中山大學

SUN YAT-SEN UNIVERSITY

实 验 报 告

课程名称： 操作系统

姓　　名： 孙广岩

学　　号： 20354242

专业班级： 智能科学与技术专业 5 班

任课教师： 吴贺俊

2022 年 10 月 9 日

实验报告成绩评定表

评定项目	内　　容	满 分	评　分	总　分
实验态度	态度端正、遵守纪律、出勤情况	10		
实验过程	按要求完成算法设计、代码书写、注释清晰、运行结果正确	30		
实验记录	展示讲解清楚、任务解决良好、实验结果准确	20		
报告撰写	报告书写规范、内容条理清楚、表达准确规范、上交及时、无抄袭，抄袭记 0 分，提供报告供抄袭者扣分。	40		
评语： **指导老师签字：**　　年　　月　　日				

实验二　仿真存储系统

一、实验目的

1. 掌握三级储存系统的思想。

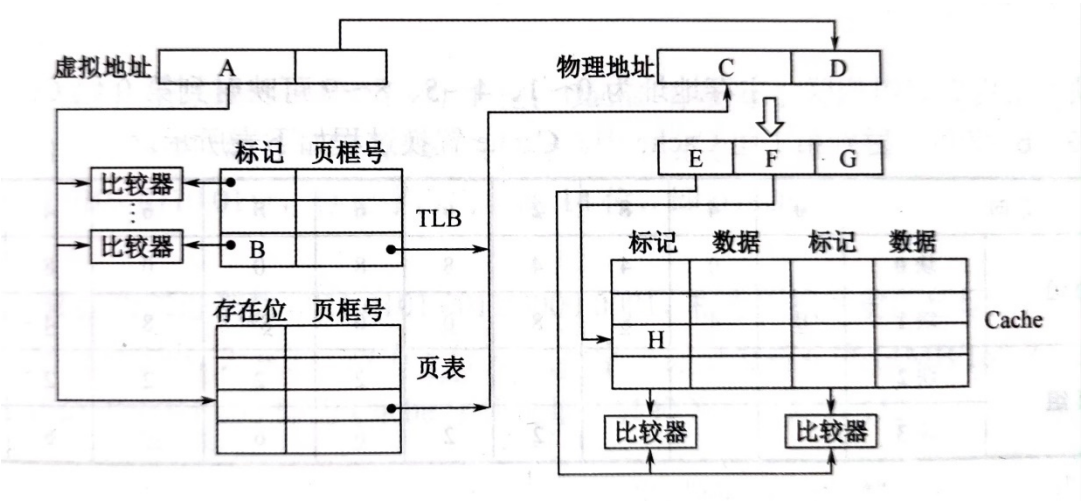
2. 掌握存储系统的工作流程。

二、实验内容

1. 任务描述

我这里采用了如下的参数设置：

仿真采用页式虚拟存储管理方式的三级存储系统。该存储系统按字节编址，虚拟地址为 24 位，物理地址为 16 位，页大小 4KB；TLB 采用全相联映射；Cache 数据区大小为 16KB，按 2 路组相联方式组织，主存大小为 64KB，主存和 CACHE 块大小为 16B。该系统的存储访问过程的示意图如下。



1）把奇数块号的模拟主存块每个字节都设为 0x55，把偶数块号的模拟主存块每个字节都设为 0xAA。打印块号为 28 和 29 中第一字节的内容。（第一题是这样的，但是后续的实验展示可能会展示主存中的数据都为随机数，这样的话展示起来会更加直观一些）

