



第0015讲 页表缓存(TLB)与巨型页

User Applications
O/S Services
Linux Kernel
Hardware Controllers



零声学院讲师: Vico老师



一、页表缓存(TLB)

二、巨型页



一、页表缓存(TLB)

处理器厂商在内存管理单元(MMU)里增加一个TLB(Translation Lookaside Buffer)的高速缓存, TLB直译为转译后备缓冲器, 也被翻译为页表缓存。

TLB为CPU的一种缓存, 由存储器管理单元用于改进虚拟地址到物理地址的转译速度。

TLB 用于缓存一部分标签页表条目。TLB可介于 CPU 和CPU缓存之间, 或在 CPU 缓存和主存之间, 这取决于缓存使用的是物理寻址或是虚拟寻址。



1、TLB表项格式

不同处理器架构的TLB表项的格式不同。ARM64处理器的每条TLB表项不仅包含虚拟地址和物理地址，也包含属性：内存类型、缓存策略、访问权限、地址空间标识符(ASID)及虚拟机标识符(VMID)。地址空间标识符区分不同进程的页表项，虚拟机标识符区分不同虚拟机的页表项。



2、TLB管理

若内核修改了可能缓存在TLB里面的页表项，那么内核必须负责使用旧的TLB表项失效，内核定义每种处理器架构必须实现的函数，具体可查阅源码分析如下：

```
arch > arm64 > include > asm > C tlbflush.h > ...
```

```
103
```

```
104 static inline void flush_tlb_all(void)
```

```
void flush_tlb_mm(struct mm_struct *mm)
```

```
void flush_tlb_range(struct vm_area_struct *vma,  
| unsigned long start, unsigned long end)
```

```
void flush_tlb_page(struct vm_area_struct *vma,  
| unsigned long uaddr)
```

```
void flush_tlb_kernel_range(unsigned long start, unsigned long end)
```



3、ARM64架构提供一条TLB失效指令

TLBI <type> <level> {IS} {,<Xt>}

- 字段<type>常见选项
- 字段<level>指定异常级别
- 字段{IS}表示内部共享，即多个核共享
- 字段Xt是X0-X31中的任何一个寄存器

案例分析：ARM64内核实现函数flush_tlb_all，用来使用所有核的所有TLB表项失效，

其源码如下：

```
104 static inline void flush_tlb_all(void)
105 {
106     dsb(ishst);
107     __tlbi(vmalle1is);
108     dsb(ish);
109     isb();
110 }
```



4、地址空间标识符

为了减少在进程切换时清空页表缓存的需要，ARM64处理器的页表缓存使用非全局（not global, nG）位区分内核和进程的页表项，使用地址空间标识符（Address Space Identifier, ASID）区分不同进程的页表项。

ARM64处理器ASID长度是由具体实现定义的，可以选择8位或者16位，寄存器ID_AA64MMFRO_EL1（AArch64内存模型特性寄存器0，AArch64 Memory Model Feature Register 0）的字段ASIDBits存放处理器支持的ASID长度。



5、虚拟机标识符

虚拟机里面运行的客户操作系统的虚拟地址换成物理地址分两个阶段：第1阶段把虚拟地址转换成中间物理地址，第2阶段把中间物理地址转换成物理地址。第1阶段转换由客户操作系统的内存控制，和非虚拟化的转换过程相同。第2阶段转换由虚拟机监控器控制，虚拟机监控器为每个虚拟机维护一个转换表，分配一个虚拟机标识符（Virtual Machine Identifier, VMID），寄存器VTTBR_EL2（虚拟化转换表基准寄存器，Virtualization Translation Table Base Register）存放当前虚拟机的阶段2转换表的物理地址。



二、巨型页



零声学院

www.0voice.com

一切只为渴望更优秀的你!

