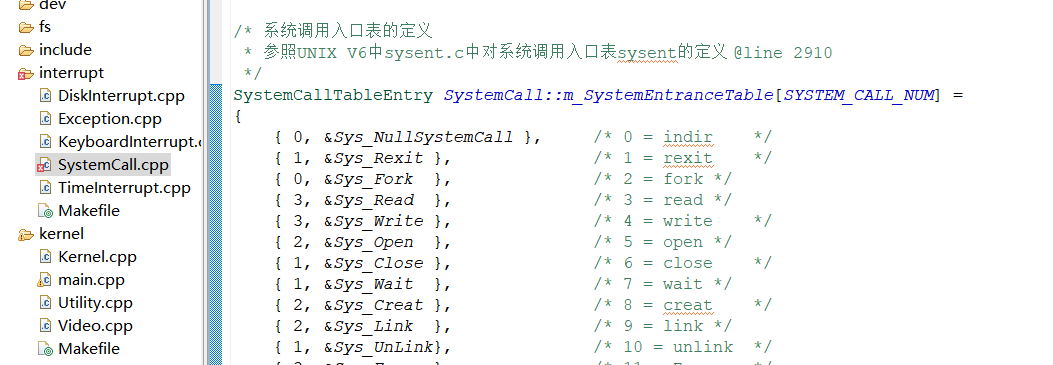
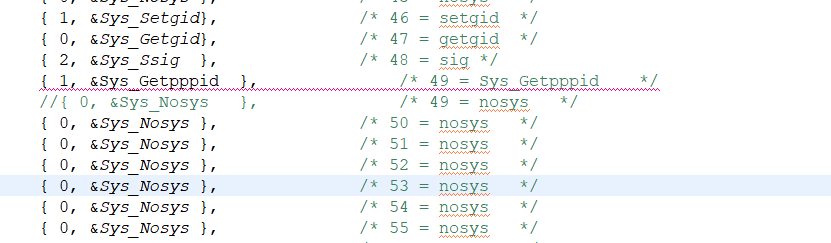
**P04：UNIX V6++中添加新的系统调用**

