# 实验四 实现多 Cache 一致性模拟器

**PB13011079 杨智**

---

# 实验目的

1. 加深对多Cache一致性的理解；
2. 进一步掌握解决多Cache一致性监听协议或（和）目录协议的基本思想；
3. 掌握在各种情况下，监听协议或（和）目录协议是如何工作的。能给出要进行什么样的操作以及状态的变化情况。

---

# 实验环境

监听法：

操作系统：mac osx 10.10
IDE：xcode 6.4
编译器：clang 7.0
界面：Qt creator 4.0.1 + Qt 5.6.1

目录法：

（平时都是用mac的，但是mac上没有c#，所以只好在windows上做）
操作系统：windows 10
IDE：Visual studio 2015 community
语言：C#

---

# 实验要求

设计与实现一个多 Cache 一致性模拟器

1. 基本要求，必须要完成：
   a. 你可以选择完成监听协议或者目录协议中的一种。
   b. 模拟器可配置cache 的个数、cache 块的大小、cache 映射方式等。
2. 较高要求：
   a. 两种协议都完成。
   b. 具有友好的操作界面及展示，如带有界面的动画显示。
3. 模拟器完成之后，请参照表1或表2填写和回答相关问题。

4. 实验需要提交的内容包括：
   a. 实验源代码
   b. 实验最终的可执行文件
   c. 实验报告（包括设计思想、实验分析结论等）

# 实现的要求

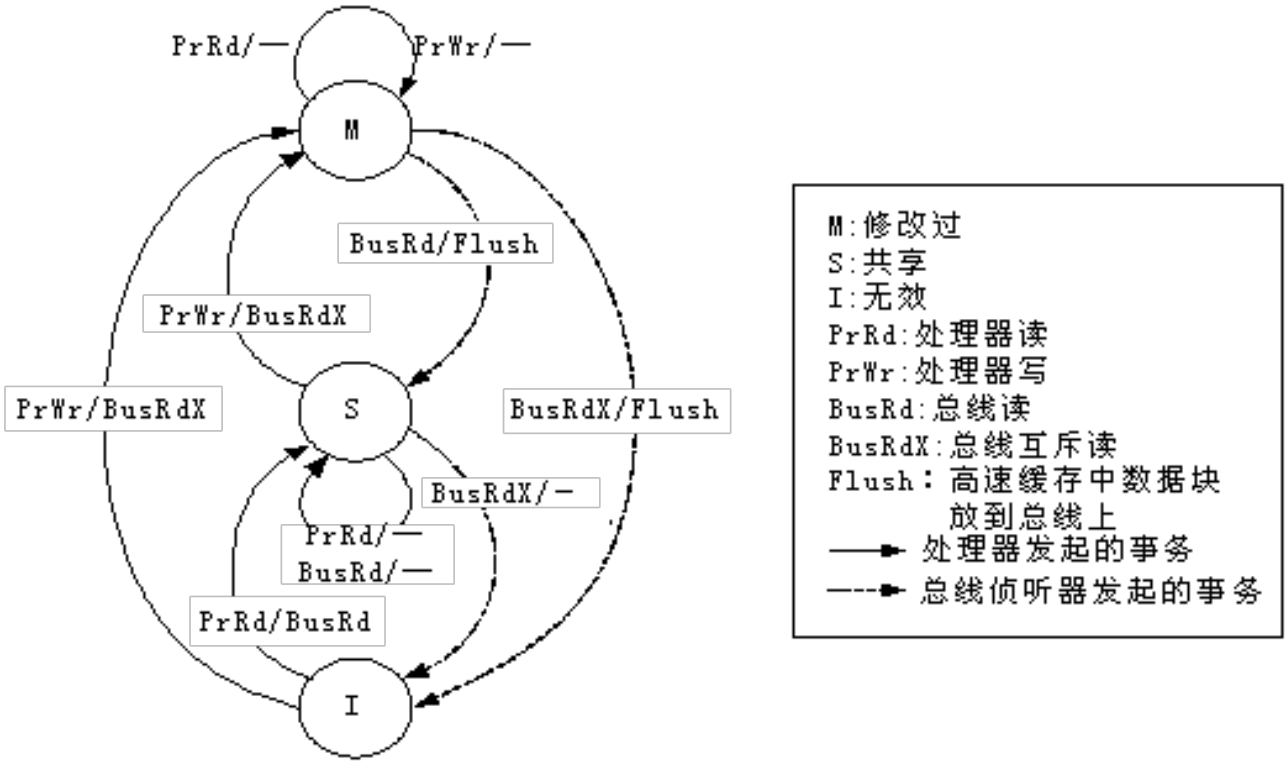
## 监听法和目录法都实现了，同时目录法可以选择Cache的映射方式，支持简单的动画效果

