Memory Consistency and Cache Coherence

- Memory Consistency and Cache Coherence 定义
- TSO（Total Store Ordering）内存模型
- x86的多核宽松内存一致性模型

Memory Consistency and Cache Coherence 定义

1. Memory Consistency (内存一致性)

- 定义：内存一致性是指多核或多处理器系统中，多个处理器对共享内存的访问顺序是否一致，以及这些访问操作是否满足特定的规则或模型。
- 关注点：内存操作的全局可见顺序，确保所有处理器对内存的读写操作按照一致的顺序进行。
- 问题背景：在多核系统中，不同处理器可能同时访问共享内存，如果没有明确的内存一致性模型，可能会导致程序行为不可预测。
- 内存一致性模型：
  - 顺序一致性（**Sequential Consistency**）：所有处理器的内存操作按照一个全局顺序执行，且每个处理器的操作顺序保持不变。
  - 弱一致性模型（**Weak Consistency**）：允许某些操作乱序执行，但需要通过同步操作（如内存屏障）来保证一致性。
- 示例：
  - 如果两个处理器分别执行写操作和读操作，内存一致性模型定义了读操作是否能立即看到写操作的结果。

---

2. Cache Coherence (缓存一致性)

- 定义：缓存一致性是指多核或多处理器系统中，每个处理器的私有缓存与共享内存之间数据的一致性，确保所有处理器看到的数据是最新的。
- 关注点：多个缓存副本之间的数据一致性，避免脏数据或过时数据。
- 问题背景：每个处理器都有自己的缓存，如果某个处理器修改了缓存中的数据，其他处理器的缓存可能仍然保存旧值，导致数据不一致。
- 缓存一致性协议：
  - **MESI**协议：通过标记缓存行的状态（Modified、Exclusive、Shared、Invalid）来维护一致性。
  - 写失效（**Write Invalidate**）：当一个处理器修改数据时，使其他处理器的缓存副本失效。
  - 写更新（**Write Update**）：当一个处理器修改数据时，将新值广播给其他处理器的缓存。
- 示例：

  - 如果处理器A修改了某个内存位置的值，处理器B的缓存中该位置的旧值必须被更新或失效。

3. 主要区别

特性	**Memory Consistency**（内存一致性）	**Cache Coherence**（缓存一致性）
关注点	内存操作的全局顺序和可见性	多个缓存副本之间的数据一致性
范围	涉及所有内存操作（读/写）的顺序	仅涉及缓存和内存之间的数据一致性
问题背景	多个处理器对共享内存的访问顺序是否一致	多个处理器的缓存中数据是否一致
解决方案	内存一致性模型（如顺序一致性、弱一致性）	缓存一致性协议（如MESI、写失效、写更新）

4. 总结

- **Memory Consistency** 关注的是多处理器系统中内存操作的全局顺序和可见性。
- **Cache Coherence** 关注的是多处理器系统中缓存副本之间的数据一致性。

TSO（Total Store Ordering）内存模型

1. TSO（Total Store Ordering）是一个被广泛使用的内存模型
2. 并在x86架构中使用，RISC-V也提供了TSO扩展，即RVTSO，人们普遍认为x86内存模型等同于TSO，然而Intel和AMD从来没有保证这一点
3. x86选择放弃SC（顺序一致性sequential consistency），以更好地支持基于FIFO的write buffer，用于加速性能
4. TSO和SC最关键的区别就是store可能被放入write buffer中且允许load的bypass
    1. 对于单核视角来说，和SC没有区别，执行顺序和程序顺序一致
    2. 对于多核来说，需要额外的手段来保证memory consistency
5. 由于TSO只允许store-load的重排，因此需要使用FENCE的场合不多，FENCE的实现方式也不是很重要。一种简单的实现是，当FENCE被执行时，清空write buffer，并在FENCE提交之前不再执行后面的load指令。

x86的多核宽松内存一致性模型

1. 被修饰的汇编指令成为 “原子的 ”
   1. 本身是原子指令，比如 “XCHG”和 “XADD”汇编指令
   2. 本身不是原子指令，但是被 `LOCK指令前缀` 修饰后成为原子指令，比如 LOCK CMPXCHG
   3. 被修饰的汇编指令 A在执行期间，会在内存总线上声言一个 `#LOCK` 信号，该信号导致内存被锁住，此时内存不能再被其他汇编指令存取，直到 A执行完成。经过分析可知，A的执行效果与 “暂停执行其他所有汇编指令直到 A执行完成等价，因此此时 A是原子的
2. fence
   1. sfence: 在 sfence指令前的写操作当必须在 sfence指令后的写操作前完成
   2. lfence: 在 lfence指令前的读操作当必须在 lfence指令后的读操作前完成
   3. mfence: 在 mfence指令前的读写操作当必须在 mfence指令后的读写操作前完成
3. C++11中支持的内存模型
   1. Acquire-Release能保证不同线程之间的 Synchronizes-With 关系，同时也约束到同一个线程中前后语句的执行顺序。
   2. Release-Consume只约束有明确的 carry-a-dependency 关系的语句的执行顺序，同一个线程中的其他语句的执行先后顺序并不受这个内存模型的影响

```
enum memory_order {
    memory_order_relaxed, // Relaxed
    memory_order_consume, // Release-Consume
    memory_order_acquire, // Acquire-Release
用来修饰一个读操作，表示在本线程中，所有后续的关于此变量的内存操作都必须在本条原子操作完成后执行
    memory_order_release, // Acquire-Release
用来修饰一个写操作，表示在本线程中，所有之前的针对该变量的内存操作完成后才能执行本条原子操作
    memory_order_acq_rel, // Acquire-Release 同时包含 memory_order_acquire和 memory_order_release标志
    memory_order_seq_cst  // 顺序一致性模型
};
```