实验二：在 UNIX V6++中添加新的系统调用

1. 实验目的

（1）通过阅读汇编程序，观察 C++程序运行时栈、寄存器和内存变量的变化，理解 C/C++编译器对函数调用的处理过程。

（2）结合课程所学知识，通过在 UNIX V6++源代码中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。

（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 UNIX V6++运行和调试环境的 PC 机一台。

3. 预备知识

（1）C/C++编译器对函数调用的处理和栈帧的构成。

（2）UNIX V6++中系统调用的执行过程；

（3）UNIX V6++中所有和系统调用相关的代码模块。

4. 实验内容

4.1. 程序运行时环境：栈、寄存器和内存变量

通过编写一个简单的 C 语言程序调用，观察汇编代码对堆栈的修改。具体过程如下。

（1）新建一个 C 工程

在 eclipse 中新建一个工程（图 1，图 2），在该工程中添加一个.c 源文件（图 3，图 4），并在该源文件中如图 5 所示的源代码。

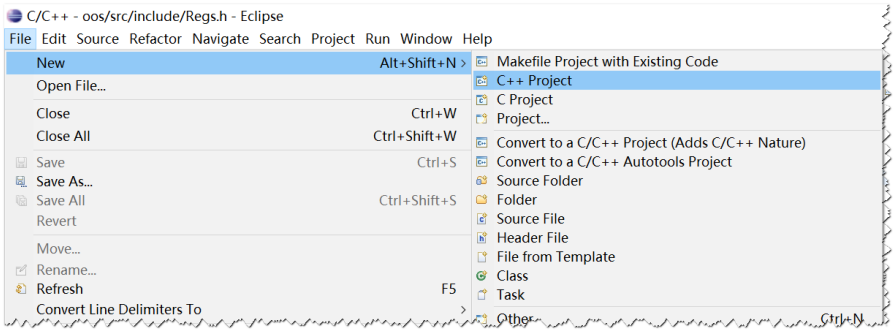


图 1

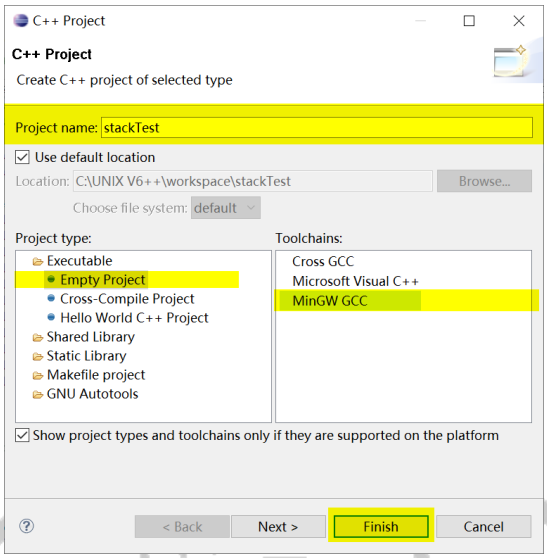

C++ Project
C++ Project
Create C++ project of selected type
Project name: stackTest
Use default location
Location: C:\UNIX V6++\workspace\stackTest
Browse...
Choose file system: default
Project type:
Executable
Empty Project
Cross-Compile Project
Hello World C++ Project
Shared Library
Static Library
Makefile project
GNU Autotools
Toolchains:
Cross GCC
Microsoft Visual C++
MinGW GCC
Show project types and toolchains only if they are supported on the platform
< Back
Next >
Finish
Cancel

图 2

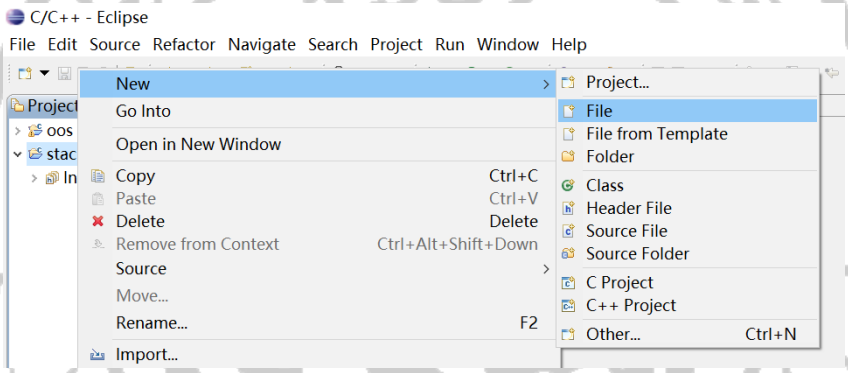

C/C++ - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
New
Go Into
Open in New Window
Copy Ctrl+C
Paste Ctrl+V
Delete Delete
Remove from Context Ctrl+Alt+Shift+Down
Source
Move...
Rename... F2
Import...
Project...
File
File from Template
Folder
Class
Header File
Source File
Source Folder
C Project
C++ Project
Other... Ctrl+N

图 3

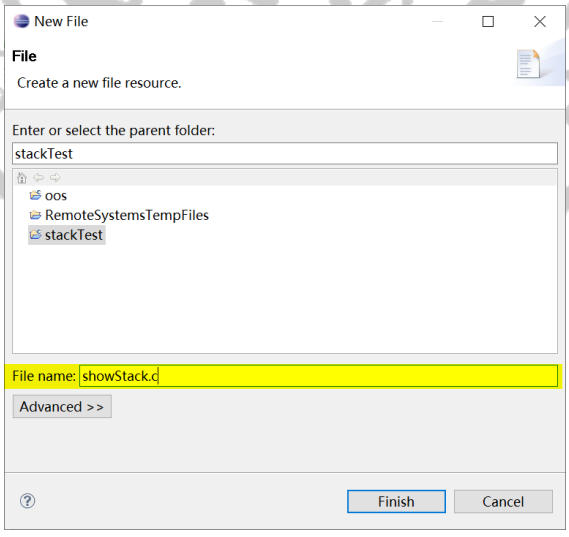

New File
File
Create a new file resource.
Enter or select the parent folder:
stackTest
oos
RemoteSystemsTempFiles
stackTest
File name: showStack.c
Advanced >>
Finish
Cancel