**2152118 史君宝**

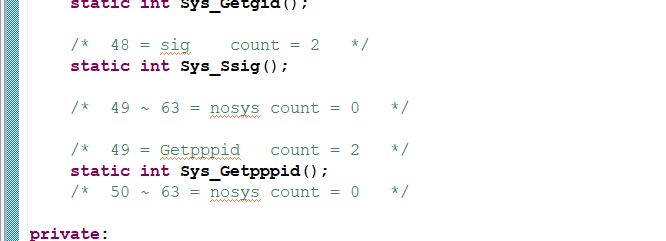
**任务一、完成实验 4.1，截图说明操作过程。**

1. 在系统调用子程序入口表中添加新的入口

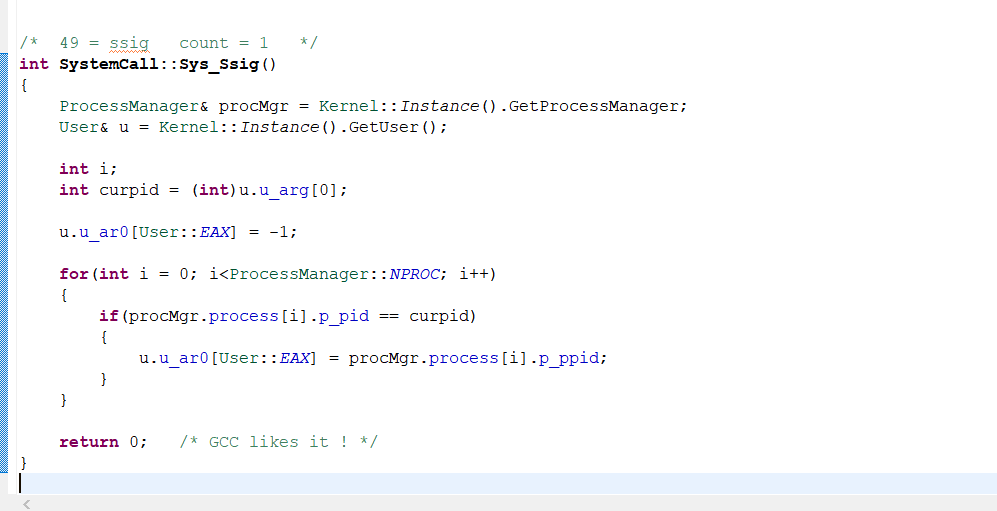




1. 在 SystemCall 类中添加系统调用处理子程序的定义



1. 在 SystemCall.cpp 中添加 Sys\_Getppid 的定义



1. 总结具体步骤：

添加系统调用的具体步骤：

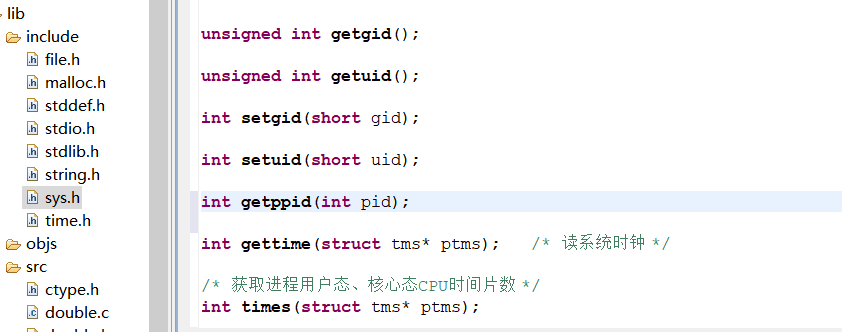
步骤一：在系统调用子程序入口表中添加新的入口。

步骤二：在 SystemCall 类中添加系统调用处理子程序的定义。

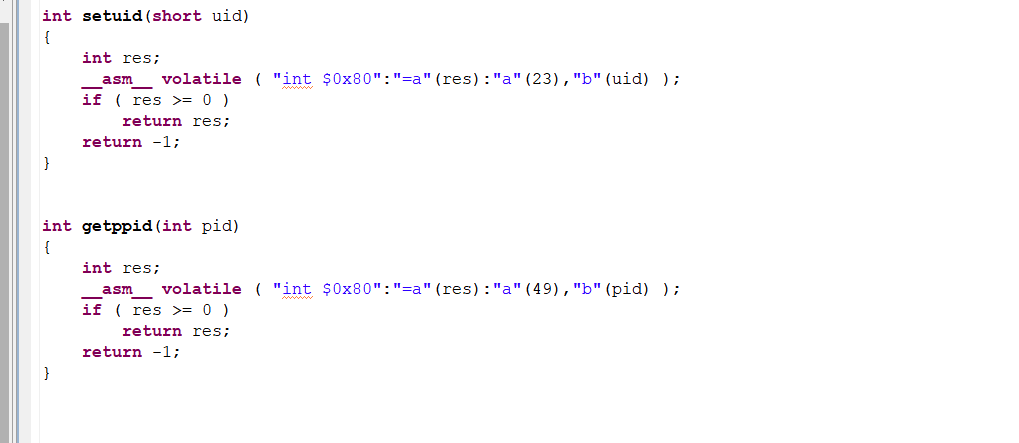
步骤三：在 SystemCall.cpp 中添加 Sys\_Getppid 的具体定义。