当运行内存需求量较大的应用程序时，如果使用长度为4KB的页，将会产生较多的TLB未命中和缺页异常，严重影响应用程序的性能。如果使用长度为2MB甚至更大的巨型页，可以大幅减少TLB未命中和缺页异常的数量，大幅提高应用程序的性能。这才是内核引入巨型页（Huge Page）的真正原因。

巨型页首先需要处理器能够支持，然后需要内核支持，内核有两种实现方式：

- 使用hugetlbfs伪文件系统实现巨型页；
- 透明巨型页。



1、处理器对巨型页的支持

ARM64处理器支持巨型页的方式有两种:

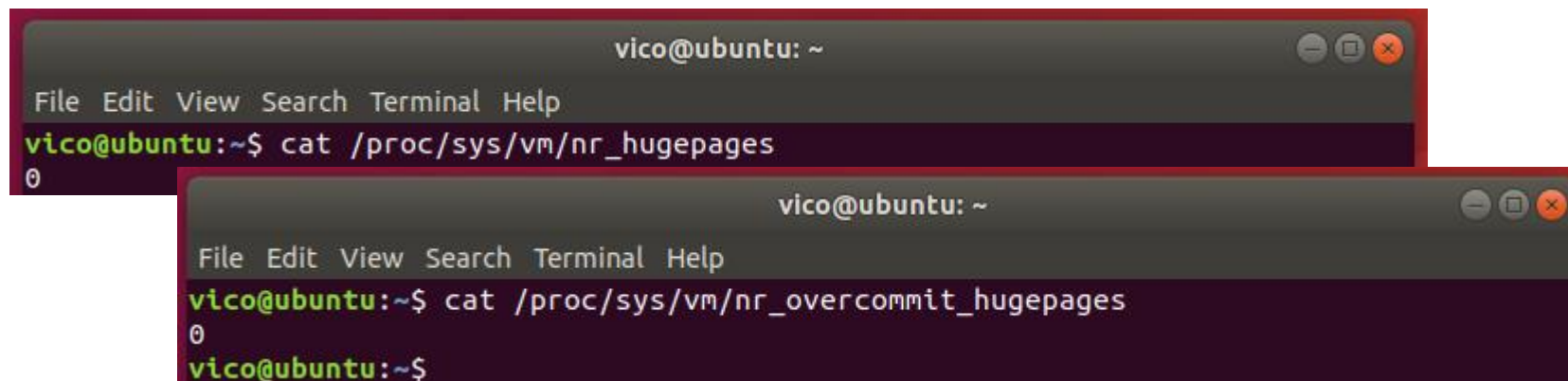
- 通过块描述符支持巨型页;
- 通过页/块描述符的连续位支持巨型页。



2、标准巨型页

通过文件 “cat /proc/sys/nr_hugepages” 指定巨型页池中永久巨型页的数量。

通过文件” cat /proc/sys/vm/nr_overcommit_hugepages “指定巨型页池中临时巨型页的数量，当永久巨型页用完的时候，可以从页分配器申请临时巨型页。



```
vico@ubuntu: ~  
File Edit View Search Terminal Help  
vico@ubuntu:~$ cat /proc/sys/vm/nr_hugepages  
0  
  
vico@ubuntu: ~  
File Edit View Search Terminal Help  
vico@ubuntu:~$ cat /proc/sys/vm/nr_overcommit_hugepages  
0  
vico@ubuntu:~$
```

