

订阅DeepL Pro以编辑此演示文稿。  
访问[www.DeepL.com/pro](https://www.deepl.com/pro?cta=edit-document)，了解更多信息。

LAB #4 - mdadm线性设备（缓存） CMPSC311 - 系统编程入门 2022年春季 - 朱森村教授和Suman Saha教授

**截止日期：美国东部时间2022年4月1日（晚上11:59）。**

就像这门课的所有实验作业一样，你被禁止从互联网上复制任何内容，包括（discord或GroupMe或其他群组消息应用程序）或讨论、分享想法、代码、配置、文本或其他任何东西，或从课内或课外的任何人那里获得帮助。咨询网上资料是可以接受的，但在任何情况下都不能抄袭。不遵守这一要求将导致我们的课程大纲中所述的处罚。

你刚刚完成了mdadm的实施，它正在工作。计划在你的存储系统上建立安全加密钱包的软件工程师一直在折磨你的存储系统，向它投掷各种I/O模式，而他们无法在你的实现中找到任何不一致的地方。这很好，因为现在你有一个工作系统，尽管它可能不是高性能的。正如斯坦福大学的John Ousterhout教授所说，"最好的性能改进是从非工作状态到工作状态的过渡"。软件工程师很高兴你的存储系统能够正常工作，但现在他们希望你能让它也变得快速。为此，你将在mdadm中实现一个块缓存。缓存是书中最古老的技巧之一，通过将经常使用的数据保存在比主存储介质更快的（更小的）存储介质中来减少请求延迟。由于我们在课堂上广泛地介绍了缓存，因此在本文中我们将跳过其细节。你必须观看讲座，以了解什么是缓存，以及最近最少的数据是如何被保存的。

你将在这份作业中实现的 "已用（LRU）"算法的工作原理。

# 概述

一般来说，缓存在快速存储介质中存储*键和值*对。例如，在CPU缓存中，键是内存地址，而值是存在于该地址的数据。当CPU想访问某个内存地址的数据时，它首先检查该地址是否作为键出现在缓存中；如果是，CPU就直接从缓存中读取相应的数据，而不去读取内存，因为从内存读取数据的速度很慢。

在浏览器的缓存中，键是图像的URL，而值是图像文件。当你访问一个网站时，浏览器从网络服务器上获取HTML文件，解析HTML文件并找到网页上出现的图像的URL。在再次从网络服务器检索图像之前，它首先检查其缓存，看URL是否作为一个键出现在缓存中，如果是，浏览器就从本地磁盘读取图像，这比通过网络从网络服务器读取图像快得多。

在这项任务中，你将为mdadm实现一个块缓存。在mdadm的情况下，键将是由磁盘号和块号组成的元组，用于识别JBOD中的特定块，而值将是块的内容。当mdadm系统的用户发出mdadm\_read调用时，你的mdadm\_read的实现将首先查看用户指定的地址所对应的块是否在缓存中，如果是，那么该块将从缓存中被复制，而不用向JBOD发出缓慢的JBOD\_READ\_BLOCK调用。如果该块不在缓存中，那么就从JBOD中读出，并插入到缓存中，这样，如果用户再次请求该块，就可以更快地从缓存中提供。

# 缓存实施

通常情况下，缓存是存储系统的一个组成部分，存储系统的用户是无法访问它的。然而，为了方便测试，在这个作业中，我们将把缓存作为一个单独的模块来实现，然后把它集成到mdadm\_read和mdadm\_write的调用中。

请看一下cache.h文件。你的缓存中的每个条目都是以下结构。

typedef struct { bool valid; int disk\_num; int block\_num;

uint8\_t block[JBOD\_BLOCK\_SIZE]; int access\_time。

} cache\_entry\_t;

valid字段表示该缓存条目是否有效。disk\_num和block\_num字段标识了该缓存条目持有的块，block字段持有相应块的数据。access\_time字段存储了该缓存元素最后被访问的时间--无论是写入还是读取。

文件cache.c包含以下预定义变量。

static cache\_entry\_t \*cache = NULL; static int cache\_size = 0;

