操作系统原理
实验报告

学生姓名	孙重豪
学　　号	8207191520
专业班级	智能科学与技术 1902
指导老师	宋虹
学　　院	自动化学院
完成时间	2021 年 6 月 19 日

目录

一 实验概述

（一）实验目的

多道系统中，进程与进程之间存在同步与互斥关系。当就绪进程数大于处理机数时， 需按照某种策略决定哪些进程先占用处理机。在可变分区管理方式下，采用首次适应算法实现主存空间的分配和回收。

本实验模拟实现处理机调度及内存分配及回收机制，以对处理机调度的工作原理以及内存管理的工作过程进行更深入的了解。

（二）实验内容及要求

1、实验内容

（1）选择一个调度算法，实现处理机调度；

（2）结合（1）实现主存储器空间的分配和回收。

2、实验具体要求

（1）设计一个抢占式优先权调度算法实现多处理机调度的程序，并且实现在可变分区管理方式下，采用首次适应算法实现主存空间的分配和回收。

（2）PCB 内容包括：进程名/PID；要求运行时间（单位时间）；优先权；状态；进程属性：独立进程、同步进程（前趋、后继）。

（3）可以随机输入若干进程，可随时添加进程，并按优先权排序；

（4）从就绪队首选进程运行：优先权-1；要求运行时间-1；要求运行时间为 0 时，撤销该进程；一个时间片结束后重新排序，进行下轮调度；

（5）自行假设主存空间大小，预设操作系统所占大小并构造未分分区表。表目内容：起址、长度、状态（未分/空表目）。对内存空间分配采用首次适应算法。

（6）进程完成后，回收主存，并与相邻空闲分区合并。

（7）设置后备队列和挂起状态。若内存空间足够，可自动从后备队列调度一作业进入。被挂起进程入挂起队列，设置解挂功能用于将制定挂起进程解挂入就绪队列。

（8）最好采用图形界面；

二 需求分析

（一）处理机器调度

处理机调度算法采用抢占式优先权调度算法，进程控制块 PCB 的内容要求包括： 进程名/PID；要求运行时间（单位时间）；优先权；状态；进程属性：独立进程、同步进程（前趋、后继）。由于需要实现内存调度，故还需添加进程的内存属性。根据处理机调度的特点，还需要设置了运行队列、就绪队列、后备队列、挂起队列、完成队列、阻塞队列。任务要求可以随机输入若干进程，可随时添加进程，故还需要设计初始化产生进程，并且使用多线程在程序运行的时候随时添加进程。在进程运行的时候要求优先级-1，时间片-1，故选择轮转的时间片为 1，同时选择手动挂起和解挂。

（二）内存管理

在内存调度模块中，内存分配与回收选用动态分区分配中的首次适应分区分配算法，根据内存特点设计内存模块类，变量包括内存的起始地址、长度、状态以及该内存属于的进程的名字，除此之外还设置了未分分区表和已分分区表，与其相关关的函数主要有内存回收函数、内存分配函数、未分分区合并函数。

（三）可视化界面

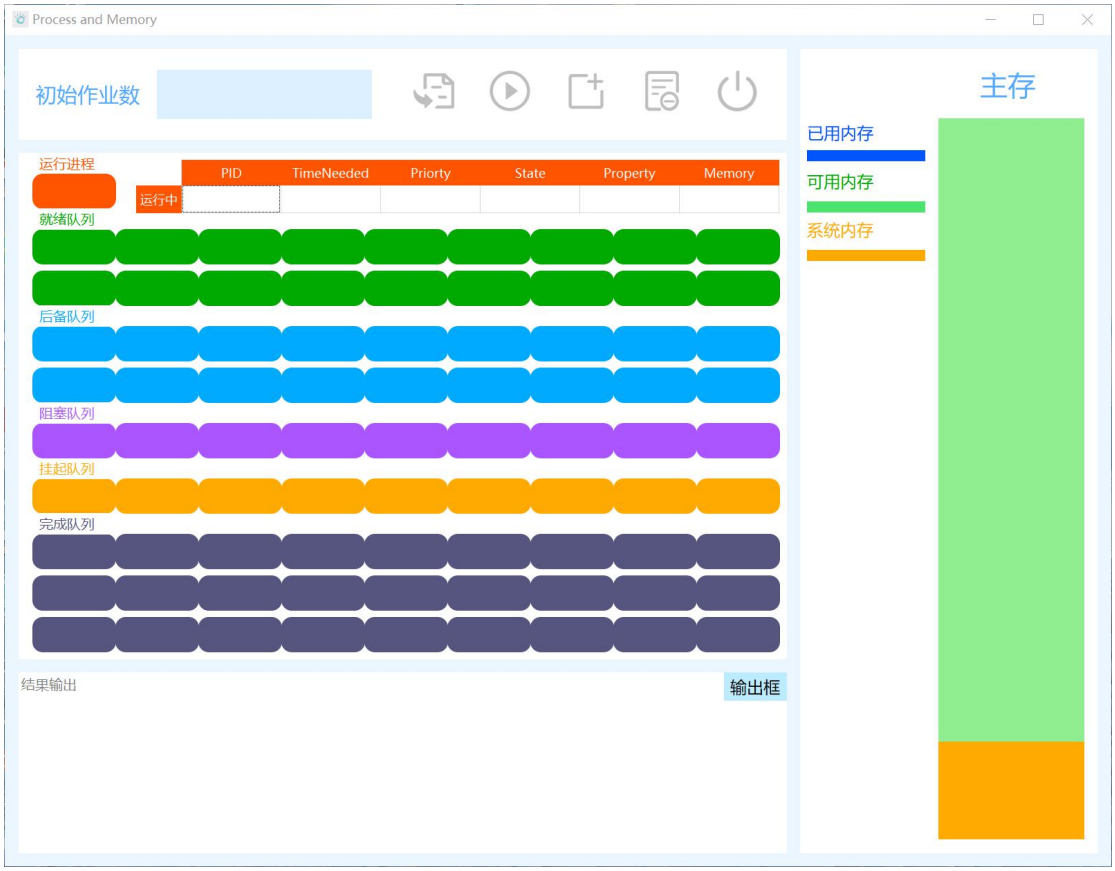
对于可视化界面主要有四个部分，首先就是各类按钮，它们是连接界面与后端功能的桥梁，然后就是各队列的展示，要求能够动态地显示变化，最重要的就是内存的展示，不仅要实现内存调度的动态变化，还要显示各内存属于哪个进程以及要实现空闲分区的合并，最后就是进程运行信息的展示，通过一个输出框就可以实现。

