实验二　线程与同步

(实验指导：os_lab_plus.pdf)

1 内容简述

本次实验的目的在于将 nachos 中的锁机制和条件变量的实现补充完整，并利用这些同步机制实现几个基础工具类。实验内容分三部分：实现锁机制和条件变量，并利用这些同步机制将实验一中所实现双向有序链表类修改成线程安全的；实现一个线程安全的表结构；实现一个大小受限的缓冲区（详细内容请看“NYU-Nachos Project Guide.pdf”）。

2 实验内容的几点说明

2.1 实现锁机制和条件变量（60%）

2.1.1 总体说明

- 这部分是本实验的重点，实际包含三个部分：第一部分要求使用 Thread::Sleep 实现锁机制和条件变量；第二部分要求使用 Semaphore 实现锁机制和条件变量；第三部分要求使用锁机制和条件变量将实验一里实现的双向有序链表修改成线程安全的，对第一、第二部分的实现应分别测试。
- 对锁机制和条件变量的实现，需要修改的文件是
  - nachos-3.4/code/threads/synch.h：class Lock 和 class Condition 分别提供锁和条件变量的接口声明，其中的注释涉及它们的具体语义，应仔细阅读，并根据需要添加适当的数据成员。课本 pp166 及课件有关于条件变量的说明，注意，实验要求采用 Mesa 语义，而非 Hoare 语义。
  - nachos-3.4/code/threads/synch.cc：你的实现应出现在其中，特别注意不要遗漏对方法 isHeldByCurrentThread 的实现。
- 可以阅读 nachos-3.4/code/synchlist.cc 和 nachos-3.4/code/synchlist.h 以加深对锁以及条件变量如何使用的理解。
- 条件变量一定要和某个特定的锁变量配合使用，任何进程在调用条件变量的相关方法（Wait，Signal 和 Broadcast）前都应先对相应锁实施加锁操作。你的程序在实现这些方法时必须做此判断（用 isHeldByCurrentThread）。
- 在实现锁和条件变量的有关方法时应特别注意考虑各种异常情况，防止对这些方法的非法调用。对各种非法调用情况均应使用 ASSERT 检查。“NYU-Nachos Project Guide.pdf”的 3.2.5 节提供了其中的一些情况，请参考。

2.1.2 用 Thread::Sleep 实现锁机制和条件变量

- 这部分实现主要参考 Semaphore 中 Semaphore::P()和 Semaphore::V()的实现（在 synch.cc 和 synch.h 中）。阅读../code/threads/list.h，../code/threads/list.cc，../code/machine/interrupt.h 和../code/machine/interrupt.cc 等文件也有帮助。
- 在必要的时候应关中断，关中断的方法可以参考 Semaphore::P()。请注意考虑究竟什么时候才真正需要关中断！
- 阻塞进程应该挂到相应的阻塞队列中，不同的等待事件应对应不同的阻塞队列。因此你的 Lock 类和 Condition 类中应包含相应的队列首指针（请参考 Semaphore）。
- 为了提交，这部分实验完成后，将 synch.cc 和 synch.h 分别保存为 synch-sleep.cc 和 synch-sleep.h（见“3.1 应提交的内容”）

2.1.3 用 Semaphore 实现锁机制和条件变量

- 不需要自行考虑关中断和阻塞队列维护等问题。
- 在用 Semaphore 实现条件变量时并不那么直接！这里，请特别注意 Semaphore 与条件变量的区别：如果在调用 Semaphore::P()前调用 Semaphore::V()，则 V 操作的效果将积累下来；而如果在调用 Condition::Wait()前调用 Condition::Signal()，则 Signal 操作的效果将不积累。
- 为了提交，这部分实验完成后，将 synch.cc 和 synch.h 分别保存为 synch-sem.cc 和 synch-sem.h（见 3.1）

2.1.4 用锁机制和条件变量修改双向有序链表

- 在实验一里你曾经在 nachos 系统中运行自己编写的链表程序演示一些并发错误，其原因是测试程序未考虑互斥。现在请根据所实现的锁和条件变量机制重写测试文件 threadtest.cc，并确保修改后的多线程并发程序是正确互斥的。
- 可以参考 nachos-3.4/code/threads/synchlist.cc 和 nachos-3.4/code/threads/synchlist.h
- 再次注意：条件变量应和锁配合使用。在使用过程中应注意，不能出现一个条件变量与两个不同的锁对应的情况！

2.2 实现一个线程安全的表结构（20%）

- 头文件 Table.h（在课程网站 lab2-headers.tar.gz 文件内）包含了相应的接口声明。
- 你的实现应该放在 Table.cc 里。请自己新建该文件，并注意修改 Makefile 文件 nachos-3.4/code/Makefile.common 里的有关部分（具体项目可以参考实验一的说明）
- Table.h 中 class Table 的构造函数声明有问题，应该修改。此外，应该在其中补充析构函数的声明，并在 Table.cc 中实现。
- 为实现线程安全，需要用到锁机制。

2.3 实现一个大小受限的缓冲区（20%）

- 头文件 BoundedBuffer.h（在课程网站 lab2-headers.tar.gz 文件内）包含了相应的接口声明。
- 你的实现应该放在 BoundedBuffer.cc 里。请自己新建该文件，并注意修改 Makefile 文件 nachos-3.4/code/Makefile.common 里的有关部分（具体项目可以参考实验一的说明）
- 请在 BoundedBuffer.h 中 class BoundedBuffer 的声明中补充析构函数的声明，并在 BoundedBuffer.cc 中实现之。
- 这个问题实际上是“生产者－消费者”问题。实现时候可以用 Semaphore 机制，也可以使用锁机制配合条件变量。如果使用 Semaphore 机制，则需要三个信号量；如果用锁机制配合条件变量，则需要一个锁和两个条件变量。
- 如果对条件变量与锁配合使用不熟悉，请阅读 nachos-3.4/code/threads/synchlist.cc。

3 实验结果的提交

3.1 应提交的内容：

Makefile.common
main.cc
threadtest.cc
dllist.h
dllist.cc
dllist-driver.cc
synch-sleep.h
synch-sleep.cc
synch-sem.h
synch-sem.cc
Table.h
Table.cc
BoundedBuffer.h
BoundedBuffer.cc
nachos02.doc　　// 实验报告。

3.2 试验提交截止时间：2018 年 5 月 4 日中午 12:00。