Memory Consistency

Memory Consistency 即是调度来自不同处理器的访存操作

关键：调度**全局**的对**所有内存地址**的访存操作

区别 cache coherence：调度局部的对每个 cache block 的访存操作

Ensuring Memory Consistency

memory ordering 即是使访存操作按照适当的顺序进行

- 单处理器
  - 按序：硬件按照程序顺序执行 load/store
  - 乱序：硬件按照程序顺序 retire load/store
- MIMD 处理器
  - 每个处理器上的访存操作按照运行其上的线程的顺序进行
- IO 设备
  - IO 设备直接读写内存

软件的需求是

- 正确地执行多线程
- 互斥访问：对共享数据的访问放在临界区中，使用同步机制来限制，同时只有一个线程运行临界区的代码

硬件能提供的是

- 支持同步的硬件原语
- 确保互斥被正确实现

Sequential Consistency

> A system is sequentially consistent if the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in the order specified by the program.

即不同处理器的操作可以交叉进行，但每个处理器的操作必须按序

但是没有优化空间

有四种 sequential consistency：R-W，R-R，W-R，W-W

通过放松不同的 consistency，可以得到不同的模型

- TSO (Total Store Ordering) 不限制 W-R
- PSO (Partial Store Ordering) 不限制 W-R 和 W-W
- RS (Release Consistency) 不做限制，用 fences 来定义边界，用 acquire and release 来 flush

Locks or Semaphores

semaphore 是一个非负整数，支持两种操作：P/V

- P 将 s 的值减一，如果 s 小于 0，则挂起
- V 将 s 的值加一，如果有挂起的线程，将其唤醒

用 PV 操作可以实现临界区的互斥访问

PV 操作必须具有原子性

semaphore 可以用软件或硬件实现

- 软件
  - 在 sequential model 里用普通 load/store 实现
- 硬件
  - Test & Set
  - Fetch & Add
  - Swap

可以实现非阻塞的同步

Compare & Swap

```
CS(m, rt, rs){
    if(rt == M[m]){
        M[m] = rs;
        rs = rt;
        status = success;
    } else {
        status = false;
    }
}
```

Load-link & Store-conditional

```
LL(r, m){
    <flag, addr> = <1, m>;
    r = M[m];
}
SC(m, R){
    if(<flag, addr> == <1, m>){
        cancel other proc reservation on m;
        M[m] = r;
        status = succeed;
    } else {
        status = fail;
    }
}
```

如果基于软件实现，需要设计比较好的协议，如 Dekker 或 Peterson（详见 OS 笔记）