# 虚拟内存技术和模拟虚拟内存实现论文综述

**引言**

虚拟内存技术作为计算机系统中的关键组成部分，是现代计算机系统中不可或缺的重要功能之一。随着计算机科技的不断发展，虚拟内存技术在操作系统和计算机硬件层面的演进与创新，为计算机系统的性能提升和资源管理提供了重要支持。在计算机系统中，虚拟内存技术允许程序访问超出物理内存容量范围的数据和指令。它通过将物理内存与磁盘空间相结合，创建了一个抽象的地址空间，为每个运行的程序提供了一种似乎是无限的内存空间。这种技术使得多个程序可以同时在计算机上运行，而无需担心物理内存的限制，从而提高了系统的整体效率和可用性。虚拟内存技术的实现涉及到诸多复杂的机制，其中包括页面置换、页面错误处理、地址映射以及页面分配等。这些机制的优化和改进直接影响着计算机系统的性能表现。例如，页面置换算法的选择将影响到程序访问数据的速度和效率，而地址映射机制的设计则直接关系到程序的内存访问方式和资源利用率。虚拟内存技术的重要性随着计算机应用的日益复杂和多样化也愈发显现。它不仅可以提高系统的整体性能，还能够改善系统的稳定性和可靠性，确保在资源有限的情况下，多个程序能够高效地共享计算机资源。

本论文综述将探讨虚拟内存技术的原理、机制和实现方式，并且对多篇模拟虚拟内存实现的论文进行整合与归纳。我将深入分析不同的模拟虚拟内存实现的技术，并探讨它们对计算机系统性能的影响。此外，我还将讨论虚拟内存技术在现代计算机系统中的应用和发展趋势，以及未来可能面临的挑战和解决方案。

**国内外研究现状**

虚拟存储管理技术是现代多任务操作系统内存管理的核心管理技术。虚拟存储管理技术使每个进程拥有自己独立且连续的线性地址空间。而事实上，进程的物理页通常是离散地存储在物理内存中，还有一些物理页被换出到磁盘上，等到需要时再换入内存。虚拟存储管理技术从逻辑上对内存容量进行了扩充，并对每个进程的线性地址空间进行了保护。在这种技术的支持下，复杂系统程序的编写变的不再困难。对真正的物理内存DRAM（Dynamic Random Access Memory，动态随机存取存储器）的使用也更有效率并且更加安全。

虚拟存储器的概念来源于英国 ATLAS 计算机。该计算机的一级存储器是磁芯存储器，内存大小为 16000 个字。其 CPU 利用 20 位的线性地址进行内存寻址。在 20 世纪 50 年代，内存的部署是通过磁核来完成的，而外存的部署则是通过磁鼓旋转来实现的。那时，程序员在编写代码时，需要努力使程序需要的内存空间不会超过内存大小。在些时候，程序员必须使用一种策略将内存中的数据换出到外存，从而腾出一部分内存空间来处理别的程序数据。20 世纪 60 年代，曼彻斯特大学的阿特拉斯团队研究出了一种策略，解决了程序员关于内外存数据置换的问题。而此时虚拟内存只是基于一个简单的概念，即使用外存作为内存的扩展。直至 70 年代，内存资源的成本变得越来越昂贵，在物理内存容量不变的情况下，使用外存作为对物理内存逻辑上扩充的支持，虚拟存储技术应运而生。美国 RCA（Radio Corporation of America，美国广播唱片公司）公司研究成功虚拟存储器系统[[[1]](#endnote-0)]。虚拟存储技术本质上是一种对物理内存在逻辑上进行扩充和保护的内存管理技术。Veritas曾对虚拟存储做过如下的定义：“虚拟存储管理技术本质上是一种将物理内存资源进行逻辑化处理的技术[[[2]](#endnote-1)]，它将复杂的存储器管理过程通过设备的逻辑进行了简化。”在支持虚拟存储的系统中，逻辑地址和物理地址相互分离，互不影响，程序员在编写程序时只需要考虑代码或者变量的逻辑地址而不用处理真实的物理地址。在一些系统中使用了分布式管理和虚拟存储管理相结合的技术，可以很好地支持数据的高扩展性、易移植性等，并且可以很方便得对数据进行集中管理和分散存储。在上世纪 70 年代，IBM（International Business Machines Corporation，国际商业机器公司）在 IBM370 系统上使用了虚拟存储技术。

上世纪 80 年代至 90 年代是 PC（Personal Computer，个人计算机）开始流行的一段时期，这段时间在计算机历史上被称作是 PC 黄金时代[[[3]](#endnote-2)]。在 PC 流行的大趋势下，研究如何提高DRAM 的性能变得尤为迫切。在这样的“PC 热”的历史背景下，一些学者提出了内存分配器的概念，之后对内存分配器的研究和优化逐渐开始进行。首先得到飞跃发展的是内存分配的概念，思想和机制以及内存管理的算法，这个发展过程持续了十年左右。一些学者在上世纪末的时候就已经开始了内存分配器优化的研究，并且这种研究是基于多线程管理环境的。1992 年到 1994 年这两年期间，Zone 以及 Grunwald 和他们的合作伙伴在内存分配器的优化方面做了很多的工作[[[4]](#endnote-3)]，对内存分配器从包括时间、局部性、空间等多个方面综合实验数据进行了大量实验评估。第一个被广泛使用的通用动态内存分配器——dlmalloc于1996年问世[[[5]](#endnote-4)]。Dlmalloc 依据所需内存块的大小使用最佳适配算法来进行物理内存的分配，使用双向循环链表数据结构来管理小块的内存，使用树的数据结构来管理大块的内存，并且如果两个内存块是相邻的，则直接合并这两个相邻的内存块以减少内存外部碎片。Dlmolloc 堆分配曾一度成为最节省内存资源且有效访存速度最快的内存分配器。在上世纪 90 年代末，由Wolfram Gloger 开发的基于 dlmalloc 的 ptmalloc 内存分配器的第一个版本在更新到了2.7版本后就被整合到了glibc2.3之中[[[6]](#endnote-5)]，在这之后，Linux 则一直将该内存分配器作为自己默认的分配器。2000年初，Lkmalloc 提出了一种内存分配器架构，该架构基于多线程的并发应用[[[7]](#endnote-6)]。Lkmalloc 深入研究了之前的内存分配器，发现之前的内存分配器的设计原则基本是针对短期运行的程序，而忽略了需要长期运行在服务器上的应用程序。从这个角度出发，Lkmalloc研究了一次性应用程序和服务器应用程序对于内存分配的不同需求并开发出了自己的内存分配器。在这之后，Hsu 和 Vee 两人从有效访存速度的角度出发，认为使用多处理器的本质原因不是追求更大的内存容量，而是加快程序运行速度，提高系统的时间性能[[[8]](#endnote-7)]。因此在支持多线程的系统中，内存分配器的时间性能要远远比空间性能重要。

2010 年之后，传统 DRAM 的发展面临严峻的挑战。DRAM 介质的能耗以及制造工艺两方面的瓶颈严重影响了系统性能，学术界不得不去寻找一种新的可以替代 DRAM 的存储介质。在材料领域的研究中，对非易失性存储介质的研究得到了快速发展，并逐渐在计算机存储领域获得了广泛的关注，之后引起了新型非易失性存储介质的研究热潮。在近几年的研究中，学者尝试在虚拟存储管理技术中引入非易失性存储介质，并研究这种引入对传统内存分

配器带来的挑战并试图从不同的角度提出相应的解决方法。

2014 年，由 Soyoon Lee 等人提出 CLOCK-DW 页替换策略，该策略基于页写入历史记录，由历史记录中的页访问频率来估计页未来被访问的可能性。

2014 年，由 Guangxia Xu 等人提出 CLRU 页替换策略，该策略通过延迟冷脏页的逐出来减少“写”操作次数，并通过优先驱逐冷页来提高页命中率。

2016 年，Omran 等人提出了一种新的页替换策略（主要通过散列算法），通过减少“写”操作的次数来降低系统功耗。

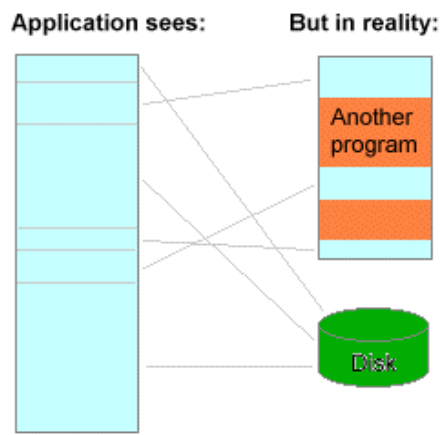
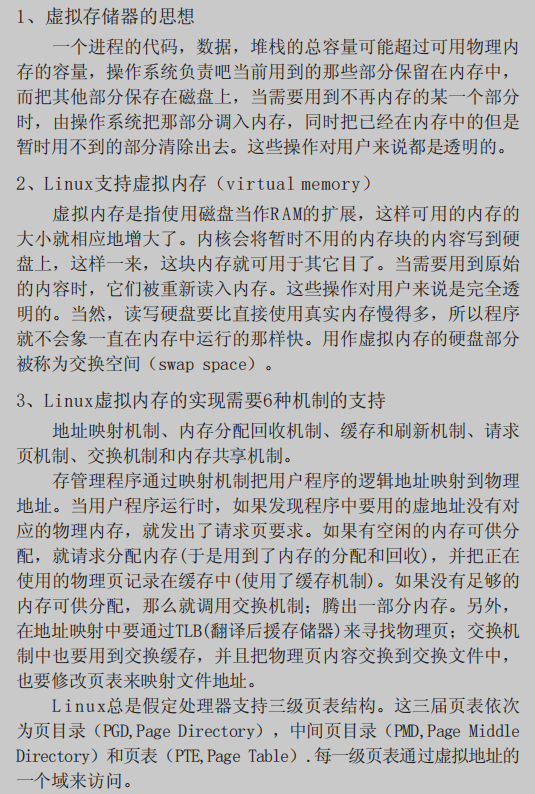
2017 年，由 Hyokyung Bahn 等人提出了一种新的内存架构，从 TLB 命中率和页错误率两个方面来考虑页管理性能。

