#!/usr/bin/env python3

import tkinter

from tkinter import \*

from tkinter import ttk

import os

import string

import time

import heapq

#Declaration of Variables

import

interval = 5

intrCount = 0

intrOld = 0

ctxtCount = 0

ctxtOld = 0

anotherFlag = 0

Cur\_FreeMemory =0

Pre\_FreeMemory =0

CurrentUserModeTime =0

CurrentKernelModeTime=0

#### Disk ####

diskf = open("/proc/diskstats", "r")

diskfileRead = diskf.read()

#print(fileRead)

diskFileList = diskfileRead.split()

##################################Disk###########################

Noofsda =0

for word in diskFileList:

if "sda" in word:

Noofsda +=1

Pre\_diskread=[0 for i in range(0,Noofsda)]

Pre\_diskwrite = [0 for i in range(0,Noofsda)]

Pre\_sectorread = [0 for i in range(0,Noofsda)]

Pre\_sectorwrite = [0 for i in range(0,Noofsda)]

###################CPU UTILIZATION#######################################

f = open("/proc/stat", "r")

fileRead = f.read()

FileList = fileRead.split()

NoofCPU =0

for word in FileList:

if "cpu" in word:

NoofCPU +=1

print ("Number of CPU is {0} \n".format(NoofCPU-2))

PreviousUserModeTime= [0 for i in range(0,NoofCPU-1)]

PreviousKernelModeTime=[0 for i in range(0,NoofCPU-1)]

###################################PROCESS#####################################################################

filenames = os.listdir("/proc") #Find all directories in /proc

count = len(filenames)

print("The number of filename in /proc are {0}".format(count))

#declaration of variable

lastRecord = 0

startTime = time.clock()

endTime = time.clock()

utime =[0 for i in range(0,count)]

pre\_utime=[0 for i in range(0,count)]

cur\_utime=[0 for i in range(0,count)]

stime=[0 for i in range(0,count)]

pre\_stime=[0 for i in range(0,count)]

cur\_stime=[0 for i in range(0,count)]

CPU\_utilization=[0 for i in range(0,count)]

UserName=[0 for i in range(0,count)]

ProgramName=[0 for i in range(0,count)]

UID=[0 for i in range(0,count)]

PID=[0 for i in range(0,count)]

ProcessVirtualMem=[0 for i in range(0,count)]

ProcessPhysicalMem=[0 for i in range(0,count)]

dict\_individualPID\_info ={}

heap =[]

localist =[]

CPU\_utilization\_proc = [0 for i in range(0,count)]

status = [0 for i in range(0,count)]

#Find Total Physical Memory from /proc/meminfo

memfile = open("/proc/meminfo","r")

memfileRead = memfile.read()

memfile.close()

memFileList = memfileRead.split()

TotalMemory = float(memFileList[1])/1024

count\_ProcessNum =0

for fn in filenames:

if(fn.isdigit()):

filestatus = open("/proc/" +fn+"/status")

datastatus = filestatus.read()

filestatus.close()

datalist\_status = datastatus.split()

UIDindex = datalist\_status.index('Uid:')

UID[count\_ProcessNum] = datalist\_status[UIDindex+1]

filehandler = open("/proc/" +fn+"/stat")

data = filehandler.read()

filehandler.close()

datalist = data.split()

file\_ProgramName = open("/etc/passwd")

data\_ProgramName = file\_ProgramName.read()

datalist\_ProgramName = data\_ProgramName.split(":")

ProgramName[count\_ProcessNum]= datalist[1]

PID[count\_ProcessNum] = datalist[0] #PID

UserName[count\_ProcessNum] = datalist[1] #UserName

ProgramNameindex = datalist\_ProgramName.index(UID[count\_ProcessNum])