2）将块号为 2022 的模拟主存块装入模拟 Cache，打印按函数映射的 Cache 组号，打印其对应的 H 字段内容。

3）模拟某个缺页处理的过程，并给出缺页处理的日志记录。

4）模拟修改某个页面内容，模拟回写过程，并给出回写过程的记录。

2. 试验方案

仿真采用 Python 语言来仿真页式虚拟存储管理方式的三级存储系统。下面介绍仿真中主要用到的类：

首先是主存，class Memory，其中需要输入的参数分别为主存的大小，主存块的大小和主存页的大小，之后还有_data 为主存中全部的数据（这里是使用了问题 1 的设定，也就是 0x55 和 0xAA 交替设置；最后的_page_mapping 为主存中的慢表，慢表事实上是存放在内存中的，按道理应该存放在_data 之中，但是为了让代码更加清晰我把这个抽象的出来成为了一个对象，这样可以更加方便的查看。

```
class Memory:

    def __init__(self, size, block_size, page_size):

        self._size = size  # Memory size
        self._block_size = block_size  # Block size
        self._page_size = page_size  # Page size

        iter = util.odd_even_mix_byte(0x55, 0xAA, block_size)# If use mix data
        self._data = [next(iter) for i in range(size)]

        self._page_mapping = Page(self._size//self._page_size)
```

之后是高速缓存，class Cache，其中需要输入的参数分别为 Cache 的大小，Cache 块的大小和主存页的大小，首先是 Cache 替换的策略（LRU,LFU,FIFO），之后是我们设定的虚拟地址与物理地址位数，然后是存储的数据，这里我将每个块都有一个 class Line 来表示，这是因为我们除了要记录 Cache 中的数据，我们还需要记录每个块是否使用过，是否修改过，是否有效，还有它的标记，之后就是一些之前说过的大小设置以及计算出来的二进制 offset，下面首先是 class Line，之后是 class Cache：

```
class Line:
    def __init__(self, size):
        self.use = 0
        self.modified = 0
        self.valid = 0
        self.tag = 0
        self.data = [0] * size
```

```
class Cache:
    # Replacement policies
    LRU = "LRU"
    LFU = "LFU"
    FIFO = "FIFO"

    virtual_address_bit = 24
    physical_address_bit = 16

    def __init__(self, size, mem_size, block_size, mapping_pol, replace_pol,
                 write_pol):
        self._lines = [Line(block_size) for i in range(size // block_size)]

        self._mapping_pol = mapping_pol  # Mapping policy
        self._replace_pol = replace_pol  # Replacement policy
        self._write_pol = write_pol  # Write policy

        self._size = size  # Cache size
        self._mem_size = mem_size  # Memory size
        self._block_size = block_size  # Block size

        # Bit offset of cache line tag
        self._tag_shift = int(log(self._size // (self._mapping_pol*block_size), 2))
        # Bit offset of cache line set
        self._set_shift = int(log(self._block_size, 2))
        self.tlb = TLB()
```

3. 实验说明

这里我首先来进行一下理论的计算（类似理论课作业），这样后面可以对之后的仿真进行一些验证。

首先是 A-G 字段的位数，页大小为 4kb，页内偏移地址为 12 位，故 A = B = 24-12 = 12；D=12；物理地址一共为 16 位，所以 C = 16 - 12 = 4；2 路组相联，每组数据区容量有 16B×2=32B，共有 16KB/32B=512 组，故 F=9；E=16-G-F=16-4-9=3；主存块大小为 16B，故 G=4。因而 **A=12，B=12，C=4，D=12，E=3，F=9，G=4，H=3。**

如果将 2022 号的主存块装入 Cache，块号 2022 为 0111 1110 0110，因此映射的组号应为 1 1110 0110=486，对应的 H 字段为 011。

对于缺页处理这个部分的实验，我首先会处理一个不缺页的正常操作，之后是一个缺页的操作。

之后回写的实验，我会首先修改刚才不缺页正常操作的一个 byte，然后之后替换页，来触发回写操作。同时会记录命中与未命中记录最后展示。

三、实验记录

1. 实施步骤

对于**第一问**，我会进行如下步骤：

(1) 创建有着对应奇偶块数据的主存

(2) 展示一下主存中的数据已确保主存创建正确

(3) 打印块号为 28 和 29 的第一字节，完成题目要求

对于**第二问**，我会进行如下步骤：

(1) 创建 Cache

(2) 将第 2022 块从主存装入 Cache

(3) 打印 Cache 组号与 H 字段内容

这里选择了 2022 是相对于 29 来说我认为可能会更加直观。

对于**第三问**，我会进行如下步骤：

(1) 首先会取出一个有对应页(不缺页)的情况

(2) 之后会站是一个缺页的情况，并设置磁盘中这个页的数据为全 0xFF

对于**第四问**，我会进行如下步骤：

(3) 修改 cache 中一个字节的内容

(4) 进行回写过程

2. 实验结果

(1) 第一问

```
==================== Problem 1 ====================

FIRST 5 BLOCK FROM MEMORY:

00000: AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA
00016: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55
00032: AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA
00048: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55
00064: AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA

FIRST BYTE FROM BLOCK 28:

BLOCK 28: AA

FIRST BYTE FROM BLOCK 29:

BLOCK 29: 55
```

