Linux内核源码学习

--浅谈内存管理

郭帆

学习步骤

- 理论调研
  - Linux内核的整体结构
  - 每个子系统的功能及子系统之间的联系
  - 子系统的组成模块及对应的详细功能和基本原理
- 源码阅读（针对特定子系统）
  - 模块分解（找到相应的源文件）
  - 熟悉主要数据结构
  - 阅读源码

第一阶段：理论调研

- 目标
  - 熟悉Linux内核的整体架构
  - 熟悉各个子系统的功能及相互之间的联系
  - 熟悉子系统内部的模块组成及功能
- 方法
  - 阅读经典书籍：《Linux内核设计与实现》、《深入理解Linux内核》等
  - 博客、论坛等

.1.1 Linux内核架构

- Linux内核特点：模块化、分层、耦合度高

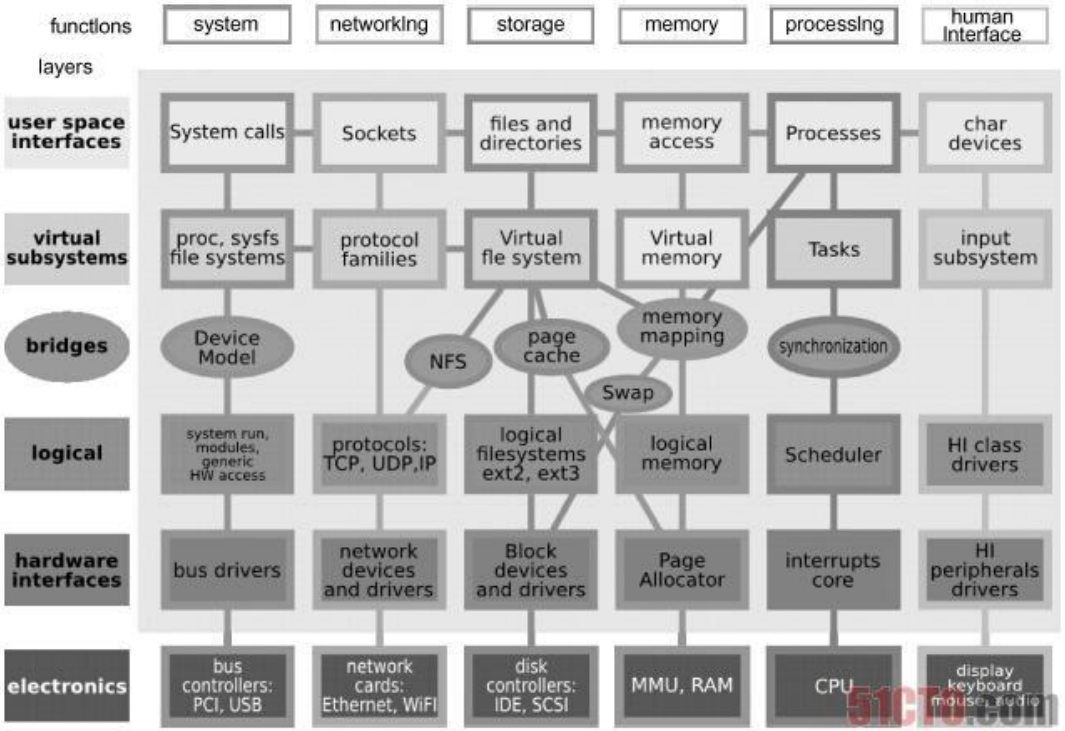
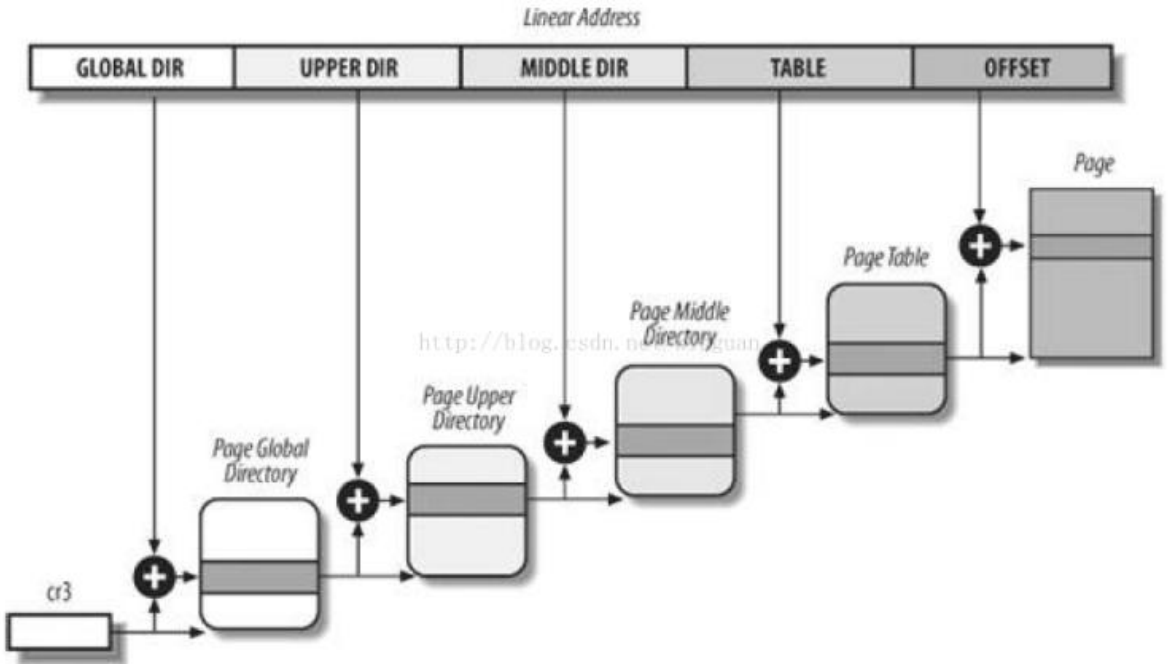