近几年，计算机技术的发展已经进入了大数据、人工智能时代，与之相对应的是海量数据处理的迫切需求。大数据和人工智能相关的应用场景对传统的虚拟存储技术提出了新的要求。

**介绍**（介绍部分综合参考了多篇关于虚拟内存的底层研究论文后归纳得出）

1. 虚拟内存的背景和概念

虚拟内存别称虚拟存储器，是计算机系统内存管理的一种技术，也属于计算机体系结构中的重要内容。计算机上运行的程序都需要拥有内存以存储指令和数据，而当执行的程序过多或占用的内存很大时，会导致内存迅速耗尽[[[9]](#endnote-8)]，此时计算机就会使用虚拟内存技术，通过将部分程序或进程的数据存储在硬盘上以扩展计算机的实际物理内存,让应用程序认为它具有连续可用的内存，从而让所有并发的进程能够正常调用内存并且正常执行，并且这种技术使得大型程序的编程变得更为容易，对于物理内存的使用也更有效率。在windows操作系统中，虚拟内存在硬盘上其实是一个名为PageFile.Sys的文件[[[10]](#endnote-9)]

 （应用程序认为自己拥有了连续可用的内存）

1. 虚拟内存涉及的硬件和软件技术[[[11]](#endnote-10)]

2.1计算机存储器

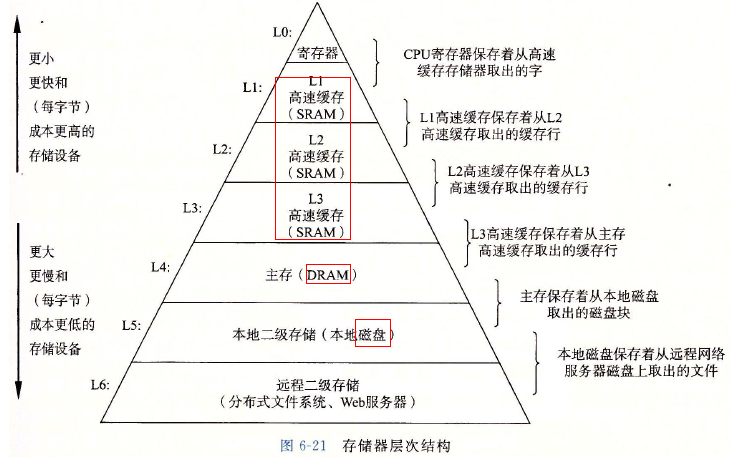
存储器是计算机的核心部件之一，在完全理想的状态下，存储器应该要同时具备以下三种特性：

1.速度足够快：存储器的存取速度应当快于 CPU 执行一条指令，这样 CPU 的效率才不会受限于存储器

2.容量足够大：容量能够存储计算机所需的全部数据

3.价格足够便宜：价格低廉，所有类型的计算机都能配备

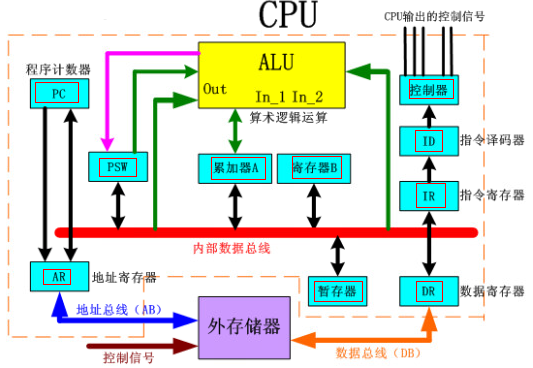
然而实际情况下计算机并不能同时满足上述的三个条件，于是现代计算机存储器设计采用了一种分层次的结构：



自顶向下分别是Register、Cache、主存、磁盘。这些存储器的访问速度逐级递减，但是容量逐级递增

* + 1. 寄存器

存取速度最快的是寄存器，寄存器的制作材料和CPU是相同的，速度和CPU一样快，CPU访问寄存器是没有时延的，然而因为价格昂贵，因此容量也极小，一般32位的CPU配备的寄存器容量是32\*32 Bit，64位的CPU则是64\*64 Bit，不管是32位还是64位，寄存器容量都小于1 KB，且寄存器也必须通过软件自行管理



* + 1. 高速缓存(cache)

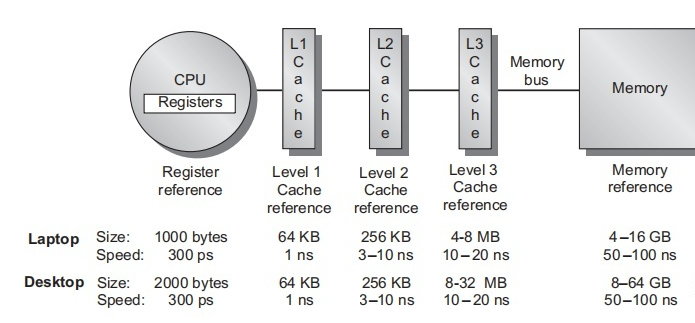
访问速度比一般随机存取存储器（RAM）快的一种RAM，通常它不像系统主存那样使用DRAM技术，而使用昂贵但较快速的SRAM技术。

第一级Cache通常访问只需要几个周期，大小通常是几十个KB。

第二级Cache比第一级cache约有2~10倍较高的延迟，但大小稍微更大些。

第三级Cache比L2更高的延迟性，但通常有数MB之大。

第四级Cache是CPU外部的DRAM，但速度较主存更高。



* + 1. 主存

也即主内存，通常称作随机访问存储器（Random Access Memory, RAM）。是与CPU直接交换数据(cache充当介质)的内部存储器。它可以随时读写（刷新时除外），而且速度较快（比cache慢），通常作为操作系统或其他正在运行中的程序的临时资料存储介质

* + 1. 磁盘

磁盘是[电脑](https://zh.wikipedia.org/wiki/%E7%94%B5%E8%84%91)上使用坚硬的旋转盘片为基础的[非易失性存储器](https://zh.wikipedia.org/wiki/%E9%9D%9E%E6%8F%AE%E7%99%BC%E6%80%A7%E8%A8%98%E6%86%B6%E9%AB%94)，它在平整的磁性表面存储和检索数字数据，数据通过离磁性表面很近的磁头由电磁流来改变极性的方式被写入到磁盘上，数据可以通过盘片被读取，磁盘和主存相比，每个二进制位的成本低了两个数量级，因此容量比之会大得多，动辄上 GB、TB，而缺点则是访问速度则比主存慢了大概三个数量级。机械硬盘速度慢主要是因为机械臂需要不断在金属盘片之间移动，等待磁盘扇区旋转至磁头之下，然后才能进行读写操作，因此效率很低

2.2 主存

2.2.1 物理内存

物理内存就是2.1.3所说的计算机存储器RAM主存，在计算机硬件中以内存条的形式存在，嵌在主板的内存槽上，用于加载程序的指令和数据供CPU运行和使用

2.2.2 ***虚拟内存*** [[[12]](#endnote-11)]

虚拟内存的核心原理是为每个程序设置一段"连续"的虚拟地址空间，把这个地址空间分割成多个具有连续地址范围的页 (Page)，并把这些页和物理内存做映射，在程序运行期间动态映射到物理内存。当程序引用到一段在物理内存的地址空间时，由硬件立刻执行必要的映射；而当程序引用到一段不在物理内存中的地址空间时，由操作系统负责将缺失的部分装入物理内存并重新执行失败的指令

**页**

虚拟地址空间按照固定大小划分成被称为页（Page）的若干单元，物理内存中对应的则是页框（Page Frame）。这两者一般来说是一样的大小，实际计算机系统中一般是512字节到1GB，这就是虚拟内存的分页技术。因为是虚拟内存空间，每个进程分配的大小是4GB (32位架构)，而实际上当然不可能给所有在运行中的进程都分配4GB的物理内存，所以虚拟内存技术还需要利用到一种交换（swapping）技术，也就是通常所说的页面置换算法，在进程运行期间只分配映射当前使用到的内存，暂时不使用的数据则写回磁盘作为副本保存，需要用的时候再读入内存，动态地在磁盘和内存之间交换数据

**页表**

进行虚拟地址到物理地址的映射时，都需要读取页表，从数学角度来说页表就是一个哈希函数，传入参数是虚拟页号（Virtual Page Number，VPN），输出是物理页框号（Physical Page Number，PPN，也就是物理地址的基址）。