三 总体设计

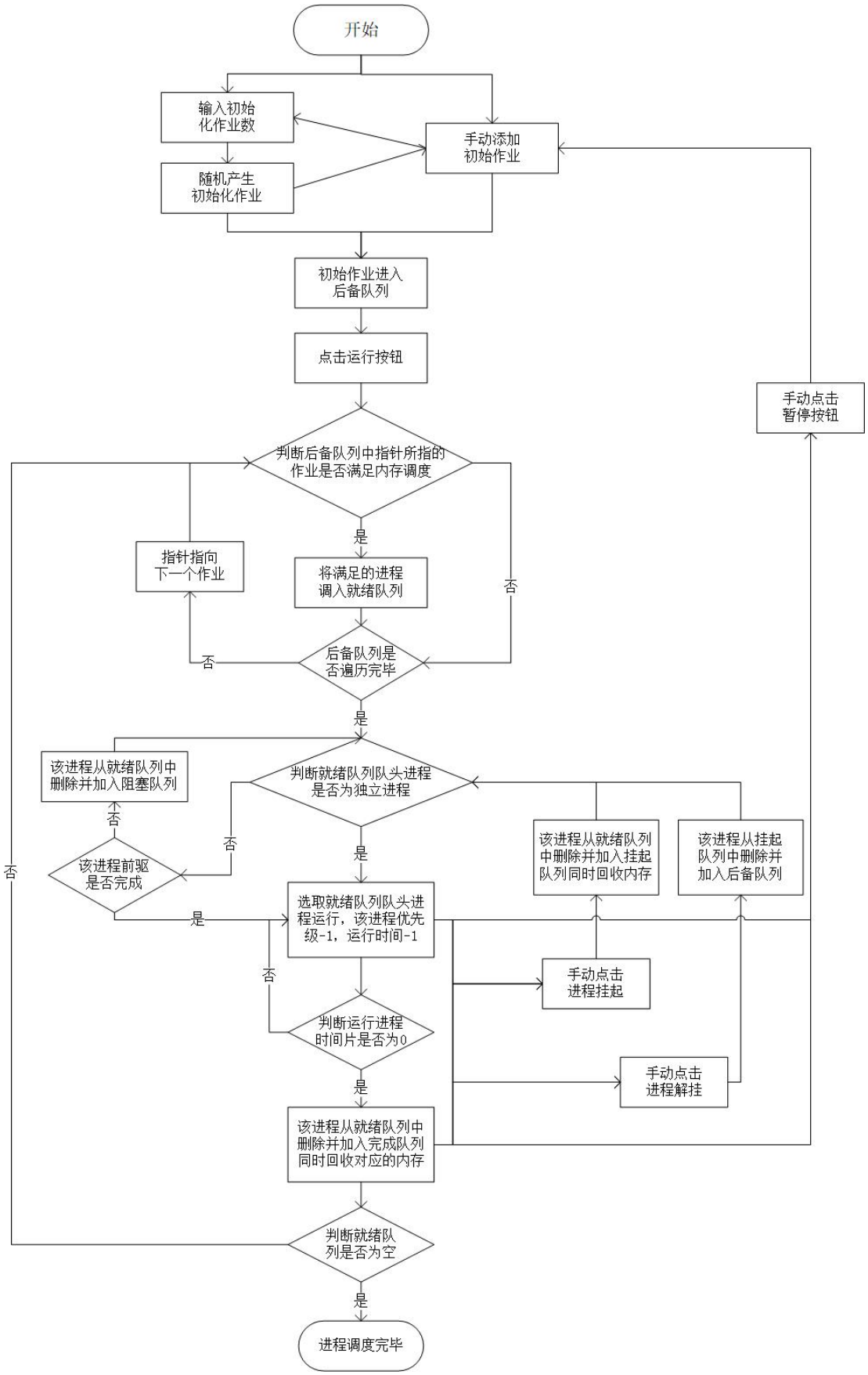
（一）整体结构框架

总体结构我采取模块化方式进行程序设计，要求程序的功能设计、数据结构设计以及整体结构设计合理，我也根据自己对题目的理解增加了新的功能模块。

在程序的总体结构框架上，我是以主界面（如下图）为底层基础的，所有的功能我都采取模块化方式进行程序设计，这样使得程序的功能设计、数据结构设计以及整体结构设计合理。界面的每一块区域对应后端的每一各模块，左边部分对应的是进程调度，右边部分对应的是内存管理。而在这两个模块之内又分了许多小的功能模块，小的功能模块对应的便是各个具体的功能函数。



（二）运行流程图


开始
输入初始化作业数
随机产生初始化作业
手动添加初始作业
初始作业进入后备队列
点击运行按钮
判断后备队列中指针所指的作业是否满足内存调度
是
否
指针指向下一个作业
将满足的进程调入就绪队列
后备队列是否遍历完毕
否
是
该进程从就绪队列中删除并加入阻塞队列
判断就绪队列队头进程是否为独立进程
否
该进程前驱是否完成
否
是
是
该进程从就绪队列中删除并加入挂起队列同时回收内存
该进程从挂起队列中删除并加入后备队列
选取就绪队列队头进程运行，该进程优先级-1，运行时间-1
否
手动点击进程挂起
判断运行进程时间片是否为0
手动点击进程解挂
是
该进程从就绪队列中删除并加入完成队列同时回收对应的内存
判断就绪队列是否为空
否
是
进程调度完毕
手动点击暂停按钮

四 详细设计与实现

（一）实验环境

本课设是是基于 Anaconda 中的 python3.8.2 环境，使用 vscode 编辑器完成的，用到的库主要有：os、sys、time、threading 等，如下所示，界面是基于 pySide2 库设计的。

```
import os
import sys
import time
from threading import Thread,Timer
import random
from PySide2.QtWidgets import *
from PySide2.QtCore import *
from PySide2.QtGui import *
from PySide2 import QtCore, QtGui, QtWidgets
from PySide2.QtUiTools import QUiLoader
```

（二）界面设计

由于有三个窗口需要展示，故定义了一个主窗口类和两个子窗口类，在类的初始化函数中是一些加载界面的语句以及连接界面与后端的语句，如下：

```
class Subwindow1(QWidget):
    def __init__(self):
        super().__init__()
        qfile_stats=QFile(os.path.split(os.path.realpath(__fil
         e__))[0]+'/subwindow1.ui')
        qfile_stats.open(QFile.ReadOnly)
        qfile_stats.close()
        self.ui=QUiLoader().load(qfile_stats)
        self.ui.pushButton.clicked.connect(self.gettext)
