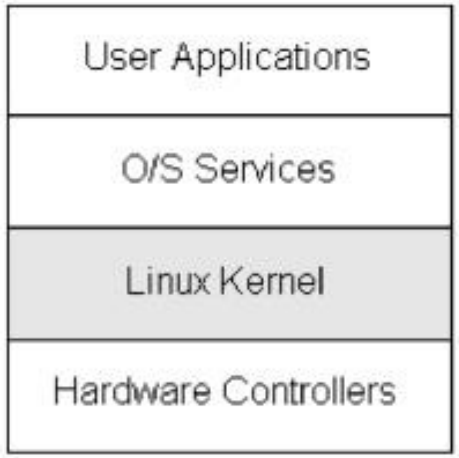
第0014讲 用户空间缺页异常



零声学院讲师：Vico老师

一、用户空间页错误异常

二、匿名页的缺页异常

三、文件页的缺页异常

在实际需要某个虚拟内存区域的数据之前，虚拟和物理内存之间的关联不会建立。如果进程访问的虚拟地址空间部分尚未与页帧关联，处理器自动地引发一个缺页异常，内核必须处理此异常。

缺页处理的实现因处理器的不同而有所不同。由于CPU采用了不同的内存管理概念，生成缺页异常的细节也不太相同。因此，缺页异常的处理例程在内核代码中位于特定于体系结构的部分。

一、用户空间页错误异常

在结束对缺页异常的特定于体系结构的分析之后，确认异常是在允许的地址触发，内核必须确定将所需数据读取到物理内存的适当方法。该任务委托给handle_mm_fault，它不依赖于底层体系结构，而是在内存管理的框架下、独立于系统而实现。该函数确认在各级页目录中，通向对应于异常地址的页表项的各个页目录项都存在。

如果页错误异常处理程序确认虚拟地址属于分配给进程的虚拟内存区域，并且虚拟内存区域授予触发页错误异常的访问权限，就会运行到函数handle_mm_fault。

具体源码分析如下：

mm > C memory.c > __handle_mm_fault(vm_area_struct *, unsigned long, unsigned int)

```
3749  static int __handle_mm_fault(struct vm_area_struct *vma, unsigned long address,
3750          unsigned int flags)
```

在处理__handle_mm_fault函数过程，最后要处理直接页表handle_pte_fault函数，其执行流程源码分析如下：

二、匿名页的缺页异常

什么情况下会发生匿名页缺页异常呢?

- 函数的局部变量比较大，或者函数调用的层次比较深，导致当前栈不够用，需要扩大栈；
- 进程调用malloc，从堆申请了内存块，只分配虚拟内存区域，还没有映射到物理页，第一次访问时触发缺页异常。
- 进程直接调用mmap，创建匿名的内存映射，只分配了虚拟内存区域，还没有映射到物理页，第一次访问时触发缺页异常。

函数do_anonymous_page处理私有匿名页的缺页异常，执行流程及源码分析如下：

fault.c | memory.c 4

mm > memory.c > do_anonymous_page(vm_fault *)

```
2859    * but allow concurrent faults), and pte mapped but not yet locked.
2860    * We return with mmap_sem still held, but pte unmapped and unlocked.
2861    */
2862   static int do_anonymous_page(struct vm_fault *vmf)
2863   {
2864       struct vm_area_struct *vma = vmf->vma;
2865       struct mem_cgroup *memcg;
2866       struct page *page;
2867       pte_t entry;
2868
```

三、文件页的缺页异常

何时会触发文件页的缺页异常呢?

- 启动程序的时候，内核为程序的代码段和数据段创建私有的文件映射，映射到进程的虚拟地址空间，第一次访问的时候触发文件页的缺页异常。
- 进程使用mmap创建文件映射，把文件的一个区间映射到进程的虚拟地址空间，第一次访问的时候触发文件页的缺页异常。

函数do_fault处理文件页和共享匿名页的缺页异常，执行流程及源码分析如下：

mm > C memory.c > __do_fault(vm_fault *)

```
2967    */
2968   static int __do_fault(struct vm_fault *vmf)
2969   {
2970       struct vm_area_struct *vma = vmf->vma;
2971       int ret;
2972
```

1、处理读文件页错误，具体处理读文件页错误的方法如下：

- 把文件页从存储设备上的文件系统读到文件的缓存 （每个文件有一个缓存，因为以页为单位，所以称为页缓存）中。
- 设置进程的页表项，把虚拟页映射到文件的页缓存的物理页。

函数do_read_fault处理读文件页错误，执行流程及源码分析如下：

fault.c　memory.c 4 ×

mm > memory.c > do_read_fault(vm_fault *)

```
3358
3359  static int do_read_fault(struct vm_fault *vmf)
3360  {
3361      struct vm_area_struct *vma = vmf->vma;
3362      int ret = 0;
3363
```

给定一个虚拟内存区域vma，函数filemap_fault读文件页的方法如下：

- 根据vma->vm_file得到文件的打开实例file;
- 根据file->f_mapping得到文件的地址空间mapping;
- 使用地址空间操作集合中的readpage方法（mapping->a_ops->readpage）把文件页读到内存中。

函数finish_fault负责设备项表项，把主要工作委托给函数alloc_set_pte，执行流程及源码分析如下：

mm > C memory.c > ...

```
3159
3160  int alloc_set_pte(struct vm_fault *vmf, struct mem_cgroup *memcg,
3161          struct page *page)
3162  {
3163      struct vm_area_struct *vma = vmf->vma;
```

2、处理写私有文件页错误的方法：

- 把文件页从存储设备上的文件系统读到文件的页缓存中；
- 执行写时复制，为文件的页缓存中的物理页创建一个副本，这个副本是进程的私有匿名页，和文件脱离系统，修改副本不会导致文件变化；
- 设备进程的页表项，把虚拟页映射到副本；

函数do_cow_fault处理写私有文件页错误，执行流程及源码分析如下：

3、处理写共享文件页错误的方法如下：

- 把文件页从存储设备上的文件系统读到文件的页缓存中；
- 设备进程的页表项，把虚拟页映射到文件的页缓存中的物理页。

函数do_shared_fault处理写共享文件页错误，执行流程及源码分析如下：

mm > C memory.c > do_shared_fault(vm_fault *)

```
3426
3427  static int do_shared_fault(struct vm_fault *vmf)
3428  {
3429      struct vm_area_struct *vma = vmf->vma;
3430      int ret, tmp;
```

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