页表由多个页表项（Page Table Entry, PTE）组成，一般来说页表项中都会存储物理页框号、修改位、访问位、保护位和 有效位等信息。

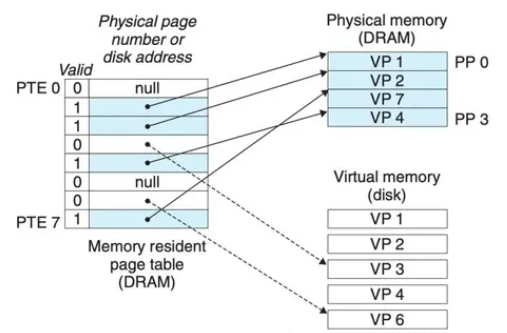
·物理页框号：这是PTE中最重要的域值，提供VPN到PPN的映射。

·有效位：表示该页面当前是否存在于主存中，1表示存在，0表示缺失，当进程尝试访问一个有效位为0的页面时，就会引起一个缺页中断。

·保护位：指示该页面所允许的访问类型，比如0表示可读写,1表示只读。

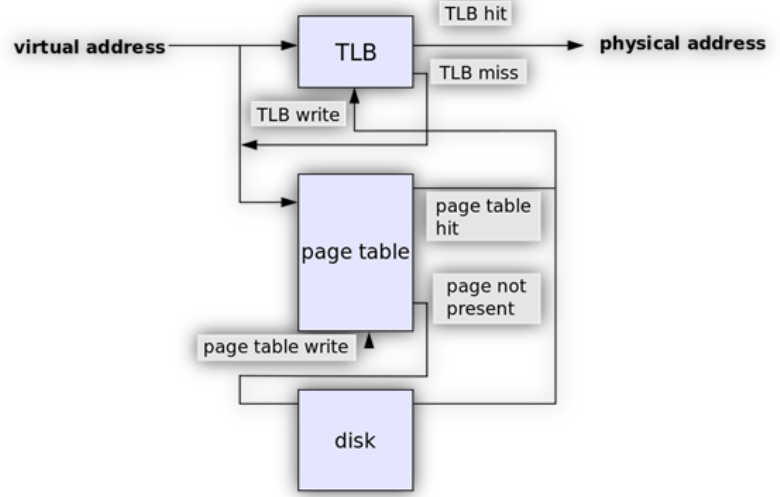
·修改位和访问位：为了记录页面使用情况而引入的，一般是页面置换算法会使用到。当一个内存页面被程序修改过之后，硬件会自动设置修改位，如果下次程序发生缺页中断需要运行页面置换算法把该页面调出以便为即将调入的页面腾出空间之时，就会先去访问修改位，从而得知该页面被修改过，也就是脏页 (Dirty Page)，则需要把最新的页面内容写回到磁盘保存，否则就表示内存和磁盘上的副本内容是同步的，无需写回磁盘；而访问位同样也是系统在程序访问页面时自动设置的，它也是页面置换算法会使用到的一个值，系统会根据页面是否正在被访问来觉得是否要淘汰掉这个页面，一般来说不再使用的页面更适合被淘汰掉。

·高速缓存禁止位：用于禁止页面被放入CPU高速缓存，这个值主要适用于那些映射到寄存器等实时I/O设备而非普通主存的内存页面，这一类实时I/O设备需要拿到最新的数据，而CPU高速缓存中的数据可能是旧的拷贝。



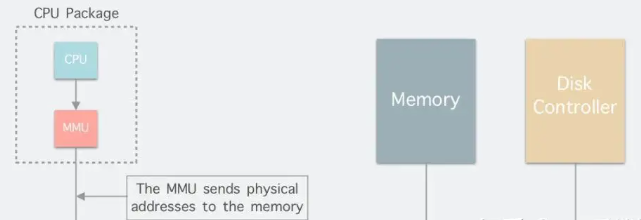
TLB：快表，相当于CPU中的cache

将虚拟地址转换为物理地址时，首先搜索TLB。如果TLB命中，则返回物理地址并继续存储器访问。然而，如果没有匹配，则存储器管理单元或操作系统TLB未命中处理器通常会查找页表中的地址映射以查看是否存在映射。如果存在，则将其写回TLB，并且重启错误指令（可以并行发生）



**地址翻译和虚拟内存寻址过程**

进程在运行期间产生的内存地址都是虚拟地址，如果不使用虚拟内存这种抽象技术，CPU会把这些地址直接送往内存地址总线，访问和虚拟地址相同地址值的物理地址；如果使用了虚拟内存，CPU会将虚拟地址送往内存管理单元（MMU），MMU将虚拟地址翻译成物理地址后再访问相应的物理地址



·虚拟地址分为两部分：虚拟页号（VPN）和偏移量（VPO）

页面**命中**时，计算机各个硬件的交互步骤如下：

·第1步：CPU处理器生成一个虚拟地址VA，通过总线发送到MMU

·第2步：MMU通过虚拟页号得到页表项的地址，通过内存总线从CPU高速缓存/主存读取这个页表项PTE

·第3步：CPU高速缓存或者主存通过内存总线向MMU返回页表项PTE

·第4步：MMU把页表项中虚拟页号对应的物理页框号PPN复制到寄存器的高位中，接着把后面位的偏移量VPO复制到寄存器的末尾几位构成总的物理地址，接着把该寄存器存储的物理内存地址PA发送到内存总线，访问高速缓存/主存

·第5步：CPU 高速缓存/主存返回该物理地址对应的数据给处理器

页面**缺失**时（Page Fault），计算机各个硬件的交互步骤如下[[[13]](#endnote-12)][[[14]](#endnote-13)]：

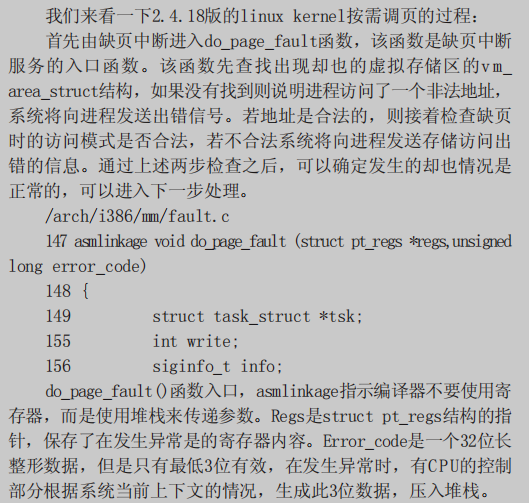
·第1步-第3步：与页面命中时的前三部一致

·第4步：MMU检查发现返回的PTE有效位为0，触发一次缺页中断异常，CPU进入操作系统内核中的缺页中断处理器

·第5步：缺页中断处理程序检查所需的虚拟地址是否合法，确认合法后系统则检查是否有空闲物理页框号PPN可以映射给该缺失的虚拟页面，如果没有空闲页框，则执行页面置换算法寻找一个现有的虚拟页面淘汰，如果该页面已经被修改过，则写回磁盘，更新该页面在磁盘上的副本

·第6步：缺页中断处理程序从磁盘调入新的页面到内存，更新页表项PTE

·第7步：缺页中断程序返回到原先的进程，重新执行引起缺页中断的指令，CPU 将引起缺页中断的虚拟地址重新发送给 MMU，此时该虚拟地址已经有了映射的物理页框号 PPN，因此会按照前面页面命中的流程走一遍，最后主存把请求的数据返回给处理器

（Linux虚拟内存管理）

**页面替换算法**

·先进先出FIFO

将进入内存的页面排队，将已经在队列中最前面的也即最先进入内存的页面淘汰

·最近最少使用LRU

根据页面调入内存后的使用情况决策，选择最近最久未使用的页面淘汰，算法实现上赋予每个页面一个访问字段T，记录一个页面上次被访问以来经历的时间总和，淘汰一个页面时，选择T值最大的页面进行淘汰

·时钟CLOCK

每页设置一个访问位，再将内存中的所有页面都通过链接指针链接成一个循环队列；当某个页面被访问时，其访问位置1。淘汰时，检查其访问位，如果是0，就换出；若为1，则重新将它置0；再按FIFO算法检查下一个页面，到队列中的最后一个页面时，若其访问位仍为1，则再返回到队首再去检查第一个页面

·最佳替换OPT

选择以后永远不使用或者是未来最长时间内不使用的页面作为被淘汰页面，可以获得最低的缺页率，但是无法预知未来哪个页面是最长时间不访问的，因此该算法无法实现

**虚拟内存和Cache**

如果一台计算机同时配备了虚拟内存技术和CPU高速缓存，那么MMU每次都会优先尝试到高速缓存中进行寻址，如果缓存命中则会直接返回，只有当缓存不命中之后才去主存寻址。通常来说，大多数系统都会选择利用物理内存地址去访问高速缓存，因为高速缓存相比于主存要小得多，所以使用物理寻址也不会太复杂；另外也因为高速缓存容量很小，所以系统需要尽量在多个进程之间共享数据块，而使用物理地址能够使得多进程同时在高速缓存中存储数据块以及共享来自相同虚拟内存页的数据块变得更加直观。

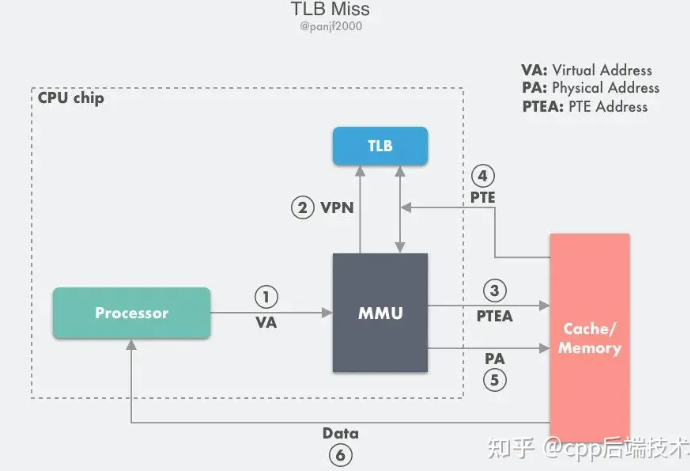
**TLB**

快表，可以理解成页表的高速缓存，保存了最高频被访问的页表项PTE。由于TLB一般是硬件实现的，因此速度极快，MMU收到虚拟地址时一般会先通过硬件TLB并行地在页表中匹配对应的PTE，若命中且该PTE的访问操作不违反保护位（比如尝试写一个只读的内存地址），则直接从TLB取出对应的物理页框号PPN返回，若不命中则会穿透到主存页表里查询，并且会在查询到最新页表项之后存入TLB，以备下次缓存命中，如果TLB当前的存储空间不足则会替换掉现有的其中一个 PTE。

