

# 网络安全管理技术作业

--基于 SNMP 的远程安全监控

姓名： 刘子阳

班级： F1603405

学号： 516021910796

时间： 2019.5.9

## 1. 目的

本实验主要是通过基于 SNMP 的远程安全监控的各项实践,加深对于 SNMP 协议的理解与认识,同时加强对于网络安全管理技术的认知,实现基于 SNMP 的远程安全监控的各项实践以及 GUI 界面的制作。本实验基于 python 3.5,并在 Windows 10 操作系统下完成,GUI 界面主要使用 python 的 Tkinter 库制作。

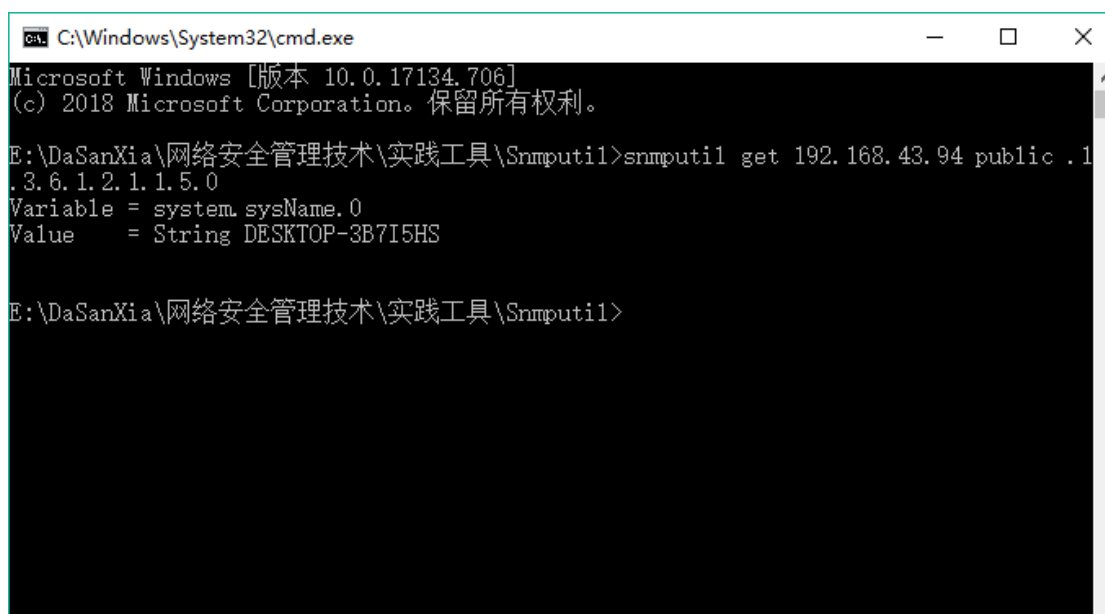
## 2. 内容（全部完成）

1. 添加并开启 Windows/Linux SNMP 代理组件; (2 分)
2. 编制控制台程序,接受用户输入的 OID 字符串,返回在 Windows/Linux SNMP 代理中的对应值; (5 分)
3. 开发 GUI 界面程序,使用户可通过控制台程序观察主机 CPU、内存、硬盘空间、流量值; (5 分)
4. 开发阈值告警功能,用户通过界面可设置性能阈值(如 CPU),当超过阈值时自动报警。(3 分)
5. 实现 SNMP Trap、Set 功能并测试。(3 分)
6. 在控制台动态显示主机 CPU、内存利用曲线。(2 分)

## 3. 步骤及测试结果

### 3.0 SNMP 初体验

按照教学 PPT 中的方法操作之后,可得到如下界面,获得了电脑的名称。



```
C:\Windows\System32\cmd.exe
Microsoft Windows [版本 10.0.17134.706]
(c) 2018 Microsoft Corporation. 保留所有权利。

E:\DaSanXia\网络安全管理技术\实践工具\Snmputil>snmputil get 192.168.43.94 public .1.3.6.1.2.1.1.5.0
Variable = system.sysName.0
Value    = String DESKTOP-3B7I5HS

E:\DaSanXia\网络安全管理技术\实践工具\Snmputil>
```

