**实验五 进程调度**

# 姓名徐昊博学号21013134成绩实验时间2023年12月7日 指导教师(签名)

**（诚信声明：本实验报告内容，均由本人亲自上机完成。 签名：\_\_\_\_\_\_\_\_\_\_\_\_\_）**

一．实验目的

1.深入了解进程调度的策略和机制

2.掌握各种调度算法的基本原理和评价指标

3.实现几种常用的调度算法

二．实验工具与设备

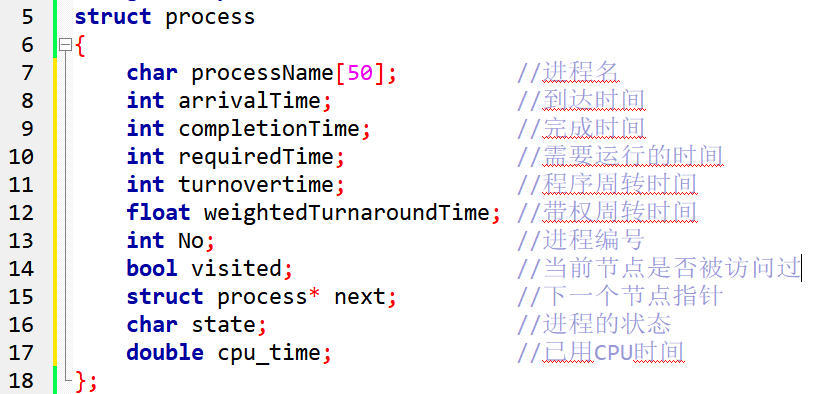
