**同济大学计算机系**

**操作系统课程实验报告**

****

**学 号 2251557**

**姓 名 代文波**

**专 业 计算机科学与技术**

**授课老师 方钰**

**1. 实验目的**

（1）结合课程所学知识，通过在 UNIX V6++源代码中实践操作添加一个新的系统调用，熟悉 UNIX V6++中系统调用相关部分的程序结构。

（2）通过调试观察一次系统调用的全过程，进一步理解和掌握系统调用响应与处理的流程，特别是 其中用户态到核心态的切换和栈帧的变化。

（3）通过实践，进一步掌握 UNIX V6++重新编译及运行调试的方法。

**2. 实验设备及工具**

已配置好的UNIX V6++运行和调试环境。

**3. 预备知识**

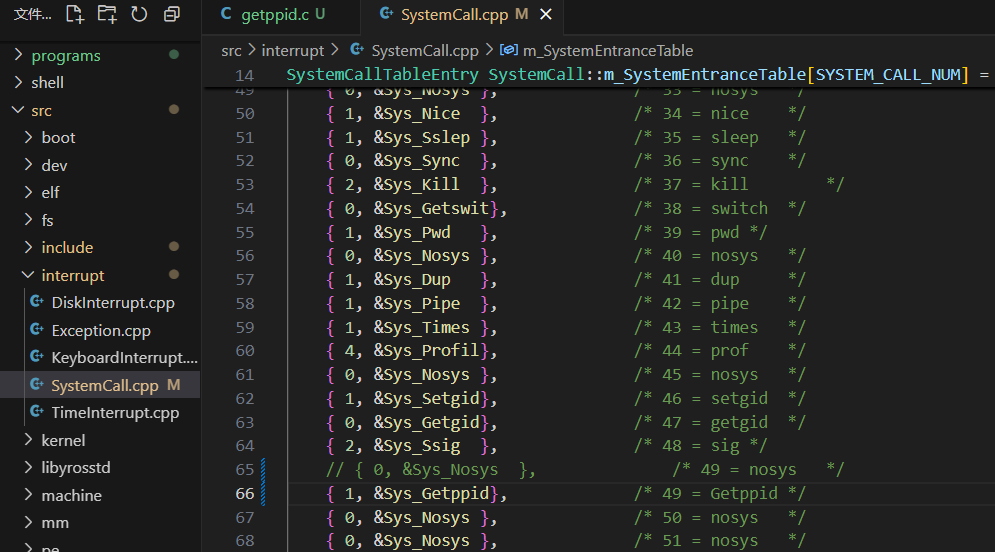
（1）UNIX V6++中系统调用的执行过程；

（2）UNIX V6++中所有和系统调用相关的代码模块。

**4. 实验内容**

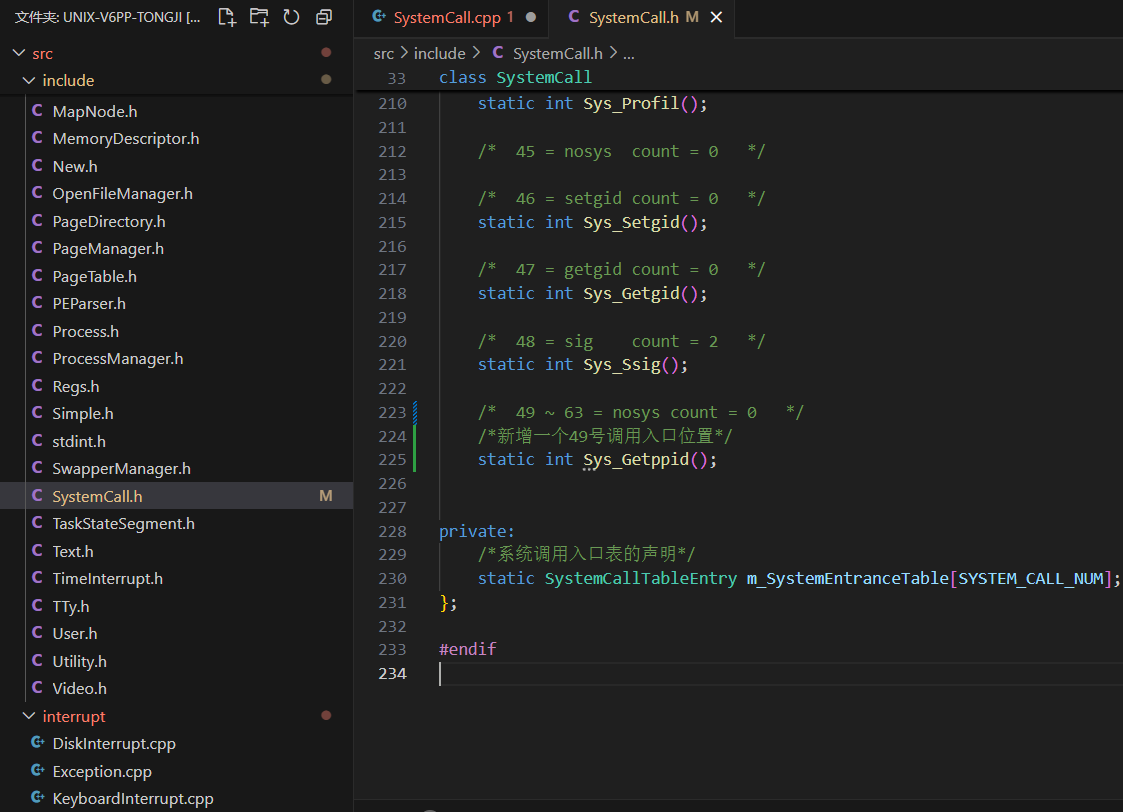
4.1 在 UNIX V6++中添加一个新的系统调用接口

4.1.1. 在系统调用处理子程序入口表中添加新的入口

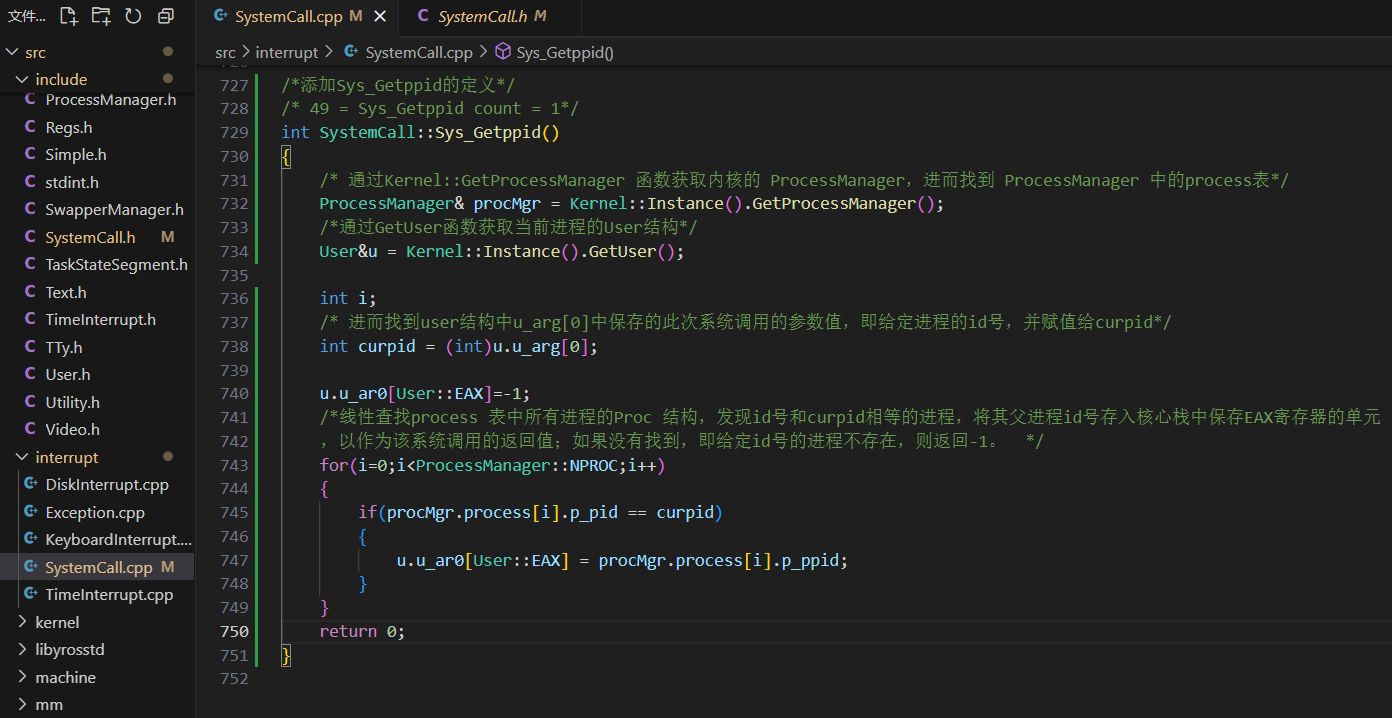


4.1.2在 SYSTEMCALL类中添加新的系统调用处理子程序

（1）在SystemCall.h 文件中添加该系统调用处理子程序Sys\_Getppid 的声明



（2）在SystemCall.cpp 中添加 Sys\_Getppid 的定义



**【总结】在UNIX V6++中添加一个新的系统调用的步骤**

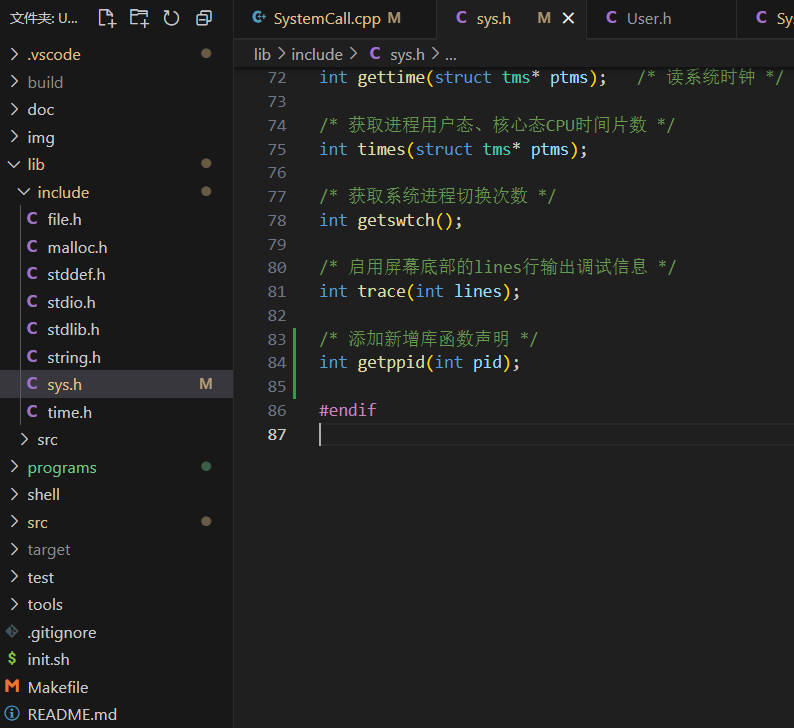
① 在系统调用处理子程序入口表中添加新的入口，具体操作为：在SystemCall.cpp中系统调用子程序入口表m\_SystemEntranceTable中选择一个空项（赋值为{ 0, &Sys\_Nosys }的项），将其修改为{所需参数个数，系统调用处理子程序的入口地址}；

