安徽大学人工智能学院《操作系统》实验报告

学号___WA2214014__ 姓名___杨跃浙___ 年级__大三__

【实验名称】_____实验二　作业调度_____

【实验内容】

在理解作业调度的基础上，实现对一个或多个作业调度的模拟，以加深对作业调度算法的理解。

（1）先来先服务调度算法；

（2）短作业优先调度算法；

（1）先来先服务调度算法

按作业提交的/到达的（到达后备队列的时间）先后次序从外存后备队列中选择几个最先进入该队列的作业为他们分配资源、创建进程，然后再放入就绪队列。

每个作业由一个作业控制块 JCB 表示，JCB 可以包含如下信息：作业名、提交时间、所需的运行时间、作业状态等等。

作业的状态可以是等待 W(Wait)、运行 R(Run)和完成 F(Finish)三种状态之一。每个作业的最初状态总是等待 W。 各个等待的作业按照提交时刻的先后次序排队。

每个作业完成后要输出该作业的开始运行时刻、完成时刻、周转

时间和带权周转时间，这一组作业完成后计算并输出这组作业的平均周转时间、平均带权周转时间。

（2）短作业优先调度算法

根据作业的估计运行时间的长短，从外存后备队列中选择若干个作业为他们分配资源、创建进程，然后再放入就绪队列。

每个作业由一个作业控制块 JCB 表示，JCB 可以包含如下信息：作业名、提交时间、所需的运行时间、作业状态等等。

作业的状态可以是等待 W(Wait)、运行 R(Run)和完成 F(Finish)三种状态之一。每个作业的最初状态总是等待 W。 各个等待的作业按照提交时刻的先后次序排队。

每个作业完成后要输出该作业的开始运行时刻、完成时刻、周转时间和带权周转时间，这一组作业完成后计算并输出这组作业的平均周转时间、平均带权周转时间。

测试用例：

调度算法 \ 作业情况	进程名	A	B	C	D	E	平均
	到达时间	0	1	2	3	4	
	服务时间	4	3	5	2	4	
FCFS (a)	完成时间						
	周转时间						
	带权周转时间						
SJF (b)	完成时间						
	周转时间						
	带权周转时间						

【实验要求】

1、针对作业调度问题，能够分析影响作业调度性能的主要因素，通过设计最优的方案实现作业调度算法。
2、针对不同作业的要求，选择不同的调度算法，满足不同作业，尤其短作业运行的需求。
3、 编写两种作业调度算法：先来先服务调度算法和短作业优先调度算法。

【实验原理】

作业调度是操作系统中管理计算资源的重要环节，其核心目标是通过合理的调度策略，在提高系统性能的同时满足不同类型作业的需求。本实验通过实现和比较两种典型的调度算法：先来先服务（First Come，First Served，FCFS）和短作业优先（Shortest Job First，SJF），深入探讨调度算法的基本原理及其性能表现。

在作业调度中，每个作业使用作业控制块（Job Control Block，JCB）来描述。JCB 包含作业名、到达时间、服务时间以及作业状态等信息。作业的状态通常分为三类：等待（Wait，W）、运行（Run，R）和完成（Finish，F）。初始状态为等待（W），调度器根据不同的调度策略从等待队列中选择作业进行运行。调度的核心问