图 1 SNMP 初步体验成功界面

### 3.1 添加并开启 Windows SNMP 代理组件

同样按照 PPT 中的操作,将整个流程完成一遍的过如下面一系列图像所示。

首先在控制面板中的程序和功能中找到启用或关闭 windows 功能,在其中找打简单网络

管理协议（SNMP）选项并勾选。

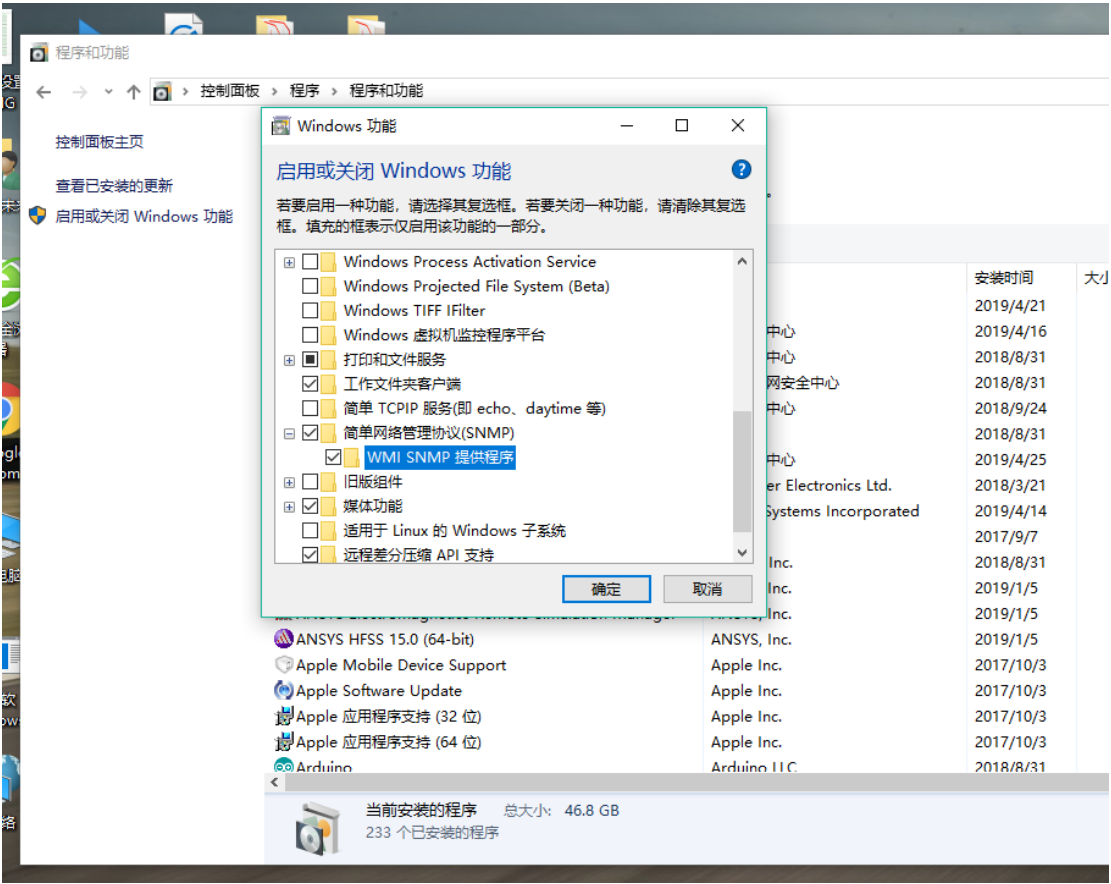


图 2 配置 SNMP 步骤 1

然后点击确定等待下载完成即可。

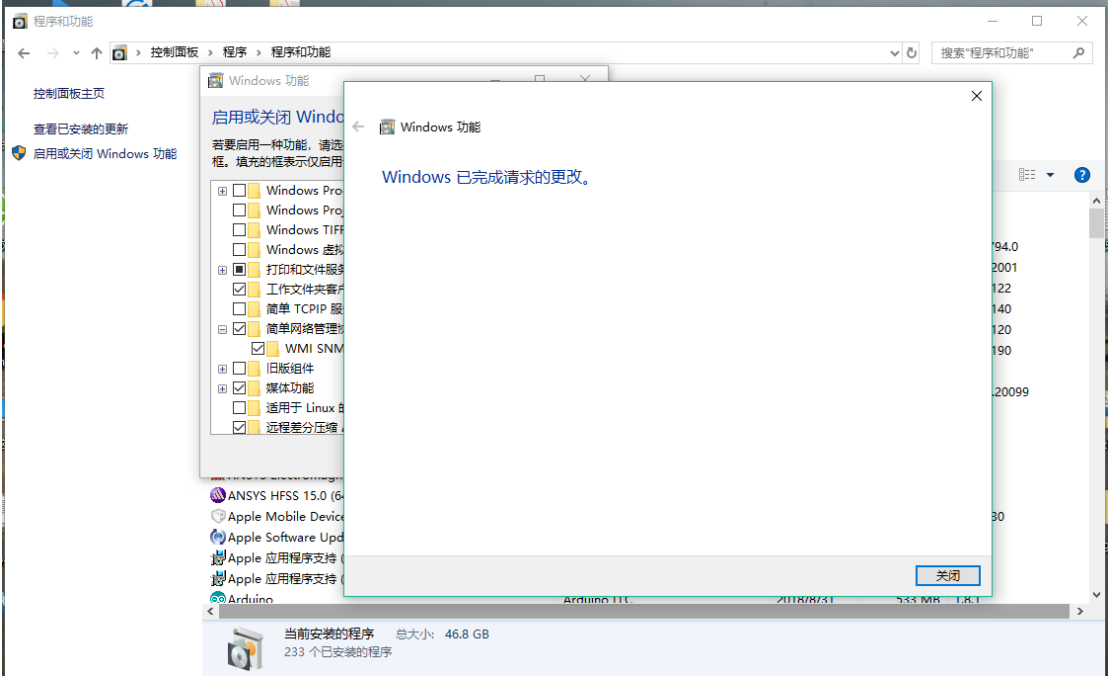


图 3 配置 SNMP 步骤 2

然后在 windows 搜索框内查询“服务”，并打开。

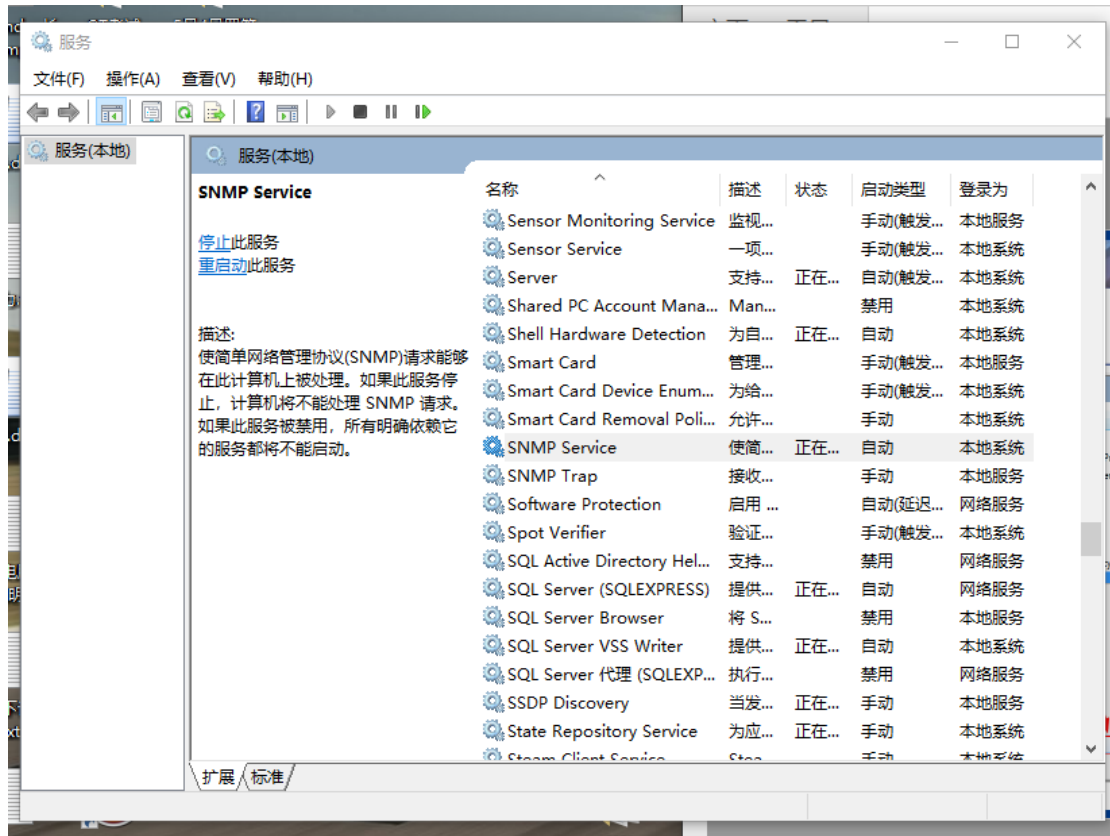


图 4 配置 SNMP 步骤 3

找到 SNMP Service 并右键选择属性，按下图中选择方式点击第一个“添加”，再按图中方式设置并确定。

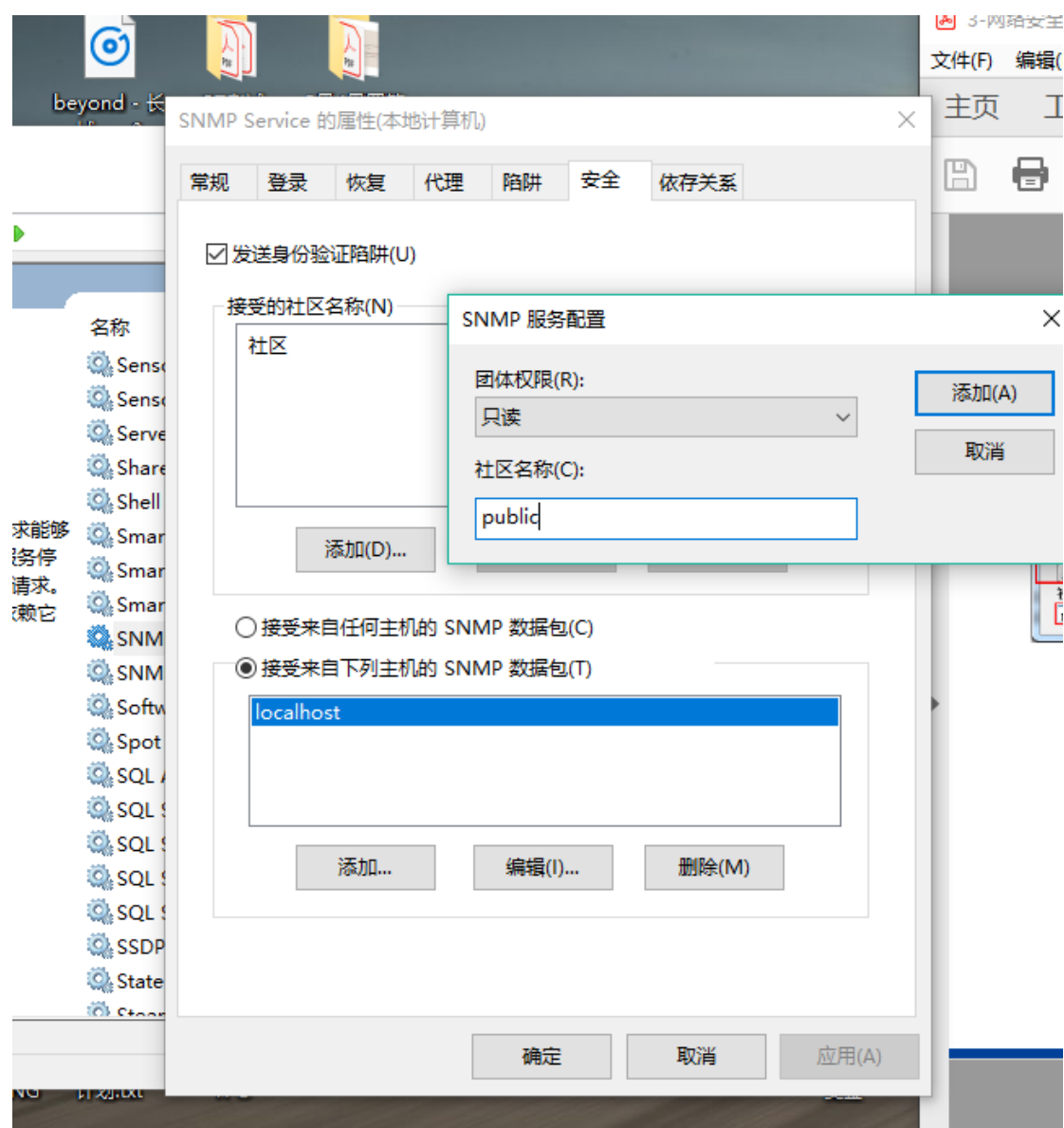


图 5 配置 SNMP 步骤 4  
再点击第二个“添加”，并输入自己电脑的 IP 地址。

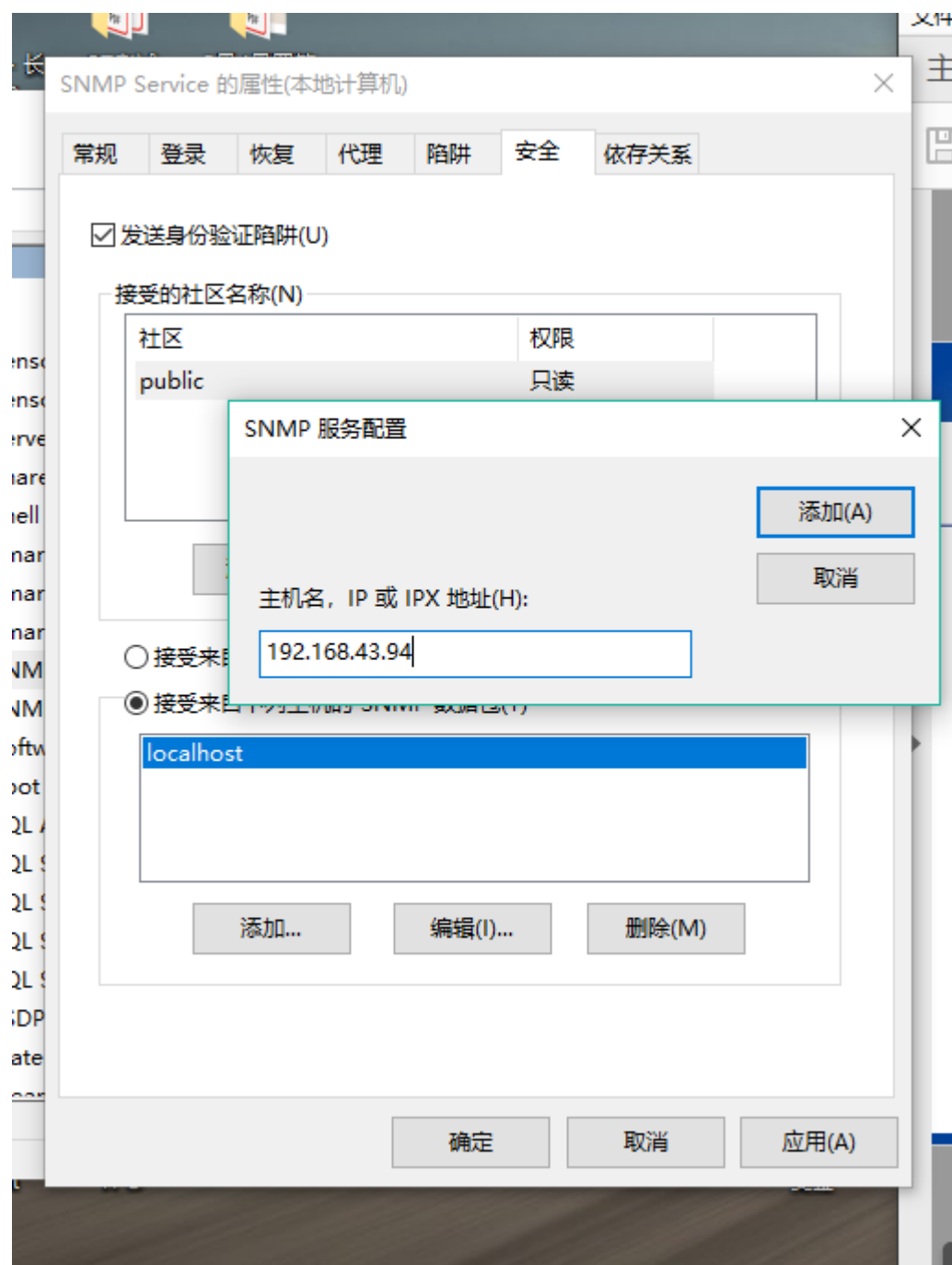


图 6 配置 SNMP 步骤 5

设置完成后出现如下界面，然后点击“应用”和“确定”即可。

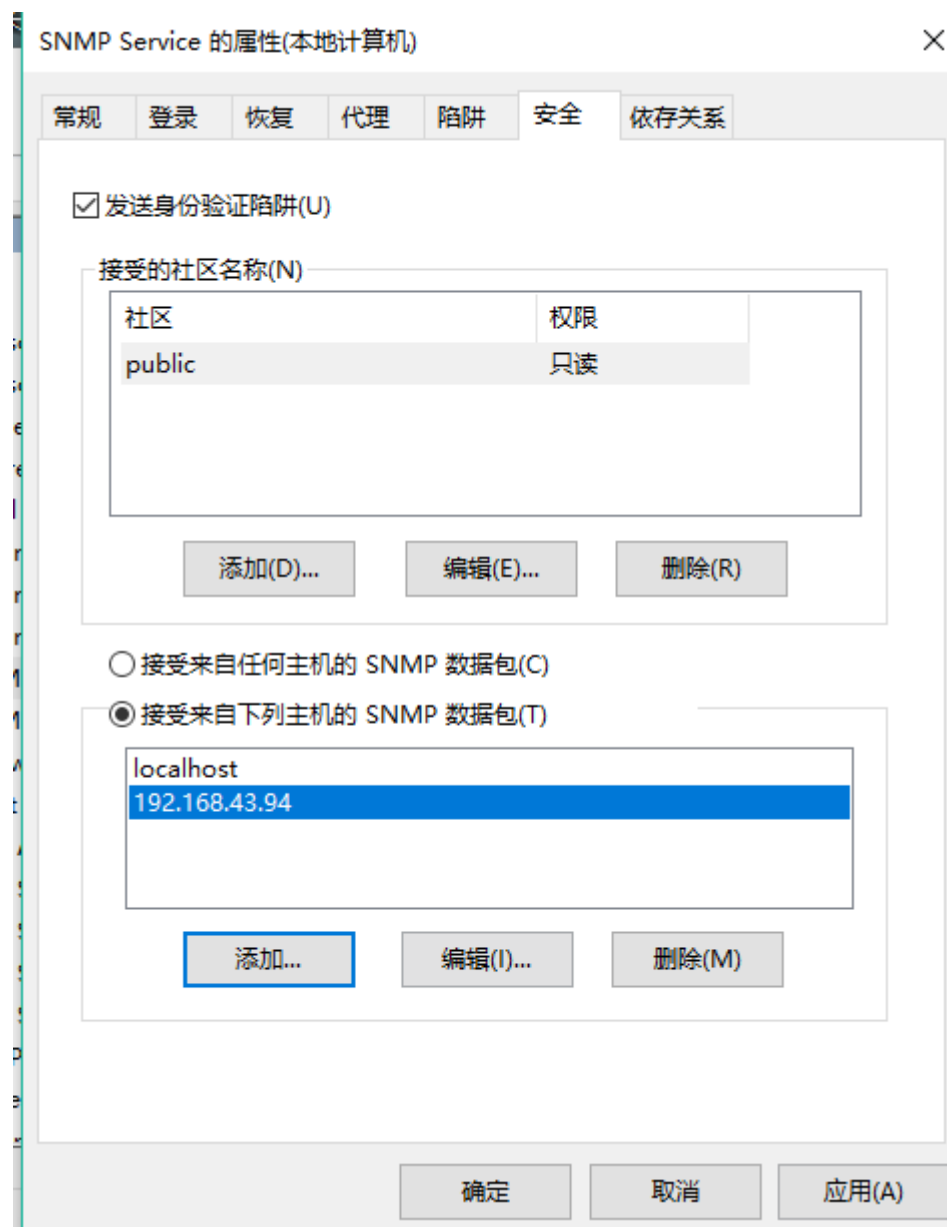


图 7 配置 SNMP 步骤 6  
接下来需要防火墙的高级设置，如下图所示。

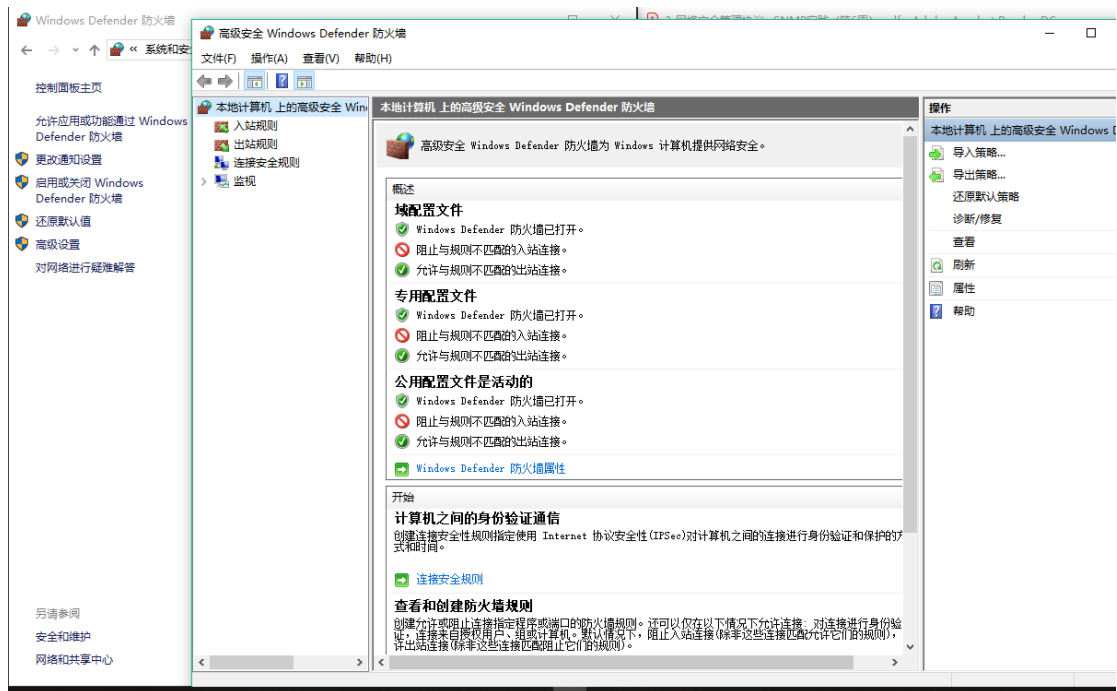


图 8 配置 SNMP 步骤 7

选择新建入站规则，并按下图中配置。

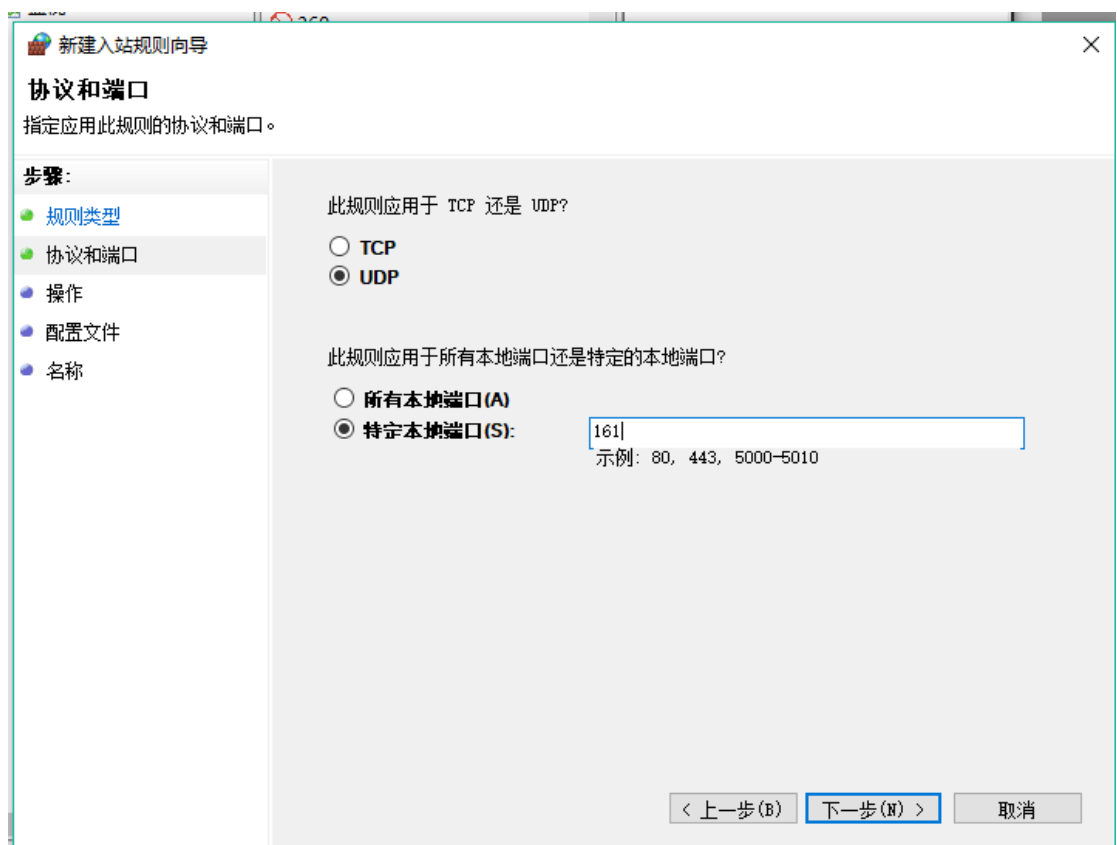


图 9 配置 SNMP 步骤 8

然后不断按下面几张图片中的选项那样点击下一步。



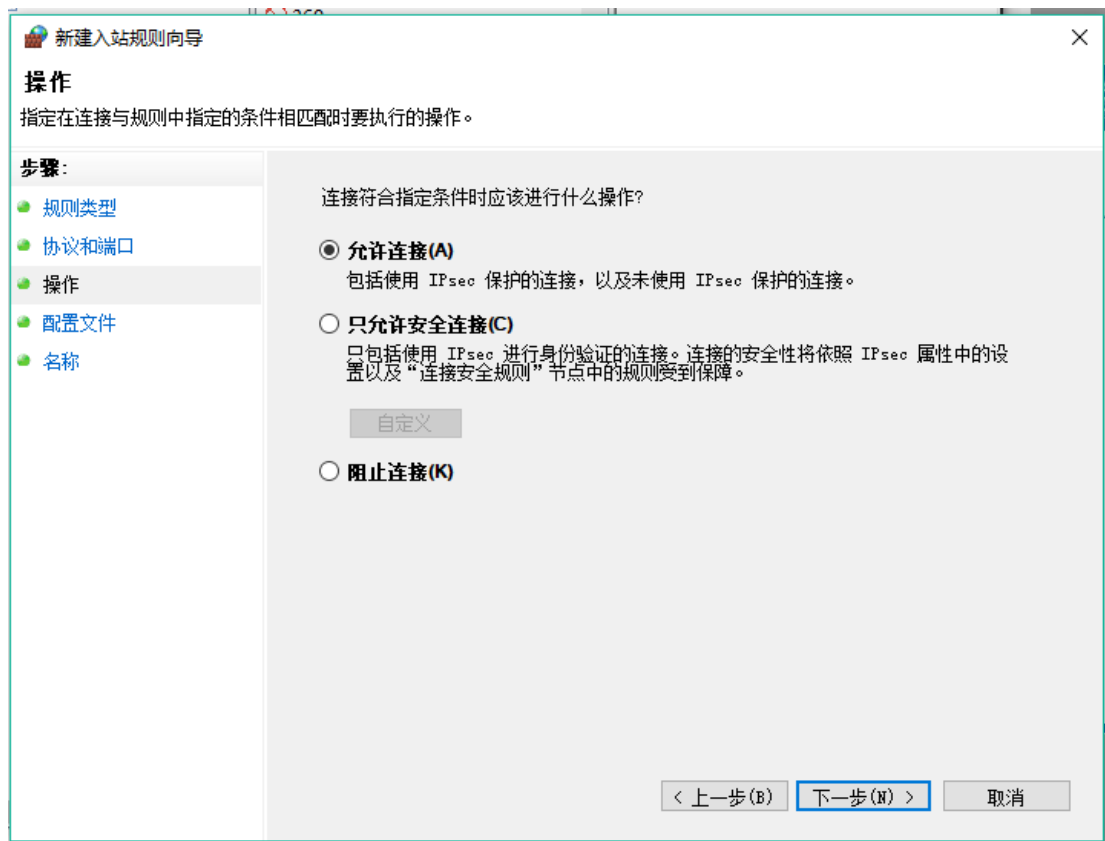


图 10 配置 SNMP 步骤 9

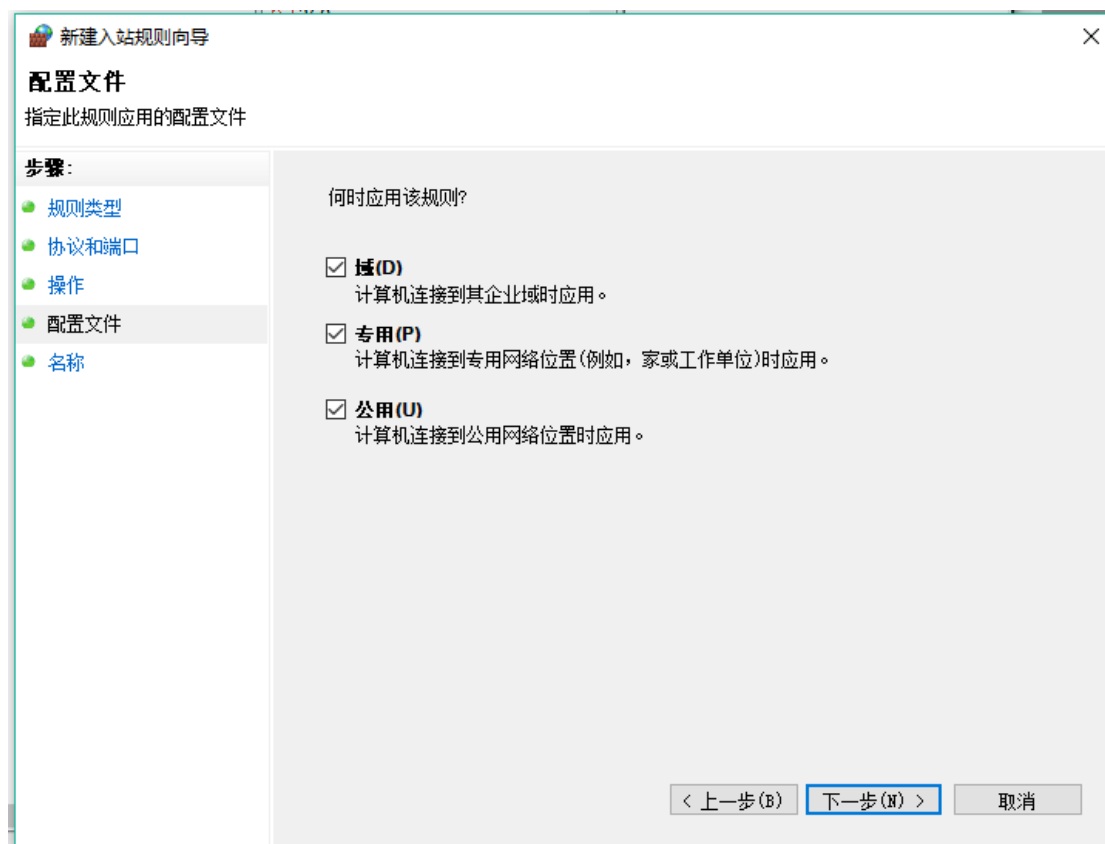


图 11 配置 SNMP 步骤 10

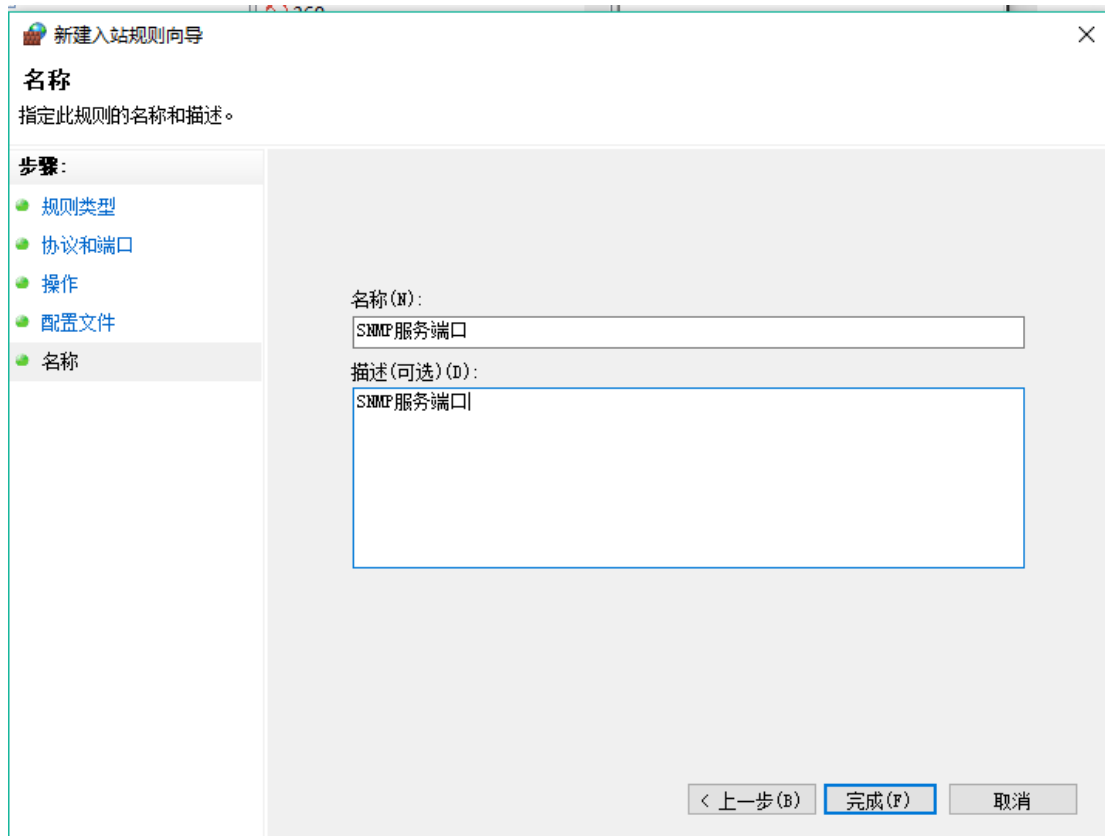


图 12 配置 SNMP 步骤 11

完成后，即可在入站规则中查看到刚才添加的新规则。

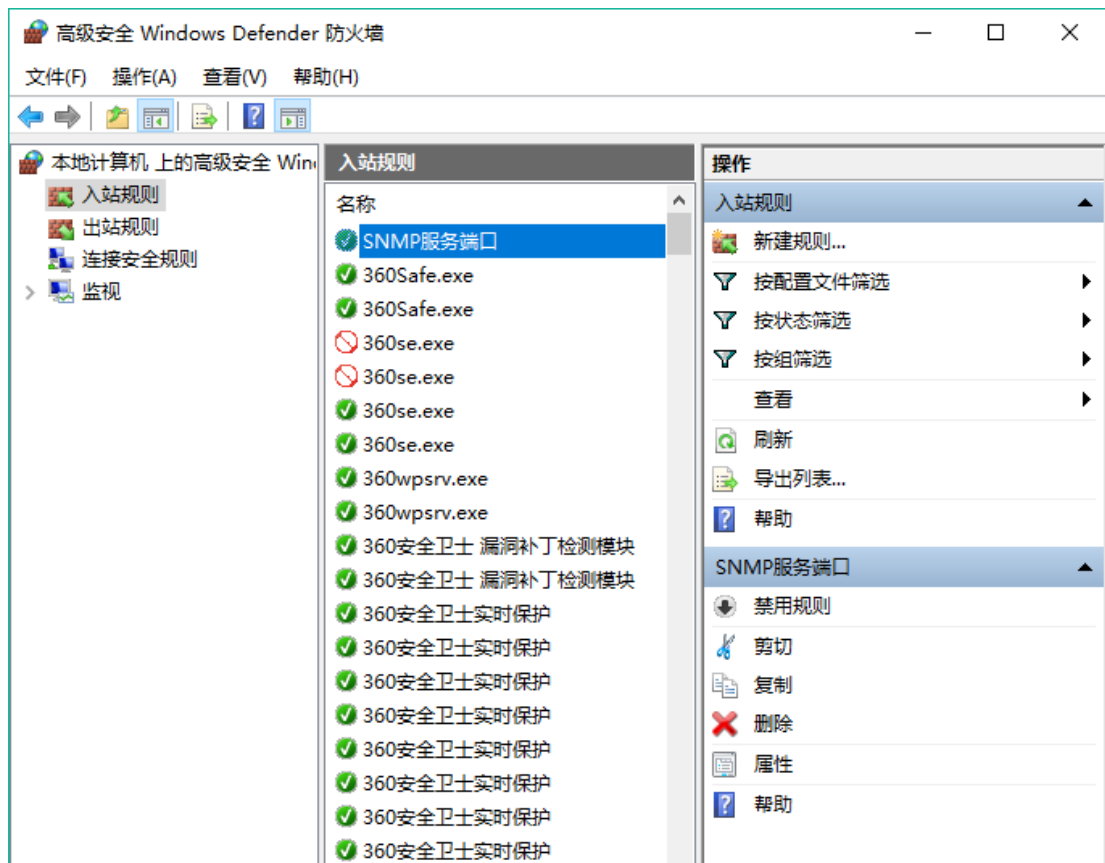


图 13 配置 SNMP 步骤 12