② 在SystemCall.h 文件中添加该系统调用处理子程序的声明

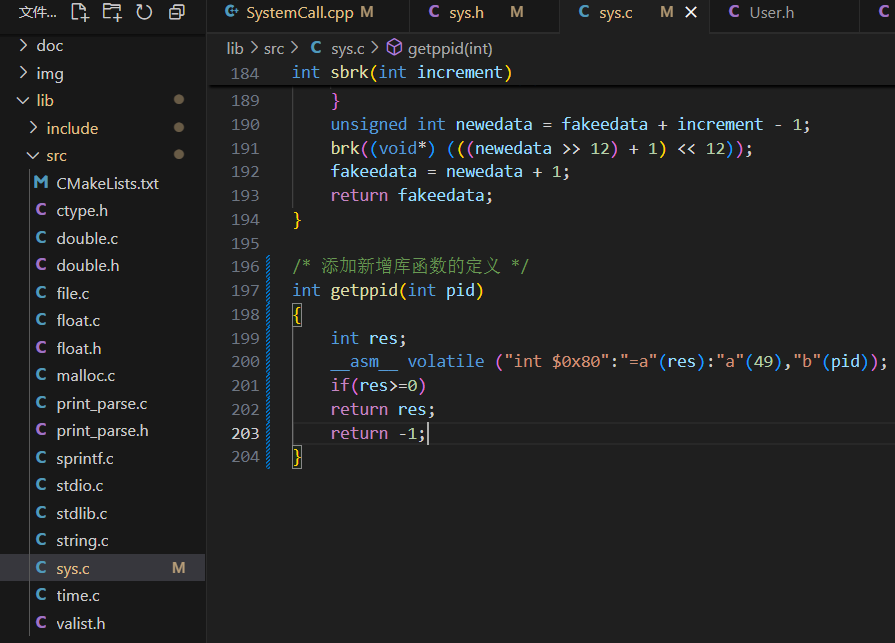
③ 在SystemCall.cpp 中添加 该系统调用子程序的定义

4.2. 为新的系统调用添加对应的库函数

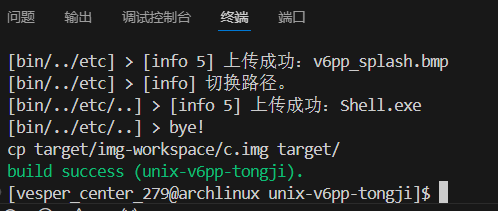
4.2.1在 SYS.H文件中添加库函数的声明

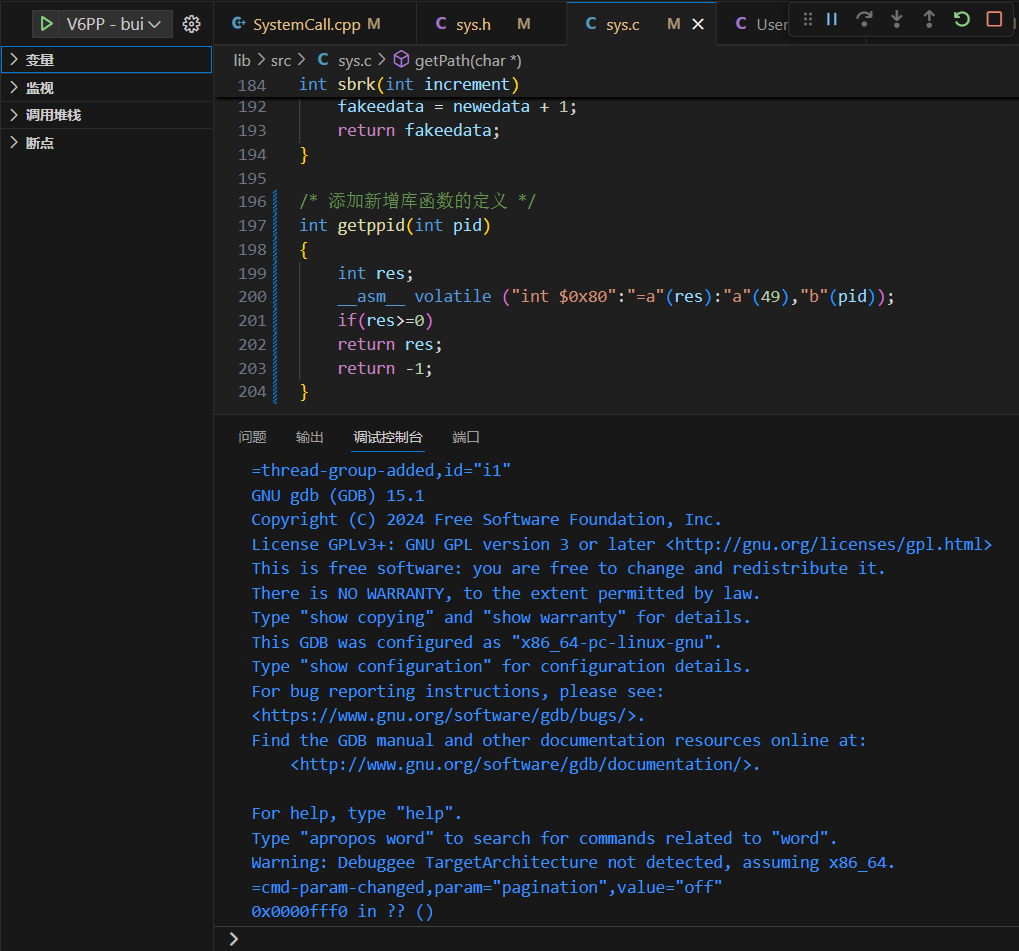


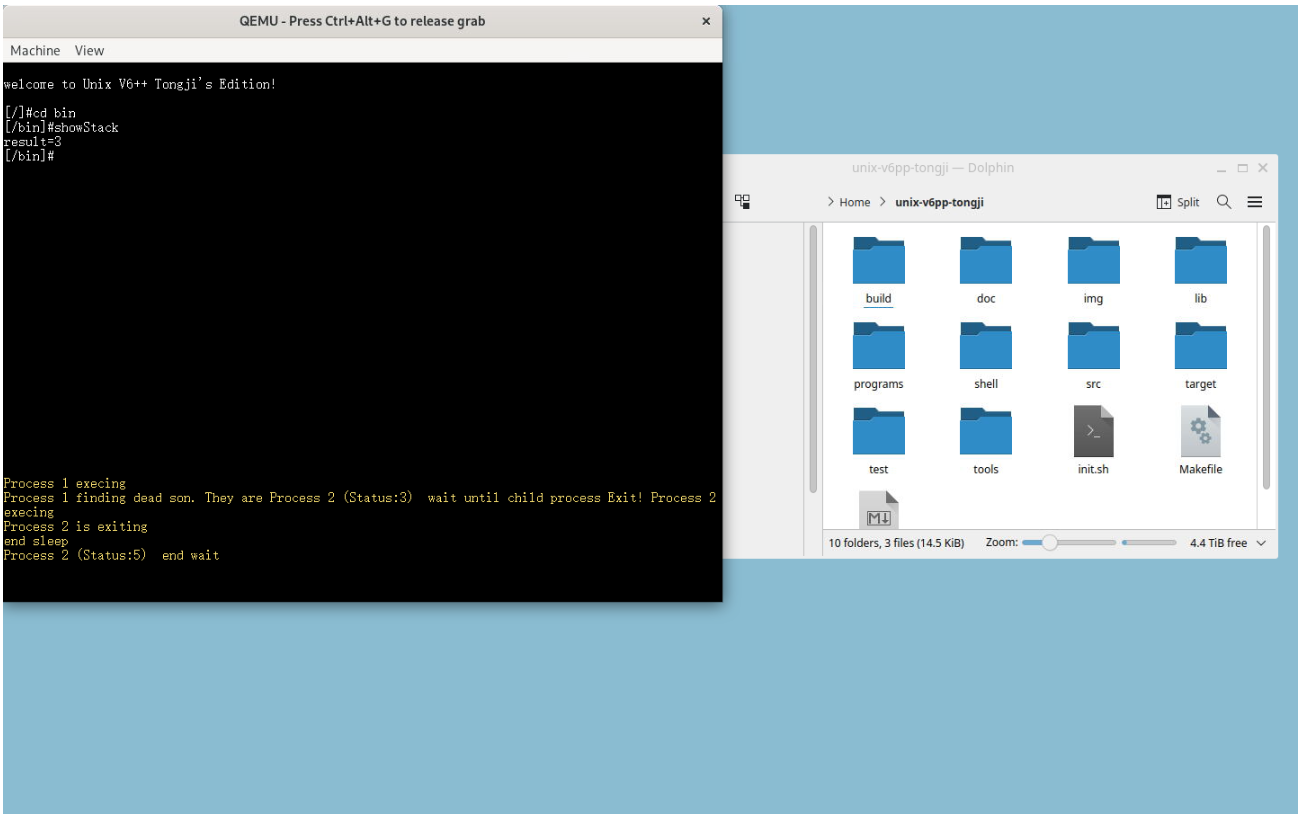
4.2.2. 在 SYS.C 中添加库函数的定义



4.2.3重新编译UNIX V6++，输入make all指令，没有问题