图 3-3　Linux 内核整体框架

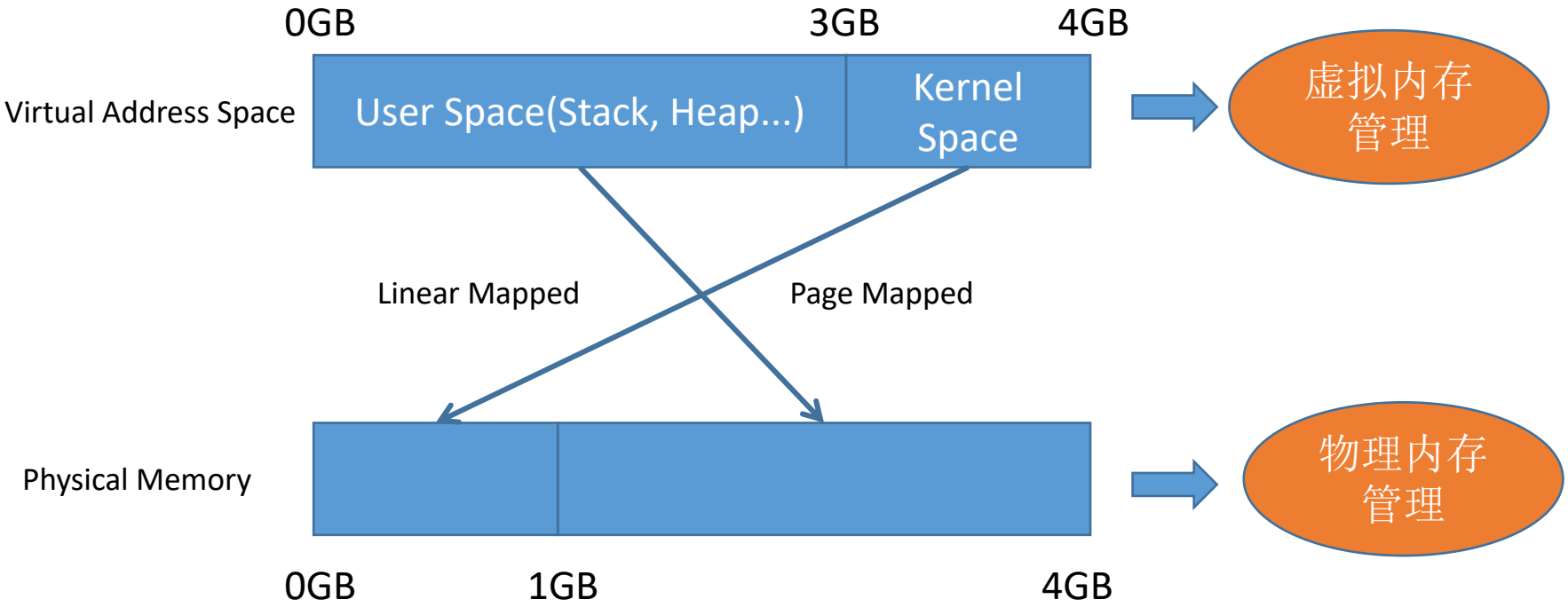
1.2 Linux内存管理--机制

- Linux采用页式内存管理
  - 应用给出的地址是虚拟地址，需要通过查询页表转换成物理内存地址

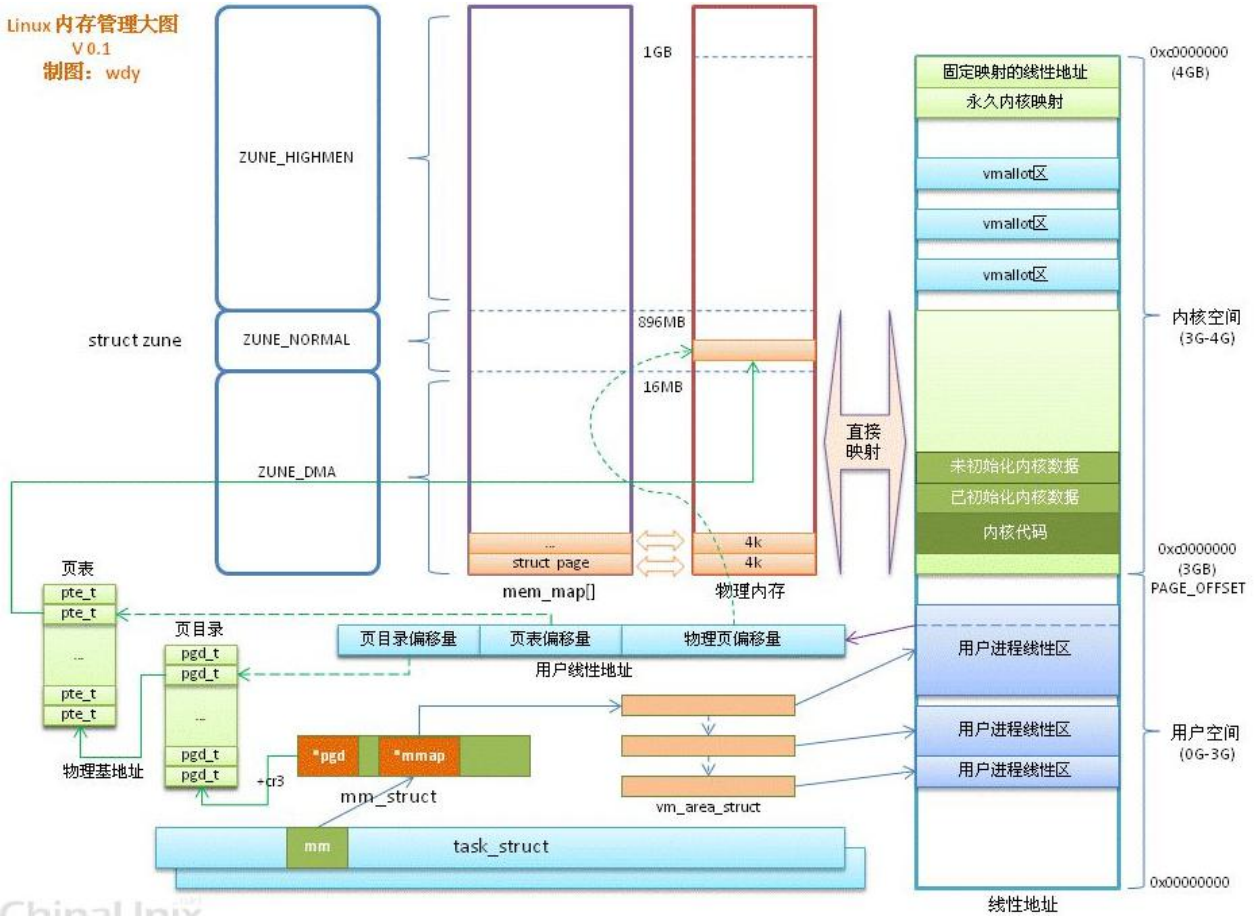


1.2 内存管理子系统--功能

- 功能：提供对虚拟内存空间和物理内存的管理



1.2 内存管理子系统--架构



1.3 内存管理子系统--子模块

- 物理内存管理
  - 物理内存页分配、回收、迁移、交换
- 虚拟内存管理
  - 虚拟内存分配、内存映射、页表管理、虚拟空间管理
- 页缓存管理
  - 页缓存的分配、管理、预取、访问
- 异常管理
  - 内存相关的异常管理机制

1.3 子模块分析--分配与映射

- C语言中的malloc在内核中包含两步
  - 分配虚拟地址区间
  - 访问虚拟地址时分配物理页，建立页表映射
- malloc(<128KB)操作的执行流程
  - 用户态
    - 应用调用malloc申请内存->malloc从空闲堆空间缓冲区链表申请内存，若有内存则返回，若不足则调用sys_brk()扩充进程的堆空间
  - 内核态
    1. 在mm_struct中的堆上界brk延伸到newbrk：即申请一块vma，vma.start=brk vma.end=newbrk
