

**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 9日**

**实验一 先来先服务FCFS和短作业优先SJF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FCFS和SJF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先来先服务FCFS和短作业优先SJF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。分别采用先来先服务FCFS和短作业优先SJF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；选择算法1-FCFS，2-SJF。

2）要求采用先来先服务FCFS和短作业优先SJF分别调度进程运行，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：**

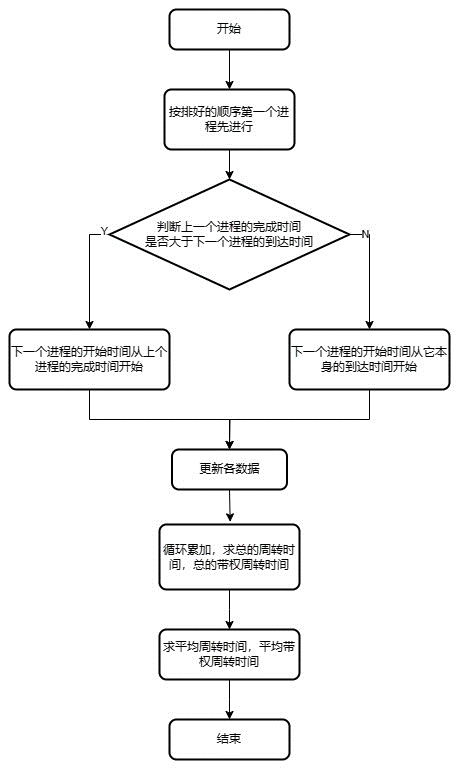


图 1-1 FCFS算法流程图

如流程图所示，先来先服务(FCFS)调度算法是一种最简单的调度算法，该算法既可用于作业调度，也可用于进程调度。当在作业调度中采用该算法时，每次调度都是从后备作业队列中选择一个或多个最先进入该队列的作业，将它们调入内存，为它们分配资源、创建进程，然后放入就绪队列。在进程调度中采用FCFS算法时，则每次调度是从就绪队列中选择一个最先进入该队列的进程，为之分配处理机，使之投入运行。该进程一直运行到完成或发生某事件而阻塞后才放弃处理机。

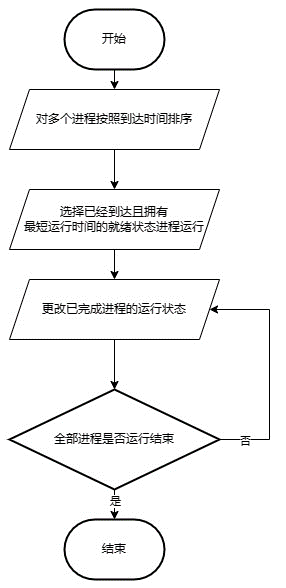


图 1-2 SJF算法流程图

如流程图所示，短作业(进程)优先调度算法SJ(P)F，是指对短作业或短进程优先调度的算法。它们可以分别用于作业调度和进程调度。短作业优先(SJF)的调度算法是从后备队列中选择一个或若干个估计运行时间最短的作业，将它们调入内存运行。而短进程优先(SPF)调度算法则是从就绪队列中选出一个估计运行时间最短的进程，将处理机分配给它，使它立即执行并一直执行到完成，或发生某事件而被阻塞放弃处理机时再重新调度。

**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 messagebox # 导入messagebox模块用于显示弹窗消息

# 定义一个函数来计算进程的完成时间、周转时间和带权周转时间

def calculate\_times(processes, n, by\_service\_time=False):

finish\_times = [0] \* n # 完成时间列表

whole\_times = [0] \* n # 周转时间列表

weight\_whole\_times = [0] \* n # 带权周转时间列表

is\_finished = [False] \* n # 进程是否完成的标志列表

start\_work\_time = 0 # 开始工作的时间

for \_ in range(n): # 对每个进程进行遍历

next\_process = None # 下一个要执行的进程

for i in range(n): # 寻找下一个要执行的进程

if not is\_finished[i] and processes[i][0] <= start\_work\_time:

if next\_process is None or (by\_service\_time and processes[i][1] < processes[next\_process][1]):

