

****

《实用操作系统》实验报告

（七）

**姓　　名 ：宋泽涛**

**学　　号 ：25120222201292**

**学 院 ：信息学院**

**专 业 ：软件工程**

**2024年 11 月**

# 实验目的

* 理解进程调度的过程。
* 掌握各种进程调度算法的实现方法
* 通过实验比较各种进程调度算法的优劣。

# 实验环境

* PC + Linux Red Hat操作系统 + GCC
* 或 Windows xp + VC
* 或 任意OS + Java

# 实验内容

随机给出一个进程调度实例，如：

进程 到达时间 服务时间

A 0 3

B 2 6

C 4 4

D 6 5

E 8 2

模拟进程调度，给出按照算法先来先服务FCFS、轮转RR（q=1）、最短进程优先SPN、最短剩余时间SRT、最高响应比优先HRRN进行调度各进程的完成时间、周转时间、响应比的值。

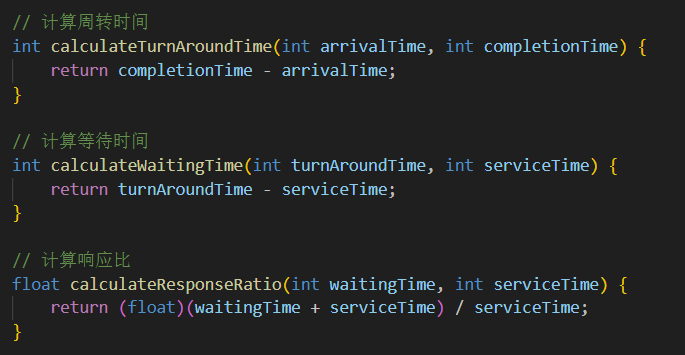
# 实验具体实现

本实验实现了五种常见的进程调度算法，具体原理如下：

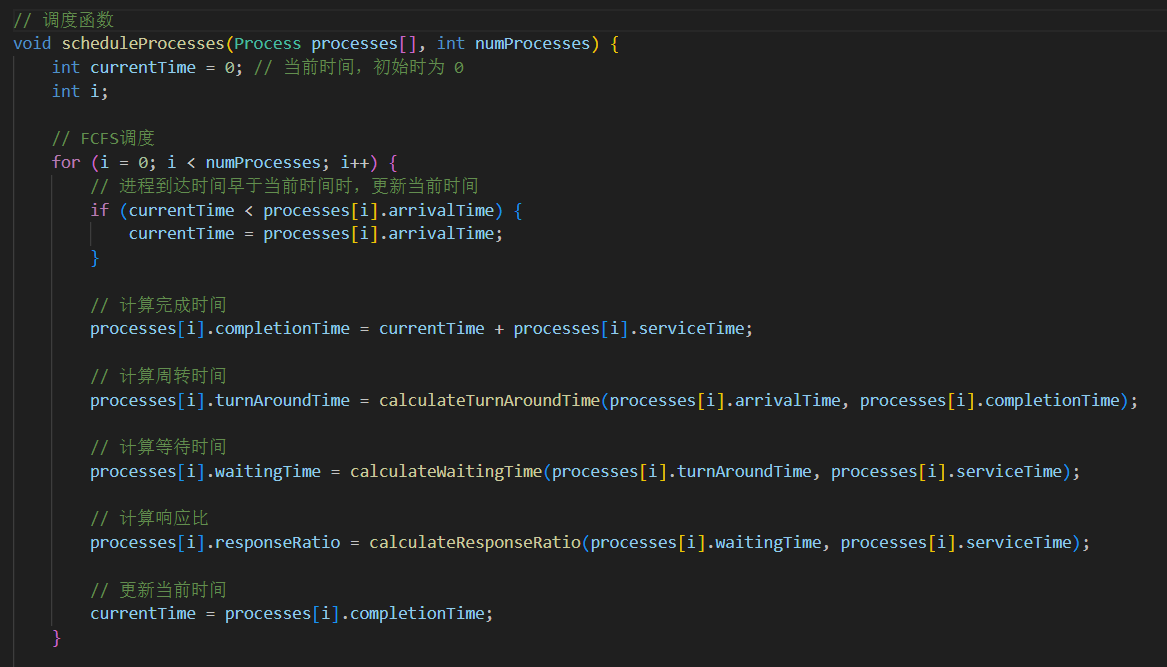
1. 先来先服务（FCFS）算法
   1. **原理**：按进程的到达顺序进行调度，进程执行时不抢占，一直执行直到完成
   2. **优点**：实现简单，容易理解。
   3. **缺点**：可能导致长进程阻塞短进程，造成**等待时间过长**，进而影响系统效率。
2. 轮转调度（RR）算法
   1. **原理**：为每个进程分配一个固定的时间片（本实验设为 1），若进程在时间片内未执行完，则轮流进行调度。
   2. **优点**：公平性较好，确保每个进程都得到CPU时间。
   3. **缺点**：时间片的选择直接影响性能。时间片过小会导致系统开销大，时间片过大会变成**FCFS**。
3. 最短进程优先（SPN）算法
   1. **原理**：根据进程的**服务时间**调度，选择服务时间最短的进程执行。
   2. **优点**：能减少**平均等待时间**，提高效率。
   3. **缺点**：容易造成**饥饿现象**，长进程可能长时间得不到执行。
4. 最短剩余时间（SRT）算法
   1. **原理**：是SPN的抢占式版本，当有新进程到达且其剩余时间比当前执行进程的剩余时间短时，当前进程被抢占。
   2. **优点**：能够有效减少**平均等待时间**，提高系统响应。
   3. **缺点**：可能导致**频繁的上下文切换**，增加系统开销。
5. 最高响应比优先（HRRN）算法
   1. **原理**：根据进程的**响应比**进行调度，响应比越高的进程优先执行，响应比由进程的等待时间和服务时间决定。
   2. **优点**：避免了**饥饿现象**，比SPN更公平。
   3. **缺点**：需要动态计算响应比，算法实现较为复杂。

这里使用C语言分别模拟五种调度算法，每组算法给出五组测试数据。

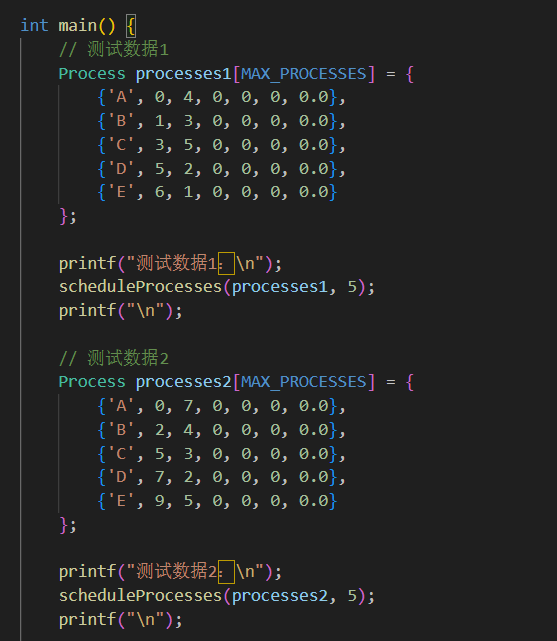
1. 先来先服务（FCFS）调度算法实现：按照进程到达时间顺序进行调度，确保每个进程按照到达的顺序执行，计算每个进程的**完成时间**、**周转时间**和**等待时间**。