TLB**命中**时：

1. CPU产生一个虚拟地址
2. MMU向TLB请求PTE并获取PTE
3. MMU从PTE将这个VA翻译成物理地址并发送到cache/内存中
4. Cache/内存将物理地址上的数据返回给CPU

如下图所示：

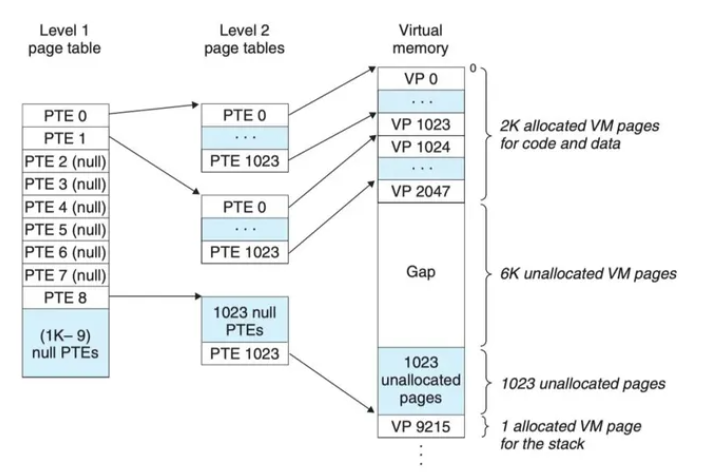


TLB**未命中**时：

1. CPU产生一个虚拟地址
2. 查询TLB失败，通过之前的虚拟地址寻址方法得到PTE，将其存入TLB中，如哦TLB存储空间不足，则淘汰掉TLB中的某个PTE
3. - 4.与TLB命中时相同

**多级页表**

理论上一台32位的计算机的寻址空间是4GB，也就是说每一个运行在该计算机上的进程理论上的虚拟寻址范围是4GB。到目前为止，前面所讨论的一直都是单页表的情形，如果每一个进程都把理论上可用的内存页都装载进一个页表里，但是实际上进程会真正使用到的内存其实可能只有很小的一部分，而我们也知道页表也是保存在计算机主存中的，那么势必会造成大量的内存浪费，甚至有可能导致计算机物理内存不足从而无法并行地运行更多进程。这个问题一般通过**多级页表**（Multi-Level Page Tables）来解决，通过把一个大页表进行拆分，形成多级的页表，我们具体来看一个二级页表应该如何设计：假定一个虚拟地址是32位，由10位的一级页表索引、10位的二级页表索引以及12位的地址偏移量组成，则PTE是4字节，页面page大小是2^12=4KB，总共需要2^20个PTE，一级页表中的每个PTE负责映射虚拟地址空间中的一个4MB的chunk，每一个chunk 都由1024 个连续的页面Page组成，如果寻址空间是4GB，那么一共只需要1024个 PTE 就足够覆盖整个进程地址空间。二级页表中的每一个 PTE 都负责映射到一个 4KB 的虚拟内存页面，和单页表的原理是一样的。多级页表的关键在于，并不需要为一级页表中的每一个 PTE 都分配一个二级页表，而只需要为进程当前使用到的地址做相应的分配和映射。因此，对于大部分进程来说，它们的一级页表中有大量空置的 PTE，那么这部分 PTE 对应的二级页表也将无需存在，这是一个相当可观的内存节约，事实上对于一个典型的程序来说，理论上的 4GB 可用虚拟内存地址空间绝大部分都会处于这样一种未分配的状态；更进一步，在程序运行过程中，只需要把一级页表放在主存中，虚拟内存系统可以在实际需要的时候才去创建、调入和调出二级页表，这样就可以确保只有那些最频繁被使用的二级页表才会常驻在主存中，此举亦极大地缓解了主存的压力。



**虚拟内存模拟实现的论文综述**

虚拟内存的模拟实现乍一看似乎并没有什么意义，实际上却有着举足轻重的作用。

首先虚拟内存的模拟实现可以有教学和学习目的：通过模拟实现虚拟内存，可以帮助我们更好地理解虚拟内存的工作原理和实现机制。这种模拟通常通过软件程序来模拟硬件系统中的内存管理过程，使我们能够在不涉及真实硬件的情况下，直观地了解虚拟内存是如何工作的。

其次，虚拟内存的模拟实现有利于性能评估与优化：虚拟内存的模拟实现可以用于评估不同内存管理策略对系统性能的影响。通过模拟不同的场景和使用不同的算法，可以分析各种情况下的性能表现，并优化内存管理策略以提高系统的效率和响应速度。正如我即将介绍的韩苏硕士的论文[[[15]](#endnote-14)]中所提及的（为了提高访存速度，需要减少数据搬移，特别是对于需要多CPU协同处理的系统。在这样的系统中，确保系统间安全有效地传输数据，降低因系统数据频繁搬移而带来的延迟，就需要研究一种适用的共享存储系统。本文将探讨多处理器多存储间数据共享的问题，结合传统存储器的设计原理提出一种全新的多端口共享存储器设计方法）新技术，如果要应用在硬件上，应当首先在软件在上进行模拟实现进行性能的评估和实现的优化。

再者，虚拟内存的模拟实现能帮助研究人员对虚拟内存的调试和错误分析：在软件开发过程中，模拟虚拟内存可以帮助开发人员更好地理解和调试与内存管理相关的问题。通过模拟实现，开发人员可以模拟出特定的内存使用情况，分析程序的内存访问模式以及可能出现的错误，有助于及时发现和解决潜在的内存管理问题。

然后，虚拟内存的模拟实现还可以实现算法验证与研究： 虚拟内存的模拟实现也可以用于验证和研究新的内存管理算法。研究人员可以在模拟环境中测试和比较不同的算法，评估其性能和有效性，为新算法的设计和实现提供可靠的参考和验证。

跨平台兼容性：软件模拟虚拟内存的实现可以更容易地在不同平台上运行，而不受特定硬件环境的限制。这样的实现可以在各种计算机系统上进行测试和应用，为跨平台软件的开发和优化提供便利。

1. 基于FPGA的虚拟内存管理模块设计(引自南京邮电大学韩苏的学位硕士论文)[[[16]](#endnote-15)]

3.1研究背景和意义

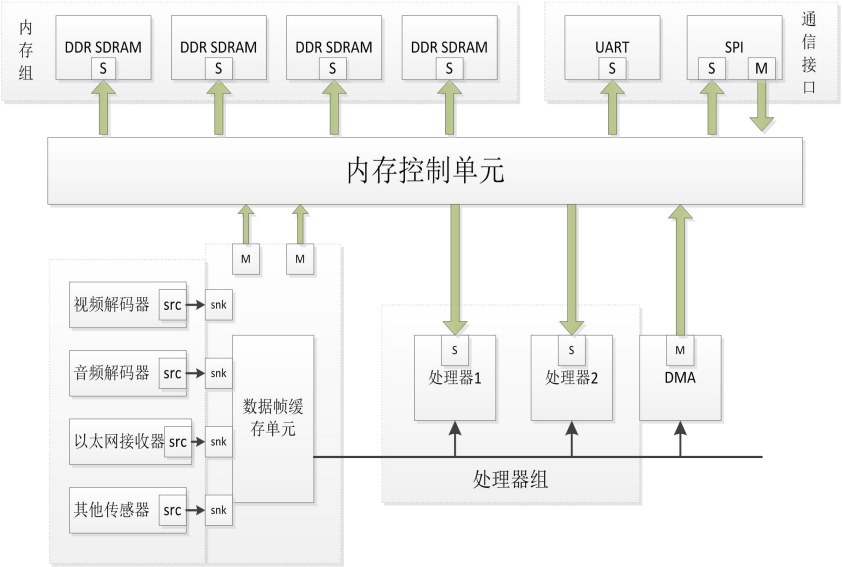
随着互联网技术的飞速发展，网络应用的种类和规模也越来越多样化，不同类型的网络之间的整合问题变得越来越明显。举个例子，阿里云的存储系统在2014年已经达到了上万台机器规模，可以支持超过1000PB的储存。如何确保这么大规模的计算机系统能够稳定运行，成为当前互联网技术领域研究的热点。

在传统的网络交互计算系统中，接收到的数据首先存储在网卡的接收缓存器中。在提取有用信息时，数据还需要搬移到主存储器进行处理，处理完成后再搬移到输出网卡的缓存器，最后通过输出缓存器传送出去。对于小规模计算来说，这种频繁搬移数据的延时还可以接受。但对于大规模甚至超大规模的并行机器系统来说，这个过程中频繁搬移数据将导致很大的数据传输时延，无法适应当前多媒体、数据库等访存密集型网络应用的发展。

尤其是对于广泛采用的多核集群系统，系统由多个磁盘组成，每块磁盘的延时大约在10毫秒左右。一个小规模公司级别的并行计算机系统至少包含10个这样的存储磁盘。而事实上，大多数应用对延时的要求可能都小于100毫秒，这还不包括在数据共享时，CPU花费在数据在各个模块间搬移和存储的延时。

