

操作系统原理

课程设计

**课设名称： 段页式存储管理­­­**

**专业班级： 2016级物联网工程**

**组 长： 章瀚中**

**成 员： 么雨**

**指导教师： 付纬娜**

**2018 - 2019 学年 第 一 学期**

目录

[1.设计题目 4](#_Toc531623223)

[2.设计目标 4](#_Toc531623224)

[3.设计内容与步骤 4](#_Toc531623225)

[3.1段页式存储管理的基本思想 4](#_Toc531623226)

[3.2概要设计 5](#_Toc531623227)

[4.设计总结 11](#_Toc531623228)

[5.参考文献 12](#_Toc531623229)

[6.程序代码： 12](#_Toc531623230)

# 1.设计题目

模拟段页式存储管理方式中的地址转换和缺页中断

# 2.设计目标

利用Python实现段页式存储管理方式中的地址转换和缺页中断

# 3.设计内容与步骤

### 3.1段页式存储管理的基本思想

段页式存储组织是分段式和分页式结合的存储组织方法，这样可充分利用分段管理和分页管理的优点。

(1) 用分段方法来分配和管理虚拟存储器。程序的地址空间按逻辑单位分成基本独立的段，而每一段有自己的段名，再把每段分成固定大小的若干页。

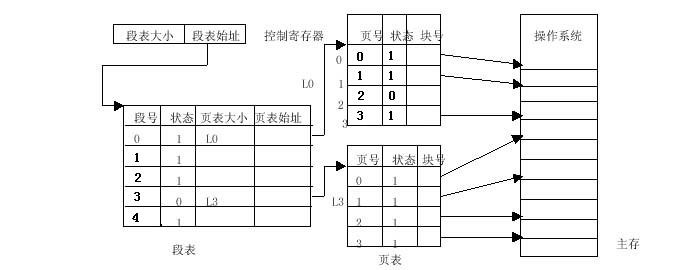
(2) 用分页方法来分配和管理实存。即把整个主存分成与上述页大小相等的存储块，可装入作业的任何一页。程序对内存的调入或调出是按页进行的。但它又可按段实现共享和保护。

(3)逻辑地址结构。一个逻辑地址用三个参数表示：段号S；页号P；页内地址d。



逻辑地址结构

（4）段表、页表、段表地址寄存器。为了进行地址转换，系统为每个作业建立一个段表，并且要为该作业段表中的每一个段建立一个页表。系统中有一个段表地址寄存器来指出作业的段表起始地址和段表长度。



2.地址变换过程（慢速地址转换过程）

一个逻辑地址为：基地址x、段号s、页号p和页内地址d。寻找段号，判断是否越界，否，即可匹配页号找到块号，页内地址迁移到块内地址。

在段页式系统中，为了便于实现地址变换，须配置一个段表寄存器，其中存放段表始址和段表长TL。

### 3.2概要设计

#### 3.2.1数据结构：

段表，页表，逻辑地址，内存中的块，段表寄存器采用列表实现

段表：[【段号，状态，页表大小（页的数量），页表始址】,【段号，状态，页表大小（页的数量），页表始址(对应着页表的页号)】...]

页表：[【【页号，状态，块号】,【页号，状态，块号】...】,【【页号，状态，块号】,【页号，状态，块号】...】...]

逻辑地址：[段号，页号，页内地址]

内存中的块：[【块号，块内地址】,【块号，块内地址】...]

段表寄存器：[段表大小（分段的数量），段表始址]

1) 进行地址变换时，首先利用段号，将它与段表长进行比较。若段号<段表长，表示未越界

2) 于是利用段表始址和段号来求出该段所对应的段表项在段表中的位置，从中得到该段的页表始址

3) 利用[逻辑地址](http://baike.baidu.com/view/893778.htm" \t "_blank)中的段内页号P来获得对应页的页表项位置，从中读出该页所在的物理块号b