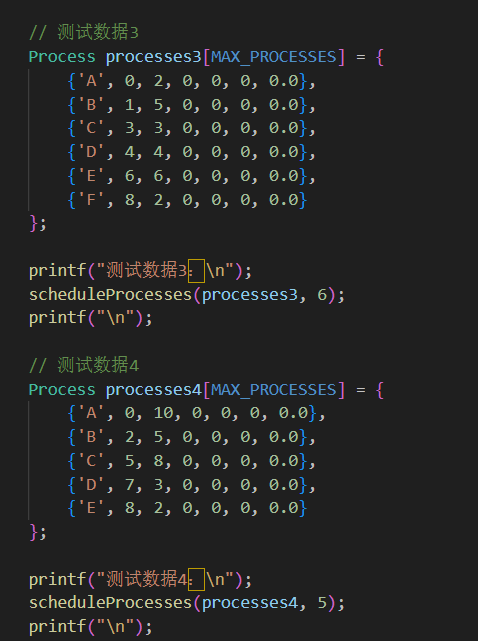


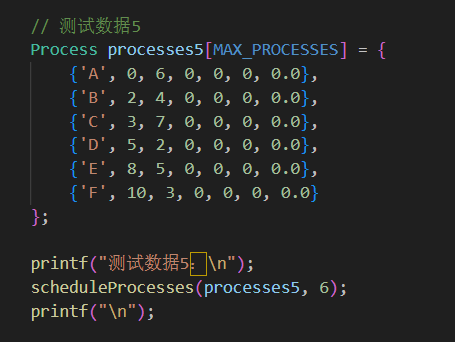
具体调度过程：



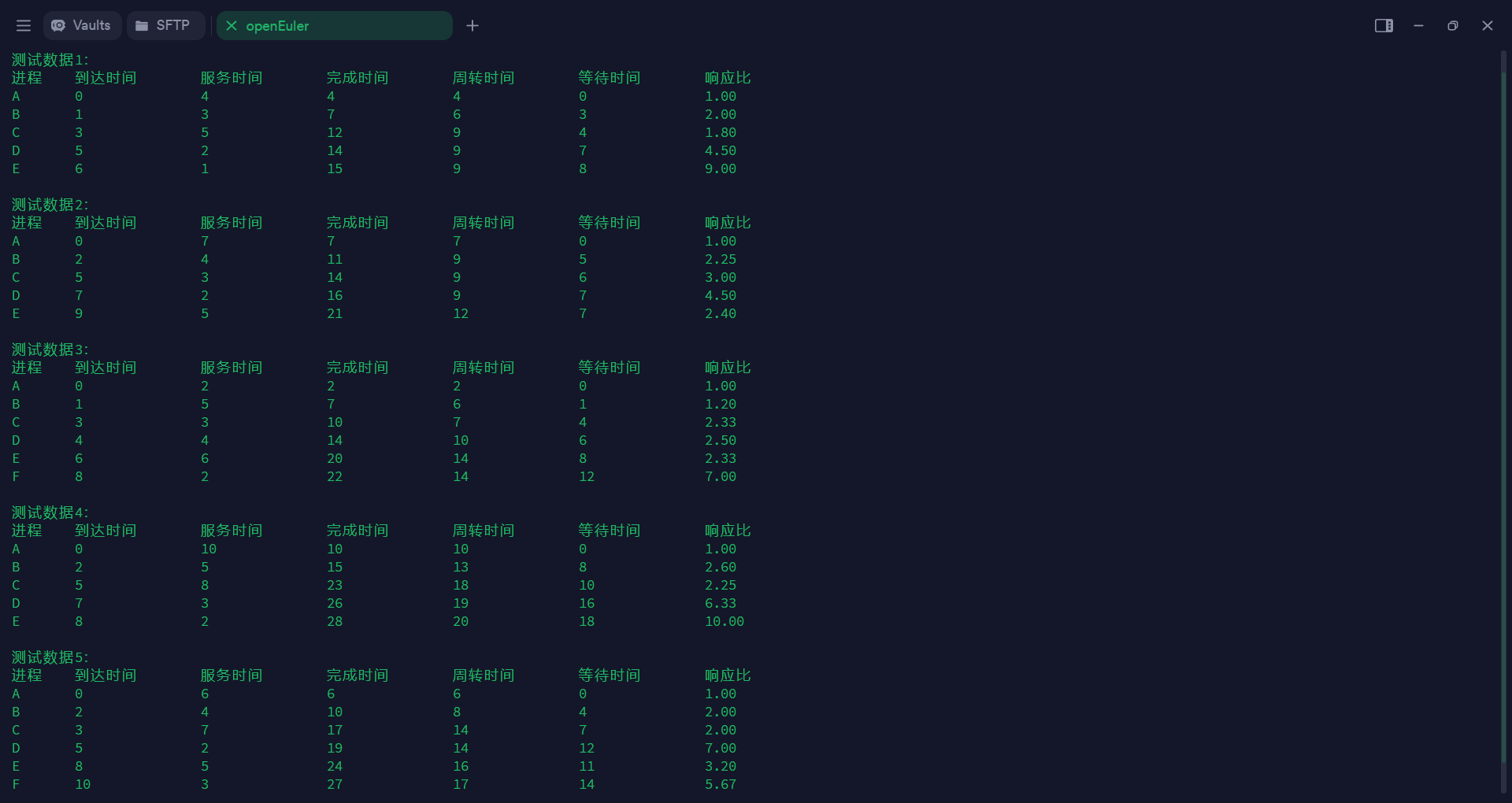
五组测试数据如下：



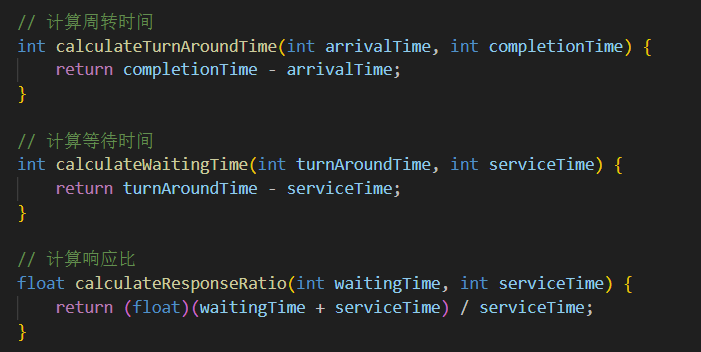




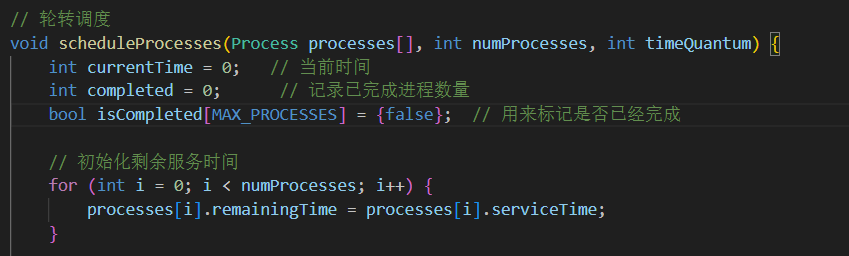