题在于如何根据作业的特性（如到达时间和服务时间）选择合适的算法，以优化系统性能。

先来先服务（FCFS）

先来先服务调度算法按作业的到达时间排序，优先调度最早到达的作业。具体实现步骤为：

1. 按照作业的到达时间进行排序。
2. 逐一调度作业，每个作业的开始时间为当前时间或其到达时间的较大值。
3. 计算作业的完成时间、周转时间和带权周转时间。

完成时间：

$$T_{finish} = T_{start} + T_{service}$$

周转时间：

$$T = T_{finish} - T_{arrival}$$

带权周转时间：

$$W = \frac{T}{T_{service}}$$

尽管 FCFS 实现简单，但它容易出现“短作业问题”，即长作业的运行会延长短作业的等待时间，从而增加平均周转时间，降低系统的公平性。

短作业优先（SJF）

短作业优先调度算法按照作业的服务时间排序，优先调度服务时间最短的作业。具体实现步骤为：

1. 在系统中维护一个动态的就绪队列，将到达的作业按服务时间从小到大排序。

2. 每次从队列中选择服务时间最短的作业运行。

3. 更新系统当前时间，并重复上述过程，直至所有作业完成。

与 FCFS 不同，SJF 能有效降低平均周转时间，适合服务时间差异较大的场景。但 SJF 的不足之处在于需要提前估计服务时间，且可能导致长作业长期得不到调度，出现“饥饿”现象。

