

**操作系统实验报告**

|  |  |
| --- | --- |
| **学 院：** | **计算机科学学院** |
| **专 业：** | **计算机科学与技术（创新实验班）** |
| **学 号：** | **42112255** |
| **姓 名：** | **何佳民** |
| **指导教师：** | **孙增国** |

**2023 年 11 月 20日**

**实验二 时间片轮转RR进程调度算法**

**一、实验目的**

通过这次实验，加深对进程概念的理解，进一步掌握进程状态的转变、进程调度的策略及对系统性能的评价方法。

**二、实验器材**

微机

**三、实验要求**

1）上机前认真复习时间片轮转RR进程调度调度算法，熟悉进程调度的执行过程；

2）上机时独立编程、调试程序；

3）根据具体实验要求，完成好实验报告（包括实验的目的、内容、要求、源程序、实例运行结果截图、发现的问题以及解决方法）

**四、实验内容**

**1.问题描述：**

设计程序模拟进程的时间片轮转RR调度过程。假设有n个进程分别在T1, … ,Tn时刻到达系统，它们需要的服务时间分别为S1, … ,Sn。分别利用不同的时间片大小q，采用时间片轮转RR进程调度算法进行调度，计算每个进程的完成时间、周转时间和带权周转时间，并且统计n个进程的平均周转时间和平均带权周转时间。

**2.程序要求：**

1）进程个数n；每个进程的到达时间T1, … ,Tn和服务时间S1, … ,Sn；输入时间片大小q。

2）要求时间片轮转法RR调度进程运行，计算每个进程的周转时间和带权周转时间，并且计算所有进程的平均周转时间和带权平均周转时间；

3）输出：要求模拟整个调度过程，输出每个时刻的进程运行状态，如“时刻3：进程B开始运行”等等；

4）输出：要求输出计算出来的每个进程的周转时间、带权周转时间、所有进程的平均周转时间以及带权平均周转时间。

**3.问题分析：**

如流程图所示，系统将所有就绪进程按照先来先服务的原则排成一个队列，每次调度把CPU分配给队首进程，并令其执行一个时间片，当执行时间片用完，调度程序停止其执行，并把它送到队列尾部。

时间片够用: 意思就是在该时间片内，进程可以运行至结束，进程运行结束之后，将进程从进程队列中删除，然后启动新的时间片。

时间片不够用:意思是在该时间片内，进程只能完成它的一部分任务，在时间片用完之后，将进程的状态改为等待状态，将进程放到进程队列的尾部，等待cpu的调用。

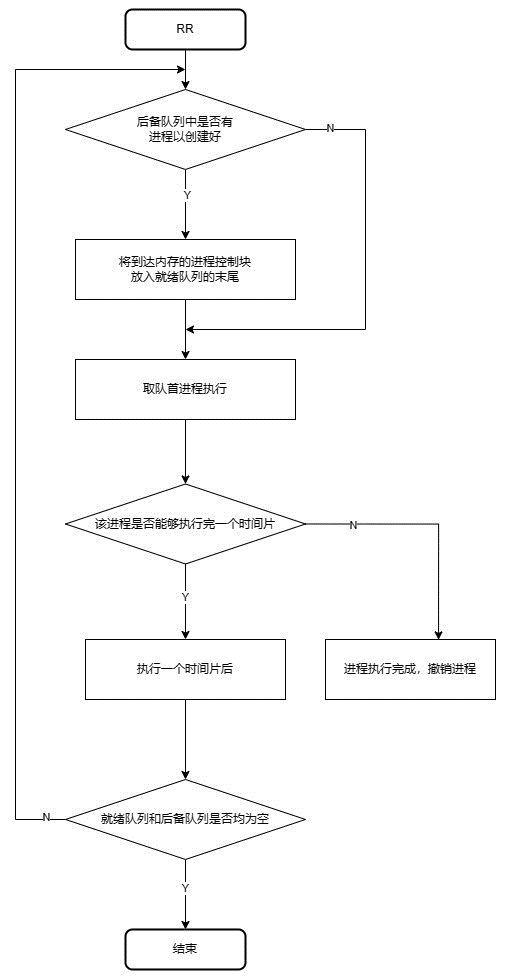


图 2-1 RR算法流程图

**4.源程序：**

**Python代码（Python版本3.9）：**

import tkinter as tk # 导入tkinter库用于创建GUI

from tkinter import ttk

import sys

class PCB:

def \_\_init\_\_(self, name, arrive, serve):