图 4

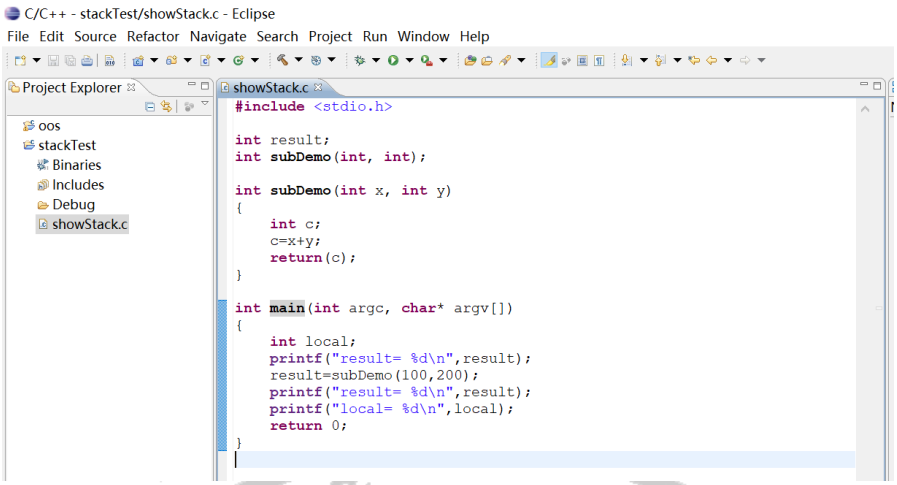


图 5

（2）编译与调试运行

编译该工程。打开 Debug 视图、反汇编视图、Variable 视图和 Register 视图。表里没有的，可以到 Others 里面去找（图 6）。

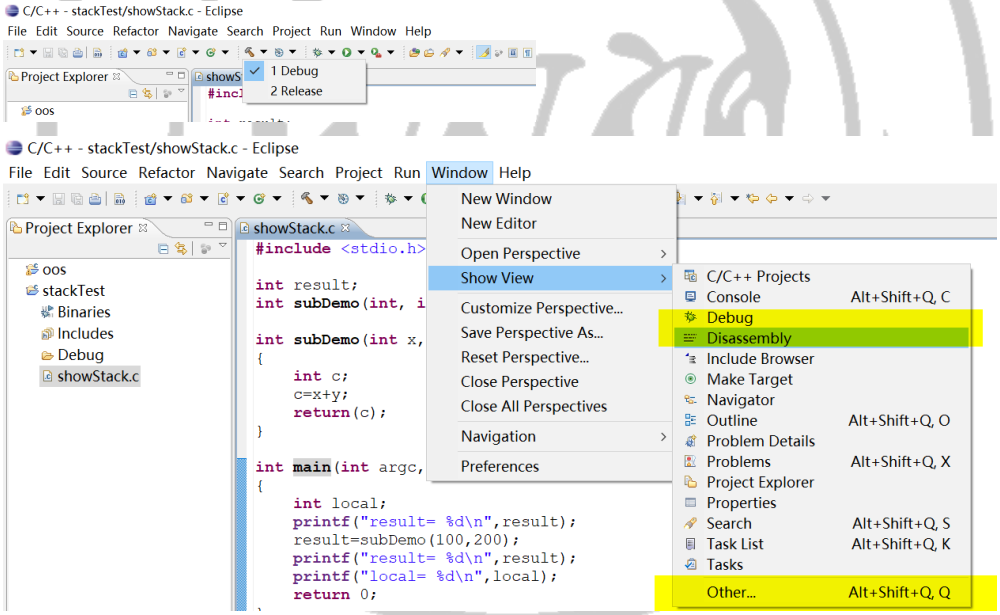


图 6

下断点（比如 15 行）之后开始调试。可以看到 subDemo 函数的汇编指令和所有寄存器的当前值。按 Step into，观察程序运行时，EIP、ESP、EBP 和 EAX 的变化。