next\_process = i

if next\_process is not None: # 如果找到了下一个要执行的进程

is\_finished[next\_process] = True # 标记为已完成

finish\_times[next\_process] = start\_work\_time + processes[next\_process][1]

start\_work\_time = finish\_times[next\_process]

whole\_times[next\_process] = finish\_times[next\_process] - processes[next\_process][0]

weight\_whole\_times[next\_process] = whole\_times[next\_process] / processes[next\_process][1]

return whole\_times, weight\_whole\_times # 返回计算结果

# 定义一个函数来显示结果

def display\_results(processes, whole\_times, weight\_whole\_times):

result = "进程相关信息如下：\n"

for i in range(len(processes)): # 遍历所有进程并构建结果字符串

result += f"进程 {i + 1}: 到达时间 = {processes[i][0]}, 服务时间 = {processes[i][1]}, 完成时间 = {processes[i][0] + whole\_times[i]}, 周转时间 = {whole\_times[i]}, 带权周转时间 = {weight\_whole\_times[i]:.2f}\n"

average\_wt = sum(whole\_times) / len(processes) # 计算平均周转时间

average\_wwt = sum(weight\_whole\_times) / len(processes) # 计算平均带权周转时间

result += f"\n平均周转时间 = {average\_wt:.2f}\n"

result += f"平均带权周转时间 = {average\_wwt:.2f}\n"

messagebox.showinfo("结果", result) # 弹出消息框显示结果

# 定义一个函数来启动算法

def start\_algorithm(choice, arrival\_times, service\_times):

n = len(arrival\_times) # 获取进程数

processes = list(zip(arrival\_times, service\_times)) # 将到达时间和服务时间配对

# 根据用户选择执行不同的算法

if choice == 1: # FCFS算法

whole\_times, weight\_whole\_times = calculate\_times(processes, n)

elif choice == 2: # SJF算法

whole\_times, weight\_whole\_times = calculate\_times(processes, n, by\_service\_time=True)

else: # 无效选择

messagebox.showwarning("警告", "无效的选择。")

return

display\_results(processes, whole\_times, weight\_whole\_times) # 显示结果

# 定义主函数，构建GUI

def main():

def on\_submit(): # 当用户点击提交按钮时执行的函数

try:

n = int(process\_count\_entry.get()) # 获取用户输入的进程数量

arrival\_times = list(map(int, arrival\_time\_entry.get().split())) # 获取到达时间列表

service\_times = list(map(int, service\_time\_entry.get().split())) # 获取服务时间列表

choice = algorithm\_var.get() # 获取用户选择的算法

# 检查输入的进程数量是否与到达时间和服务时间的数量匹配

if len(arrival\_times) != n or len(service\_times) != n:

messagebox.showwarning("警告", "输入的进程数量与到达时间或服务时间数量不匹配。")

return

start\_algorithm(choice, arrival\_times, service\_times) # 启动算法

except ValueError: # 如果输入不是有效的数字

messagebox.showwarning("警告", "请输入有效的数字。")

root = tk.Tk() # 创建Tkinter主窗口

root.title("42112255何佳民 进程调度算法") # 设置窗口标题

# 创建和排列GUI组件

tk.Label(root, text="进程数量：").pack()

process\_count\_entry = tk.Entry(root) # 进程数量输入框

process\_count\_entry.pack()

tk.Label(root, text="到达时间（空格分隔）：").pack()

arrival\_time\_entry = tk.Entry(root) # 到达时间输入框

arrival\_time\_entry.pack()

tk.Label(root, text="服务时间（空格分隔）：").pack()

service\_time\_entry = tk.Entry(root) # 服务时间输入框

service\_time\_entry.pack()

algorithm\_var = tk.IntVar() # 算法选择变量

tk.Radiobutton(root, text="FCFS", variable=algorithm\_var, value=1).pack() # FCFS单选按钮

tk.Radiobutton(root, text="SJF", variable=algorithm\_var, value=2).pack() # SJF单选按钮

submit\_button = tk.Button(root, text="开始计算", command=on\_submit) # 提交按钮

submit\_button.pack()

root.mainloop() # 启动Tkinter事件循环

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

main() # 运行主函数

**5.运行结果与分析：**

在 Python 中使用 Tkinter 来创建一个图形用户界面（GUI），包括添加一个窗口，其中包括文本输入框以输入进程数、到达时间和服务时间，以及按钮来选择算法并显示结果。

