线程机制实习报告

姓名 **彭广举** 学号 1200012964
日期 2015-3-24

目录

内容一：总体概述

本次 lab 主要工作是对线程管理和线程调度的扩展。

1. 阅读代码，理解 nachos 的中断和时钟机制，学习 nachos 的线程调度和线程切换的过程。

2. 实现了基于时间片轮转的优先级抢占式调度。在 system.cc 中显式地声明时钟对象，开启以 TimeTicks 为周期的时钟中断。为每个线程增加 Priority 属性，表示线程优先级，在 Scheduler 类中进行线程调度时，根据运行时间和优先级高低决定将要调度的线程，并且在 FindNextToRun 中实现对就绪队列和当前运行线程的优先级调整。

【用简洁的语言描述本次 lab 的主要内容；阐述本次 lab 中涉及到的重要的概念，技术，原理等，以及其他你认为的最重要的知识点。这一部分主要是看大家对 lab 的总体的理解。

要求：简洁，不需要面面俱到，把重要的知识点阐述清楚即可。】

内容二：任务完成情况

任务完成列表（Y/N）

	Exercise1	Exercise2	Exercise3	Exercise4
第一部分	Y	Y	Y（基于时间片的优先级抢占）	

具体 Exercise 的完成情况

第一部分

Exercise1

调研 **linux** 或 **windows** 中采用的进程**/**线程的调度算法

Linux 中进程分为实时进程和普通进程，实时进程对调度延迟的要求最高，要求立即响应并执行，调度策略是：FIFO 和 Round Robin，普通进程有交互式进程（间或处于睡眠态，对响应速度要求比较高）和批处理进程（后台执行，能够忍受响应延迟），调度策略是 CFS。

CFS，完全公平调度（Completely Fair Scheduler），核心思想是根据进程的优先级按比例分配运行时间。分配给进程的运行时间=调度周期*进程权重/所有进程的权重之和，调度周期是将所有处于 Task_RUNNING 态进程都调度一遍的时间。CFS 引入了虚拟运行时间，

vruntime=实际运行时间*NICE_0_LOAD/进程权重。从虚拟运行时间来看，所有进程分配的时间是相同的。CFS 维护了一个以 vruntime 为顺序的红黑树，每次选择 vruntime 最小的进程运行。

Windows 线程调度，调度单位是线程，采用基于动态优先级的，抢占式调度，结合时间配额调整，总是运行最高优先级的线程。windows 使用 32 个线程优先级，分成 3 类：实时优先级(16-31)，可变优先级(1-15)，系统优先级 0。对线程优先级的维护主要是对可变优先级的线程，调度策略有主动切换，抢占和时间配额用完。

Exercise2

阅读代码

scheduler.h 和 scheduler.cc 是进行线程调度的，主要是一个 Scheduler 类，类里有一个链表 List 用于保存 Ready 状态的线程，实现的函数有 ReadyToRun 接收一个线程指针作为参数，将该线程放入 Ready 队列；FindNextToRun 从 Ready 队列中取出下一个将要被调度的线程；Run 函数进行线程切换，从当前的线程切换到 nextThread 线程，如果是用户线程，先保存虚拟机状态，之后检查运行线程栈是否溢出，设置 currentThread 为 nextThread，并设为运行态，用 SWITCH 完成线程上下文和栈的切换，之后释放 threadToBeDestroyed 线程，最后如果是用户线程，恢复当前虚拟机状态。

switch.s 有用汇编写的 ThreadRoot 函数和 SWITCH 函数，ThreadRoot 函数是除 main 函数以外，其他线程的入口。先调用 StartupPC 函数，进行一些初始化工作，比如开中断，然后，InitialPC 指明新生成线程的入口函数地址，最后 WhenDonePC 是线程结束时需要做的工作。线程第一次被切换到 CPU 时调用 ThreadRoot 函数。SWITCH 函数完成上下文和栈的切换，首先保存原线程的状态，恢复新线程的状态，在新运行线程的栈空间上运行新线程。**对与 SWITCH 中，esp+4 而不是+0 的问题，首先这么做是为了第一次切换到该线程时能够执行 ThreadRoot 指令，但在初始化时 machestate 数组中 PC 也指向 ThreadRoot，所以我认为这个地方+4 和+0 的结果是一样的。**