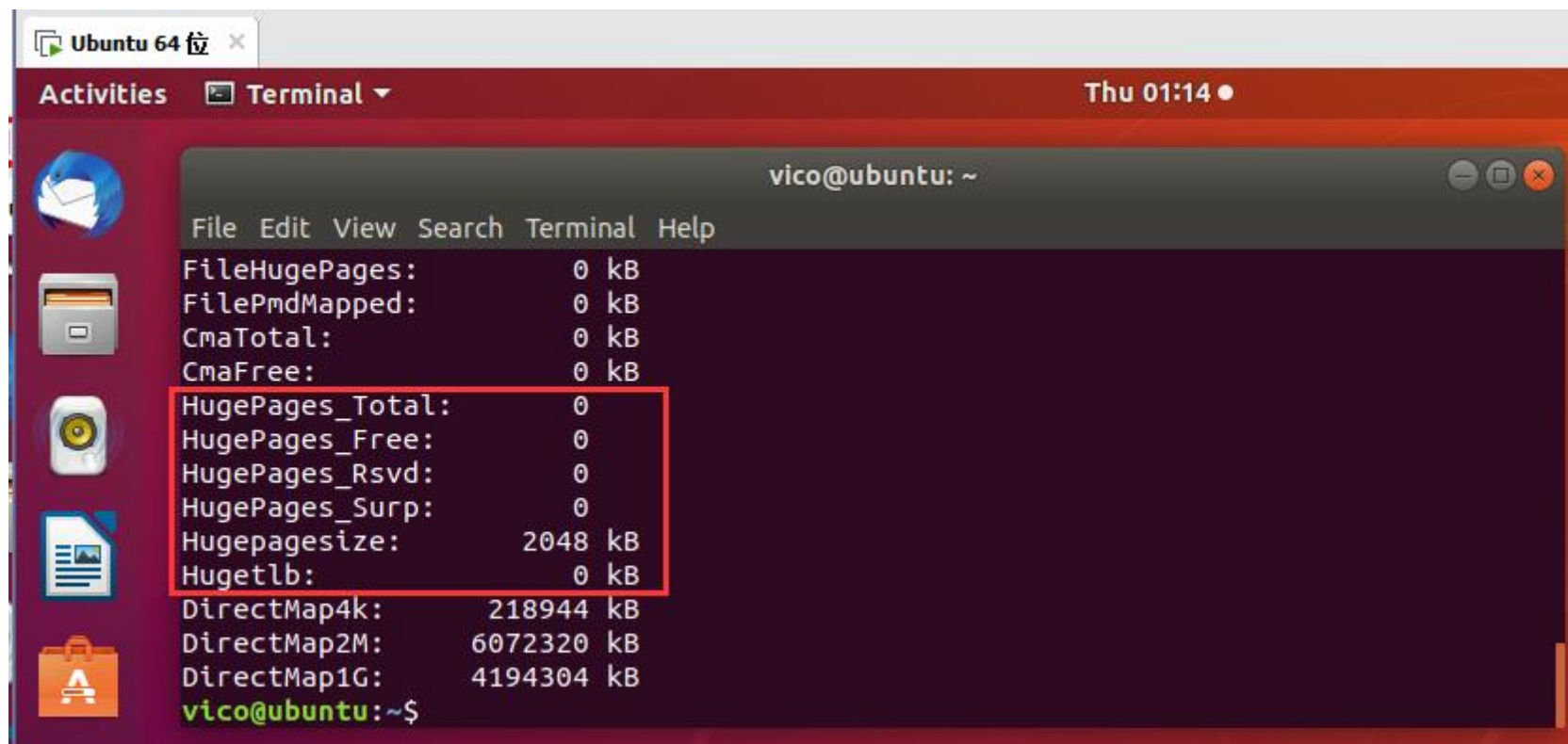
nr_hugepages是巨型页池的最小长度，

(nr_hugepages+nr_overcommit_hugepages) 是巨型页池的最大长度，这两个参数默认值都是0，至少要设置一个，否则分配巨型页会失败。



3、查看巨型页信息

通过文件执行: `cat /proc/meminfo`



```
File Edit View Search Terminal Help
FileHugePages:          0 kB
FilePmdMapped:         0 kB
CmaTotal:              0 kB
CmaFree:               0 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:         2048 kB
Hugetlb:               0 kB
DirectMap4k:          218944 kB
DirectMap2M:          6072320 kB
DirectMap1G:          4194304 kB
vico@ubuntu:~$
```



4、巨型页池

内核使用巨型页池管理巨型页。有的处理器架构支持多种巨型页长度，每种巨型页长度对应一个巨型页池，也有一个默认的巨型页长度，默认只创建巨型页长度是默认长度的巨型页池。



办学宗旨：一切只为渴望更优秀的你

办学愿景：让技术简单易懂