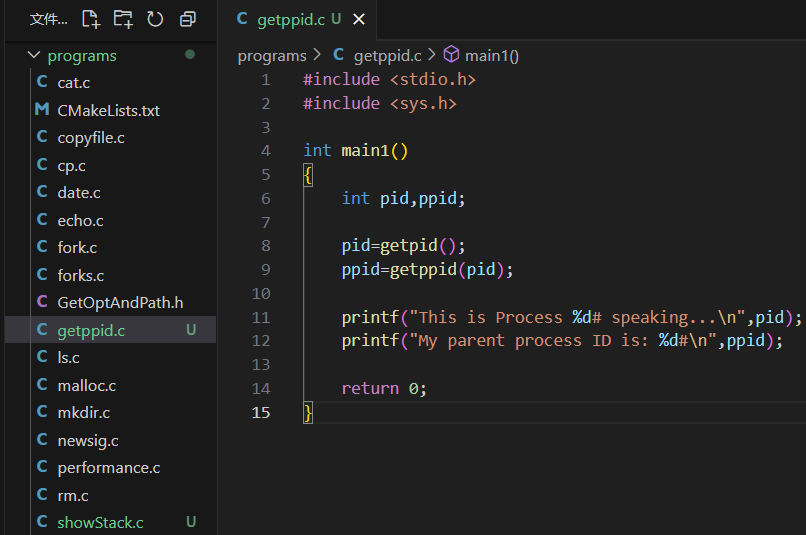




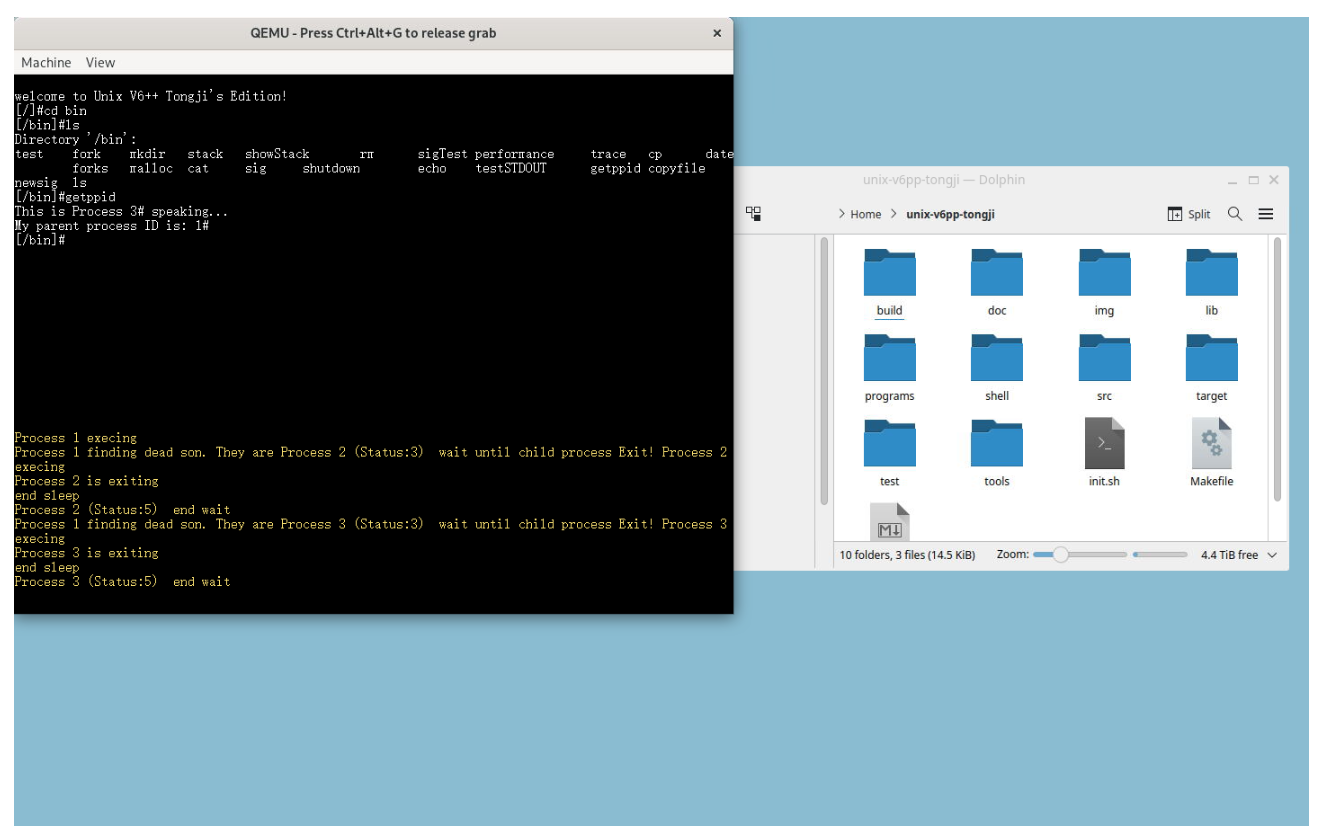


4.3编写测试程序

4.3.1 编写getppid.c文件



4.3.2在运行模式下启动 UNIX V6++，观察程序的输出结果



可以看到，实验结果显示正确。

**【总结】**：

1、在 UNIX V6++中添加库函数的步骤：

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

2、重新编译的步骤：

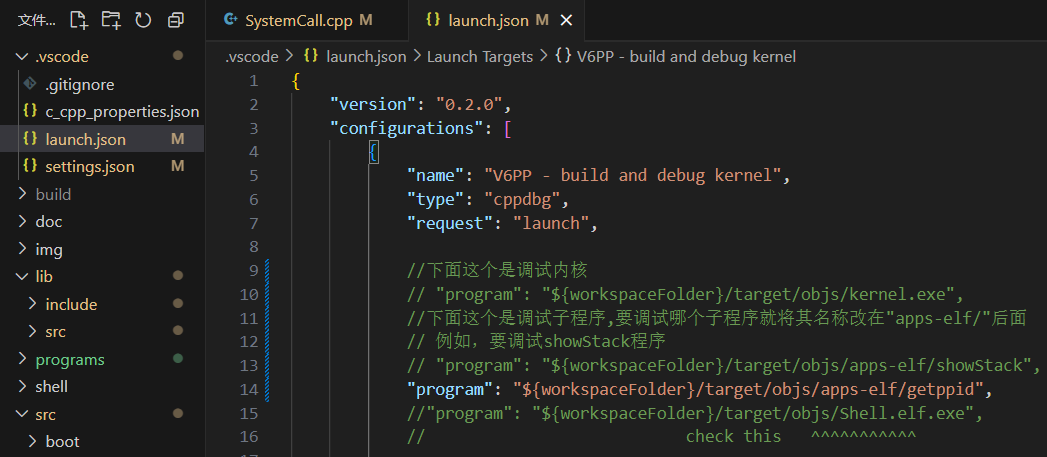