4) 再利用块号b和页内地址来构成[物理地址](http://baike.baidu.com/view/883168.htm" \t "_blank)。

#### 3.2.2详细设计

段表，页表，块表，段表寄存器，逻辑地址均使用Treeview显示。

Segment 段表

Page 页表

LogicAddress 块表

Memory 段表寄存器

SegmentR 逻辑地址

SegmentNum1,PageNum1, MemoryNum1 用户输入的段表，页表，块表号

def treeSegmentClick(none):#负责显示鼠标点击段会显示段内页表内容(定义到前面,是因为有这个函数，页表才可以建立)

treeSegment 段表用Treeview控件显示时的名称

treePage 页表用Treeview控件显示时的名称

treeMemory 块号用Treeview控件显示时的名称

treeSegmentR 段表寄存器用Treeview控件显示时的名称

treeLogicAdd 逻辑地址用Treeview控件显示时的名称

labelFinally1 物理地址最后显示的控件

def BuildSegmentTable(SegmentNum1,PageNum1): #建立段表并添入Treeview控件

delSegmentTable():#删除段Treeview里的所有信息

BuildPageTable(SegmentNum1,PageNum1) #建立段内页表并添入Treeview控件

delPageTable():#删除页表控件里的所有信息

BuildMemoryTable(MemoryNum1) #创建块表并显示在控件里\*\*\*\*\*\*\*\*\*

delMemoryTable():#删除块表Treeview里的所有信息

buildSegmentR():#填入段表寄存器数据,并显示在Treeview控件里

delSegmentR():#清空段表寄存器Treeview

LogicAdd():#逻辑地址填入数据,并显示在treeview里

delLogicAdd():#清空逻辑地址treeview里的信息

BuildTable():#将输入数值录入并实现可视化及放在treeview中

InputMemory()将页调入块表中

FindMemory()找物理地址

DelAll()：掉用所有删除函数，清空所有信息

#### 3.2.3流程图

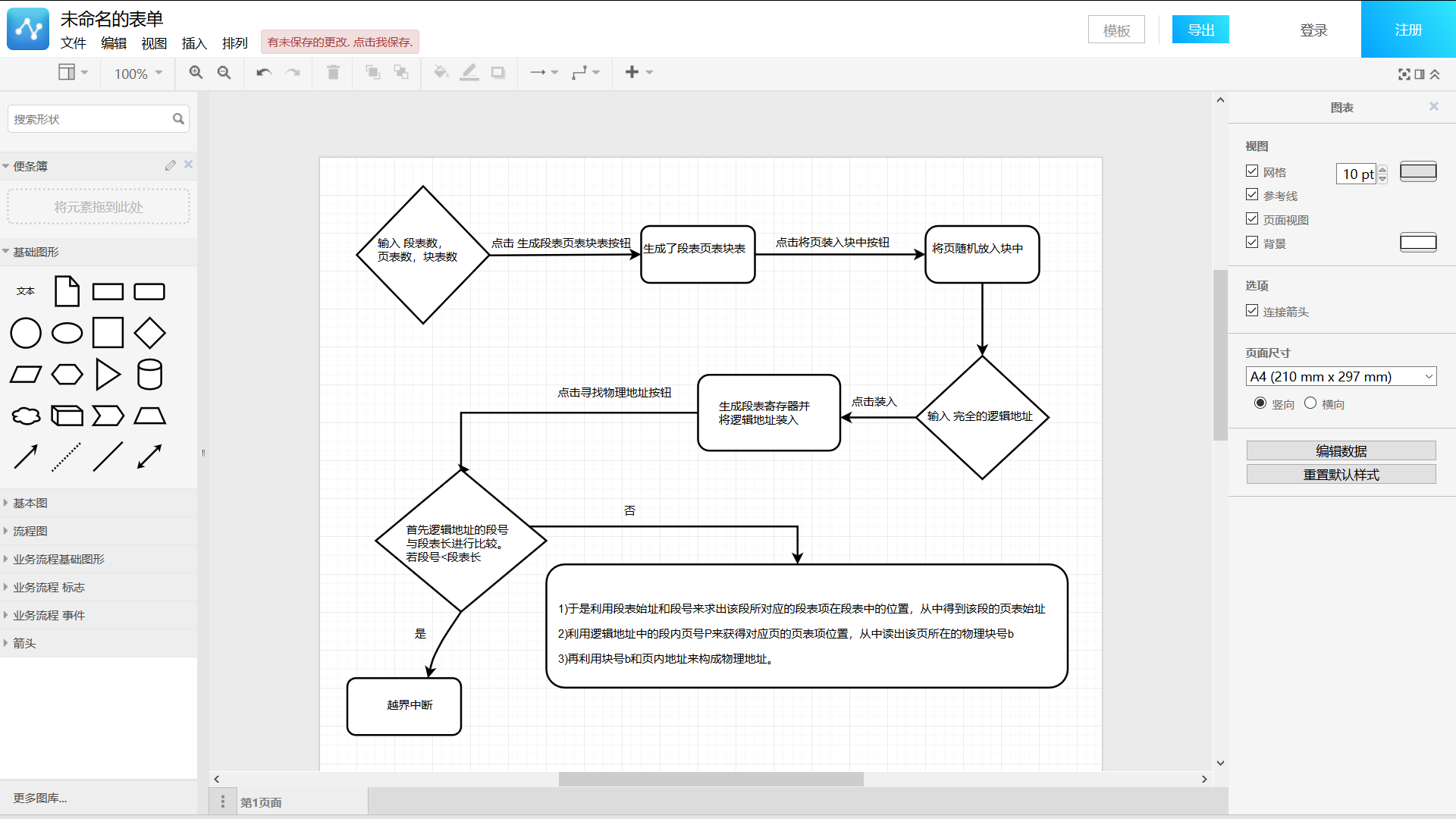
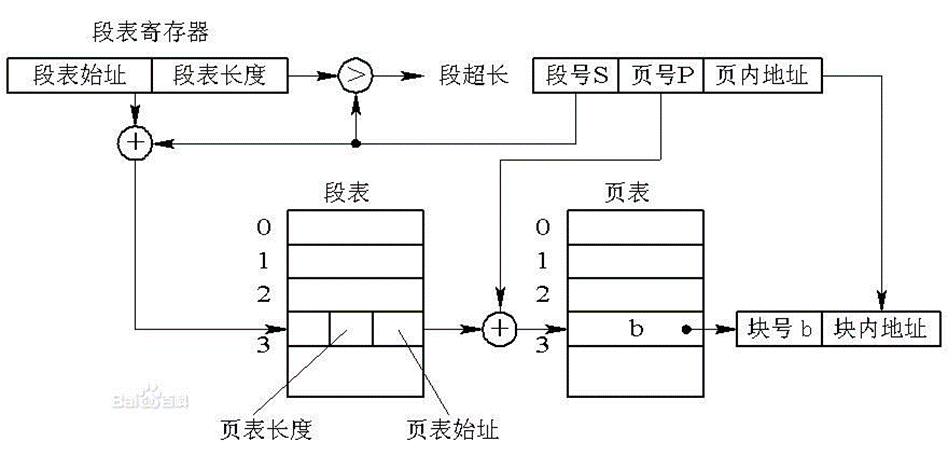
图 一 流程图

