天津大学

计算机系统课程实验

选择题目：第九章—虚拟内存

学生姓名 王雨朦

学生学号 2016229082

学院名称 国际工程师学院

专业 计算机

时间 2017/11/17

目录

一、 实验条件

机器型号	**Acer Aspire E1-471G**	
内存大小	10GB	
系统类型	64 位操作系统	
SSD	Sumsung 256GB	
CPU	Intel® Core™ i5-3210M CPU @2.50GHz 2.50GHz	
高速缓存	L1	2*32KB，8 路
	L2	2*256KB，8 路
	L3	3.0MB，12 路
操作系统	VM 虚拟机 Ubuntu 16.04TLS 64 位	

二、 大实验

实验一：设计并实现一个虚拟存储器，包括 cache、M，TLB，VM

A. 实验描述

根据学到的虚拟内存的机制，通过设计并实现一个虚拟存储器，检验对存储机制理解的程度，并深入理解虚拟内存、高速缓存等相互作用原理。

虚拟存储器的作用方式：当 CPU 需要读取数据时，会发送一个虚拟地址给 MMU，MMU 拿着虚拟地址去 TLB 中查找 PTE，如果找到了，将虚拟地址翻译为物理地址，并去高速缓存和内存中请求数据；如果未找到，再去页表中找，找到后更新 TLB，将虚拟地址翻译成物理地址，去高速缓存和内存中请求数据；如果再页表中还没找到，就启动缺页中断，缺页就需要进行页面置换，被置换掉的页面如果是脏页，需要写回磁盘。页面置换完成后 CPU 会重新发送物理地址进行请求。

我们的思路是要设计符合上述虚拟存储器工作过程和原理的仿真虚拟存储器，这个虚拟存储器的程序是在 linux 环境下执行的，程序设计如下：

Memory:模拟内存，定义了逻辑地址空间 16 位和物理地址空间 14 位，低八位都为偏移量。用二维数组模拟物理内存。

TLB:用 map 和 list 模拟 TLB，map 记录映射关系，list 记录 PTE 顺序，方便 TLB 更新，更新采用 LRU 算法。TLB 主要功能是判断 PTE 是否存在和返回 PTE；

PageTable.:用 map, set 和 list 模拟页表，map 记录映射关系，list 记录 PTE 顺序，set 记录空闲 PTE。页表负责存放逻辑地址和物理地址的对应关系。

BackingStore:对 BACKING_STORE.bin 模拟的磁盘进行读写操作。

Main:主函数，模拟虚拟存储器的工作过程。

B. 实验结果

```
calmon@ubuntu:~/9.1$ make
g++ -g -Wall -std=c++11 -c TLB.cpp
g++ -g -Wall -std=c++11 -c PageTable.cpp
g++ -g -Wall -std=c++11 -c BackingStore.cpp
g++ -g -Wall -std=c++11 -c main.cpp
g++ -g -Wall -std=c++11 -o a.out TLB.o PageTable.o BackingStore.o main.
o
calmon@ubuntu:~/9.1$ ./a.out -F sample_input.txt
4 0 N F 0
8 0 N F 256
12 0 N F 512
16 0 N F 768
4 1 H N 1
8 1 H N 257
16 1 H N 769
```

```
Page Table:
0 36
2 34
3 43
9 28
11 56
14 50
19 0
20 5
23 47
24 41
26 38
文本编辑器
39 9
42 59
47 20
61 57
69 49
71 19
75 2
83 46
86 35
87 55
88 21
95 16
```

```
196 1
199 8
209 13
214 24
215 30
221 63
226 22
230 4
231 51
234 52
236 40
244 12
250 27
251 29
252 37
253 18


76 Page Faults
calmon@ubuntu:~/9.1$ ./a.out -L sample_input.txt
4 0 N F 0
8 0 N F 256
12 0 N F 512
16 0 N F 768
4 1 H N 1
8 1 H N 257
```

