期末报告 基于 ArceOS 的调度算法综合测试及改进

秦若愚 计 05 班 2019011115

一、研究主题

ArceOS 是一个由 Rust 编写的实验性的模块化操作系统（也即 Unikernel），其特点是模块化、精简化，同时具有较高的性能和安全性。在所有的操作系统中，任务调度管理都是最为重要的一部分，在 ArceOS 中也是如此。ArceOS 采用模块化设计，可以根据应用需要自主选择多种调度算法，如先入先出算法（FIFO）、时间片轮转法（RR）以及完全公平算法（CFS）等。

作为一个 Unikernel 系统，虽然 ArceOS 可以适配现有的操作系统调度算法，但是这些算法没有充分利用 ArceOS 的特性（如单应用、组件化）以达到最优的调度性能。如果将 ArceOS 视为一个非交互式多次执行相同应用的操作系统，那么可以针对其特点做改进，以提升调度时间（任务调度的额外开销）、平均周转时间（平均每个任务从提交到完成的时间）等调度算法指标。

因此，在本操作系统专题训练课程中，我的研究题目是**基于 ArceOS 的调度算法综合测试及改进**。首先，我在 ArceOS 已有的三种调度算法上新实现两种传统调度算法；其次，设计全面的测例以评估调度算法的性能；最后，提出基于程序行为的调度算法（PBG, Program Behavior Guided），通过实验证明 PBG 算法相较于已有算法在多个指标上均有提升。

二、研究方法

1. 调度算法补充

ArceOS 原有的三种调度算法分别为先入先出算法（FIFO）、时间片轮转法（RR）以及完全公平算法（CFS）。为了提升 ArceOS 调度算法应用广泛性和灵活性，我参考现有的调度算法设计，分别添加了最短作业优先算法（SJF）和多级反馈队列算法（MLFQ）。

对于最短作业优先算法，其原理是优先调度预估执行时间最短的进程，优点是平均周转时间最小。我的实现方式是使用 Rust 中的数据结构 BTreeMap 维护带有期望运行时间的优先队列，根据移动平均计算更新期望运行时间。

对于多级反馈队列算法，其原理是根据进程的行为和估计的执行时间将其放入不同的队列中，通常在前面的队列中分配更多的CPU时间；如果进程在一个队列中消耗了其全部时间片，则被移至下一个队列，优点是既能优先处理短进程，又能保证长进程得到执行机会。我的实现方式是设置 8 级优先级队列，第 0 级队列分到 1 个时间片，其余队列时间片数指数增长，每过 100000 个时间片就重置队列。

此外，为了提升任务调度性能，我还将原有的单队列调度算法扩展为多队列调度算法。

2. 调度算法测例

为了对已有调度算法进行综合全面的评测，同时找出现有的不足以便做出针对性改进，我参考往届同学的工作，实现了五个调度算法测试用例。这五个测试用例的设计及用途如下：

1. equal 测例。建立 4 个进程，每个执行 100 组用时相同的任务，每组任务结束后主动 yield 一次。该测例用于初步测试调度算法的公平性。
2. unequal 测例。建立 4 个进程，每个执行饱和数量的用时和线程号成正比的任务，每组任务结束后主动 yield 一次，测试时长为 5 秒。该测例用于测试调度算法的公平性。

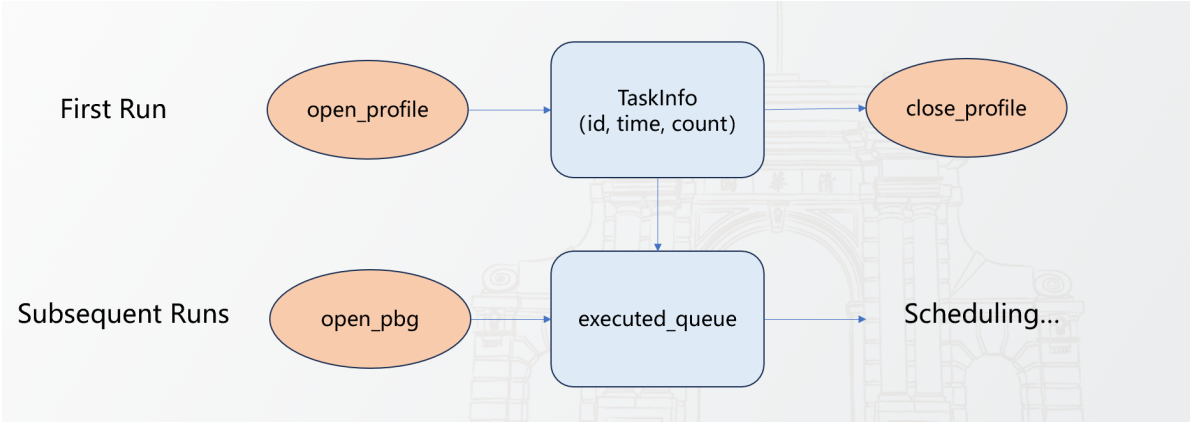
3. yield 测例。建立 4 个进程，每个执行 100000 组用时固定的任务（100次循环），每组任务结束后主动 yield 一次，用于测试调度算法的基础速度。
4. realtime 测例。建立 4 个短任务，yield 次数分别是 50000, 100000, 200000, 500000，循环长度分别是 10, 5, 2, 1。同时建立一个长任务，yield 次数 8，循环长度 100000000。该测例用于测试调度算法的实时性。
5. occupy 测例。建立 2 个短进程，运行饱和次；建立 1 个长进程，运行 80 次，每次循环长度为 10000000 。通过测试长进程的完成时间，该测例反映调度算法的周转时间。