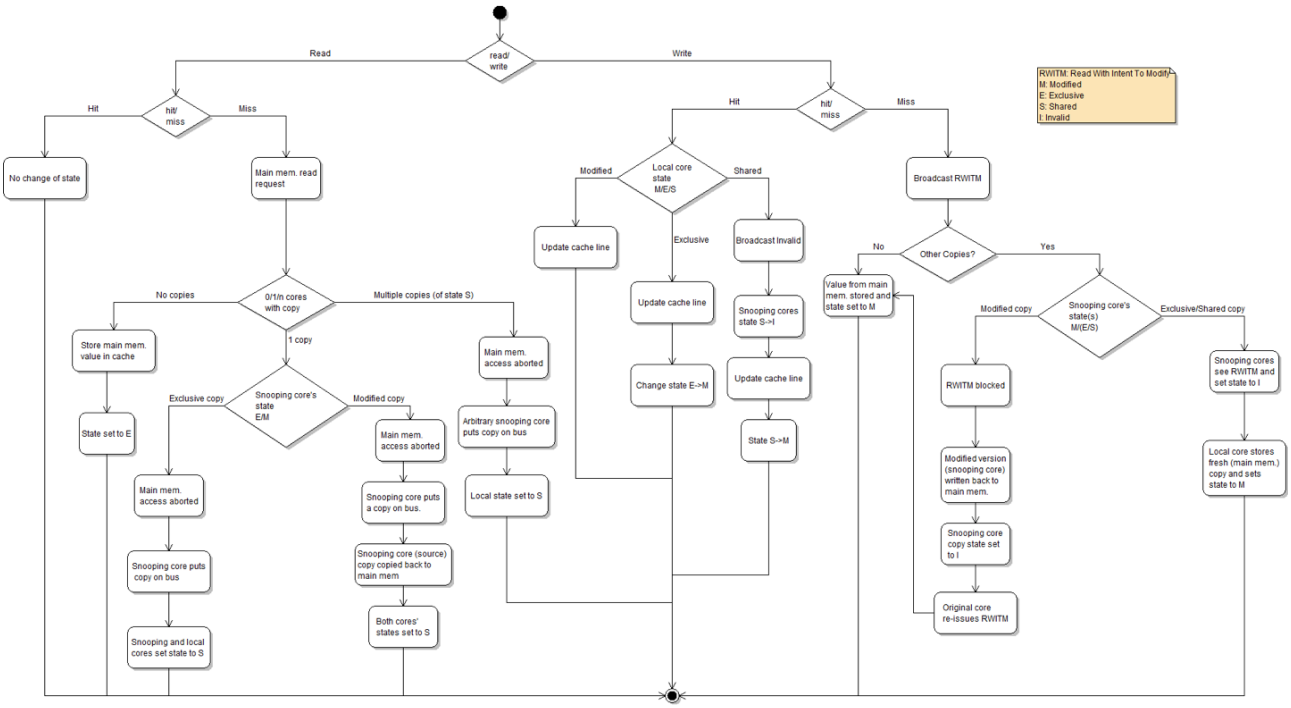
# 监听协议

| 所进行的访问 | 是否发生了替换? | 是否发生了写回? | 监听协议所进行的操作 |
|---|---|---|---|
| CPU A 读第5块 | 否 | 否 | CPU A读第5块不命中，在总线上放read miss信号，之后第5块从存储器传送到Cache A 1，Cache A 1设为shared |
| CPU B 读第5块 | 否 | 否 | CPU B读第5块不命中，在总线上放read miss信号，之后第5块从存储器传送到Cache B 1，Cache B 1设为shared |
| CPU C 读第5块 | 否 | 否 | CPU C读第5块不命中，在总线上放read miss信号，之后第5块从存储器传送到Cache C 1，Cache C 1设为shared |
| CPU B 写第5块 | 是 | 否 | CPU B写命中，在总线上放invalid信号，把其他Cache中的相应的块设为invalid，把本Cache中的相应的块设为exclusive |
| CPU D 读第5块 | 否 | 是 | CPU D读不命中，在总线上放read miss信号，CPU B收到这个信号之后Flush，把该块从Cache B传送到Cache D，同时把这个块都设为shared |
| CPU B 写第21块 | 是 | 否 | CPU B写不命中，在总线上放write miss信号，从mem中读入该块，把本Cache中的相应的块设为exclusive |

| | | | |
|---|---|---|---|
| CPU A 写第23块 | 否 | 否 | CPU A写不命中，在总线上放write miss信号，从mem中读入该块，把本Cache中的相应的块设为exclusive |
| CPU C 写第23块 | 否 | 是 | CPU C写不命中，在总线上放write miss信号，Cache A收到这个信号之后Flush，该块从Cache A传送到Cache C，Cache C中该块都设为exclusive，Cache A中该块设为invalid |
| CPU B 读第29块 | 是 | 是 | CPU B读第29块不命中，在总线上放read miss信号，之后第29块从存储器传送到Cache B，原本在Cache B 1的块被替换Bing写回，Cache B 1设为shared |
| CPU B 写第5块 | 是 | 否 | CPU B写不命中，在总线上放write miss信号，从mem中得到该块，并设为exclusive，原本在这个位置的块被替换；而在Cache中，缓存的第5块被置为invalid |