性能指标

为了评估调度算法的性能，本实验主要关注以下指标：

1.平均周转时间：

$$\overline{T} = \frac{\sum T}{N}$$

其中，T 为单个作业的周转时间，N 为总作业数。

2．**平均带权周转时间**：

$$\overline{W} = \frac{\sum W}{N}$$

其中，W 为单个作业的带权周转时间。

时间推进逻辑

在动态调度中，如果当前无作业可运行，系统时间需推进到下一个作业的到达时间：

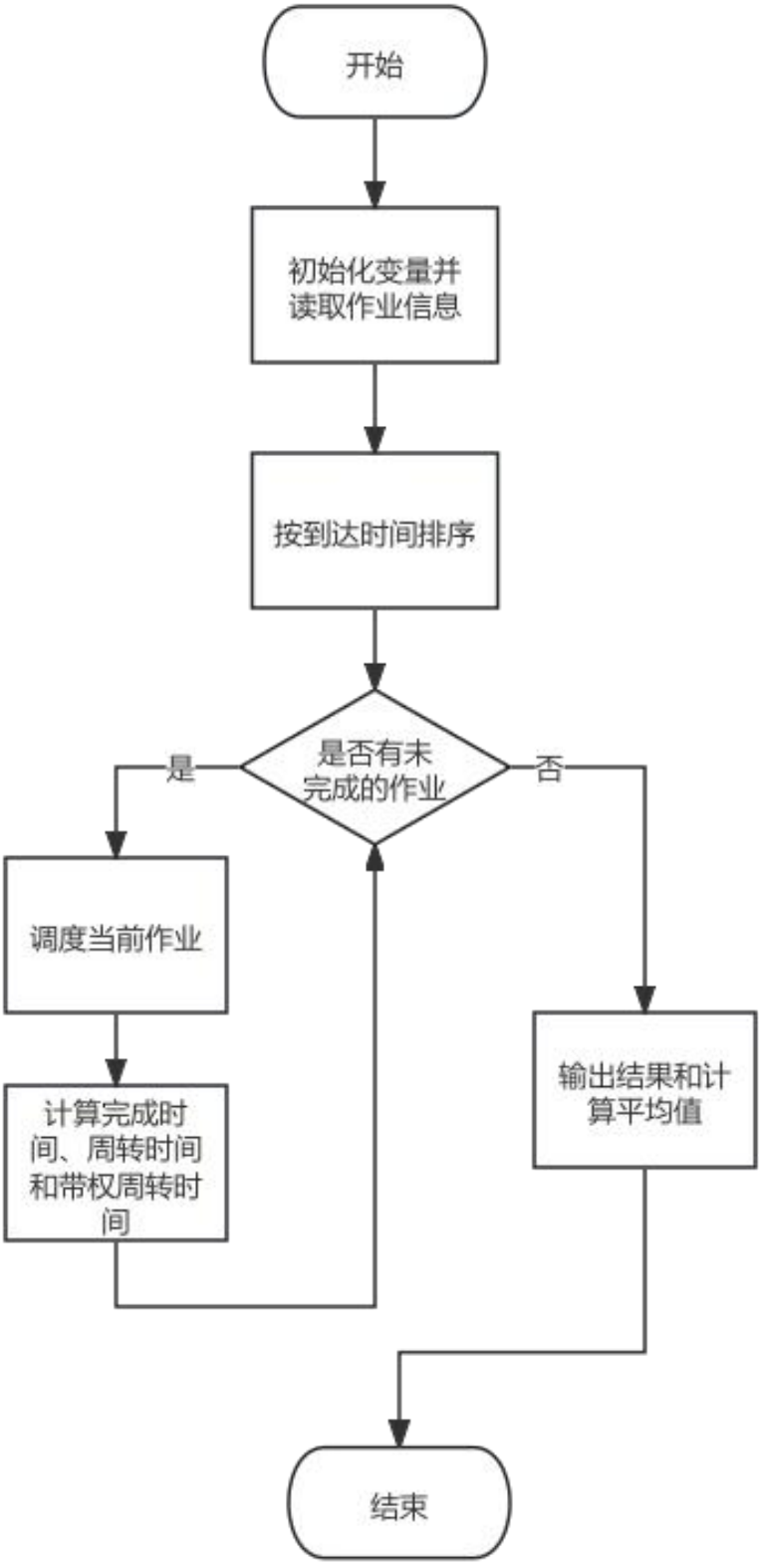
$$T_{current} = \min(T_{arrival})$$

这一机制确保调度器能够及时处理后续作业，避免系统资源空闲。通过本实验，可以清晰地理解 FCFS 和 SJF 的工作原理及其优缺点。FCFS 简单易实现，但公平性较差；SJF 能优化周转时间，但需要准确的服务时间估计。实验结果将通过公式计算的性能指标(如平均周转时间和平均带权周转时间)进行量化分析，进一步验证调度算法在不同场景下的表现。

