lab5-report

- 实验目的
  - 加深对多Cache一致性的理解
  - 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
  - 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
- 实验要求

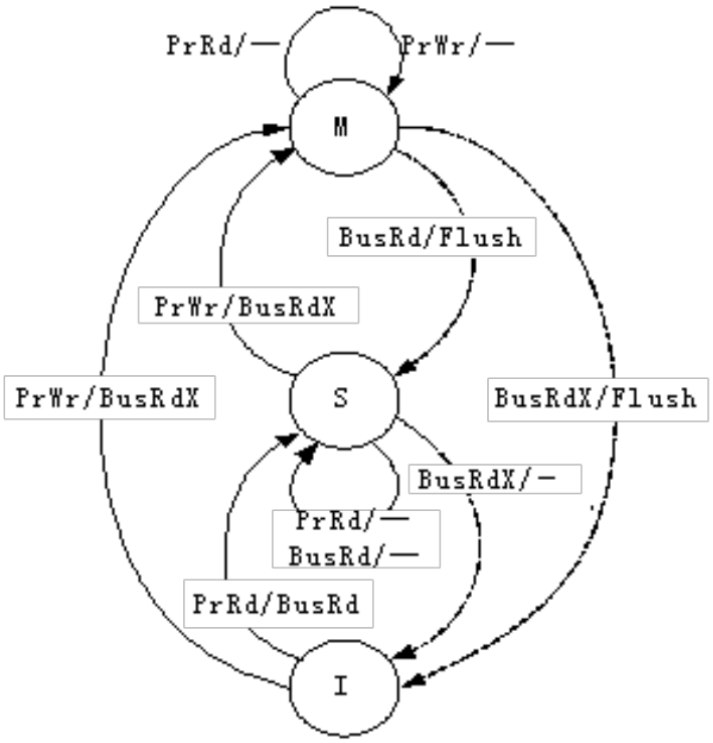
  设计与实现一个多 Cache一致性模拟器
  - 完成监听协议和目录协议
  - 模拟器可配置cache 映射方式
  - 带有界面
- 实验环境
  - 操作系统：windows 10
  - 编译器：jdk 1.8
  - CPU：Intel i5-7200U
  - memory：8GB
- 编译执行说明

```
// 编译指令
javac snoop.java
javac directory.java
// 执行
java snoop
java directory
```

