实验四：在 UNIX V6++中添加新的系统调用

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

在 SystemCall.cpp 中找到给系统调用子程序入口表 m_SystemEntranceTable 赋值的程序代码（见图 1），可选择其中任何一个赋值为{ 0, &Sys_Nosys }的项（表示对应的系统调用目前未定义，为空项）来添加新的系统调用。例如，这里我们选择第 49 项，并用{ 1, &Sys_Getppid}来替换原来的{ 0, &Sys_Nosys }，即：第 49 号系统调用所需参数为 1 个，系统调用处理子程序的入口地址为：&Sys_Getppid。

这里加入的子程序的名字是 Sys_Getppid，因为我们后续实现的该系统调用的功能为返 回指定进程的父进程的 ID 号。读者可以根据自己的想法取名，但建议最好和实现的具体的系统调用功能相关，以便于理解。

4.1.2. 在 SystemCall 类中添加新的系统调用处理子程序

首先，在 SystemCall.h 文件中添加该系统调用处理子程序 Sys_Getppid 的声明，如图 2 所示。建议按顺序添加，并写好注释。

其次，在 SystemCall.cpp 中添加 Sys_Getppid 的定义，如图 3 所示。这里同样建议按顺序添加，并写好注释。

Sys_Getppid 函数完成的功能是根据给定的进程 id 的值，返回该进程 的父进程，代码如图 3 所示，实现步骤包括：

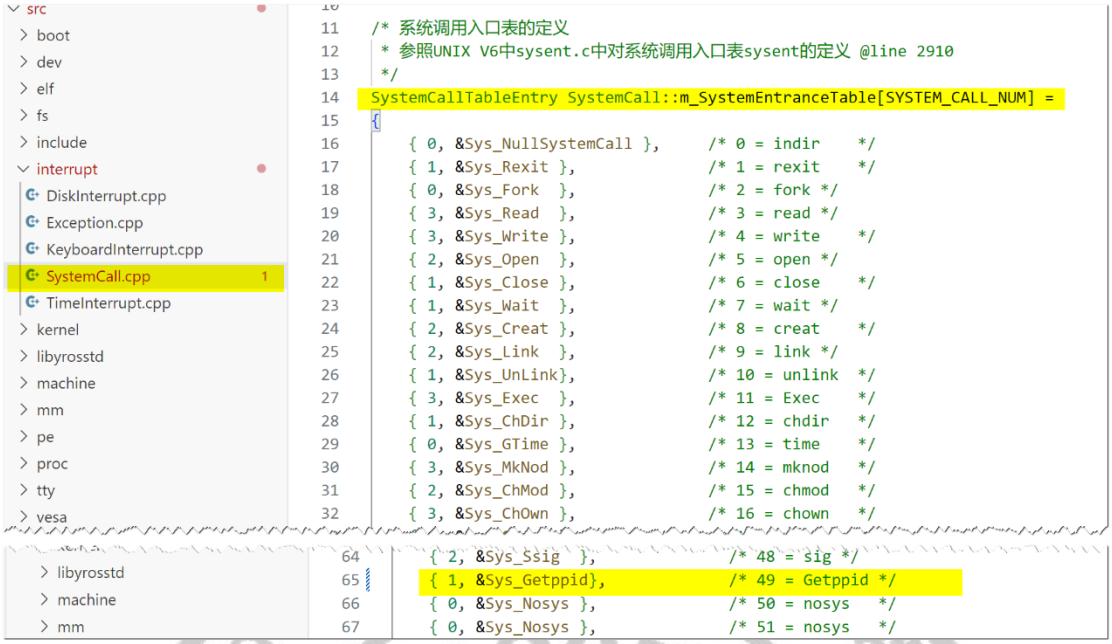

图 1：在系统调用子程序入口表 m_SystemEntranceTable 中添加新的入口

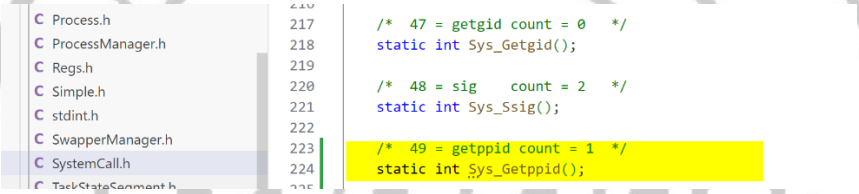

图 2：添加系统调用处理子程序声明

（1）通过 Kernel::GetUser 函数获取当前进程的 User 结构（详见实验三），进而找到 User 结构中 u_arg[0]保存的此次系统调用的参数值，即给定进程的 id 号，并赋值给 curpid；