**5.1 FCFS算法结果:**



图 1-3 FCFS算法结果图

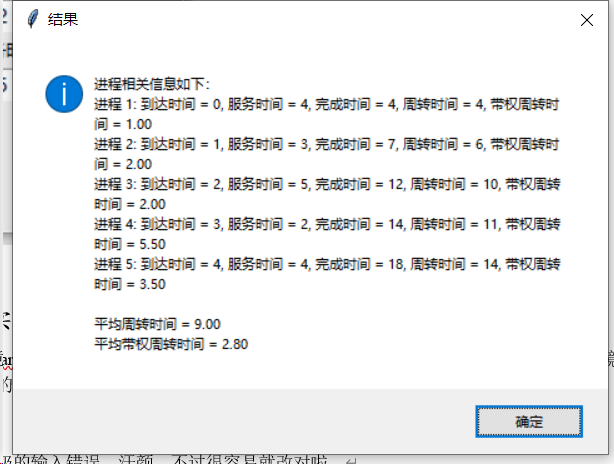


图 1-4 FCFS算法结果图

结果正确。

**5.2 SJF算法结果:**



图 1-5 SJF算法结果图

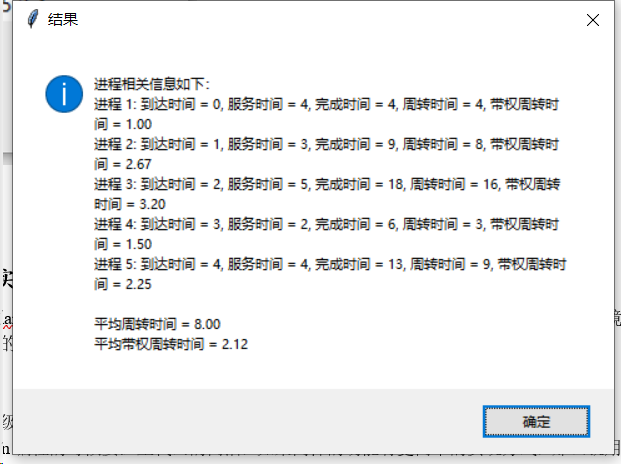


图 1-6 SJF算法结果图

结果正确。

**5.3对FCFS算法的分析：**

如下图所示的1、2、3、4四个进程分别到达系统的时间、要求服务的时间、开始执行的时间及各自的完成时间，并计算出各自的周转时间和带权周转时间



图 1-7 SJF算法结果图

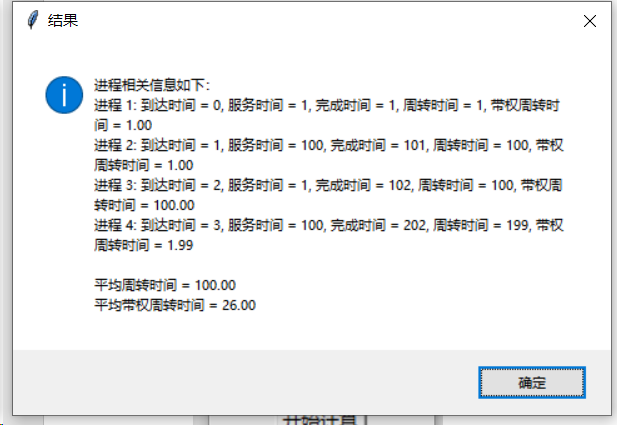


图 1-8 SJF算法结果图

从结果可以看出，其中短进程3的带权周转时间高达100，而长进程4的带权周转时间仅为1.99。可以得出，FCFS算法比较有利于长作业(进程)，而不利于短作业(进程)。