C. 实验分析

输入	.txt 文件存储的一系列读写操作： R 读+虚拟地址 或 W 写+虚拟地址+写入的值

输出	虚拟存储器执行每条读写操作的过程和结果： 页号+偏移量+TLB 是否命中（命中 H，不命中 N）+是否缺页（缺页 F，不缺页 N） +物理地址+值（读操作的值） 最后一条命令执行完的页表 缺页总数

如实验结果所示，程序模拟了虚拟存储器的工作过程，对于输入的每条虚拟地址，翻译为物理地址，并分析出页号和偏移量等量，从而去一项一项的检查 TLB 是否命中，是否缺页，直至最后成功进行读写操作。

实验结果统计了缺页数，在此基础上可以进一步进行页面置换算法实验实验，设置不同的页面置换算法，分析不同的结果，从而理解不同页面置换算法的性能。

实验二：设计实验分析 TLB 对顺序读的影响

A. 实验描述

在《存储器-高速缓存》文件中，图 3-12 给出了数据存储的两种不同方式：一种是一行存一个数据；另一种是一页存一个数据，以此来探究 TLB 对顺序读的影响。顺序读的过程是在单循环列表中实现的，单循环列表每个结点的结构体：

```
struct li {
        struct li *next;
        long long pad[NPAD];
};
```

我们要做的是编写代码来验证 TLB 对顺序读的影响，也就是在两种存储方式下，设定不同大小的工作集，每个工作集中循环遍历单循环链表，计算出 CPU 所需要的周期数，从而分析对比在每个工作集中，两种方式下所需周期数的差别，最后理解 TLB 的作用。不同的存储方式在本实验中是由单链表的每个结点所占空间控制的，循环遍历的时候是挨个读取每个结点。实际上按行存储结点就是大部分情况下 TLB 命中；而按页存储结点时，每次读数据都可能会 TLB 不命中，需要去页表中查找。本实验对比的是这两种极端情况下的 CPU 耗时，以此来体现出能够合理的利用高速缓存机制是多么的重要。

经过查看，本实验的环境下，高速缓存每行存 64 字节，每页 4096 字节。

```
calmon@ubuntu:~$ getconf PAGE_SIZE
4096
```

在 64 位系统上，数据结构中 long long 类型占八个字节，所以每行存 64 字节，若要按行存我们需要设置结构体中的 NPAD=7，让结构体 l 占满整行；而若要让每页 4096 字节占满，应该设置 NPAD=511。

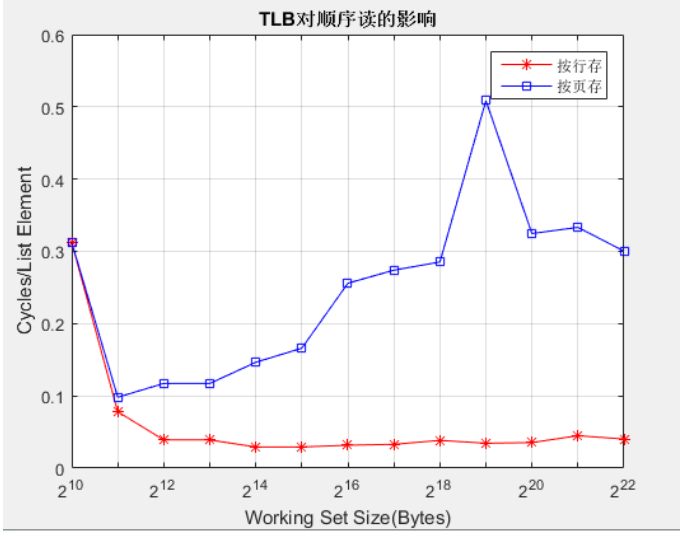
实验过程为：在每个单链表结点大小的情况下，构造不同大小的工作集，每个工作集情况下，循环读取单链表一次，记录循环遍历单链表的时间，再根据 CPU 频率计算得到 CPU 周期，从而计算出遍历需要的 CPU 周期的数量，即 CPU 耗时。最后比较结点行/页存储方式下，在每个工作集中，循环遍历的 CPU 耗时。