# 初始化PCB类的实例变量

self.name = name # 进程名称

self.arrive = arrive # 到达时间

self.serve = serve # 服务时间

self.remaining\_serve = serve # 剩余服务时间

self.finish = 0 # 完成时间

def \_\_lt\_\_(self, other):

# 比较函数，用于排序

if self.arrive != other.arrive:

return self.arrive < other.arrive

return self.serve < other.serve

def run(pcb\_list, time\_slice):

# 运行时间片轮转调度算法

output = [] # 用于存储输出结果的列表

output.append("当前时刻\t运行进程\t剩余服务时间")

time = 0 # 当前时间

queue = [] # 待处理的进程队列

finished\_processes = 0 # 完成的进程数

original\_pcb\_list = pcb\_list.copy() # 原始进程列表的副本

while pcb\_list and pcb\_list[0].arrive <= time:

queue.append(pcb\_list.pop(0)) # 将已到达的进程加入队列

while finished\_processes < len(original\_pcb\_list):

if queue:

current\_process = queue.pop(0) # 从队列中取出一个进程

for \_ in range(time\_slice):

# 对当前进程进行时间片处理

output.append(f"{time}\t\t{current\_process.name}\t\t{current\_process.remaining\_serve}")

current\_process.remaining\_serve -= 1 # 减少剩余服务时间

time += 1 # 增加当前时间

while pcb\_list and pcb\_list[0].arrive <= time:

queue.append(pcb\_list.pop(0)) # 将新到达的进程加入队列

if current\_process.remaining\_serve == 0:

# 如果当前进程已完成

current\_process.finish = time # 设置完成时间

finished\_processes += 1

break

else:

if current\_process.remaining\_serve > 0:

queue.append(current\_process) # 如果进程未完成，重新加入队列

else:

time += 1 # 如果队列为空，时间继续流逝

return original\_pcb\_list, output # 返回处理后的进程列表和输出结果

def calculate\_turnaround\_time(pcb\_list):

# 计算各进程的周转时间和带权周转时间

output = []

output.append("\n进程名字\t完成时间\t周转时间\t带权周转时间")

total\_turnaround\_time = 0

total\_weighted\_turnaround\_time = 0

for process in pcb\_list:

# 遍历每个进程

turnaround\_time = process.finish - process.arrive # 计算周转时间

weighted\_turnaround\_time = turnaround\_time / process.serve if process.serve else 0 # 计算带权周转时间

total\_turnaround\_time += turnaround\_time

total\_weighted\_turnaround\_time += weighted\_turnaround\_time

output.append(f"{process.name}\t\t{process.finish}\t\t{turnaround\_time}\t\t{weighted\_turnaround\_time:.2f}")

average\_turnaround\_time = total\_turnaround\_time / len(pcb\_list) if pcb\_list else 0

average\_weighted\_turnaround\_time = total\_weighted\_turnaround\_time / len(pcb\_list) if pcb\_list else 0

output.append(f"平均周转时间: {average\_turnaround\_time}")

output.append(f"平均带权周转时间: {average\_weighted\_turnaround\_time}")

return output

class Application(tk.Tk):

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

super().\_\_init\_\_()

self.title("42112255 何佳民 时间片轮转调度算法")

self.create\_widgets() # 创建界面组件

def create\_widgets(self):

# 创建输入框、标签、按钮和文本区域

tk.Label(self, text="进程数").grid(row=0, column=0)

self.n\_entry = tk.Entry(self) # 进程数输入框

self.n\_entry.grid(row=0, column=1)

tk.Label(self, text="时间片").grid(row=1, column=0)

self.q\_entry = tk.Entry(self) # 时间片输入框

self.q\_entry.grid(row=1, column=1)