### 3.2 编制控制台程序, 接受用户输入的 OID 字符串, 返回在 Windows SNMP 代理中的对应值。

实验中的 GUI 界面程序是基于 Tkinter 库制作而成, 主要有四个功能: 响应输入的 OID 号, 观察主机各参数值, 设置阈值 (以 CPU 和内存使用率为例)、自动报警以及动态显示 CPU、内存利用曲线。

GUI 界面的主界面如下图所示。三个按键分别对应不同的模块, 其中, 设置阈值、自动报警模块是集成在观察主机各参数值的模块里面的。分别点击每个按键, 即可进入各个子模块。

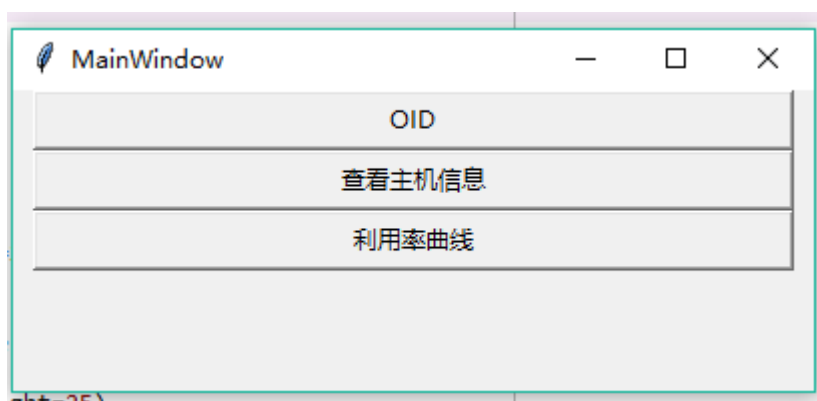


图 14 GUI 主界面

#### OID 响应界面:

本部分控制台程序被一并集成到了 GUI 界面程序中, 主要基于 net-snmp 来做的。在开始运行程序前, 要在环境变量中添加 net-snmp 的路径, 为文件夹 usr/bin, 每次从 StringVar() 控件中获取用户输入的 OID 号时, 调用系统命令行, 将相应的命令输入到命令行中, 并返回命令行的输出, 此过程均可以直接在命令行中操作, 如图 15 所示。本实验采用将其打印至 Text() 控件上。该部分主要提供了 get 方式和 walk 方式两种响应方式, 对应两种不同的命令, 可自行选取。当输入框中不输入任何数值时, 等于执行 `snmpget -v 2c -c public localhost` 或 `snmpwalk -v 2c -c public localhost`, 会出现非常详细的信息。

```
C:\Windows\System32\cmd.exe
Microsoft Windows [版本 10.0.17134.706]
(c) 2018 Microsoft Corporation. 保留所有权利。

C:\usr\bin>snmpwalk -v 2c -c public localhost 1.3.6.1.2.1.25.3.3.1.2
HOST-RESOURCES-MIB::hrProcessorLoad.4 = INTEGER: 12
HOST-RESOURCES-MIB::hrProcessorLoad.5 = INTEGER: 6
HOST-RESOURCES-MIB::hrProcessorLoad.6 = INTEGER: 8
HOST-RESOURCES-MIB::hrProcessorLoad.7 = INTEGER: 7
HOST-RESOURCES-MIB::hrProcessorLoad.8 = INTEGER: 8
HOST-RESOURCES-MIB::hrProcessorLoad.9 = INTEGER: 10