根据上述结果，画出相关的状态转换图。
（我没懂这个是什么意思，是画MSI协议的状态转移图吗？还是画上面这个表的状态转移？）





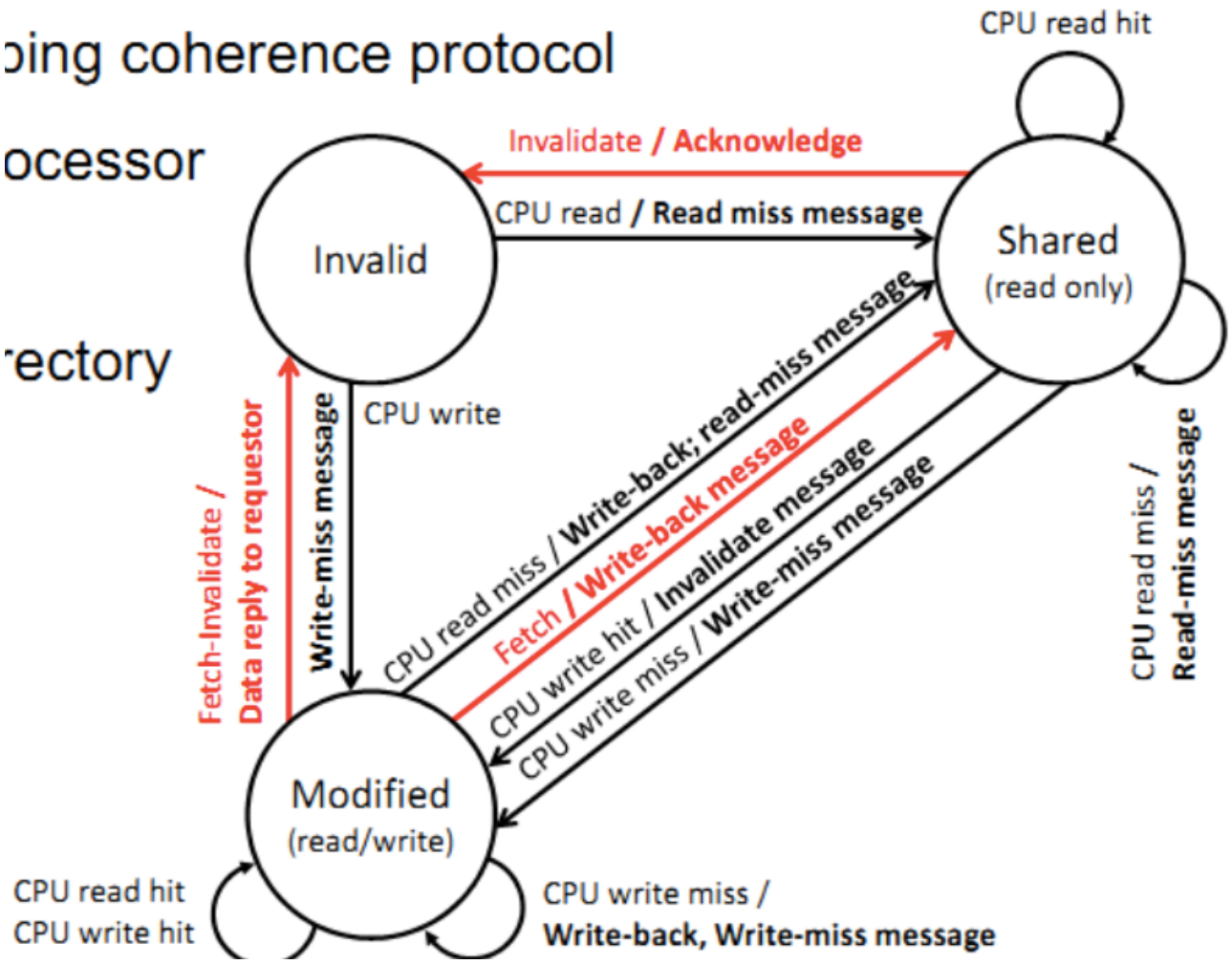

# 目录协议

| 所进行的访问 | 目录协议所进行的操作 |
| --- | --- |
| CPU A 读第6块 | 1．读;2.不命中;3 本地:向宿主结点发读不命中(A，6)消息；4.宿主：把数据 块送给本地结点；5.共享集合为：{A} |
| CPU B读第6块 | 1． 读；2.不命中；3. 本地:向宿主结点发读不命中(B，6)消息；4．宿主：把数据块发送给本地结点；5.共享集合为：{A}+{B} |
| CPU D 读第6块 | 1．读；2.不命中；3. 本地:向宿主结点发读不命中(D，6)消息；4．宿主：把数据块发送给本地结点；5.共享集合为：{A，B}+{D} |
| CPU B 写第6块 | 1．写；2.命中；3. 本地:向宿主结点发写命中(B，6)消息，宿主：向远程结 点 A 发作废（6）消息，宿主：向远程结点 D 发作废（6）消息； 4.共享集 合为： {B} |
| CPU C 读第6块 | 1 读；2.不命中；3. 本地:向宿主结点发读不命中(C，6)消息；4．宿主：给远程结点发取数据块（6）的消息；5.远程：把数据块送给宿 主结点；6.宿主：把数据块送给本地结点；7.共享集合为：{B}+{C} |