# 创建进程输入区域

tk.Label(self, text="进程名 到达时间 服务时间（之间空格隔开）").grid(row=2, column=0, columnspan=2)

self.process\_entries = [] # 存储进程输入框的列表

for i in range(10): # 允许输入最多10个进程

entry = tk.Entry(self)

entry.grid(row=3+i, column=0, columnspan=2, sticky="ew")

self.process\_entries.append(entry)

self.run\_button = tk.Button(self, text="运行", command=self.run\_algorithm) # 运行按钮

self.run\_button.grid(row=13, column=0, columnspan=2)

# 创建结果显示区域和滚动条

self.result\_frame = tk.Frame(self) # 创建一个框架容纳文本区域和滚动条

self.result\_area = tk.Text(self.result\_frame, height=30, width=80) # 结果显示文本区域，大小可调整

self.scrollbar = tk.Scrollbar(self.result\_frame, command=self.result\_area.yview) # 创建滚动条

self.result\_area.configure(yscrollcommand=self.scrollbar.set) # 将滚动条与文本区域关联

# 布局文本区域和滚动条

self.result\_area.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

self.scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

# 布局整个框架

self.result\_frame.grid(row=14, column=0, columnspan=2, sticky='nsew')

def run\_algorithm(self):

# 运行算法的逻辑

n = int(self.n\_entry.get()) # 获取进程数

q = int(self.q\_entry.get()) # 获取时间片

pcb\_list = [] # 存储进程的列表

for entry in self.process\_entries[:n]:

name, arrive, serve = entry.get().split() # 获取每个进程的信息

pcb = PCB(name, int(arrive), int(serve)) # 创建PCB对象

pcb\_list.append(pcb) # 添加到列表

pcb\_list.sort() # 对进程列表排序

finished\_pcb\_list, run\_output = run(pcb\_list, q) # 运行算法

turnaround\_output = calculate\_turnaround\_time(finished\_pcb\_list) # 计算周转时间

self.result\_area.delete(1.0, tk.END) # 清空结果显示区域

for line in run\_output + turnaround\_output:

self.result\_area.insert(tk.END, line + '\n') # 将结果输出到文本区域

if \_\_name\_\_ == "\_\_main\_\_":