HOST-RESOURCES-MIB::hrProcessorLoad.10 = INTEGER: 15
HOST-RESOURCES-MIB::hrProcessorLoad.11 = INTEGER: 5

C:\usr\bin>snmpdf -v 1 -c public localhost
Description size (kB) Used Available Used%
C:\ Label: OS Serial Number 56ff7a3f 127366800 97692516 29674284 76%
D:\ Label: Serial Number 52bc6ef1 204799996 199001600 5798396 97%
E:\ Label: Serial Number c6c0b5f4 307199996 263484096 43715900 85%
F:\ Label: Serial Number lac47a07 323364860 143378464 179986396 44%
G:\
Virtual Memory 9124544 7291520 1833024 79%
Physical Memory 8293696 5203968 3089728 62%

C:\usr\bin>
```

图 15 直接在命令行中使用 net-snmp 提供的指令

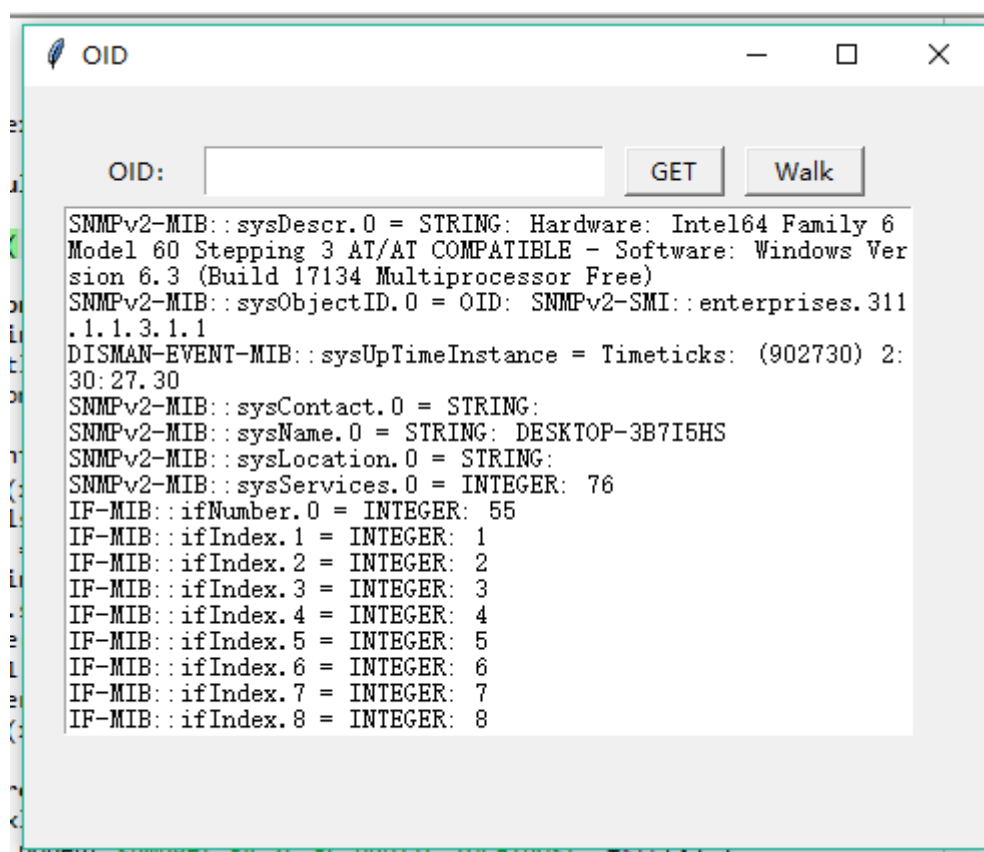


图 16 不输入 OID 的情况

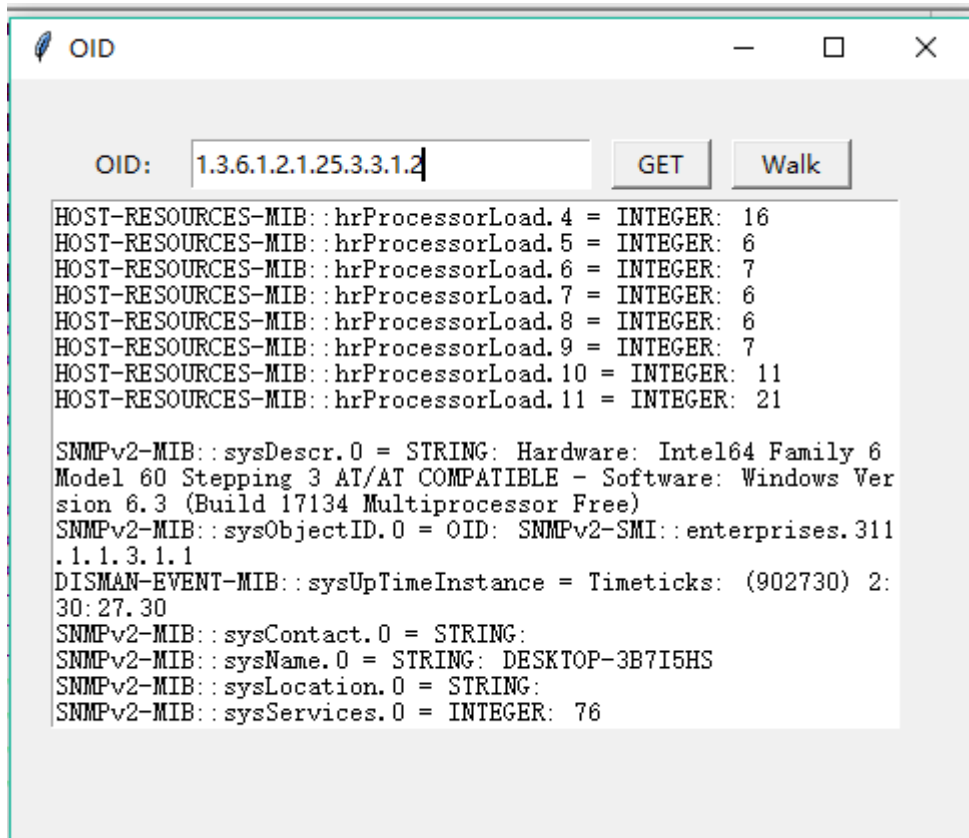


图 17 输入获取 CPU 状态的 OID

### 3.3 开发 GUI 界面程序，使用户可通过控制台程序观察主机 CPU、内存、硬盘空间、流量值。

本部分程序同样是基于 net-snmp 来做的。进入该模块后，当检测到点击“开始刷新”按键的事件时，打开一个新的线程，设置一个定时器，每过 2s（该刷新时间可以设置，改变定时器刷新时间即可）使用 snmp，从主机系统中获取当前的 CPU、内存、硬盘、流量值等信息，并将返回值实时刷新到界面的 Text() 控件上。对于 CPU 利用率，主要使用 snmpwalk 方式，从 OID 为 1.3.6.1.2.1.25.3.3.1.2 的节点获取；对于内存和硬盘的相关信息，主要使用 snmpdf 命令来获取；而对于流量值信息，主要使用 snmpwalk 方式，从 OID 为 1.3.6.1.2.1.2.2.1.10 和 1.3.6.1.2.1.2.2.1.16 的节点获取。每次刷新获取后，将需要的信息打印到 Text() 上即可。当点击“停止刷新”后，则各参数信息不再刷新，显示为上一次的刷新值。

