# detours 存储技术实验报告

章彦恺
计 22
2012011284

2013 年 5 月 21 日

# 目录

# 1 实验目的与要求

本次实验的目的主要是对本学期所学的存储技术基础有一个实践性的了解，以及了解计算机对磁盘信息读写的主要流程。此外由于实验要求基于 Microsoft 提供的开源开发库 detours，因而掌握如何利用 detours 将程序对文件系统的操作进行截获也是实验的目的之一。

实验对程序所实现的功能也有所要求。要求所实现的程序能够截获目标程序对文件系统的访问并对其进行透明的重定向，且重定向后目标程序读写正常，运行稳定。

而更具体的，我对本次实验的目标是将编写的动态链接库注入到系统的各个预设的程序中，并使其对特定位置的文件与目录的操作被重定向至另一个位置。即系统内的程序均能够被注入。

# 2 实验环境

这次实验主要是基于 windows 平台进行开发和测试的。程序开发的主要环境是在 Microsoft Visual Studio 2010 x86 下，使用 Windows 8 Enterprise 64bit Ent 操作系统。本次实验所设计到的所有程序均能够在该环境下编译通过。其中两个 C# 程序 test（工程名：test）和 launcher（工程名：settings）均依赖与运行库.Net framework 4.0。

实验测试在 Vmware 虚拟机模拟的 windows XP SP3 在 vc2010 运行库下以及.Net Framework 4.0 下执行。

此外实验报告的撰写在 Ubuntu 12.04 desktop i386 操作系统下，使用 CTex 与 pdflatex 编译生成。

# 3 实验过程与实现

本次实验为名为 detourist 解决方案下 5 个工程：

- detourist
- injector
- runTest
- settings
- test

本次实验主要通过以下几个步骤完成：

1. 学习并编译 detours 库
2. 在程序内调用 detours 函数并对本程序的文件操作进行截获
3. 编写注入目标程序所使用的 dll（即 detourist.dll 对应的工程 detourist）
4. 编写将 detourist.dll 注入到目标程序的控制程序（即 injector.exe 对应的工程 injector，并对 detourist.dll 进行测试
5. 编写图形化界面（launcher.exe 对应的工程 settings），对 injector.exe 进行可视化参数设置。
6. 编写 runTest（工程名：runFile）。runTest 对事先设定的目录调用简单的文件操作命令，并记录下执行时间，用以评估运行性能
7. 编写 test（工程名：test），作为控制 runTest 执行的程序，使运行性能的评估图形化。

程序整体的流程如图 3 所示：

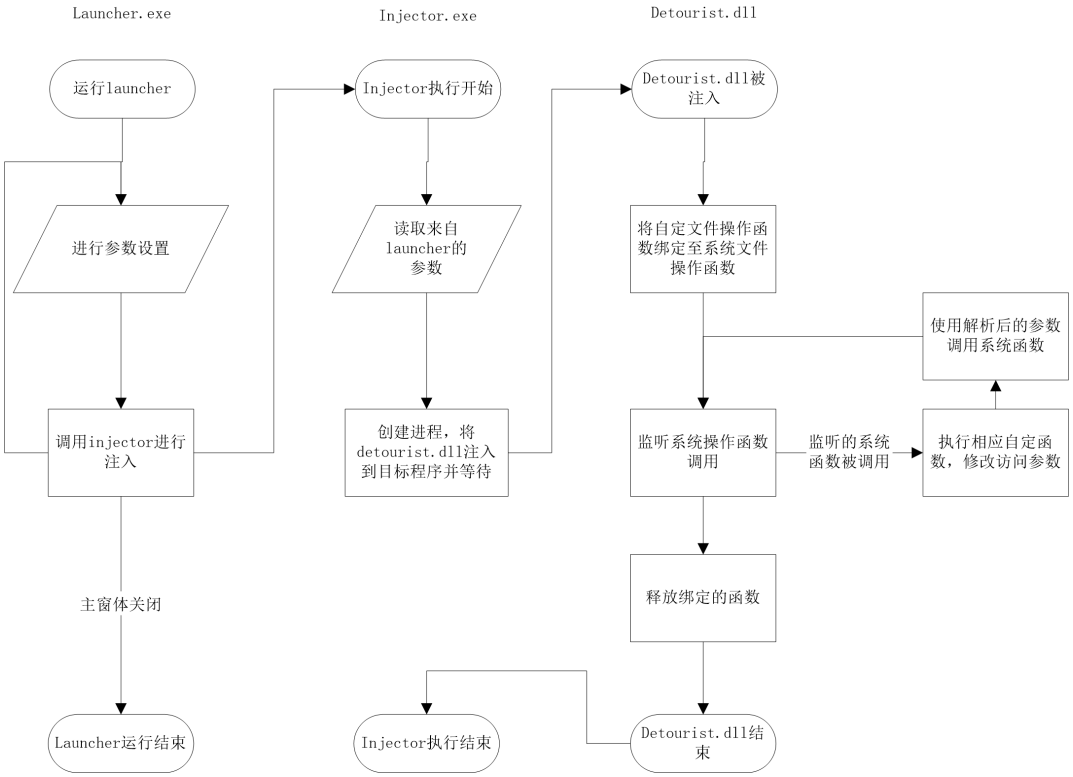



图 1: 实验的总体流程

接下来我将对实验的各个重要的步骤进行具体的记录描述。

## 3.1 学习并编译 detours

### 3.1.1 编译 detours

通过查阅相关资料我了解到，detours 是一个能够对任何 win32 函数进行操纵的类。它能够通过重写目标函数在内存中的代码，使得程序对系统函数的调用被重定向至一个用户级别的函数。

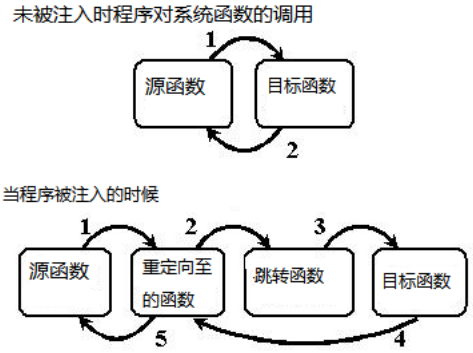



图 2: detours 的截获运行比对

由于 Windows SDK 的版本问题，本次实验中我所使用的 detours 版本是 detours 2.1[1]。在编译 detours 的过程中，我也遇到了许多问题。所幸的是，我所遇到的这些问题最终都在我的不断尝试中得到了解决。

在编译的过程中遇到的问题与解决方案如下表：

| 问题 | 原因 | 解决方案 |
| --- | --- | --- |
| cd src 时显示系统无法找到指定的目录 | 使用了 mingw 的 make | 在查看了 detours 文件夹下的 readme 文档后发现，编译 detours 所需要的 make 程序是 nmake。于是将 VC 目录下的 bin 文件夹加入到环境变量 PATH 中解决 |

[1]Microsoft Research Detours Package, Express Version 2.1 Build_216.

| | | |
|---|---|---|
| 找不到 windows.h | nmake 程序找不到 windows.h | 在 program files 文件夹下找到了 windows SDK 的位置，在其中的 include 资料夹中找到了 windows.h。于是在环境变量找哦还能够添加 include 项，其值设为 Windows SDK 下的 include 文件夹解决。 |
| 找不到 excpt.h | nmake 程序找不到 excpt.h | 在 VC 的 include 文件夹下找到了 excpt.h。于是将其加入到刚刚设置的 include 环境变量中解决。 |
| 在 sourceannotation.h(27) 中重定义了 size_t | 64 位的系统和 32 位的编译器提供的库冲突 | 这是我解决的问题中最棘手的问题。网上也没有类似的解决方案。从这个问题看似乎和系统的字长毫无关系，然而在仔细检查了错误信息之后发现编译过程中有一个警告：'Wp64' 选项被废弃。于是我发现自己所使用的编译器是 32 位版本。为了欺骗 detours 的 make，让其以 32 位模式进行编译，我修改了 common.mak 中的第 30 行，DETOURS_TARGET_PROCESSOR 由 X64 改为 X86，问题解决。 |
| 无法打开 ws2_32.lib | nmake 找不到 ws2_32.lib | 新建名为 lib 的环境变量，将 windows SDK 和 VC 的 lib 所在目录加入到环境变量中 |