app = Application() # 创建并运行应用程序

app.mainloop() # 进入事件循环

**5.运行结果与分析：**

用时间片轮转法RR调度进程A、B、C、D和E，时间片q分别为1、2、4、10、20。在Python代码中使用 Tkinter 来创建一个图形用户界面（GUI）。

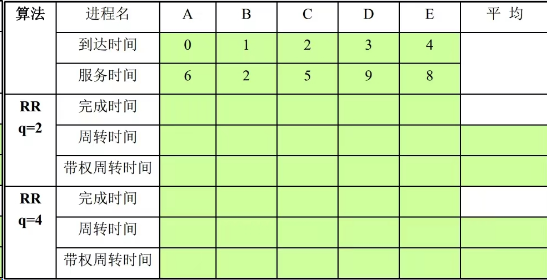


图 2-2 进程的信息截图

结果如下图所示，因为页面限制，所以没有完整截图。

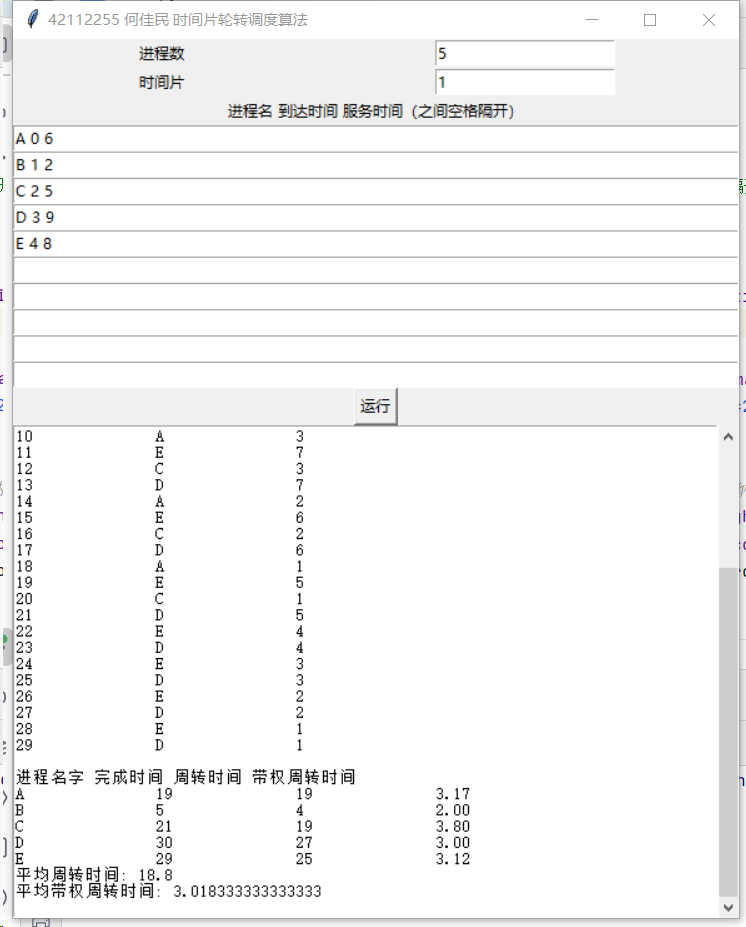