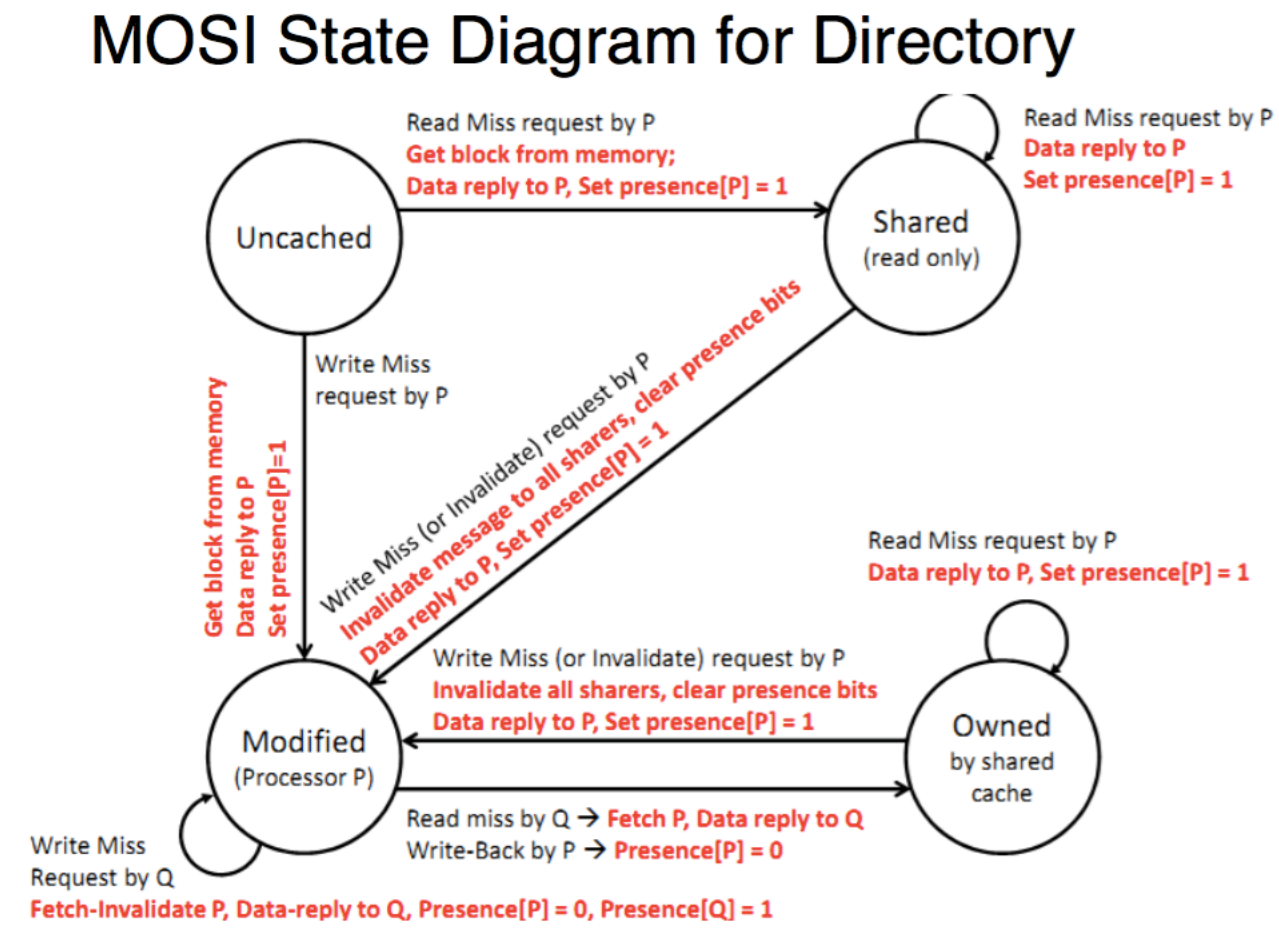
| | |
|---|---|
| CPU D写第20块 | 1．写；2.不命中；3. 本地:向宿主结点发写不命中(D，20)消息；4．宿主： 把数据块发送给本地结点； 5.共享集合为： {D} |
| CPUA写第20块 | 1．写；2.不命中；3. 本地:向宿主结点发写不命中(A，20)消息；4．宿主： 给远程结点发送取并作废（20）消息；5.远程：把数据块送给宿主结点把 Cache 中的该块作废；6.宿主：把数据块送给本地结点；7.共享集合为：{A} |
| CPU D写第6块 | 1．写；2.不命中；3. 本地:向宿主结点发写不命中(D，6)消息；4.宿主：向 远程结点发作废（6）消息；5.宿主：向远程结点发作废（6）消息；6.宿主： 把数据块送给本地结点；7.共享集合为：{D} |
| CPU A 读第12块 | 1． 写； 2.不命中； 3.本地： 向被替换块的宿主结点发写回并修改共享集 （A， 20）消息；4.本地：向宿主结点发写不命中（A，20）消息；5.宿主：把数据 块送给本地结点；6.共享集合为：{A} |