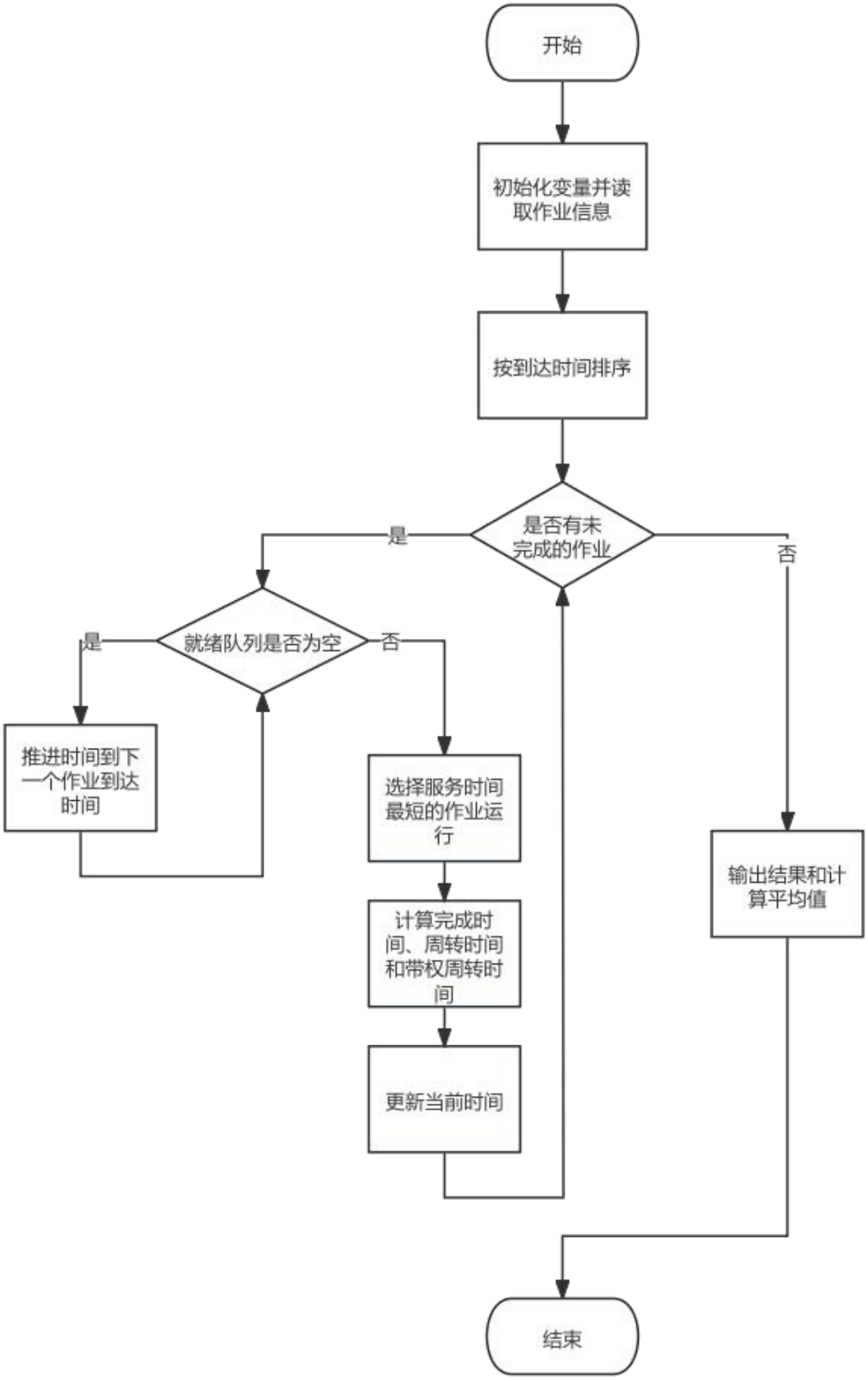
【实验步骤】

1.程序流程图

先来先服务（FCFS）


开始
初始化变量并读取作业信息
按到达时间排序
是否有未完成的作业
是
否
调度当前作业
计算完成时间、周转时间和带权周转时间
输出结果和计算平均值
结束

短作业优先（SJF）


开始
初始化变量并读取作业信息
按到达时间排序
是否有未完成的作业
是
否
就绪队列是否为空
是
否
推进时间到下一个作业到达时间
选择服务时间最短的作业运行
计算完成时间、周转时间和带权周转时间
更新当前时间
输出结果和计算平均值
结束

2.核心代码

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

// 定义 Job 结构体，用于表示作业的相关信息
struct Job {
string name; // 作业名
int arrivalTime; // 到达时间
int serviceTime; // 服务时间（执行时间）
int startTime; // 开始时间
int finishTime; // 完成时间
int turnaroundTime; // 周转时间（完成时间 - 到达时间）
double weightedTurnaroundTime; // 带权周转时间（周转时间 / 服务时间）
};

// 比较函数：按到达时间排序（用于先来先服务 FCFS 调度）
bool compareByArrivalTime(const Job& a, const Job& b) {
return a.arrivalTime < b.arrivalTime;
}

// 比较函数：按服务时间排序（用于短作业优先 SJF 调度）
bool compareByServiceTime(const Job& a, const Job& b) {
return a.serviceTime < b.serviceTime;
}

// 先来先服务调度算法（FCFS）
void FCFS(vector<Job>& jobs) {
// 按到达时间排序
sort(jobs.begin(), jobs.end(), compareByArrivalTime);

int currentTime = 0; // 当前时间
double totalTurnaroundTime = 0; // 总周转时间
double totalWeightedTurnaroundTime = 0; // 总带权周转时间

// 遍历每个作业，按照到达顺序处理
for (auto& job : jobs) {
// 作业开始运行的时间为当前时间或到达时间中的较大值
```

```
job.startTime = max(currentTime, job.arrivalTime);
job.finishTime = job.startTime + job.serviceTime; // 计算完成时间
job.turnaroundTime = job.finishTime - job.arrivalTime; // 计算周转时间
job.weightedTurnaroundTime = (double)job.turnaroundTime / job.serviceTime; // 计算带权周转时间

// 累加总周转时间和总带权周转时间
totalTurnaroundTime += job.turnaroundTime;
totalWeightedTurnaroundTime += job.weightedTurnaroundTime;

