

# 华中科技大学

## 本科生课程设计报告

课 程： 操作系统原理设计与实践

题 目： 阻塞设备驱动开发与内核调试

院 系： 软件学院

专业班级： 软件工程 2102 班

学 号： U202117282

姓 名： 赵展

2023 年 06 月 18 日

## 1 任务一 阻塞设备驱动开发

### 1.1 实验任务概述

编写驱动程序并支持多个测试程序对内核缓冲区的有序读/写

实验目的：

- (1) 理解设备驱动机制和开发技术
- (2) 掌握内核同步机制概念和机制
- (3) 掌握内核开发和调试技术

### 1.2 实验设计思路

首先要编写驱动程序，在程序中设置缓冲区的大小，以及阻塞处理和缓冲区的读写操作等，同时对设备进行自定义配置：

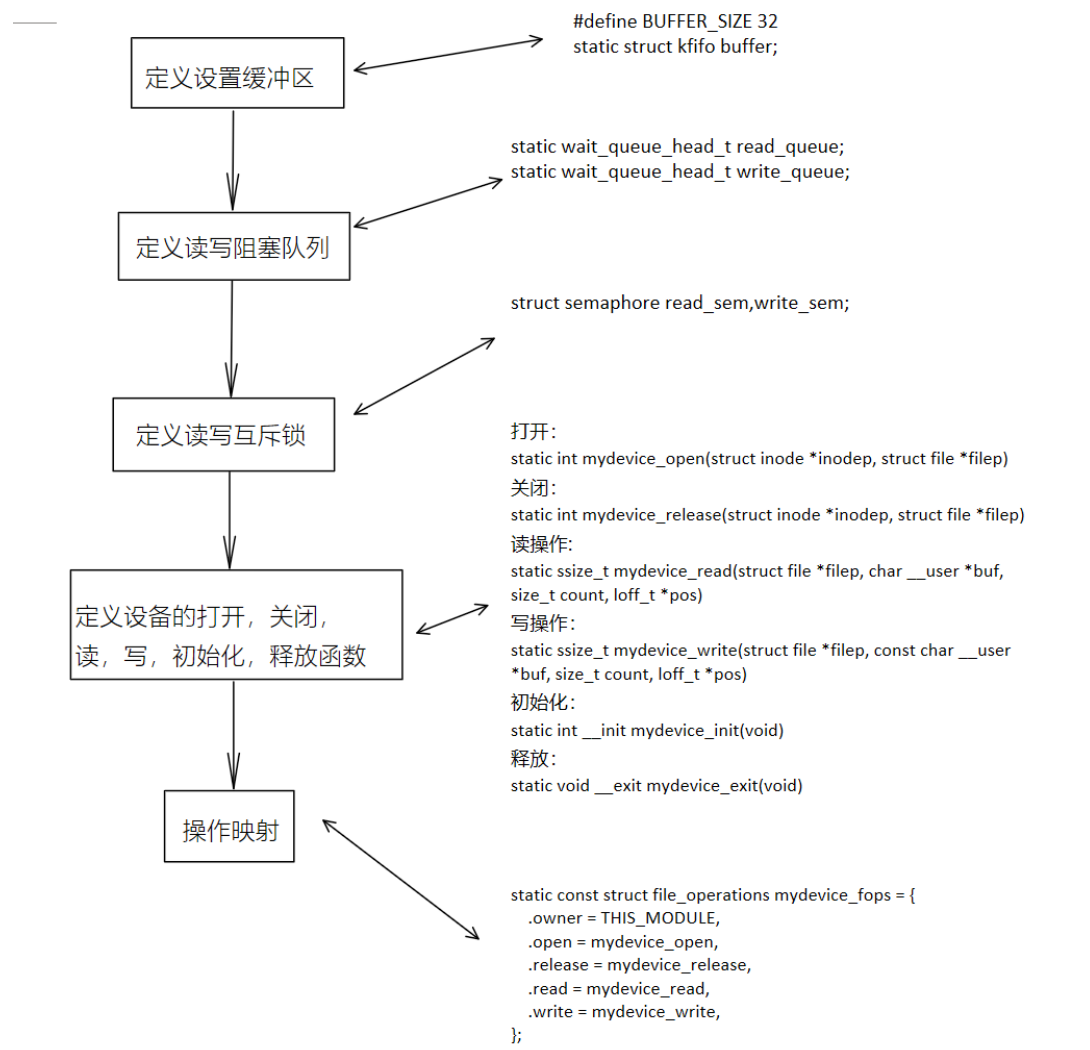


图 1: 设备驱动程序 driver 结构

如图 1 为整个设备驱动程序的代码结构,设备的打开和关闭只需 printk 相关信息,下面就是具体实现设备 kfifo 缓冲区的读写操作:

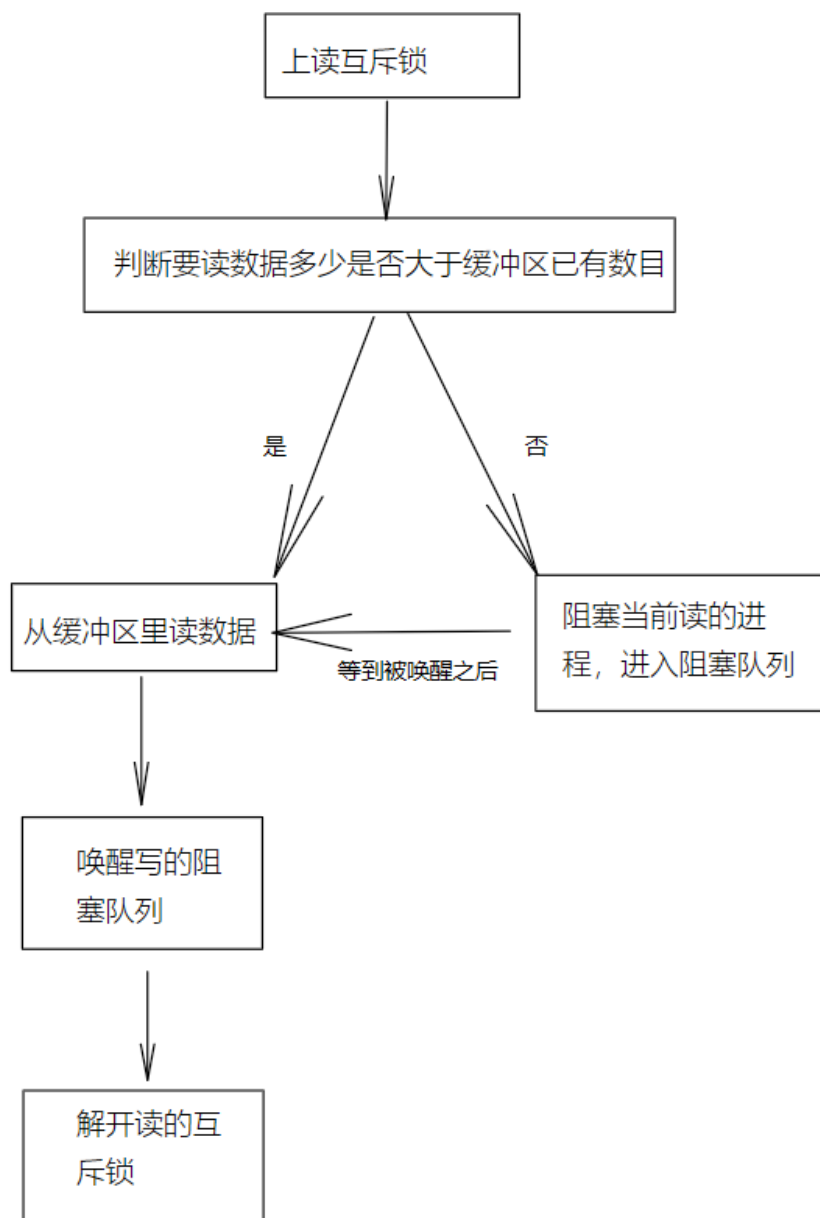


图 2：设备驱动程序读操作函数思路

如图 2 为设备驱动程序的读操作函数的设计思路，包括是否阻塞的条件判断以及对写的阻塞队列的操作。

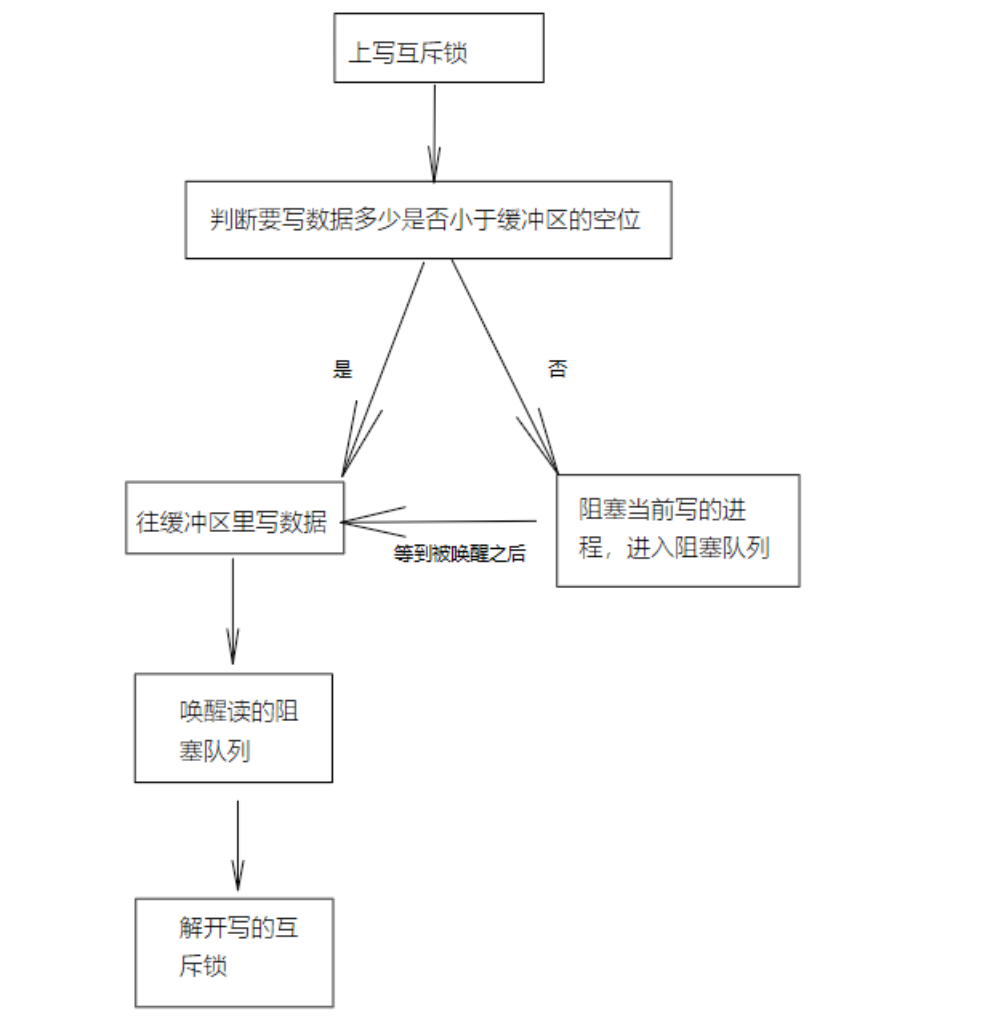


图 3：设备驱动程序写操作函数思路

如图 3 为设备驱动程序的写操作函数的设计思路，与读操作函数类似。

### 1.3 程序的重点/难点/核心技术分析

#### 1. 重点：

每次读写操作后都有唤醒相反的写读阻塞对列，而不是唤醒一个进程了，这里使用到 `wake_up_interruptible()` 函数正好是唤醒的一个队列

#### 2. 难点：

```
ret = wait_event_interruptible(read_queue, !(kfifo_len(&buffer) < count));
```

```
if (ret) {
    return ret;
}
```

图 4: 判断是否阻塞的读操作代码 (初始的错误写法)

```
while(kfifo_len(&buffer) < count){
    ret = wait_event_interruptible(read_queue, !(kfifo_len(&buffer) < count));
}
if (ret) {
    return ret;
}
```

图 5: 判断是否阻塞的读操作代码 (修改之后的写法)