3. PBG 调度算法

在调度算法测例的测试中反映出 ArceOS 中现有的调度算法存在的两个问题： 一是由于维护了调度任务队列以及使用了较复杂的数据结构，导致任务调度的额外开销较大；二是由于不能真正预知每个任务的执行时间，所以不能实现真正的最短作业优先调度。为解决这两个问题，我设计了基于程序行为的调度算法 PBGScheduler，下面介绍其实现原理及优点。

PBGScheduler 的应用场景是操作系统将多次执行相同应用且不进行交互。在第一次运行时，PBGScheduler 内部会对每次调度过程进行计时和记录，记录每个任务的预期执行时间、执行次数等。在之后的执行过程中，PBGScheduler 可以直接利用之前记录的结果，将任务调度顺序事先安排好，执行最短作业优先调度。

在具体实现方面，我新增了三个系统调用，分别用于开启程序测量、结束程序测量以及开启 PBG 调度算法。在 PBGScheduler 内部我维护了待运行的任务的信息（包括 id、执行时间和执行次数），在调度时按照最短作业优先进行调度。具体流程可见下图：



这样做的优点是无需为了执行最短作业调度而估计任务执行时间，而是可以实现真正的最短作业调度。此外，由于实现安排好了大部分任务的调度顺序（idle、gc_task 等任务除外），在每次任务调度时几乎不需任何操作，从而大大降低了任务调度的额外开销。