装有 Linux 操作系统的计算机。

三．实验内容

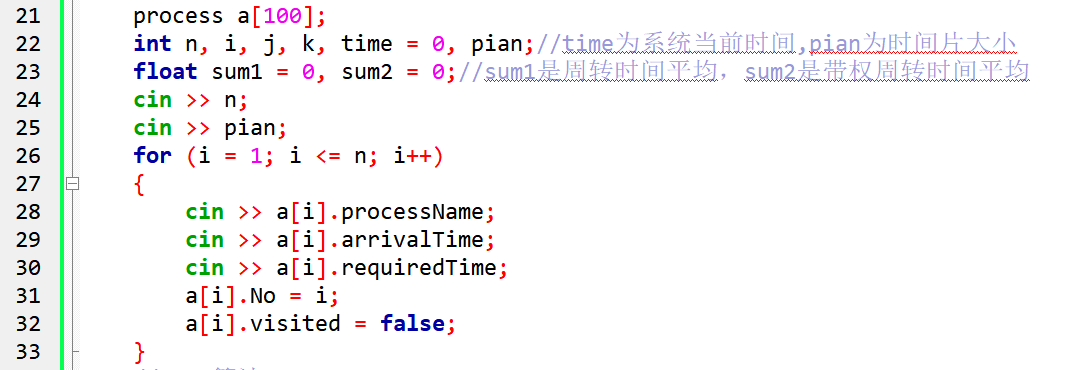
1. 用 C 语言实现 FCFS、SPN 和 RR 算法

（可另行加页打印）

一、首先先将与进程有关的变量进行结构化，将一个进程的下列信息装在一个结构体当中以便于后续操作，相关代码如下：



二、变量初始化，代码如下：



1. FCFS