以读操作为例, 使用如图 4 的方式判断是否可以读数据, 不可以就阻塞当前进程了, 而每次写操作之后都会唤醒读的阻塞队列, 但是经实验发现, 唤醒之后, 被唤醒的读操作对应进程不会再次判断是否满足可读的条件, 比如, 初始缓冲区为空, 一个读操作进程读 10 个数, 另一个写操作进程写 1 个数, 一开始读的进程运行时阻塞, 进入阻塞队列。这时写的进程运行后写了一个数同时唤醒读进程队列, 读 10 个数的进程被唤醒, 然而缓冲区内的数据小于 10, 读进程应该继续阻塞, 但是现实是读进程被唤醒后没有再判断阻塞条件, 因此我改成了如图 5 所示的样子, 加了一个循环就可以解决这个问题, 写操作与之类似。

## 1.4 运行和测试过程

```
fd = open(DEVICE_NAME, O_RDWR);
if (fd < 0) {
    perror("open");
    exit(EXIT_FAILURE);
}
write(fd, "1234567890", 10);
```

图 6: write10 测试源码

```
fd = open(DEVICE_NAME, O_RDWR);
if (fd < 0) {
    perror("open");
    exit(EXIT_FAILURE);
}
int n;
// 读取数据
read(fd, buffer, 16);
printf("%s\n", buffer);
```

图 7: read16 测试源码

```
zhaozhan@zhaozhan-virtual-machine:~/KFIFotest$ sudo ./read16
[sudo] password for zhaozhan:
```

图 8: 一开始运行 read16 阻塞情况

```
zhaozhan@zhaozhan-virtual-machine:~/KFIF0test$ sudo ./write10
zhaozhan@zhaozhan-virtual-machine:~/KFIF0test$ sudo ./write10
zhaozhan@zhaozhan-virtual-machine:~/KFIF0test$
```

图 9: 两次运行 write10

```
zhaozhan@zhaozhan-virtual-machine:~/KFIF0test$ sudo ./read16
1234567890123456
```

图 10: read16 被唤醒

```
[16621.927643] mydevice: opening device
[16621.927660] mydevice: process 7051 (read16) going to sleep
[16624.655362] mydevice: opening device
[16624.655369] mydevice: put 10 bytes data
[16624.655422] mydevice: closing device
[16625.434381] mydevice: opening device
[16625.434388] mydevice: put 10 bytes data
[16625.434460] mydevice: closing device
[16625.434523] mydevice: read process 7051 (read16) woken up
[16625.434529] mydevice: get data: 16
[16625.434675] mydevice: closing device
zhaozhan@zhaozhan-virtual-machine:~/KFIF0test$
```

图 11: 日志显示读写情况

如图 6, 7 是一次读 16 个数据和一次写 10 个数据的源码, 写的数字全为数字且按照顺序, 如图 8 初始缓冲区为空时, 先运行 read16 后阻塞, 之后如图 9, 两次运行 write10 之后缓冲区数据为 12345678901234567890, 多于 16 个, 所以 read16 进程被唤醒, 读取前 16 个数据如图 10 所示。图 11 是日志中查看读写情况, 输入命令 `sudo dmesg | grep mydevice` 后查看到的 `printk` 打印的信息, 其中就有被唤醒的进程的 id 号。

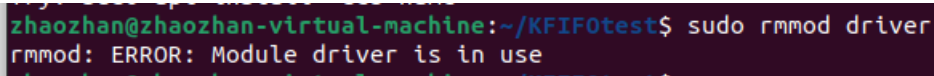
## 1.5 实验心得和建议

### 1. 环境配置:

刚在 VMware 中装的 Ubuntu 系统发现整个桌面很小, `sudo apt-get install open-vm-tools`, 安装一个包就行了; 同时还出现了在主机上复制的内容在虚拟机中没法粘贴, 上网查过之后也是下载 `sudo apt-get install`

open-vm-tools-desktop -y 之后再重启虚拟机之后问题解决

## 2. 命令操作:



```
zhaozhan@zhaozhan-virtual-machine:~/KFIF0test$ sudo rmmod driver
rmmod: ERROR: Module driver is in use
```

图 12: 无法删除驱动程序

如果一个进程阻塞时关闭终端，杀死该进程，这时重新修改驱动程序时会没法删除，这是因为被杀死的进程打开设备后并没有关闭，导致设备一直在使用中，没法删除，我发现重启虚拟机可以解决该问题。