（2）通过 Kernel::GetProcessManager 函数获取内核的 ProcessManager，进而找到 ProcessManager 中的 process 表；

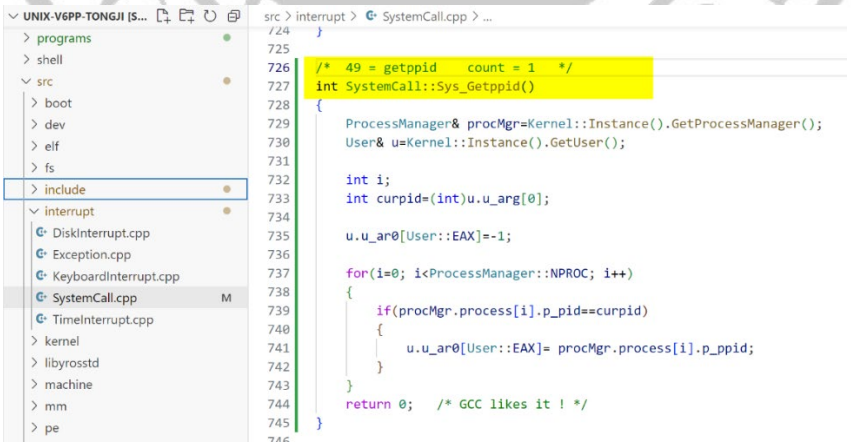

图 3：添加系统调用处理子程序定义

（3）线性查找 process 表中所有进程的 Proc 结构，发现 id 号和 curpid 相等的进程，将其父进程 id 号存入核心栈中保存 EAX 寄存器的单元，以作为该系统调用的返回值；如果没有找到，即给定 id 号的进程不存在，则返回-1。

4.2. 为新的系统调用添加对应的库函数

我们知道，任何一个系统调用，为了用户态程序使用方便，都必须有一个对应的用户态的库函数。UNIX V6++中，所有的库函数的声明在文件/lib/include/sys.h 中，而所有库函数的定义在文件 src/lib/src/sys.c 中。这里，我们完成与 Sys_Getppid 系统调用对应的库函数的添加工作。

4.2.1. 在 SYS.H 文件中添加库函数的声明

找到 sys.h 文件，在其中加入名为 getppid 的库函数的声明（如图 4 所示）。这个名字可以根据读者的喜好任意命名，这里强烈建议和定义的系统调用的名字相同，便于理解和使用。

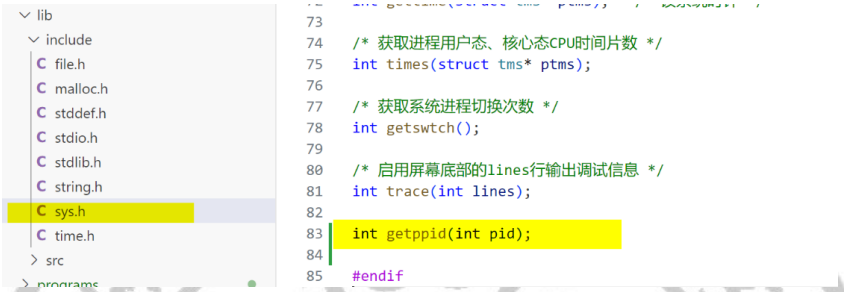

图 4：添加新的系统调用对应的库函数的声明

4.2.2. 在 SYS.C 中添加库函数的定义

在 sys.c 文件中添加库函数 getppid 的定义图 5 所示。这里需要特别注意的是系统调用号的设置。在我们的例子里这里设为 49，读者需要根据自己定义的系统调用在子程序入口表中的实际位置，填入正确的系统调用号。

