说明：

Sever控制端：

1. 1.png来自被控端的摄像头截图
2. background.jpg背景图片
3. Connect.py主要包含对象Connection将一个连接作为对象方便调用管理.
4. GUI.py GUI界面主体，也包括按钮所连接到的函数
5. main.py mian函数，只保留必要操作让代码简洁
6. key.apl 利用pickle的打包文件含有键盘记录的信息
7. p.txt 被控端进程信息
8. read.py 对文件的读写操作相关函数

Client被控制端：

1. 1.png摄像头图片
2. Cam.py 对摄像头的控制
3. connect.py同上
4. GUI.py GUI界面主体，也包括按钮所连接到的函数

5. key.apl 利用pickle的打包文件含有键盘记录的信息

6. 所有的keylog.txt用于键盘记录

7. main.py mian函数，只保留必要操作让代码简洁

8. p.txt 被控端进程信息

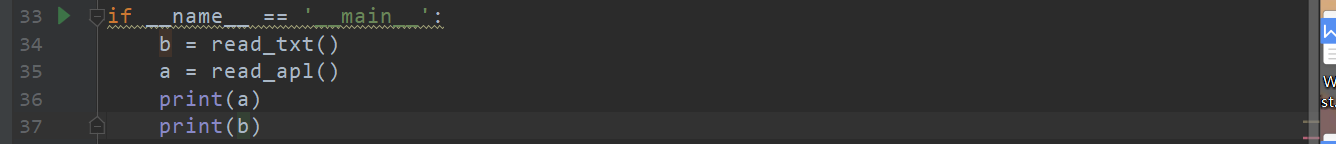
9. process.py 进程信息的获取

10. reg.py 现在程序用的键盘记录

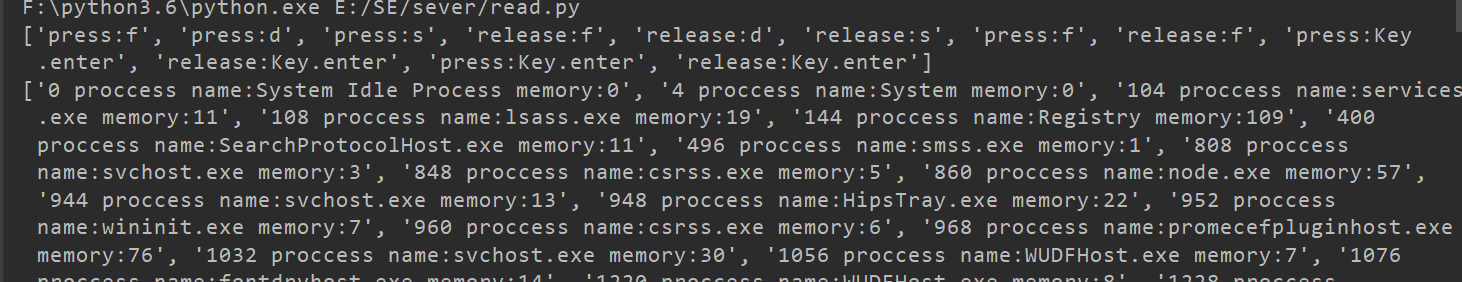
11. keylogger.py 利用钩子实现的键盘记录版本。

关于测试：很多单独的文件都有自己的main函数来测试其功能。

read.py测试：



测试两个函数的输入与输出：



输出正常read.py覆盖率90%

Sever中的Connection对象方法：

creat\_link创造连接

send\_pid 发送进程号

send\_require 发送TYPE3类型指令让被控端开始键盘监控

get\_type 获取消息的类型，然后返回以int类型

get\_length 获取长度消息返回int类型单位字节

get\_image 接受图像数据并写入，循环接受1024字节直到接收完毕，会有记录现有文件长度变量，现有文件长度=上次文件长度-实际接受数。

get\_txt 接受文本并写入，接受参数数据长度。算法与上面相同

get\_apl 接受apl文件，参数为数据大小，算法与上面相同

close 关闭链接

屏幕监控：

屏幕监控被控制的通过Cam.py开启摄像头，并进行截图在单机跑的时候我们截图分辨率可以高点（640\*480），在两台计算机时我们可以调低分辨率以提升速度。截图会储存为1.png,之后会通过connect的send\_image方法将图片发送给控制端，控制端播放图片，其实这里可能出现的问题是同时读写文件，但我们加了互斥锁，而且两边的sleep都经过设置，实际可能会发生读写冲突但很少见。而且出错处理会让这种错误几乎看不出来。

进程管理：

利用python库获取所有进程，包括进程号，进程名，以及进程占用内存写在txt里然后发送，这里并没有CPU信息什么的是因为准确的CPU占用需要时间计算，往往时间还较长所以这里并没有统计。之后控制端会读取txt然后将其写入窗口的selectView组件，组件会自动刷新，新开的或者关闭进程都会在其中显示。

键盘记录：

我们这里有2种键盘记录方式一个是pyHook3钩子去读键盘输入，一个是python自带的库。之后会保存为数组通过pickle打包发送，这个录制在发送端发送指令后才会开始，同样这个程序方法一个新的线程里。控制端得到apl文件后读取然后显示和进程管理中的显示逻辑差不多。