        self.ui.pushButton.clicked.connect(self.ui.close)
class Subwindow2(QWidget):
    def __init__(self):
        super().__init__()
        qfile_stats=QFile(os.path.split(os.path.realpath(__fil
         e__))[0]+'/subwindow2.ui')
        qfile_stats.open(QFile.ReadOnly)
        qfile_stats.close()
        self.ui=QUiLoader().load(qfile_stats)
        self.ui.pushButton.clicked.connect(self.ui.close)
        self.ui.pushButton_2.clicked.connect(self.ui.close)
        self.ui.pushButton.clicked.connect(self.h1)
        self.ui.pushButton_2.clicked.connect(self.h2)
```

```
class Mainwindow(QWidget):
    def __init__(self):
        super().__init__()
        qfile_stats=QFile(os.path.split(os.path.realpath(__fil
         e__))[0]+'/mainwindow.ui')
        qfile_stats.open(QFile.ReadOnly)
        qfile_stats.close()
        self.ui=QUiLoader().load(qfile_stats)
        self.ms=MySignals()
        canvas = QtGui.QPixmap(211, 1041)
        canvas.fill(QColor("lightgreen"))
        self.ui.label_4.setPixmap(canvas)
        self.down=False
        self.ms.text_print.connect(self.printToGui)
        self.ms.draww1.connect(self.d1)
        self.ms.draww2.connect(self.d2)
        self.ui.pushButton.clicked.connect(self.onButtonClick)
        self.ui.pushButton_5.clicked.connect(self.ButtonClick)
        self.ui.pushButton_3.clicked.connect(self.ui.close)
        self.ui.pushButton_4.clicked.connect(self.create)
```