图 2-3 q=1截图

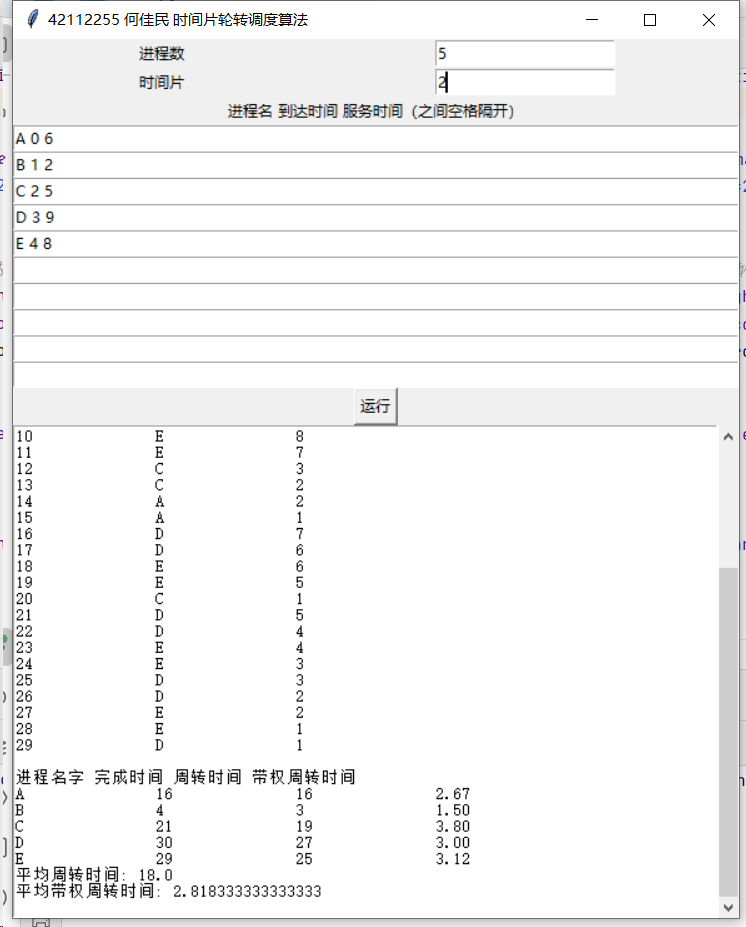


图 2-4 q=2截图

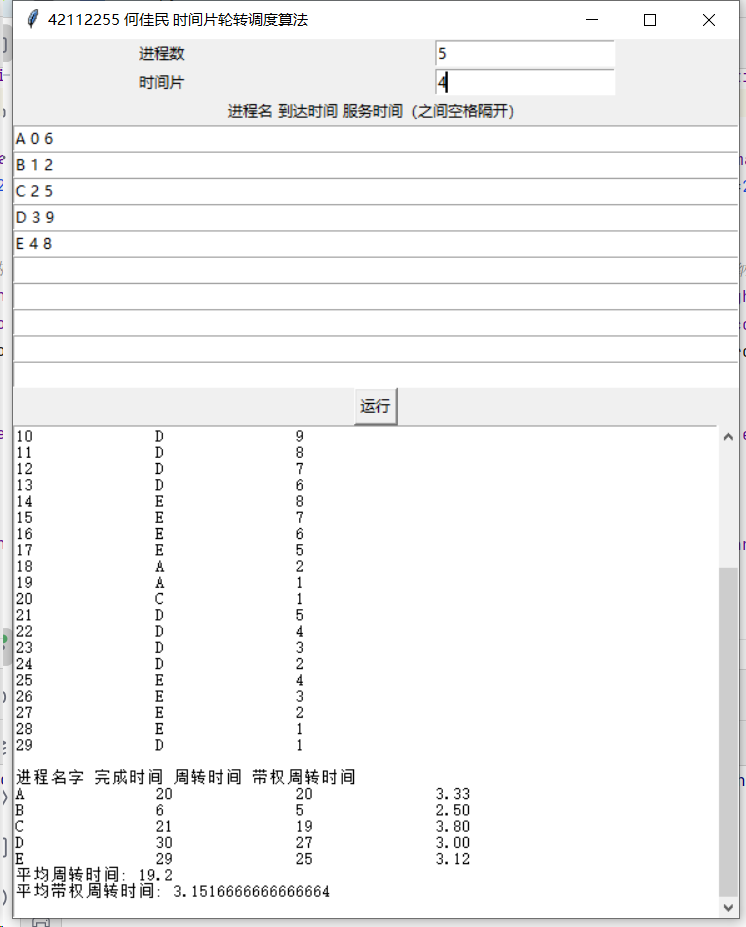


图 2-5 q=4截图

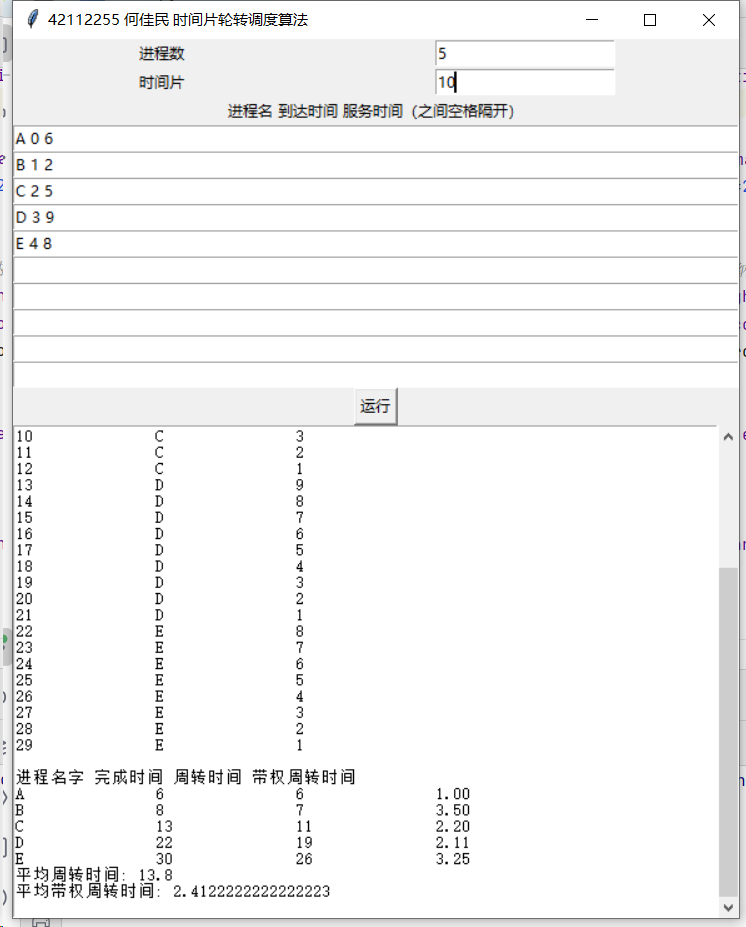


图 2-6 q=10截图

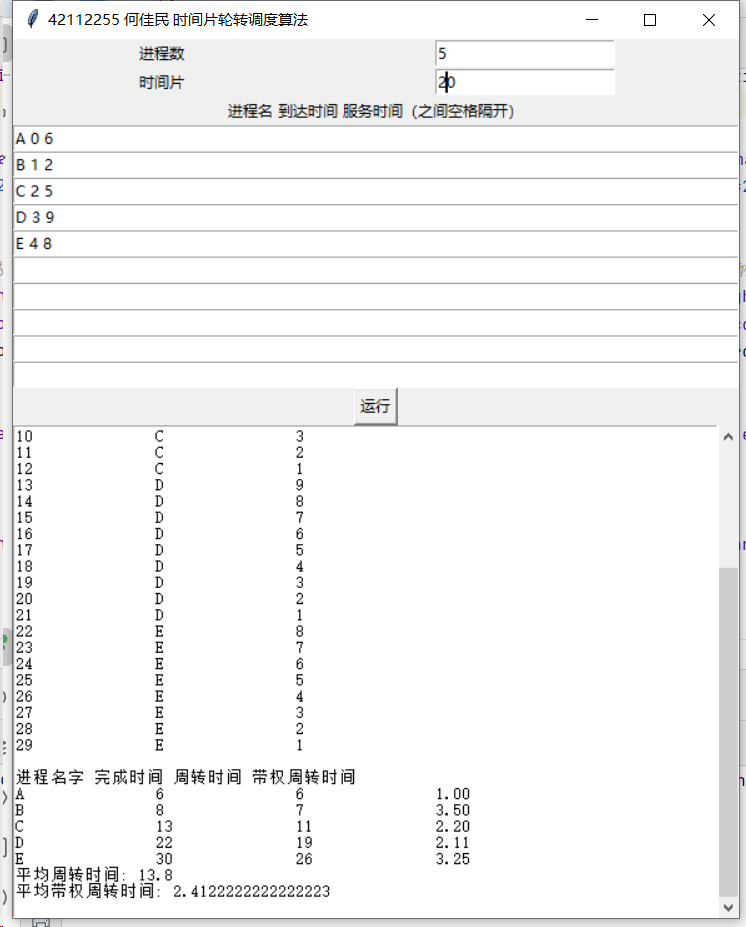


图 2-7 q=20截图