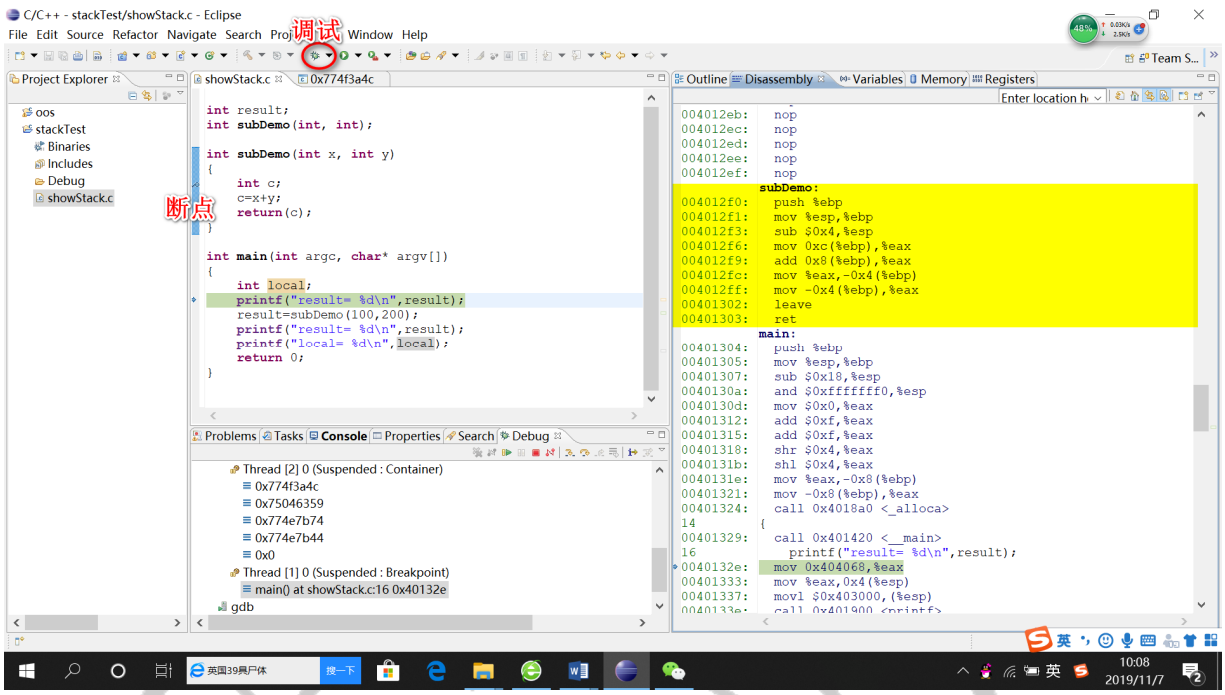

C/C++ - stackTest/showStack.c - Eclipse
调试
断点
Project Explorer
showStack.c
0x774f3a4c
Outline
Disassembly
Variables
Memory
Registers
Problems
Tasks
Console
Properties
Search
Debug
Thread [2] 0 (Suspended : Container)
Thread [1] 0 (Suspended : Breakpoint)
main() at showStack.c:16 0x40132e
gdb

图 7

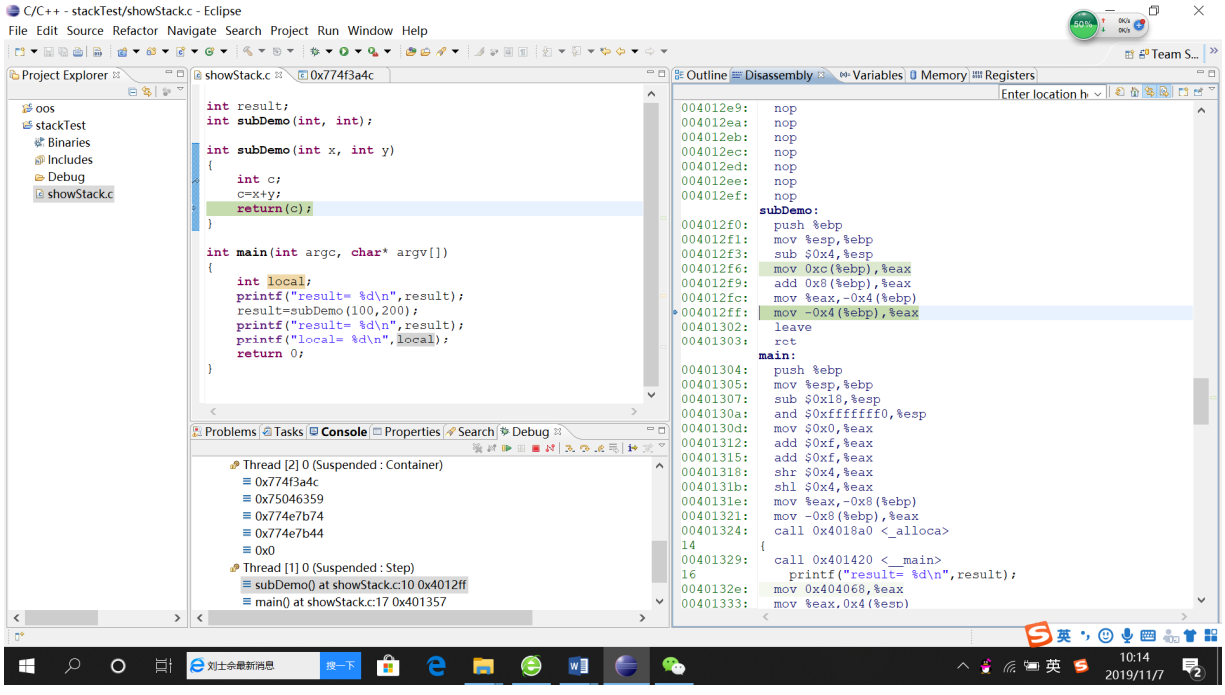

C/C++ - stackTest/showStack.c - Eclipse
Project Explorer
showStack.c
0x774f3a4c
Outline
Disassembly
Variables
Memory
Registers
Problems
Tasks
Console
Properties
Search
Debug
Thread [2] 0 (Suspended : Container)
Thread [1] 0 (Suspended : Step)
subDemo() at showStack.c:10 0x4012ff
main() at showStack.c:17 0x401357