PBGScheduler 还具有极高的扩展性，未来可以实现用户通过定制化记录指定任务的优先级、最晚容忍时间等，以至于自定义任务调度顺序。

三、实验

在本节中，我将报告针对 PBG 调度算法的实验及分析。

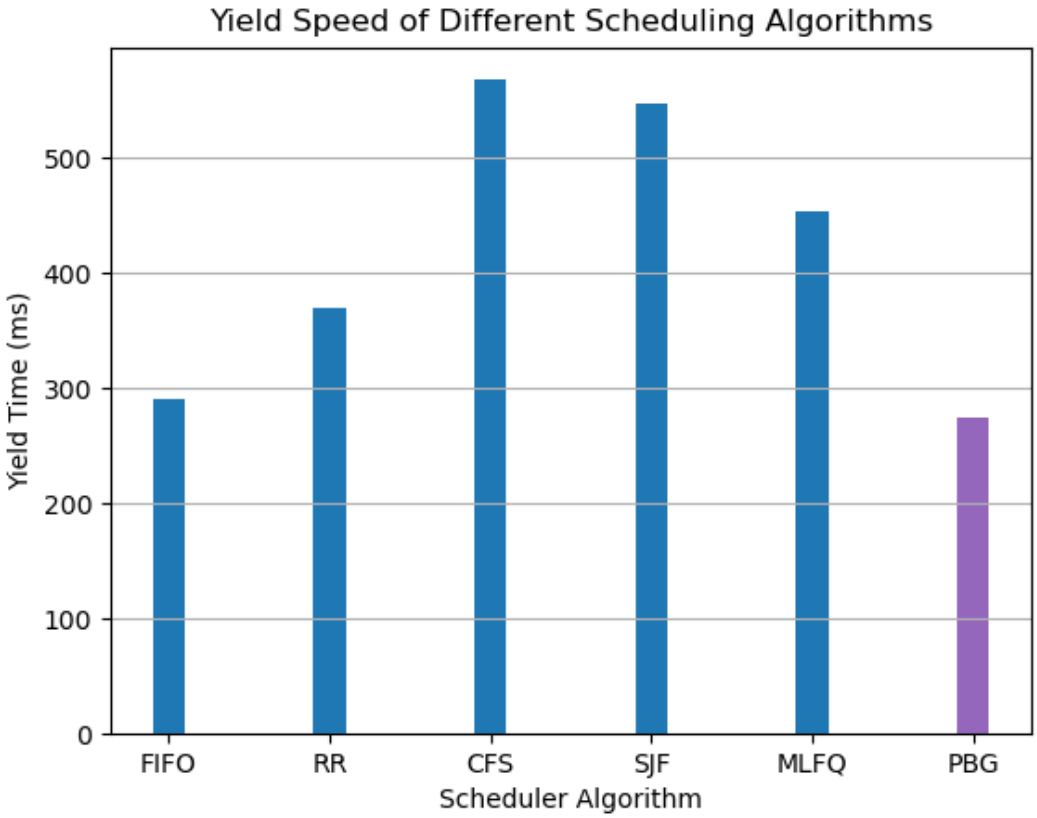
1. PBG 调度算法实验结果

为了测试 PBG 调度算法的性能，我设计了调度开销和平均周转时间两个实验。

1. 调度开销实验

该实验的实验设置为建立 4 个进程，每个执行 400000 组用时固定的任务，每组任务结束后主动 yield 一次。该实验用于测试调度算法的基础速度。

实验结果如下图所示：

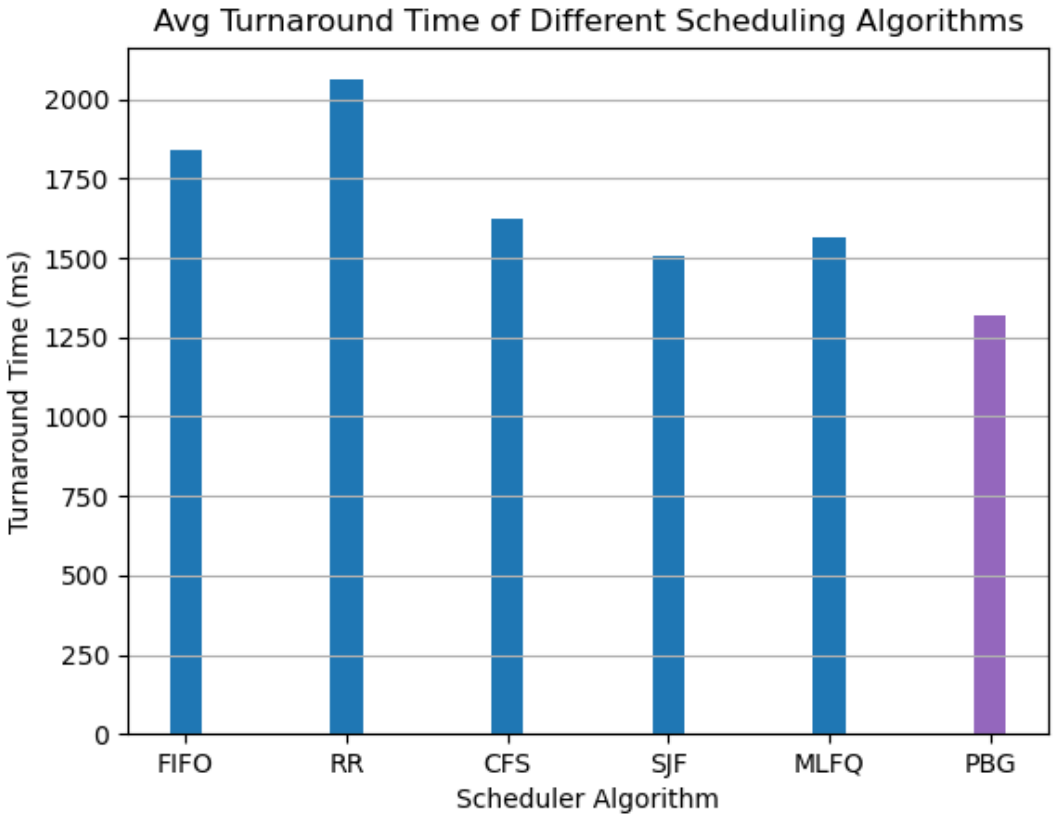


可以看到，由于 PBG 调度算法在任务调度前就计算好了调度顺序，无需在调度时做任何复杂的操作，因此其调度开销大幅减少，甚至低于最简单的 FIFO 算法。

2. 平均周转时间实验

该实验的实验设置为建立 4 个短任务，yield 次数分别是 50000, 100000, 200000, 500000，循环长度分别是 10, 5, 2, 1，同时建立一个长任务，yield 次数 8，循环长度 100000000。该实验用于用于测试调度算法的平均周转时间。

实验结果如下图所示：



可以看到，由于 PBG 调度算法实现了真正意义上的最短作业优先，且在调度开销上有所优化，所以其平均周转时间最短（同时注意到 SJF 算法实现了第二短的平均周转时间，说明该任务适合最短作业优先的调度设计）。

四、总结

在本课程中，我对 ArceOS 系统的调度算法进行了全面且深入的研究。我一方面完成了一些基础工作，包括新增经典调度算法、利用测例评估调度算法性能等；另一方面进行了针对性的创新，即提出在 ArceOS 上的基于程序行为的调度算法（PBG, Program Behavior Guided），该算法在调度速度和平均周转时间上均超过已有算法。这次训练让我受益匪浅，感谢老师的指导！