经过编译，我得到了 detours.lib 和 detours.dll。再加上头文件 detours.h，这样 detours 库就已经能够被加入到我的程序中了。

### 3.1.2 学习 detours 使用

在成功对 detours 进行编译之后，通过上网查找资料，我了解了 detours 的基本用法。在动态链接库中使用 detours 进行注入主要分为以下几个步骤：

1. 创建于 WINAPI 相应函数的参数与返回值都相容的函数指针，并将这些指针赋为 WINAPI 函数。

2. 编写与 WINAPI 相应函数的参数与返回值都相容的函数，这些函数将在被注入的程序调用相应 WINAPI 时被调用。称这些函数为 detours 函数。

3. 当 dll 被接入的时候，进行 detours 的初始化工作。即利用 detours 库中的 attach 函数将原始 WINAPI 与 detours 函数相关联。

4. 利用 detours 函数等待截获的信息并对截获的信息进行处理。

5. 当 dll 被解除（detach）的时候，释放关联的 WINAPI 函数和 detours 函数。

在测试的过程中，我发现，如果声明定义的 WINAPI 函数指针的返回值与 WINAPI 原函数不相容，在编译过程中并不会产生错误，而在创建线程进行注入的时候，将会导致线程创建失败。

在了解了 detours 库的使用方法之后，我便开始了具体功能的实现，即 detourist.dll 的编写。

## 3.2 工程的开发与各模块功能

### 3.2.1 detourist.dll 的编写

detourist.dll 是整个工程中的两个调用 detours 库的程序中的一个。另一个程序是 detourist 的控制程序 injector.exe。而 detourist.dll 在被注入到目标程序中之后起到了直接捕获目标程序的系统函数调用并进行处理的功能。程序的执行流程如图 3 所示：

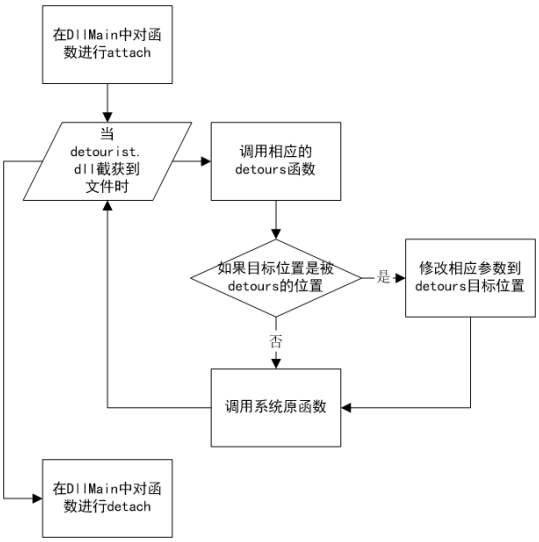



图 3: detourist.dll 流程图

detourist.dll 的代码实现分为 detourist.cpp，basic.h/cpp，func.h/cpp。

detourist.cpp 即 DLLMain 所在的 dll 入口，其中在 DllMain 中对 detours 函数进行了 attach 和 detach。通过 attach，使得 detourist.dll 能够拦截被注入程序的函数，并在 func.h/cpp 中对其进行操作。

func.h/cpp 中定义以及实现了所有的 29 个被重定向的系统操作函数。在 func 中所使用的基本功能均调用 basic.h/cpp 中的函数。

basic.h/cpp 中定义实现了 detourist 所需要的基本功能。

- 宽窄字符的互换
- 判断一个字符串是否是另一个字符串的前缀
- 进行日志、调试信息的输出。
- 设置、运行参数信息的读取
- 从文件句柄获得文件路径
- 判断位置是否是被 detours 的位置，如果是则将其修改为 detours 的目标位置

可以看到，对路径的解析工作在 basic 中完成。更细节的说，basic.cpp 中的 replace 函数在每一次截获的系统调用中都会被调用执行。在 replace 函数中检查该系统调用的相关目录是否在被 detours 的位置列表中。如果存

在一个条目，使得该条目中被 detours 位置是该路径的一个前缀，则将该前缀替换成相应的目标路径，来完成对该路径操作的重定向。

detourist.dll 几乎截获了除了 Transacted 函数以外的所有涉及到字符串（文件名或目录路径）的函数，共 29 个函数。具体如下

| 系统函数名 | 函数功能 |
|---|---|
| CreateFile | 创建（读取）文件 |
| ReadFile | 从一个句柄读取一个文件 |
| WriteFile | 从一个句柄向一个文件写入 |
| CreateDirectory | 创建文件夹 |
| CreateDirectoryA | CreateDirectory 的 ANSI 版本 |
| CreateDirectoryEx | CreateDirectory 的扩展 |
| GetCurrentDirectory | 获得当前工作目录 |
| GetCurrentDirectoryA | GetCurrentDirectory 的 ANSI 版本 |
| RemoveDirectory | 删除目录 |
| RemoveDirectoryA | RemoveDirectory 的 ANSI 版本 |
| SetCurrentDirectory | 设置当前工作目录 |
| SetCurrentDirectoryA | SetCurrentDirectory 的 ANSI 版本 |
| FindFirstFile | 得到某目录下的文件结构中的首节点 |
| FindFirstFileA | FindFirstFile 的 ANSI 版本 |
| FindFirstFileEx | FindFirstFile 的扩展版本 |
| FindFirstFileExA | FindFirstFileEx 的 ANSI 版本 |
| MoveFile | 移动一个文件 |
| MoveFileA | MoveFile 的 ANSI 版本 |
| MoveFileEx | MoveFile 的扩展 |
| MoveFileExA | MoveFileEx 的 ANSI 版本 |
| DeleteFile | 删除一个文件 |
| DeleteFileA | DeleteFile 的 ANSI 版本 |
| OpenFile | 打开一个文件（已经被系统很少或不再使用） |
| ReplaceFile | 覆盖一个文件 |
| ReplaceFileA | ReplaceFile 的 ANSI 版本 |
| CopyFile | 复制一个文件 |
| CopyFileA | CopyFile 的 ANSI 版本 |
| CopyFileEx | CopyFile 的扩展 |
| CopyFileExA | CopyFileEx 的 ANSI 版本 |