图 8

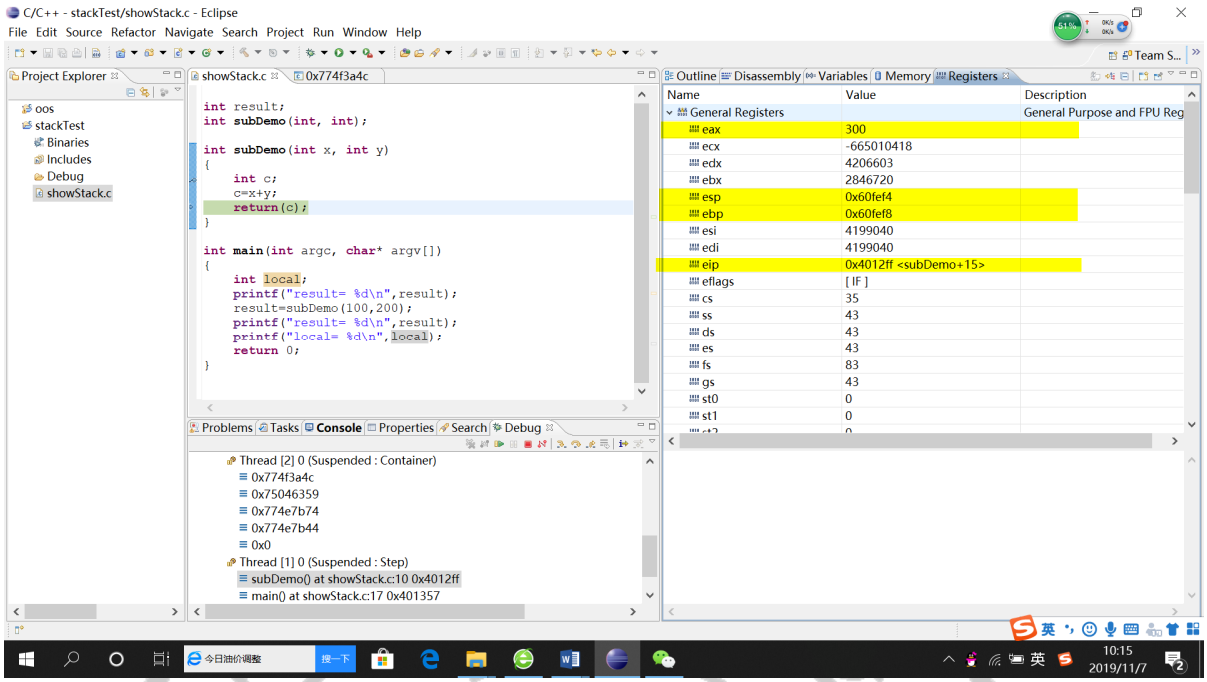


图 9

4.2. 在 UNIX V6++中添加一个新的系统调用接口

（1）在 SystemCall 类中添加系统调用处理子程序的定义

首先，在 SystemCall.h 文件中添加一个新的系统调用处理子程序的声明，如图 10 所示。

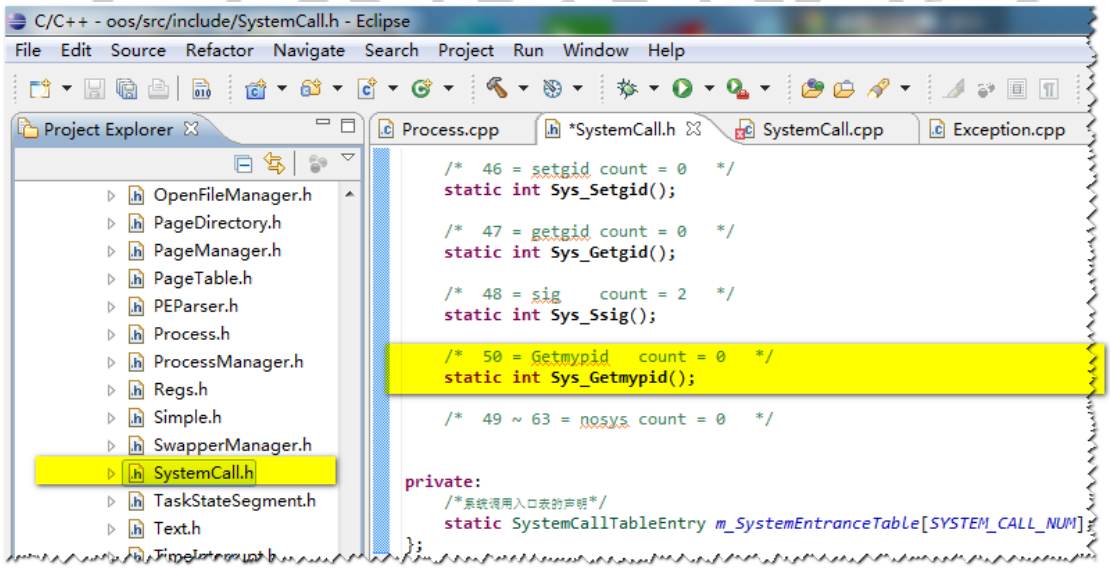


图 10：在 SystemCall.h 中添加新的系统调用处理子程序的声明

这里加入的子程序的名字是 Sys_Getmypid，读者可以根据自己的想法取名，但最好和实现的具体的系统调用的功能相关，以便于理解。

其次，在 SystemCall.cpp 中添加 Sys_Getmypid 的定义，如图 11 所示。可见，这里我们添加的 Sys_Getmypid 函数和 Sys_Getpid 函数完成的功能是完全一样的，即：返回当前进程的进程号。读者可根据需要，为新的系统调用处理函数定义不同的操作。