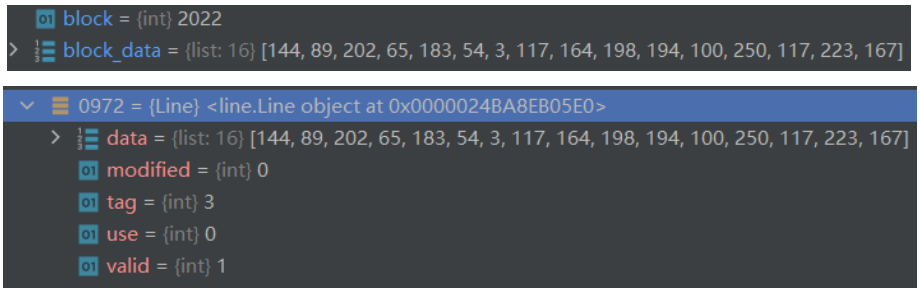
可以看到首先程序打印了主存中前面五个块的内容数据，可以看到符合题目要求的设置，之后的第 28 主存块与第 29 主存块分别为 AA 和 55 也是符合题目的要求。之后的展示都**使用了随机数据**，这样更好看一些。

(2) 第二问

```
==================== Problem 2 ====================

Corresponding Cache Block Number:  486
Result of H:  011
```

对于 2022 块，它对应的 Cache 中的块是 486，对应的 H 字段标记为 011，我们可以在调试模式中看到 Cache 中对应的数据来确认编程的正确：



可以看到 memory 中的 block_data 是完全对应 972 行 Line 中的 data 的，同时 tag 为 3，也就是 011，972 行是因为 Cache 为 2 路组相联，所以一组有两个 Line。

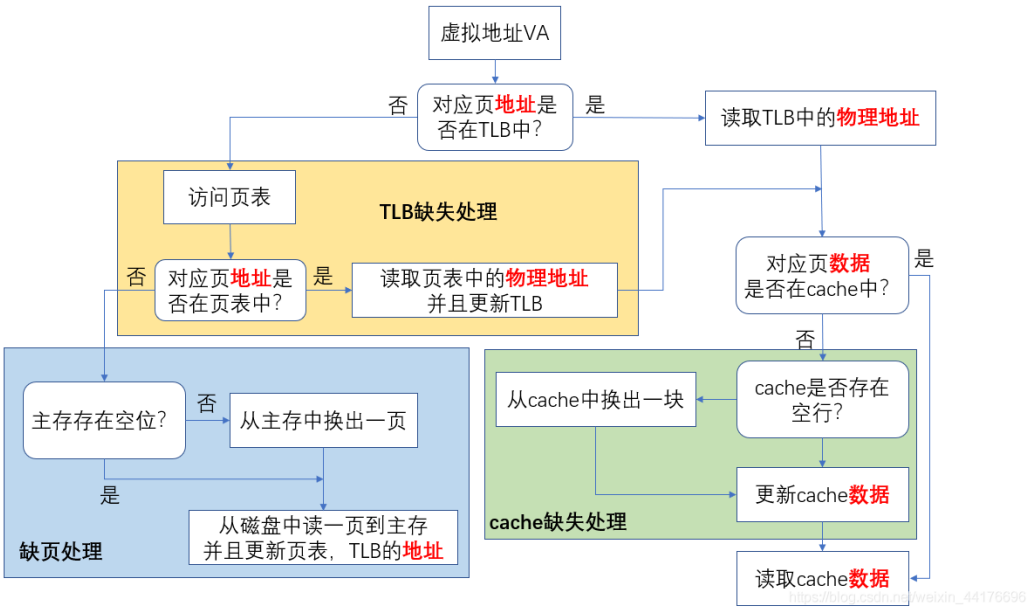
(3) 第三问

```
==================== Problem 3 ====================

Scenario 1: Find Page Successfully
Fail to find page number in TLB
The Corresponding Page in Cache is:  1

Scenario 2: Fail to find Page
Fail to find page number in TLB
Fail to find page in memory
Loading from disk
```