在主函数中加载界面：

```
if __name__ == '__main__':
    app = QApplication([])
    app.setWindowIcon(QIcon(os.path.split(os.path.realpath(__f
    ile__))[0]+'/icon/logo.ico'))
    mainwindow=Mainwindow()
    mainwindow.ui.show()
    subwindow1=Subwindow1()
    subwindow2=Subwindow2()
    mainwindow.ui.pushButton_2.clicked.connect(subwindow1.ui)
    subwindow1.ui.pushButton.clicked.connect(mainwindow.puttex
    t)
    mainwindow.ui.pushButton_5.clicked.connect(subwindow2.ui.s
    how)
    subwindow2.ui.pushButton.clicked.connect(mainwindow.hh1)
    subwindow2.ui.pushButton_2.clicked.connect(mainwindow.hh2)
    subwindow2.ui.pushButton.clicked.connect(mainwindow.onButt
    onClick)
    subwindow2.ui.pushButton_2.clicked.connect(mainwindow.onBu
    ttonClick)
    app.exec_()
```

（三）进程调度设计

1 队列设置

```
E1=[]#运行进程
E2=[]#就绪队列
E3=[]#后备队列
E4=[]#挂起队列
E5=[]#完成队列
E6=[]#阻塞队列
```

2 PCB

```
class PCB():
  def __init__(self):
      self.pid = ''      # 进程名称
      self.time=0    # 需要时间片长度
      self.priorty=0      #进程的优先级
      self.state=0      # 进程运行状态，0 表示就绪，1 表示运行，
        2 表示挂起，3 表示阻塞，-1 表示完成   -2 表示在后备队列
      self.property=0  #进程的属性，0 表示独立进程
      self.memory=0 #进程的内存大小
```

3 随机创建作业

```
for i in range(num):
    pcb=PCB()
    pcb.pid='Process_'+str(i+1)
    pcb.time=random.randint(1,10)
    pcb.state=-2
    #pcb.property=0
    if i>1 and num>3 and random.random()<0.2:
      pcb.property=random.randint(1,i)
    else:
      pcb.property=0
      #pcb.memory=random.randint(5,15)*10
      pcb.memory=random.randint(50,150)
    if pcb.property==0:
      pcb.priorty=random.randint(1,20)
    elif pcb.property!=0:
       pcb.priorty=E3[pcb.property-1].priorty-5
    E3.append(pcb)
    p='PID: '+pcb.pid+' 时间片: '+str(pcb.time)+'优先级
          '+str(pcb.priorty)+'状态: '+str(pcb.state)+'属性:
          '+str(pcb.property)+'大小: '+str(pcb.memory)
    self.ui.out.append(p)
    self.ui.out.moveCursor(QTextCursor.End)
```

4 判断进程阻塞

```
while i < len(E2):
  ss='Process_'+str(E2[i].property)
  if E2[i].property==0 or ss in lst:
     runningPCB=E2.pop(i)#进程结点出队
     break
  if E2[i].property!=0 and ss not in lst:
     E6.append(E2.pop(i))#进程阻塞
     i-=1
     for ii in range(len(M2)):
        if E6[-1].pid==M2[ii].name:
           tran=M2.pop(ii)
           tran.m_state=0
           M1.append(tran)
           break
   M1.sort(key=lambda x: x.start,reverse=False)
   M1=self.combine_memory(M1)
   self.drawmemory()
   E2,E3=self.job_scheduling(E2,E3)#从后备队列中调度
   self.fill6(E6)
   i+=1
```

5 进程运行

```
runningPCB.state=1#队头进程进入运行态
runningPCB.time-=1#时间片减一
runningPCB.priorty-=1#优先级减一
self.ui.E1.setText(runningPCB.pid)
self.filltable(runningPCB)
self.fill2(E2)
 time.sleep(pausetime)
```

6 进程挂起

```
for i in range(len(M2)):
   if E4[-1].pid==M2[i].name:
      tran=M2.pop(i)
      tran.m_state=0
      M1.append(tran)
      break
M1.sort(key=lambda x: x.start,reverse=False)
M1=self.combine_memory(M1)
self.drawmemory()
E2,E3=self.job_scheduling(E2,E3)
self.fill2(E2)
self.fill3(E3)
```

```
self.fill4(E4)
self.fill6(E6)
sss=E4[-1].pid+'挂起成功！'
self.ui.out.append(sss)
self.ui.out.moveCursor(QTextCursor.End)
```

7 进程解挂

```
self.fill3(E3)
self.fill4(E4)
self.fill6(E6)
sss=E6[-1].pid+'解挂成功！'
self.ui.out.append(sss)
self.ui.out.moveCursor(QTextCursor.End)
```