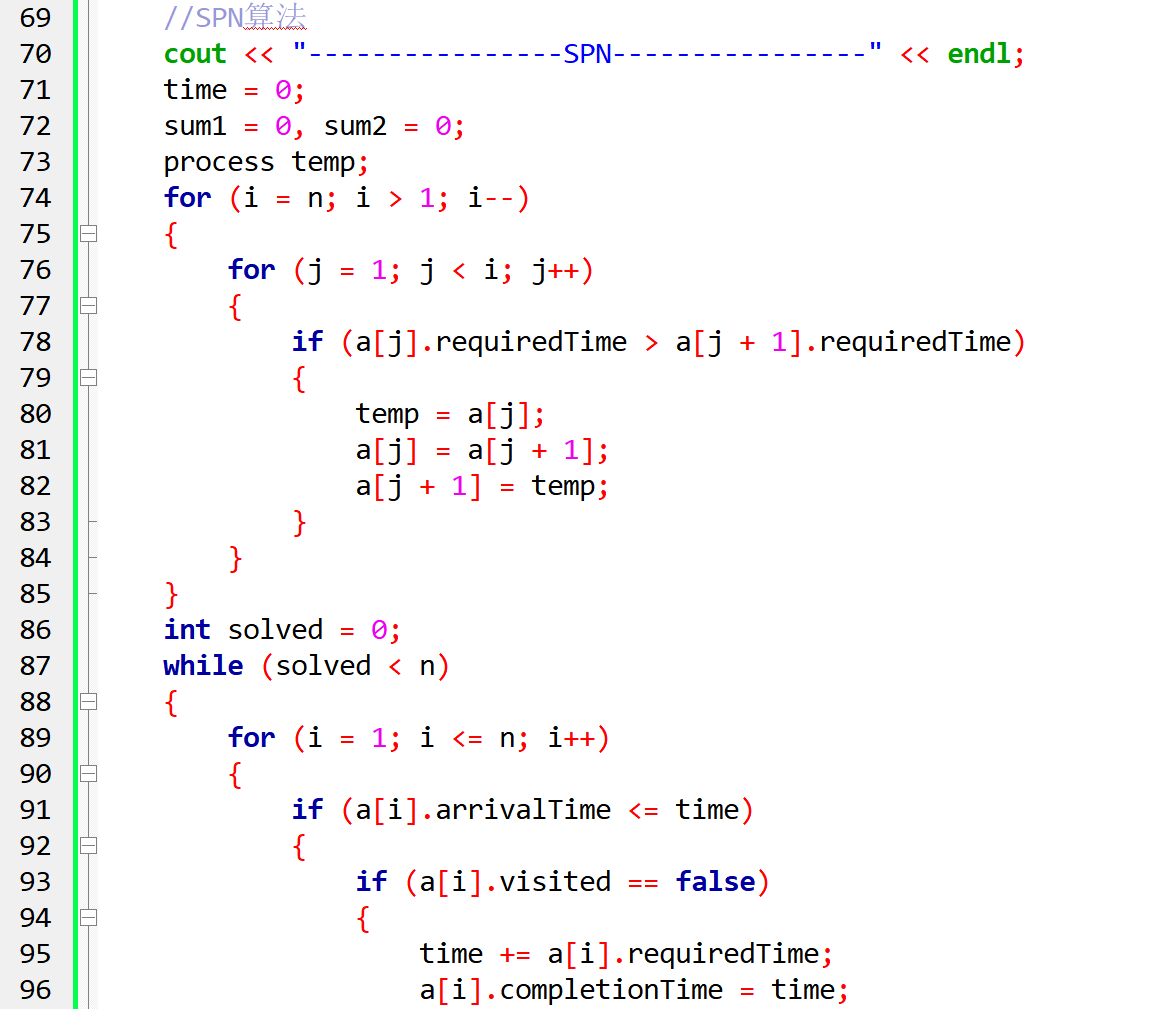
相关代码：

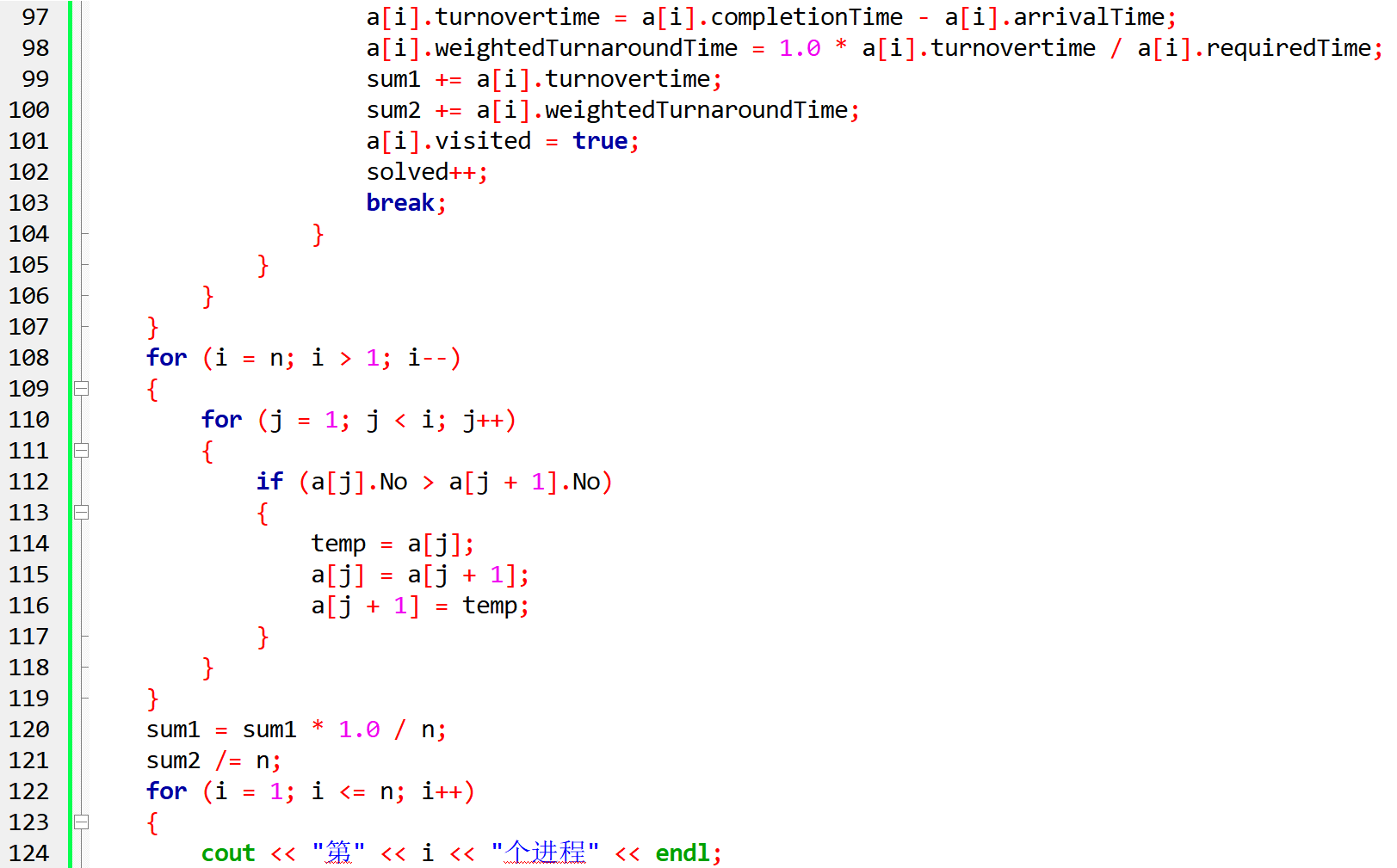


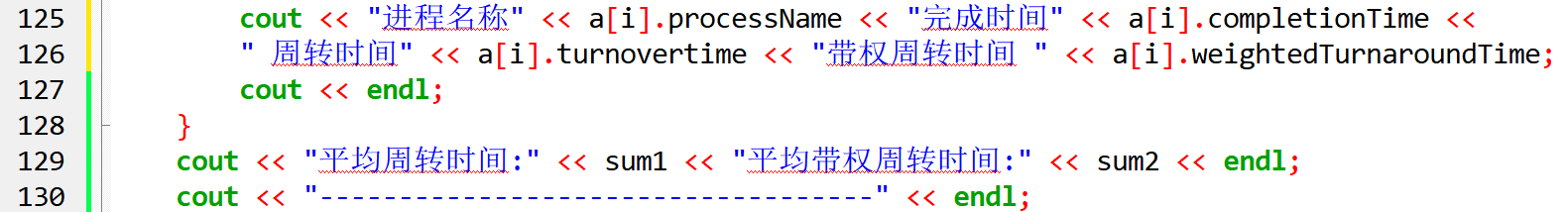
说明：根据先来先服务算法按照数组的顺序进行每个进程的计算，当进程的到达时间小于当前的时间时，该进程的开始时间就是到达时间，否则按照不抢夺原则，需要等到进程结束后的当前时间才是进程的实际开始时间，之后进行周转时间等相关计算。