// 更新当前时间
currentTime = job.finishTime;
}

// 输出结果
cout << "FCFS 调度结果：" << endl;
cout << "作业名 开始时间 完成时间 周转时间 带权周转时间" << endl;
for (const auto& job : jobs) {
cout << job.name << " " << job.startTime << " "
<< job.finishTime << " " << job.turnaroundTime << " "
<< job.weightedTurnaroundTime << endl;
}

// 输出平均周转时间和平均带权周转时间
cout << "平均周转时间: " << totalTurnaroundTime / jobs.size()
<< endl;
cout << "平均带权周转时间: " << totalWeightedTurnaroundTime /
jobs.size() << endl;
}

// 短作业优先调度算法（SJF）
void SJF(vector<Job>& jobs) {
// 按到达时间排序
sort(jobs.begin(), jobs.end(), compareByArrivalTime);

vector<Job> readyQueue; // 就绪队列，用于存储可运行的作业
int currentTime = 0; // 当前时间
```

```
double totalTurnaroundTime = 0; // 总周转时间
double totalWeightedTurnaroundTime = 0; // 总带权周转时间
int completedJobs = 0; // 完成的作业数量
cout << "SJF 调度结果: " << endl;
cout << "作业名 开始时间 完成时间 周转时间 带权周转时间" << endl;
// 模拟调度过程
while (!jobs.empty() || !readyQueue.empty()) {
// 将已到达的作业加入就绪队列
while (!jobs.empty() && jobs.front().arrivalTime <=
currentTime) {
readyQueue.push_back(jobs.front());
jobs.erase(jobs.begin());
}

// 按服务时间排序，选择短作业优先
sort(readyQueue.begin(), readyQueue.end(),
compareByServiceTime);

if (!readyQueue.empty()) {
// 从就绪队列中选择短作业
auto job = readyQueue.front();
readyQueue.erase(readyQueue.begin());

// 计算作业的开始时间、完成时间、周转时间和带权周转时间
job.startTime = max(currentTime, job.arrivalTime);
job.finishTime = job.startTime + job.serviceTime;
job.turnaroundTime = job.finishTime - job.arrivalTime;
job.weightedTurnaroundTime = (double)job.turnaroundTime /
job.serviceTime;

// 累加总周转时间和总带权周转时间
totalTurnaroundTime += job.turnaroundTime;
totalWeightedTurnaroundTime += job.weightedTurnaroundTime;
completedJobs++; // 更新完成的作业数量

// 更新当前时间
currentTime = job.finishTime;

// 输出作业的调度结果
```

```
cout << job.name << " " << job.startTime << " "
<< job.finishTime << " " << job.turnaroundTime << " "
<< job.weightedTurnaroundTime << endl;
} else {
// 如果没有作业可运行，时间推进
currentTime++;
}
}

// 输出平均周转时间和平均带权周转时间
if (completedJobs > 0) {
cout <<"平均周转时间: " << totalTurnaroundTime / completedJobs
<< endl;
cout << "平均带权周转时间: " << totalWeightedTurnaroundTime /
completedJobs << endl;
}
}

int main() {
// 初始化作业队列
vector<Job> jobs = {
{"A", 0, 4}, {"B", 1, 3}, {"C", 2, 5}, {"D", 3, 2}, {"E", 4,
4}
};

// 运行先来先服务调度算法（FCFS）
vector<Job> jobsFCFS = jobs;
FCFS(jobsFCFS);

cout << endl;

// 运行短作业优先调度算法（SJF）
vector<Job> jobsSJF = jobs;
SJF(jobsSJF);

return 0;
}
```

3.运行结果

```
FCFS 调度结果：
作业名  开始时间  完成时间  周转时间  带权周转时间
A       0         4         4         1
B       4         7         6         2
C       7         12         10         2
D       12         14         11         5.5
E       14         18         14         3.5
平均周转时间： 9
平均带权周转时间： 2.8