之前说了我们有两个情景，一个是正确找到页了，一个是没有正确找到页的情景，我的整体代码基于以下的流程实现：



可以看到第一个情景，首先没有在 TLB 中找到对应的页地址，这是由于一开始 TLB 是空的，之后访问主存中的页表，之后在页表中找到了对应的页地址（这个是一开始指定的），之后会读取页表中的物理地址并且更新 TLB，之后就是取数据的操作了，然后会输出 Cache 中对应的页号。

第二个情景中，首先一样 TLB 中没有找到对应的地址，之后访问主存中的页表，这时找到的页表是无效的，所以进入缺页处理的阶段，然后这里是将磁盘中的页假设为 0xFF，之后会将数据存放到主存中，并且更新页表和 TLB。

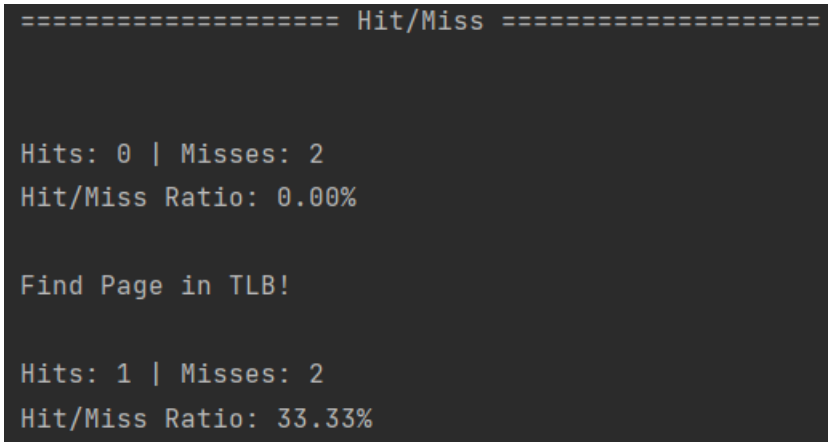
(4) 第四问

```
==================== Problem 4 ====================

Now We need to replace the modified data block.
[255, 115, 63, 212, 163, 29, 103, 173, 39, 53, 125, 32, 170, 2, 193, 184]
```

将刚才成功找到页放入 Cache 的第一个 byte 设置为 0xFF，之后使用全部为 0xFF 的来替换来触发回写，这里的话打印出了经过修改的 block，可以看到第一个 byte 改为了 0xFF。

(5) 课后问题（命中与未命中的仿真）



这里首先我是先打印出命中与未命中的数量，因为一开始 Cache 是空的，所以全部都是 miss 的，之后我又重新的找刚才正确找到页的那个情景的代码，然后在进行统计，可以看到这里成功命中了。

四、总结与讨论

这次的实验还是花费了比较多的时间，主要还是需要深刻的理解整个储存系统的工作流程，在对整个系统的工作流程熟悉之后，将思路整理后，编程的过程就比较轻松了，整体我没有使用比较复杂的数据结构，主存中储存数据就是以列表的形式来储存的，包括 Cache 的数据由于需要统计所以单独创建了一个类，但是具体的数据还是 list，基本上页与块是计算出来的，并不是一个单独的数据结构，包括页表应该储存在主存当中，但是为了简便一些我把它直接当作了主存的一个对象，当然还有许多功能没有增加，比如页面置换的相关算法，与磁盘的交互都是设定好的，但是整个的仿真我认为是可以体现整个存储系统的大致工作流程的，我也更加深刻的理解了储存系统的工作流程，一开始会觉得比较复杂，但是了解之后会觉得其实还是很符合直觉的，最后回答一下课后问题：