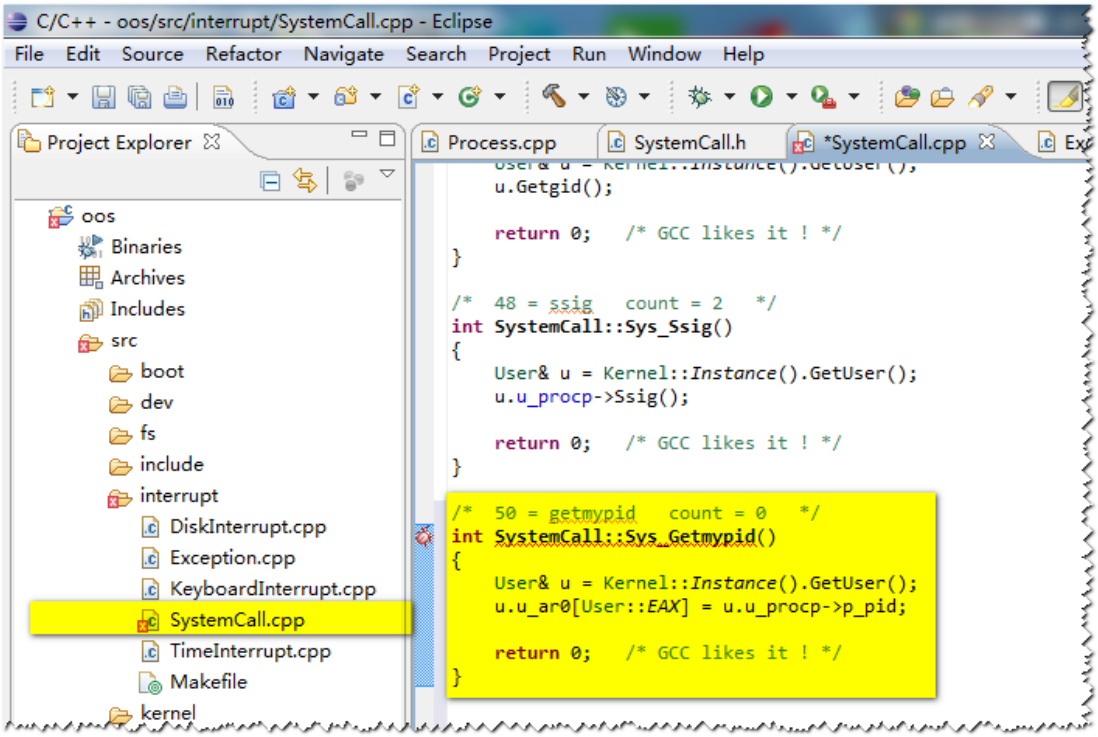


图 11：在 SystemCall.h 中添加新的系统调用处理子程序的声明

（2）在新的系统调用处理子程序加入系统调用子程序入口表中

在 SystemCall.cpp 中找到对系统调用子程序入口表 m_SystemEntranceTable 赋值的一段程序代码，如下：

```
SystemCallTableEntry SystemCall::m_SystemEntranceTable[SYSTEM_CALL_NUM] = ……
```

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 50 项，并用{ 0, &Sys_Getmypid}来替换原来的{ 0, &Sys_Nosys }，即：第 50 号系统调用所需参数为 0 个，系统调用处理子程序的入口地址为：&Sys_Getmypid。

上述流程参见图 12 所示。

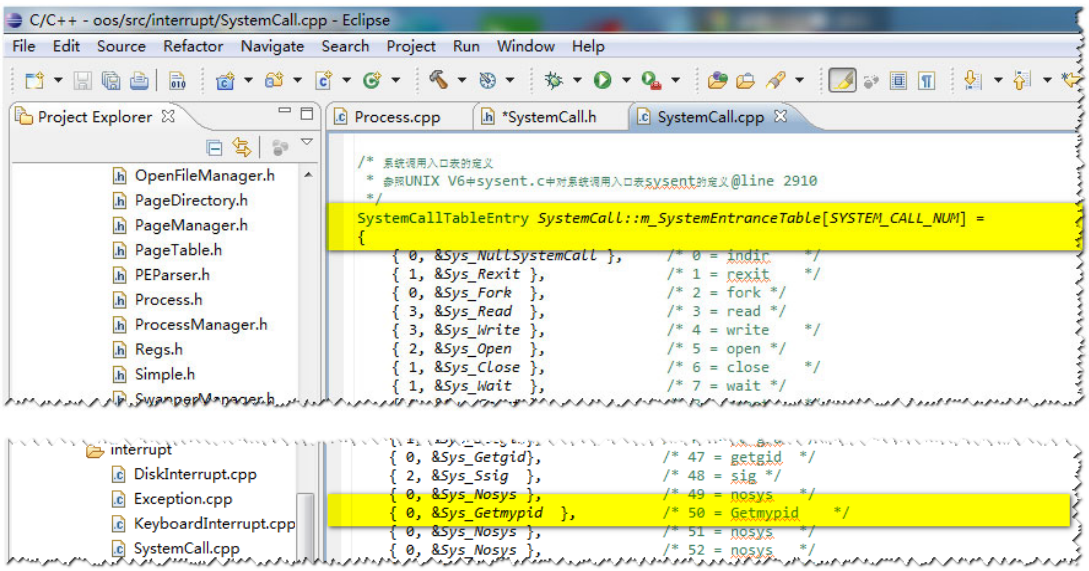


图 12：将新定义的系统调用处理程序加入处理程序入口表中

4.3. 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与之对应。UNIX V6++中，所有的库函数的声明在文件 src/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_Getmypid 系统调用对应的库函数的添加工作。

（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 getmypid 的库函数的声明（如图 13 所示）。这个名字可以根据读者的喜好任意命名，但是强烈建议和定义的系统调用的名字相同，便于理解和使用。

（2）在 sys.c 中添加库函数的定义

在 sys.c 文件中添加库函数 getmypid 的定义如下：

```
int getmypid()
{
    int res;
    __asm__ volatile ("int $0x80":"=a"(res):"a"(50));
    if ( res >= 0 )
        return res;
    return -1;
}
```

如图 14 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 50，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。