SJF 调度结果：
作业名  开始时间  完成时间  周转时间  带权周转时间
A       0         4         4         1
D       4         6         3         1.5
B       6         9         8         2.66667
E       9         13         9         2.25
C       13         18         16         3.2
平均周转时间： 8
平均带权周转时间： 2.12333
```

4.结果分析

以下是实验中定义的初始数据：

输入数据

1.到达时间：

$$A = 0, B = 1, C = 2, D = 3, E = 4$$

2.服务时间：

$$A = 4, B = 3, C = 5, D = 2, E = 4$$

1．先来先服务（FCFS）

调度顺序：按到达时间 A, B, C, D, E

1.作业 A：

· 开始时间= $\max(0,0) = 0$

· 完成时间=0+4=4

· 周转时间=4−0=4

· 带权周转时间=4/4=1.0

2.作业 B：

· 开始时间= $\max(4,1) = 4$

· 完成时间=4+3=7

· 周转时间=7−1=6

· 带权周转时间=6/3=2.0

3. 作业 C：

- 开始时间 = max(7, 2) = 7
- 完成时间 = 7 + 5 = 12
- 周转时间 = 12 − 2 = 10
- 带权周转时间 = 10 / 5 = 2.0

4. 作业 D：

- 开始时间 = max(12, 3) = 12
- 完成时间 = 12 + 2 = 14
- 周转时间 = 14 − 3 = 11
- 带权周转时间 = 11 / 2 = 5.5

5. 作业 E：

- 开始时间 = max(14, 4) = 14
- 完成时间 = 14 + 4 = 18
- 周转时间 = 18 − 4 = 14
- 带权周转时间 = 14 / 4 = 3.5

结果表格

作业	完成时间	周转时间	带权周转时间
A	4	4	1.0
B	7	6	2.0
C	12	10	2.0
D	14	11	5.5
E	18	14	3.5

平均值计算

· 平均周转时间：

$$\overline{T} = \frac{4 + 6 + 10 + 11 + 14}{5} = 9$$

· 平均带权周转时间：

$$\overline{W} = \frac{1.0 + 2.0 + 2.0 + 5.5 + 3.5}{5} = 2.8$$

2．短作业优先（SJF）

调度顺序：按服务时间选择作业，先调度短作业。

1．时间 t = 0 ：只有 A 到达，调度 A

- 开始时间 = 0
- 完成时间 = 0 + 4 = 4
- 周转时间 = 4 − 0 = 4
- 带权周转时间 = 4 / 4 = 1.0

2．时间 t = 4 ：到达的作业有 B，C，D，E ，选择服务时间最短的 D

- 开始时间 = 4

- 完成时间 = 4 + 2 = 6
- 周转时间 = 6 − 3 = 3
- 带权周转时间 = 3 / 2 = 1.5

3. 时间 t = 6 ：剩余作业 B, C, E ,选择服务时间最短的 B

- 开始时间 = 6
- 完成时间 = 6 + 3 = 9
- 周转时间 = 9 − 1 = 8
- 带权周转时间 = 8 / 3 = 2.67

4. 时间 t = 9 ： 剩余作业 C, E , 选择服务时间最短的 E

- 开始时间 = 9
- 完成时间 = 9 + 4 = 13
- 周转时间 = 13 − 4 = 9
- 带权周转时间 = 9 / 4 = 2.25

5. 时间 t = 13 ： 剩余作业 C , 调度 C

- 开始时间 = 13
- 完成时间 = 13 + 5 = 18
- 周转时间 = 18 − 2 = 16
- 带权周转时间 = 16 / 5 = 3.2

结果表格

作业	完成时间	周转时间	带权周转时间
A	4	4	1.0
D	6	3	1.5
B	9	8	2.67

E	13	9	2.25
C	18	16	3.2

平均值计算

· 平均周转时间：

$$\overline{T} = \frac{4+3+8+9+16}{5} = 8$$

· 平均带权周转时间：

$$\overline{W} = \frac{1.0+1.5+2.67+2.25+3.2}{5} = 2.12$$

FCFS 结果

- 平均周转时间： 9
- 平均带权周转时间： 2.8

SJF 结果

- 平均周转时间： 8
- 平均带权周转时间： 2.12

通过手算验证，调度算法的结果是正确的，SJF 在减少平均周转时间和平均带权周转时间方面优于 FCFS，符合预期。

【实验总结】

在本次实验中，通过实现和比较先来先服务（FCFS）和短作业优先（SJF）两种作业调度算法，深入理解了作业调度在操作系统中的基本原理及其对性能的影响。这两种算法在设计思路和适用场景上各有特点，通过代码实现、测试数据验证和性能分析，掌

握了不同调度算法在实际应用中的优势和不足。

1．先来先服务（FCFS）

FCFS 是一种基于到达时间的简单调度策略，其核心思想是按照作业到达的先后顺序依次分配资源。实验中，FCFS 的实现通过对作业按到达时间排序后依次运行来完成。尽管算法简单易实现，但 FCFS 对长作业友好，却容易导致短作业长时间等待，从而增加了系统的平均周转时间，降低了整体公平性。

2．短作业优先（SJF）

SJF 是一种以服务时间为优先级的调度策略，其核心目标是通过优先运行服务时间最短的作业来最小化平均周转时间。实验中，SJF 的实现通过动态维护一个就绪队列，并根据服务时间对队列进行排序。在测试中发现，SJF 对短作业更加友好，显著降低了平均周转时间，但可能因长作业被反复推迟而导致“饥饿”现象。

实验过程中出现的错误及修改过程

1．调度顺序错误

在初次实现 SJF 时，未正确维护就绪队列的动态性，导致服务时间排序未能实时更新。问题的原因在于未在每次时间推进时将新到达的作业加入队列。通过增加一个循环，将所有到达时间小于等于当前时间的作业从等待队列移动到就绪队列，成功解决了这一问题。

2．平均值计算错误

在初始代码中，平均周转时间和平均带权周转时间的分母错误地使用了动态变化的就绪队列大小（readyQueue.size()），而非作业总数（jobs.size()）。这一问题导致最终输出为 inf 或错误的数值。通过改用一个独立变量 completedJobs 记录已完成的作业数量，并以此为分母计算平均值，修复了该问题。

错误的输出：