1. SPN

相关代码：





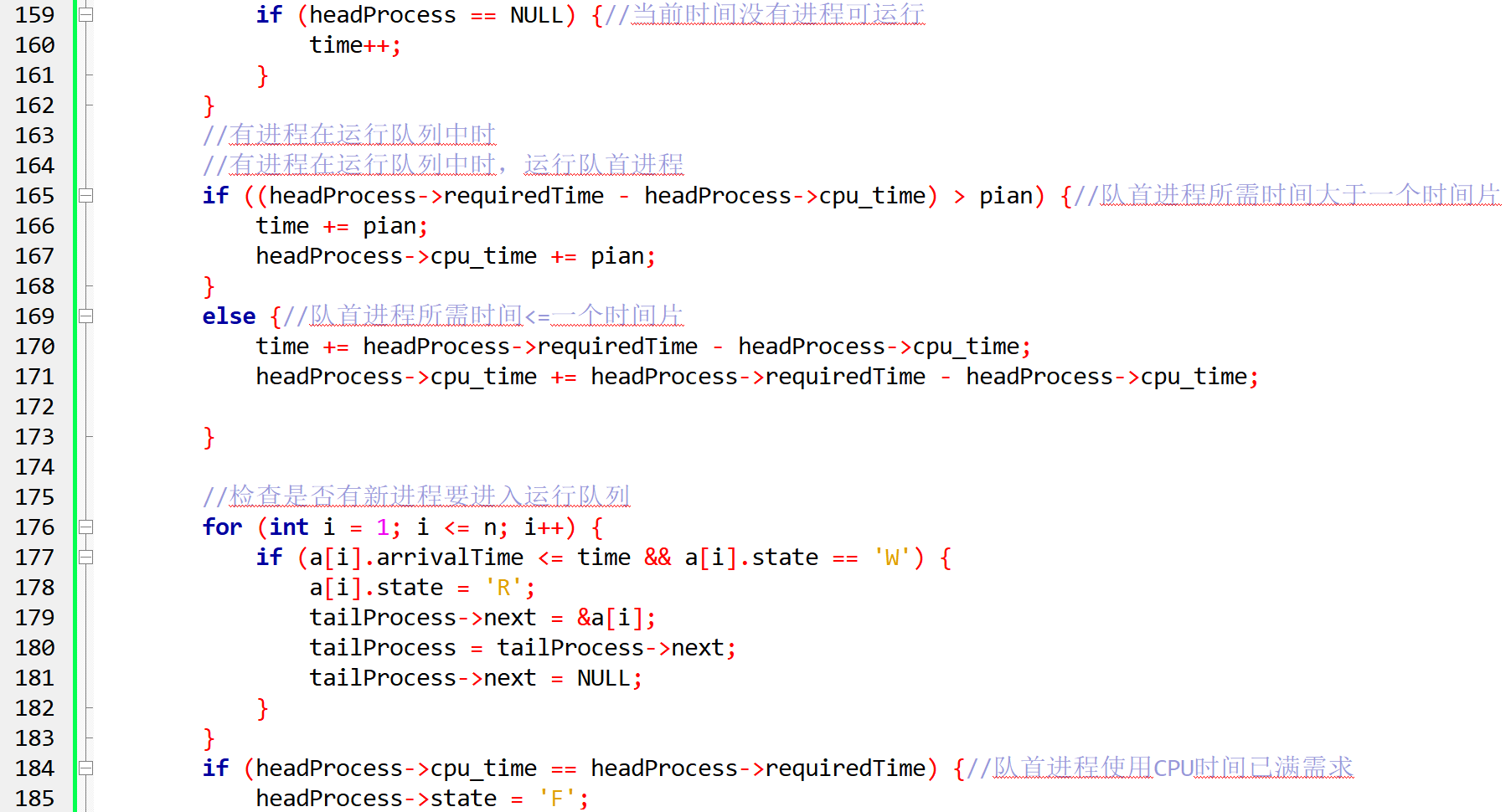


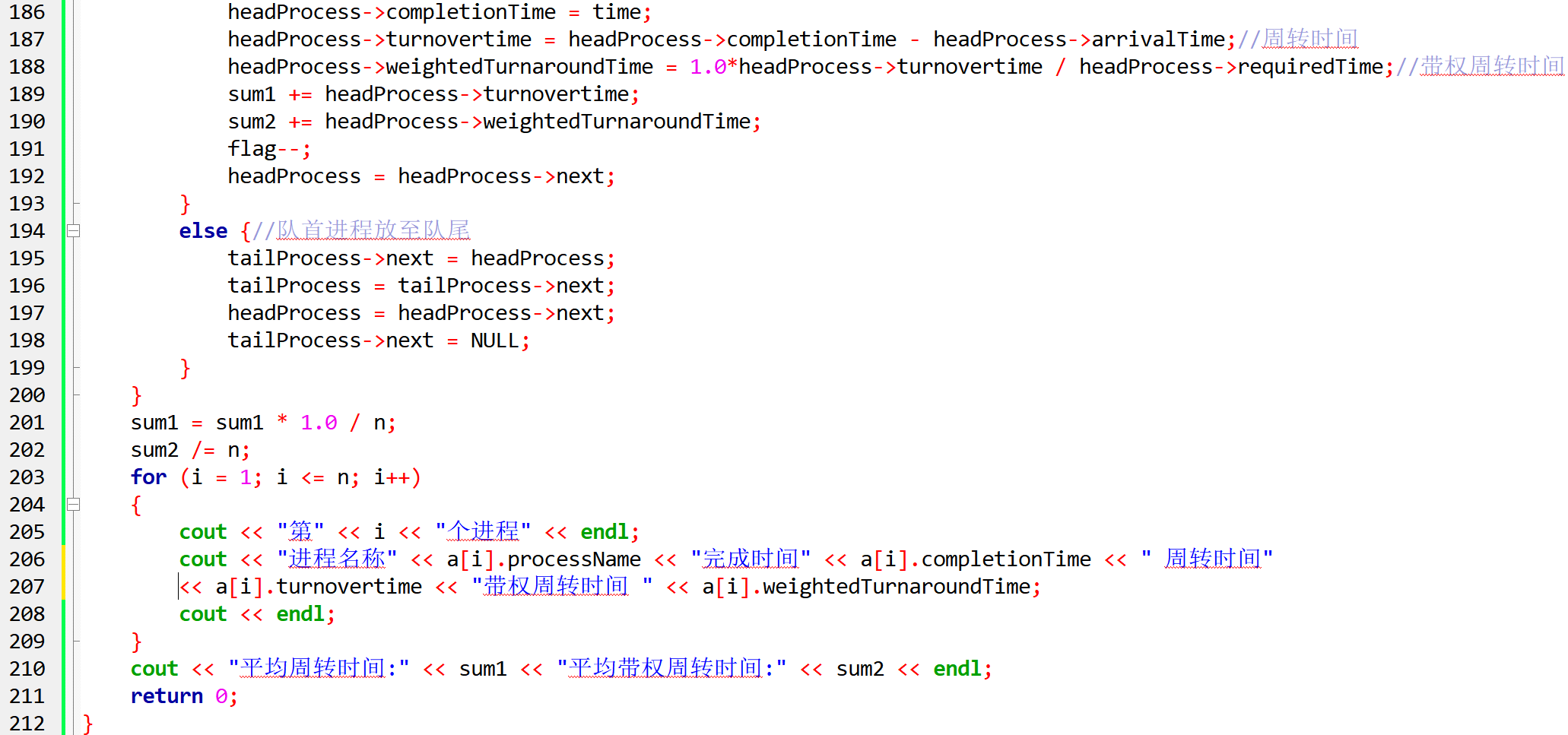
说明：首先将进程的结构体数组进行排序，按照进程的服务时间从小到大进行排序，然后从一开始遍历数组，当有到达时间小于等于当前时间的进程则符合当前最短服务时间的进程，将其完成，完成后将节点访问变量设置为true，即已经被访问过，然后将solved自增一，表示已经解决的进程增加一个，当solved等于n时代表所有进程已经解决完毕，算法结束。