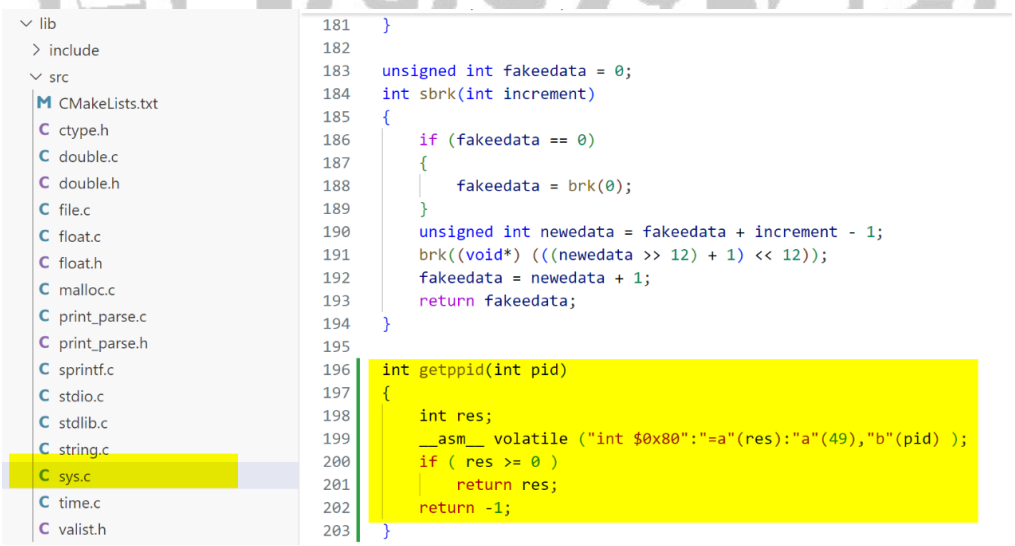

图 5：添加新的系统调用对应的库函数的定义

至此，可重新编译 UNIX V6++。如果编译成功，则一个新的系统调用及和它对应的库函数已添加完毕。

4.3. 编写测试程序

这里，我们可以尝试编写一个简单的测试程序来测试添加的新的系统调用能否正常工作。如何在 UNIX V6++中添加一个可执行程序的方法在实验二中已经用到，这里不再赘述。以图 6 中的代码为例，建