图1.程序运行时逻辑图：



2.程序运行时操作流程图：

图（1）填写数据生成段页表，块表：



图（2）将块表放入页表：



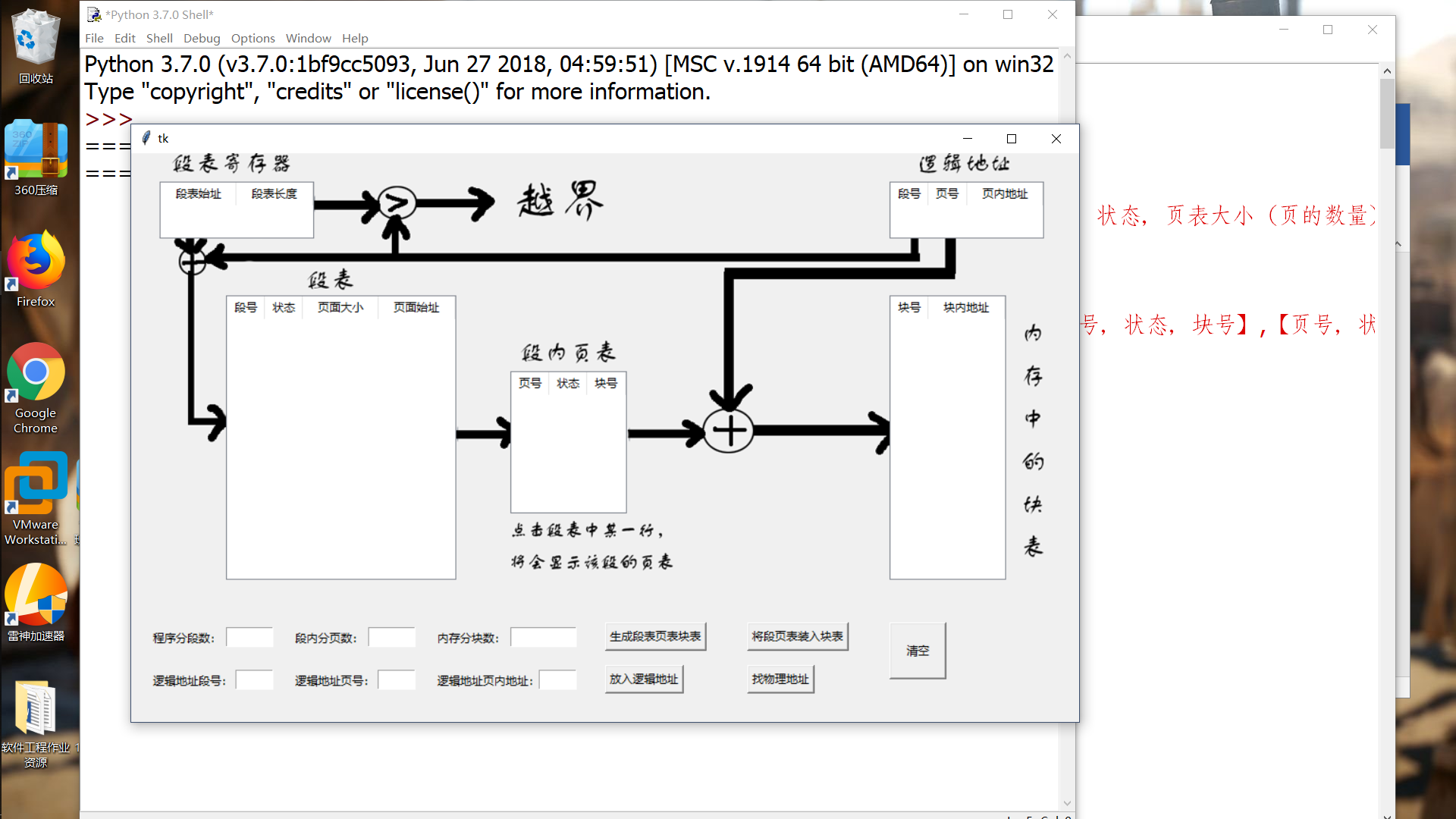
图（3）填写逻辑地址并生成段表寄存器

图（4）完成逻辑地址转换为物理地址

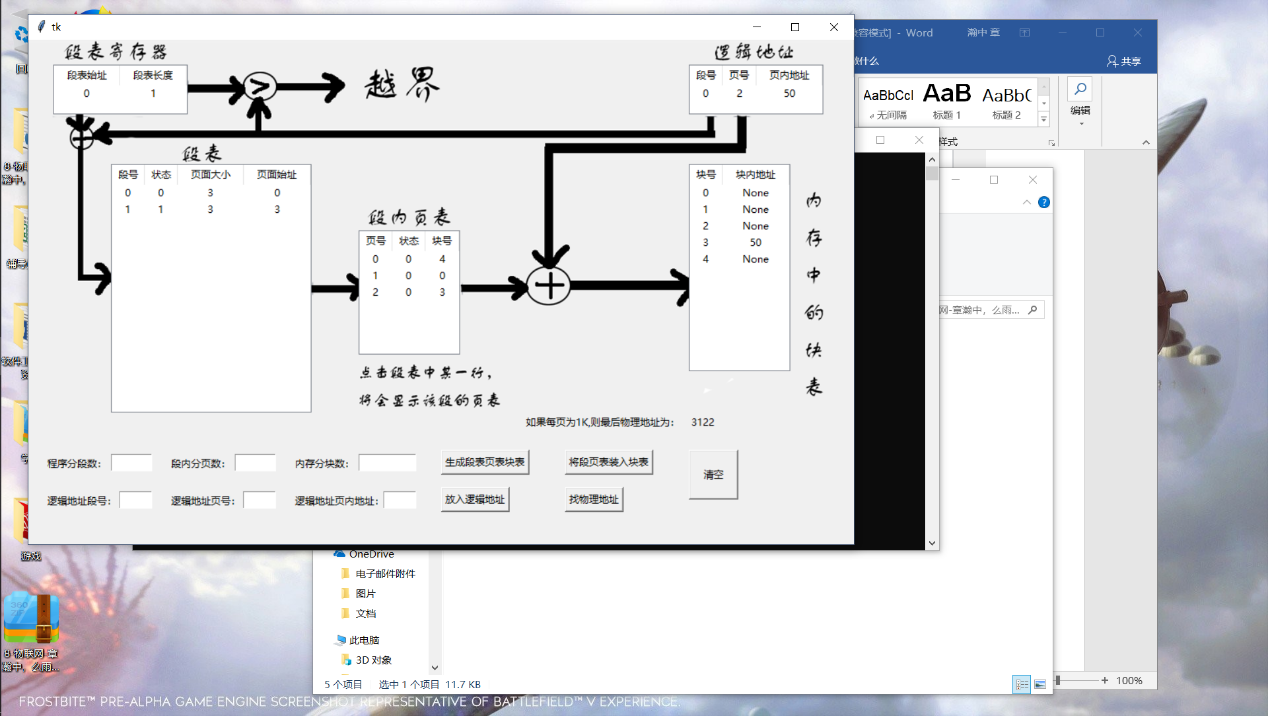
图（5）清空所有消息：  


#### 运行效果截图

图一 程序刚打开的图



图二 程序执行完找到物理地址的图



# 4.设计总结

当块表数量小于（段数\*页数）时，如果某段表未完全放入块表，此段表的状态时未放入。

# 5.参考文献

【1】ISBN 978-7-5606-3350-3 计算机操作系统（第四版）汤小丹 西安电子科技大学出版社 2014年5月第4版

【2】图片来源 <https://www.cnblogs.com/luffya/p/3470563.html>

【3】CSDN <https://blog.csdn.net/sinat_27382047/article/details/80161637>

【4】Python3 说明文档 在编译环境下按F1出现

【5】Google翻译 <https://translate.google.cn/>

# 6.程序代码：

from tkinter import \*

from tkinter import ttk

import random

from tkinter import messagebox

#段表

Segment=[]

#页表

Page=[]

#逻辑地址

LogicAddress=[]

#内存中的块

Memory=[]

#段表寄存器

SegmentR=[]

window=Tk()

window.geometry('1000x600')

img=PhotoImage(file=r"背景图.png")

labelBg=Label(window,image=img)

labelBg.pack()

#输入框区@@@@@@@@@@@@@

#程序分段数量的输入

labelSegmentNum=Label(window,text="程序分段数：")

labelSegmentNum.place(x=20,y=500)

entrySegmentNum=Entry(window)

entrySegmentNum.place(x=100,y=500,width=50)

#段内分页数的输入

labelPageNum=Label(window,text="段内分页数：")