timer.h 和 timer.cc 定义了一个与时钟有关的类，用于模拟时钟中断。这里需要注意的是，在 Timer 的初始化函数中，使用了一个 TimerHandler 的一个静态函数，而没有使用初始化函数中的 timerHandler 参数作为中断处理函数，这是因为如果这样做的话，当中断来临时就直接进入中断处理了，这样处理结束后也没有机会将下一个时钟中断插入到等待处理的中断队列中，无法实现周期性的中断。TimerHandler 中调用 TimerExpired 函数，该函数先将下一个时钟中断加入等待队列，然后调用真正的时钟中断函数。不能直接用 TimerExpired 方法作为时钟中断的原因是 C++不能直接引用一个类内部方法的指针 。TimeOfNextInterrupt 函数计算下次发生时钟中断的时机。

Exercise3

实现了基于时间片的优先级抢占算法

首先需要选定一个合适的时间片，时间片的初始值已经在 stats.h 中定义 TimeSticks=100，对于目前的测试线程来说，这个时间片大小是差不多合适的，所以这里就没有再做修改。

然后在 scheduler.h 中增加了如下变量，对算法进行改进

```
#define CreatePriority 100     // 创建时的优先级

#define BlockedPriority 75     // 从 Sleep 唤醒时的优先级
```

#define AdaptPace -5 // 调整幅度

#define MinTicks (TimerTicks/4) // 最小时间片，防止线程频繁切换

由于 Ready 队列存储在 List 中，为了与 List 中的 sort 保持一致，这里的优先级的数越大表示其优先级越低，所以 BlockedPriority 是为了从 Sleep 唤醒的线程能够较早地被调度到 CPU。

同时在 Thread 类中增加 priority 变量，表示其优先级，数值越大，优先级越低，并增加对其维护的 getPriority 和 setPriority 函数。在构造函数中设定其初始优先级数值是 CreatePriority。

之后的工作主要都是对 Scheduler 类的修改。首先，在函数 ReadyToRun 中，将线程添加到 Ready 队列需要用 SortedInsert 按照其优先级从小到大插入。然后增加函数 FlushPriority，每当时钟中断到来（发生切换）时，调用该函数将 Ready 队列的所有线程的优先级执行+AdaptPace 操作，即提高其优先级，是为了能够将 Ready 的线程切换到 CPU。为 Scheduler 类增加变量 lastSwitchTicks 记录上次切换时间。当正在运行的线程主动放弃 CPU 或者时间片到达时，当前的线程的优先级执行+(stats->totalTicks – scheduler->getLastSwitchTicks())/25 操作，即根据其运行的时间适当降低其优先级。

最后修改 FindNextToRun 函数，为了区分是从 Yield 还是从 Sleep 调用该函数，为其增加了一个参数 fromSleep 来指明。当 fromSleep 为 true 时，若 readyList 不为空，则直接将链表头元素返回；当 fromSleep 为 false 时，若 readyList 为空返回 NULL，否则需要判断距离上次切换的时间是否大于 MinTicks，若小于则返回 NULL，否则再判断 nextThread 的优先级是否高于当前线程，若不高，则返回 NULL，并把 nextThread 放回队列，否则返回 nextThread 指针，并且设定 lastSwitchTicks。

测试：

同上次的 main,thread1,thread2,thread3 为了体现时间片的执行，在其中断处理函数中添加了输出，运行结果如下：

```
one@ONE-Y400:~/nachos/nachos-3.4/code/threads$ ./nachos
*** thread 0  name:"main" looped 0 times
*** thread 1  name:"thread1" looped 0 times
*** thread 1  name:"thread1" looped 1 times
*** thread 1  name:"thread1" looped 2 times
*** thread 2  name:"thread2" looped 0 times
*** thread 2  name:"thread2" looped 1 times
Ticks: 100, TimeInterrept!
*** thread 3  name:"thread3" looped 0 times
*** thread 3  name:"thread3" looped 1 times
*** thread 3  name:"thread3" looped 2 times
*** thread 3  name:"thread3" looped 3 times
*** thread 1  name:"thread1" looped 3 times
*** thread 1  name:"thread1" looped 4 times
*** thread 1  name:"thread1" looped 5 times
*** thread 2  name:"thread2" looped 2 times
*** thread 2  name:"thread2" looped 3 times
Ticks: 200, TimeInterrept!
*** thread 0  name:"main" looped 1 times
*** thread 0  name:"main" looped 2 times
*** thread 0  name:"main" looped 3 times
*** thread 0  name:"main" looped 4 times
*** thread 3  name:"thread3" looped 4 times
*** thread 3  name:"thread3" looped 5 times
*** thread 2  name:"thread2" looped 4 times
Ticks: 300, TimeInterrept!
*** thread 2  name:"thread2" looped 5 times
*** thread 0  name:"main" looped 5 times
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 400, idle 60, system 340, user 0
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
```