根据上述结果，画出相关的状态转换图。

# local cache 状态转移图

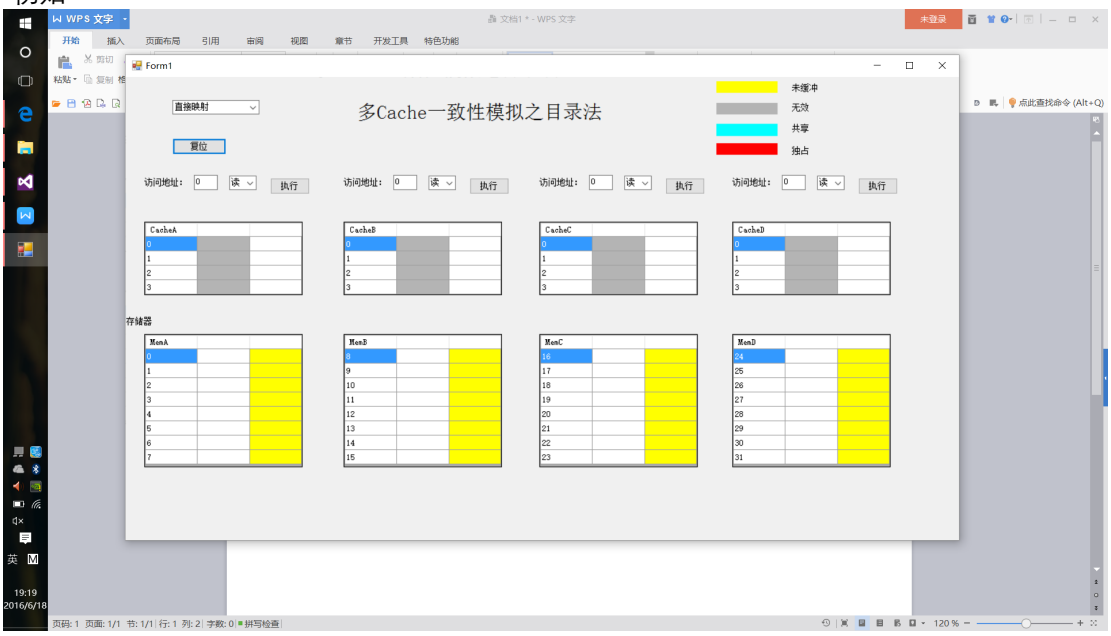


## Directory 状态转移图

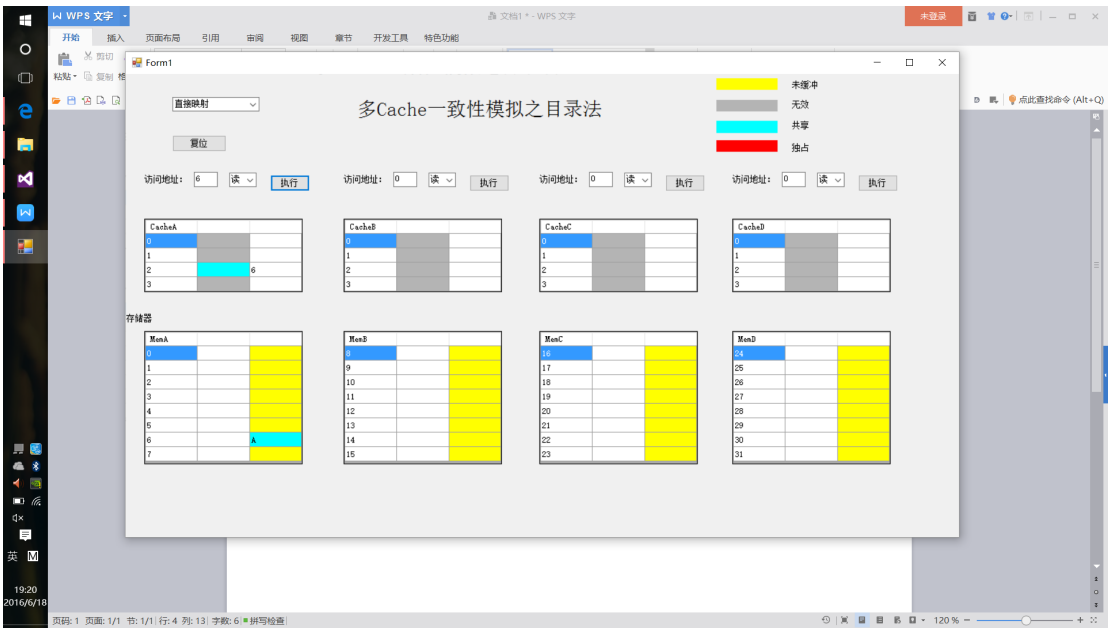



## 实验截图

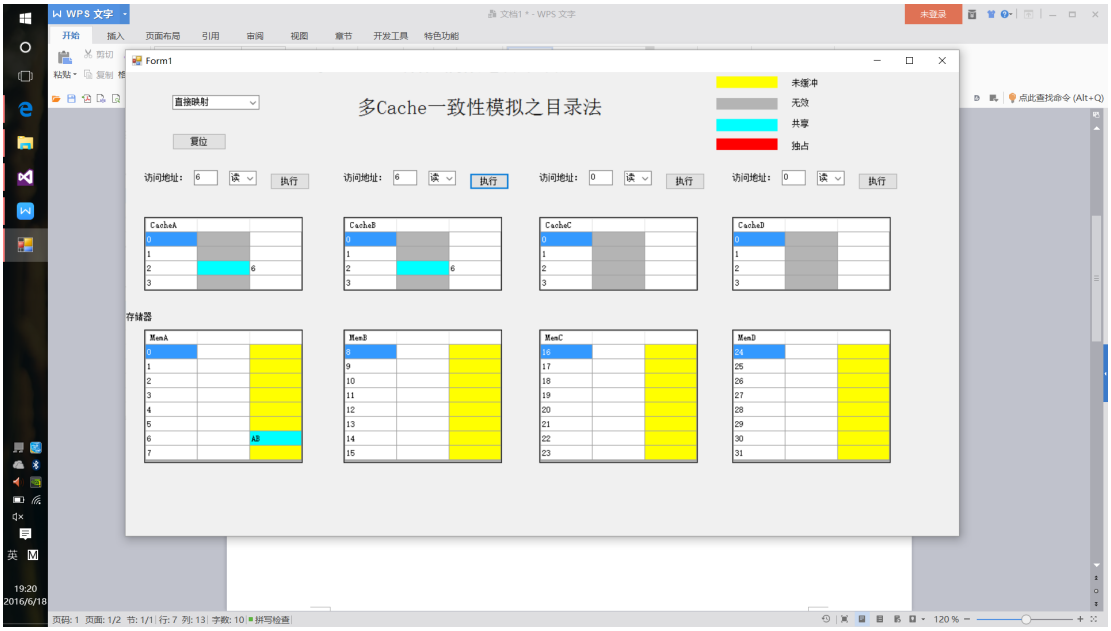
初始



