|  |
| --- |
| 1. 实验目的及实验内容   （本次实验所涉及并要求掌握的知识；实验内容；必要的原理分析） |
| **实验目的：**  内核模块的编写：完成一个Linux/Windows内核/驱动模块的编写，  能够实现对文件访问的监控、或者对键盘设备、USB设备、网络设备、  蓝牙设备等的监控。  **实验内容：**  通过python编程调用windows的api，编写键盘和鼠标监控的hook，将相关信息保存记录到txt文档中。 |
| 1. 实验环境及实验步骤   （本次实验所使用的器件、仪器设备等的情况；具体的实验步骤） |
| 实验环境：  Windows10  Python3.7  Anaconda3，Pycharm  关键技术：  Windows的Hook技术  实验步骤：   1. **Hook技术，pyHook3和pywin32简介**    1. **Hook简介**   windows应用程序是基于消息驱动的。各种应用程序对各种消息作出响应从而实现各种功能。  hook（钩子）是一种特殊的消息处理机制，它可以监视系统或者进程中的各种事件消息，截获发往目标窗口的消息并进行处理。所以说，我们可以在系统中自定义钩子，用来监视系统中特定事件的发生，完成特定功能，如屏幕取词，监视日志，截获键盘、鼠标输入等等。  钩子实际上是一个处理消息的[程序段](https://baike.baidu.com/item/%E7%A8%8B%E5%BA%8F%E6%AE%B5" \t "https://baike.baidu.com/item/hook/_blank)，通过[系统调用](https://baike.baidu.com/item/%E7%B3%BB%E7%BB%9F%E8%B0%83%E7%94%A8" \t "https://baike.baidu.com/item/hook/_blank)，把它挂入系统。每当特定的消息发出，在没有到达目的窗口前，[钩子程序](https://baike.baidu.com/item/%E9%92%A9%E5%AD%90%E7%A8%8B%E5%BA%8F" \t "https://baike.baidu.com/item/hook/_blank)就先捕获该消息，亦即[钩子函数](https://baike.baidu.com/item/%E9%92%A9%E5%AD%90%E5%87%BD%E6%95%B0" \t "https://baike.baidu.com/item/hook/_blank)先得到控制权。这时钩子函数即可以加工处理（改变）该消息，也可以不作处理而继续传递该消息，还可以强制结束消息的传递。Windows消息带了一些程序有用的信息，比如Mouse类信息，就带有鼠标所在窗体句柄、鼠标位置等信息，拦截了这些消息，就可以利用这些信息做出一些功能  每一个Hook都有一个与之相关联的[指针](https://baike.baidu.com/item/%E6%8C%87%E9%92%88" \t "https://baike.baidu.com/item/hook/_blank)列表，称之为钩子[链表](https://baike.baidu.com/item/%E9%93%BE%E8%A1%A8" \t "https://baike.baidu.com/item/hook/_blank)，由系统来维护。这个列表的指针指向指定的，应用程序定义的，被Hook子程调用的[回调函数](https://baike.baidu.com/item/%E5%9B%9E%E8%B0%83%E5%87%BD%E6%95%B0" \t "https://baike.baidu.com/item/hook/_blank)，也就是该钩子的各个处理子程序。当与指定的Hook类型关联的消息发生时，系统就把这个消息传递到Hook子程。一些Hook子程可以只监视消息，或者修改消息，或者停止消息的前进，避免这些消息传递到下一个Hook子程或者目的窗口。最近安装的钩子放在链的开始，而最早安装的钩子放在最后，也就是后加入的先获得控制权。  [系统钩子](https://baike.baidu.com/item/%E7%B3%BB%E7%BB%9F%E9%92%A9%E5%AD%90" \t "https://baike.baidu.com/item/hook/_blank)与线程钩子：  SetWindowsHookEx()函数的最后一个参数决定了此钩子是系统钩子还是线程钩子。  线程钩子用于监视指定线程的事件消息。线程钩子一般在当前线程或者当前线程派生的线程内。  系统钩子监视系统中的所有线程的事件消息。因为系统钩子会影响系统中所有的应用程序，所以[钩子函数](https://baike.baidu.com/item/%E9%92%A9%E5%AD%90%E5%87%BD%E6%95%B0" \t "https://baike.baidu.com/item/hook/_blank)必须放在独立的动态链接库(DLL) 中。系统自动将包含“钩子[回调函数](https://baike.baidu.com/item/%E5%9B%9E%E8%B0%83%E5%87%BD%E6%95%B0" \t "https://baike.baidu.com/item/hook/_blank)”的DLL映射到受钩子函数影响的所有进程的[地址空间](https://baike.baidu.com/item/%E5%9C%B0%E5%9D%80%E7%A9%BA%E9%97%B4" \t "https://baike.baidu.com/item/hook/_blank)中，即将这个DLL注入了那些进程。  