**实验六：存储管理报告（阐述设计程序的思路）**

**上机作业51题:**

首先给出代码部分，再依次说明构造思想：

图形用户界面, 文本, 应用程序

描述已自动生成

图形用户界面, 文本, 应用程序

描述已自动生成

图形用户界面, 文本, 应用程序

描述已自动生成

51题题目要求：编写一个程序，它使用老化算法模拟一个分页系统。页框的数量是一个参数。页面访问序列从文件中读取。对于一个给定的输入文件，列出每1000个内存访问中发生缺页中断的数目它是可用页框数的函数。

分析题目要求：主体思想是实现老化算法，用户需要输入的参数只有一个—页框数量。同时程序还要从文件中读取页面号码，最终输出的结果是每1000次内存访问中发生缺页中断的次数。

我的实现：首先定义了宏INTERVAL用来表示间隔1000（每1000次遍历就要输出结果），全局变量cnt表示缺页中断次数（每1000次记得要清0），frame\_number表示用户输入的页框数量。

接着是main函数：首先实现文件读取，我是在网上找工具随机生成了一定范围内的5000个数（范围分别有1-600，1-700和1-800三个样例，这样可以比较不同的测试结果）。定义文件读取名fp，存取文件中数据的数组arr，先打开文件（我的命名都是data\_5000\_1\_\*\*\*.txt）并记录在fp中，别忘了需要判断是否打开成功（fp是否不等于NULL），然后就是依次将文件中的数据搬运到数组arr中，最后也不要忘了关闭文件fp，至此前期保存页面访问序列的准备工作就做完了。

接着是打印一行提示让用户输入页框数量frame\_number,同时方便起见我也将它输出了出来并提示用户开始遍历页面。接着就是定义最重要的几个参数：frame\_in\_memory用来记录每个位置上的页框编号，-1表示还未使用；frame\_cnt表示每个位置上的这个页框的参数值；frame\_left表示当前还剩多少个页框未使用。注意frame\_cnt的类型是unsigned int型，转换成二进制就是16位，长度适中，满足老化算法的要求。初始化frame\_in\_memory为-1，frame\_cnt为0，frame\_left为frame\_number。

接着就是遍历了：在每1000次循环中，首先就是把每个位置上的frame\_cnt值减半(为0减半也没有影响，主要是对有值的数减半)。然后定义想要访问的页面号page\_to\_visit，期望访问的页面号的地址pos\_page\_to\_visit，首先是判断是否发生缺页，为此，遍历frame\_in\_memory数组,比较每一个位置上的值是否与page\_to\_visit相等，若相等则未缺页，否则进行缺页部分的程序段。

先说不缺页的程序段吧：根据老化算法，只需要将访问页面的计数值最高位置为1即可（注意它的最高位一定是0，因为进入循环时就对每一个计数值进行了减半操作），所以给这个页面位置的frame\_cnt加上2<<15即可。

再介绍缺页如何处理：缺页首先将缺页中断计数器cnt加1，然后判断是否还有空余位置可以使用（这样就不用淘汰某个页面），即看frame\_left是否不为0。若不为0，则找一个空余位置分配给访问的页面，然后要记得同时把它的frame\_cnt值置为2<<15并且break跳出for循环（当时debug了好久才发现了这一点小错误，虽然看起来无伤大雅，但有很大的影响！）也要让frame\_left减减；若为0，则找出当前frame\_cnt计数值最小的页面的位置,把正在访问的页面放到这个位置上，并且置frame\_cnt值为2<<15。