    2. 当进程访问其中某个虚拟地址，触发缺页中断再申请物理页并进行映射

第二阶段：源码阅读

- 目标
  - 熟悉每个源文件的功能
  - 熟悉主要数据结构
  - 详细分析特定模块的源码
- 方法：
  - 从点切入，追踪调用路径
  - 用调试手段分析源码（printk、kgdb、 dump_stack）
  - 仔细阅读注释，参考源码解读

2.1 源文件分类

- 物理内存管理
  - 内存分配
    - page_alloc：伙伴分配系统
    - debug-pagealloc：伙伴分配系统辅助函数
    - slab、slub、slob：基于伙伴系统之上的内存分配算法
    - huge_memory、hugetlb：大页模式的支持（分配）
    - slob：用于嵌入式的简单内存分配器
    - slab_common：slab和slub共用的功能函数
    - bootmem：启动期间的内存分配器
    - memblock：初始化期间物理内存块的管理，是对bootmem的改进替代
    - mmzone：management codes for pgdats, zones and page flags
    - mempool：内存池，内存资源极度紧张情况下使用，可保证无死锁、内存分配不会失败
    - …

2.1 源文件分类

- 物理内存管理
  - 内存回收与迁移
    - vmscan：内存回收算法
    - compaction：内存碎片整理（调用migrate）
    - migrate：Memory Migration functionality
  - 内存交换
    - swapfile：内存页交换到硬盘空间
    - swap：对物理页换入换出的支持函数

2.1 源文件分类

- 虚拟内存管理
  - 页表（映射）建立与管理
    - memory：页表映射管理（MMU）
    - huge_memory、hugetlb：大页模式的支持（映射）
    - rmap：physical to virtual reverse mappings
    - mmap：虚拟空间内存管理（struct mm_struct / struct vm_area_struct）
    - mempolicy：内存映射策略
    - vmalloc：虚拟内存分配
    - bounce：高端内存的临时映射访问机制
    - highmem：High memory handling common code and variables
    - pagewalk：页表遍历函数
    - pgtable-generic：页表帮助函数
    - mmu_context：任务的内存地址空间切换
    - init-mm：初始化进程的内存地址空间

