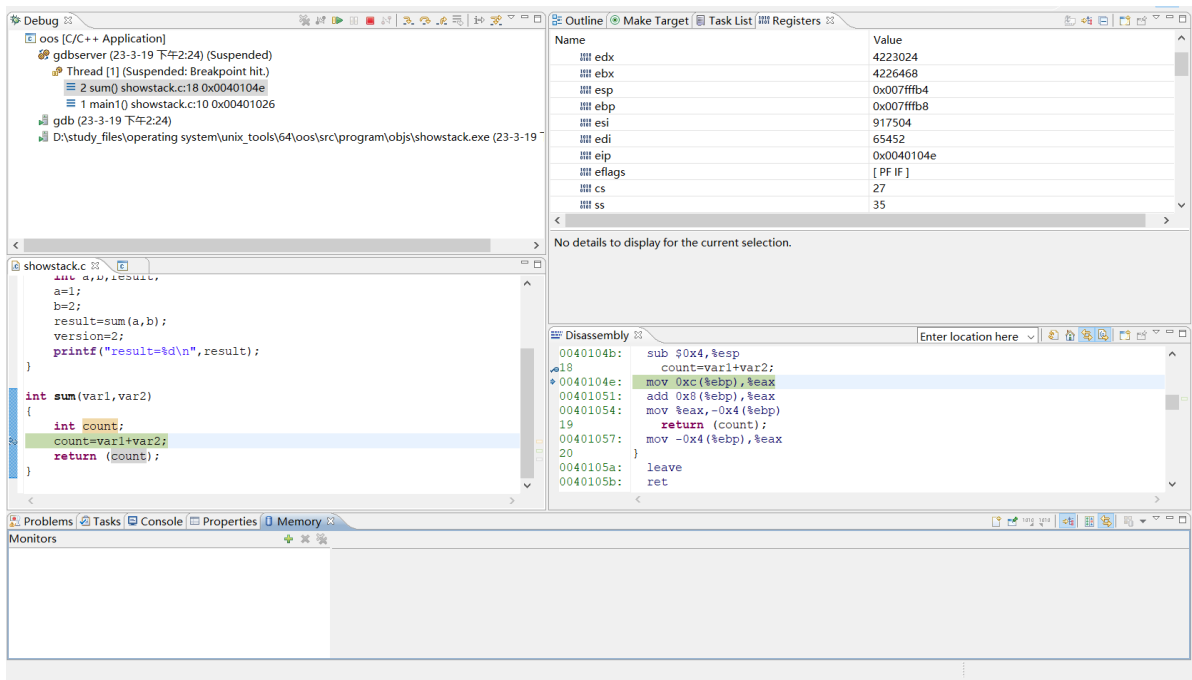
然后以调试模式打开UNIXV6++，在虚拟机中执行cd bin和showstack.exe命令，开始调试

Bochs for Windows - Display

A: B: CD USER Copy Paste Snapshot CONFIG Reset SUSPEND Power

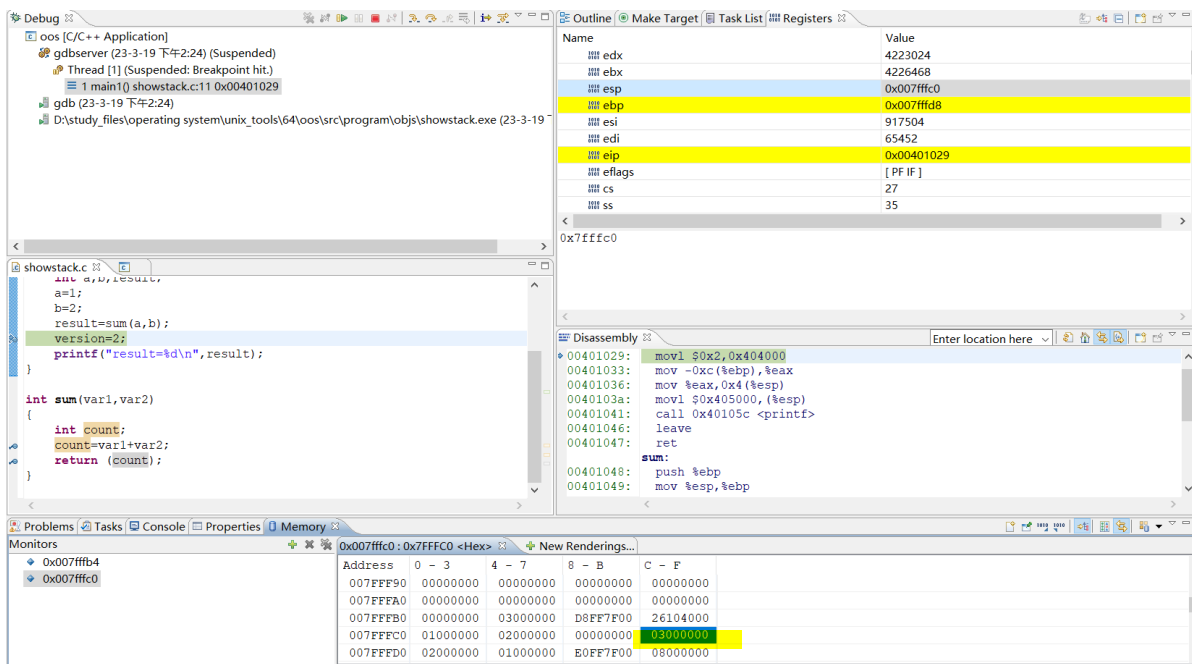
```
[/]#cd bin
[/bin]#showstack.exe
result=3
[/bin]#showstack.exe
```