在openEuler上运行结果如下：



1. 轮转调度（RR）调度算法实现：按照固定时间片为每个进程分配 CPU 时间。如果进程执行完，则计算其**完成时间**、**周转时间** 和**等待时间**。

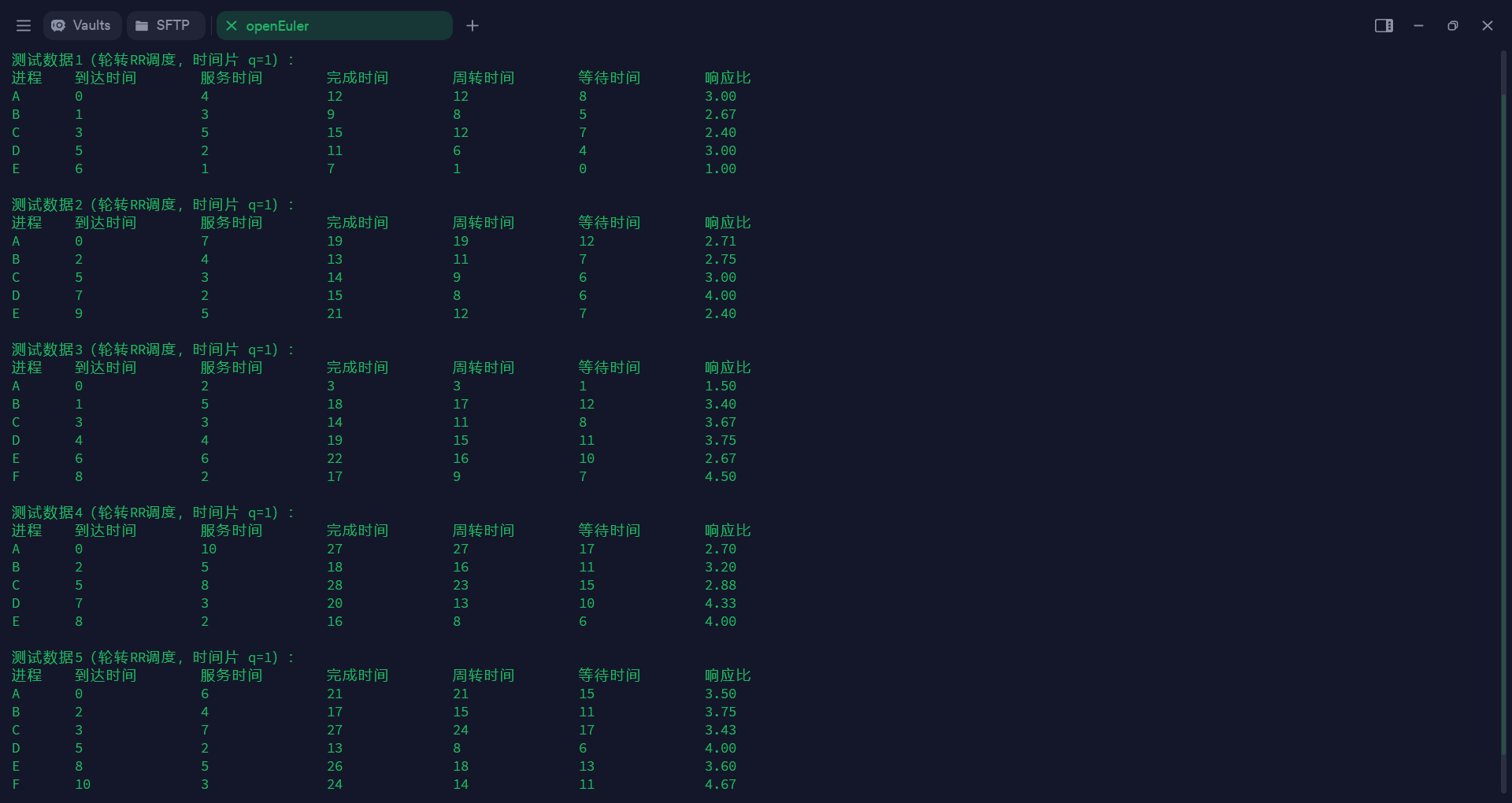


轮转调度：

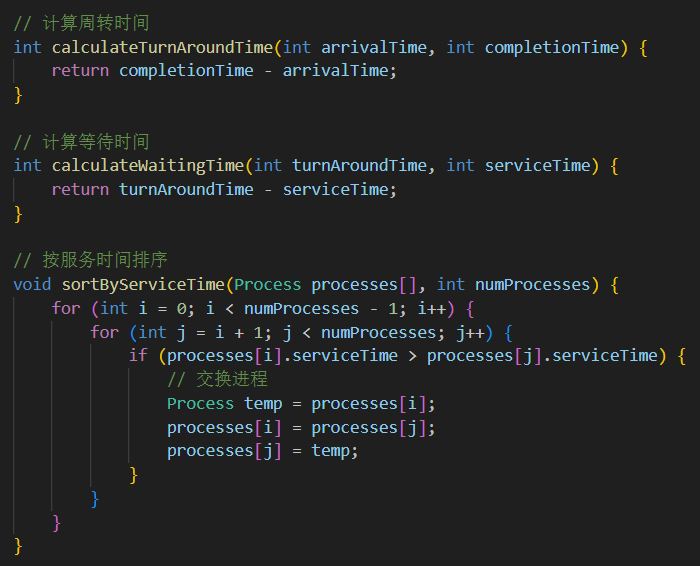




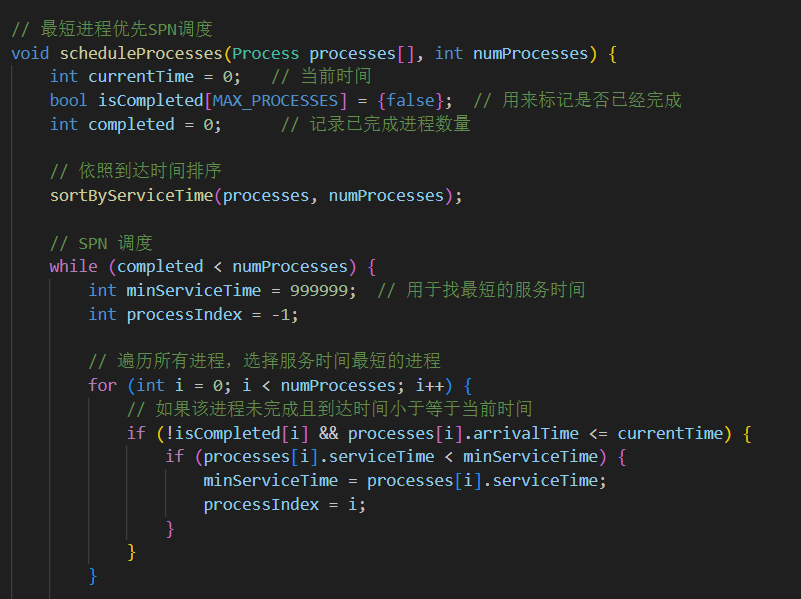