在每次提取出命令行中的信息时，都要先提前查看所需信息的位置，比如在第几行的那几个位置，然后在程序中通过读取这些位置的信息即可，所以对于几乎每个指令，都要仔细去数清位置，这一点非常麻烦而耗时。

结果如下图 18 所示。通过测试可以发现，点击“开始刷新”之后，每隔 2s 时间，显示值刷新一次，当前主机的各参数信息即被实时显示在界面上。而点击“停止刷新”后，参数信息不再刷新。

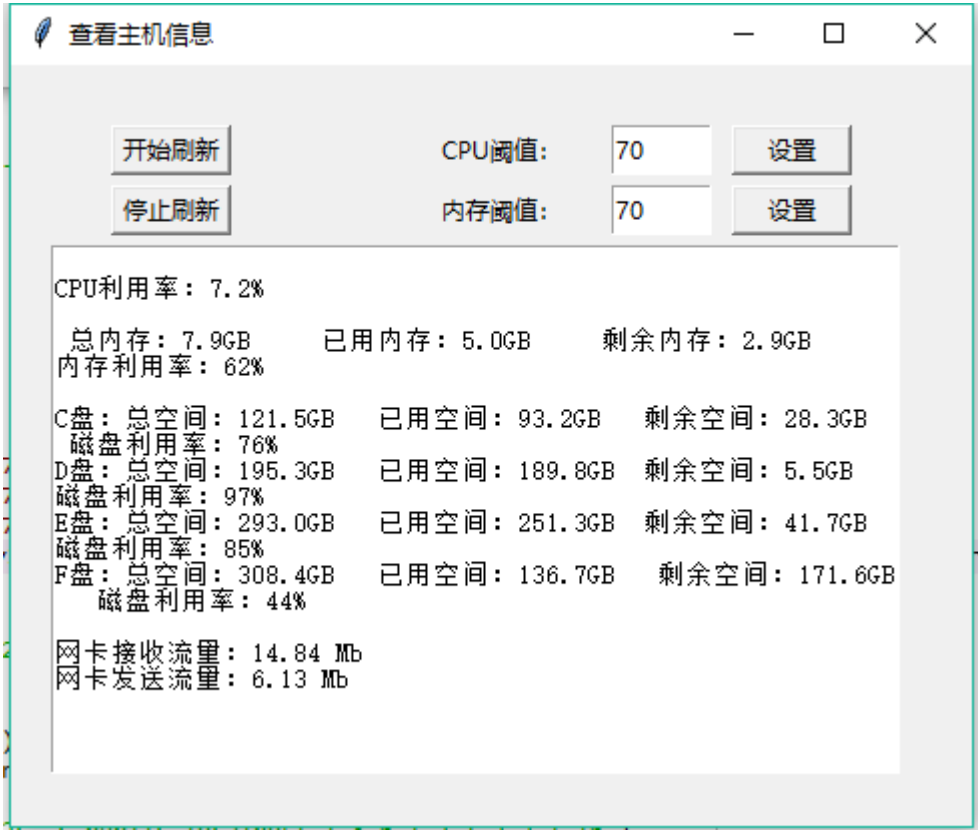


图 18 获得主机信息

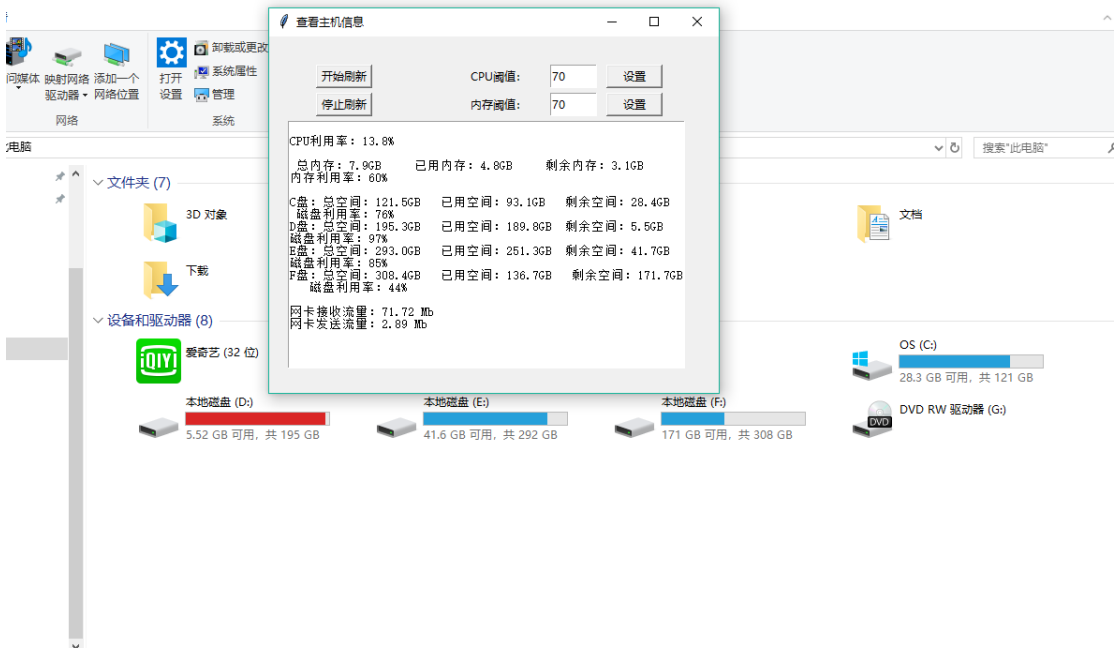


图 19 磁盘信息验证

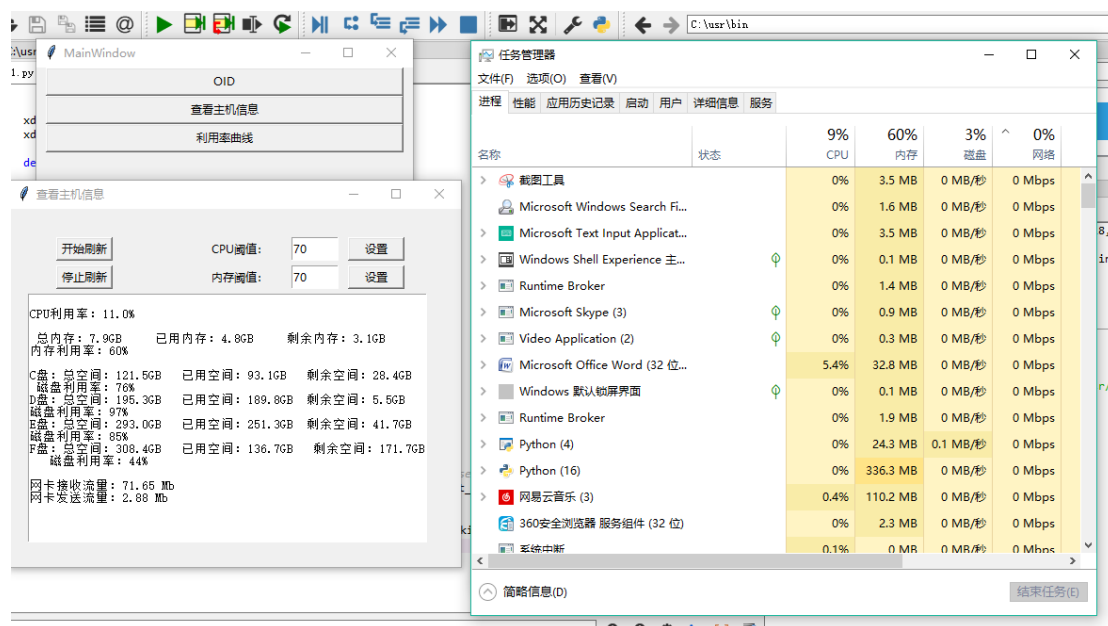


图 20 多个主机信息验证

### 3.4 开发阈值告警功能,用户通过界面可设置性能阈值(CPU 和内存),当超过阈值时自动报警。

本部分程序被内置在查看参数信息的程序界面中,通过设置两个 `StringVar()` 控件,分别对应 CPU 利用率阈值和内存利用率阈值,并实时获取用户输入的两种阈值。调用之前模块中已经获取的 CPU 利用率及内存利用率,定时器每次刷新时,都将阈值与其做比较,若其高于阈值,则分别调用相应的 Tkinter 中的警示框,提示用户该值已经超过所设阈值。本程序中将两种阈值初始值都设为 70%。