2.2 主要数据结构分析

- 与物理内存管理相关的数据结构
  - 内存节点描述符：struct pglist_data
    - 记录每个内存（CPU）节点的内存情况，例如包含几个内存区域(zone)，以及node内的页框数
  - 内存区域描述符：struct zone
    - 记录单个内存节点内部不同用途的内存区域，例如ZONE_DMA、ZONE_NORMAL、ZONE_HIGHMEM
  - 页描述符：struct page
    - 保存在全局数组mem_map中，记录物理页的属性与状态，例如该页的引用计数，映射计数，是否为脏，是否锁定等

2.2 主要数据结构

- 与虚拟内存管理相关的数据结构
  - struct mm_strcut 用来描述一个进程的虚拟地址空间，保存在进程的task_struct 结构中，包含页目录基址，堆栈信息，该进程的虚拟内存区（vma）信息
  - struct vm_area_struct用来描述一个连续的虚拟地址区间，用来管理进程的不同虚拟内存区域，比如堆、栈、代码区、数据区、各种映射区、等等

2.3 源码阅读（内存分配）

- 虚拟地址空间分配
  - Malloc()调用sys_brk()扩充堆
  - sys_brk()调用do_brk()扩充堆的虚拟区间(vma)

- 建立虚拟地址到物理地址的页表映射
  - 访问虚拟地址触发缺页中断
  - 中断处理程序调用do_page_fault()处理该异常
  - do_page_fault()区分异常类型，调用handle_mm_fault()
  - handle_mm_fault()找到虚拟地址对应的页表项，并调用handle_pte_fault()
  - handle_pte_fault()进一步区分异常类型，调用do_anonymous_page()分配物理页并完成pte到物理页的映射

2.3 源码阅读（虚拟空间分配）

- 当堆空间不足时，Malloc()调用sys_brk()扩充堆

```
unsigned long sys_brk(unsigned long brk)
{
    //...
    if (oldbrk == newbrk)//如果新边界与旧边界相等，不用进行空间的伸缩操作
        goto set_brk;

    if (brk <= mm->brk) {//如果新边界比现在的边界要小，那说明要执行收缩操作
        if (!do_munmap(mm, newbrk, oldbrk-newbrk))
            goto set_brk;
        goto out;
    }
    //...
    if (do_brk(oldbrk, newbrk-oldbrk) != oldbrk)//扩大虚拟区间，建立相应vma
        goto out;
    //...
}
```

2.3 源码阅读（虚拟空间分配）

- sys_brk()调用do_brk()扩充堆的虚拟区间(vma)

```
unsigned long do_brk(unsigned long addr, unsigned long len)
{
    ...
 munmap_back:
    //find_vma_prepare找到前一个VMA区
    vma = find_vma_prepare(mm, addr, &prev, &rb_link, &rb_parent);
    ...
    //判断是否可以合并，如果可以合并，就将基合并为一个VMA区
    vma = vma_merge(mm, prev, addr, addr + len, flags,
                    NULL, NULL, pgoff, NULL);
    if (vma)
        goto out;
    //不可以合并，新建一个VMA /*分配映射虚拟区空间*/
    vma = kmem_cache_zalloc(vm_area_cachep, GFP_KERNEL);
    ...
    //设值VMA的值
    INIT_LIST_HEAD(&vma->anon_vma_chain);
    vma->vm_mm = mm;
    vma->vm_start = addr;
    vma->vm_end = addr + len;
    ...
    //将新分配的VMA插入到进程的VMA链表
    vma_link(mm, vma, prev, rb_link, rb_parent);
out:
    ...
}
```