1. RR

代码如下：



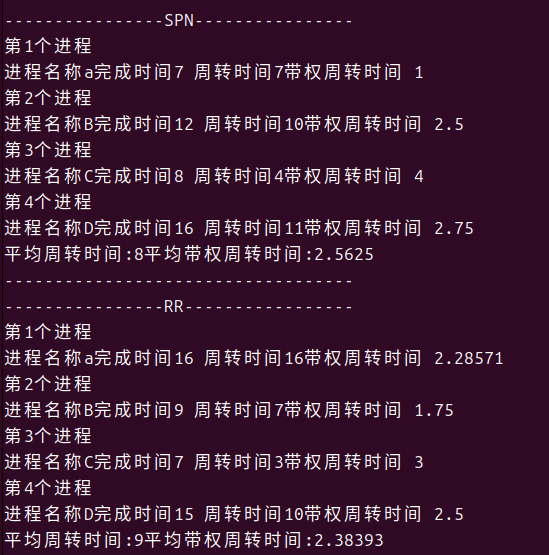
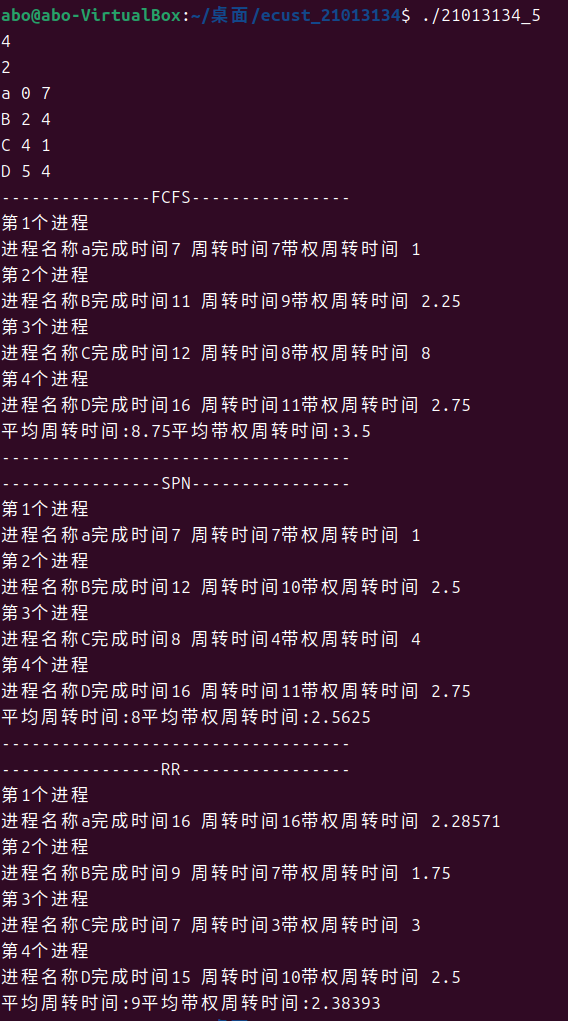




说明：RR代码由于设计到进程的抢夺中断行为且根据时间片算法需要轮转运行进程，所以需要将进程结构体作为队列进行存储，需要注意的是当新进程抵达时先进入队列进行存储，而不是先运行之前没有运行完的进程，对于每个进程需要额外设置其当前剩余服务时间，当服务时间为0的时候将flag自减1代表一个进程已经完成。

1. 计算每种算法下的平均周转时间、平均带权周转时间

计算结果如下所示



说明：上机验收时

提供的测试数据如下：

|  |  |  |
| --- | --- | --- |
| 进程编号 | 到达时间 | 服务时间 |
| 1 | 0 | 7 |
| 2 | 2 | 4 |
| 3 | 4 | 1 |
| 4 | 5 | 4 |

# 四．思考题

1. 在实际的进程调度中，除了按调度算法选择下一个执行的进程外，还应该处理哪些工作？

在实际的进程调度中，除了选择下一个执行的进程外，还需要处理以下工作：

（1）上下文切换：在切换进程执行时，需要保存当前进程的上下文（包括寄存器状态、程序计数器、内存映射等信息），并加载下一个进程的上下文，这个过程被称为上下文切换。上下文切换会带来一定的开销，因此需要尽量减少上下文切换的次数，以提高系统效率。

（2）进程状态管理：进程可能处于不同的状态，例如就绪、运行、阻塞等。调度器需要管理进程的状态转换，例如当进程从就绪状态变为运行状态时，或者当进程需要等待I/O完成而进入阻塞状态时，都需要相应的状态管理。