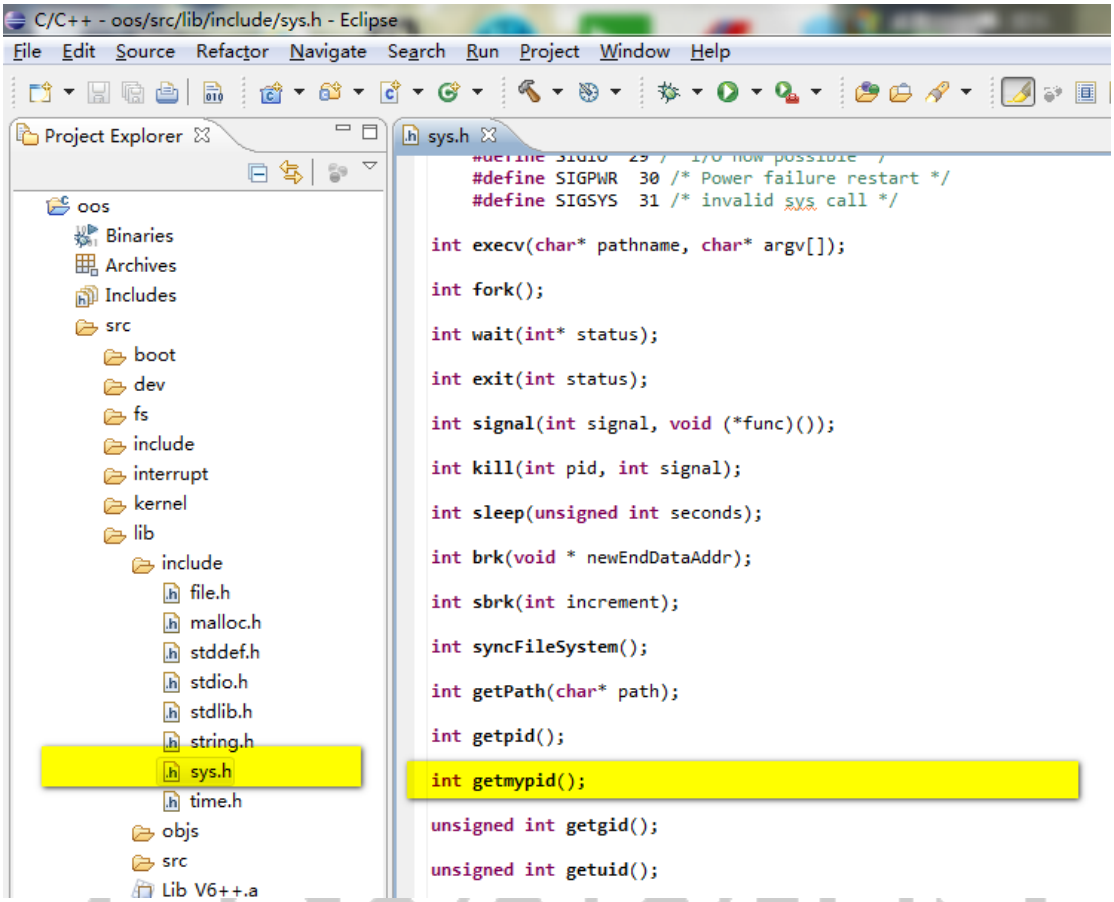


图 13：添加新的系统调用对应的库函数的声明

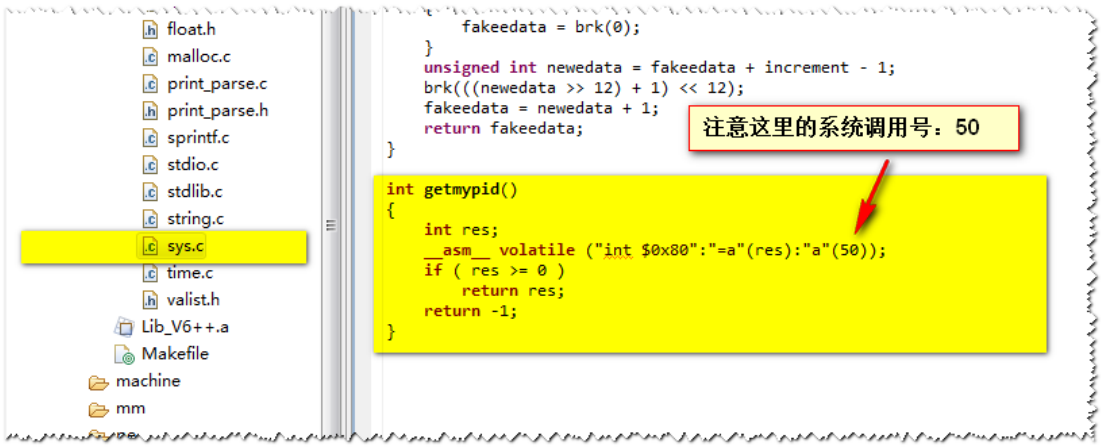


图 14：添加新的系统调用对应的库函数的定义

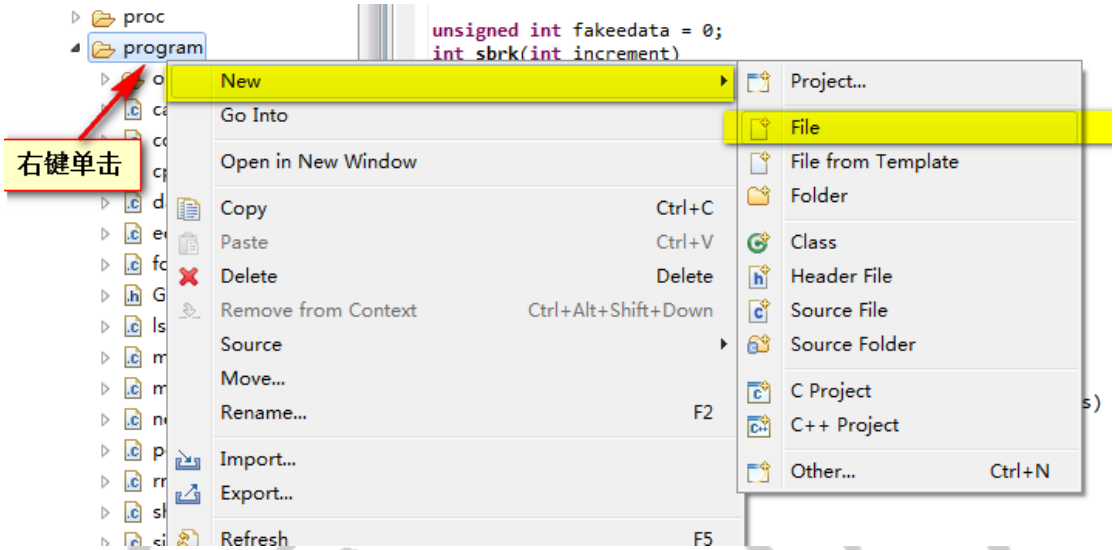
至此，一个新的系统调用及和它对应的库函数已添加完毕。

4.4. 编写测试程序

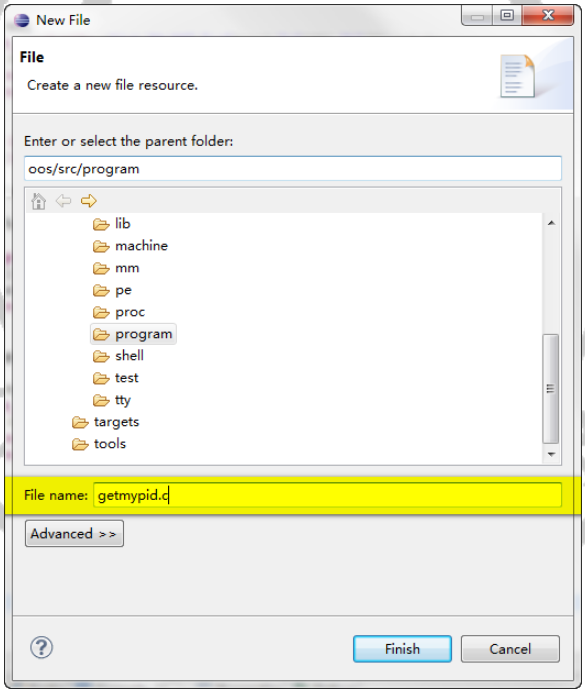
这里，我们可以尝试编写一个简单的测试程序来测试我们添加的新的系统调用能否正常工作。

在 UNIX V6++中添加一个可执行程序，需要在 src/program 文件夹下添加一个源程序文件，并编译通过之后，才可以运行。

（1）在 program 文件加入一个新的 c 语言文件



（a）



（b）

图 15：添加一个新的源文件