在UNIX V6的终端窗口输入“make all”指令

**4.4 调试程序**

4.4.1. 观察系统调用参数和返回值的传递

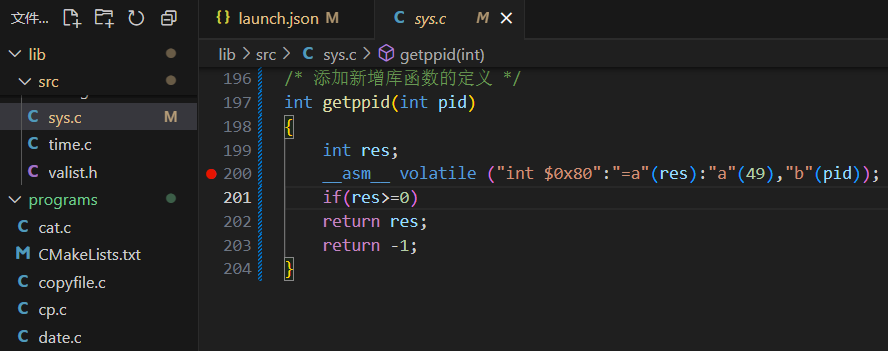
（1）观察用户态到核心态参数的传递

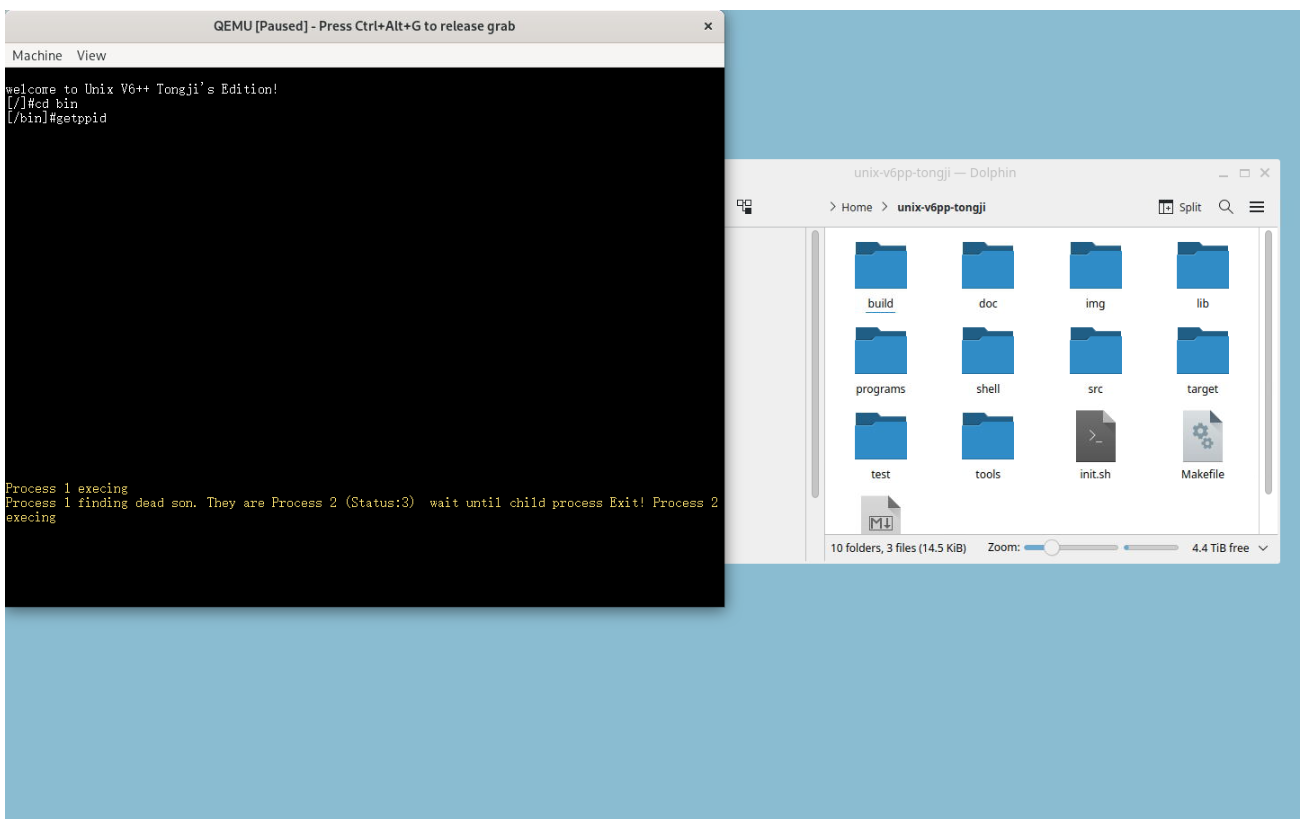
① 将调试目标设为应用程序getppid

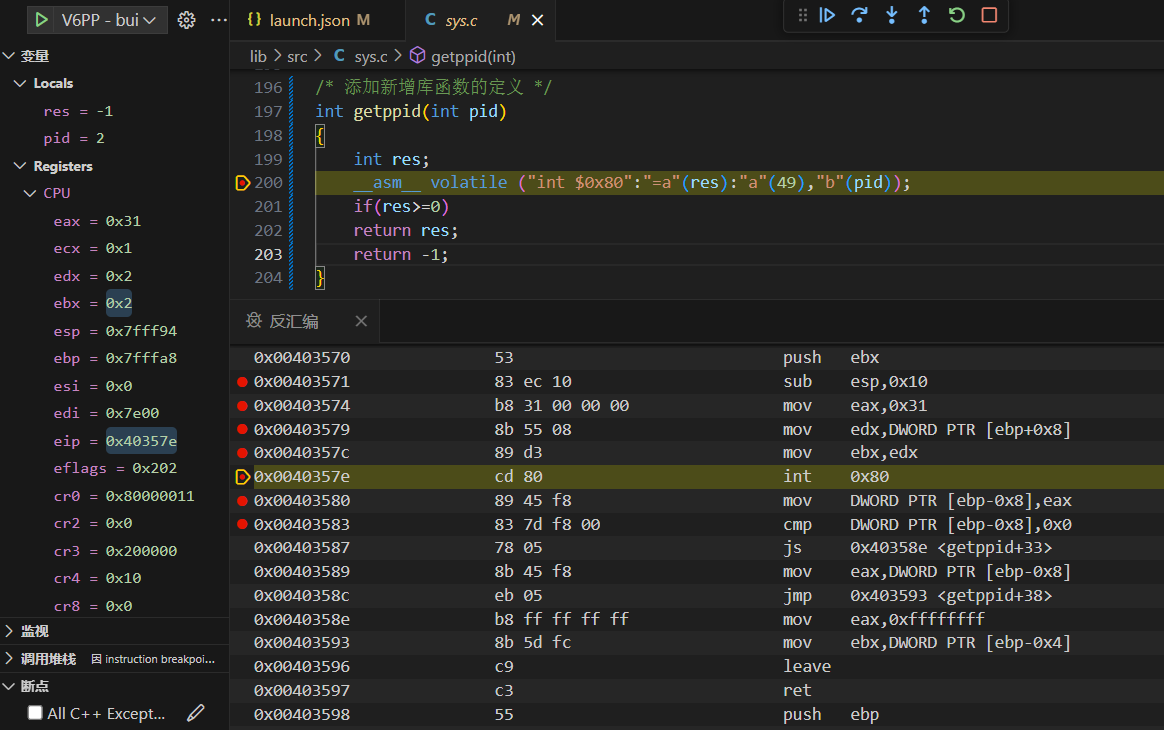


② 将断点设置在库函数getppid中的语句：

“\_\_asm\_\_ volatile ("int $0x80":"=a"(res):"a"(49),"b"(pid));”