ProgramName[count\_ProcessNum] =datalist\_ProgramName[ProgramNameindex+2]

count\_ProcessNum +=1

####### Network variables Declaration#####################################################

cur\_Packetsent=0

cur\_Packetrecv =0

cur\_OutRequest=0

cur\_ActiveOpen =0 #No of TCP connections

cur\_CurrentEstablish = 0 # TCP connection establish

cur\_InSeg =0 #TCP segments

cur\_OutSeg =0 # TCP out segments

cur\_InDataGrams=0 ##UDP

cur\_OutDataGrams =0 ## UDP

pre\_Packetsent=0

pre\_Packetrecv =0

pre\_OutRequest=0

pre\_ActiveOpen =0 #No of TCP connections

pre\_CurrentEstablish = 0 # TCP connection establish

pre\_InSeg =0 #TCP segments

pre\_OutSeg =0 # TCP out segments

pre\_InDataGrams=0 ##UDP

pre\_OutDataGrams =0 ## UDP

speed=1000 ####Mb/s

class DataCollection:

def \_\_init\_\_(self):

self.firsttimeflag = 0

self.firsttimeflag\_disk = 0

#self.h\_proc={}

def performance(self):

global intrCount,intrOld, ctxtCount, ctxtOld,anotherFlag, Cur\_FreeMemory, Pre\_FreeMemory, CurrentUserModeTime, CurrentKernelModeTime, PreviousUserModeTime, PreviousKernelModeTime, TotalMemory,FreeMemory,TotalUtilization

self.ctxtValue =0

self.intrValue =0

self.dict\_CPU = {}

list\_CPUdata =[0,0,0,0]

file = open ("/proc/stat","r")

fileRead = file.read()

file.close()

FileList = fileRead.split()

intrOld = intrCount

ctxtOld = ctxtCount

intrindex = FileList.index("intr")

if (self.firsttimeflag ==1):

intrCount =int(FileList[intrindex+1])

self.intrValue =(float(intrCount - intrOld)/interval)

print("intr count is {0}".format(self.intrValue))

elif(self.firsttimeflag ==0):

intrCount =int(FileList[intrindex+1])

self.intrValue = intrCount

print("intrCount is {0}".format(self.intrValue))

ctxtindex = FileList.index("ctxt")

if (self.firsttimeflag ==1):

ctxtCount =int (FileList[ctxtindex + 1])

self.ctxtValue =(float(ctxtCount - ctxtOld)/interval)

print("ctxt count is {0}".format(self.ctxtValue))

elif(self.firsttimeflag ==0):

ctxtCount =int (FileList[ctxtindex + 1])

self.ctxtValue =ctxtCount

print ("ctxtCount {0}".format(self.ctxtValue))

#User Mode and kernel mode CPU calculation

CPUcount = 0

while (CPUcount <= (NoofCPU-2)):

name = "cpu"+ str(CPUcount)

print(name)

print("NoofCPU = {0}".format(NoofCPU-2))

print("PreviousUserModeTime = {0}".format(PreviousUserModeTime[0]))

indexCPU = FileList.index(name)

CurrentUserModeTime =float( FileList[indexCPU+1])/100

CurrentKernelModeTime = float(FileList[indexCPU+3])/100

UserMode = (CurrentUserModeTime -PreviousUserModeTime[CPUcount])/interval

KernelMode =(CurrentKernelModeTime -PreviousKernelModeTime[CPUcount])/interval

Utilization = ((UserMode+KernelMode)/interval)\*100

# if (firsttimeflag == 0):

# firsttimeflag =1

# else:

print("User Mode for {0} is {1}sec ".format(name,UserMode))

print("Kernel Mode for {0} is {1}sec ".format(name,KernelMode))

print("Utilization for{0} is {1}%\n ".format(name,Utilization))

list\_CPUdata = [name,UserMode,KernelMode,Utilization,CurrentUserModeTime,CurrentKernelModeTime]

self.dict\_CPU.update({CPUcount:list\_CPUdata})

print(self.dict\_CPU)

PreviousUserModeTime[CPUcount] = self.dict\_CPU[CPUcount][4]

PreviousKernelModeTime[CPUcount]= self.dict\_CPU[CPUcount][5]

CPUcount += 1

memfile = open("/proc/meminfo","r")

memfileRead = memfile.read()