最后时展示其效果图，3张图片分别对应了范围为1-600，1-700和1-800的访问序列：

文本

低可信度描述已自动生成

图片包含 文本

描述已自动生成

文本

低可信度描述已自动生成

**上机作业53题:**

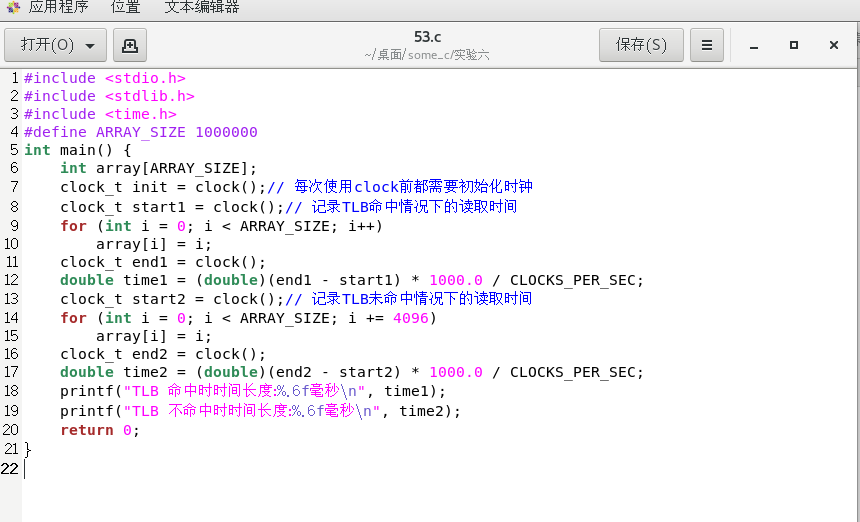
53题要求：编写一个程序，说明TLB未命中对有效内存访问时间的影响，内存访问时间可以通过计算每次遍历大数组时的读取时间来衡量。(a)解释编程思想，并描述所期望输出如何展示一些实际的虚拟内存体系结构。(b) 运行该程序，并解释运行结果与你的预期有何出入。

c)在一台更古老的且有着不同体系结构的计算机上重复b，并解释输出上的主要区别

注：小问c暂时没有能力完成，因为无法找到一台古老的（例如还运行着Windows xp系统）且有不同体系结构的计算机，但前两问还是可以做的。

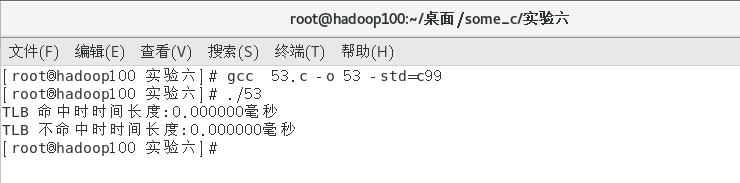
这里我们需要用程序—即软件的方式展示TLB未命中对有效内存访问时间的影响，而题目说内存访问时间可以通过计算每次遍历大数组时的读取时间来衡量。回想学习程序语言时的知识可知，数组是存储在一块连续的物理磁盘上的，本身就具有很好的局部性，因此再顺序遍历的话就会将局部性展现得淋漓尽致。而如果跳跃式访问数组中元素（例如二维数组中按列访问），则每次访问元素间隔很大，程序不会进行“提前读”的操作。

基于上述想法，我设计了如下程序段：



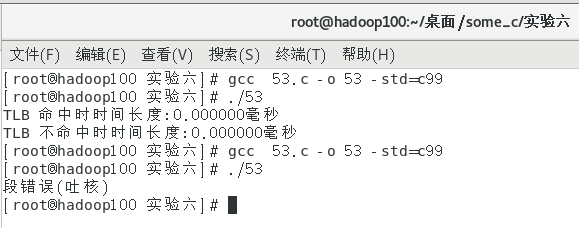
可以看到主题思想就是计算顺序遍历数组和跳跃遍历数组的时间长度，分别对应TLB命中和不命中的情况。注意每次使用clock函数前都需要初始化时钟。

程序运行结果如下：



很奇怪的是，这跟我们设想的不太一样，大概也是小问b想了解的。我能给出的合理解释就是现代计算机运行速度已经比较快了，因此对它们来说遍历1000000个数已经不会消耗太多时间，所以输出结果为0.000000毫秒。

有趣的是，自己试着将数组的大小改成10的7次方，结果发生了以下错误：



因此我没有再改动数组的大小，这里的段错误大概就是内存溢出了，可能数组空间开到1e6就快满了，我也没有再测试其他的样例，但没有得到期望的结果无疑是一种遗憾。