结果如下图 21、图 22 所示,当设置两种阈值之后,每当利用率高于阈值时,都会出现如图所示,自带音效的警示框。

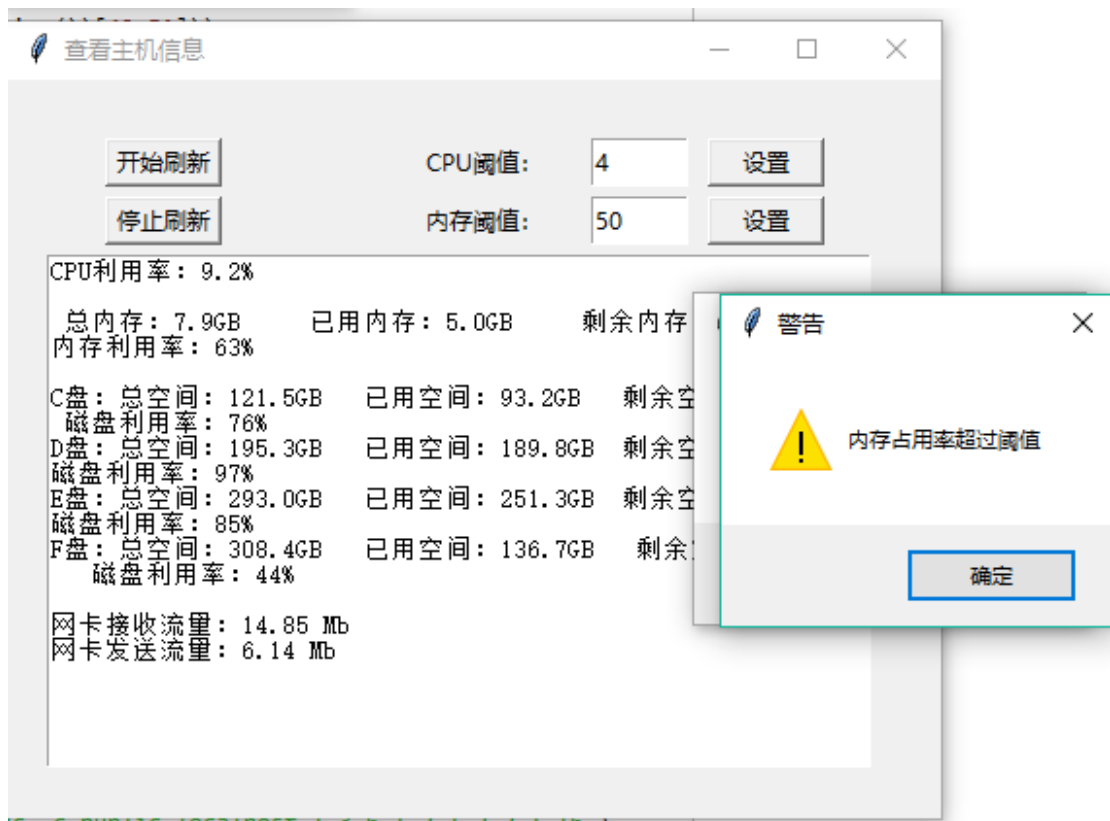


图 21 内存占用率过高

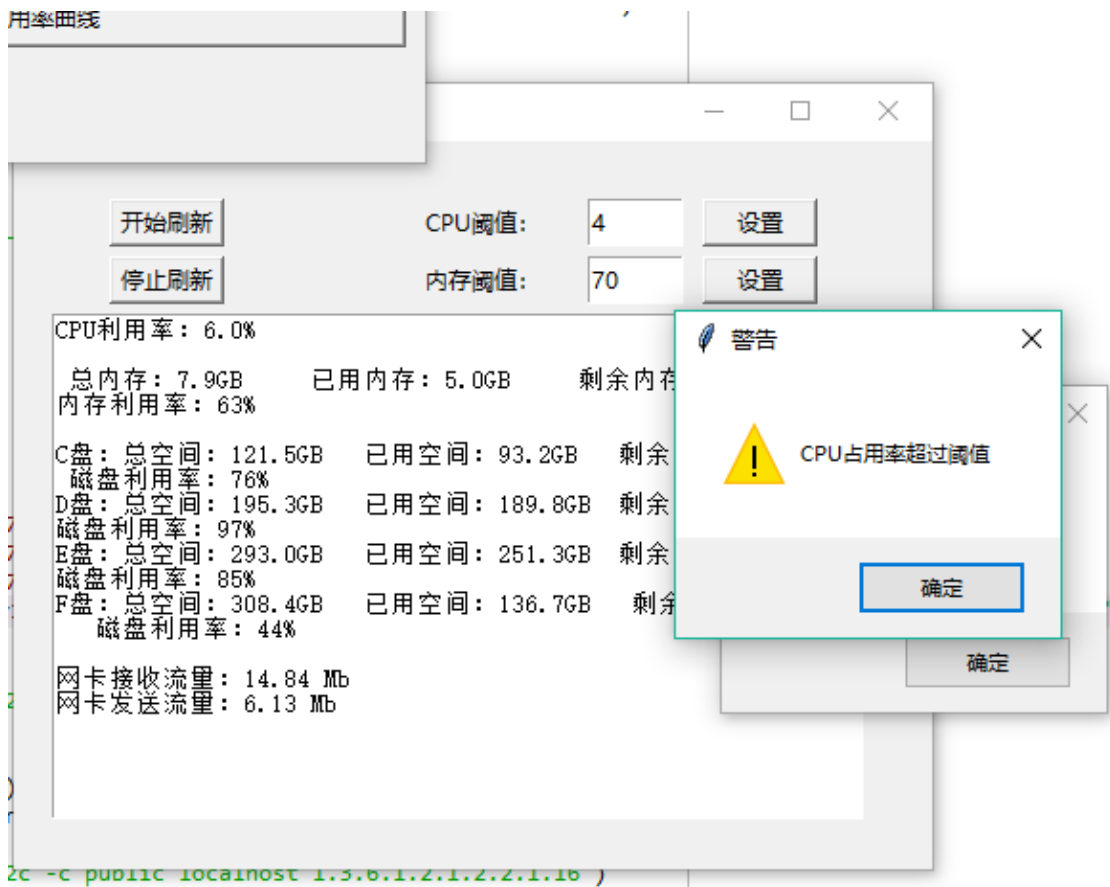


图 22 CPU 占用率过高



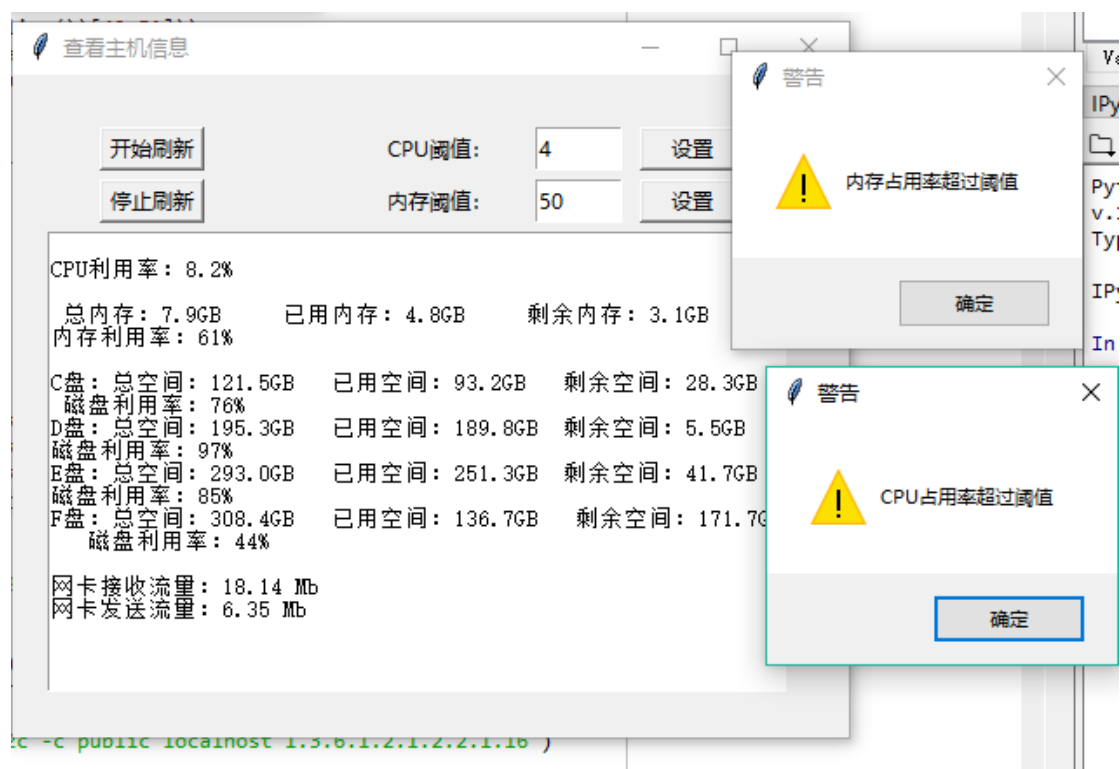


图 23 内存及 CPU 占用率同时过高

### 3.5 实现 SNMP Trap、Set 功能并测试。

#### Trap:

本部分同样使用 net-snmp 的 snmptrap 来实现,需要对接收端和发送端进行配置。我以自己的电脑 (192.168.43.94) 作为接收端,以另一台电脑 (192.168.43.2) 作为发送端。

首先配置接收端。接收端电脑任意路径下新建一个配置文件 snmpd.conf,为了方便接收所有的 trap,在里面添加配置 disableAuthorization yes,然后打开命令行,在其内运行接收 Trap 的程序,命令为

```
snmptrapd -c C:\usr\etc\snmp\snmptrapd.conf -f -Le -d
```

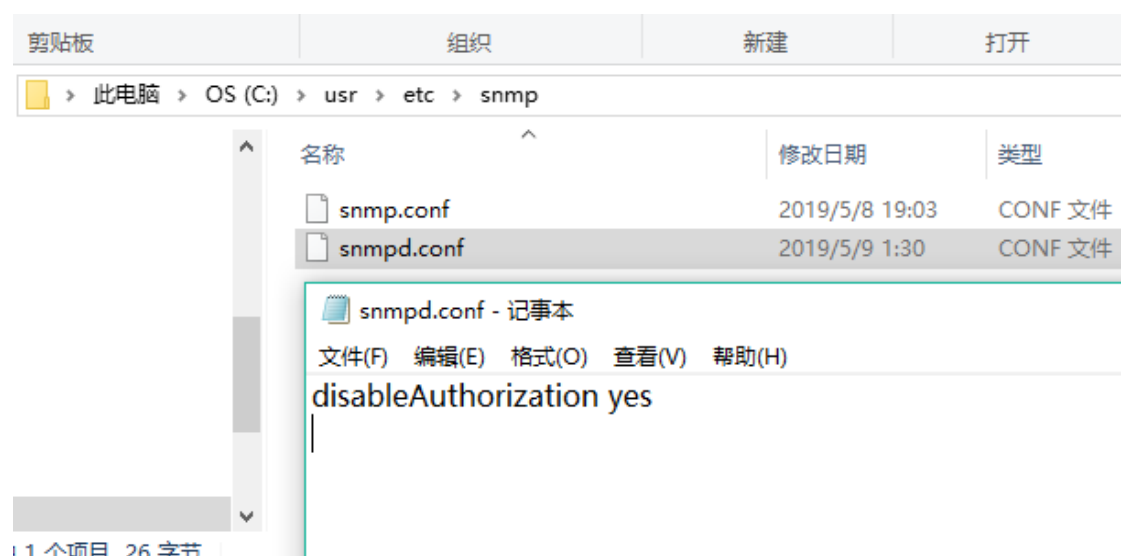


图 24 配置文件 snmpd.conf

此时即可进入 Trap 接收模式，等待发送端发送消息，如下图所示。

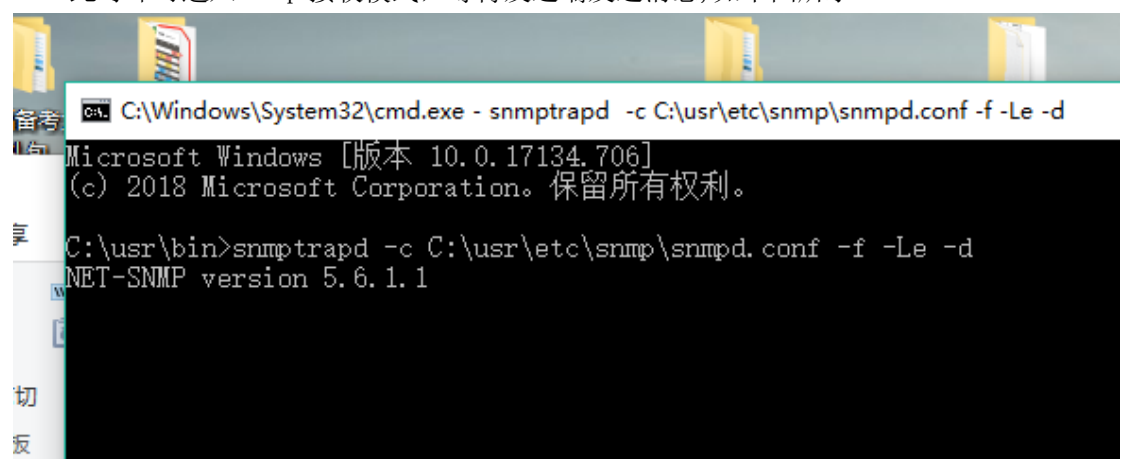


图 25 等待发送端发送消息

同样按照之前的配置方法，配置好发送端电脑的 snmp，并安装 net-snmp，然后进入命令行中，输入命令：

```
C:\usr\bin\snmptrap.exe -v 1 -c public 192.168.43.94 .1.3.6.1.4.1.229
4.3 192.168.8.74 6 100 12345 .1.3.6.1.4.1.2294.3.1.4 s SNMPTrap .1.3.6.
1.4.1.2294.3.1.5 i 1122 .1.3.6.1.4.1.2294.3.1.6 s SNMPTrap
```

如下图所示。

```
C:\usr\bin>snmpget -v 2c -c public localhost .1.3.6.1.2.1.1.5.0
SNMPv2-MIB::sysName.0 = STRING: 王煜童的laptop
C:\usr\bin>
```

图 26 发送端信息

```
C:\Windows\System32\cmd.exe
Microsoft Windows [版本 10.0.17134.706]
(c) 2018 Microsoft Corporation。保留所有权利。

C:\usr\bin>snmptrap.exe -v 1 -c public 192.168.43.94 .1.3.6.1.4.1.2294.3 192.168.8.74 6 100 12345 .1.3.6.1.4.1.2294.3
.1.4 s SNMPtrap .1.3.6.1.4.1.2294.3.1.5 i 1122 .1.3.6.1.4.1.2294.3.1.6 s SNMPtrap
No log handling enabled - using stderr logging
Created directory: C:/usr/snmp/persist/mib_indexes

C:\usr\bin>
```

图 27 配置发送端

此时，可以看到，接收端已经获得了上报的 Trap 信息，如下图所示。

```
C:\Windows\System32\cmd.exe - snmptrapd -c C:\usr\etc\snmp\snmpd.conf -f -Le -d
Microsoft Windows [版本 10.0.17134.706]
(c) 2018 Microsoft Corporation。保留所有权利。

C:\usr\bin>snmptrapd -c C:\usr\etc\snmp\snmpd.conf -f -Le -d
NET-SNMP version 5.6.1.1

Received 109 byte packet from UDP: [192.168.43.2]:49692->[0.0.0.0]:0
0000: 30 6B 02 01 00 04 06 70 75 62 6C 69 63 A4 5E 06   Ok....public.^
0016: 08 2B 06 01 04 01 91 76 03 40 04 C0 A8 08 4A 02   .+....v.@....J.
0032: 01 06 02 01 64 43 02 30 39 30 42 30 16 06 0A 2B   ....dC.090B0...+
0048: 06 01 04 01 91 76 03 01 04 04 08 53 4E 4D 50 74   ....v....SNMPt
0064: 72 61 70 30 10 06 0A 2B 06 01 04 01 91 76 03 01   rap0...+....v..
0080: 05 02 02 04 62 30 16 06 0A 2B 06 01 04 01 91 76   ....b0...+....v
0096: 03 01 06 04 08 53 4E 4D 50 74 72 61 70           ....SNMPtrap

2019-05-09 01:04:07 192.168.8.74(via UDP: [192.168.43.2]:49692->[0.0.0.0]:0) TRAP,
SNMP v1, community public
    SNMPv2-SMI::enterprises.2294.3 Enterprise Specific Trap (100) Uptime: 0:02:
03.45
    SNMPv2-SMI::enterprises.2294.3.1.4 = STRING: "SNMPtrap" SNMPv2-SMI::enterpr
ises.2294.3.1.5 = INTEGER: 1122 SNMPv2-SMI::enterprises.2294.3.1.6 = STRING: "SNMPt
rap"
```