（3） 中断处理：操作系统需要处理各种中断，例如定时器中断、I/O中断等。当出现中断时，可能需要暂停当前进程的执行，执行相应的中断服务程序，然后根据中断类型决定是否需要进行进程调度。

（4） 资源分配和释放：进程调度器需要考虑系统资源的分配和释放，包括CPU时间、内存空间、I/O设备等资源的合理分配，以保证系统资源的高效利用。

（5） 优先级调度：有些调度算法基于优先级进行调度，即根据进程的优先级来确定执行顺序。因此，调度器可能需要考虑进程的优先级设置和动态调整。

（6）死锁检测和处理：在多道程序设计中，可能存在进程之间的资源竞争问题，导致死锁。调度器可能需要实现死锁检测算法，并采取相应的措施来解除死锁状态。

1. 几种进程调度算法有何区别与联系？

（1） 先来先服务（FCFS）：

工作原理：按照进程到达就绪队列的顺序进行调度，即最早到达的进程首先执行。

特点：简单易实现，没有优先级的概念，避免饥饿（starvation）情况，但可能导致长作业等待时间长的问题（即所谓的“短作业等待时间长”）。

优点：公平，实现简单。

缺点：平均等待时间可能较长，不适合处理长时间运行的任务。

（2） 最短进程优先（SPN）或最短作业优先（SJN）：

工作原理：选择估计运行时间最短的进程优先执行。一旦有进程到达就绪状态，系统会选择预期运行时间最短的进程进行执行。

特点：能够最小化平均等待时间，但需要对进程的运行时间进行准确的预测，且可能因为需要对进程长度的估计而带来一定的开销。

优点：能够减少平均等待时间，适合处理短作业。

缺点：可能需要准确的进程运行时间估计，对于长作业会产生饥饿问题。

（3） 时间片轮转（RR）：

工作原理：将CPU时间分成若干个时间片，每个进程在一个时间片内执行一定量的时间，然后被放回队列尾部等待下一次调度。

特点：确保每个进程都能获得执行，避免了长作业占用CPU时间过长的问题，但可能引入一定的上下文切换开销。

优点：公平，避免长作业占用CPU时间过长。

缺点：可能存在上下文切换开销，对于短作业可能会有一定的响应延迟。

联系点：

这些算法都是用于处理进程调度的方法，旨在最大化系统资源利用率，减少等待时间或提高响应性能，同时，这些算法都有优点和缺点，适用于不同的场景和需求。进程调度算法的选择取决于系统的特定要求，可能需要根据实际情况综合考虑各种算法的优缺点来做出决策。

1. RR 调度算法中，如果将时间定量增长为一个任意大的数目，那么会产生什么影响？

（1）减少上下文切换频率：增加时间片大小会减少上下文切换的频率。因为一个进程在获得CPU时间后可以执行更长的时间，不会在短时间内被强制切换到其他进程，从而减少了调度开销。

（2） 增加响应时间：随着时间片的增加，进程在等待CPU时间时需要等待更长的时间片才能再次获得CPU执行时间，因此对于短期任务而言，响应时间可能会变长。这可能会影响系统对实时性要求较高的应用程序。

（3）不利于交互式任务：对于需要快速响应用户输入的交互式任务来说，较大的时间片可能导致用户感知到的延迟增加，因为这些任务需要等待较长时间才能获得CPU时间片执行。

（4）降低系统吞吐量：大时间片可能导致长时间运行的进程长时间占据CPU，而其他就绪状态的进程需要等待更长时间才能获得CPU时间片。这可能导致系统整体的吞吐量下降。

（5）不利于实时性要求高的任务：如果系统中存在实时性要求高的任务，较大的时间片可能使得这些任务无法按时得到执行，因为时间片太长可能导致实时任务的响应时间过长。

总的来说，增加时间片大小会减少调度开销，但可能会影响系统的响应时间、交互性能以及实时性能，可能不利于系统的整体性能表现，尤其是对于要求高响应性和实时性的场景。因此，在设置时间片大小时，需要权衡不同应用场景的需求，选择适当大小的时间片来平衡系统的性能。