测试数据与上一个调度算法相同，下面是openEuler上运行的结果：

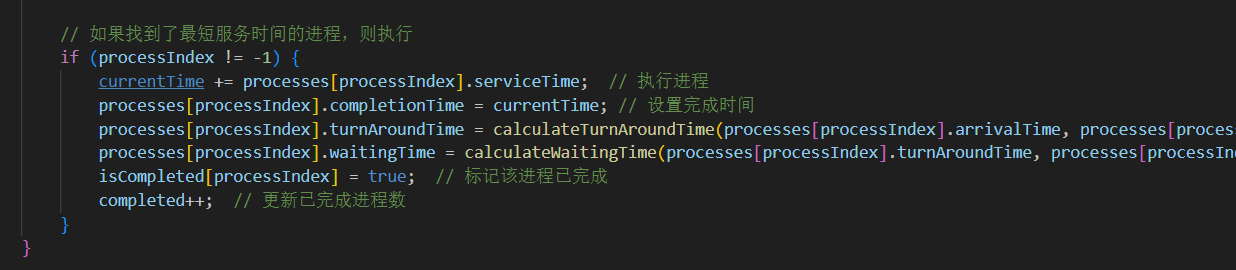
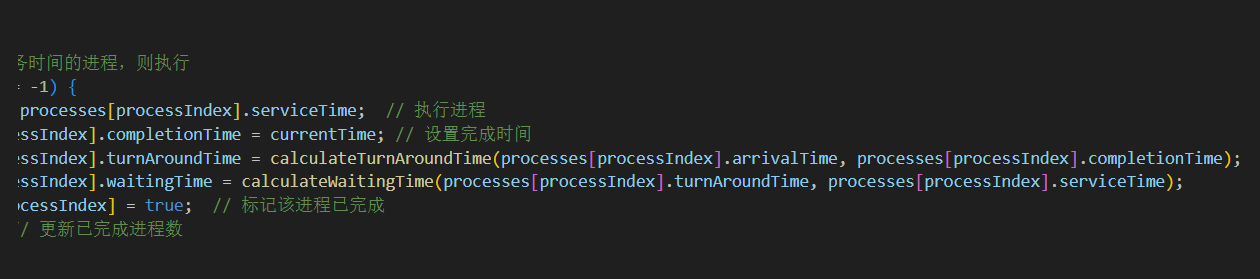


1. 最短进程优先（SPN）调度算法实现：对所有进程按**服务时间**排序，选择服务时间最短的进程执行，计算每个进程的**完成时间**、**周转时间** 和 **等待时间**。