**5.4 SJF与FCFS的比较：**



图 1-9 FCFS算法结果图

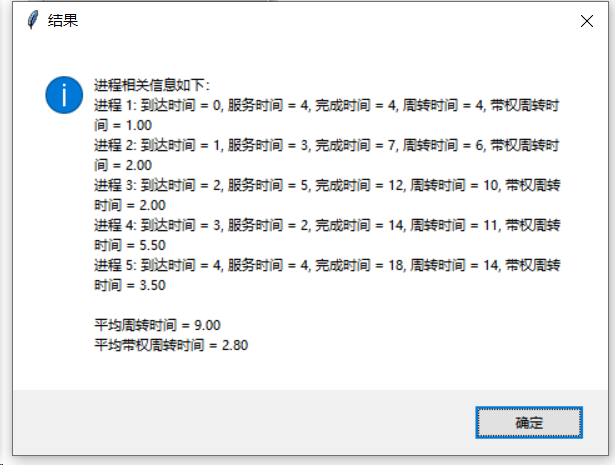


图 1-10 FCFS算法结果图



图 1-11 SJF算法结果图

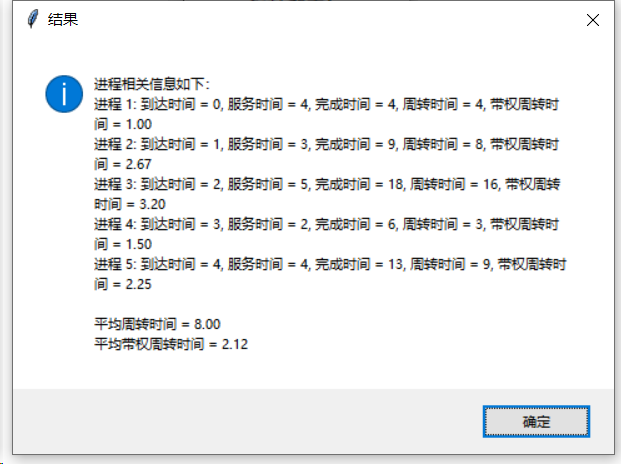


图 1-12 SJF算法结果图

由上图可知，采用SJF算法后，不论是平均周转时间还是平均带权周转时间，都有较明显的改善，尤其是对短作业4，其周转时间由原来的11降为3；而平均带权周转时间是从5.5降到1.5。这说明SJF调度算法能有效地降低作业的平均等待时间。

SJF算法有以下缺点：1.该算法对长作业不利，如作业3的周转时间由10增至16，其带权周转时间由2增至3.2。2. 该算法完全未考虑作业的紧迫程度，因而不能保证紧迫性作业(进程)会被及时处理。

**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。

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。

整数溢出或除以零错误:

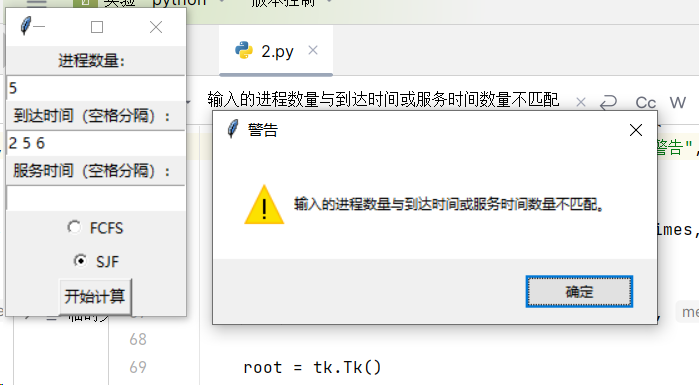


图 1-13 截图

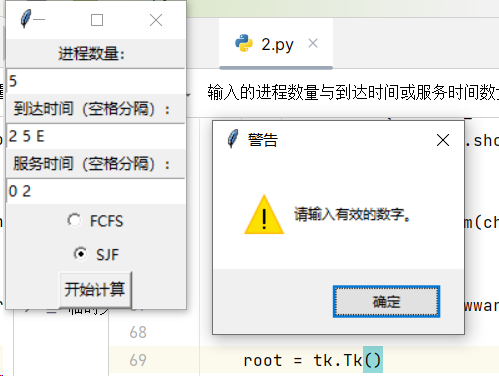


图 1-14 截图

2.如果服务时间为零，计算带权周转时间时可能会发生除以零的错误。

解决方法: 在计算带权周转时间前检查服务时间是否为零，如果是，则处理这种特殊情况。

3.对于极端或不常见的输入（如非常大的进程数量），程序可能不会正确执行。

解决方法: 为边界情况添加特殊处理，如限制输入的最大进程数量。