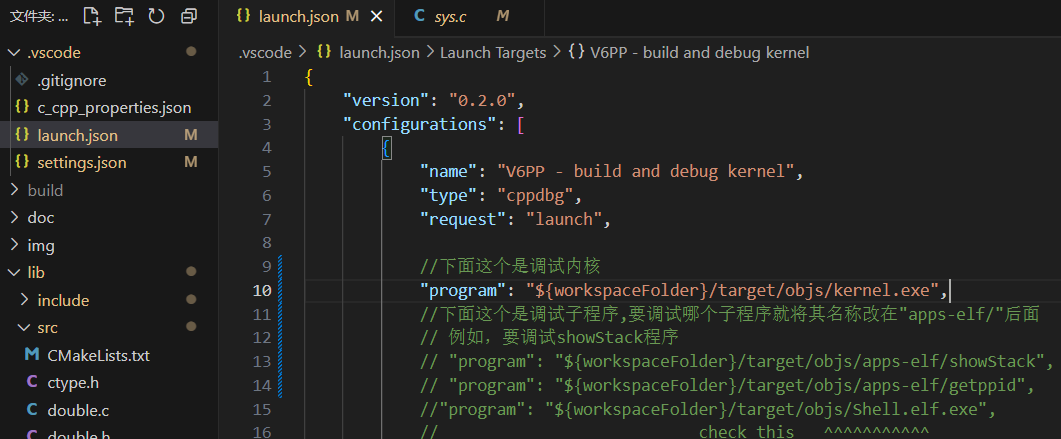




从上面的图片，我们可以看到，当将要执行int 80指令时，eax中为系统调用号49（16进制0x31），ebx中为参数值2（现运行进程的ID号）， eip 的值正好是“int $0x80”的地址。