储存系统为什么要设置为三级结构？

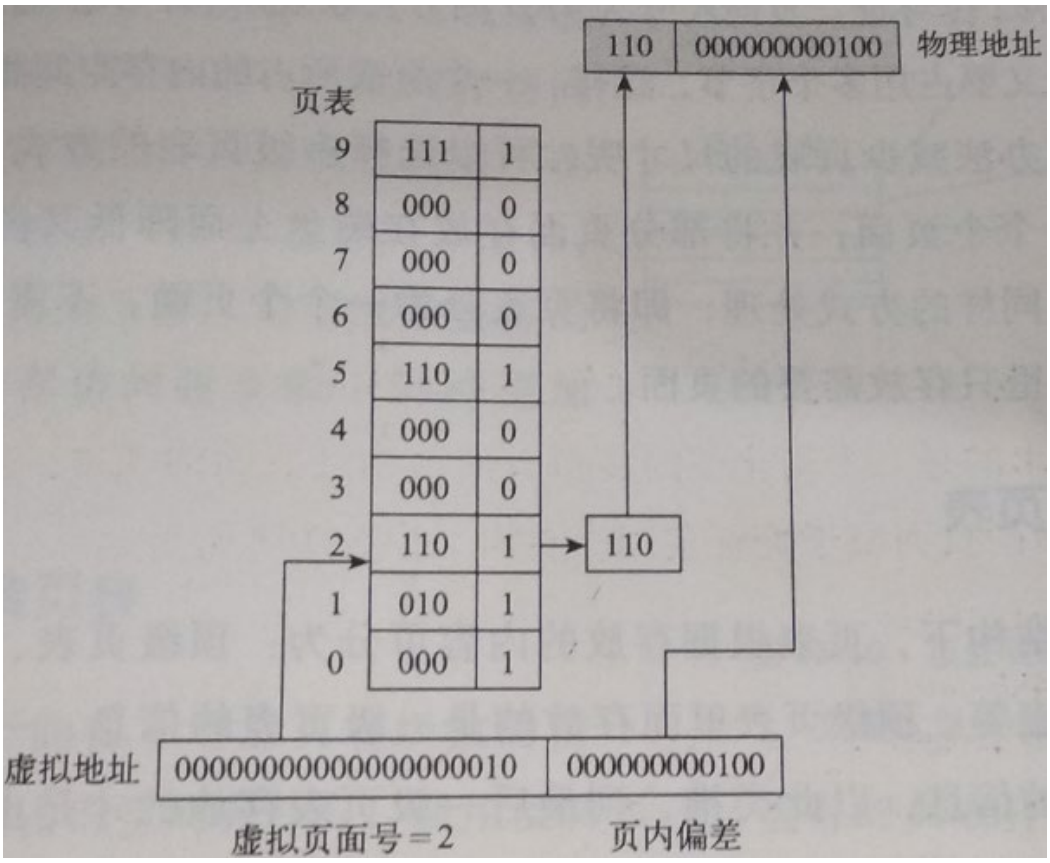
这应该是一个在成本和性能比较权衡之后的结果，事实上现在真正的 Cache 中也有三级的结构，辅助存储器也分为很多种类，如果可以直接使用与主存同样大的高速缓存，那理论上确实不需要分级，但是这样成本会过高，不能接受，而分太多级也肯定会使系统过于复杂，同时需要更多的通信反而可能会导致性能下降，分为三级应该既可以充分利用低速度存储设备容量比较大的优势，也可以充分发挥高速存储设备的速度优势。

高速缓存命中和未命中如何仿真？

这次的实验中命中与未命中分别设为了两个全局变量 hit 和 miss，使用函数 report 可以查看本次实验中命中的次数和未命中的次数以及命中率，可以在前面的实验结果查看。

虚拟存储的地址怎么转化为实际地址？

虚拟地址与物理地址的转换需要依靠页表，可以通过这张图片清晰的看出如何转换：



可以看到页表保存了三个信息，页表的索引是虚拟页面号，图片中第一列为物理页面号，第二列为是否有效，0 表示无效，1 表示有效。页内偏差是一样的。