8 进程执行完毕

```
runningPCB.state=-1
s4='进程'+runningPCB.pid+'执行完毕！'
self.ms.text_print.emit(self.ui.out,s4)
E5.append(runningPCB)#将进程添加到完成队列
self.fill5(E5)
```

（四）内存管理设计

1 分区表设置

```
M1=[]#未分分区表
M2=[]#已分分区表
```

2 内存块类

```
class Memorylist():
    def __init__(self):
        self.name = ''
        self.start = 0
        self.length=0
        self.m_state=0  #0 表示未分，1 表示已分
```

3 内存分配

```
    def job_scheduling(self,E2,E3):  #作业调度
        global M1
        global M2
        e2=E2
        e3=E3
        i=0
        N=0
        while i < len(e3):
            for j in range(len(M1)):
                if e3[i].memory<=M1[j].length:
```

```
                if N==0:
                  N=1
                    memorylist2=Memorylist()
                    memorylist2.name=e3[i].pid
                    memorylist2.start=M1[j].start
                    memorylist2.length=e3[i].memory
                    memorylist2.m_state=1
                    M2.append(memorylist2)
                    M2.sort(key=lambda x: x.start,reverse=Fals
                    e)
                    M1[j].start=M1[j].start+e3[i].memory
                    M1[j].length=M1[j].length-e3[i].memory
                    e=e3.pop(i)
                    e.state=0
                    e2.append(e)
                    i-=1
                    break
            i+=1

        M1=self.combine_memory(M1)
        e2.sort(key=lambda x: x.priorty,reverse=True)
        self.fill2(e2)
        self.fill3(e3)
        self.drawmemory()
        for i in range(len(e2)):
            e2[i].state=0
        return e2,e3
```

4 内存合并

```
  def combine_memory(self,M1):
        Mm=[]
        ii=0
        while ii < len(M1):
            if M1[ii].length==0:
                ii+=1
            else:
                st= M1[ii].start
                sum=0
                if ii<len(M1)-1:
                    while M1[ii].start+M1[ii].length==M1[ii+1].
                    start:
                        sum+=M1[ii].length
                        ii+=1
```

```
                    if ii==len(M1)-1:
                        break
                sum+=M1[ii].length
                ii+=1
                memorylist1=Memorylist()
                memorylist1.start=st
                memorylist1.length=sum
                Mm.append(memorylist1)
        Mm.sort(key=lambda x: x.start,reverse=False)
        return Mm
```

5 内存绘制

```
def drawmemory(self):
    global M1
    global M2
    for i in range((len(M1))):
      self.ms.draww1.emit(str(M1[i].start),str(M1[i].length))
      #time.sleep(0.1)
    for i in range((len(M2))):
      self.ms.draww2.emit(M2[i].name,str(M2[i].start),str(M2[
      i].length))
```

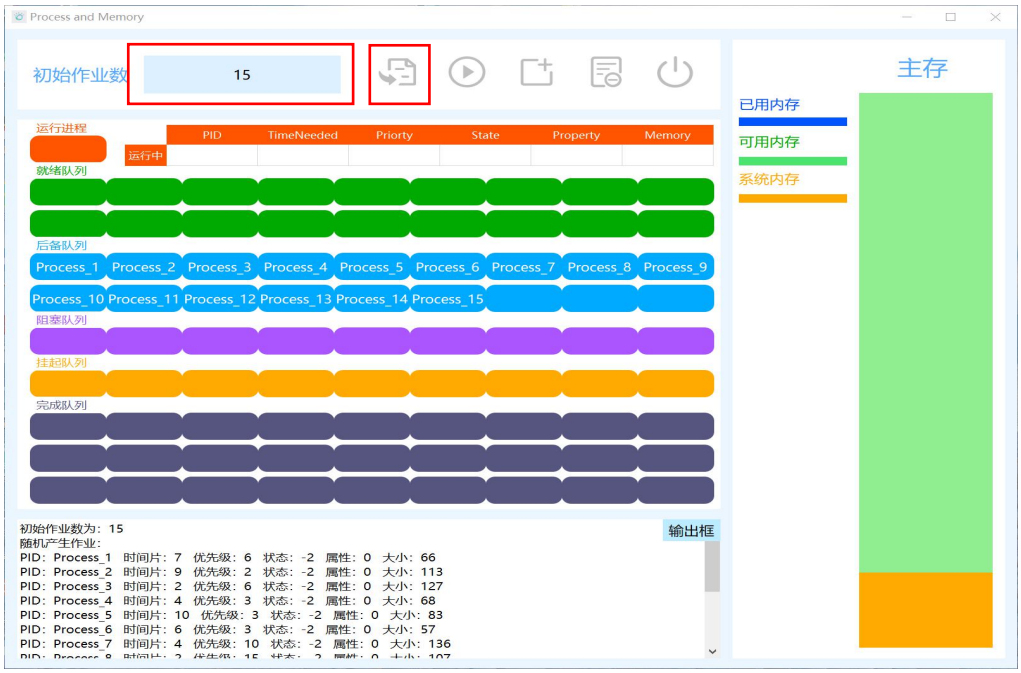
6 内存回收

```
memorylist2=Memorylist()
memorylist2.name=E6[i].pid
memorylist2.start=M1[j].start
memorylist2.length=E6[i].memory
memorylist2.m_state=1
M2.append(memorylist2)
M2.sort(key=lambda x: x.start,reverse=False)
M1[j].start=M1[j].start+E6[i].memory
M1[j].length=M1[j].length-E6[i].memory
M1.sort(key=lambda x: x.start,reverse=False)
M1=self.combine_memory(M1)
self.drawmemory()
E2.append(E6.pop(i))
E2.sort(key=lambda x: x.priorty,reverse=True)
self.fill2(E2)
self.fill6(E6)
i-=1
break
E2,E3=self.job_scheduling(E2,E3)#从后备队列中调度
```