图 28 发送端接收消息

#### Set:

在进行 set 功能测试前，需要特别注意应将最开始设置的只读模式改为读写模式，并添加 private 社区，不然一定会出现 Timeout 提示。

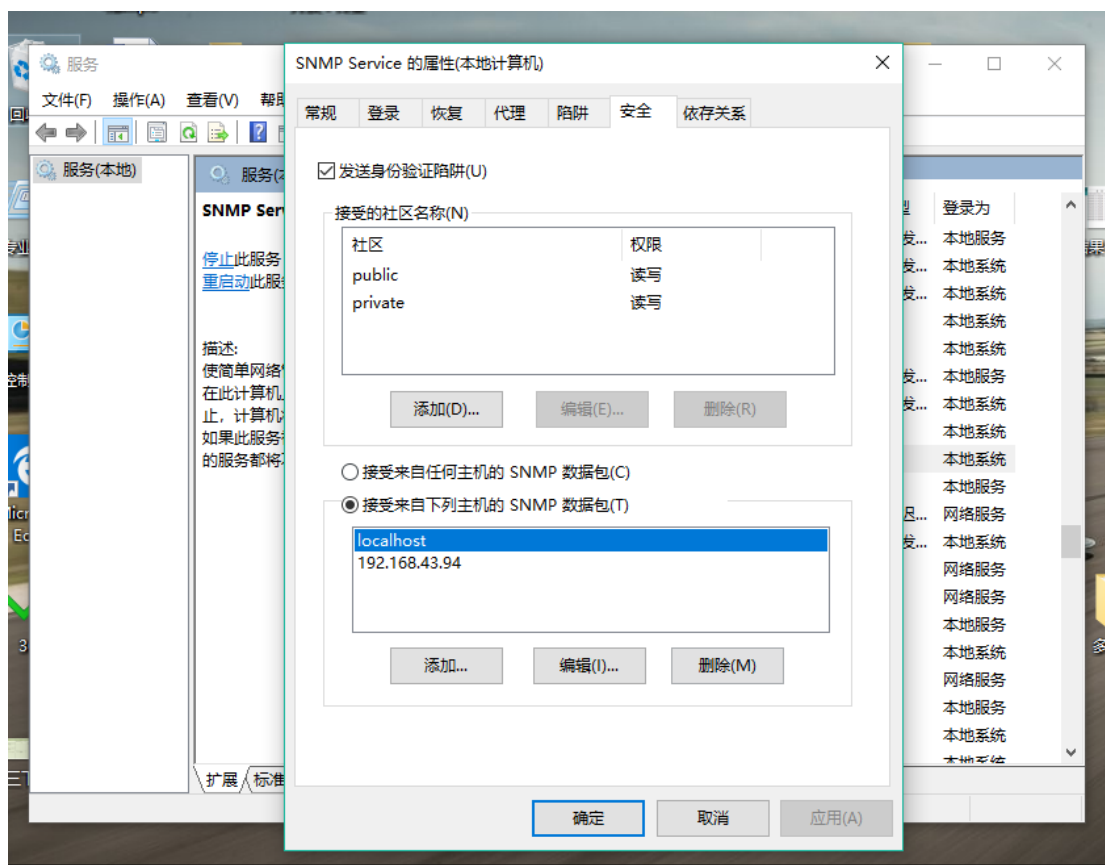


图 29 重新配置 SNMP Service

当用下图所示查看主机名和修改主机名的指令成功时，也说明 set 功能测试成功。

```
C:\usr\bin>snmpset -v 2c -c private localhost sysName.0 s abc
SNMPv2-MIB::sysName.0 = STRING: abc

C:\usr\bin>snmpwalk -v2c -c public localhost sysname
SNMPv2-MIB::sysName.0 = STRING: abc

C:\usr\bin>snmpset -v 2c -c private localhost sysName.0 s MyComputer
SNMPv2-MIB::sysName.0 = STRING: MyComputer

C:\usr\bin>snmpwalk -v2c -c public localhost sysname
SNMPv2-MIB::sysName.0 = STRING: MyComputer

C:\usr\bin>snmpset -v 2c -c private localhost sysName.0 s 516021910796
SNMPv2-MIB::sysName.0 = STRING: 516021910796

C:\usr\bin>snmpwalk -v2c -c public localhost sysname
SNMPv2-MIB::sysName.0 = STRING: 516021910796
```

图 30 Set 功能成功测试

### 3.6 在控制台动态显示主机 CPU、内存利用曲线。

该部分程序主要使用 python 的 matplotlib 作图，每隔一段时间，调用上述模块已经实现的函数，获取当前的 CPU 利用率以及内存利用率，作为两个子图的纵坐标，横坐标则以刷新时间为一个刻度，每次获取两个点后，则将其添加至存储的列表中，并在图中画出折线，即可实现动态显示 CPU 与内存的利用率曲线。由于 matplotlib 做的图可以实时显示当前鼠

标所指位置的点的值，因此可以很方便的查看每个时刻的具体 CPU、内存利用率。

需要特别注意的是，在 spyder 中直接作图无法出现动态效果，需要更改一下 Tools 中 Preferences 中的 IPython console，将其中的 Graphics backend 改为 Qt5 即可。

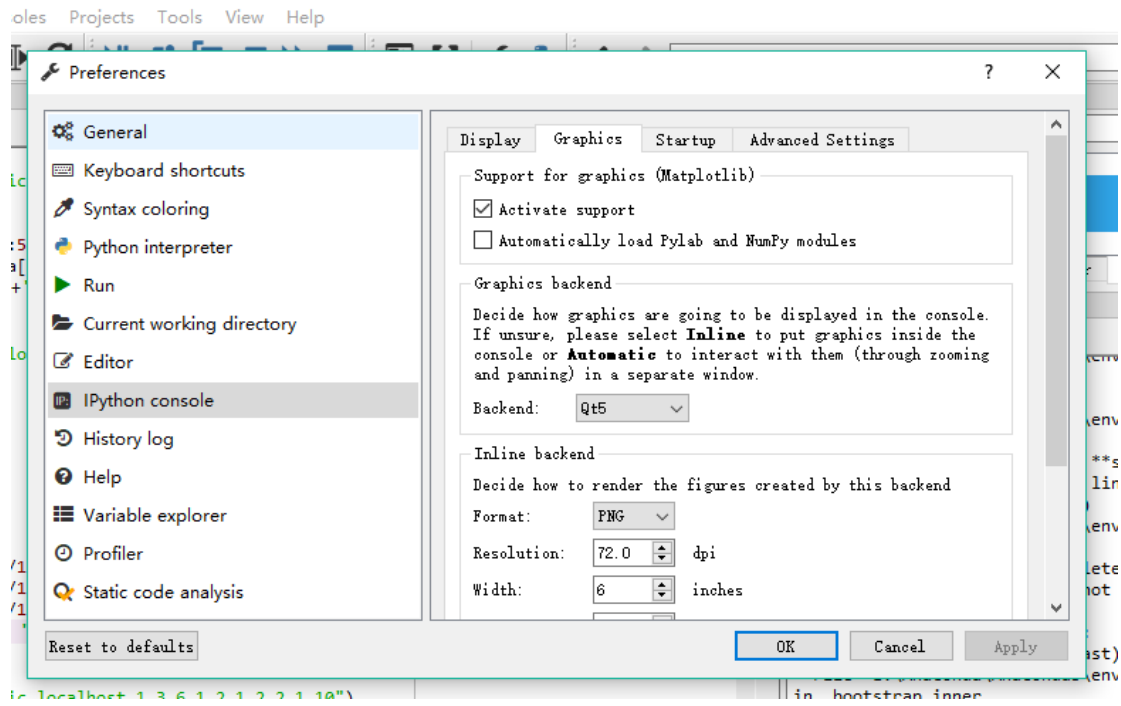


图 31 配置 Graphics backend

结果如下图所示，正常状况下的 CPU、内存利用率都比较低，而在跑了机器学习代码之后，CPU 与内存利用率都出现了较大增长。

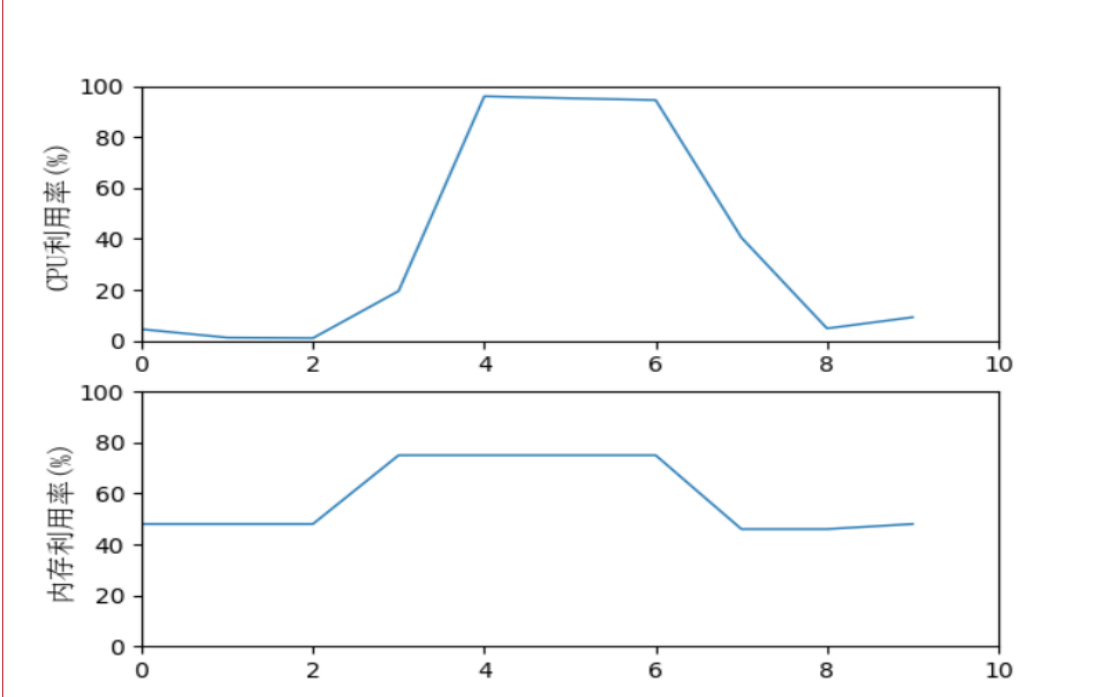


图 32 动态效果

全部功能效果图如下。

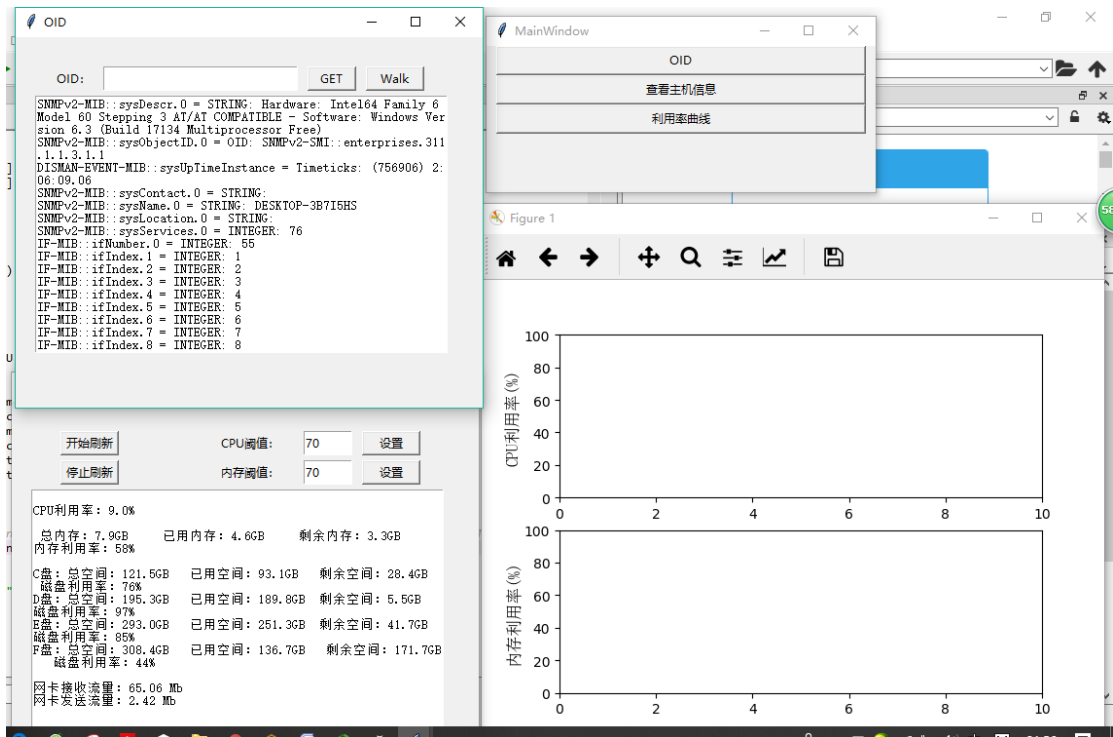


图 33 全部效果

## 4. 心得体会

这次实验新安装了一些从来没用过的软件，虽然对于 windows 系统来说网络上有很多可以参考的想法，但是中间还是遇到了很多花很多时间去解决的问题。这是我第一次接触 SNMP，通过这次实验，对 SNMP 有了较深入的了解，并亲自体验了一些操作流程，感觉从中学到了很多新知识和技能，以及新软件的使用方法。不过，也相对花了很多时间来解决遇到的问题，但是这个过程本身也是一个锻炼自己解决问题的能力过程，所以花掉的时间也不算浪费，为未来积累了更多工程经验。

并且还发现，有一些平时听课不一定能完全理解的东西，在几天的反复调试中会大大增加理解的程度。还有些地方，平时感觉记住了，在真正开始做作业时却又要反反复复查很多资料才能真正会用，这都是不断进步与提高的过程，所以虽然感觉这个过程效率很低，但又感觉花掉的时间还是有价值的。并且我意识到在自己毫无头绪时，不应只是一个人独自思考，还应多与其他人交流，也许其他人觉得很正常的东西正是自己从未想到的，只有在自己不断思考的同时保持与他人的交流，才能更加全面的掌握一门技术或得到快速的提高。