如图 15 所示，在 program 文件夹中加入一个新的文件，示例中命名为 getmypid.c。读者可根据自己定义的系统调用的功能进行命名。

（2）编辑该 c 语言源文件实现测试功能

在 getmypid.c 文件中加入如下的示例代码（如图 16 所示）：

```c
#include <stdio.h>
#include <sys.h>

int main1()
{
    int pid = getmypid();
    printf("My Process ID is: %d", pid);
    printf("\n");
    return 1;
}
```

代码完成的功能是：通过调用 getmypid 库函数，在屏幕输出当前进程的 ID 号。

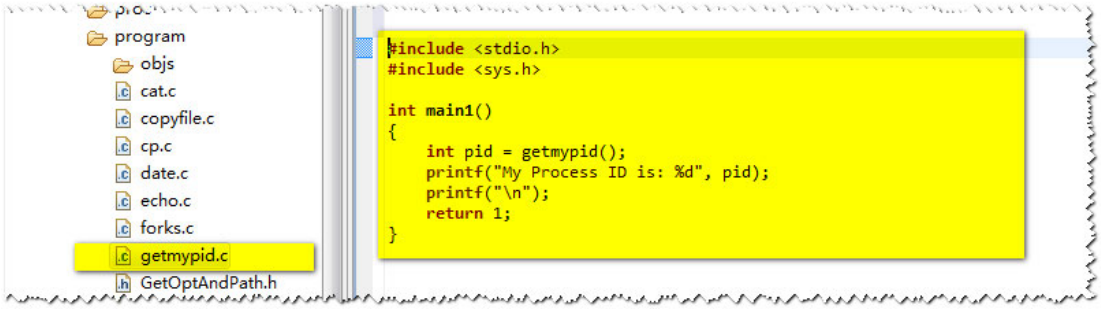
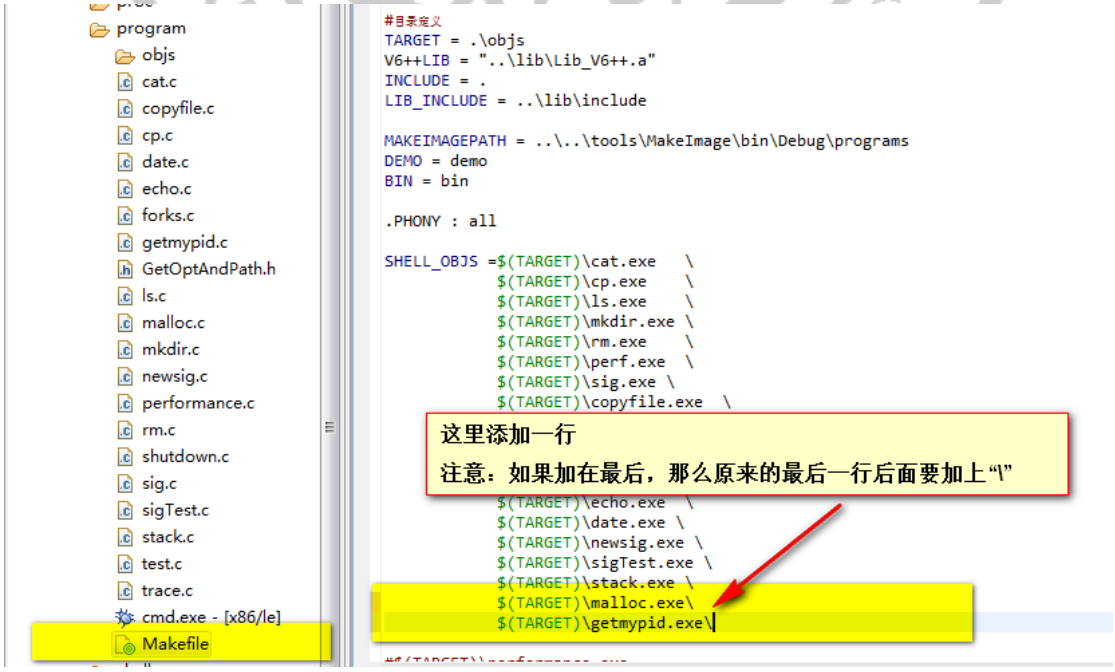