# print(memfileRead)

memfile.close()

memFileList = memfileRead.split()

Pre\_FreeMemory =Cur\_FreeMemory

Cur\_FreeMemory = float(memFileList[4])/1024

TotalMemory = float(memFileList[1])/1024

FreeMemory = float(Pre\_FreeMemory+Cur\_FreeMemory)/2

TotalUtilization = (float(TotalMemory - FreeMemory)/float(TotalMemory))\*100

if (self.firsttimeflag == 1):

print("Total Memory is {0} MB".format(TotalMemory))

print("Free Memory is {0} MB".format(FreeMemory))

print("Total Utilization is {0} % ".format(TotalUtilization))

def sys\_Disk(self):

file = open ("/proc/diskstats","r")

diskstat\_LineRead = file.readlines()

self.dict\_diskstat={}

count =0

for eachline in diskstat\_LineRead:

if eachline.find('sda')!= -1:

word = eachline.split()

name = word[2]

diskread =int(word[3])

diskwrite = int(word[7])

sectorread = int(word[5])

sectorwrite = int(word[9])

print("self.firsttimeflag\_disk {0}".format(self.firsttimeflag\_disk) )

Value\_diskread =(float(diskread - Pre\_diskread[count])/interval)

Value\_diskwrite =(float(diskwrite - Pre\_diskwrite[count])/interval)

Value\_sectorread =(float(sectorread - Pre\_sectorread[count])/interval)

Value\_sectorwrite =(float(sectorwrite - Pre\_sectorwrite[count])/interval)

list\_disk =[name,Value\_diskread,Value\_diskwrite,Value\_sectorread,Value\_sectorwrite]

self.dict\_diskstat.update({count:list\_disk})

# print("AFTER diskread is {0}".format(Value\_diskread))

# print("diskwrite is {0}".format(Value\_diskwrite))

# print("sectorread is {0}".format(Value\_sectorread))

# print("sectorwrite is {0}\n".format(Value\_sectorwrite))

Pre\_diskread[count] = diskread

Pre\_diskwrite[count] =diskwrite

Pre\_sectorread[count] =sectorread

Pre\_sectorwrite[count] =sectorwrite

count +=1

###############################PROCESS#######################################################################

def sys\_process(self):

global locallist,heap

print("Inside while loop")

#if (endTime - startTime - lastRecord > 5):

# interval =endTime - startTime - lastRecord

# lastRecord = endTime - startTime

count\_ProcessNum =0

for fn in filenames:

if(fn.isdigit()):

filehandler = open("/proc/"+fn+"/stat")

data = filehandler.read()

#print(data)

filehandler.close()

datalist = data.split()

status[count\_ProcessNum] = datalist[2]

pre\_utime[count\_ProcessNum-1] = cur\_utime[count\_ProcessNum-1]

cur\_utime[count\_ProcessNum] = (float(datalist[13]))/100 #user mode time

# print("cur {0}".format(cur\_utime[count\_ProcessNum]))

utime[count\_ProcessNum] = cur\_utime[count\_ProcessNum]- pre\_utime[count\_ProcessNum]

#print("Utime is {0}".format(utime[ProcessNum]))

pre\_stime[count\_ProcessNum-1] = cur\_stime[count\_ProcessNum-1]

cur\_stime[count\_ProcessNum] = (float(datalist[14]))/100 # kernel mode time

stime[count\_ProcessNum] = cur\_stime[count\_ProcessNum]- pre\_stime[count\_ProcessNum]

print("Addition {0}".format(stime[count\_ProcessNum] + utime[count\_ProcessNum]))

CPU\_utilization\_proc[count\_ProcessNum] = ((stime[count\_ProcessNum] + utime[count\_ProcessNum])/5)

#cur\_stime[count\_ProcessNum]+cur\_utime[count\_ProcessNum])

rss = datalist[23]#rss

#print("RSS is {0}".format(rss))

VirtualMem = datalist[22] #Virtual memory Size

#print("Virtual Mem Size is {0}".format(VirtualMem))