**任务二、完成实验 4.2，掌握在 UNIX V6++中添加库函数的方法。**

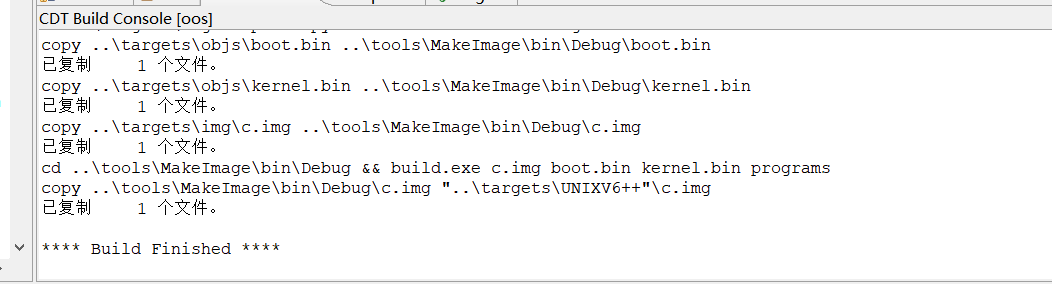
1. 在 sys.h 文件中添加库函数的声明



（2）在 sys.c 中添加库函数的定义



（3）编译程序：



成功编译。

（4）总结具体步骤：

添加库函数的具体步骤：

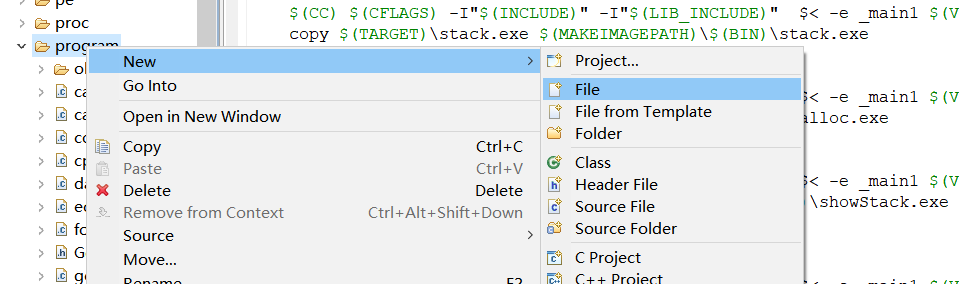
步骤一：在 sys.h 文件中添加库函数的声明。

步骤二：在 sys.c 中添加库函数的定义。

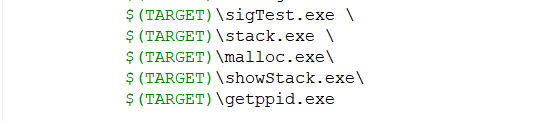
步骤三：编译程序。

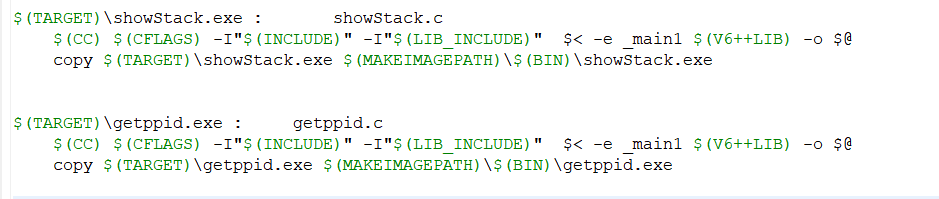
**任务三、完成实验 4.3-4.4，编写测试程序。**

（1）创建测试程序：

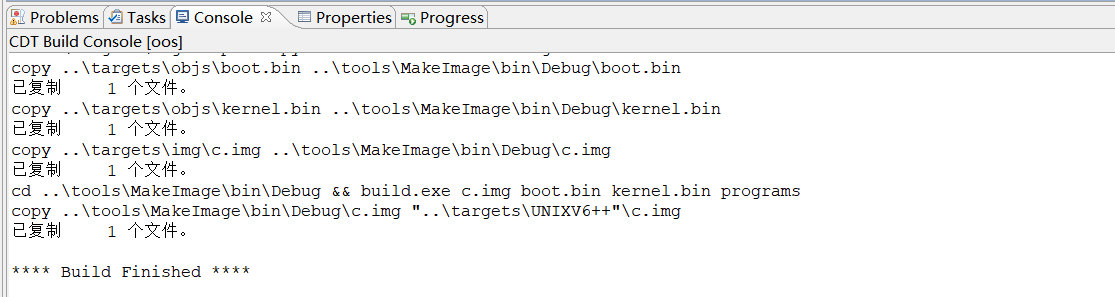


1. 修改编译使用的Makefile文件：

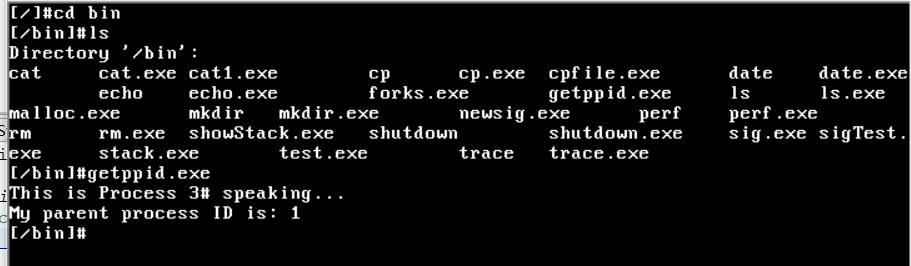




1. 编译程序：