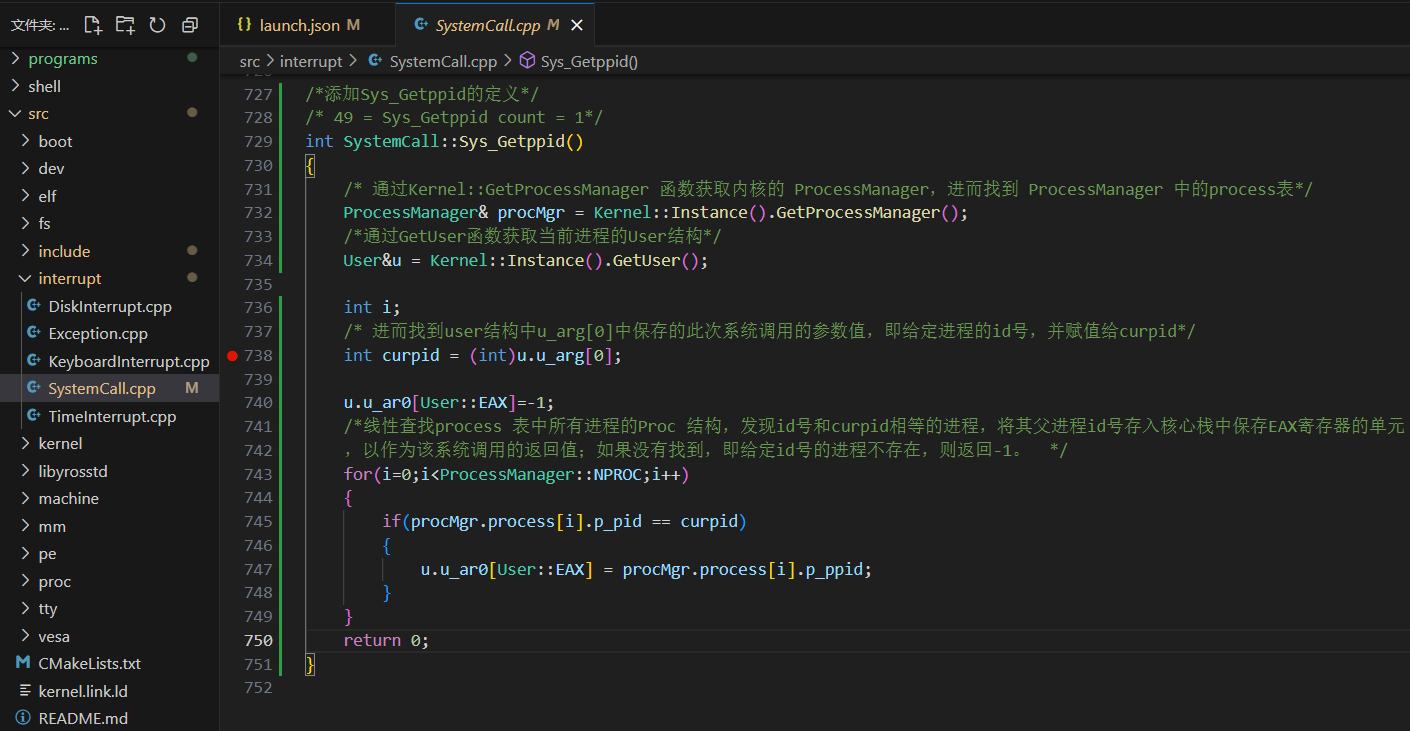
（2）观察核心态到用户态的返回值的传递

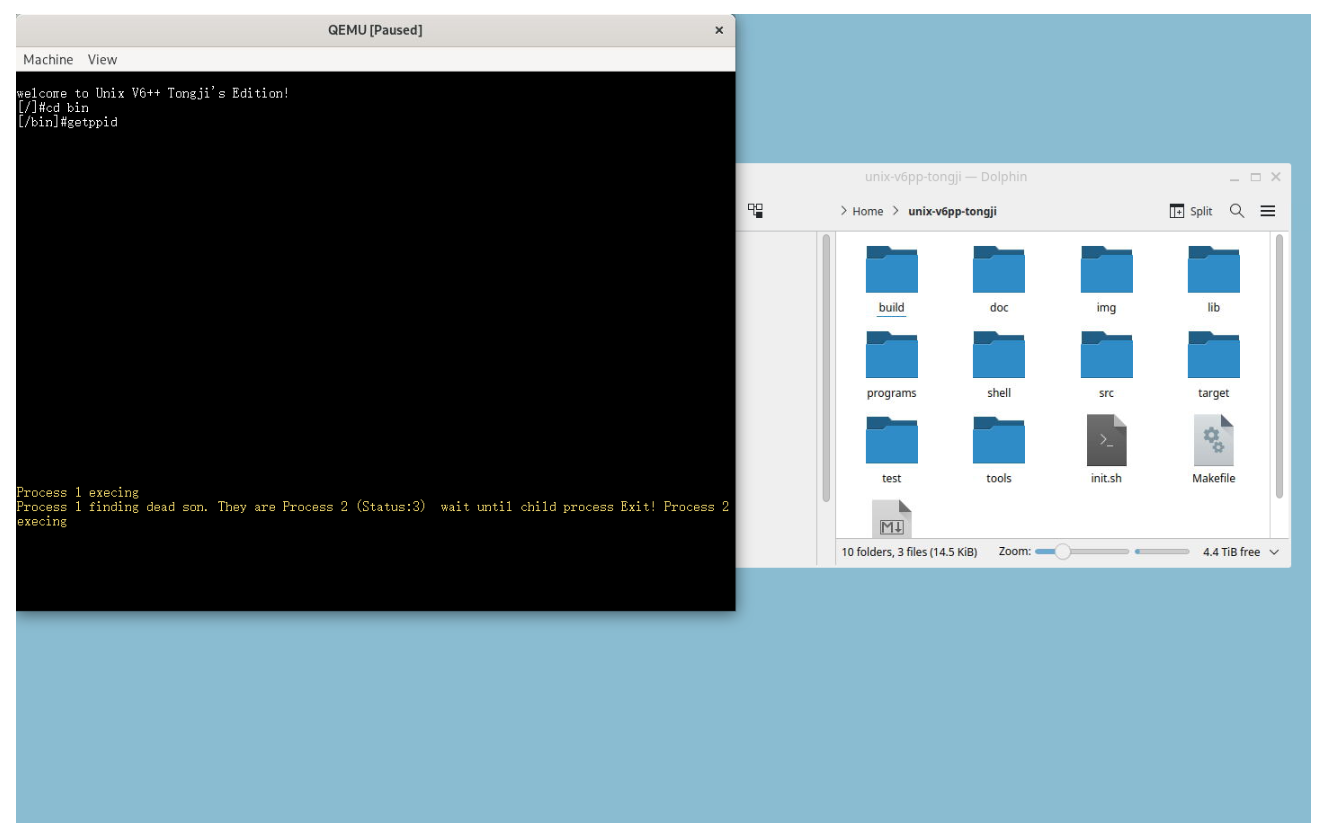
① 将调试目标修改为内核调试

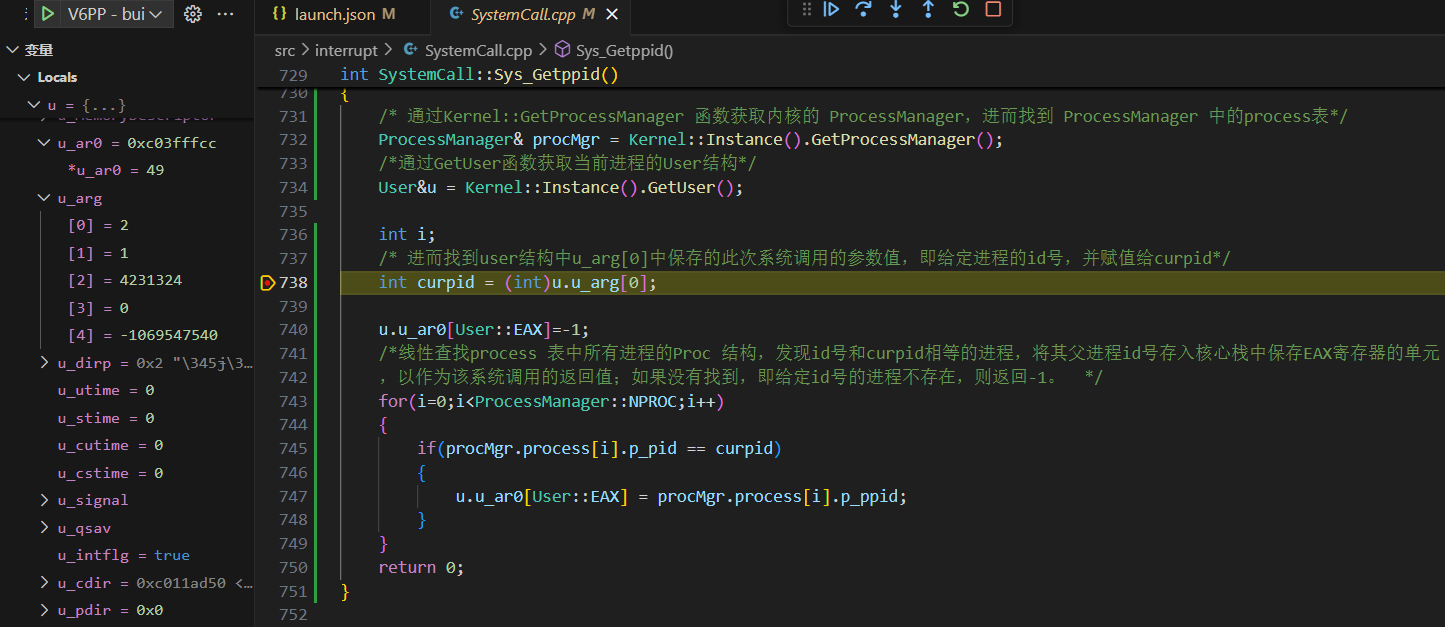


② 修改端点位置

1）在 Sys\_Getppid 函数的“int curpid=(int)u.u\_arg[0]” 赋值语句处添加断点，并进行调试

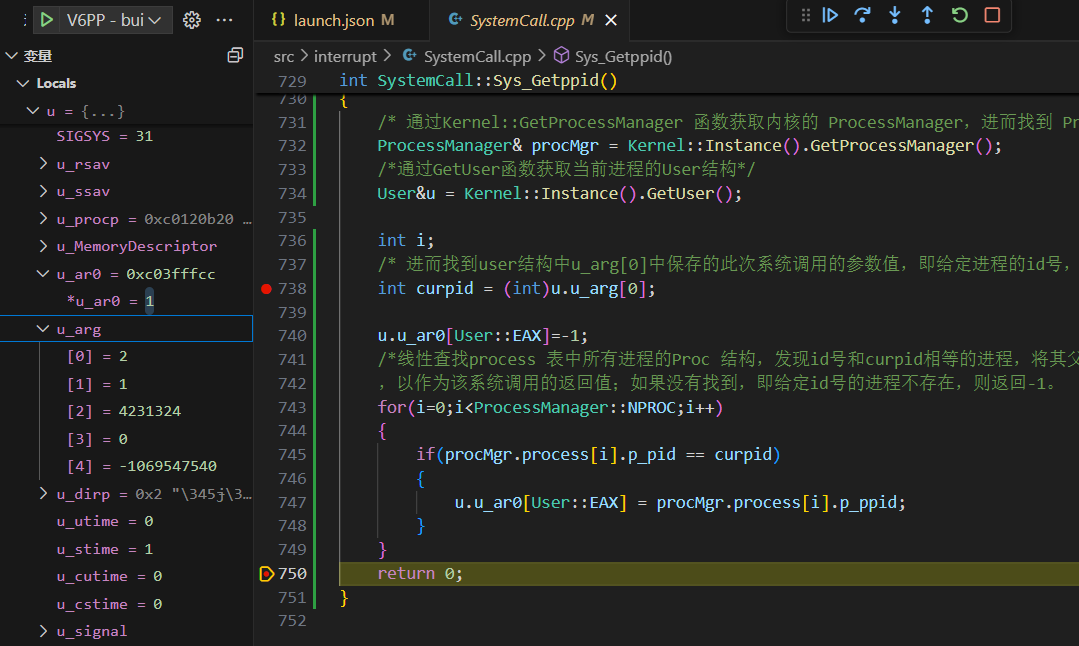




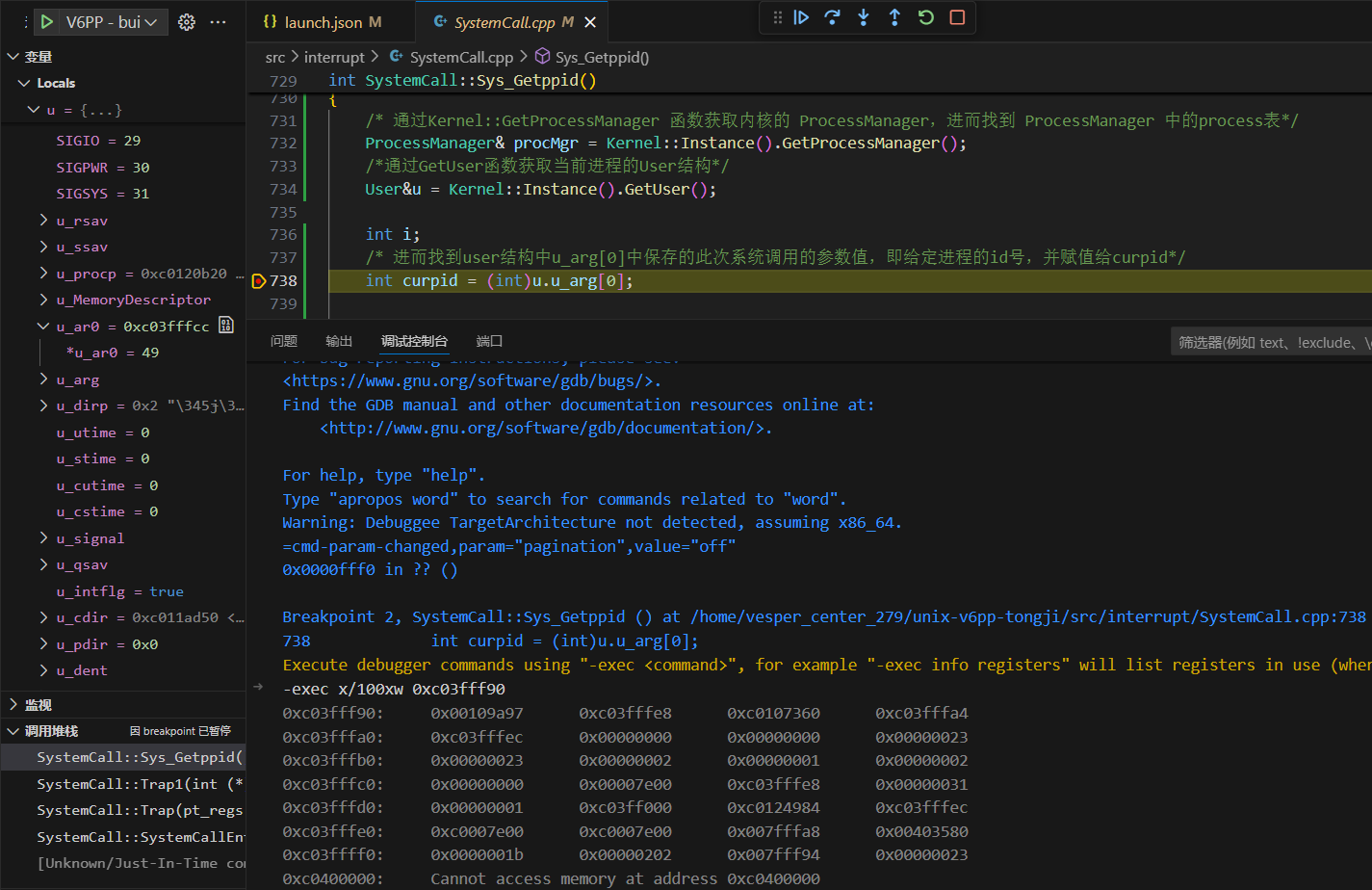


当程序停在该断点处时，系统调用已经开始执行。可以看到，此时u\_ar0 指向的核心栈中保存 EAX单元的值为49，说明系统调用号49已经通过系统调用的压栈操作由 EAX 寄存器带入到进程核心栈，u\_arg[0]处的值为2， 说明参数已进入进程的User结构。

2）在 Sys\_Getppid 函数的“return 0；” 返回语句处添加断点，并继续调试



当程序执行到该断点时，可以看到，u\_ar0指向核心栈中保存EAX的位置的值变为1，即返回值已经被保存到核心栈中用于保存EAX寄存器值的单元。



|  |  |  |  |
| --- | --- | --- | --- |