ProcessVirtualMem[count\_ProcessNum] = (int(VirtualMem)\*1024 / (2\*\*64))\*100

#print("Process Virtual Mem Size is {0}".format(VirtualMem))

ProcessPhysicalMem[count\_ProcessNum] = (int(rss) /int(TotalMemory) )\*100

#print("Process Physical Mem Size is {0}".format(ProcessPhysicalMem))

count\_ProcessNum +=1

for count in range(1,count\_ProcessNum):

dict\_individualPID\_info.update({PID[count]:{'UID':UID[count],'ProgramName':ProgramName,'UserName':UserName[count],'utime':utime[count],'stime':stime[count],'ProcessVirtualMem':ProcessVirtualMem[count],'ProcessPhysicalMem':ProcessPhysicalMem[count]}})

###########heap#############

listheap = [CPU\_utilization\_proc[count],PID[count],ProgramName[count],status[count],UserName[count],ProcessVirtualMem[count],ProcessPhysicalMem[count]]

heapq.heappush(heap,listheap)

# print(dict\_individualPID\_info[PID[count]]['UserName'])

locallist=heapq.nlargest (50,heap)

##################################Networking################################################

def sys\_networking(self):

####First part########

#snmpFileListj

global cur\_Packetsent,cur\_Packetrecv ,cur\_OutRequest,cur\_ActiveOpen ,cur\_CurrentEstablish ,cur\_InSeg ,cur\_OutSeg ,cur\_InDataGrams,cur\_OutDataGrams,pre\_Packetsent,pre\_Packetrecv,pre\_OutRequest,pre\_ActiveOpen,pre\_CurrentEstablish,pre\_InSeg,pre\_OutSeg ,pre\_InDataGrams ,pre\_OutDataGrams

self.Packetsent=0

self.Packetrecv =0

self.OutRequest=0

self.ActiveOpen =0 #No of TCP connections

self.CurrentEstablish = 0 # TCP connection establish

self.InSeg =0 #TCP segments

self.OutSeg =0 # TCP out segments

self.InDataGrams=0 ##UDP

self.OutDataGrams =0 ## UDP

filesnmp = open("/proc/net/snmp","r")

snmpfileRead = filesnmp.read()

snmpFileList = snmpfileRead.split()

pre\_Packetsent= cur\_Packetsent

pre\_Packetrecv =cur\_Packetrecv

pre\_OutRequest= cur\_OutRequest

pre\_ActiveOpen = cur\_ActiveOpen #No of TCP connections

pre\_CurrentEstablish = cur\_CurrentEstablish # TCP connection establish

pre\_InSeg = cur\_InSeg #TCP segments

pre\_OutSeg = cur\_OutSeg # TCP out segments

pre\_InDataGrams= cur\_InDataGrams ##UDP

pre\_OutDataGrams = cur\_OutDataGrams ## UDP

cur\_Packetsent = float(snmpFileList[21])

cur\_Packetrecv = float(snmpFileList[22])

cur\_OutRequest = float(snmpFileList[30])

indexTcp = snmpFileList.index("Tcp:")

cur\_ActiveOpen =float(snmpFileList[indexTcp+21]) #No of TCP connections

cur\_CurrentEstablish = float(snmpFileList[indexTcp+22]) # TCP connection establish

cur\_InSeg = float(snmpFileList[indexTcp+26]) #TCP segments

cur\_OutSeg =float(snmpFileList[indexTcp+27]) # TCP out segments

indexUDP = snmpFileList.index("Udp:")

cur\_InDataGrams =float(snmpFileList[indexUDP+9]) #No of TCP connections

cur\_OutDataGrams = float(snmpFileList[indexUDP+12]) # TCP connection establish

self.Packetsent= (cur\_Packetsent - pre\_Packetsent)/interval

self.Packetrecv =(cur\_Packetrecv- pre\_Packetrecv)/interval

self.OutRequest=(cur\_OutRequest -pre\_OutRequest)/interval

self.ActiveOpen = (cur\_ActiveOpen + pre\_ActiveOpen)/2 #No of TCP connections

self.CurrentEstablish = (cur\_CurrentEstablish + pre\_CurrentEstablish)/2 # TCP connection establish