对于 Hook 技术，可以分为两块，第一块是在 Ring3 层的 Hook，俗称应用层 Hook 技术，另外一块自然是在 Ring0 层得 Hook，俗称为内核层 Hook 技术。   * 1. **Hook的windows API**   操作系统支持多种类型的钩子，每种类型都提供了它特有的消息处理机制。  对于每种类型的钩子，系统都维护一个各自独立的钩子链，钩子链是一个指向用户提供的回调函数钩子过程的链表指针。  **Hook的安装：**  HHOOK SetWindowsHookEx{  int idHook,//要安装的钩子的类型  HOOKPPROC lpfn,//钩子过程的指针，拦截到制定系统消息后的预处理过程  HINSTANCE hMod,//应用程序实例的句柄，如果是全局钩子，hInstance是DLL句柄（DllMain中给的模块地址。就是包含HookProc的动态库加载地址。否则给0就可以了，即勾自己。 ）  DWORD dwThreadId //要安装钩子的线程id，指定被监视的线程，如果明确指定了某个线程的id就只监视该线程，此时的钩子为线程钩子；如果该参数被设置为0，则表示此钩子为监视系统所有线程的全局钩子  }  返回值**:**若此函数执行成功,则返回值就是该挂钩处理过程的句柄;若此函数执行失败,则返回值为NULL(0).  **Hook过程：**  LRESULT CALLBACK HookProc{ //  int nCode, //该参数是一个钩子标识码，钩子过程会利用它决定下一步的进行的操作。这个标识嘛的值与安装的钩子类型相关  WPARAM wParam,//后面两个参数的定义都依赖于nCode参数，一般用于存放于窗口消息相关的内容  LPARAM lParam}  LRESULT 就是long型，是Windows API的一种返回类型；CALLBACK表示这个函数是给系统调用的，实际上CALLBACK就是\_\_stdcall（回调函数）。HookProc指代自定义的函数。  **Hook卸载：**  BOOL WINAPI UnhookWindowsHookEx( \_\_in HHOOK hhk);  HHOOK要删除的钩子的句柄。这个参数是函数[SetWindowsHookEx](https://baike.baidu.com/item/SetWindowsHookEx" \t "https://baike.baidu.com/item/UnhookWindowsHookEx/_blank)的返回值。返回值类型: BOOL，如果函数成功，返回值为非零值。如果函数失败，返回值为零。 要获得更多的错误信息，调用[GetLastError](https://baike.baidu.com/item/GetLastError" \t "https://baike.baidu.com/item/UnhookWindowsHookEx/_blank)函数.  **Hook信息传递：**  CallNextHookEx是一种函数，可以将钩子信息传递到当前钩子链中的下一个子程，一个钩子程序可以调用这个函数之前或之后处理钩子信息。  LRESULT WINAPI CallNextHookEx( \_In\_opt\_ HHOOK hhk, \_In\_ int nCode, \_In\_ WPARAM wParam, \_In\_ LPARAM lParam);  **Hook类型：**  有很多，举两个例子：  WH\_KEYBOARD     //当敲击键盘时将触发此钩子  WH\_MOUSE    //当有鼠标操作时将触发此钩子   * 1. **pyHook3与pywin32简介**   **Pywin32：**一个开源的python项目，该模块包含了几乎所有的windows API，供调用。https://github.com/mhammond/pywin32。封装方式是从windows的DLL中按类提取了API函数放在不同位置，比如process进程类API放在win32process模块。  **PyHook3:**依赖于Pywin32，用于捕捉特定的Windows事件，封装了所有底层调用（比如对SetWindowsHookEx()，UnhookWindowsHookEx(),等函数封装），我们只需要关注程序逻辑。提供键盘和鼠标的Hook。   1. **安装pyHook3和pywin32（pycom）**   需要注意pyHook和pywin32与python版本的对应。  **2.1 查看python版本**  Python 3.7，AMD64    **2.2 pyHook3安装**  pyHook官方只支持python2，所以安装pyhook3  Conda install swig  Pip install pyhook3  **2.3 Pycom安装：**  Pycom即pywin32  <https://github.com/mhammond/pywin32/releases>    **2.4 查看是否安装成功：**  Conda list  安装成功       1. **利用Hook实现键盘监控与鼠标监控**    1. **思路：**   **没有Hook时：**  键盘输入--> 系统消息队列 --> 对应应用程序的消息队列 --> 将消息发送到对应的窗口中  **添加Hook后：**  键盘输入 --> 系统消息队列 --> 对应应用程序消息队列 --> 将消息发送到钩子链中 --> 消息一一调用完毕所有的钩子函数(需要调用CallNextHookEx函数才能将消息传递下去) --> 将消息发送到对应的窗口中  添加两个Hook函数分别对应键盘和鼠标。在Hook函数接收到信息后保存到txt中，完成监控，然后把信息传给目标程序。   * 1. **代码实现**   **思路:**  添加一个键盘hook，会获取到键盘信息。Hook函数功能为将信息转化为string后写入txt文档，然后将信息继续传递给目标窗口。  有用的信息为Time 事件时间，MessageName事件名称，WindowName事件所在窗口名，Ascii事件ASCII码，KEY 事件KEY，Scancode扫描码，Alt控制信息。  添加一个鼠标Hook，获取鼠标信息，功能类似于键盘Hook。  有用的信息为Time 事件时间，MessageName事件名称，WindowName事件所在窗口名，Position鼠标所在坐标位置，Wheel鼠标滚轮的信息。  在键盘Hook的函数中判断键盘输入是否为Q,为Q时quit。  存入的文档设置为F://records.txt。  **具体代码见实验结果记录** |
| 1. 实验过程分析   （实验分工，详细记录实验过程中发生的故障和问题，进行故障分析，说明故障排除的过程及方法。根据具体实验，记录、整理相应的数据表格等） |
| **遇到的问题及解决：**   1. **对于Hook技术不了解**   查询看雪论坛，csdn等资料后有所了解。   1. **pyHook在python3环境有bug**   **问题描述：**  会因为编码问题导致移动到窗口标题含有非ASCII码时报错  TypeError: MouseSwitch() missing 8 required positional arguments: 'msg', 'x', 'y', 'data', 'time', 'hwnd', and 'window\_name'  **问题解决：**  安装适应python3的pyHook3。  **实验结果记录：**   1. **code：**   # -\*- coding: utf-8 -\*- """ Created on Wed Dec 25 11:33:41 2019 @author: erio """ #USB 接口的键盘鼠标 *import* pythoncom *import* PyHook3 *as* pyHook *import* win32api *import* time  path = 'F://records.txt'  *def* onMouseEvent(*event*):  *with* open(path, 'a+') *as* f:  f.write("-----Mouse Event Start-----\n")  # 监听鼠标事件  *print* ("MessageName:", *event*.MessageName)  *print* ("WindowName:", *event*.WindowName)  *print* ("Position:", *event*.Position)  *print* ("Wheel:", *event*.Wheel)  *print* ("---")  # 返回 True 以便将事件传给其它处理程序  # 注意，这儿如果返回 False ，则鼠标事件将被全部拦截  t = time.localtime()  t = time.asctime(t)  result = "Time : " + t + "\n" + "MessageName: "+ str(*event*.MessageName)+'\n'+"WindowName: " + str(*event*.WindowName) + \  "\n" +"Position: "+str(*event*.Position)+'\n'+"Wheel: " +str(*event*.Wheel)+'\n'  f.write(result)  f.write("-----Mouse Event End-----\n\n\n")  f.close()  *return* True  *def* onKeyboardEvent(*event*):  *with* open(path, 'a+') *as* f:  # 监听键盘事件  f.write("-----KeyBoard Event Start-----\n")  *print* ("MessageName:", *event*.MessageName)  *print* ("WindowName:", *event*.WindowName)  *print* ("Ascii:", *event*.Ascii, chr(*event*.Ascii))  *print* ("Key:", *event*.Key)  *print* ("ScanCode:", *event*.ScanCode)  *print* ("Alt", *event*.Alt)  *print* ("---")  t=time.localtime()  t=time.asctime(t)  result ="Time : " + t + "\n" +"MessageName: "+ str(*event*.MessageName)+'\n'+"WindowName: " + str(*event*.WindowName) + " \n" +\  "Ascii: "+str(*event*.Ascii)+ ' '+chr(*event*.Ascii)+'\n'+"Key: " + str( *event*.Key) + "\n" +"ScanCode: "+str(*event*.ScanCode)+"\n"+\  "Alt: "+str(*event*.Alt)+'\n'  f.write(result)  f.write("-----KeyBoard Event End-----\n\n\n")  