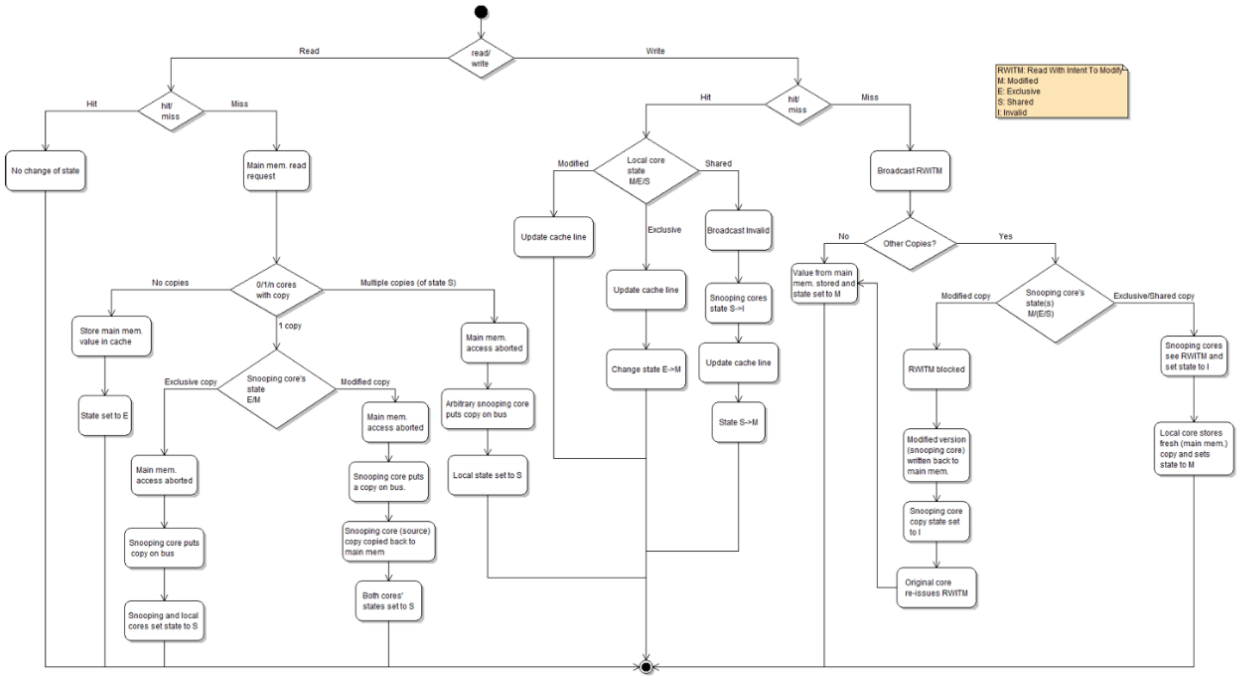
- 实验问题回答
  - 对于监听协议：

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。



M:修改过
S:共享
I:无效
PrRd:处理器读
PrWr:处理器写
BusRd:总线读
BusRdX:总线互斥读
Flush：高速缓存中数据块放到总线上
处理器发起的事务
总线侦听器发起的事务