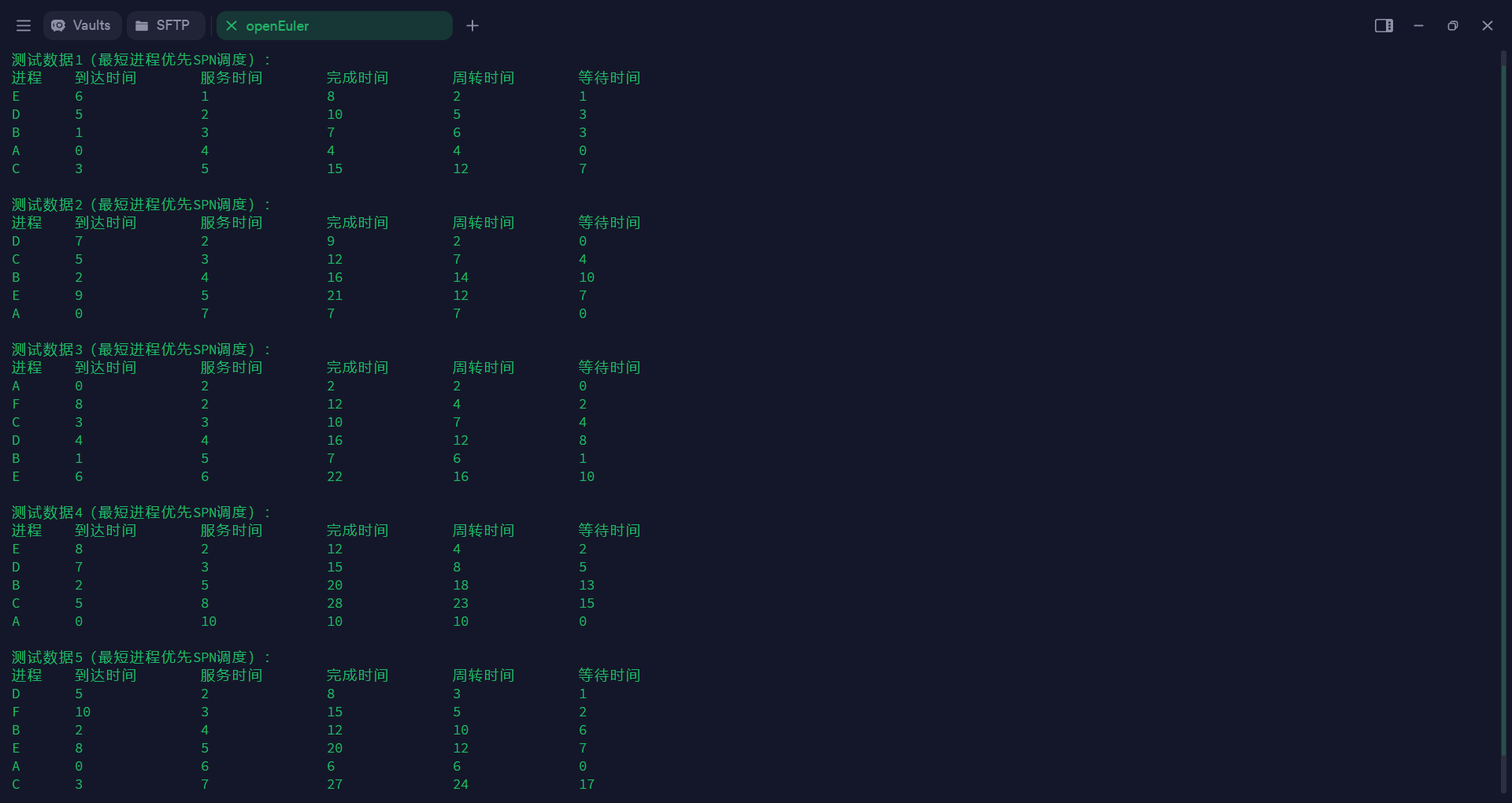


最短进程优先SPN调度

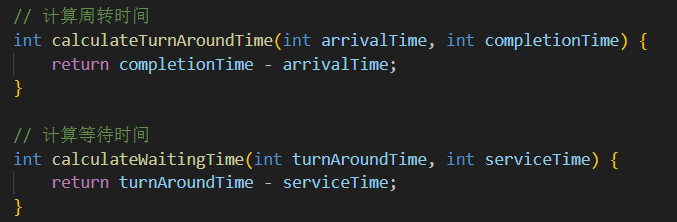


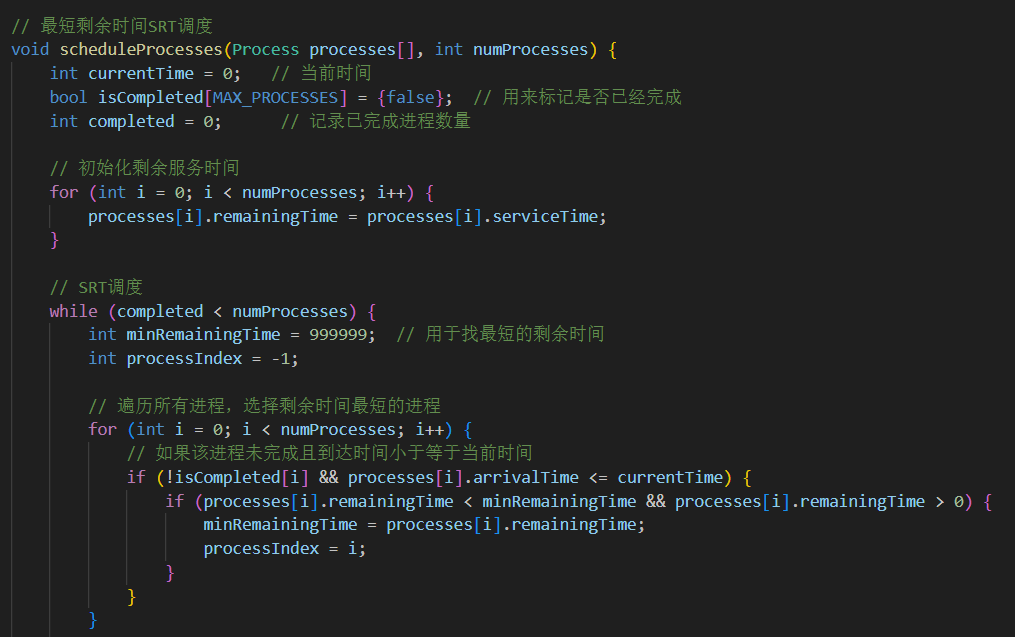
测试数据与上一个调度算法相同，下面是openEuler上运行的结果：

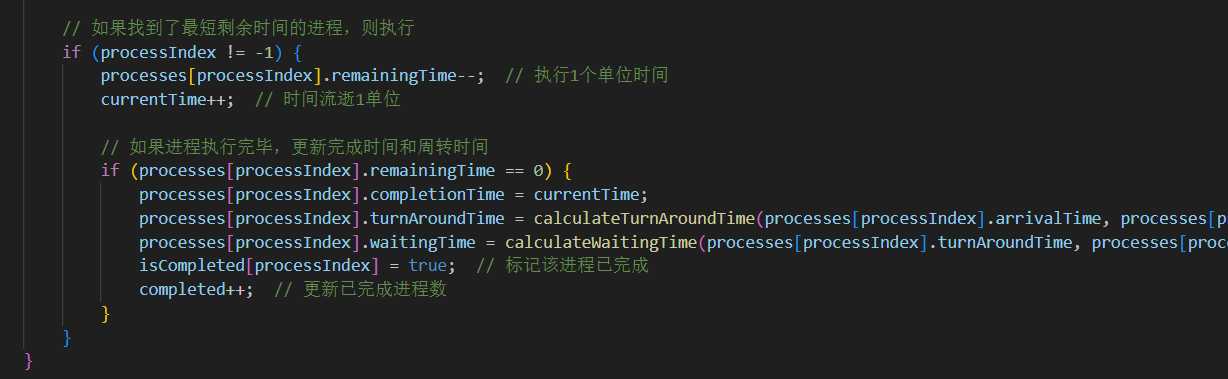


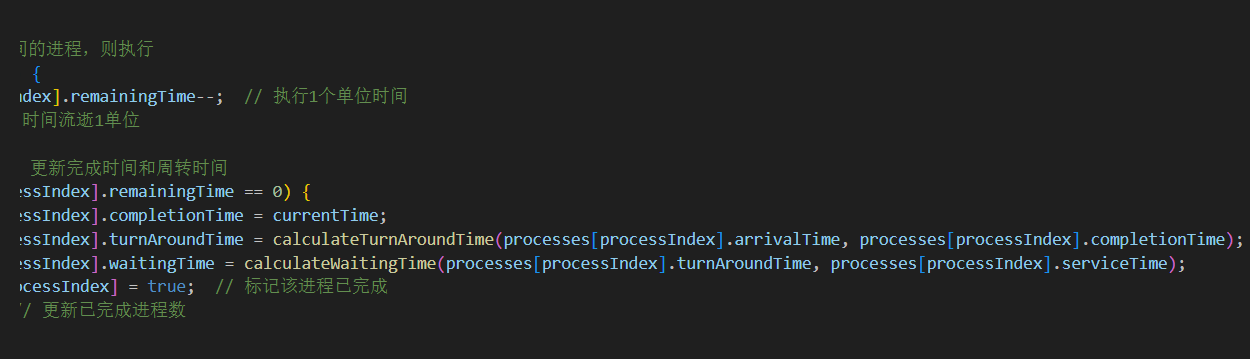
1. 最短剩余时间（SRT）调度算法实现：基于**剩余服务时间**进行调度，选择剩余时间最短的进程进行执行。每次进程执行完毕后，更新其**剩余时间**，直到所有进程执行完成。