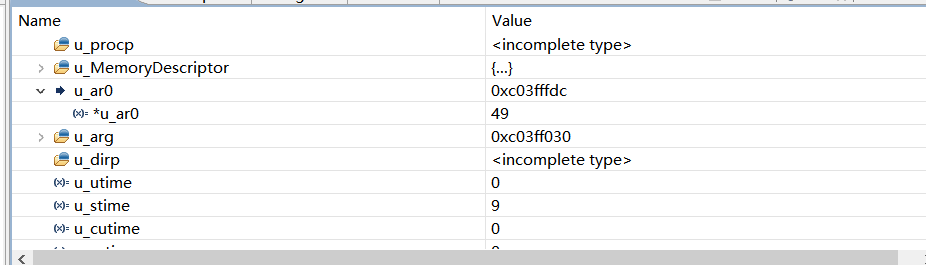


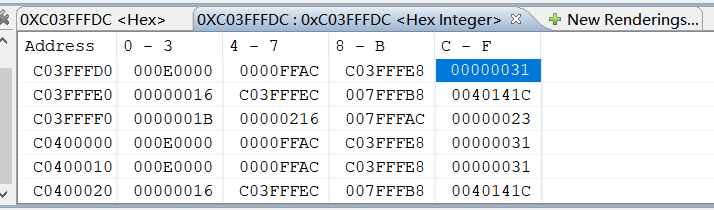
1. 调试程序观察结果：



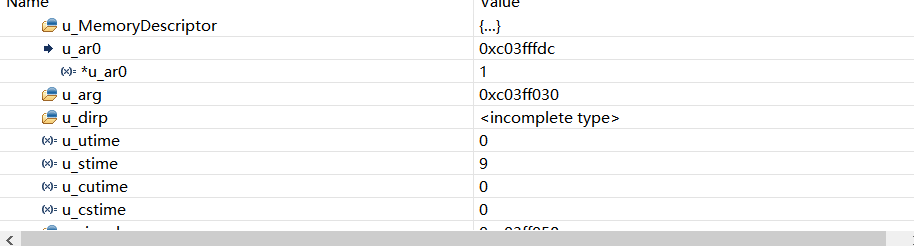
1. 设置断点，具体调试观察现象。

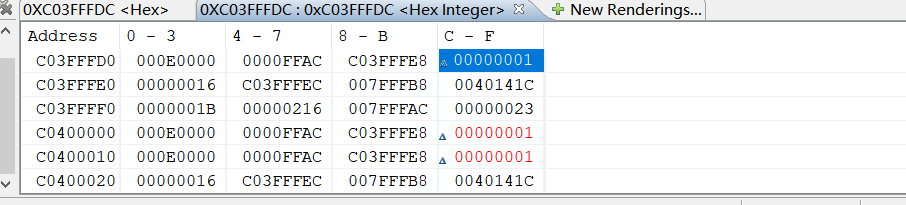
执行到断点： int curpid = (int)u.u\_arg[0];





执行到断点：Getppid函数的最后一句。





|  |  |  |
| --- | --- | --- |
|  | EBP | 0xC03FFFE8 |
| 0xC03FFFDC | EAX | 49 |
|  |  | 0x00000016 |
|  |  | 0xC03FFFEC |
| 0xC03FFFE8 |  | 0xC07FFFE8 |
| 0xC03FFFEC | EIP | 0x0040141C |
|  | CS | 0x0000001B |
|  | EFLAGS | 0x00000216 |
|  | ESP | 0x007FFFAC |
| 0xC03FFFFC | SS | 0x00000023 |