由上图可知，时间片大小如果太小利于短作业，但是会频繁中断，进程上下文切换，增加系统开销;如果太大则每个进程都能在时间片内完成，则退化为FCFS算法，无法满足交互式用户的需求。

**五、实验总结**

1.输入验证不足:如果用户输入的不是数字，或者输入的数字数量不符合预期，程序可能会抛出异常或产生不正确的结果。

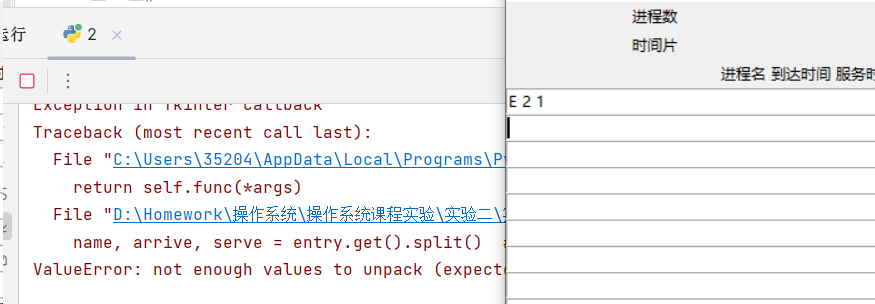


图 2-8 错误截图

解决方法: 加强输入验证，确保用户输入了正确数量的数字，并且所有输入都是有效的数字。

2.多线程和并发处理：在当前的实现中，所有的操作都在主线程中进行，这可能导致用户界面在执行长时间操作时无响应。