因此，为了提高速度，需要减少数据搬移，特别是对于需要多CPU协同处理的系统。在这样的系统中，确保系统间安全有效地传输数据，降低因系统数据频繁搬移而带来的延迟，就需要研究一种适用的共享存储系统。本文将探讨多处理器多存储间数据共享的问题，结合传统存储器的设计原理提出一种全新的多端口共享存储器设计方法，并在Altera公司的FPGA上进行实现验证。

3.2 研究内容



论文中提到了一个如上所示的综合业务通信平台系统，其中包括视频解码器、音频解码器、以太网接收器以及其他传感器通过数据帧缓存单元接入系统。核心处理器单元（处理器1和处理器2）负责处理和转发这些数据帧。由于处理的缓冲区数据量巨大，处理器组的SRAM不能满足需求，因此通过内存控制单元连接了四块DDR内存作为内存组扩展单元。

为了满足系统扩展的需求，该综合业务平台还扩展了UART串口通信接口作为系统的配置接口，并引入了SPI总线以提高板间通信质量。为了实现对DDR内存块的快速访问，引入了DMA引擎模块，通过DMA引擎，音视频解码器和其他外设可以快速访问系统DDR内存组，无需系统处理器组干预。

在整个综合业务通信系统中，内存控制单元被提到作为系统的核心单元，提供操作接口、整合存储资源、控制协同操作。该控制模块具有内存管理、全局写入、全局读出以及并发调用等功能，能够实现多处理器系统间并行操作时的数据共享和交换，减少数据搬移，实现内存的虚拟化管理和高速并发调用。

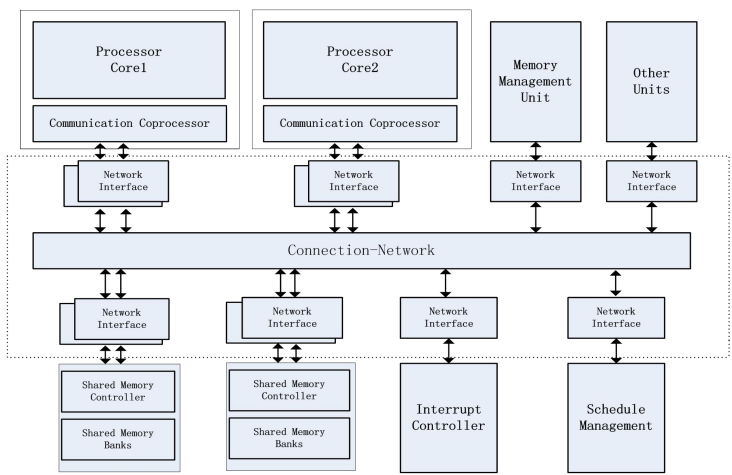
此外，论文还提到了研究多处理器多内存系统面临的问题，基于FPGA的内存拷贝和内存分离技术，以及内存管理单元的拓扑结构，重点采用了Crossbar架构。还涉及了单核和多核情况下DDR内存的接口和时序操作，以及系统支持的多种类型总线通信，如PCIE总线和Avalon总线。DMA扩展模块也被研究，并在开发的系统上进行了验证。

最后，论文提到了采用FPGA设计的典型方法，使用Quartus II 13.1软件编写各模块功能代码，Qsys系统开发组件对Nios核和接口软核进行例化。逻辑验证使用Modelsim进行仿真，时序分析则利用Quartus自带的SignalTap逻辑分析仪。整体方案完成后，通过Altera公司的GX系列FPGA搭建测试环境，对各功能进行实际测量，并进行了结果分析和总结。论文最后提到，这是网络通信融合教研室研究的课题，对于后续研究提供了实际的参考意义。

3.3 基于虚拟内存共享内存概述

3.3.1多处理器虚拟内存控制系统

随着网络技术的发展，系统的网络规模和形式也越来越多元化，异构网络间任务协作融合问题也越来越突出。在这样的大背景下，多处理器系统（MP-S）在 网络融合尤其是嵌入式系统融合中显的尤为重要。在嵌入式领域，多处理器融合 系统架构正在成为嵌入式多媒体应用设计中最新的解决方案。 一般来说，多处理器融合系统架构由多个核心处理器、存储器和一些通用和专用的外部接口组成，模块机间通信由高级互连结构提供。多处理器融合系统广泛应用在数据交换 领域，例如，用于以太网网络信号处理的高性能网络处理器片上系统（NoCs）



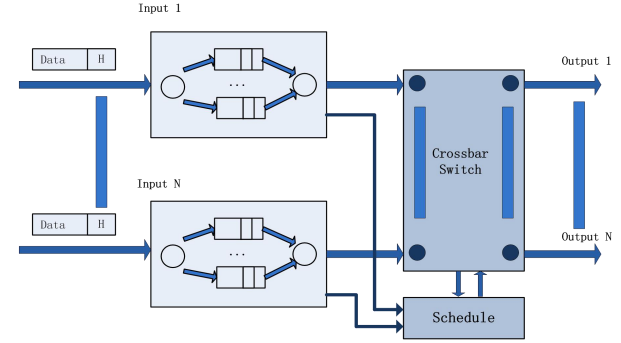
论文指出了多处理器共享内存架构系统的三个主要特点。

1.首先，论文提到了系统通信，强调多处理器系统通常是由模块化器件相互连接而成的，这种连接方式可以采用不同的片上网络拓扑结构。这些片上路由通常是分布式的，调度算法一般是静态的或者基于数据表。片上路由的类型包括总线式（Bus）分布和基于交叉开关（Crossbar）的结构。

2.其次，论文强调了存储器系统在多处理器共享内存架构中的重要性。MP-S架构的关键支出在于支持处理模块（PE）之间的同步和数据交换。因此，MP-S架构通常采用非一致性存储器存取（NUMA）方式。存储器访问延迟在这种系统中是不均匀的，受时间和器件本身的约束。由于系统通过互连拓扑连接而成，存储器空间不仅考虑相互独立的物理地址，还需考虑分布式互连带来的逻辑共享的存储器地址块，以及每个处理模块（PE）的专用存储器地址空间。

3.最后，论文提到了内存管理单元（MMU）的重要性。MMU的主要功能是在硬件中实现内存的等效划分，以减少由于操作数据时带来的存储器数据搬移开销和性能方面的消耗。MMU在动态管理物理分布式共享内存空间上的数据分配和释放方面发挥关键作用。

论文的主要焦点是探索适用于FPGA及FPGA环境下多CPU分布式共享内存控制器系统。在图 2.2 所示的基于Crossbar 内存数据数据交换系统中，访问主机采取访问控制和数据分开的方式对内存块寻址。首先，访问主机发出访问请求消息命令，该命令被存放在消息缓存队列（比如 FIFO）中，调度器模块（Schedule）按照一定的排序方法（比如轮询）读取来自访问主机的访问命令格式消息，该消息中包含本主机访问路由信息。Crossbar作为支持全部端口访问网络，其节点由多路选择开关组成，负责访问主机数据总线和访问内存块数据总线选断，具体路径选择方法由调度器读取到的片选及内存地址分发。



3.3.2 虚拟内存控制器互连架构

常见的控制器互连架构有如下三种：

1.点到点连接

2.共享总线连接

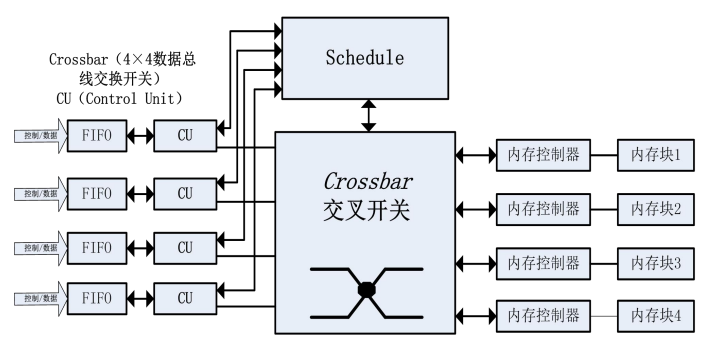
3.全局crossbar交换矩阵连接

3.4 FPGA虚拟内存管理模块关键技术的研究

3.4.1虚拟内存控制管理模块的设计目标和组成

论文的目标是建立一个高效的内存控制模块，该模块能提供作接口，整合存储器资源。同时该控制模块具有内存管理接口，支持内存组全局写入、全局读出、单块内存独占式访问以及并发调用等功能。该控制块包括数据包缓存单元、 调度控制单元、数据总线交叉开关矩阵单元以及内存控制器单元等模块组成。主要研究内容分为两点：第一，内存空间复制，多CPU共享数据；第二，多分布内存整合服务，单存储空间独占式访问服务。

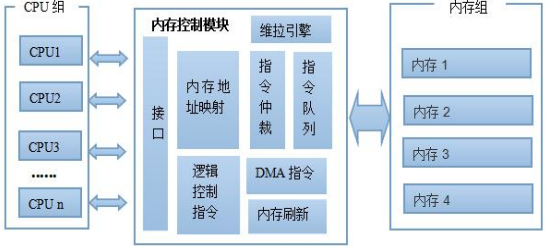
下图为论文用于虚拟内存管理的内存控制模块设计架构。在图示的设计系统中，访问主机采取访问控制和数据分开的方式对内存块寻址。首先，访问主机发出访问请求消息命令，该命令被存放在消息缓存队列（比如 FIFO）中，调度器模块（Schedule）按照一定的排序方法读取来自访问主机的访问命令格式消息，该消息中包含本主机访问路由信息。Crossbar 作为支持全部端口访问网络， 其节点由多路选择开关组成，负责访问主机数据总线和访问内存块数据总线选断，具体路径选择方法由调度器读取到的片选及内存地址分发。