```
FCFS 调度结果：
作业名  开始时间  完成时间  周转时间  带权周转时间
A       0         4         4         1
B       4         7         6         2
C       7         12         10         2
D       12         14         11         5.5
E       14         18         14         3.5
平均周转时间： 9
平均带权周转时间： 2.8

SJF 调度结果：
作业名  开始时间  完成时间  周转时间  带权周转时间
A       0         4         4         1
D       4         6         3         1.5
B       6         9         8         2.66667
E       9         13         9         2.25
C       13         18         16         3.2
平均周转时间： inf
平均带权周转时间： inf
```

3．时间推进逻辑的缺陷

在调度空闲时未正确推进系统时间，导致部分作业因未及时被加入就绪队列而无法被调度。为解决此问题，增加了一段逻辑，当就绪队列为空且当前时间没有可运行作业时，将当前时间直接跳跃到下一个作业的到达时间。

4．结果验证的错误

初次运行实验时，计算出的部分结果与理论不符。在对公式逐一检查后发现，个别作业的周转时间和带权周转时间未正确更新。通过打印调试信息逐步验证每个作业的状态（开始时间、完成时间等），发现问题并进行修改。

通过本次实验，全面了解了 FCFS 和 SJF 调度算法的设计和实现，尤其是如何动态维护作业队列以及如何计算周转时间和带权周转时间。在 FCFS 中，我们体会到简单算法的优势和局限性；在 SJF 中，则更深刻地理解了如何通过排序优化系统性能，同时也意识到需要权衡公平性和效率。

实验中的调试过程也让我更加熟悉了通过打印调试信息排查逻辑错误的重要性。此外，对于调度算法的性能评估，实验验证了理论结论，即 SJF 能显著降低平均周转时间和平均带权周转时间，但其依赖于服务时间的准确性，并可能导致长作业的饥饿问题。

通过本次实验，我不仅加深了对调度算法理论的理解，还提高了编码实现能力，尤其是在处理动态数据和逻辑错误时的调试能力。本实验的结果也为后续研究更复杂的调度算法（如多级反馈队列调度）奠定了基础。