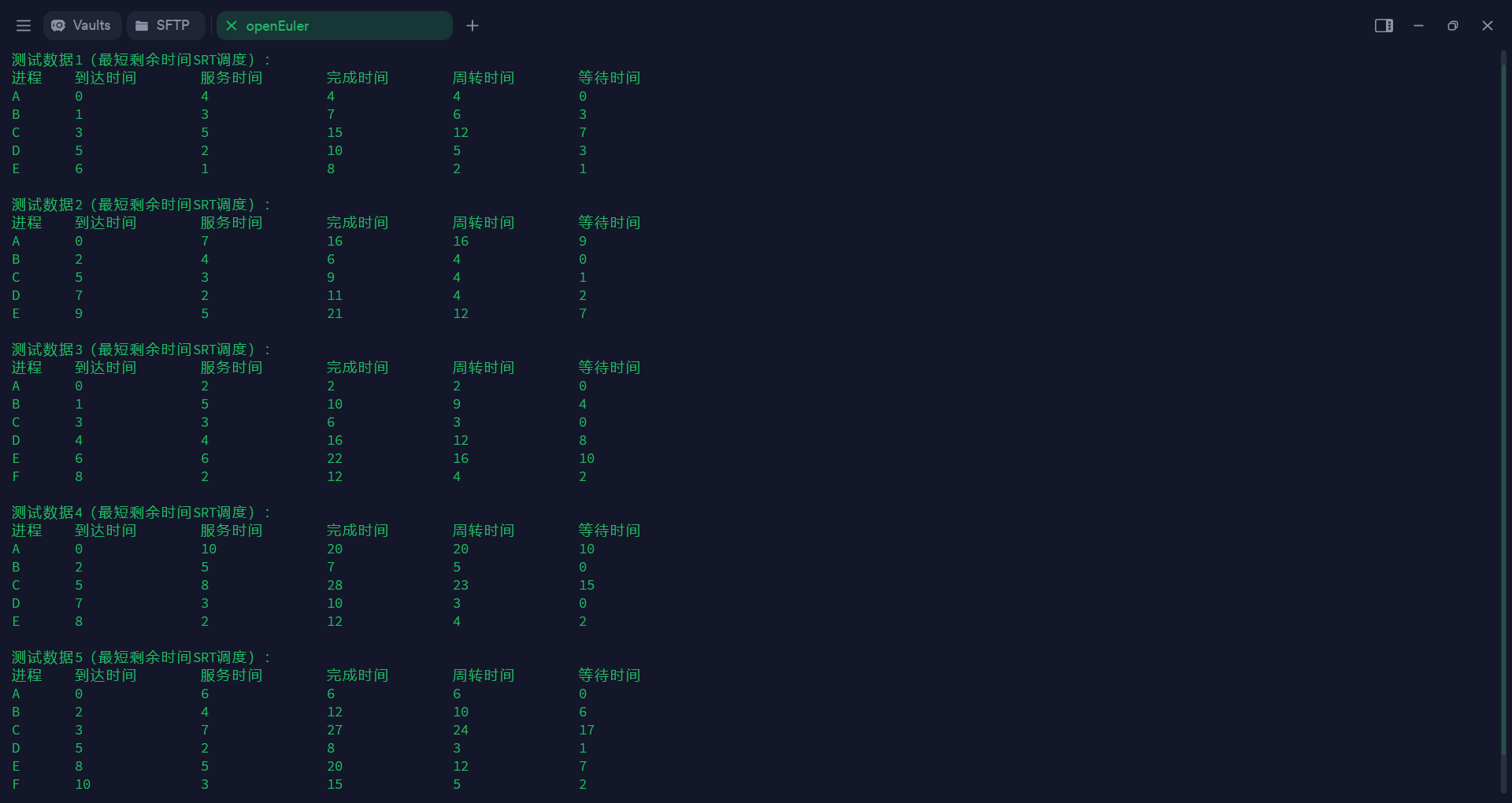
最短剩余时间SRT调度



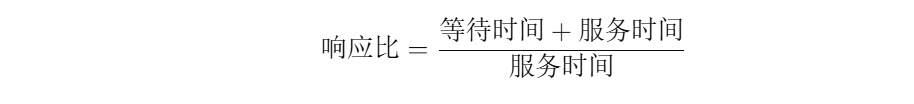


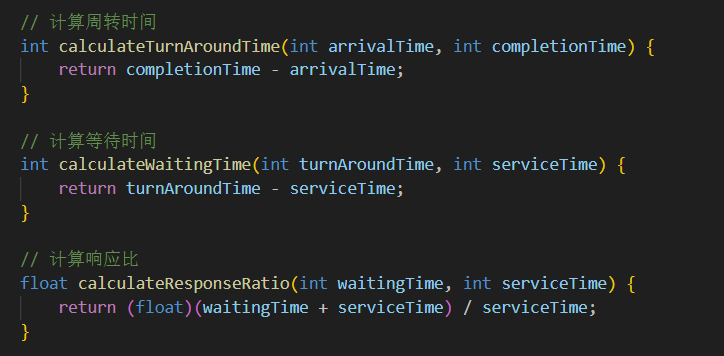


测试数据与上一个调度算法相同，下面是openEuler上运行的结果：

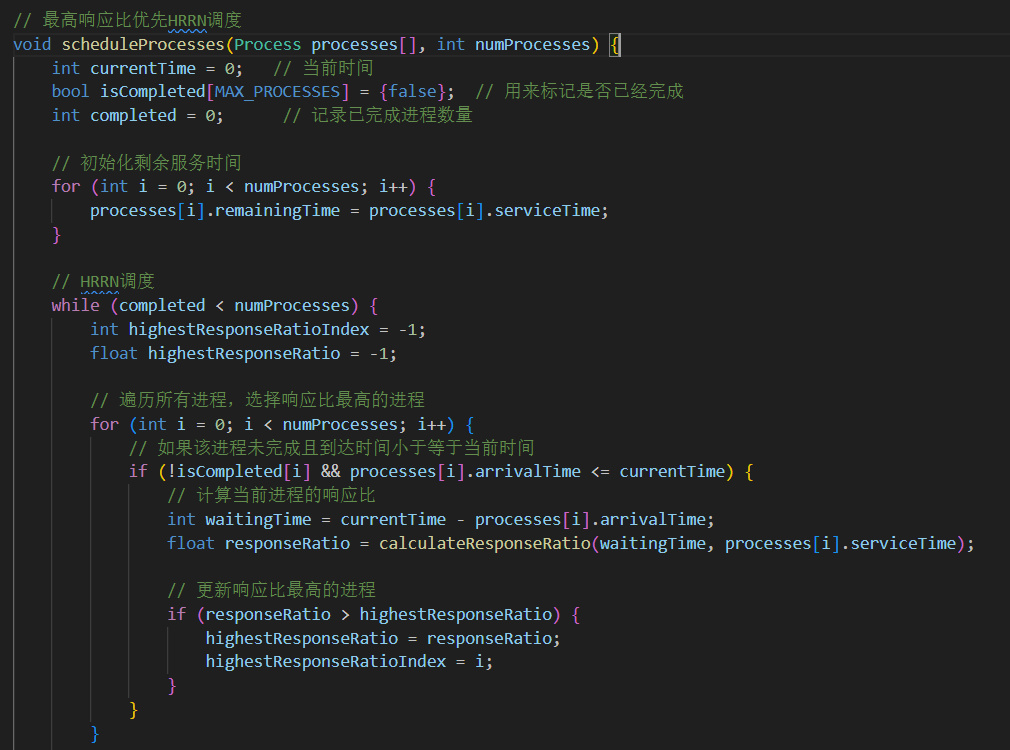


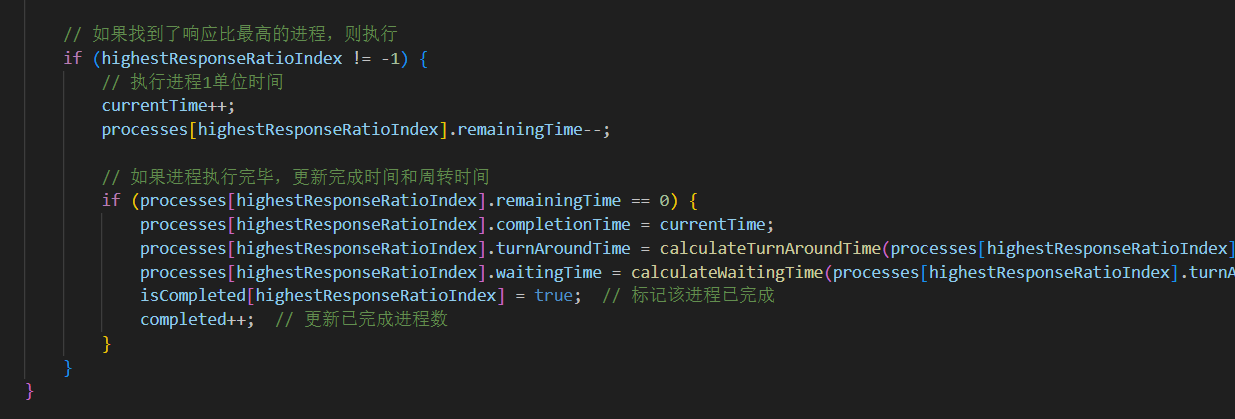
1. 最高响应比优先（HRRN）调度算法实现：计算每个进程的 **响应比**，并选择响应比最高的进程进行调度。响应比计算公式为：