静态 int clock = 0；静态 int num\_queries = 0；静态 int num\_hits = 0。

现在我们来看看cache.h中声明的函数，你将实现这些函数，并描述上述变量与这些函数的关系。你必须查看cache.h以了解每个函数的更多信息。

1. int cache\_create(int num\_entries); 动态分配num\_entries缓存条目的空间，并应在cache全局变量中存储创建的缓存地址。num\_-条目参数最小可以是2，最大可以是4096。它还应该将cache\_size设置为num\_entries，因为这描述了缓存的大小，也会被其他函数使用。一旦缓存被创建，cache\_size是固定的。你可以把它看作是缓冲区的最大容量。因此，为了简单起见，你应该把它实现为一个大小为cache\_size的数组，而不是一个链接列表，尽管后者允许动态地添加或删除缓存条目。在没有调用cache\_destroy（见下文）的情况下，调用这个函数两次应该会失败。
2. int cache\_destroy(void); 释放动态分配的缓存空间，并应将缓存设置为NULL，cache\_size为0。在没有调用cache\_- create的情况下，调用这个函数两次应该会失败。
3. int cache\_lookup(int disk\_num, int block\_num, uint8\_t \*buf); 在缓存中查找由disk\_num和block\_num标识的块。如果找到，将该块复制到buf中，buf不能为空。这个函数在每次执行查找时必须增加num\_queries全局变量。如果查找成功，这个函数也应该增加num\_hits全局变量；它还应该增加时钟变量，并将其分配给相应条目的access\_time字段，以表明该条目最近被使用过。我们将使用num\_queries和num\_hits变量来计算缓存的命中率。
4. int cache\_insert(int disk\_num, int block\_num, uint8\_t \*buf); 将disk\_num和block\_num标识的块插入缓存，并将buf（不能为空）复制到相应的缓存条目。插入不应该失败：如果缓存已经满了，那么应该根据LRU策略用这个插入操作的数据覆盖一个条目。这个函数还应该递增并将时钟变量分配给新插入的条目的访问时间。
5. void cache\_update(int disk\_num, int block\_num, const uint8\_t \*buf); 如果缓存中存在条目，用buf中的新数据更新其块内容。如果成功的话，还应该更新access\_time。
6. bool cache\_enabled(void); 如果缓存被启用，返回true（cache\_size大于最小值2）。当把缓存集成到你的mdadm\_read和mdadm\_write函数中时，这将很有用。也就是说，在你的mdadm函数中，只要可能涉及到缓存，你就应该首先调用这个函数。

# 实施战略

测试器现在包括对你的缓存实现的新测试。你应该首先致力于实现cache.c中的函数，并通过所有测试器的单元测试。一旦你通过了测试，你应该把你的缓存纳入你的mdadm\_read和mdadm\_write函数中--你需要在mdadm\_write中也实现缓存，因为我们要使用写过的缓存策略，正如在类中所描述的。一旦你这样做了，请确保你仍然能通过所有的测试。

接下来，在跟踪文件上试试你的实现，看看是否能提高性能。为了评估性能，我们在JBOD中引入了一个新的成本是衡量你的缓存有效性的指标，它是根据执行的操作数来计算的。每个JBOD操作都有不同的成本，通过有效的缓存，你可以减少读操作的数量，从而降低你的成本。现在，测试器在与工作负载文件一起使用时，也会取一个缓存大小，并在最后打印出成本和命中率。成本是由JBOD内部计算的，而命中率是由cache.c中的cache\_print\_hit\_rate函数打印的。它打印的值是基于num\_queries和num\_hits变量，你应该增加这些变量。

下面是参考实现的结果。你的实现可能会产生

不同的成本和命中率值，取决于你如何实现它（优化或不优化）。你不需要输出相同的值，但它们应该与所给的值处于同一量级。首先，我们在随机输入文件上运行测试器。

$ ./tester -w traces/random-input >x Cost: 18948700

命中率: -nan%

是18948700，而命中率是未定义的，因为我们没有启用缓存。接下来，我们重新运行测试器，并使用-s选项，指定缓存大小为1024条。

$ ./tester -w traces/random-input -s 1024 >x Cost: 17669400

命中率：24.5

正如你所看到的，鉴于我们有非零的命中率，缓存是有效的，因此，现在的成本已经降低。让我们再试一次最大的缓冲区大小。

$ ./tester -w traces/random-input -s 4096 >x Cost: 13091800

命中率。87.9%

$ diff x traces/random-expected-output

$

再一次，我们使用更大的缓存大大降低了成本。我们还通过比较输出结果来确保引入缓存不违反正确性。**如果引入缓存违反了你的mdadm实现的正确性，你将在相应的跟踪文件中得到一个零分。**

# 分级

**评分标准** 评分将根据以下评分标准进行。

* 通过缓存测试案例（总共19个案例，但由于大部分是lab2和lab3的读/写测试案例，我们只计算2个写案例和6个缓存案例。满分是10分）：70%。
* 用缓存传递随机和线性跟踪文件，以减少成本：15%。
* 添加有意义的描述性评论。5%
* 成功的 "制作 "和执行，没有错误和**警告**。5%
* 提交提交ID: 5%

**处罚。**逾期提交的每天10%（最多3天）。如果逾期超过3天，实验作业将不被评分。