### 3.2.2 injector.exe 的编写

injector.exe 没有对被注入的程序进行具体的操作。它作为设置程序 launcher.exe 和 detourist.dll 之间的中介。injector 主要的作用是收到来自 launcher 的设置参数信息，然后根据这些信息来调用 DetourCreateProcessWithDll 函数，将 detorist.dll 注入到目标程序中实现 detour。

injector 实现的功能较为单一。其流程如下：

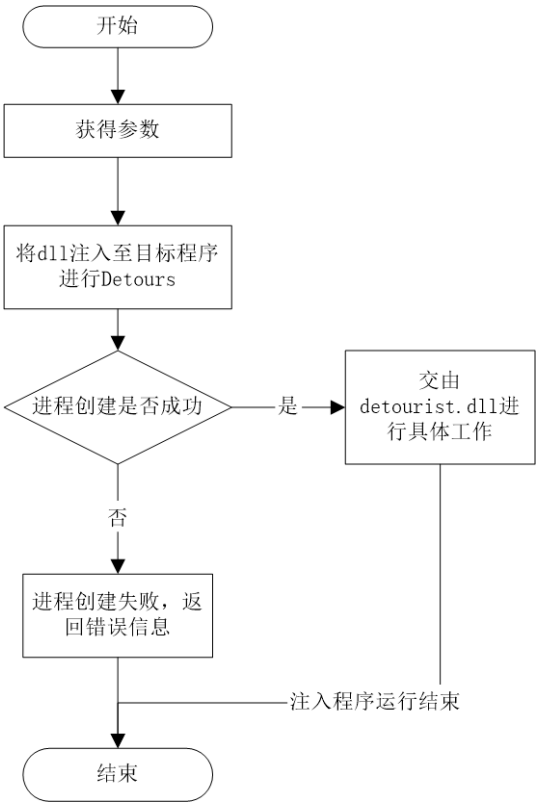



图 4: injector.exe 流程图

injector 工程中只有 injector.cpp 一个有效的源文件。它实现了 injector.exe 的所有功能。其他在 injector 工程下的 log.h/cpp，XUnzip.h/cpp 和 zip.h/cpp 均是在实验过程中进行尝试实现的部分，但最终没有集成到工程内成为有效代码。有关其详细的信息，参见 3.3.3 节。

### 3.2.3 settings 工程编写

setting 工程使用 C# 环境将对 injector.exe 和 detourist.dll 的参数进行图形化的设置。settings 只含有一个窗体 fmMain。功能也比较简单。其编译结果为 launcher.exe。

setting 的工作流程如下：

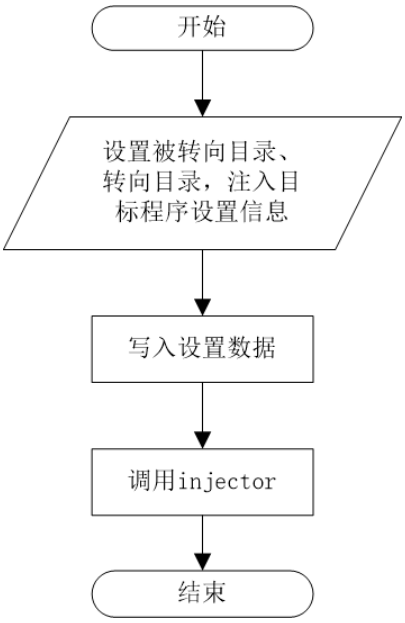



图 5: launcher.exe 运行流程

launcher 运行时的界面如图：

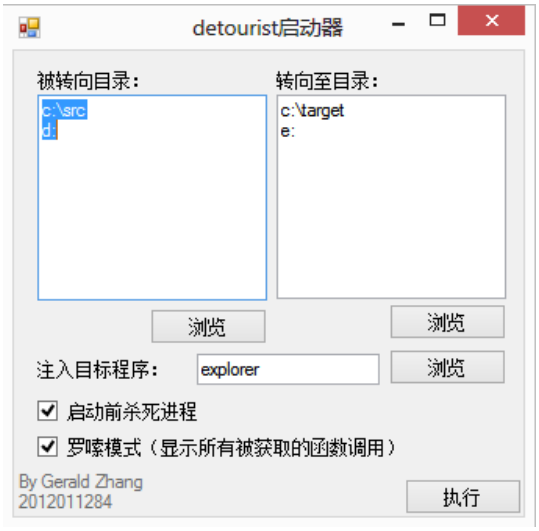



图 6: launcher.exe 的运行界面

可见，launcher 并没有有关与 detours 库的调用。它可对执行 detourist.dll 的工作模式进行图形化的设置。可设置的功能包括：

- 被转向的目录
- 转向至的目录
- 设置被注入的目标程序
- 是否在注入程序之前结束正在运行的原始进程。该设置常在注入 explorer 时被使用。
- 是否显示未被 detours 的目录文件操作的日志信息

程序要求只有在被转向的目录和转向至的目录个数相符的情况下，对 injector.exe 的执行操作才被允许。以及当关闭啰嗦模式时，只有以 $ 开始的调试信息和被有效 detours 的路径操作才会被显示出来。

### 3.2.4 测试模块的开发

测试模块包括 test 工程和 runTest 工程。test 由 C# 编写，是测试程序的设置界面，但由于由 test.exe 执行 runTest.exe 将会导致两个程序处于不同的进程空间内，将会到至对 runTest 的 detours 操作失败，runTest.exe 对文件目录的操作将无法被 detourist.dll 截获，因而随后 test 被弃置。

从 test.exe 的界面上能够直观地看到 runtest 所进行的测试细节。

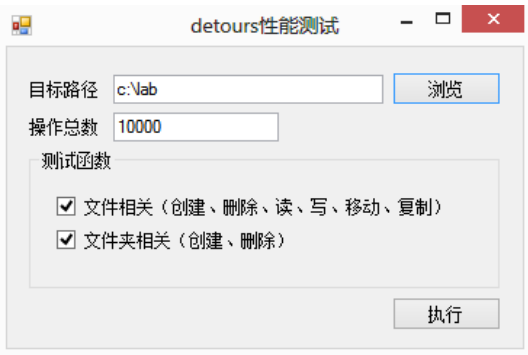



图 7: test.exe 的运行界面

可以看到，runTest 的测试分为两块，分别是文件的创建、删除、读写、移动和复制，以及目录的创建和删除。对于测试的细节，参见 4.2 节。

## 3.3 开发过程中遇到的困难

这次的实验对于大一学生来说是有一些挑战的。由于缺乏工程开发上的相关经验，在本次开发的过程中遇到了各种各样的困难。一些困难被成功地解决，但一些困难由于种种原因没有解决而在工程中最后 disable 了相应的实现。

### 3.3.1 日志反馈与调试

由于使用 dll 进行注入，因而在开发的初期，我无法对 detours.dll 的工作情况进行监控。因而程序开发的初期，只能通过查看代码的方式对程序进行调试。随后，随着代码长度的增加，我使用 windows 系统提供的 MessageBox 弹出窗口来监视程序的运行情况。然而使用这种方式将会导致我运行调试的效率低下。由于在注入 explorer 之前工程的注入目标程序是 windows7 系统的 notepad.exe。notepad 在程序运行的初始阶段会有大量的 CreateFile 操作。这样的日志和反馈方式将会导致大量的弹窗。

于是我尝试在 injector.exe 和 detourist.dll 之间建立本地 socket 网络连接，利用 127.0.0.1 回环地址使用 7001 端口进行通信。可以看到在 detourist 工程中仍旧保留着使用本地通信进行调试的残留代码，包括 initLog_far()、uninitLog_far() 和 log_far() 函数，以及在 injector 工程中的 log.h/cpp。

然而使用这种方式仍然有十分明显的弊端。首先连接的建立是令人不放心的。其次当调试信息的数量和产生的速度较为集中的时候，导致了一些调试信息的丢失问题。此外使用 winsock 的过程中还出现了没有 Unicode 函数的问题。

于是我上网查找了资料，尝试创建一个命令行，绑定 IO 流到命令行直接进行调试信息的读入和输出。使用在 detourist 工程中 basic.cpp 中实现的 RedirectIOToConsole 函数，调用 AllocConsole 等系统函数创建了一个命令行并将 stdin、stdout 和 stderr 绑定至命令行中。

这是一种十分稳定高效的调试方式，也成为了我理想的最终的测试方案。

### 3.3.2 ANSI 字符集和 Unicode 字符集

由于工程所涉及的函数绝大多数都是基于 windows API、winsock 等系统函数，且 detourist.dll 和 injector.exe 的开发环境为集成度较低的 C++ 环境，函数的字符集没有进行统一，因而在字符集的处理上出现了问题。