2.3 源码阅读（建立物理页映射）

- 经过上面过程，malloc()返回虚拟地址，如果用户进程访问该地址，则触发缺页中断，中断处理程序调用do_page_fault()处理该异常

- do_page_fault()会区分出引发缺页的两种情况：
  - 由编程错误引发异常
  - 由进程地址空间中还未分配物理内存的虚拟地址引发
    - 内核空间引发的缺页异常
    - 用户空间引发的缺页异常
      - 通过handle_mm_fault完成

2.3 源码阅读（建立物理页映射）

- handle_mm_fault()找到虚拟地址对应的页表项，并调用handle_pte_fault()

```
int handle_mm_fault(struct mm_struct *mm, struct vm_area_struct *vma,
        unsigned long address, unsigned int flags)
{
    //通过进程内存描述符mm，找到全局页目录项(pgd)，再通过页目录一级一级找到最终的页表项pte
    if (unlikely(is_vm_hugetlb_page(vma)))
        return hugetlb_fault(mm, vma, address, flags);

    pgd = pgd_offset(mm, address);
    pud = pud_alloc(mm, pgd, address);
    if (!pud)
        return VM_FAULT_OOM;
    pmd = pmd_alloc(mm, pud, address);
    if (!pmd)
        return VM_FAULT_OOM;
    if (unlikely(pmd_none(*pmd)) && __pte_alloc(mm, vma, pmd, address))
        return VM_FAULT_OOM;
    if (unlikely(pmd_trans_huge(*pmd)))
        return 0;
    pte = pte_offset_map(pmd, address);
    //申请物理页并完成页表项的映射
    return handle_pte_fault(mm, vma, address, pte, pmd, flags);
}
```

2.3 源码阅读（建立物理页映射）

- handle_pte_fault又将异常分为两大类
  - 请求调页：被访问的页框不再主存中，那么此时必须分配一个页框。
    - 如果页表项为空（pte_none(entry)），那么必须分配页框。
      - 如果vma->vm_ops不为空，那么这种情况属于基于文件的内存映射，它调用do_linear_fault()进行分配物理页框。
      - 否则，内核将调用针对匿名映射分配物理页框的函数do_anonymous_page()
    - 如果检测出该页表项为非线性映射（pte_file(entry)），则调用do_nonlinear_fault()分配物理页
    - 如果页框事先被分配，但是此刻已经由主存换出到了外存，则调用do_swap_page()完成页框分配
  - 写时复制：被访问的页存在，但是该页是只读的，内核需要对该页进行写操作，需要分配页框并复制内容

2.3 源码阅读（建立物理页映射）

- do_anonymous_page完成物理页分配与页表映射

```
static int do_anonymous_page(struct mm_struct *mm, struct vm_area_struct *vma,
        unsigned long address, pte_t *page_table, pmd_t *pmd,
        unsigned int flags)
{
    page = alloc_zeroed_user_highpage_movable(vma, address);//分配物理页
    if (!page)
        goto oom;
    __SetPageUptodate(page);

    if (mem_cgroup_newpage_charge(page, mm, GFP_KERNEL))
        goto oom_free_page;

    entry = mk_pte(page, vma->vm_page_prot);
    if (vma->vm_flags & VM_WRITE)
        entry = pte_mkwrite(pte_mkdirty(entry));

    page_table = pte_offset_map_lock(mm, pmd, address, &ptl);
    if (!pte_none(*page_table))
        goto release;

    inc_mm_counter_fast(mm, MM_ANONPAGES);
    page_add_new_anon_rmap(page, vma, address);//建立反向映射
setpte:
    set_pte_at(mm, address, page_table, entry);//将pte指向新分配的页表项，完成映射
    ...
}
```

Linux源码学习--总结

- 两阶段
  - 理论学习
    - 熟悉整体架构和框架
    - 熟悉各部分代码的大概作用与实现原理
  - 源码阅读
    - 做好准备工作：熟悉数据结构、对源文件进行分类
    - 从点出发阅读源码：从单个功能（模块）出发层层解析代码。
    - 仔细阅读注释，参考源码解读

Thanks!