self.InSeg = (cur\_InSeg - pre\_InSeg)/interval #TCP segments

self.OutSeg = (cur\_OutSeg-pre\_OutSeg)/interval # TCP out segments

self.InDataGrams= (cur\_InDataGrams -pre\_InDataGrams)/interval ##UDP

self.OutDataGrams = (cur\_OutDataGrams -pre\_OutDataGrams)/interval ## UDP

print( "No of Ip packets sent = {0} ,No of packets received {1} and OutRequest = {2} ".format(self.Packetsent,self.Packetrecv,self.OutRequest))

print( "ActiveOpen = {0} ,CurrentEstablish ={1}, InSeg = {2} and OutSeg = {3} ".format(self.ActiveOpen, self.CurrentEstablish, self.InSeg,self.OutSeg))

print ("InDataGrams = {0} and OutDataGrams ={1}".format(self.InDataGrams, self.OutDataGrams))

##############table################################################

self.dict\_tcp\_upd={}

self.connectioncount =0

list\_connection=[]

filenames = os.listdir("/proc")

fileTCPopen = open("/proc/net/tcp")

fileTCPread = fileTCPopen.readlines()

fileTCPopen.close()

for TCPline in fileTCPread:

datalist\_TCP = TCPline.split()

protocol = "TCP"

list\_connection=[datalist\_TCP[11],datalist\_TCP[9],datalist\_TCP[2],datalist\_TCP[1],protocol]

self.dict\_tcp\_upd.update({self.connectioncount:list\_connection})

self.connectioncount += 1

"""print("inode {}".format(datalist\_TCP[11]))

print("uid {}".format(datalist\_TCP[9]))

print("rem\_address {}".format(datalist\_TCP[2]))

print("local\_address {}".format(datalist\_TCP[1]))

print("Protocol TCP")"""

fileUDPopen = open("/proc/net/udp")

fileUDPread = fileUDPopen.readlines()

fileUDPopen.close()

for UDPline in fileUDPread:

datalist\_UDP = UDPline.split()

protocol = "UDP"

list\_connection=[datalist\_UDP[11],datalist\_UDP[9],datalist\_UDP[2],datalist\_UDP[1],protocol]

self.dict\_tcp\_upd.update({self.connectioncount:list\_connection})

self.connectioncount += 1

"""print("inode {}".format(datalist\_UDP[11]))

print("uid {}".format(datalist\_UDP[9]))

print("rem\_address {}".format(datalist\_UDP[2]))

print("local\_address {}".format(datalist\_UDP[1]))

print("Protocol UDP") """

print(self.dict\_tcp\_upd)

fileDEVopen = open("/proc/net/dev")

fileDEVread = fileDEVopen.readlines()

fileDEVopen.close()

for fileDEVline in fileDEVread:

word= fileDEVline.split()

print(word)

if (len(word)==17):

byte = float(word[9])

self.networkUtilization = byte/speed

print(self.networkUtilization)

#############################################################################################################

root = Tk()

root.title("Task Manager")

#root.geometry('%dx%d+%d+%d' % (w, h, x, y))

root.geometry('%dx%d+%d+%d' % (750, 600, 0, 0))

#Tab for root and individual tab click

note = ttk.Notebook(root)

tab\_CPU = Frame(note)

tab\_Disk = Frame(note)

tab\_Network = Frame(note)

tab\_Process = Frame(note)

note.add(tab\_CPU, text = "System Performance" )

note.add(tab\_Disk, text = "Disk I/O")

note.add(tab\_Network, text = "Network I/O")

note.add(tab\_Process, text = "Process")

Network\_textBox=Text(tab\_Network,height=400,width=200)

Network\_textBox.grid(row=0,column=0,columnspan=3)

CPU\_textBox=Text(tab\_CPU,height=400,width=200)