SNMP 虽然叫做简单网络管理协议，但是其实并不简单，我们目前基本都是基于别人已经做好的 net-snmp 来实现相应功能，要做到完全掌握协议，并开发复杂协议，还有很长的路要走。在面对一个大问题时，一个人的力量是有限的，和别人的交流有时候能避免走很多的弯路，也能为解决问题增添很多种可能。GUI 界面的编写看起来容易，做起来却并不是那

么简单。再难的问题，总还是有解决方法的，一种途径不行就换一种途径，只要肯尝试，总是能解决问题的，因此在面对难题时一定要不气馁，坚持不懈。

## 5. 遇到问题及解决思路

1. 在进行 SNMP Set 功能测试时，发现想更改主机名称时总是会发生 Timeout 的问题，搜集了很多资料依然没能解决，后来在同学的提醒下意识到应该在 SNMP Service 中将社区的权限改为读写并增加 private，经过修改后，解决了问题。

2. 最初写好大体框架后，每次运行总是报错不能使用 net-snmp 的命令，后来添加了环境变量，并发现可以直接用命令行成功运行命令后，意识到在 spyder 上运行不成功一定是路径问题，后来将 py 文件一起放到 bin 文件中后即可顺利运行。

3. 在每次提取出命令行中的信息时，都要先提前查看所需信息的位置，比如在第几行的那几个位置，然后在程序中通过读取出这些位置的信息即可，所以对于几乎每个指令，都要仔细去数清位置，这一点非常麻烦而耗时。虽然后来与同学交流后，发现其实可以直接读出三个 ‘:’ 之后的数据到 end 即可，不禁感叹自己写程序时远不够灵活，花费了很大时间却还不能通用，但是由于花了这么多时间用来数位置。。。所以也没有再打算更改代码，因为这些数据目前也可正常显示。

4. 在超过阈值报警时，写完程序曾报了大量错误，当时会逐条查询，但是始终不知错在了哪里就停了一段时间，后来重新开机后的偶然尝试发现不再报错了。。。这是什么原因我至今也没有发现，不过感觉自己还是很幸运的。

5. 一开始在 spyder 中直接作图无法出现动态效果，得到的曲线只是静态的，后来在网上发现有人碰上过类似问题，是因为 spyder 的 console 中本身就无法展现出动图，后来通过更改 Tools 中 Preferences 中的 IPython console，将其中的 Graphics backend 改为 Qt5 即可使用。

7. 由于对于 Tkinter 库不太熟悉，一开始直接使用最简单的 pack（）函数布局，结果发现界面无法像我想的那样进行排版。后来又尝试了 grid（）函数布局。虽然大体布局排版合理了，但是会出现很多空格及不对齐的情况，十分难看。最后才了解到精准布局的 place（）函数，使得界面稍微美观了一些。

8. 发现不在同一个局域网下，直接用命令行查出来的 IP 无法完成相应配置。后来选择了同在一个局域网的室友的电脑，解决了该问题。

9. tkinter 中，定时器必须存在于 mainloop() 之中。我的 GUI 程序由于有子模块，相当于初

始化了新的窗口，于是必须设置新的子窗口的 `mainloop()`。由于没加定时器之前，这个问题得不到体现，因此一直没注意到，以至于后来出现问题时一头雾水，思考了很久才找到问题所在。

## 6. 附录（代码）

```
# -*- coding: utf-8 -*-
"""

Created on Wed May  8 22:05:39 2019

@author: 回到未来
"""

import tkinter

import os

import psutil

import threading

import time

import psutil as p

import matplotlib.pyplot as plt

import matplotlib.font_manager as font_manager

from tkinter import simpledialog

import tkinter.messagebox

import numpy as np

import matplotlib.animation as animation

from matplotlib.font_manager import FontProperties

#localhost=['192.168.43.94']

root = tkinter.Tk()    #主窗口

root.title("MainWindow")

root.geometry('400x150')
```



```
oid=""
```

```
global xls_text
```

```
def printresult(x):
```

```
    global t
```

```
    t.insert('1.0', x)
```

```
def OID_response():
```

```
    root1=tkinter.Tk()
```

```
    root1.title("OID")
```

```
    root1.geometry('480x380')
```

```
    l1 = tkinter.Label(root1, text="OID: ")
```

```
    l1.place(x = 40, y = 30, width=40, height=25)
```

```
    global xls_text
```

```
    xls_text = tkinter.StringVar(root1)
```

```
    xls = tkinter.Entry(root1, textvariable = xls_text)
```

```
    xls_text.set("")
```

```
    xls.place(x = 90, y = 30, width=200, height=25)
```

```
    global t1
```

```
    t1=tkinter.Text(root1,height=20, width=60)
```

```
    t1.place(x = 20, y = 60)
```

```
def get_resonse():
```

```
    x = xls_text.get()
```

```
    p=os.popen("snmpget -v 2c -c public localhost "+str(x) )
```

```
    t1.insert('1.0', p.read()+'\n')
```

```

def walk_resonse():

    x = xls_text.get()

    p=os.popen("snmpwalk -v 2c -c public localhost "+str(x) )

    t1.insert('1.0',p.read()+'\n')


tkinter.Button(root1, text="GET", command = get_resonse).place(x = 300, y = 30,
width=50, height=25)

tkinter.Button(root1, text="Walk", command = walk_resonse).place(x = 360, y =
30, width=60, height=25)

C_yuzhi=80
M_yuzhi=80

def every_vaule():

    global root2

    root2=tkinter.Tk()

    root2.title("查看主机信息")

    root2.geometry('480x380')

    global t

    t=tkinter.Text(root2,height=20, width=60)

    t.place(x = 20, y = 90)

    tkinter.Button(root2, text="开始刷新", command = fun_timer).place(x = 50, y =
30, width=60, height=25)

    l1 = tkinter.Label(root2, text="CPU 阈值: ")

    l1.place(x = 200, y = 30, width=90, height=25)

    global cpu_yuzhi

```

```

def getCPU_yuzhi():
    global C_yuzhi
    C_yuzhi=cpu_yuzhi.get()

def getMEM_yuzhi():
    global M_yuzhi
    M_yuzhi=mem_yuzhi.get()

cpu_yuzhi = tkinter.StringVar(root2)
cpu_ = tkinter.Entry(root2, textvariable = cpu_yuzhi)
cpu_yuzhi.set("70")
cpu_.place(x = 300,y = 30, width=50, height=25)

tkinter.Button(root2, text="设置", command = getCPU_yuzhi).place(x = 360,y =
30, width=60, height=25)

tkinter.Button(root2, text="停止刷新", command = stop_timer).place(x = 50,y =
60, width=60, height=25)

l2 = tkinter.Label(root2, text="内存阈值: ")
l2.place(x = 200,y = 60,width=90,height=25)

global mem_yuzhi
mem_yuzhi = tkinter.StringVar(root2)
mem_ = tkinter.Entry(root2, textvariable = mem_yuzhi)
mem_yuzhi.set("70")
mem_.place(x = 300,y = 60, width=50, height=25)

tkinter.Button(root2, text="设置", command = getMEM_yuzhi).place(x = 360,y =
60, width=60, height=25)

root2.mainloop()

```

```

def cipankongjian():
    p=os.popen("snmpdf -v 1 -c public localhost")
    a=[]
    a.append(p.readline())
    a.append(p.readline())
    a.append(p.readline())
    a.append(p.readline())
    a.append(p.readline())
    a.append(p.readline())
    a.append(p.readline())
    a.append(p.readline())
    memory1=format((float(a[1][43:59])/1024/1024),'.1f')
    memory2=format((float(a[1][59:75])/1024/1024),'.1f')
    memory3=format((float(a[1][75:84])/1024/1024),'.1f')
    memory4=format((float(a[2][41:50])/1024/1024),'.1f')
    memory5=format((float(a[2][57:66])/1024/1024),'.1f')
    memory6=format((float(a[2][73:82])/1024/1024),'.1f')
    memory7=format((float(a[3][41:50])/1024/1024),'.1f')
    memory8=format((float(a[3][57:66])/1024/1024),'.1f')
    memory9=format((float(a[3][73:82])/1024/1024),'.1f')
    memory10=format((float(a[4][41:50])/1024/1024),'.1f')
    memory11=format((float(a[4][57:66])/1024/1024),'.1f')
    memory12=format((float(a[4][73:82])/1024/1024),'.1f')
    x1="D 盘: "+"总空间: "+str(memory4)+"GB    "+"已用空间: "+str(memory5)+"GB    "+"
    剩余空间: "+str(memory6)+"GB    "+"磁盘利用率: "+a[2][85:88]+' \n'
    x2="C 盘: "+"总空间: "+str(memory1)+"GB    "+"已用空间: "+str(memory2)+"GB
    "+"剩余空间: "+str(memory3)+"GB    "+"磁盘利用率: "+a[1][87:90]+' \n'
    x7="E 盘: "+"总空间: "+str(memory7)+"GB    "+"已用空间: "+str(memory8)+"GB    "+"
    剩余空间: "+str(memory9)+"GB    "+"磁盘利用率: "+a[3][85:88]+' \n'

```

```

x8="F 盘: "+"总空间: "+str(memory10)+"GB    "+"已用空间: "+str(memory11)+"GB
"+"剩余空间: "+str(memory12)+"GB    "+"磁盘利用率: "+a[4][85:88]+' \n'

```

```

#CPU():

```

```

p=os.popen("snmpwalk -v 2c -c public localhost 1.3.6.1.2.1.25.3.3.1.2")
a=[]
for i in range(4):
    a.append(int((p.readline())[49:51]))
cpu_usage=format(float((a[0]+a[1]+a[2]+a[3])/4),'.1f')
x3="CPU 利用率: "+str(cpu_usage)+'%'+' \n'

```

```

#memory():

```

```

p=os.popen("snmpdf -v 1 -c public localhost")
a=[]
a.append(p.readline())
a.append(p.readline())
a.append(p.readline())
a.append(p.readline())
a.append(p.readline())
a.append(p.readline())
a.append(p.readline())
a.append(p.readline())
memory1=format((float(a[7][27:34])/1024/1024),'.1f')
memory2=format((float(a[7][43:50])/1024/1024),'.1f')
memory3=format((float(a[7][59:66])/1024/1024),'.1f')
x4=" 总内存: "+str(memory1)+"GB    "+"已用内存: "+str(memory2)+"GB    "+"
剩余内存: "+str(memory3)+"GB    "+"内存利用率: "+a[7][69:73]

```

```

#liuliangzhi():

```

```

p=os.popen("snmpwalk -v 2c -c public localhost 1.3.6.1.2.1.2.2.1.10")

```

```

a=[]

for j in range(5):
    a.append(p.readline())

bytes_re=' {0:.2f} Mb'.format(float(a[4][35:45])/1024/1024)


p=os.popen("snmpwalk -v 2c -c public localhost 1.3.6.1.2.1.2.2.1.16")
a=[]

for j in range(5):
    a.append(p.readline())

bytes_se=' {0:.2f} Mb'.format(float(a[4][36:45])/1024/1024)


x5="网卡接收流量: "+(str)(bytes_re)+'\n'
x6="网卡发送流量: "+(str)(bytes_se)+'\n'


printresult(x3+'\n'+x4+'\n'+x2+x1+x7+x8+'\n'+x5+x6)


def fun_timer():
    t.delete(1.0, tkinter.END)

    cipankongjian()

    cpu_usage,mem_usage=getXandY()
    if (cpu_usage>float(C_yuzhi)):
        tkinter.messagebox.showwarning('警告','CPU 占用率超过阈值')
    if (mem_usage>float(M_yuzhi)):
        tkinter.messagebox.showwarning('警告','内存占用率超过阈值')

    t.insert('1.0', '\n')

global timer

```

[illegible]

```

mem_usage=float(b[7][69:71])

return float(cpu_usage),float(mem_usage)

xC=-1

xM=-1

def liyonglv():

    fig=plt.figure()

    ax_CPU = fig.add_subplot(2,1,1,xlim=(0, 10), ylim=(0, 100))

    ax_MEM = fig.add_subplot(2,1,2,xlim=(0, 10), ylim=(0, 100))

    ax_CPU.set_ylabel(u"CPU 利用率(%)",fontproperties=font)

    ax_MEM.set_ylabel(u"内存利用率(%)",fontproperties=font)

    lineC, = ax_CPU.plot([], [], lw=1)

    lineM, = ax_MEM.plot([], [], lw=1)

def init():

    lineC.set_data([], [])

    lineM.set_data([], [])

    return lineC,lineM

xdataC, ydataC = [], []

xdataM, ydataM = [], []

def run(frame):

    global xC,xM

    xC=xC+1

    xM=xM+1

    yC, yM = getXandY()

    xdataC.append(xC)

```



```

ydataC.append(yC)

xdataM.append(xM)

ydataM.append(yM)

xmin, xmax = ax_CPU.get_xlim()

if xC >= xmax:

    ax_CPU.set_xlim(xmin+1, xmax+1)

    ax_CPU.figure.canvas.draw()

    ax_MEM.set_xlim(xmin+1, xmax+1)

    ax_MEM.figure.canvas.draw()

lineC.set_data(xdataC, ydataC)

lineM.set_data(xdataM, ydataM)

return lineC, lineM

animation.FuncAnimation(fig, run, interval=6000, repeat=False, init_func=init,
blit=True)

plt.show()

tkinter.Button(root, text="利用率曲线", command =
liyonglv).pack(fill=tkinter.X, padx=10)

root.mainloop()

```