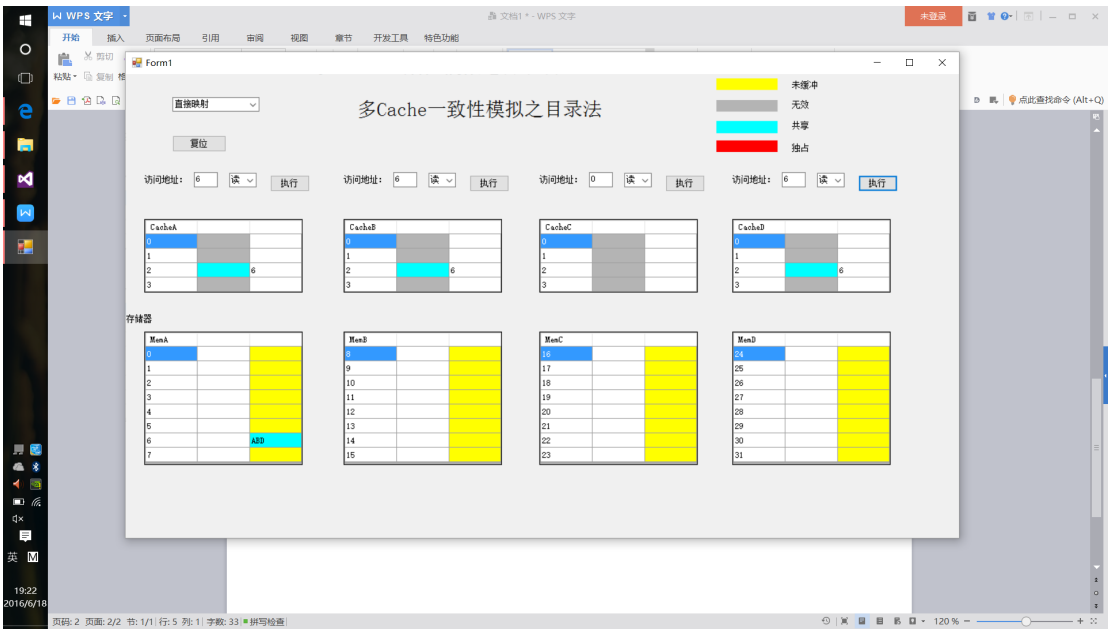

CPU A read 6



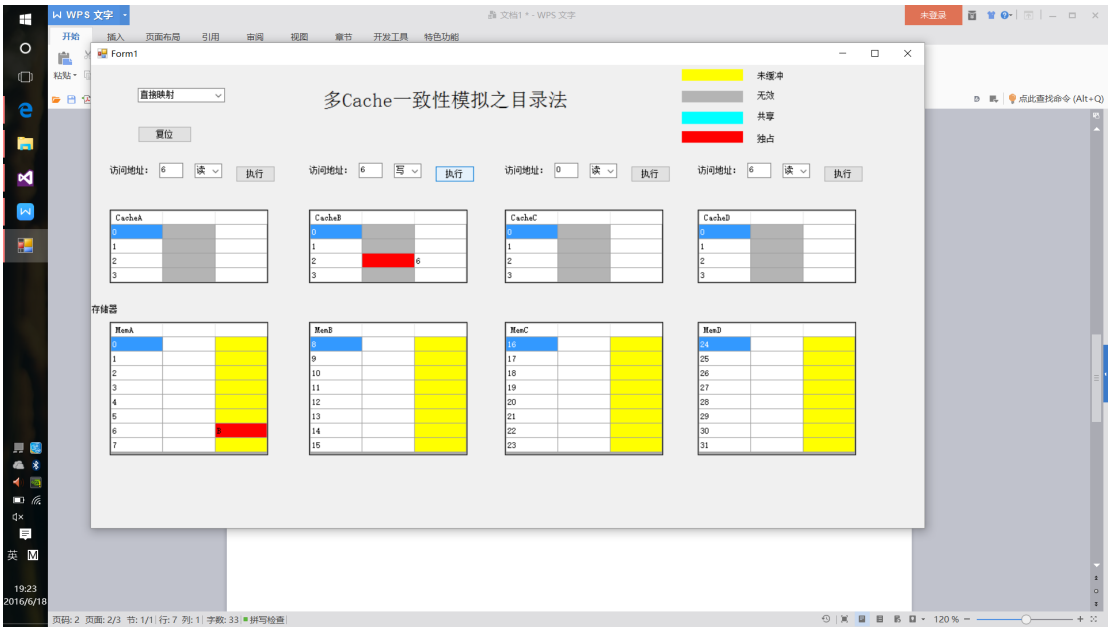

CPU B read 6



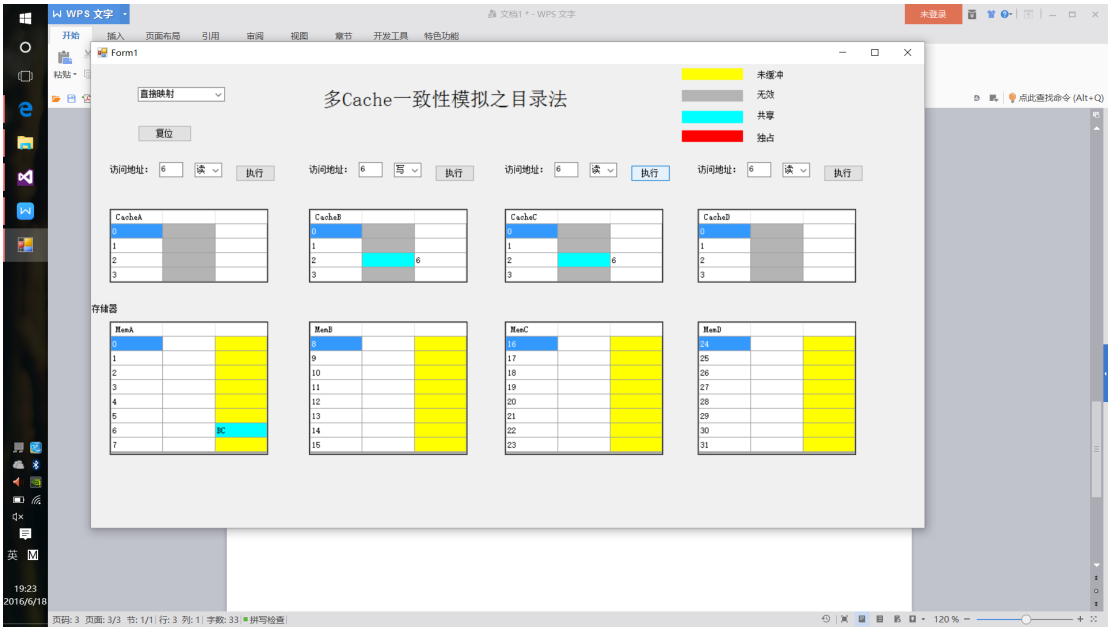

CPU D read 6



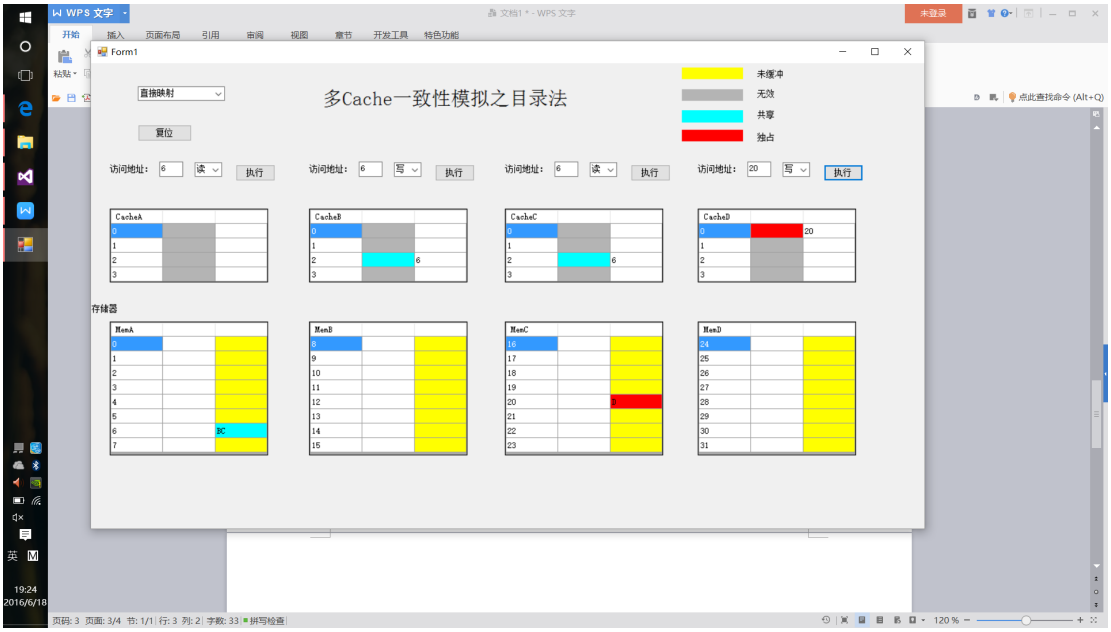