CPU\_textBox.grid(row=0,column=0,columnspan=3)

Label(tab\_Process, text = "Filter :", borderwidth = 0).grid(row = 0, column = 0)

t =Text(tab\_Process, borderwidth = 0,width =10,height =1).grid(row = 0, column = 1)

Key\_Button=Button(tab\_Process,text ="Search", borderwidth = 1,command = "SearchButtonclick")

Key\_Button.grid(row=0,column=2)

def SearchButtonclick (self):

s = self.t.get("1.0",END)

print(s)

def sys\_performance():

data.performance()

CPU\_textBox.delete('1.0',END)

CPU\_textBox.insert(END,"CPU Utilization \n\n")

CPU\_textBox.insert(END,"Name\t"+"User Mode Time\t\t\t"+"Kernel Mode Time"+"\t\t\t"+"CPU\_Utilization\t\t"+"\n")

for i in range(0,NoofCPU-1):

print("NoofCPU is {0}".format(i))

CPU\_textBox.insert(END,str(data.dict\_CPU[i][0])+'\t'+str(round(data.dict\_CPU[i][1],2))+'\t\t'+str(round(data.dict\_CPU[i][2],2))+'\t\t'+str(round(data.dict\_CPU[i][3],2))+'\n')

CPU\_textBox.insert(END,"Number of Interrupts Serviced\t\t")

CPU\_textBox.insert(END,str(data.intrValue)+'\t'+'\n')

CPU\_textBox.insert(END,"Number of Context switches\t\t")

CPU\_textBox.insert(END,str(data.ctxtValue)+'\t'+'\n')

CPU\_textBox.insert(END,"----------------------------------------------------------------------------------------------------------\n\n")

CPU\_textBox.insert(END,"Memory Info \n\n")

CPU\_textBox.insert(END,"TotalMemory\t\t\t"+"FreeMemory\t\t"+"Mem\_Util\t\t"+"\n")

CPU\_textBox.insert(END,str(TotalMemory)+'\t\t\t'+str(FreeMemory)+'\t\t'+str(TotalUtilization)+'\n')

CPU\_textBox.insert(END,"----------------------------------------------------------------------------------------------------------\n\n")

root.after(5000,sys\_performance)

def Tab\_disk():

if (data.firsttimeflag\_disk ==0):

data.sys\_Disk()

data.firsttimeflag\_disk =1

else:

data.sys\_Disk()

Label(tab\_Disk,text ="Name",borderwidth =0,width=10).grid(row=4,column=0,padx =1,pady =1)

Label(tab\_Disk,text ="Disk Reads",borderwidth =0,width=10).grid(row=4,column=3,padx =1,pady =1)

Label(tab\_Disk,text ="Disk Writes",borderwidth =0,width=10).grid(row=4,column=6,padx =1,pady =1)

Label(tab\_Disk,text ="Sector Reads",borderwidth =0,width=13).grid(row=4,column=13,padx =1,pady =1)

Label(tab\_Disk,text ="Sector Writes",borderwidth =0,width=13).grid(row=4,column=25,padx =1,pady =1)

b =5

for i in range(0,Noofsda):

b =b+1

Label(tab\_Disk,text =data.dict\_diskstat[i][0],borderwidth =0,width=10).grid(row=b,column=0,padx =1,pady =1)

Label(tab\_Disk,text =str(data.dict\_diskstat[i][1]),borderwidth =0,width=10).grid(row=b,column=3,padx =1,pady =1)

Label(tab\_Disk,text =str(data.dict\_diskstat[i][2]),borderwidth =0,width=10).grid(row=b,column=6,padx =1,pady =1)

Label(tab\_Disk,text =str(data.dict\_diskstat[i][3]),borderwidth =0,width=13).grid(row=b,column=13,padx =1,pady =1)

Label(tab\_Disk,text =str(data.dict\_diskstat[i][4]),borderwidth =0,width=13).grid(row=b,column=25,padx =1,pady =1)

root.after(5000,Tab\_disk)

def Tab\_process():

global locallist

print(time.ctime())

data.sys\_process()