B. 实验数据

```
calmon@ubuntu:~/tlb$ make
gcc -c TLB-SR.c
gcc TLB-SR.o -o TLB-SR
calmon@ubuntu:~/tlb$ ./TLB-SR
When NPAD= 7, the total time is:
0.3125000000
0.0781250000
0.0390625000
0.0195312500
0.0390625000
0.0341796875
0.0341796875
0.0354003906
0.0585937500
0.0372314453
0.0427246094
0.0444793701
0.0286483765
0.0367164612
0.0319385529
```

```
calmon@ubuntu:~/tlb$ make
gcc -c TLB-SR.c
gcc TLB-SR.o -o TLB-SR
calmon@ubuntu:~/tlb$ ./TLB-SR
When NPAD= 511, the total time is:
0.3125000000
0.0781250000
0.1171875000
0.1171875000
0.1464843750
0.1660156250
0.3247070312
0.3332519531
0.3002929688
0.6085205078
0.2558898926
0.2739715576
0.2853012085
```

C. 实验结果



D. 实验分析

如图中，实验结果和 3.12 略有不同，但是不影响实验结果。图中可以看出，在每个工作集情况下，按行存储数据时 TLB 的读取时间都小于按页读，因为按页读大部分情况下 TLB 会不命中，是最不利于 TLB 工作的一种存储方式。而且，随着工作集的增大，按行读的优势就更加明显。

与原实验结果不同的地方，在按页存储的时候，在工作集为2^{16}和2^{19}时，耗时有明显增长。原因是我的实验环境的一级缓存大小为 64KB，二级缓存大小为 512KB。所以在使用到不同的高速缓存区时，运行速度是不同的，高速缓存的运行速度是按照级数增加而有所变慢。

TLB 是一个小的类似于高速缓存中页表的机制，它用来加速地址翻译。如果 TLB 命中，就可以直接进行地址翻译；不命中的话才去页表中查找更新。通过这次实验，理解到 TLB 按顺序读的工作原理，对以后合理利用硬件环境改善程序性能有新的体会。

实验三： Malloc 实验

A. 实验要求

独立完成课后题 9.20Malloc 实验；并针对视频操作来实现，要求视频帧行大小为大于 2048，列>行。读取并保存十分钟以内的视频。

B. 实验描述

在使用 C 语言时，知道了动态内存分配的概念，也常常用到 malloc()，但是通过此实验实现自己的 malloc()和 free()方法可以帮助我们深入理解动态内存分配的实现原理。

在 LINUX 系统下程序的堆是通过内存块进行管理的，即将堆分成了很多大小不一的内存块。而管理这些内存块需要用到链表，详细的数据结构如下:

```
struct li{
    size_t size;  //每一块数据区的大小
    int free; //是否为空的标志位
    struct li *pre;
    struct li *next;
}*L;
```

整个实验过程是输入一段视频，用 malloc 分配空间将该视频进行存储,将存储的视频先输出，然后 free 掉。

malloc 在分配空间时，需要考虑两种情况：

1. 链表为空，即第一次分配空间，需要分配一个新的块作为链表头结点。
2. 链表非空，即以后每次分配时，首先在链表中查找空闲块，找到后判断是否可以对其分割，以保证不会浪费；如果没找到，需要分配新块，并加入链表。

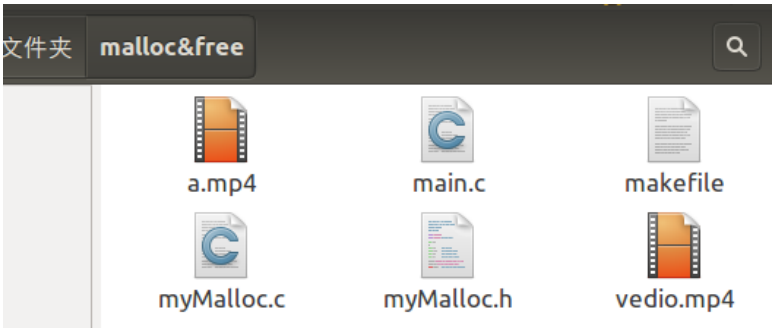
free 在释放时，先将该块内存释放，再判断其是否可以和前、后块进行内存合并，合并可增加内存的利用率，避免产生太多的碎块。