CPU B write 6



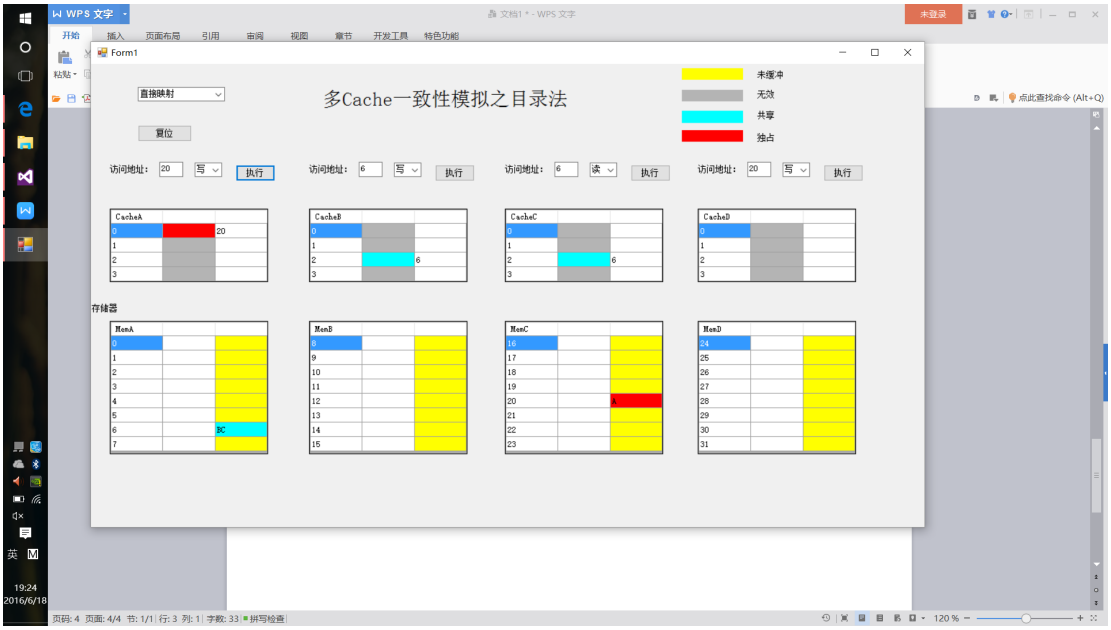

CPU C read 6



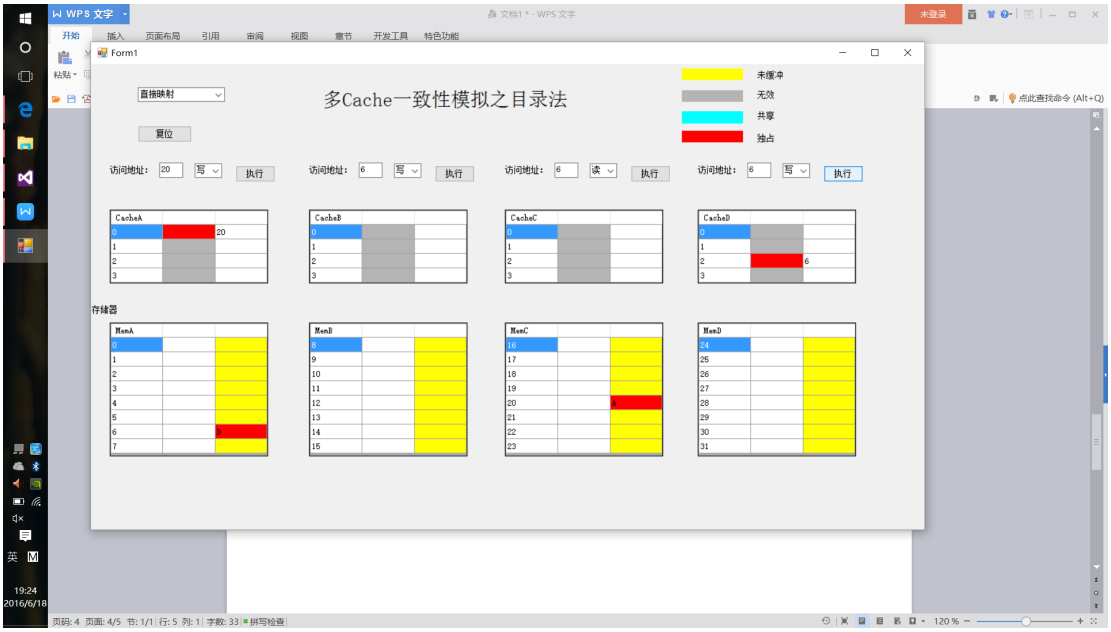

CPU D write 20



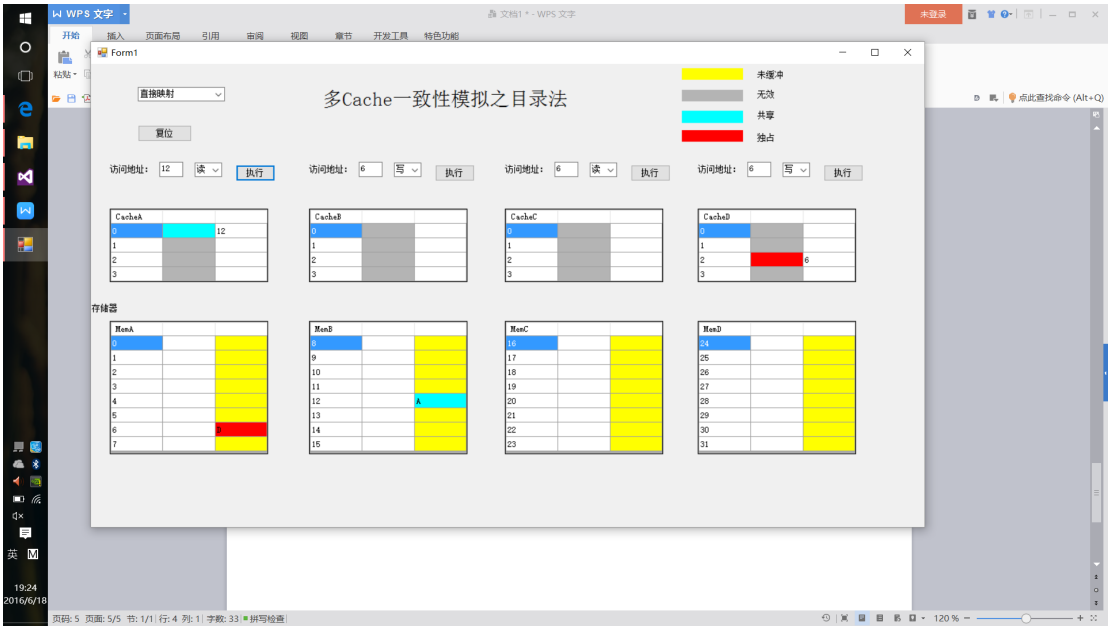

CPU A write 20




CPU D write 6




CPU A read 12




# 设计思想

设计思路即目录法和监听法的原理，在上面就画了状态转移图，按着这个状态转移图就可以用编程语言实现一个自动状态机。这里不赘述。

# 实验总结

通过这次实验我了解了目录法和监听法的基本原理。通过这两种协议，我们可以实现Cache的一致性，实现多CPU、Cache的协同工作。

# 实验结论

监听法和目录法都可以很好地维护**Cache**一致性。