Assignment 4 - Scheduling

2025 年 5 月 7 日

1 Revision History

- 2025-05-02：第一次发布。
- 2025-05-07：修改了 `schedtest` 内容，它现在使用 `ticks` 作为时间参考，并且测试条件更加严格。

  修改了 `Makefile`，现在它包含了 `handin` 目标。

  以下文件被修改了：

  - *user/src/schedtest.c*
  - *os/ktest/ktest_syscall.c*, *os/ktest/ktest.h*
  - *Makefile*

2 Background

在目前的 xv6 上，我们使用的是 Round Robin 调度算法，每次需要调度时，将旧任务放置到队列尾部，并从队列头部取出即将要执行的任务（即进程）。我们在理论课上了解到 RR 等调度算法有时间片的概念，表示可以执行多少个时间单位。在内核中，每个时间单位即是两次中断之间的时间。在目前的 xv6 RR 算法中，每个时间片的长度为一次时钟中断的间隔时间。

本次作业要求你在 xv6 上实现一个调度算法：

1. 允许时间片的长度不为 1，即进程的时间片长度可以为 N 次时钟中断的时间。该值是每个进程各自的属性，即不同的进程可以拥有不同的时间片。
2. 实现 Priority Scheduling，对高优先级的进程，它每次执行的时间片更多。
3. 在进程退出后打印出进程的调度信息，包括 Waiting Time 与 Turnaround Time 等。

Build commit hash: 3eeca26

3 Hints

每当时钟中断发生时 (`usertrap()`)，内核会使用 `yield()` 放弃 CPU，该方法会将 CPU 控制权交还给 `scheduler`；`scheduler` 会将其放入就绪队列 (Ready Queue) 的尾部，然后从就绪队列中选择队列头的进程来执行。

当下一个进程被挑选出来时，它要么从 `first_sched_ret` 开始执行第一次被调度时的代码，要么从它从上次 `usertrap` 中的 `yield` 返回。这种情况下，调用链为 `usertrap -> yield -> sched -> swtch --- (to scheduler) --- (go back from scheduler) -- swtch 返回 -> sched 返回 -> yield 返回`。总之，被挑选的进程将最终调用 `usertrapret` 来返回到用户态。

在创建进程时，xv6 会为每个进程分配一个 `struct proc` 结构体，其中包含了该进程的所有信息。你需要在该结构体中添加一些成员变量来保存进程的调度相关的信息。

在 xv6 中，进程的调度是通过 `sched.c` 中的 `scheduler()` 函数来实现的；切换到 scheduler 是通过 `sched()` 函数来实现的。你可能需要在这些函数中修改以完成本次作业。

4 Specifictions

你需要首先实现时间片的长度不为 1 的功能。

你可以在 `struct proc` 中添加一个成员变量来保存每个进程的可用时间片。在每次即将切换进程时，扣除一个时间片，只在剩余时间片为 0 时才真正切换进程以及将其可用时间片设置为 `FULL_QUANTUM`（定义在 *defs.h* 中的宏）。

然后，实现系统调用 `setpriority`。

`setpriority` 系统调用接收一个整数参数，表示进程的优先级，其取值范围为 `[0, 10)`。数值越低的进程优先级越高；即每次重置可用时间片时，优先级越高的进程可用时间片越多。我们推荐使用以下表达式来计算可用时间片：`FULL_QUANTUM - priority * 2`。

syscall 处理代码已经为你在 *syscall_ids.h*、*syscall.c* 和 *user/syscall.h* 中添加了。你需要在 `sched.c` 中实现该系统调用的具体逻辑。

5 Checkpoints

本次作业一共 4 分，有 1 个 Checkpoint 和一份报告。通过唯一的一个 Checkpoint 即 4 分，报告不占分，但是强制要求写。你的报告应该符合以下要求：

- PDF 文件格式的报告。
- 每个 checkpoint 运行成功的截图。
- 对于指引中提出的思考问题，你可以不在报告中回答，但是我们鼓励你思考这些问题并做出回答。

- 记录完成这个作业一共花了多少时间。
- （可选）描述你在完成这个作业时遇到的困难，或者描述你认为本次作业还需要哪些指引。
- （可选）你可以在报告中分享你完成这个作业的思路。
- 越简洁越好。

要求 你应该使用 `make runsmp` 启动**多核心**的 xv6。本次作业我们提供了 `usertest`，在启动内核后，你应该能在 `applist:` 提示符后看到一项 `schedtest`。能稳定通过即视为通过 Checkpoint。你最终打印出来的 Time 等信息可能每次都不一致，但是你只需要保证它们是合理（优先级高的执行更快）的即可。

如果通过了测试，你应该能看到类似下面的输出：

```
===
schedtest: 10-times tests passed
===
```

Checkpoint 说明 该 Checkpoint 会 fork 5 个进程，并调用 `setpriority` 设置它们的优先级。优先级数值越低的进程优先级越高。优先级越高的进程应该使用越少的时间来完成相同的任务（worker）。

6 提交

将你的报告重命名为 report.pdf，放到与该 PDF 文件的同目录下，运行 `make handin`，这会生成一个 handin.zip 压缩包。上传该文件到 Blackboard 即可.