Process 1 finding dead son. They are Process 3 (Status:3) wait until child process Exit! Process 3 execing



可以在语句停在sum函数中某一个位置时，通过多个窗口查看到寄存器、调试状态和汇编语言的情况

4.3 观察堆栈变化



通过memory中监视esp寄存器的地址可以发现a, b, result三个变量的储存位置。

```

1  sum:
2  0040103e: push %ebp           //将main函数的ebp压栈,传入当前函数
3  0040103f: mov %esp,%ebp       //将当前函数的esp赋值给ebp,指向当前栈帧
4  00401041: sub $0x4, %esp      //esp上移1个字,给局部变量count
5  17 version=2;
6  00401044: movl $0x2,0x404000  //把2赋值给全局变量version的地址,
7  18 count=var1+var2;
8  0040104e: mov 0xc(%ebp), %eax //从ebp+12的位置把 var1 的值送入 eax
9  00401051: add 0x8(%ebp), %eax //从ebp+8的位置把var2的值送入eax并相加
10 00401054: mov %eax,-0x4(%ebp) //将eax寄存器中的值存储在ebp-4的位置,即局部变
    量count的地址
11 19 return(count);

```

```

12 00401057: mov -0x4(%ebp), %eax    //将局部变量count的值送入eax寄存器中，作为函数的
    返回值
13 0040105a: leave
14 0040105b: ret

```

sum函数核心栈：（从上往下是ESP->EBP）

1	ESP			
2			+-----+	
3	EIP	0x007FFBC	00401026	sum函数返回地址
4			+-----+	
5		0x007FFCC	3	局部变量result 00401054语句后变为3
6			+-----+	
7		0x007FFCC	2	第二个参数var1，值为2
8			+-----+	
9		0x007FFD8	1	第一个参数var1，值为1
10			+-----+	
11		0x007FFD8	上一函数（main）调用的ebp地址	0040103e语句压栈，0040103f语句 将esp更改到此
12			+-----+	
13			返回地址 00000008	
14	EBP		+-----+	

问题二：关于地址0x404000 其实存放就是全局变量version的地址

通过监视改地址的值，发现在没有执行代码时是一个随机值，在进入主函数后变成了1，执行完version=2后又变成了2，说明其实存放的就是全局变量version

The screenshot displays the GDB interface with the following components:

- Thread [1] (Suspended: Breakpoint hit.):** Shows the current thread and the function `main1()` at address `0x00401000`.
- Source Code:** The file `showstack.c` is open, showing the `main1()` function which calls `sum(a, b)` and updates the `version` variable.
- Disassembly:** The assembly code for `main1()` is shown, starting with `push %ebp` and `mov %esp, %ebp`.
- Memory:** The memory window shows a hex dump of memory addresses. The address `0x404000` is highlighted, showing a value of `0x00000001`.

- oos [C/C++ Application]
 - gdbserver (23-3-19 下午4:26) (Suspended)
 - Thread [1] (Suspended: Breakpoint hit.)
 - 1 main() showstack.c:13 0x00401048
 - gdb (23-3-19 下午4:26)
 - D:\study_files\operating system\unix_tools\64\oos\src\program\objs\showstack.exe (23-3-19

stdio.h

- version: int
- main1()
- sum(,): int

```
showstack.c
a=1;
b=2;
printf("%d",version);
result=sum(a,b);
version=2;
printf("result=%d\n",result);
}

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

Disassembly

```

00401048: mov -0xc(%ebp),%eax
0040104b: mov %eax,0x4(%esp)
0040104f: movl $0x405003, (%esp)
00401056: call 0x401074 <printf>
14
0040105b: leave
0040105c: ret
17
{
sum:
0040105d: push %ebp
```

Problems Tasks Console Properties Memory Variables Call Hierarchy Search

Monitors

- 0x007fffb8
- 0x00404000
- 0x007ffd40
- 0x404000

0x404000 <Hex>		0x404000 : 0x404000 <Hex Integer>		New Renderings...	
Address	0 - 3	4 - 7	8 - B	C - F	
00404000	00000002	00405010	00000015	00000000	
00404010	00000000	00000000	00000000	00000000	
00404020	00000000	00000000	00000000	00000000	
00404030	00000000	00000000	00000000	00000000	