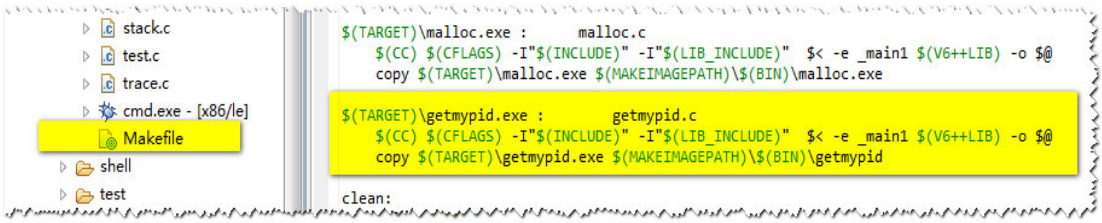
图 16：编辑测试程序源文件

（3）修改编译需要使用的 makefile 文件

为了完成上述测试程序源文件的编译，需要修改 program 文件夹下的 Makefile 文件。需要修改的两个地方分别如图 17 中的（a）和（b）所示。



(a)



(b)

图 17：对 program 文件夹下的 makefile 文件的修改

4.5. 重新编译 UNIX V6++代码

在 eclipse 中选择 project-Build All，完成对 UNIX V6++代码的重新编译。如图 18 所示。

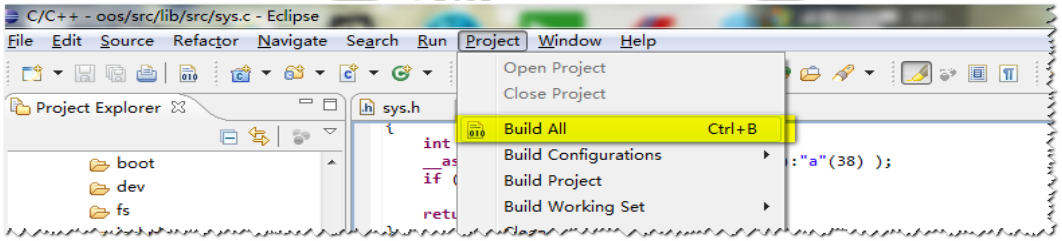


图 18：重新编译 UNIX V6++的源代码

如果编译成功，则启动 UNIX V6++之后，进入 bin 文件夹，可以看到该文件夹下有刚编译通过形成的可执行文件 getmypid。键入 getmypid 可以看到屏幕输出当前的进程号（如图 19 所示）。

图 19：测试程序运行结果

4.6. 程序调试

可以在程序运行过程中添加断点，通过调试观察程序的运行情况。例如：可以在如图 20 所示位置添加断点。

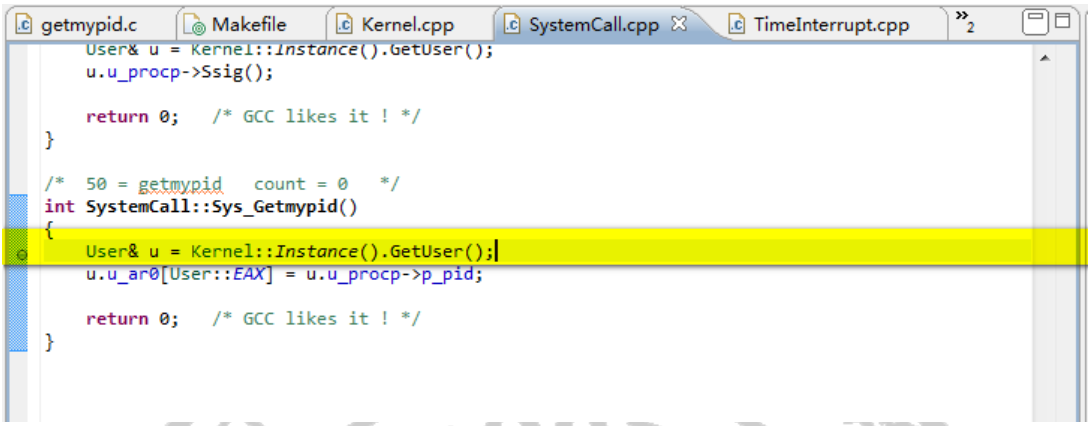


图 20：设置断点

断点设置好后，再次启动 UNIXV6++，在 eclipse 中启动调试，在 UNIX V6++中仍然键入如图 10 所示的命令，程序将在断点处停下来。这个时候，可以使程序单步执行，并观察程序中各个变量和寄存器的值。

5. 实验报告要求

（1）（1 分）完成实验 4.1，截图说明编译形成的维护栈帧的汇编指令的作用。

（2）（2 分）完成实验 4.2~4.5，按上述过程向 UNIX V6++中添加一个新的系统调用，要求实验报告中说明新的系统调用的功能和运行的效果（请选择需附代码和截图）。

（3）（1 分）完成实验 4.6，在调试环境下单步执行，观察 UNIX V6++的调试运行，截图说明变量与寄存器的值的变化，特别是 EAX 寄存器的值。