例如在使用 winsock 的 send 时，没有宽字符版本的函数。而在 detourist.dll 中，必须添加对宽字符版本的系统函数的支持。于是我尝试了各种宽、窄字符的转换方法。最终调用了 WideCharToMultiByte 和 MultiByteToWideChar 系统函数进行了转换。转换后我还处理了字符串的结束符问题。

字符集的不统一问题还提高了程序的代码复杂度。可以看到，在被重定向的 29 个函数中，有 12 个函数重定向了 Unicode 函数的 ANSI 版本。

### 3.3.3 增加文件解压缩功能

本次实验的本意是希望在注入 explorer 后，当 explorer 尝试打开一个 zip 压缩文件时，将其解压到一个设定的文件夹并将对压缩文件内容的访问重定向至已被解压缩的文件夹内。

在下载学习开源的 zlib 库的使用后，我成功地在程序中集成了 zip 文件的解压缩功能。可以在 detourist 工程的 zip.h/cpp 和 Xunzip.h/cpp 中看到。

然而，在注入到 windows XP 的 explorer 后，通过查看调试信息发现，当 explorer 尝试打开一个文件的时候，首先会寻找其打开方式，产生大量的 CreateFile 调用，然后使用 CreateFile 被打开的文件和 CreateFile 执行的应用程序来打开目标程序。OpenFile 函数在这个过程中并没有被调用。因而导致了创建文件和打开文件的命令难以加以区分，致使了这一个部分的实验内容失败。

### 3.3.4 其他困难

我在编译 detours 库的过程中遇到了如 3.1.1 节中提到的各种问题。这

些问题在我的各种尝试之后都得到了一一的解决，最终获得了 detours 库的编译结构 detours.lib 和 detours.dll。

程序原本期望当程序在注入到 explorer.exe 之后，能够使得所有对特定路径的操作都被重定向到一个目标的位置。然而 explorer 执行程序的方式是创建一个新的进程空间，使得新的进程空间内的程序不再能够被重定向。这样就导致了我的注入行为只能局限地对单一进程的行为进行截获。一个解决方法是截获创建子线程的调用，然后将 detourist.dll 注入到该子线程当中。这个方法的实现我将留予日后的学习探究中进行。

除去工程开发上遇到的问题外，在实验报告的书写上我也遇到了一些困难。由于报告的书写使用的是 latex 语言，其中 itemize 环境的默认缩进为段落首。我尝试了在环境内修改 leftmargin、设置全局参数等方法，最终通过查找资料，调用 enumitem 包通过设置 itemize 的参数最终成功设置了 itemize 环境的缩进。

# 4 实验结果

## 4.1 工程运行成果

程序能够成功地注入到 Windows XP 的 explorer.exe 中，使得 explorer 对指定路径的访问、文件的读写以及制定路径下文件的创建删除、目录的创建删除都被重定向至对应的目标目录中。其操作体验和在另一个目录中的操作体验完全相同。