立可执行程序 getppid.exe。代码完成的功能是：通过调用 getppid 库函数，在屏幕输出当前进程父进程的 ID 号。

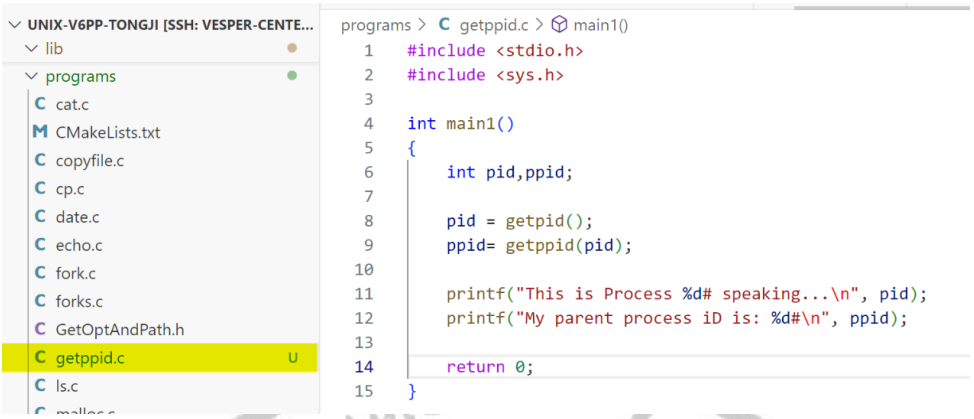

图 6：编写测试程序

在运行模式下启动 UNIX V6++，观察程序的输出是否正确。如图 7 所示。

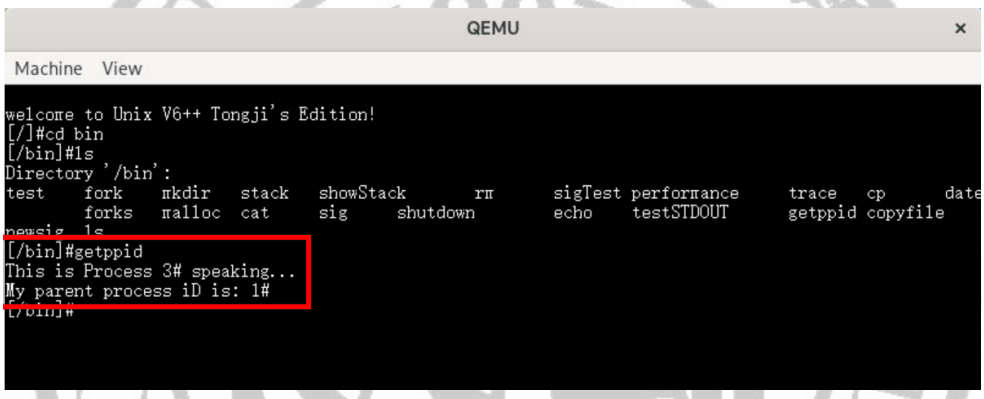

图 7：测试程序运行结果

4.4. 调试程序

4.4.1. 观察系统调用参数和返回值的传递

在开始调试程序之前，请读者回忆在前面的实验中，如何根据调试内核还是调试应用程序的不同需要设置正确的调试目标。

首先，在应用程序的调试目标下，我们将断点设置在库函数 getppid 中的语句：

```
__asm__ volatile ("int $0x80":"=a"(res):"a"(49),"b"(pid));
```

处（如图 8 所示），待程序停在此处时，在汇编指令“int $0x80”处增加一个断点并让程序运行到这里。此时可以看到此时，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。

接下来，我们将调试目标修改为内核调试，可以在 Sys_Getppid 函数的“int curpid=(int)u.u_arg[0]”赋值语句处添加断点，并重启一次调试，以观察系统调用发生后程序的执行状态。如图 9 所示。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。

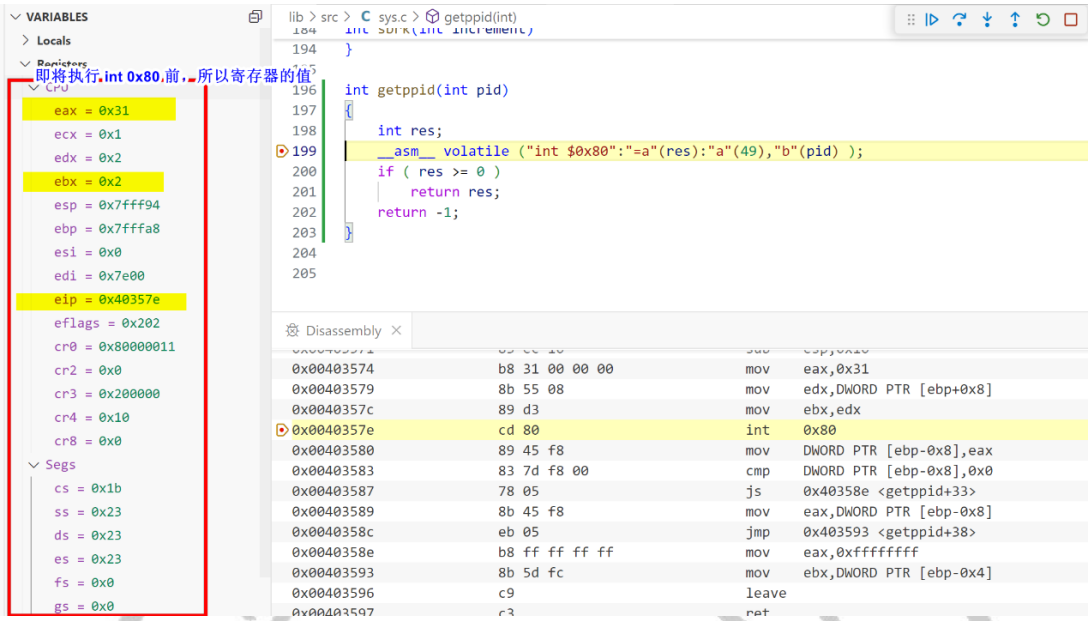

图 8：系统调用发生前

当执行到 Sys_Getppid 的最后一条语句时，可以看到此时 u_ar0 指向的核心栈中保存 EAX 单元的值变为 1，即返回值被存入核心栈中保存 EAX 的单元（如图 10 所示）。