五 实验运行结果

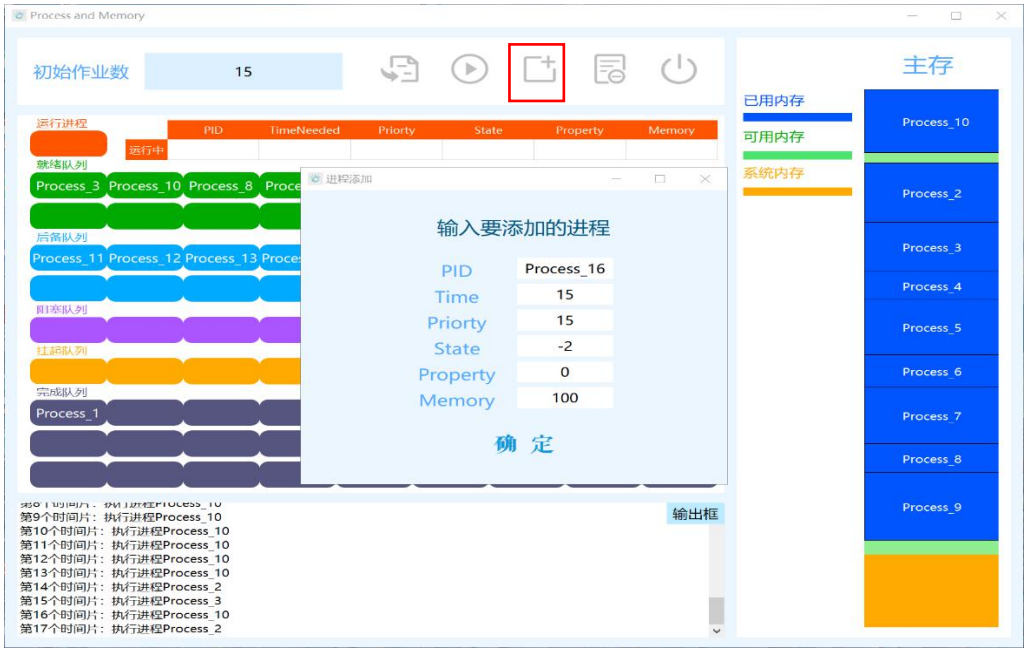
（一）初始随机产生作业

在初始作业数框中输入初始随机产生的作业数，然后点击左一生成按钮，系统便会按照既定规则产生一定数量的作业数（由于还在后备队列中，这里只能称为作业），并在界面上显示各作业的详细信息，如下所图：