如下，我使用了一系列运行时截图来展示运行结果。其中被定向目录为 $c:\backslash src$，目标目录为 $c:\backslash target$。

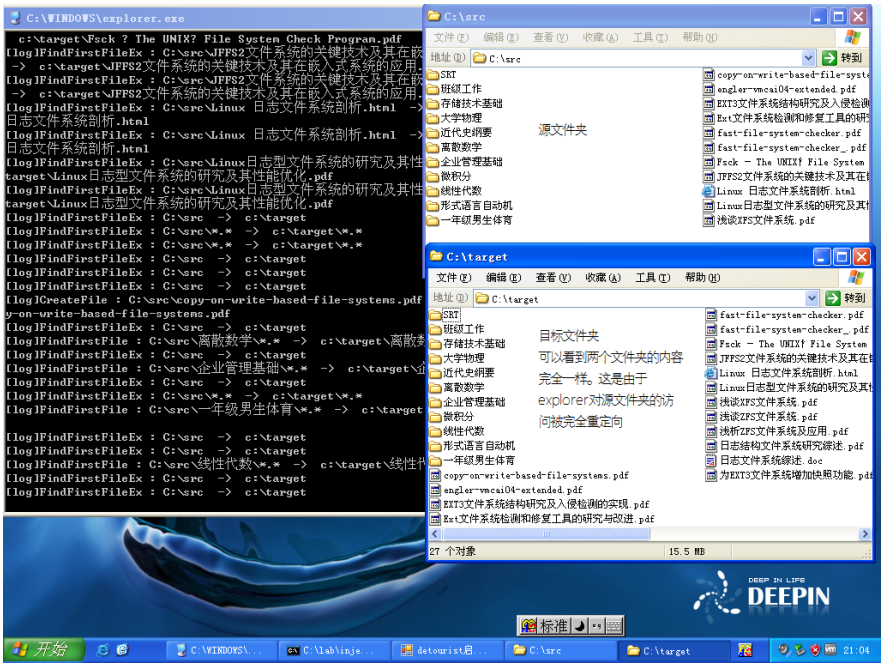



图 8: 对 explorer 进行注入后的 src 目录和 target 目录

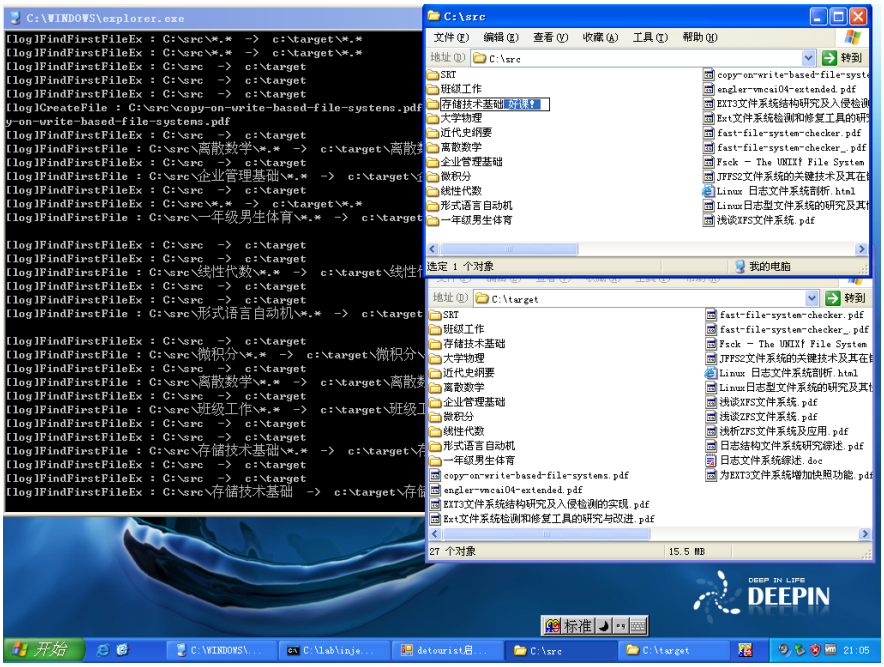



图 9: 对源文件夹中的目录重命名，操作被重定向至目标资料夹

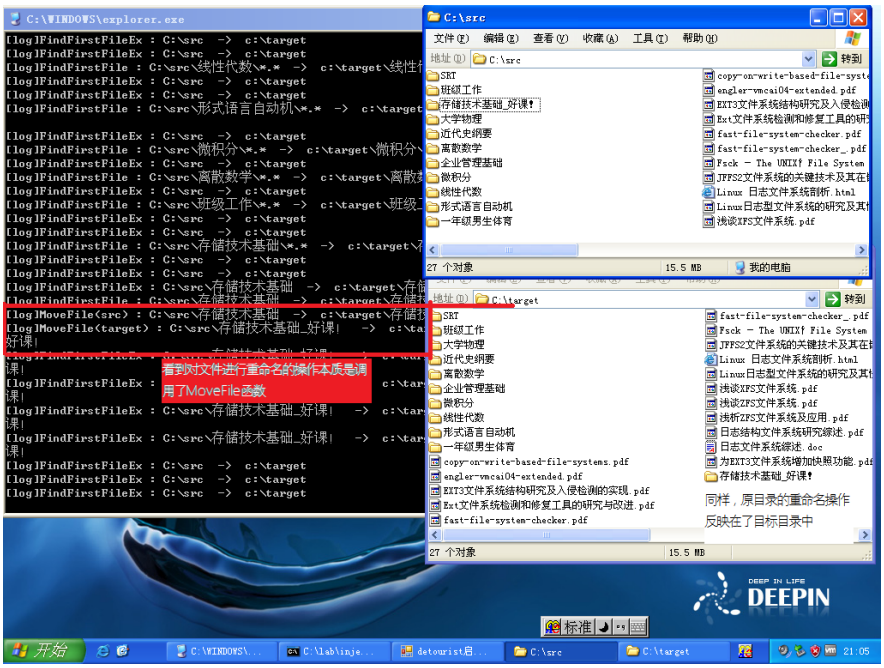



图 10: 重命名结束后，看到 target 路径下的相应目录文件夹名同时进行了修改

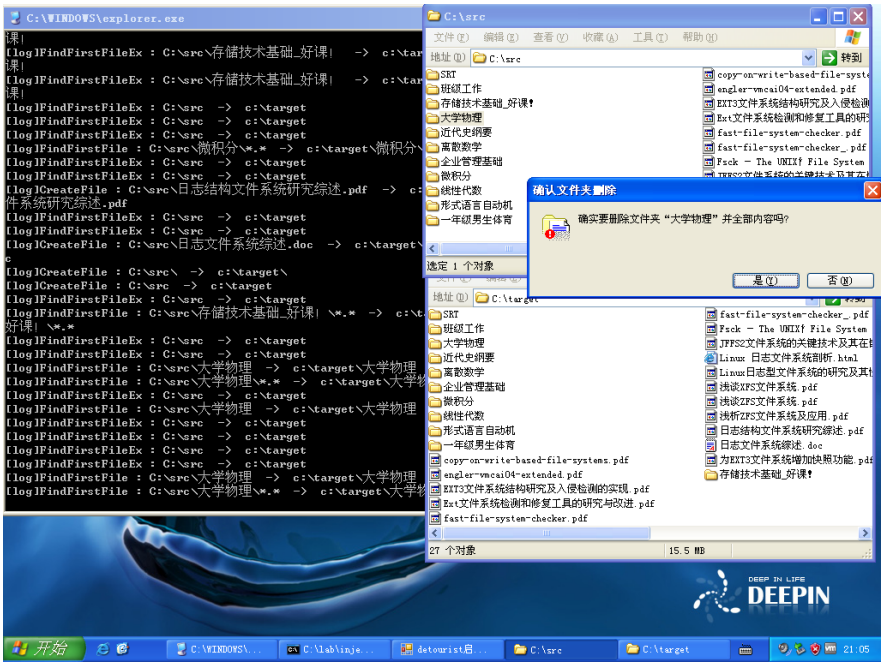



图 11: 对 src 路径下的文件夹进行删除操作