| 地址 | 核心栈内容 | 说明 | 解释 |
| 0xC03FFF90 | 0x00109A97 | 局部变量 |  |
| 0xC03FFF94 | 0xC03FFFE8 | OLD EBP | 指向系统调用入口程序栈帧的EBP |
| 0xC03FFF98 | 0xC0107360 | 返回地址 | 返回系统调用入口程序的地址 |
| 0xC03FFF9C | 0xC03FFFA4 | 软件现场指针 | 指向软件现场GS |
| 0xC03FFFA0 | 0xC03FFFEC | 硬件现场指针 | 指向硬件现场EIP |
| 0xC03FFFA4 | 0x00000000 | GS |  |
| 0xC03FFFA8 | 0x00000000 | FS |  |
| 0xC03FFFAC | 0x00000023 | DS |  |
| 0xC03FFFB0 | 0x00000023 | ES |  |
| 0xC03FFFB4 | 0x00000002 | EBX | 系统调用处理程序所需要的参数pid |
| 0xC03FFFB8 | 0x00000001 | ECX |  |
| 0xC03FFFBC | 0x00000002 | EDX |  |
| 0xC03FFFC0 | 0x00000000 | ESI |  |
| 0xC03FFFC4 | 0x00007E00 | EDI |  |
| 0xC03FFFC8 | 0xC03FFFE8 | EBP | 指向系统调用入口程序栈帧的EBP |
| 0xC03FFFCC | 0x00000031 | EAX | 保存EAX单元，值为系统调用号 |
| 0xC03FFFD0 | 0x00000001 | 局部变量 |  |
| 0xC03FFFD4 | 0xC03FF000 | 局部变量 |  |
| 0xC03FFFD8 | 0xC0124984 | 局部变量 |  |
| 0xC03FFFDC | 0xC03FFFEC | 局部变量 |  |
| 0xC03FFFE0 | 0xC0007E00 | 局部变量 |  |
| 0xC03FFFE4 | 0xC0007E00 | 局部变量 |  |
| 0xC03FFFE8 | 0x007FFFA8 | OLD EBP | 指向进行系统调用的用户态进程的EBP |
| 0xC03FFFEC | 0x00403580 | EIP | 从INT 0x80返回后将要执行的下一条指令 |
| 0xC03FFFF0 | 0x0000001B | CS | 末2位为1，之前为用户态 |
| 0xC03FFFF4 | 0x00000202 | EFLAGS |  |
| 0xC03FFFF8 | 0x007FFF94 | ESP | 用户栈栈顶指针 |
| 0xC03FFFFC | 0x00000023 | SS | 末2位为1，之前为用户态 |

经检验，该表格与图8对比正确