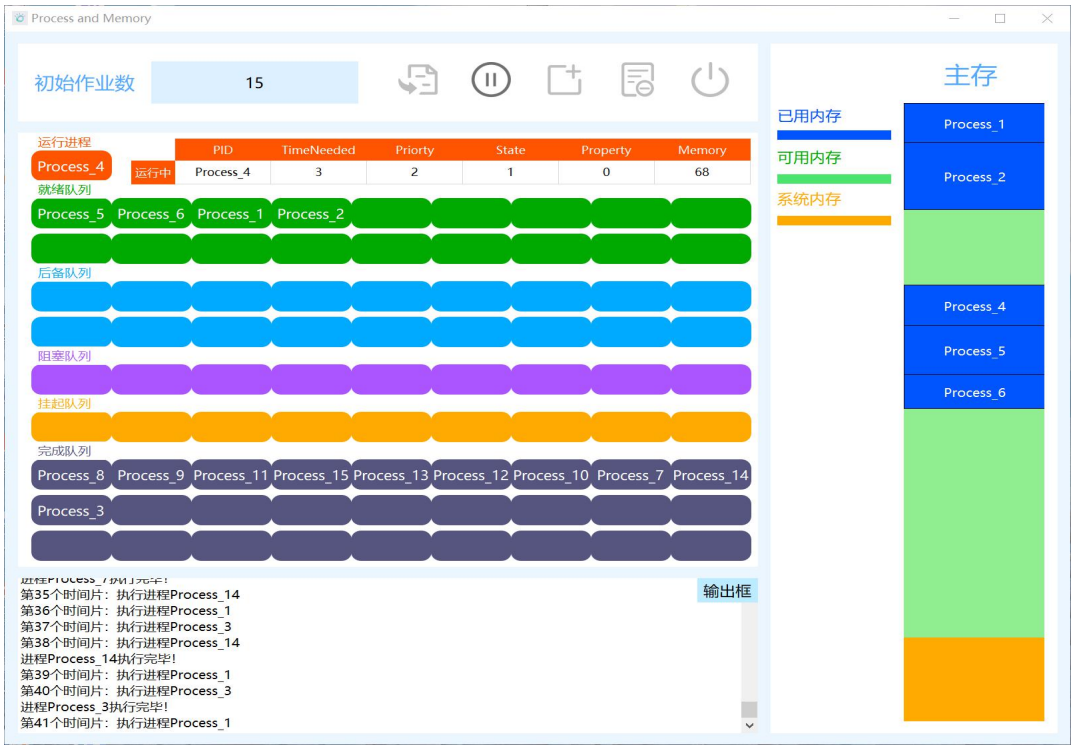
（二）添加作业

添加作业分为两种情况，一种是添加初始作业，另一种是在进程调度的过程中添加作业，两者唯一的区别是后者需要先点击暂停键再添加，它们在后端代码上是没有区别的，我们只需点击中间的添加按钮，然后在输入框输入添加作业的信息，最后点击确定就可以实现进程的添加，具体效果如下图：



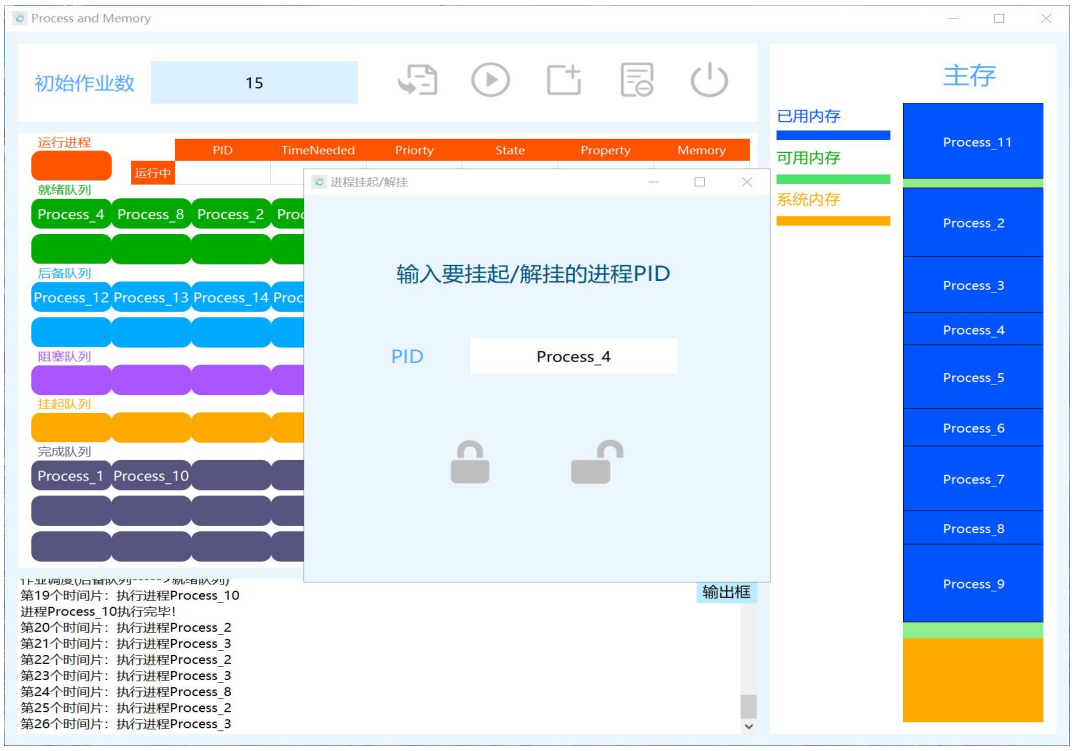
（三）运行过程

在进程调度运行过程中，运行进程表格会动态展示正在运行进程的详细信息，并且后备队列、就绪队列、完成队列也会随着进程调度不断地变化，下方输出框会不停地刷新进程调度信息，除此之外右边主存也会动态显示内存调度的结果，如下图所示：