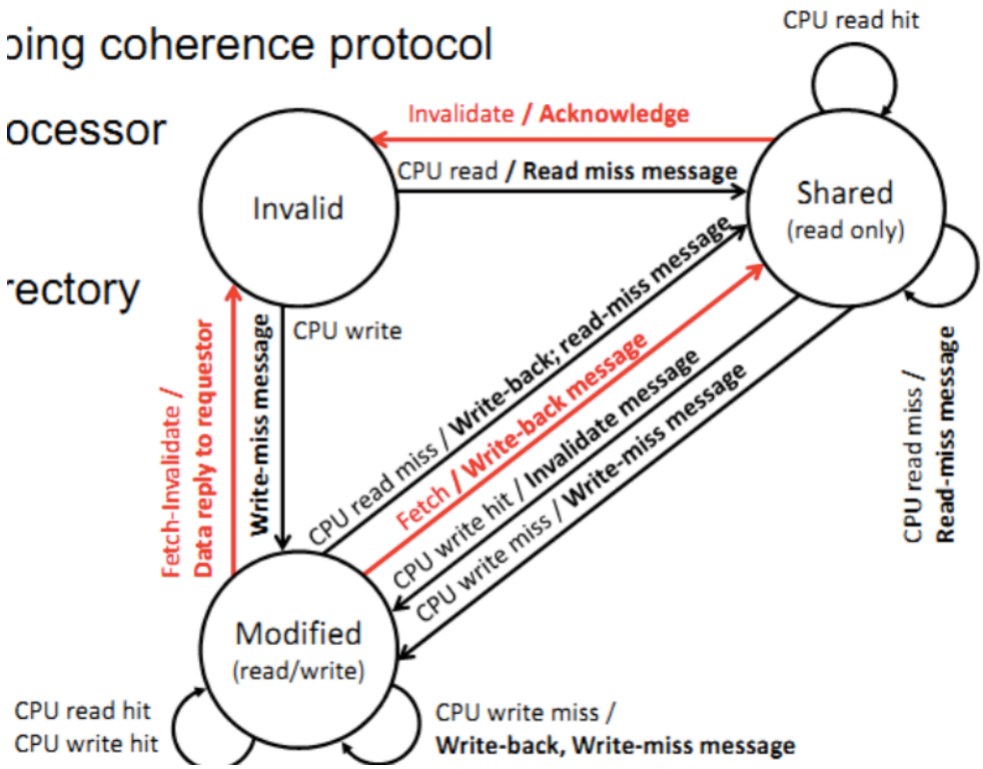


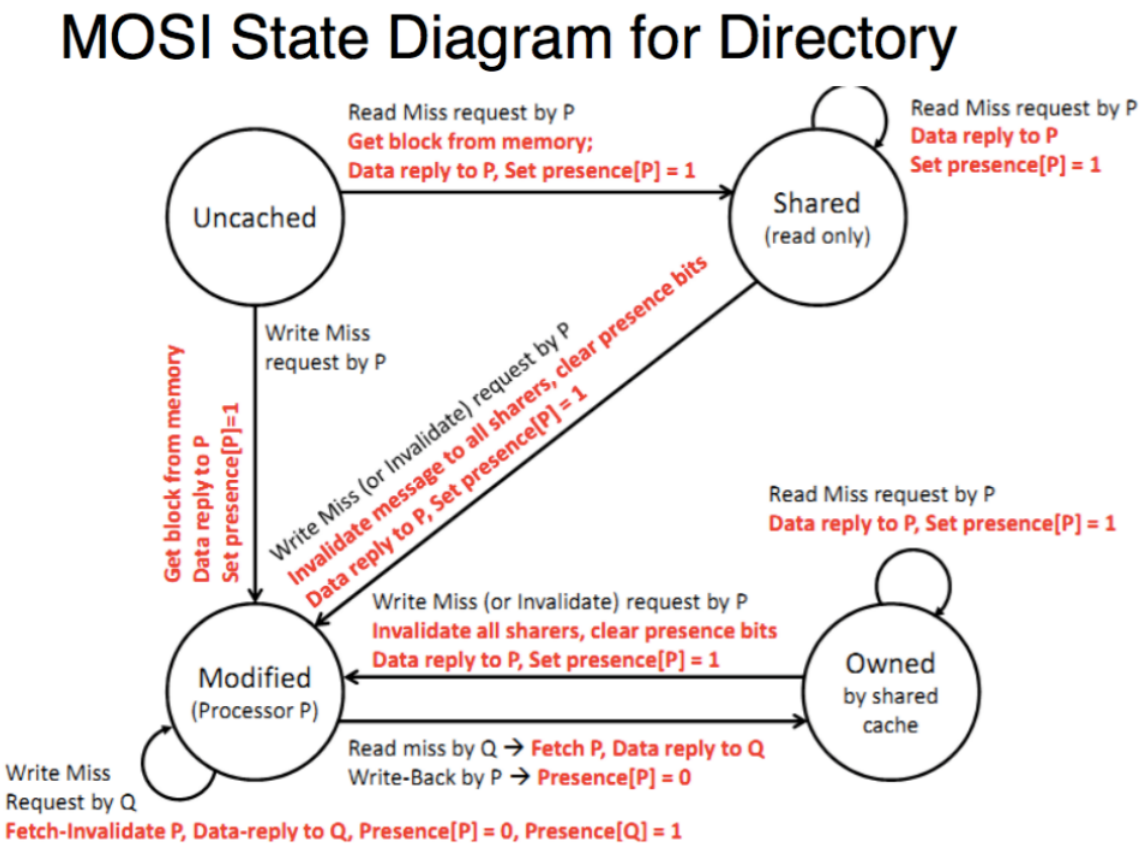
- 对于目录协议：

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。





- 实验设计思想

  根据实验要求，需要模拟多处理器中cache的一致性的实现，包括目录协议和监听协议。

  - 实验中目录协议和监听协议的实现

监听/目录协议是对每个处理器中的cache进行状态标记，在cpu发出读写操作请求时，如果在cache中读命中直接读取回cpu即可，如果写命中则要根据cache中的现有状态进行相应的处理，以及产生总线事务；不命中时需要从memory /远程cache中读取，其中可能需要考虑到块的替换操作，之后具体的操作根据上述的状态图实现即可，不再赘述。

- 配置**cache**映射方式的实现

  在实验中实现了直接映像，两路组相联映射和四路组相联映射。在发送cache缺少时选定的cache块要根据这三种方式的选择来决定，当通过前端gui交互的选择栏中获取到映射方式为直接映像时，在块不命中时，对于该地址的cache块位置唯一固定为addr mod 4，如果已经存在其他地址，需要进行替换。同理对于两路组相联，一个地址addr所能占用的cache块的编号为addr mod 2，四路组相联可以选择任意一个空闲的cache块。

- 传块优化的实现

  以监听法为例，当cpu发出读操作请求，在cache中读不命中并且此时该地址在另一处理器中进行写操作后独占，如果不进行传块优化，当前cpu读取该地址数据时需要等到另一处理器将其独占的cache块写回memory然后改cpu的cache再从memory读回。采用传块优化后，在此情况下可以直接从独占块的处理器的cache传回到当前请求cpu的cache中，即发生cache to cache，这样可以降低通信延迟，具体实现时需要检测地址是否被某个处理器独占然后进行cache to cache的操作即可。

- 单步执行的实现

  为了更清晰的展示cache一致性的模拟过程，在实现时允许设置单步执行，在交互的前端界面可以选择"单步执行"或者"连续执行"，当检测到选择为单步执行时，在模拟实现中最关键的就是功能函数在执行到某一位置处需要停止并且更新UI，我采用的策略是在实现函数中每个单步需要停下的地方设置标记断点，即每一次读/写请求操作都会初始化一个step值为0，并且每执行一步step++，通过一个全局变量来控制整个函数的执行，需要停止时返回更新UI，执行结束后step恢复为0。

实验分析总结

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，可以实现Cache的一致性，实现多CPU、Cache的协同工工作，监听法和目录法都可以很好地维护Cache一致性。