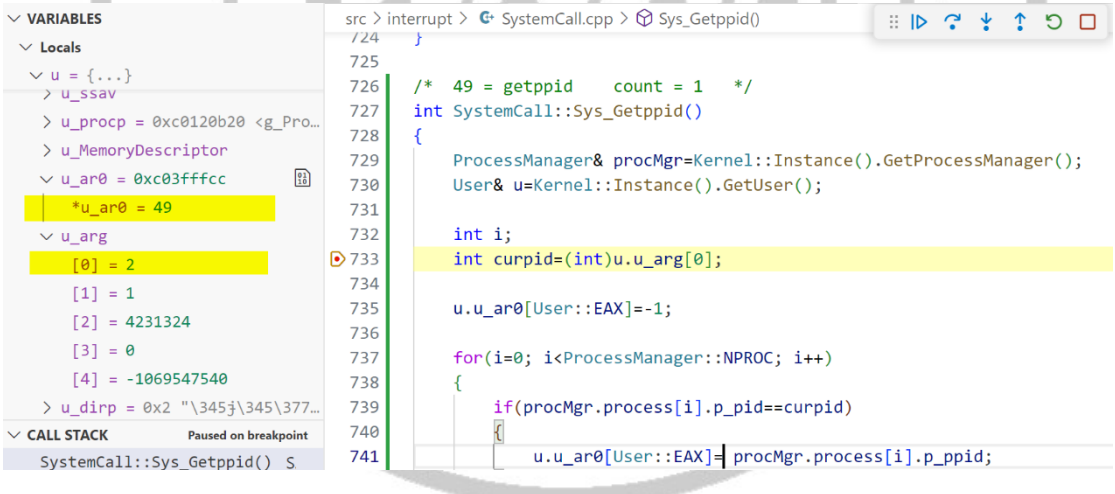

图 9：系统调用号进入核心栈

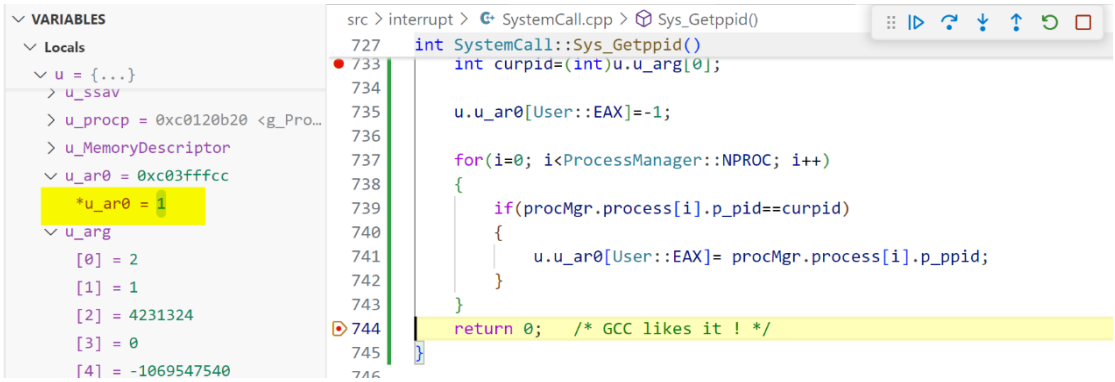

图 10：系统调用准备好返回值

4.4.2. 观察系统调用过程中核心栈的变化

从 u.u_ar0 的值 0xC03FFFCC 入手，我们可以通过查看内存单元的值恢复整个核心栈系统调用栈帧的全部内容。请读者通过设置合适的断点，并观察内存单元的值，将图 11 补充完整（阴影部分可忽略）。其中，压栈保存的寄存器的值，可以与图 8 中记录的进入核心态之前的寄存器的值验证。