labelPageNum.place(x=170,y=500)

entryPageNum=Entry(window)

entryPageNum.place(x=250,y=500,width=50)

#内存分块数的输入

MemoryNum=StringVar()

labelMemoryNum=Label(window,text="内存分块数：")

labelMemoryNum.place(x=320,y=500)

entryMemoryNum=Entry(window)

entryMemoryNum.place(x=400,y=500,width=70)

#逻辑地址段号的输入

labelLogicSegmentNum=Label(window,text="逻辑地址段号：")

labelLogicSegmentNum.place(x=20,y=545)

entryLogicSegmentNum=Entry(window)

entryLogicSegmentNum.place(x=110,y=545,width=40)

#逻辑地址页号的输入

labelLogicPageNum=Label(window,text="逻辑地址页号：")

labelLogicPageNum.place(x=170,y=545)

entryLogicPageNum=Entry(window)

entryLogicPageNum.place(x=260,y=545,width=40)

#逻辑地址页内地址的输入

labelLogicPageinadd=Label(window,text="逻辑地址页内地址：")

labelLogicPageinadd.place(x=320,y=545)

entryLogicPageinadd=Entry(window)

entryLogicPageinadd.place(x=430,y=545,width=40)

#@@@@@@@@@@@@@@@@@@

#显示区&&&&&&&&&&&&&&&&&&&&

def treeSegmentClick(none):

#负责显示鼠标点击段的页表内容(定义到前面,是因为有这个函数，页表才可以建立)

delPageTable()

a=treeSegment.selection()#得到一个元组

b=treeSegment.item(a,"values")[0]

b=int(b)

## print(b)

global Page

listPage=Page[b]

for i in listPage:

treePage.insert('','end',values=i)