f.close()  *if event*.Key== 'Q': # 按下F12后终止adsw  win32api.PostQuitMessage()  *return* True  *def* main():  # 创建一个“钩子”管理对象aaavv q  hm = pyHook.HookManager()  # 监听所有键盘事件  hm.KeyDown = onKeyboardEvent  # 设置键盘“钩子”  hm.HookKeyboard()  # 监听所有鼠标事件  hm.MouseAll = onMouseEvent  # 设置鼠标“钩子”  hm.HookMouse()  # 进入循环，如不手动关闭，程序将一直处于监听状态  pythoncom.PumpMessages()   *if* \_\_name\_\_ == "\_\_main\_\_":  main()   1. **records.txt文档展示**   实际随机操作后记录在records.txt中。由于记录较长（13kb），截取部分展示。    **键盘信息记录：包含普通按键和Alt**  信息为Time 事件时间，MessageName事件名称，WindowName事件所在窗口名，Ascii事件ASCII码，KEY 事件KEY，Scancode扫描码，Alt控制信息。  -----KeyBoard Event Start-----  Time : Wed Dec 25 17:19:28 2019  MessageName: key down  WindowName: Pyhook [D:\Pyhook] - ...\records.py [Pyhook] - PyCharm  Ascii: 13  Key: Return  ScanCode: 28  Alt: 0  -----KeyBoard Event End-----  -----KeyBoard Event Start-----  Time : Wed Dec 25 17:19:32 2019  MessageName: key sys down  WindowName: Pyhook [D:\Pyhook] - ...\records.py [Pyhook] - PyCharm  Ascii: 0  Key: Lmenu  ScanCode: 56  Alt: 32  -----KeyBoard Event End-----  -----KeyBoard Event Start-----  Time : Wed Dec 25 17:19:34 2019  MessageName: key sys down  WindowName: Pyhook [D:\Pyhook] - ...\records.py [Pyhook] - PyCharm  Ascii: 0  Key: Lmenu  ScanCode: 56  Alt: 32  -----KeyBoard Event End-----  -----KeyBoard Event Start-----  Time : Wed Dec 25 17:19:34 2019  MessageName: key sys down  WindowName: Pyhook [D:\Pyhook] - ...\records.py [Pyhook] - PyCharm  Ascii: 0  Key: Numpad2  ScanCode: 80  Alt: 32  -----KeyBoard Event End-----  **鼠标信息记录：包含移动和按下左键**  信息为Time 事件时间，MessageName事件名称，WindowName事件所在窗口名，Position鼠标所在坐标位置，Wheel鼠标滚轮的信息。  -----Mouse Event Start-----  Time : Wed Dec 25 17:19:42 2019  MessageName: mouse move  WindowName: None  Position: (997, 458)  Wheel: 0  -----Mouse Event End-----  -----Mouse Event Start-----  Time : Wed Dec 25 17:19:42 2019  MessageName: mouse move  WindowName: None  Position: (995, 458)  Wheel: 0  -----Mouse Event End-----  -----Mouse Event Start-----  Time : Wed Dec 25 17:19:42 2019  MessageName: mouse left down  WindowName: None  Position: (995, 458)  Wheel: 0  -----Mouse Event End-----  -----Mouse Event Start-----  Time : Wed Dec 25 17:19:42 2019  MessageName: mouse left up  WindowName: Pyhook [D:\Pyhook] - ...\records.py [Pyhook] - PyCharm  Position: (995, 458)  Wheel: 0  -----Mouse Event End-----   1. **Records记录的信息分析**   通过观察可知由于鼠标信息中包含坐标，而坐标改变比较频繁，所有鼠标信息高频被Hook。同时鼠标信息有较强的关联性，比如连续信息的位置接近，左键按下和松开成对出现。  而键盘信息出现频率较低，同时随机性很大，和外部输入有关。 |