可以还原出核心栈应该为上面。

**任务四、在完成 4.4 的基础上，设计调试方案，确定图 10 中黄色标注的几个地址单元分别是什么。**



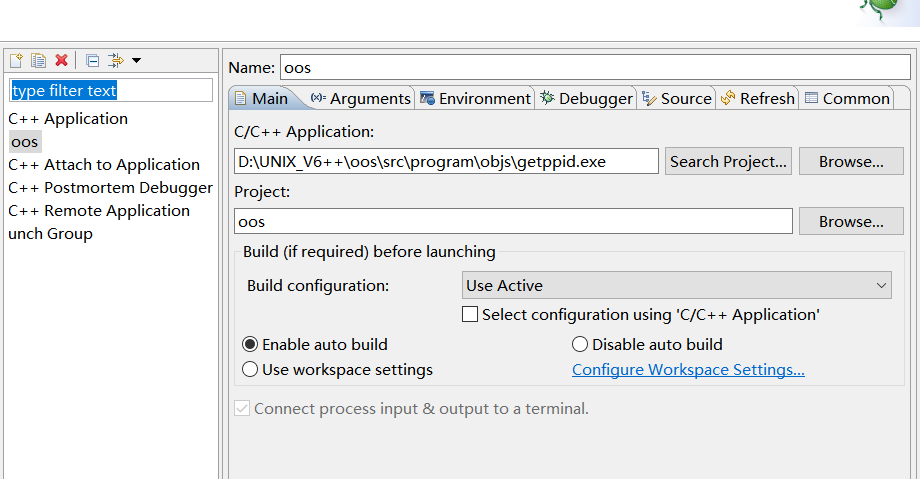
ebp 值是 getppid 函数栈帧的栈基址。

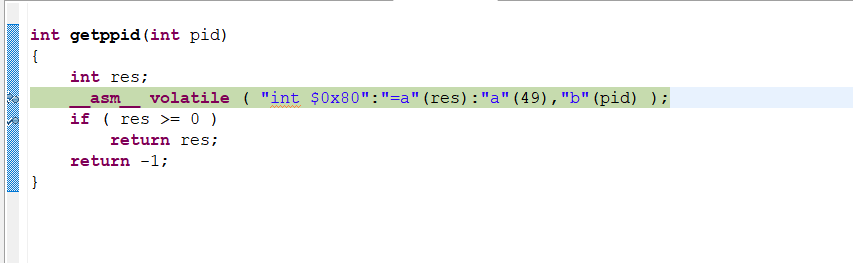
esp 为指向 getppid 函数栈帧的栈顶。

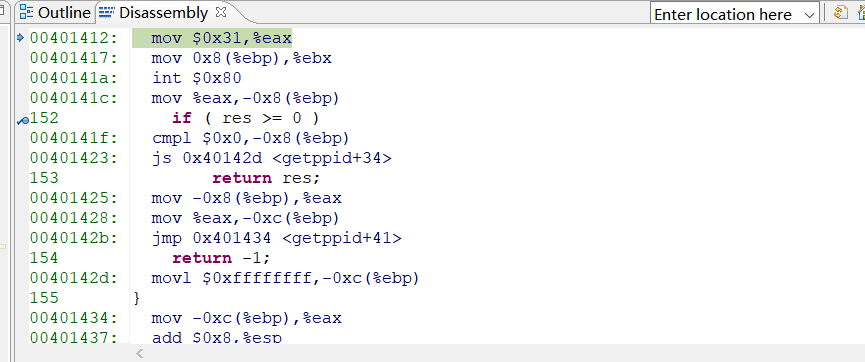
eip 寄存器的值为是下⼀指令的地址，执行现场保护后存入核心栈。

在 getppid 函数的

\_ asm \_ volatile( "int$0x80":"=a"(res):"a"(49),"b"(pid)); 语句处设置断点，修改调试对象和调试入口。



、



我们通过查看在getppid具体程序的执行过程时的 memory 和 disassemebly 可以发现，

第一个黄色地址单元中，esp 的值为 0x007FFFAC，是执行系统调用前的用户栈的栈顶位置。

第二个黄色单元中，eip 的值为 0x0040141C，表示执行完 int $0x80 指令后的

下一条指令的地址。

第三个黄色单元中，0xC03FFFE8 是指向 getppid 函数栈帧的 ebp，即当前栈

帧的基址指针。

第四个黄色单元中，ebp 的值为 0xC07FFFE8，表示指向 49 号系统调用入口程序的栈帧的地址。