设定 thread1 初始优先级数值为 120，thread2 为 80，thread3 为 95，则运行结果如下：

```
one@ONE-Y400:~/nachos/nachos-3.4/code/threads$ ./nachos
*** thread 0  name:"main" looped 0 times
*** thread 2  name:"thread2" looped 0 times
*** thread 2  name:"thread2" looped 1 times
*** thread 2  name:"thread2" looped 2 times
*** thread 2  name:"thread2" looped 3 times
*** thread 2  name:"thread2" looped 4 times
Ticks: 100, TimeInterrept!
*** thread 3  name:"thread3" looped 0 times
*** thread 3  name:"thread3" looped 1 times
*** thread 3  name:"thread3" looped 2 times
*** thread 3  name:"thread3" looped 3 times
*** thread 2  name:"thread2" looped 5 times
*** thread 0  name:"main" looped 1 times
*** thread 0  name:"main" looped 2 times
*** thread 0  name:"main" looped 3 times
Ticks: 200, TimeInterrept!
*** thread 3  name:"thread3" looped 4 times
*** thread 3  name:"thread3" looped 5 times
*** thread 0  name:"main" looped 4 times
*** thread 0  name:"main" looped 5 times
*** thread 1  name:"thread1" looped 0 times
*** thread 1  name:"thread1" looped 1 times
*** thread 1  name:"thread1" looped 2 times
Ticks: 300, TimeInterrept!
*** thread 1  name:"thread1" looped 3 times
*** thread 1  name:"thread1" looped 4 times
*** thread 1  name:"thread1" looped 5 times
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 400, idle 60, system 340, user 0
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
```

发现 thread1 最后被调度执行，也是符合预期的。

Exercise4

【对于阅读代码类的 exercise，请对其中你认为重要的部分（比如某文件，或某个类、或某个变量、或某个函数……）做出说明。

对于要编程实现的 exercise，请对你增加或修改的内容作出说明。如果增加或修改了某个类，请写出这个类写在那个文件中，它的的功能是什么，你自己添加或修改的成员变量以及成员函数有哪些，它们的功能是什么；特别的，对于复杂的函数，请说明你的实现方

法。不需要贴具体的实现代码。

要求：表述清楚即可， 可采用图表来辅助说明，不需要贴代码】

内容三：遇到的困难以及解决方法

困难 1

……

困难 2

……

【描述你在实习过程中遇到的困难，是与实习直接相关的技术方面的难题。突出说明你是如何攻克这些难关的。

要求：只需写一下有较深体会的困难。如果觉得整个过程都比较简单的话此部分可不用写。】

内容四：收获及感想

对系统的线程的切换过程以及时间片，中断等有关的调度有了更具体的感受和理解。

【自己的收获，任何关于实习的感想，可以是技术方面的或非技术方面的，可随意发挥。

要求：内容不限，体裁不限，字数不限，多多益善，有感而发。】

内容五：对课程的意见和建议

……

【请写下你认为课程需要改进的地方，任何方面，比如进度安排、难易程度、课堂讲解、考核方式、题目设置.....甚至如果你认为源代码哪里写得不好也欢迎提出。

各位同学反馈的信息对课程建设会有极大帮助。】

内容六：参考文献

nachos 文档

【我们希望大家不要使用复制粘贴来拼凑你的报告。详细地列出你在完成 lab 的过程中引用的书籍，网站，讲义，包括你咨询过的大牛们。

要求：诚实，格式尽量按照论文的要求，请参考“论文参考文献格式.doc”】