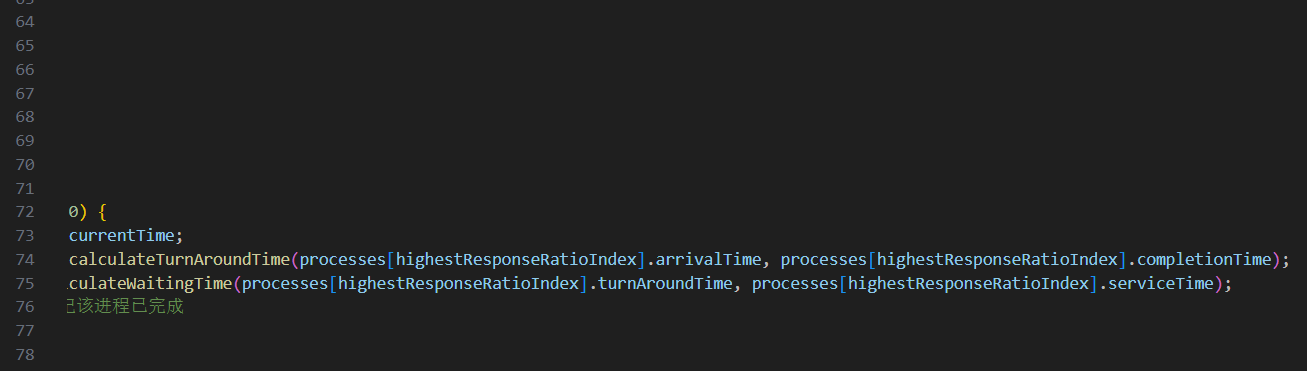




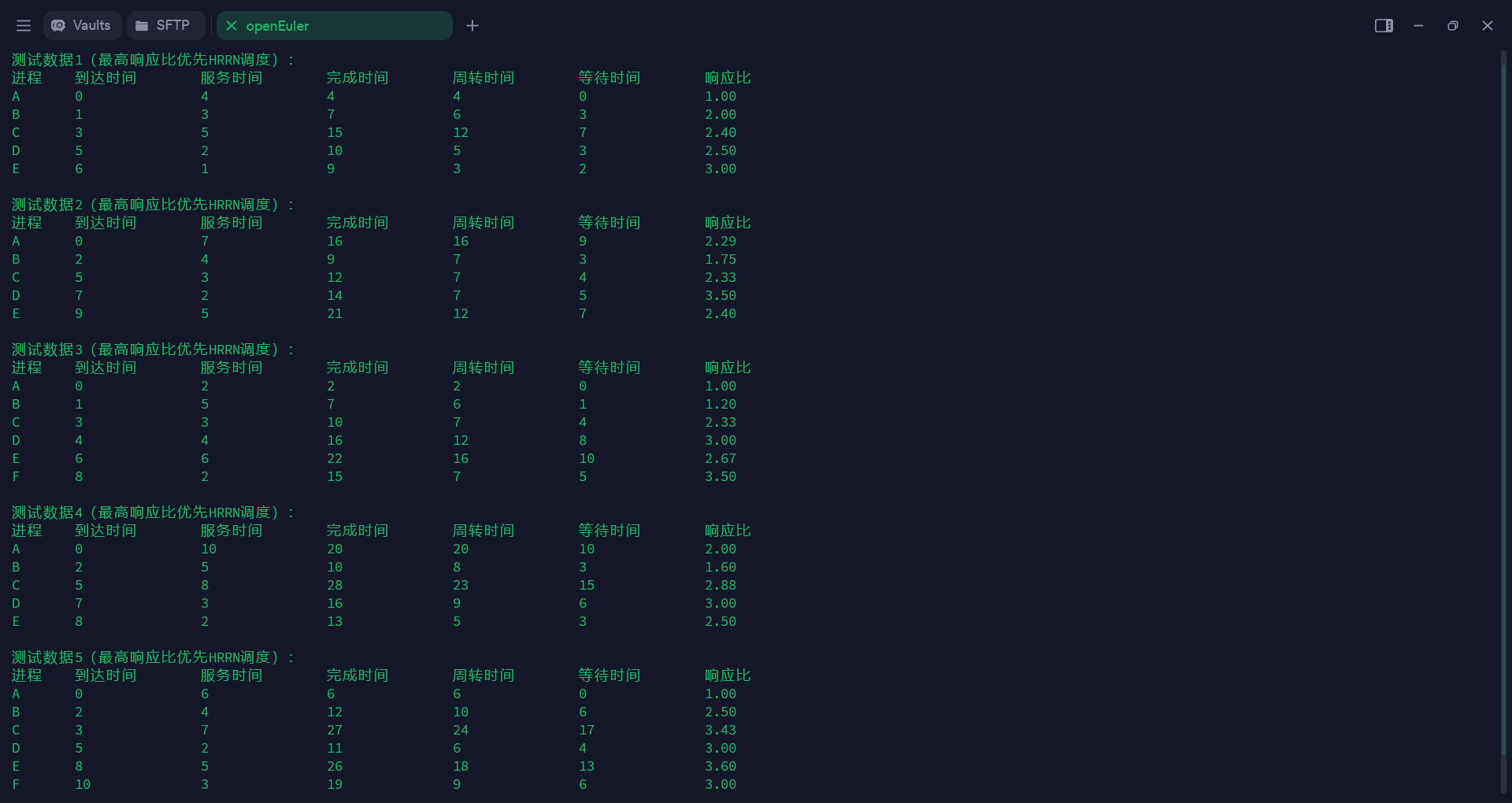
最高响应比优先HRRN调度







测试数据与上一个调度算法相同，下面是openEuler上运行的结果：



# 实验分析与总结

1. 先来先服务算法
   1. 优点：
      1. **简单易实现**：FCFS 算法按进程的到达时间顺序调度，算法逻辑直观，易于实现。
      2. **公平性**：每个进程按其到达顺序依次执行，没有优先级的问题，确保了调度的公平性。
   2. 缺点：
      1. **平均等待时间长**：从实验结果来看，对于长时间服务的进程，FCFS 可能导致短进程需要等待较长时间，进而导致系统整体的 **平均等待时间较长**。
      2. **容易产生饥饿现象**：如果有某些进程的服务时间很长，它们可能会拖慢其他进程的执行，尤其是当有短进程等待时。
      3. **没有考虑进程的优先级或紧急程度**：因此，可能导致 **低响应性**，无法在短时间内响应高优先级的进程。

在测试数据中，FCFS 算法对于短进程的等待时间较长（如进程 **A** 和 **B**），而长进程（如 **C**）会拖慢后续进程的调度，导致 **周转时间** 和 **等待时间** 增长，效率较低。

1. 轮转调度（RR）算法：
   1. 优点：
      1. **公平性高**：RR 算法保证每个进程得到 **相同的 CPU 时间**，避免了长进程阻塞短进程的情况。
      2. **适用于响应时间要求较高的系统**：尤其在多个交互性进程的调度中，轮转调度能够保证各进程有足够的响应时间。
   2. 缺点：
      1. **时间片过小导致频繁上下文切换**：如果时间片设得过小，会导致进程频繁被抢占，从而增加 **上下文切换** 的开销。
      2. **时间片过大可能退化为 FCFS**：如果时间片过大，轮转调度算法可能会退化为 **FCFS**，失去时间片的优势。

从实验结果来看，**RR** 算法的表现相对均衡，各进程的 **等待时间** 和 **周转时间** 相对接近，但由于时间片为 1，其 **平均等待时间** 和 **周转时间** 都比其他非抢占式算法略高，尤其在 **测试数据 1** 和 **测试数据 2** 中，较短进程的等待时间受到了增加。

1. 最短进程优先 (SPN) 算法：
   1. 优点：
      1. **最小化平均等待时间**：SPN 优先调度 **服务时间最短的进程**，能有效减少 **平均等待时间** 和 **平均周转时间**。
      2. **提高系统效率**：通过选择最短服务时间的进程，能够较为高效地利用 CPU，减少长时间等待的现象。
   2. 缺点：