C. 实验数据

实验数据没有找到 2560*2048 大小的视频，选取了韩剧《任意依恋》第 15 集中的一段，这段数据应该对实验结果影响不大，视频各参数如下：

D. 实验结果

如下图，vedio.mp4 为选取的原视频，而 a.mp4 为通过分配空间复制到的视频。



E. 实验分析

经检查，复制到视频 a.mp4 与原视频 vedio.mp4 几乎没有差别,验证了自己写的 malloc 方法的性能是没有问题的。通过实验深入理解了 malloc 和 free 两个方

法的工作原理和内部工作机制。

三、 小实验

练习题 9.5：mmap 实验

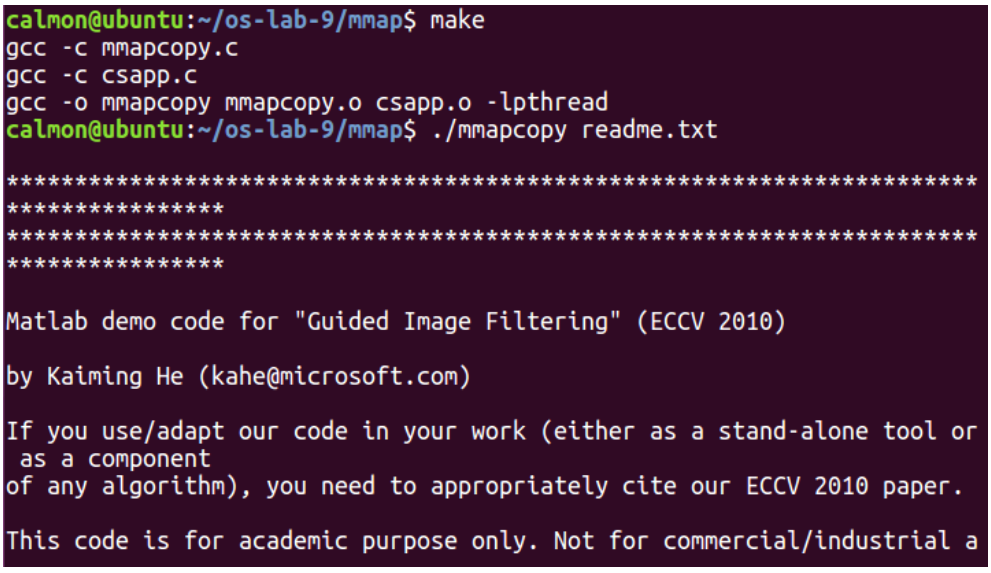
A. 实验内容

这个实验要求我们编写一个 mmapcopy.c，使用 mmap 将一个任意大小的磁盘文件复制到 stdout。输入文件的名字必须作为一个命令行参数来传递。

实验主要目的是使用 mmap 函数。mmap 能将一个文件或其他对象映射进内存，系统调用 mmap()主要用于共享内存。然而，mmap 系统调用并不是完全为了用于共享内存而设计的，它本身提供了不同于一般对普通文件的访问方式，进程可以像读写内存一样对普通文件进行操作。

实验过程就是要编写代码，将文件映射到内存中，然后再用 Write 函数将内存显示在终端上。

B. 小实验结果



C. 小实验分析

如图中所示，我们从内存中得到的数据内容写在终端上的结果，这个结果与文件中的内容一致。可以确定 mmap 函数已经将文件映射到了内存中。

通过实验进一步理解了 mmap 函数的作用，但其具体有效的应用还待以后进行学习。