（四）挂起与解挂

点击右二的挂起与解挂按钮，并输入要操作的进程 PID，即可实现进程的挂起与解挂，如下图所示：

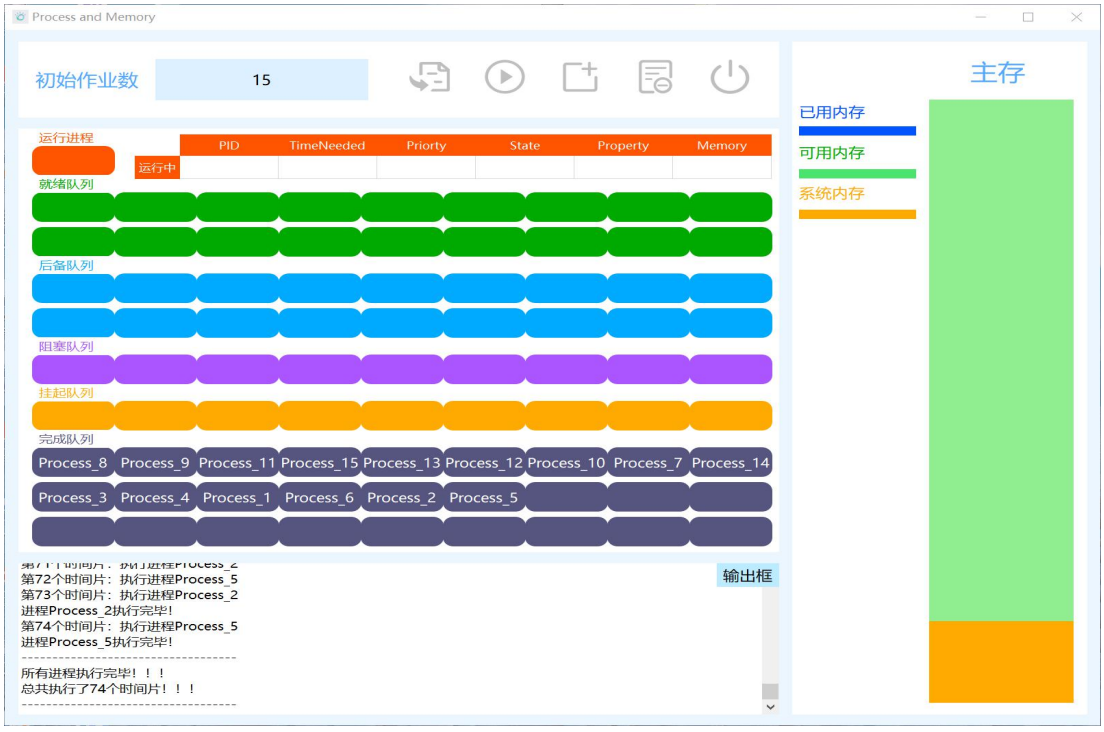


（五）内存展示

在进程运行过程中，内存的展示是不断变化的，严格遵守动态分区分配中的首次适应分区分配算法，如下图所示：

（六）运行完毕

进程调度运行完毕后，处完成队列外，其余队列均清空，同时内存也清空，下方输入框输出“所有进程执行完毕”、所用时间片总数等信息，如下图所示：



结束语

通过本次课程设计让我对于图形界面设计有了一定的思路和看法，同时我对抢占式优先权调度算法和实现主存空间的分配和回收采用的首次适应算法有了更详尽的认识。从这次实验中我发现我对于 python 掌握也有所不足，程序经过了多次修改才得以完善，在以后应该注重编程方面的训练。

此外我还更深入的理解了各个进程调度算法，及实现过程。在编写程序时查询了很多资料，间接提高了我的搜索能力。在此次课程设计过程中，对进程的相关知识有了一定的加深。特别是对进程的进程控制块的存在和价值有了更进一步的认识。在编写程序的过程之中，对进程自身信息的设计和管理以及调度的算法都有助于对书本知识的理解和掌握。特别是设计时间片轮转调度算法的时候，对进程的调度算法有了更好的深入理解。对进程管理中的等待队列，就绪队列，时间片等概念有了更深刻的印象。

在设计此模拟操作系统的课设中，也加深了对 python 知识的把握。解决了一些以往在编程中遇到了困难。通过此次的课程设计，不仅提高了对操作系统的认知，也在同时提高了编程的能力，加强了实践。另外，我觉得此次课程设计虽然主要问题是在编程上，但是经过不断的去调试，还是成功的调试了出来。但是这几个程序用了多天的时间进行分析和修改，虽然出现了不少问题，但收获颇多！

最后，感谢宋老师认真教授我们知识，带我们打开了操作系统之门，我也将不负老师教诲，继续努力！

参考文献

[1] 操作系统实验，https://blog.csdn.net/weixin_42542313/article/details/88720286

[2] python 实现动态高优先权优先调度算法，https://blog.csdn.net/weixin_42437183/article/details/105956483

[3] python 语言实现时间片轮转算法，https://blog.csdn.net/weixin_44882485/article/details/105075969?utm_medium=distribute.pc_relevant_download.none-task-blog-baidujs-1.nonecase&depth_1-utm_source=distribute.pc_relevant_download.none-task-blog-baidujs-1.nonecas

[4] 操作系统实验-内存管理 ，https://blog.csdn.net/weixin_48297950/article//details/112548841