在调度器完成访问地址任务分发后，Crossbar 交叉开关建立按地址寻址的数据信道通路，此时，访问主机真正实现了到访问内存的地址写控制命令和数据总线信道创建。当访问主机完成本次访问（读/写）后，内存控制块负责刷新选通路由及访问控制标志。调度器进行下一个访问任务的调度。

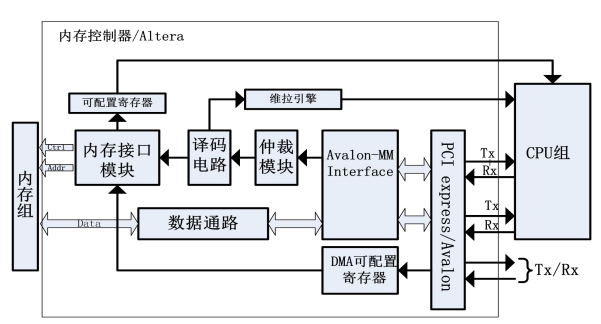
3.4.2 虚拟内存管理的内存控制器逻辑功能

下图为论文用于虚拟内存管理的内存控制器逻辑功能图，该图描述CPU组、内存控制模块以及内存组之间的连接模型，并对内存控制模块内部主要功能做了标识。内存控制模块连接 CPU 组和内存组。当 CPU 组中 CPU(CPU1,2...)发出请求寻址内存组中某块内存（内存 1,内存 2...）时，首先 CPU 组会把请求消息发送到内存控制模块，内存控制模块内存地址映射以及仲裁指令把请求消息转换成 SDRAM 内存总线的消息形式，这个消息可以访问到内存组中每一块内存块。首先，CPU 组把上层的逻辑地址消息通过接口总线推送到内存控制模块，内存控制模块把对应的逻辑地址转换成可被内存组识别的物理地址并推送到内存组中。反过来，当内存控制器读取到内存组中内存的消息时，还负责应答把读取到的消息封装成内部消息，最终转化成可被 CPU 组识别的消息格式，并通过双向数据总线推回给 CPU 组。值得注意的是，内存控制模块和 CPU 组之间以及内存组之间采用双向数据总线，并且为了节省宝贵的 I/O 资源的开销，通常情况下这些数据总线均采用复用方式传递消息。



3.5 FPGA虚拟内存管理单元关键技术的实现

论文基于FPGA虚拟内存管理单元的硬件实现，主要包括 CPU 组（Nios II 处理器+DMA 引擎）单元、系统总线（PCIE+Avalon 总线）单元、DDR 单核内存控制器接口模块、仲裁模块以及内存组（HYS64T128021）单元。



3.5.1 软件开发平台

Quartus II 13.1 是 Altera 公司 CPLD/FPGA 专用综合性开发软件，它支持原理图构建，支持 Verilog，VHDL 以及 AHDL 语言编程，内嵌功能强大的 IP 核库，同时，Quartus II 13.1 还支持逻辑仿真和时序仿真功能。和早期版本比起来，

Quartus II 13.1 还支持全新的 Qsys 片上设计组件，为 FPGA 提供全新的片上（SOPC）设计环境。本系统中，Quartus II 13.1 负责全部系统设计，包括代码编写及编译、系统集成、组件接口设计、综合布线、FPGA 片上下载以及运行时

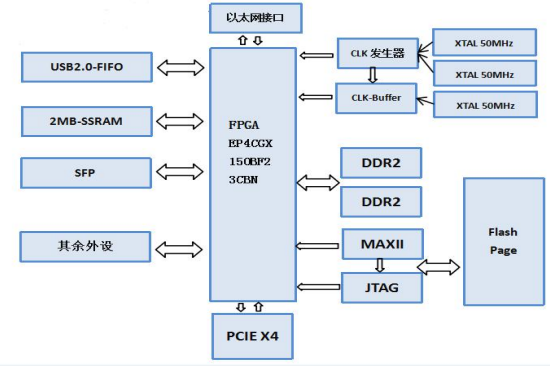
逻辑分析等功能。Modelsim 是数字逻辑设计领域中公认的最为优秀的 HDL 语言仿真软件，它支持 VHDL 和 Verilog 两种语言本地编译的混合仿真。同时还具有对 RTL 综合和开发优化的优点。本系统中 Modelsim 负责单模块功能仿真和接口时序仿真。

3.5.2 硬件开发平台

硬件开发平台采用 Altera 公司提供的低成本、高效能的 Cyclone IV GX 系列 FPGA，采用单块 FPGA 的多核设计模式。

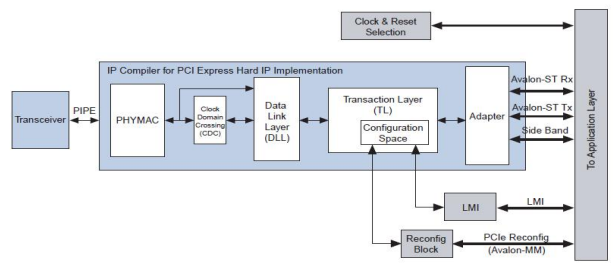
本系统以 EP4CGX150BF23CBN FPGA 为核心，该型 FPGA 具有非常丰富的片上资源，它集成 8 个 3.125Gbps 速率高速收发器，同时内嵌 18 个 M9K 型存储块（Blocks），内存容量高达 2048k bytes，总逻辑门（LE）超过 15 万门。此

外，该型 FPGA 还内嵌 8 个锁相环（PLL）硬核模块，超过 4 个 PCI\_Express 硬核模块。如图 4.2，系统集成了 USB2.0/3.0 通讯接口，支持高达 512MB DDR2 条以扩展大容量缓冲内存。在数字系统设计中，时钟就像系统的心脏一样，本系统具有专用的时钟发送器，系统板时钟高达 150Hz，最大时钟输出 10GHz。此外，本系统还支持 PCIExpress 1.0、PCIExpress 1.1、PCIExpress 2.0、PCIExpress 2.1版本的 PCIE IP 核设计。最高支持 X1、X2、X4 PCIE 链路 能够满足本系统需要的板卡-板卡-PC 主机之间的高速通信。



3.5.3 PCIE总线及接口设计

PCIE 总线设备是一种严格分层结构的总线。在系统中 PCIE 设备和 PCIE 设备之间逐层通过消息包（TLP、DLLP 和 PLP）互相通信。PCIE 设备自顶向下方向上可表示为应用层接口、事务层接口、数据链路层接口以及物理层接口，PCIE 总标准严格规定各层的功能，各个层之间相互配合，分层实现 PCIE总线链路传送的功能。在本系统中，采用 PCIE IP 核实现 PCIE 设备间消息传递的主要功能，PCIE IP核能大大的减少系统设计的工作，同时还可以提供可靠的传输质量。图 1.2 为Altera FPGA 的 PCIE 硬核的工作模型。在图中，按照 PCIE 分层标准协议，PCIEIP包括了传输链路（TS）、物理层（PHYMAC）、数据链路层（DLL）传输层（TL）以及应用层接口。各层完成完成自身的协议，还负责向其他层推送消息包以及接收来自其他层包的功能。其中物理层通过专用通信链路（PIPE）负责与其他 PCIE设备互连通信。在 CycloneIV EP4CGX150BF23CBN 为核心的 FPGA 中，PCIE IP核支持 64 位 Avalon-ST 和 32/64 位 Avalon-MM 两种高带宽通信接口。PCIE IP 核分层工作模型如下图所示：



下面为 Altrea 公司 PCIE IP 核分层模型中模块说明：

1）应用层接口（Application Layer）：用来实现 PCIE IP 核应用层功能，可以使用 MegaWizard 为 PCI Express 根端口或端点创建 IP 编译器插件管理器，并指应用层接口适配器模块（Avalon-ST、Avalon-MM）。

2）配置空间接口（Configuration Space）：用来为链路层和应用层逻辑提供配置空间寄存器，该模块实现 PCIE 的配置空间的主要几个参数的获取功能。

3）PCIE IP 核模块（PCIE IP）：用来实现标准化 PCIE 协议包事务层、数据链路层以及物理层的包处理功能。

4）LMI 反馈模块（LMI）：用来连接传输层和应用层接口，可以刷新基址寄存器用来检测消息数据包 TLP 在传输过程中的错误。

5）重新配置时钟模块（Clock &Reset Selection）：用来完成整个 PCIE IP 核和其他模块的时钟管理工作，同时还能够复位系统。

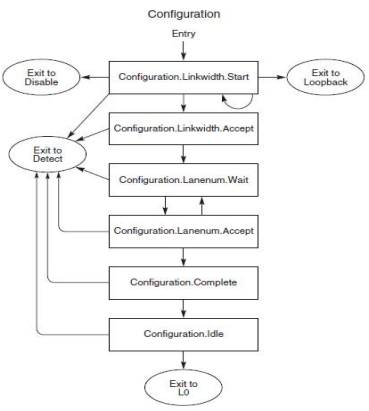
6）PCIE IP 核检测模块：用来测试系统工作时候数据包检测工作，一旦发现错误，本模块将自动发送系统重置信号。