#建立段表

treeSegment=ttk.Treeview(window,columns=['0','1','2','3'],show='headings')

treeSegment.place(x=100,y=150,height=300)

treeSegment.column('0',width=40,anchor='center')

treeSegment.heading('0',text='段号')

treeSegment.column('1',width=40,anchor='center')

treeSegment.heading('1',text='状态')

treeSegment.column('2',width=80,anchor='center')

treeSegment.heading('2',text='页面大小')

treeSegment.column('3',width=80,anchor='center')

treeSegment.heading('3',text='页面始址')

treeSegment.bind('<ButtonRelease-1>',treeSegmentClick)

#建立段内页表

treePage=ttk.Treeview(window,columns=['0','1','2'],show='headings')

treePage.place(x=400,y=230,height=150)

treePage.column('0',width=40,anchor='center')

treePage.heading('0',text='页号')

treePage.column('1',width=40,anchor='center')

treePage.heading('1',text='状态')

treePage.column('2',width=40,anchor='center')

treePage.heading('2',text='块号')

#显示块号

treeMemory=ttk.Treeview(window,columns=['0','1'],show='headings')

treeMemory.place(x=800,y=150,height=300)

treeMemory.column('0',width=40,anchor='center')

treeMemory.heading('0',text='块号')

treeMemory.column('1',width=80,anchor='center')

treeMemory.heading('1',text='块内地址')

#段表寄存器

treeSegmentR=ttk.Treeview(window,columns=['0','1'],show='headings')

treeSegmentR.place(x=30,y=30,height=60)

treeSegmentR.column('0',width=80,anchor='center')

treeSegmentR.heading('0',text='段表始址')

treeSegmentR.column('1',width=80,anchor='center')

treeSegmentR.heading('1',text='段表长度')

#逻辑地址

treeLogicAdd=ttk.Treeview(window,columns=['0','1','2'],show='headings')

treeLogicAdd.place(x=800,y=30,height=60)

treeLogicAdd.column('0',width=40,anchor='center')

treeLogicAdd.heading('0',text='段号')

treeLogicAdd.column('1',width=40,anchor='center')

treeLogicAdd.heading('1',text='页号')

treeLogicAdd.column('2',width=80,anchor='center')

treeLogicAdd.heading('2',text='页内地址')

#&&&&&&&&&&&&&&&&&&&&&&&&

#创建段表和页表的过程￥￥￥￥￥￥￥￥￥￥

#建立段表

def BuildSegmentTable(SegmentNum1,PageNum1):

global Segment

for Segmenti in range(SegmentNum1):

Segment.append([])

Segment[Segmenti].append(Segmenti)

Segment[Segmenti].append(1)

Segment[Segmenti].append(PageNum1)

Segment[Segmenti].append(Segmenti)

for Segmentj in Segment:

treeSegment.insert('','end',values=Segmentj)

## print('S:',Segment)

def delSegmentTable():

#删除段表里的所有信息

num=treeSegment.get\_children()

for i in num:

treeSegment.delete(i)

#建立段内页表

def BuildPageTable(SegmentNum1,PageNum1):

global Page

for Segi in range(SegmentNum1):

Page.append([])

for Pagei in range(PageNum1):

Page[Segi].append([])

Page[Segi][Pagei].append(Pagei)

Page[Segi][Pagei].append(1)

Page[Segi][Pagei].append('None')

## print('P:',Page)

for i in Page[0]:

treePage.insert('','end',values=i)

def delPageTable():

#删除页内表里的所有信息

num=treePage.get\_children()

for i in num:

treePage.delete(i)

#￥￥￥￥￥￥￥￥￥￥￥￥￥￥￥￥￥

#创建块表\*\*\*\*\*\*\*\*\*

def BuildMemoryTable(MemoryNum1):

global Memory

for Memoryi in range (MemoryNum1):

Memory.append([])

Memory[Memoryi].append(Memoryi)

Memory[Memoryi].append('None')

for Memoryk in Memory:

treeMemory.insert('','end',values=Memoryk)

## print('M:',Memory)

def delMemoryTable():

#删除块内表里的所有信息

num=treeMemory.get\_children()

for i in num:

treeMemory.delete(i)

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

def buildSegmentR():

#把段表寄存器填入数据

global SegmentR

Segnum=0

for i in range(len(Segment)):

if Segment[i][1]==0:

Segnum+=1

SegmentR=[0,Segnum]

treeSegmentR.insert('','end',values=SegmentR)

def delSegmentR():

#清空段表寄存器

num=treeSegmentR.get\_children()

for i in num:

treeSegmentR.delete(i)

def LogicAdd():

#把逻辑地址填入数据

global LogicAddress

SegmentNum=int(entryLogicSegmentNum.get())

PageNum=int(entryLogicPageNum.get())

Pageinadd=int(entryLogicPageinadd.get())

LogicAddress=[SegmentNum,PageNum,Pageinadd]

treeLogicAdd.insert('','end',values= LogicAddress)

entryLogicSegmentNum.delete(0,END)#清空输入框内容

entryLogicPageNum.delete(0,END)#清空输入框内容

entryLogicPageinadd.delete(0,END)#清空输入框内容

def delLogicAdd():

#清空逻辑地址

num=treeLogicAdd.get\_children()

for i in num:

treeLogicAdd.delete(i)

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

def BuildTable():

#将输入数值录入并实现可视化

SegmentNum1=int(entrySegmentNum.get())

PageNum1=int(entryPageNum.get())

MemoryNum1=int(entryMemoryNum.get())

BuildSegmentTable(SegmentNum1,PageNum1)

BuildPageTable(SegmentNum1,PageNum1)

BuildMemoryTable(MemoryNum1)

entrySegmentNum.delete(0,END)#清空输入框内容

entryPageNum.delete(0,END)#清空输入框内容

entryMemoryNum.delete(0,END)#清空输入框内容

##++++++将页调入块表中+++++++++++

def InputMemory():

global Segment

global Page

global Memory

Flag=True

MemoryIsUsed=[]

MemoryIsUsed=Memory.copy()#将块表的划分传给一个使用块表

random.shuffle(MemoryIsUsed)#打乱块号的顺序

for S in Segment:

js=0

for P in Page:

for i in range(len(Page[0])):

if len(MemoryIsUsed):

P[i][1]=0

P[i][2]=MemoryIsUsed[0][0]#页得到打乱后的块表的第一个

MemoryIsUsed.pop(0)#删除块表第一个

else:

Flag=False

break

if Flag:

Segment[js][1]=0

js=js+1

else:

break

## print(Segment)

delPageTable()

for i in Page[0]:

treePage.insert('','end',values=i)

delSegmentTable()

for j in Segment:

treeSegment.insert('','end',values=j)

def InputLogic():

buildSegmentR()

LogicAdd()

def FindMemory():

global LogicAddress

global SegmentR

global Page

global Memory

if LogicAddress[0]<=SegmentR[1]:

段号=LogicAddress[0]+SegmentR[0]

页号=LogicAddress[1]

块号=Page[段号][页号][2]

Memory[块号][1]=LogicAddress[2]#块内地址=页内地址

delMemoryTable()

for Memoryk in Memory:

treeMemory.insert('','end',values=Memoryk)

else:

print('越界')

messagebox.showerror(title="越界",message="所求逻辑地址未调入内存中")

def DelAll():

global LogicAddress

global SegmentR

global Segment

global Page

global Memory

LogicAddress.clear()

SegmentR.clear()

Segment.clear()

Page.clear()

Memory.clear()

delSegmentTable()

delPageTable()

delMemoryTable()

delSegmentR()

delLogicAdd()

##+++++++++++++++++++++++

#@@@@@@@@按钮区@@@@@@@@

#生成段表页表块表按钮

buttonInput0=Button(window,text="生成段表页表块表",command=BuildTable)

buttonInput0.place(x=500,y=495)

#将页放入块中

buttonInput1=Button(window,text="将段页表装入块表",command=InputMemory)

buttonInput1.place(x=650,y=495)

#将逻辑地址放入并找到物理地址

buttonInput2=Button(window,text="放入逻辑地址",command=InputLogic)

buttonInput2.place(x=500,y=540)

#找物理地址

buttonInput2=Button(window,text="找物理地址",command=FindMemory)

buttonInput2.place(x=650,y=540)

#清空所有存储值

buttondel=Button(window,text="清空",command=DelAll)

buttondel.place(x=800,y=495,width=60,height=60)

#@@@@@@@@@@@@@@@@@@@

window.mainloop()