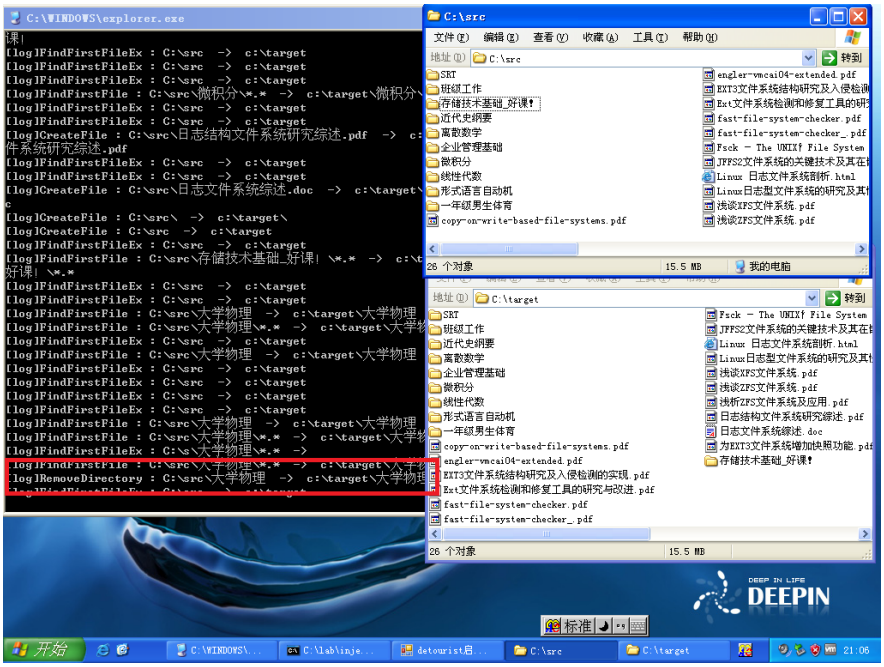



图 12: 删除结束后，看到目标路径内相应目录已被删除。同时日志中记录了删除操作

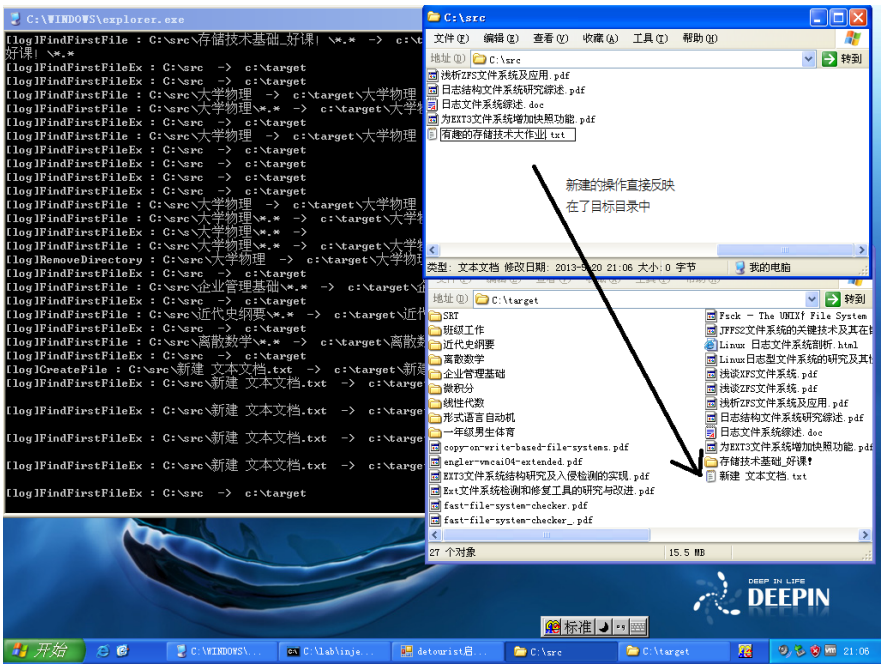



图 13: 在源目录中创建一个新的文本文档。同时可以看到 target 路径下已经产生了新的文件

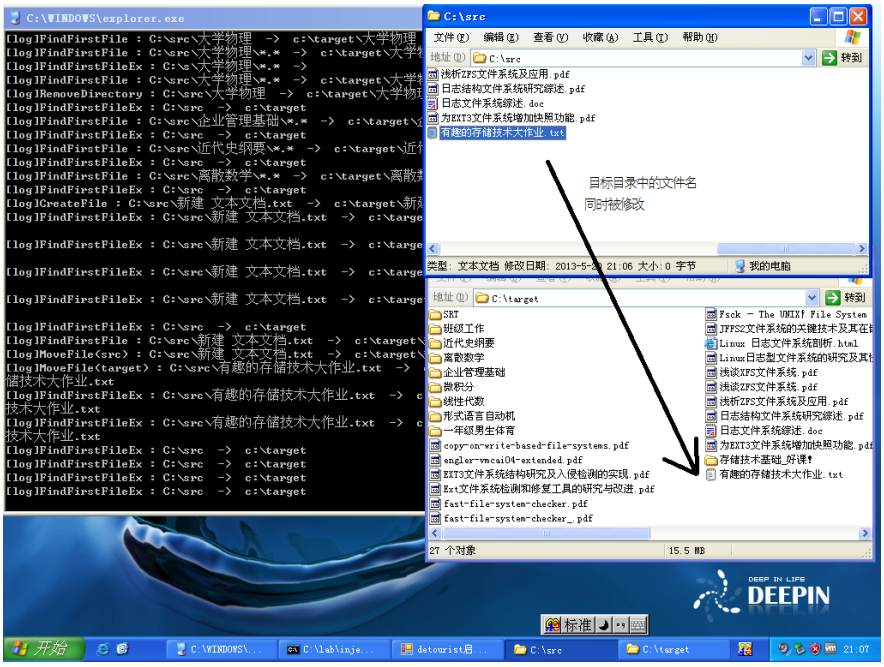



图 14: 输入新的文件的文件名后看到 target 目录下的相应文件名的改变

以上是将 detourist.dll 注入到 winXP 的 explorer 中获得的结果。为了演示对文件读写的重定向功能，接下来我选择将 detourist 注入到 notepad 中。该 notepad 即为 winXP 自带的 notepad。事实上，在 win7 环境下对 notepad 的注入实验同样十分成功。

