CS61第十次课程记录

傅海平

Institute Of Computing Technology,

Chinese Academy Of Sciences

haipingf@gmail.com

December 4, 2011

Contents

1 Topics

线程

2 progress

早上8:30点开始，8:30 - 10:30 学习 lec18.pdf 课程讲义，然后11:00开始讨论学习过程中遇到的问题。

3 learning details

3.1 course sketch

3.1.1 Why to synchronize multiple threads

- Interleaved Execution
- 多个线程执行可以抢占
- 顺序一致性
- 多处理器&多核情况下调度，Cache 管理
- Relaxed Memory Models(Sequential-consistent ordering & Relaxed ordering & acquire-release ordering)
  - Seqential Consistent Ordering
    - sequential-consistent model 是沿袭 Java 的内存模型(确切的说是 sequential-consistent with data-race-free)，所有的 atomic 操作都可以 看作满足唯一 total order 的操作，也就是说，可以把这样的多线程程序理 解成一个有先有后交叉运行的单线程程序(对atomic type而言)，这在推理时 是有帮助

的。每个 shared atomic 对象的改变在每个线程看来都是一样的， 包括时间顺序以及值。

- Relaxed Memory Ordering
  - 完全不参与多线程间的同步，编译器可以随便优化。
  - 对于自身线程，data-dependency 产生的依赖需要满足，这是单线程程序 正确性的要求。
  - 此外它唯一的作用就是属于原子操作，所以不会产生 data race。

- Acquire Release Ordering
  - 该语义即同步语义，每一个 acquire 对应一个 release(跨线程的)，详 细定义参见标准(29.3-2)。需要注意的是 C++ 的同步语义不是先验的，我们 无法通过之前的执行判断之后两个操作是否同步了，而只能通过推理所有的情 况来验证程序的正确性。具体来说，对于一个可能的 acquire-release pair ，我们必须考虑他们同步时的情况，也需要考虑他们没有同步时的情况。推理 C++ 多线程程序的正确性，特别是在这两种 ordering 情况下，需要塑模 happens-before 的关系，通俗点说，就是推理出相关操作之间 所有可能发生的顺序(或者根本就无序)，所以说带锁的多线程的程序很难写正 确，更别说无锁的，可见一斑。

3.1.2 Race conditions

- 并行程序访问共享变量时没有线程&进程间的同步。
- 程序执行结果是非确定的。
- 现实中的例子
- 解决方案

3.1.3 Mutual exclusion and critical sections

- 互斥和临界区
- 临界区定义

3.1.4 Locks

- acquire
- release
- 锁机制实现
  - 实现锁机制需要硬件支持
  - 禁止中断：防止上下文切换，但是该方法只在单处理器上有效
- spinlock自旋锁
- 原子操作：CPU保证整个操作是原子的
  - Test-and-set
  - Compare-and-swap
- 忙等待，效率
- 用户空间锁机制实现：futex

3.1.5 Efficiently implementing locks

- 见讲义

3.2 Problems

3.3 Solutions

4 测试