      1. **可能导致饥饿现象**：对于服务时间较长的进程，可能会一直被后续短进程抢占，从而导致长进程的 **饥饿问题**。
      2. **无法处理动态变化的进程**：SPN 需要知道进程的 **服务时间**，对于 **动态进程**（如实时任务）不适用。

在测试数据中，SPN 算法明显在 **平均等待时间** 和 **周转时间** 上表现较好，尤其是在 **测试数据 1** 和 **测试数据 2** 中，短进程（如进程 **E**）能够得到较优先执行。然而，由于 **饥饿现象** 的存在，长进程的等待时间明显增大，进程 **C** 在某些测试数据中的等待时间过长。

1. 最短剩余时间 (SRT) 算法
   1. 优点：
      1. **抢占式调度，响应更快**：SRT 根据 **剩余服务时间** 进行调度，能够较为有效地减少长进程的 **平均等待时间**。
      2. **减少系统平均等待时间**：尤其是在 **短进程优先** 的情况下，SRT 能够最大限度地提高短进程的执行效率。
   2. 缺点：
      1. **频繁的上下文切换**：SRT 是 **抢占式** 的，每次调度都可能导致进程被抢占，增加了上下文切换的开销，特别是在系统负载较高时。
      2. **对系统资源要求较高**：由于要频繁计算剩余时间，可能会增加额外的 **计算开销**，影响系统的整体效率。

在实验结果中，SRT 算法能够有效减少 **平均等待时间**，尤其是在进程 **E** 执行时，能明显减少等待时间。然而，频繁的 **上下文切换** 导致了某些进程的 **响应时间** 增加，尤其是在较为复杂的进程调度场景中，如 **测试数据 5**，当进程数较多时，SRT 显得有些低效。

1. 最高响应比优先 (HRRN) 算法
   1. 优点：
      1. **避免饥饿现象**：HRRN 通过 **响应比** 动态调整优先级，能有效避免进程因长时间等待而被永远抢占的情况。
      2. **综合考虑等待时间和服务时间**：HRRN 算法能够较好地平衡进程的 **等待时间** 和 **服务时间**，实现 **公平性** 和 **效率** 的平衡。
   2. 缺点：
      1. **计算开销较大**：每次调度时都需要计算每个进程的 **响应比**，这增加了算法的计算复杂度。
      2. **适用于负载较低的系统**：HRRN 在负载较高时可能无法发挥其优势，响应比的动态调整增加了 **系统负担**。

在实验数据中，HRRN 算法的表现优于 **SPN** 和 **SRT**，尤其是在 **测试数据 2** 和 **测试数据 3** 中，它能够有效避免 **饥饿现象**，同时减少了 **平均等待时间**。HRRN 算法能够综合考虑进程的 **等待时间** 和 **服务时间**，尤其在 **进程混合负载** 时表现优越。尽管如此，HRRN 的 **计算复杂度** 在处理大规模进程时可能成为瓶颈。