由于 PCIE IP 核实现的是事务层和数据链路层的功能，在使用时候必须调用物理子层（PHY）来用作传输。物理子层使用 FPGA 内部 GTX 高速收发器来实现。在本系统中采用 CycloneIV EP4CGX150BF23CB 型号 FPGA，调用内部 GTX 告诉收发器，此时 GTX 用作 PCIE IP 核的物理 PHY 子层，收发器工作在 8 位链路管道（PIPE）模式下，对上，分别和传输链路（Transceiver）、物理层（PHYMAC）、数据链路层（DLL）传输层（TL）以及应用层接口相连接。Cyclone IV GX 系列 FPGA 支持最高 150M 外部晶振。系统通过调用锁相环（PLL）模块，可以倍频提供 250MHz 的时钟。在设计中，通过调用 Quartus II 13.1软件下的 IP 核专业例化工具 megawizard plug-in manager 对 PCIE IP 核进行实例化，选择 IP Compiler For PCI Express。如图，进入 PCIE IP 核的配置界面，包括 PCIE 核类型、PCIE 基本系统设定、PCIE 寄存器设定以及电源管理等等。本系统使用 CyclonelV GX 系列 FPGA 中的 PCIE 硬核(Hard IP)，PCIE IP 核选择

2.0 版本标准。开发板提供的外部时钟为 125MHz，同时采用 4 通道设计

3.5.4 链路配置空间

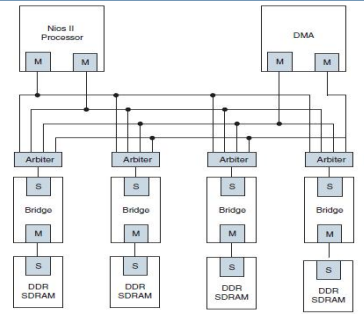
空间配置采样模块的状态机如下图所示



状态机包含 7 个状态，分别是 Start、Accept、Wait、Complete、Exit、Detect 以及 Idle。默认情况下状态机处于初始状态 Idle，Complete 是状态机的完成状态，其他均为状态机的配置状态

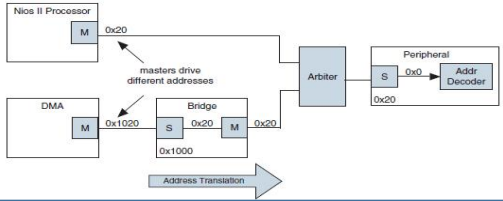
3.5.5 crossbar系统设计

为了满足 Nios II(CPU)和DMA作为主访问设备对四位DDR内存的访问。系统至少采用4×4Crossbar作为主从设备连接器。4×4Crossbar交叉开关具有 2 个独立的数据端口，因此能够执行四个主从设备的同时通信。同时4× 4 Crossbar 交叉开关还具有四个先进先出（FIFO）存储器缓冲器，四个控制单元以及仲裁器和输入消息数据单元。



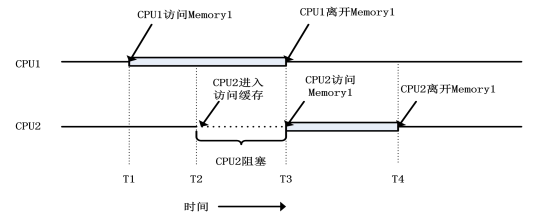
3.5.6 地址空间转换

在Crossbar与PCIE IP核桥接出必须考虑到从设备端口因为基地址而产生的地址跨度。 一般有两种方法，第一，可以自己设置基地址大小，这样方便设计的时候自动增加这一部分的寻址。第二，允许 SOPC Builder 自动设置，这样的处是设计人员不需要管理地址的偏移大小，但是资源消耗可能成倍的增加。考虑到系统只采用到 4 端口 Crossbar，规模不算很大，采用第二种方案。另外， Slave 端口的地址是连接到桥接器的所有组件的基本偏移地址。 连接到桥接器的组件的地址是基地址偏移量和该组件的地址的总和。为了避免设计中不必要的复杂性，系统所以的访问都是从机相同的位置。系统中处理器（NiosII）将缓冲器位置传递给 DMA 控制器，Nios 核和 DMA 控制器可以在同一位置访问同一块同内存。下图为系统中 Nios 核和 DMA 控制器之间的地址转换示意。



3.5.7 访问控制以及多内存调度设计

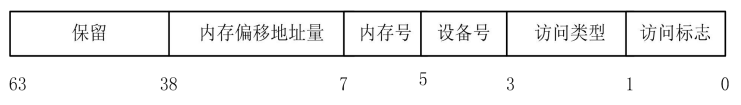
多处理机共享内存资源系统中，多个用户或者多个处理机进程都不可避免的 会对同一块内存区发出访问需求。对于单个用户或者单进程处理机而言，其他用户进程的何时访问以及何时结束是不可知的。所以，为了保证内存块数据的安全 性，系统对每一个内存区实施处理机访问控制，确保一个处理机进程对所访问的 内存块进行独占式访问。访问结束，打开独占权限，允许其他处理机访问。多访问同一块内存（Memory1）的访问策略如下图所示

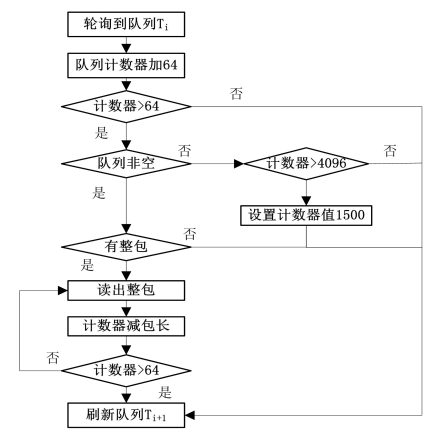


CPU1 首先发起对 Memory1 的访问请求，此时内存控制标志位（Flag）为空闲状态 0，控制器接受请求。CPU1 在 T1 时刻进入 Memory1 独占式访问，并置 Flag 为忙状态 1。随后 CPU2 发起访问请求。但是，Memory1 内存块处于忙状态，不支持访问请求，T2时刻控制器将CPU2的访问请求放入消息缓存队列中，并挂起CPU2 的访问请求。T3 时刻，CPU1 执行进程结束，释放内存占用，并刷新 Flag 为空闲状态。同时，CPU2 获得访问权限，置 Flag 为 1，进入 Memory1 进行数据存取。 T4 时刻，CPU2 执行进程结束，释放内存占用，并刷新 Flag 为空闲状态。CPU1 和 CPU2 完成了独占式访问控制，避免了数据不同步带来的“垃圾”值。

3.5.8 访问设备轮询调度设计

通常情况下，各个处理器系统处理任务不同，这样会导致访问主机自身处理速率和内存接口速率也不同，而且，随着任务的变化，其速率也会发生变化。所以，本系统在设计时候，采用缓存缓冲技术来分配处理机访问任务。即内存控制块接收到来自访问主机的访问任务时候，首先将访问请求存入缓冲队列中。当请求消息被响应，访问标志被置位。由调度器按照一定的规则统一调度处理机的请求任务。考虑到本系统访问主机任务接近、优先级相同，本系统采用轮询方式对访问设备进行调度。

(主机访问消息的格式)

(轮询流程)

3.6 FPGA实现虚拟内存管理模块设计小结

本论文是基于 FPGA 的虚拟内存管理模块的设计与实现。研究了多处理器系统协同工作时的内存共享问题。重点研究了基于 DDR 内存的虚拟内存共享的网络拓扑结构，并研究决定采用 Crossbar 交叉矩阵拓扑结构作为本系统的共享交换互连架构。本文采用自上而下的设计思想，先确定内存管理的目标和关键技术，然后对系统进行模块化设计和分析。重点研究了虚拟内存管理器的内存共享设计方法。在主控 CPU 部分，本文研究了多处理器系统并发访问时的访问控制问题，研究并采用了 NiosII 软核作为系统的主控访问设备。同时，为了提高系统的扩展性，本文研究了 DMA 引擎作为访问控制器访问系统主内存。在系统总线部分，重点研究了 PCIE 总线，PCIE 总线负责控制器和 FPGA以及受访主机内存间的通信，在控制器接口部分，研究采用了 Avalon-ST 和 Avalon-MM 两种通信接口。在完成各功能模块的设计分析之后，本文采用 Verilog 和 VHDL 语言编程，通过仿真软件 Modelsim 对模块进行了时序和逻辑功能的仿真，并在 Altera公司 FPGA 上进行了顶层实现。测试表明，本文设计的虚拟内存控制单元能提供多 CPU多内存访问接口，能满足多处理机间并行操作时的数据共享和交换，能有效减少数据模块之间的搬移。

4. **高性能虚拟存储系统实现技术的研究**(引自工程硕士论文《Research on Implementation Technology of Virtual Storage System with High Performance》)[[[17]](#endnote-16)]

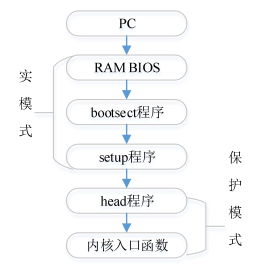
引言：在参考完上一篇基于FPGA的虚拟内存管理模块设计论文以后，我又对比了另一篇实现了虚拟内存模拟实现的论文，本文试图通过实现一个操作系统内核的原型系统研究基于分页的虚拟内存管理功能实现的关键技术,探索提高有效访存速度的页置换策略及算法。

本文详细阐述了具有基本支撑功能的内核原型系统及内存管理功能的实现。其中包括采用GRUB方式的系统引导的实现、内核初始化程序的功能与实现、时钟中断处理程序的实现、进程的创建与管理。在此基础上实现了物理内存的分配与回收、分页机制的启用过程、页表的建立与地址映射、缺页中断处理程序的实现,研究了页置换策略,给出了周期性淘汰内存页的实现方法、淘汰页的时间周期、内存中应保留的最小空闲页数量的阈值,实现了淘汰页的算法。本文研究了页淘汰策略,给出了一种能减少磁盘访问次数,有效提高系统时间性能的方法。通过实验数据验证了改进后的页淘汰策略的有效性。该课题的研究对研发高性能的内存管理系统具有一定的实践意义。