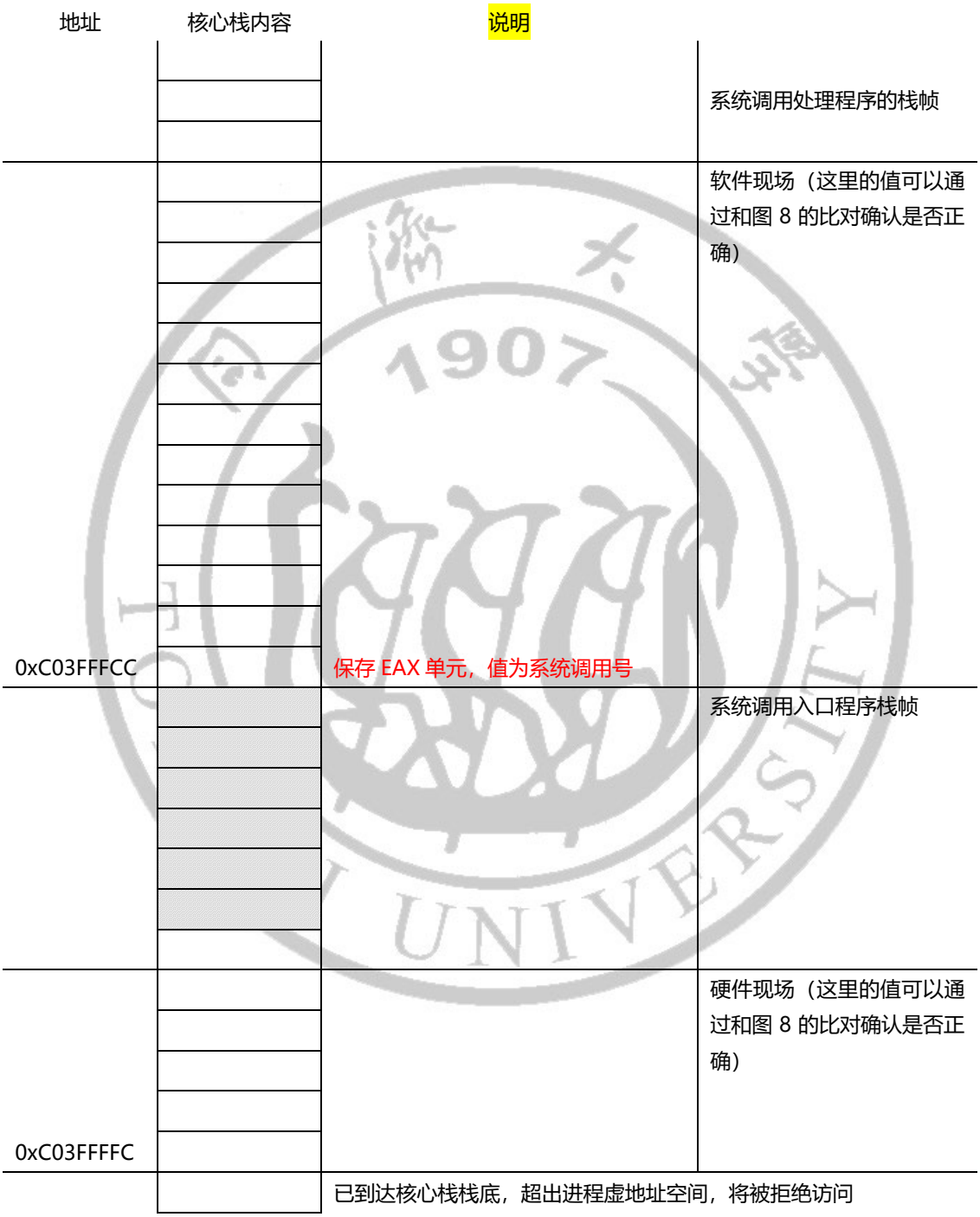

图 11：系统调用的核心栈

5. 实验报告要求

本次实验报告，需完成以下内容：

（1）（1 分）完成实验 4.1，截图说明操作过程，掌握在 UNIX V6++中添加一个新的系统调用的方法，并总结出主要步骤。

（2）（1 分）完成实验 4.2~4.3，掌握在 UNIX V6++中添加库函数的方法，并编写测试程序，通过运行说明添加的系统调用的正确，截图说明主要操作步骤。

（3）（2 分）完成实验 4.4，编写测试程序，通过设置合适的断点和观察内存，补充完成图 11，截图说明主要的调试过程和关键结果。