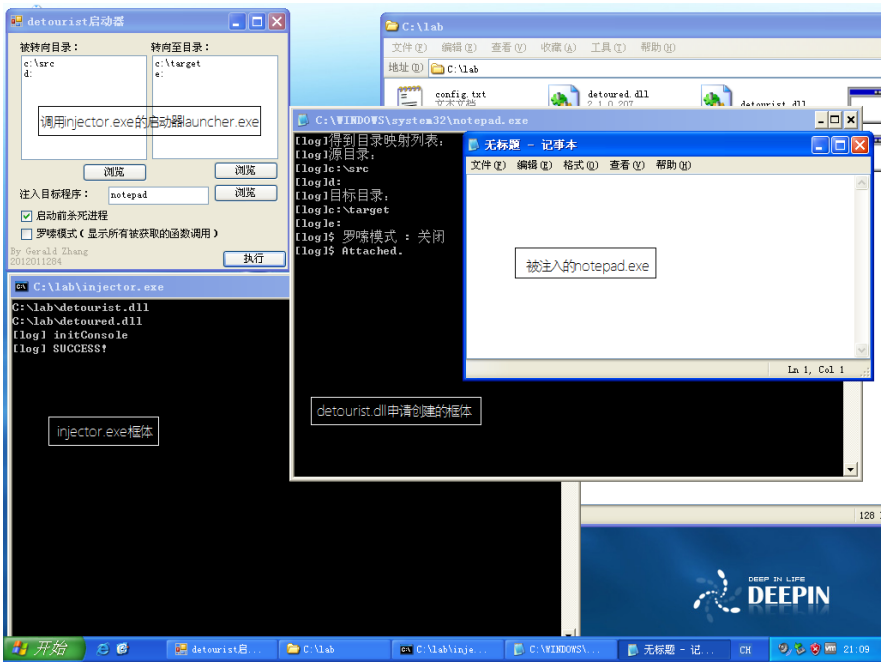



图 15: 执行 launcher，选择注入程序为 notepad 并执行

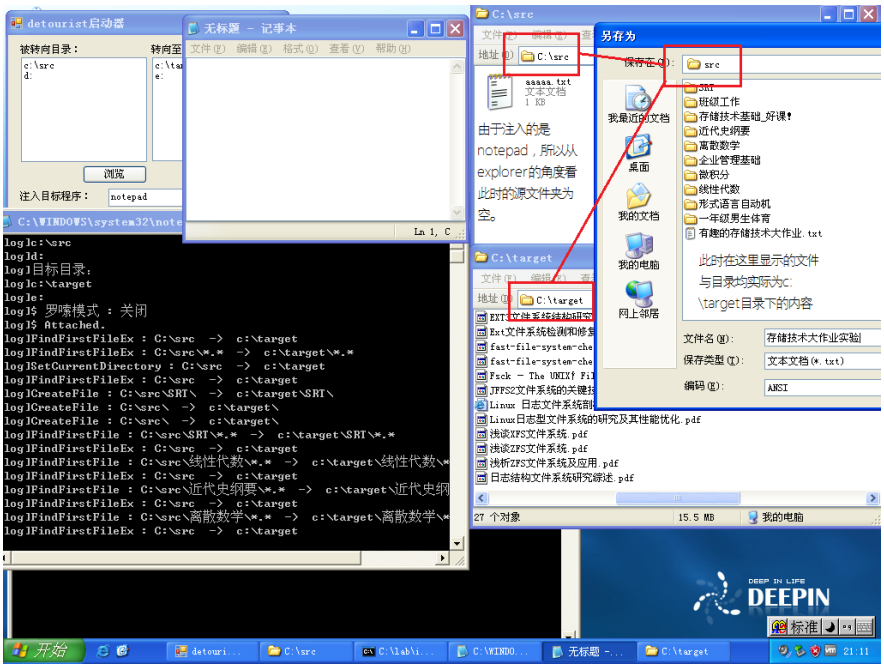



图 16: 对默认新建的无标题文本文件进行保存。当保存路径选择为 src 时，看到的文件夹内的内容为 target 文件夹内的内容

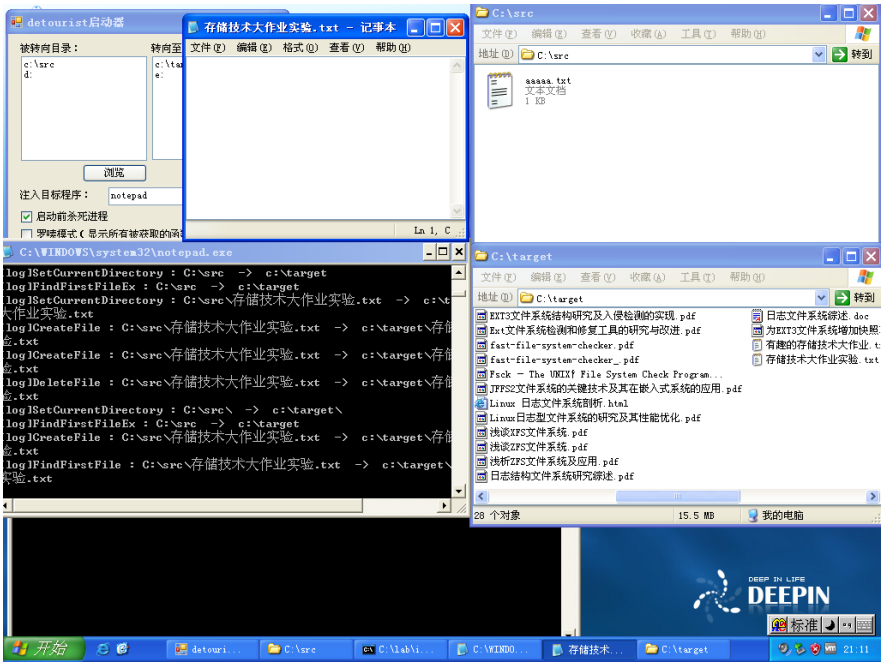



图 17: 完成保存之后，看到 target 路径下已经多了刚刚保存的文本文件

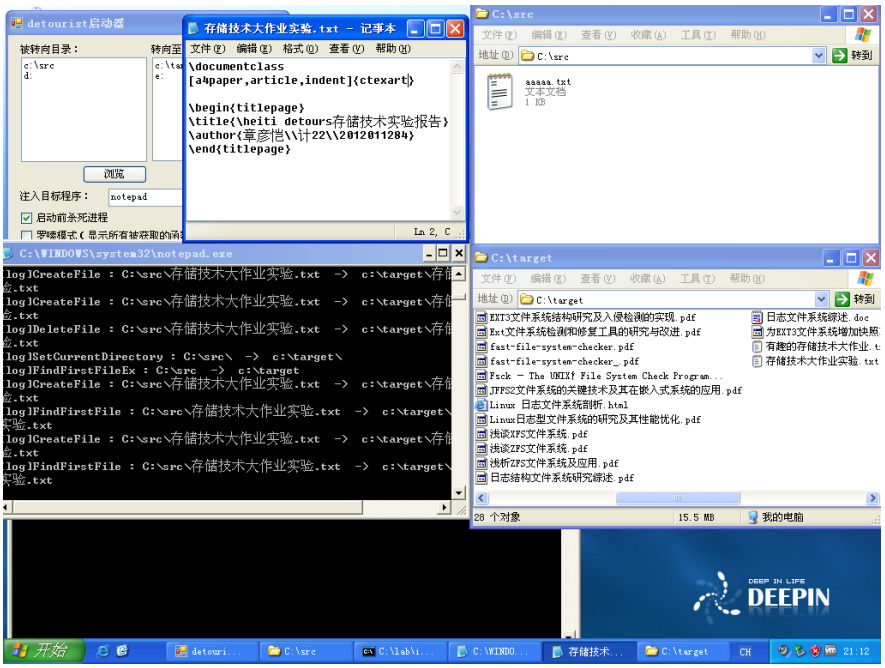



图 18: 对刚刚保存的文本文件内容进行修改

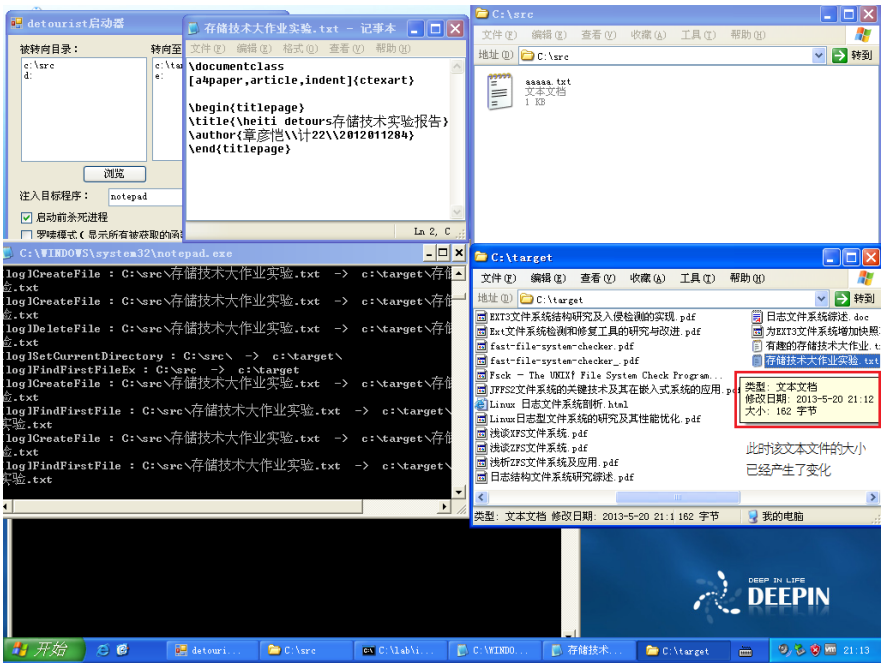



图 19: 保存后看到该修改成功作用在 target 目录下的相应文件。该文件的大小已经产生了改变

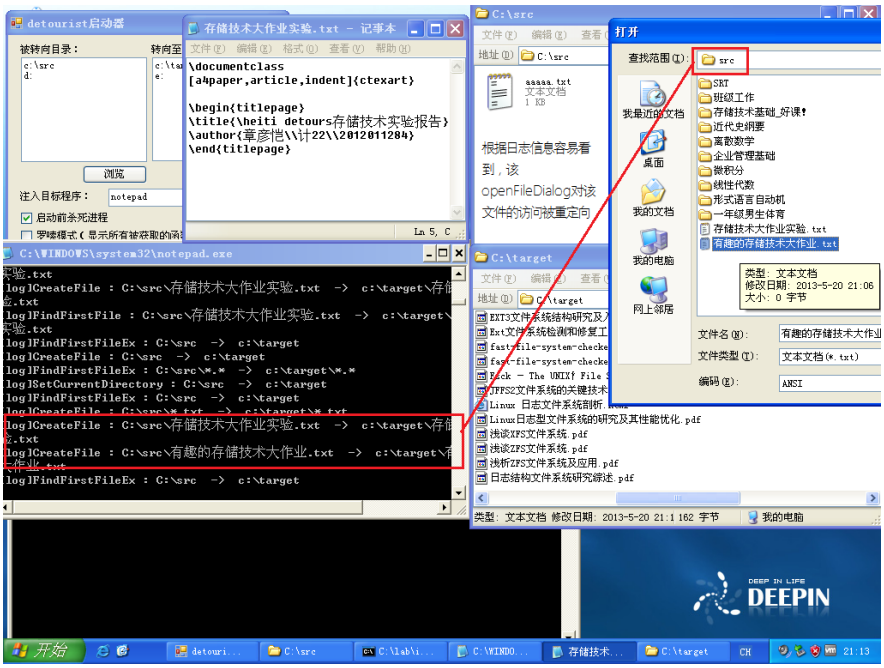



图 20: 打开 src 目录下被重定向至 target 目录中的其他文件

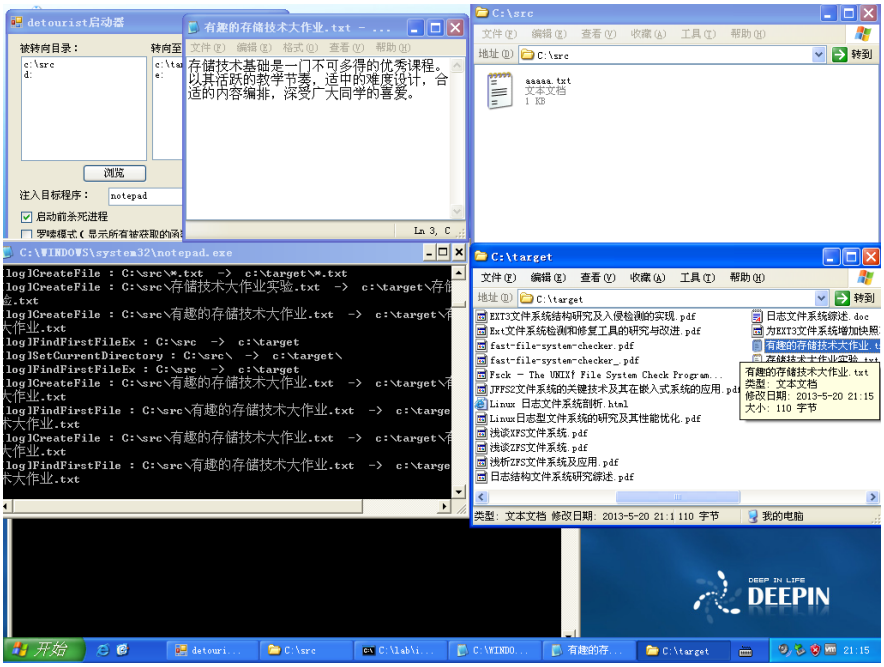



图 21: 保存后看到，相应文件内容成功被写入到重定向的目标文件中

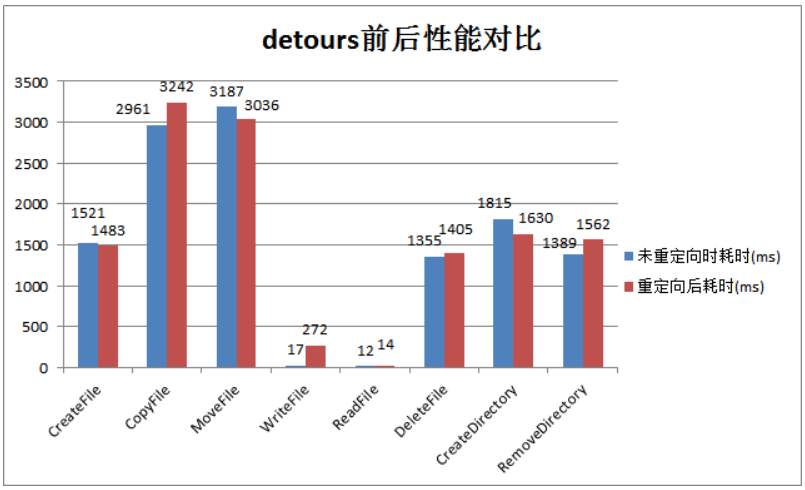
从上面两个例子可见，detourist.dll 已经能够对应用程序的文件操作进行截获并将其解析至另一个目标位置。且这样的注入方式不依赖于目录、文件的个数。只要求所实现的功能已经可以基本满足不影响被注入应用程序运行的要求。实验目标已较好地完成。

## 4.2 detours 前后性能检测

为了进行严谨的性能检测，我编写了 runTest.exe 来继续测试。runTest 将读取 runtest.txt 中的测试参数，并根据这个参数进行测试，将结果输出到 report.txt 中。测试的内容如 3.2.4 节所示。

在本次的测试中，分为不使用 detourist 注入和 detourist 注入两组，每组都对所列举的函数进行了 10000 次操作。执行本次测试的环境是 Windows 7 Ultimate，程序运行在 vcredist2010 环境下，相关硬件环境为 CPU: i5 450M 2.4GHz，可用物理内存 3.8GB，硬盘：西数 WDC WD5000BEVT-24A0RT0, 500 GB, 5400 r/min。

测试的结果制成图表如下：




对图表进行简要的分析可以比较容易地得到以下两点结论：