4.1 课题的研究内容：

论文首先阐明设计实现内存管理模块，就需要实现自启动系统，对内核进行一系列初始化。接着论文实现了物理内存管理,核心模块包括物理内存页的分配、回收与释放。接着论文实现了虚拟内存管理,核心模块包括请求分页（请求分页的页表机构、缺页中断处理）和页替换两大功能模块。利用页（4KB）作为最基本的单位进行内存分配，能够为进程分配所需的页框并能够回收和管理空闲页框，实现页置换功能。紧接着设计实现多进程管理实现内核对多个进程的管理，包括进程的控制、进程调度和进程切换，此用来测试内存管理模块。然后设计实现有效的页淘汰策略，对Linux内核采用的页替换策略进行了详细分析，并针对其替换策略的缺陷进行改进，以提高系统的时间性能。最后测试内存管理的有效性，通过安装QEMU虚拟机，并在虚拟机上加载内核，测试内核内存管理和页淘汰算法的有效性。

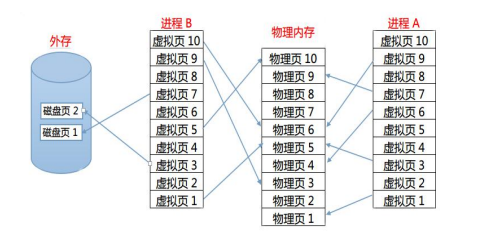
内核的加载过程如图所示：



时钟中断处理的实现如下所示：



基于分页的内存管理机制：

(页的地址映射关系)

接着实现物理内存管理，页目录与页表的创建，实现物理内存的分配与回收，开启分页机制的实现，虚拟内存管理的初始化，缺页中断处理的实现，制定页淘汰策略，最后实现进程管理

4.2 论文小结

本论文中对于虚拟内存的研究颇为深入，也实现了不少内存管理的模拟实现和改进工作：

1.使用GRUB进行内核的引导，并定义实现一些显卡驱动函数和端口读写函数。制作一个使用 FAT32 文件系统格式化的硬盘映像，将该硬盘映像文件挂载到/mnt目录下。修改链接器脚本，定义内核入口函数，使用 GRUB 进行系统的自动引导。同时由于需要在屏幕进行输出，定义了显卡驱动函数来控制屏幕的打印输出，同时封装了端口读写函数。

2.进行中断的初始化，实现中断处理函数。对中断控制器芯片 8259A 进行编程。定义时钟中断和缺页中断的中断处理入口函数，初始化中断描述符表，实现时钟中断处理程序及缺页中断处理程序。

3.实现物理内存管理。了解内核的物理内存管理的实现原理，在内存管理模块中实现基本的物理内存管理，能够以页为单位分配内存和释放物理内存。

4.实现基于分页的虚拟内存管理。研究和学习分页机制下的寻址模式，了解 MMU 进行地址翻译的原理。在内存管理。

模块中实现开启分页机制的代码，并初始化虚拟内存管理，包括建立二级页表并实现页表的切换，实现缺页中断处理程序、设计并实现了页淘汰机制和算法。

5.实现进程调度。本课题在时钟中断和分页机制的基础上实现了内核进程的调度、进程的切换。

6.页替换策略的改进算法本文实现了一种改进的页淘汰算法，通过实验数据实现两种算法的在缺页率、磁盘I/O 次数以及系统时间性能等方面的比较。

随着实验的不断深入，论文初步完成了具有内存管理功能的能够在裸机上直接运行的内核程序，实现了基本的物理内存管理和虚拟内存管理。研究分析了操作系统内核内存管理模块的实现方式以及关键技术。同时初步在分页机制的基础上成功进行了内核进程的切换。最后，针对目前内核页替换策略进行分析研究，提出改进的页淘汰算法。

5. 综述总结

本篇中文综述报告聚焦于计算机体系结构的重要技术之一 —— 虚拟内存技术的原理、应用和模拟实现上，重点介绍了虚拟内存提出的背景以及其概念、相关的硬件软件技术、和虚拟内存相关的页面替换算法以及各种用于解决虚拟内存性能问题的中间产物——快表、多级页表、cache等等相关概念。最后引用了来自南京邮电大学的硕士论文和来自中国石油大学的徐晨升的研究论文，介绍了前沿的基于FPGA的虚拟内存管理模块设计和高性能虚拟存储系统实现的研究。

随着计算机体系结构和技术的不断发展，对于虚拟内存的优化和改进也是研究的热点之一。一些先进的硬件技术，比如FPGA（现场可编程门阵列），被引入到虚拟内存管理模块的设计中。这些硬件在虚拟内存管理中的应用能够带来更高的效率和性能。例如，基于FPGA的虚拟内存管理模块能够更快速地执行地址映射和页表查找，从而加速内存访问的速度。除了硬件层面的创新，针对虚拟内存管理的软件优化也是研究的重点。页面替换算法的改进和优化，比如LRU（最近最少使用）算法、FIFO（先进先出）算法等，都旨在提高内存利用率和系统性能。此外，多级页表结构的设计和优化也是软件层面的重要改进，它可以降低页表的访问时间，并有效管理大容量的虚拟地址空间。前沿研究也探索了高性能虚拟存储系统的实现。这些系统不仅仅关注于单一计算机的内存管理，还考虑了分布式系统或云计算环境下的虚拟内存管理，以应对大规模数据处理和存储的挑战。这些研究为未来计算机系统的发展提供了有益的思路和技术支持。

南京邮电大学和中国石油大学等机构的研究论文为虚拟内存技术的探索提供了宝贵的案例和实践。特别是他们基于FPGA的虚拟内存管理模块设计，为硬件加速在虚拟内存管理中的应用提供了新的思路。这种基于硬件的优化为虚拟内存系统的性能提供了更大的空间，也为未来虚拟内存技术的发展暗示了一部分新的方向。通过虚拟内存这一项冰山一角的技术我们得以更加深入地窥探和研究到了宏观的计算机的组成原理和内部的体系结构。

**引用文献**

[[[18]](#endnote-17)]

1. [] 肖竟华, 陈岚. Linux 内存管理实现的分析与研究[J]. 计算机技术与发展, 2007, 17(2): 187-189 [↑](#endnote-ref-0)
2. [] 楼程辉, 孙守迁. Linux 下物理内存管理技术探讨[J]. 计算机应用研究, 2000, 17(8): 92-93. [↑](#endnote-ref-1)
3. [] 杨峰. 基于 Linux 内核的动态内存管理机制的实现[J]. 计算机工程，2010, 36(9): 85-86, 89 [↑](#endnote-ref-2)
4. [] 胡滨, 孙健力, 张永平, 等. 一种内存管理技术的研究与实现[J]. 计算机工程与设 计, 2007, 28(5): 1226-1228. [↑](#endnote-ref-3)
5. [] 谢长生, 刘志斌. Linux 2.6 内存管理研究[J]. 计算机应用研究, 2005, (3): 58-60 [↑](#endnote-ref-4)
6. [] Jiang L., Wang K., Zhao D.. Davram: Distributed virtual memory in user space[A]. 2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)[C]. IEEE, 2018: 344-347. [↑](#endnote-ref-5)
7. [] Zhang Y., Zhan J., Yang J., et al. Dynamic memory management for hybrid dram-nvm main memory systems[A]. 2016 13th International Conference on Embedded Software and Systems (ICESS)[C]. IEEE, 2016: 148-153 [↑](#endnote-ref-6)
8. [] Shin I.. Hot/cold clustering for page mapping in NAND flash memory[J]. IEEE Transactions on Consumer Electronics, 2011, 57(4): 1728-173 [↑](#endnote-ref-7)
9. [] 刘立圆,于松波,吕晓娴.Linux虚拟内存管理[J].数字技术与应用,2011(05):92.DOI:10.19695/j.cnki.cn12-1369.2011.05.063. [↑](#endnote-ref-8)
10. [] 刘海军.Windows操作系统虚拟内存管理[J].内蒙古电大学

    刊,2011,(05):75-76.DOI:10.16162/j.issn.1672-3473.2011.05.029 [↑](#endnote-ref-9)
11. [] 白洛,李俊奎译.深入理解linux虚拟内存管理.el Gorman.北京航空航天大学出版社. [↑](#endnote-ref-10)
12. [] 李鸿.虚拟内存简介[J].科技创新导报,2010,(09):23.DOI:10.16660/j.cnki.1674-098x.2010.09.097 [↑](#endnote-ref-11)
13. [] 胡希明,毛德操.linux内核源代码情景分析.江大学出版社. [↑](#endnote-ref-12)
14. [] 深入理解linux内核 （第二版）.国电力出版社. [↑](#endnote-ref-13)
15. [] 韩苏.基于FPGA的虚拟内存管理模块设计[D].南京邮电大学,2017. [↑](#endnote-ref-14)
16. [] 韩苏.基于FPGA的虚拟内存管理模块设计[D].南京邮电大学,2017. [↑](#endnote-ref-15)
17. [] 徐晨升. 高性能虚拟存储系统实现技术的研究[D].中国石油大学(华东),2022.DOI:10.27644/d.cnki.gsydu.2020.001200. [↑](#endnote-ref-16)
18. [↑](#endnote-ref-17)