Label(tab\_Process,text ="PID",borderwidth =0,width=10).grid(row=1,column=0,padx =1,pady =1)

Label(tab\_Process,text ="Username",borderwidth =0,width=10).grid(row=1,column=3,padx =1,pady =1)

Label(tab\_Process,text ="Status",borderwidth =0,width=10).grid(row=1,column=6,padx =1,pady =1)

Label(tab\_Process,text ="Vitual Memory",borderwidth =0,width=13).grid(row=1,column=13,padx =1,pady =1)

Label(tab\_Process,text ="Physical Memory",borderwidth =0,width=13).grid(row=1,column=25,padx =1,pady =1)

Label(tab\_Process,text ="%CPU",borderwidth =0,width=13).grid(row=1,column=30,padx =1,pady =1)

Label(tab\_Process,text ="Command",borderwidth =0,width=15).grid(row=1,column=35,padx =1,pady =1)

b=1

for f in locallist:

b =b +1

Label(tab\_Process,text = str(f[1]),borderwidth =0,width=10).grid(row=b,column=0,padx =1,pady =1)

Label(tab\_Process,text = str(f[2]),borderwidth =0,width=10).grid(row=b,column=3,padx =1,pady =1)

Label(tab\_Process,text = str(f[3]),borderwidth =0,width=10).grid(row=b,column=6,padx =1,pady =1)

Label(tab\_Process,text = str(round(f[5],2)),borderwidth =0,width=13).grid(row=b,column=13,padx =1,pady =1)

Label(tab\_Process,text = str(round(f[6],2)),borderwidth =0,width=13).grid(row=b,column=25,padx =1,pady =1)

Label(tab\_Process,text = str(round(f[0],2)),borderwidth =0,width=13).grid(row=b,column=30,padx =1,pady =1)

Label(tab\_Process,text = str(f[4]),borderwidth =0,width=15).grid(row=b,column=35,padx =1,pady =1)

root.after(5000,Tab\_process)

def Tab\_network():

data.sys\_networking()

Network\_textBox.delete('1.0',END)

Network\_textBox.insert(END,"\n")

Network\_textBox.insert(END, "Packets sent \t\t Packets received \t\tOutRequest\n ")

Network\_textBox.insert(END,str(data.Packetsent)+" \t\t\t"+str(data.Packetrecv)+"\t\t"+str(data.OutRequest)+"\n")

Network\_textBox.insert(END, "ActiveOpen \t CurrentEstablish \t\tInSeg \t\tOutSeg\n ")

Network\_textBox.insert(END,str(data.ActiveOpen)+"\t\t"+str( data.CurrentEstablish)+"\t\t"+ str(data.InSeg)+"\t"+str(data.OutSeg)+'\n')

Network\_textBox.insert(END,"InDataGrams \t\t OutDataGrams\n")

Network\_textBox.insert(END,str(data.InDataGrams)+"\t\t"+ str(data.OutDataGrams)+'\n')

Network\_textBox.insert(END, "UserName \t\t Program Name \t\tRemote address \t\tLocal address \t\t Protocol\n")

for i in range(1,data.connectioncount):

Network\_textBox.insert(END,data.dict\_tcp\_upd[i][0]+'\t'+data.dict\_tcp\_upd[i][1]+'\t\t'+data.dict\_tcp\_upd[i][2]+'\t\t'+data.dict\_tcp\_upd[i][3]+'\t'+data.dict\_tcp\_upd[i][4]+'\n')

Network\_textBox.insert(END,"----------------------------------------------------------------------------------------------------------\n\n")

Network\_textBox.insert(END,"Network Utilization" +'\n')

Network\_textBox.insert(END,str(data.networkUtilization) +'\n')

Network\_textBox.insert(END,"----------------------------------------------------------------------------------------------------------\n\n")

root.after(5000,Tab\_network)

data = DataCollection()

sys\_performance()

Tab\_disk()

Tab\_process()

Tab\_network()

note.pack()

root.mainloop()