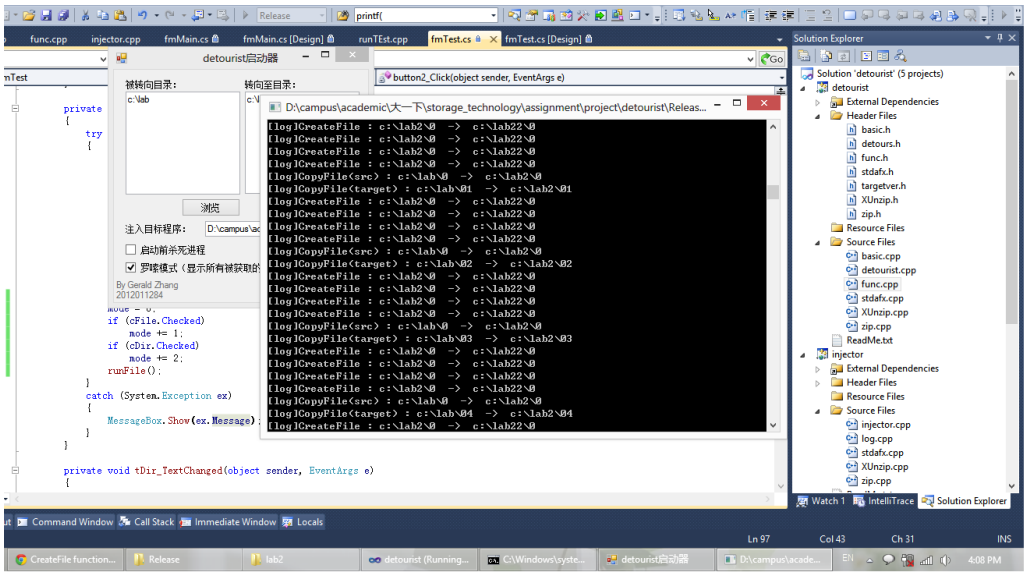
1. 容易看到，使用 detours 的前后运行效率差距并不明显。或者说 detours 前后的运行效率较大程度上受到的是硬盘读写速度的影响。重定向所造成的额外时间与硬盘读写的效率相比能够被忽略。

2. 对文件和目录的创建操作和删除操作等涉及到不同文件的操作所需要的时间远比对单一文件的读写操作耗时多。

3. 系统创建和删除一个目录与创建和删除一个文件所需要的时间相近。

4. 对文件的复制和移动操作所需要的时间高于测试中的其他所有函数。

我对文件的复制和移动操作的时间消耗原因十分好奇，因而对次做了一些更多的实验。我发现在进行复制和移动操作的时候日志的记录如图：




注意到图中每一次的 CopyFile 操作都调用了四次额外的 CreateFile 操作。我猜想这些操作是对文件的存在性进行判断的调用。也是这些操作导致了文件的复制和移动操作的缓慢。

# 5 体会与感想

本次实验给我带来的收获颇丰。

作为直接的收获，我在这次实验中学会了如何去寻找、编译并使用一个外部的类，这在我之前的编程中所从未有过的。其次我对文件系统的工作原理、运行机制有了更进一步的了解，我也体会到存储技术开发过程中的乐趣与辛劳。更具体的，我学会了将自己编写的 dll 代码注入到目标程序中后使用 detours 库来截获目标程序的系统调用。在学会 detours 的使用方法的同时，它还为我提供了一种利用钩子 hook 来实现不修改原始代码而对

程序运行的结果产生影响的思想。这种思想将在我今后的学习开发过程中令我受益。

作为一个大一入学不足一年的学生，我对计算机专业的了解还比较狭窄，代码能力还有待加强，同时周围没有能够与我进行讨论合作的同学，整个开发是我完全独立地进行的。在这样的环境下，我以花费更多的时间为代价，我认为依然较好地完成了本次实验的要求。即使开发的过程中遇到了许许多多的困难（3.3 节），然而我也都一一独立地进行了处理。我的意志在实验中获得了磨练，工程开发的能力也得到了较大的增强。这也是我第一次书写实验报告。这份实验报告的撰写也将为我提供在实验报告的书写方面的宝贵经验。

在这里我要感谢 0 字班的陈可卿学长。是他告知了我书写实验报告时的框架与需要注意的地方。我要感谢 FIT1-512 的各位老师同学，在宿舍断电的环境下为我提供安静高效，供电稳定的开发环境。

最后我要感谢张老师提供这一次实验的机会。这一次宝贵的实验开发的经历使我感受到，通过计算机专业的学习与实践，我的专业能力在得到不断的锻炼和提升。我将继续努力奋斗，争取早日成为一个具有丰富开发经验与科研能力，有追求的计算机科学学者。