## 3. 编译错误:

make 总是报错:

module verification failed: signature and/or required key missing - tainting kernel

在 makefile 文件中加入 CONFIG\_MODULE\_SIG=n 表示内核不支持启用模块签名检查，问题解决

## 4. 运行错误:

见 1.3 板块中的难点

## 1.6 学习和编程实现参考网址、

[2021 级-课设说明.pdf](#)

[\(165 条消息\) linux kernel insmod 模块出现的两个错误以及解决方案 module verification failed: signature and/or requi 悟空明镜的博客-CSDN 博客](#)

[\(165 条消息\) 驱动程序开发的步骤 多个驱动程序测试程序怎么写 \\_ustcjin 的博客-CSDN 博客](#)

[\(165 条消息\) 华中科技大学操作系统实验课 实验四 Elsa 的迷弟的博客-CSDN 博客](#)



## 2 任务二 理解、跟踪和调试页式虚拟内存管理机制

### 2.1 实验任务概述

1. 阅读内核源代码，跟踪页式内存管理模块，理解哪些函数与页面/页框的管理有关，哪个函数分配页框？哪个函数处理缺页？哪个函数调整页表？画出相关函数的执行流程图。
2. 修改内核，检测特定应用程序调入内存后占用了哪些页框，`printk` 出来；页框何时被释放的，`printk` 出来。特定应用程序可以指定名字，例：HustSSE(可修改)，只要该程序运行，则内核 `printk` 相应信息。

### 2.2 理解、跟踪和调试页式虚拟内存管理机制

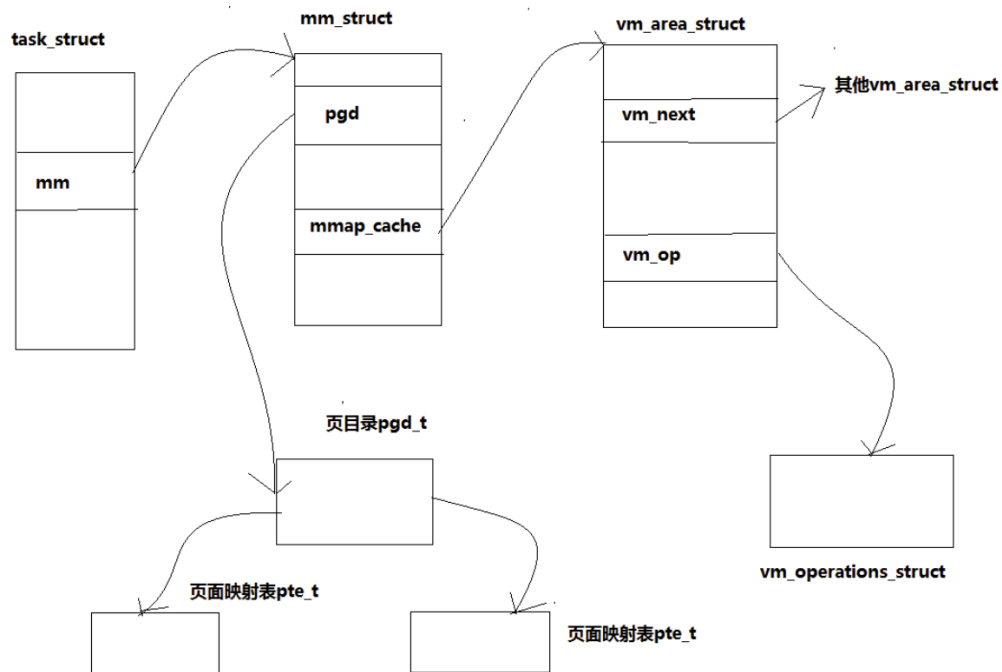


图 13: 进程虚存管理示意图

如图 13 是进程虚拟内存的管理相关数据结构示意图，在 linux 内核中，`vm_area_struct` 数据结构是对虚存空间的抽象是一个重要的数据结构。每个进程中的 `task_struct` 结构中，有一个指针指向该进程的 `mm_struct` 结构，`mm_struct` 数据结构是进程整个用户空间的抽象。一个虚拟地址有相应的虚存区间存在，并不保证该地址所在页面已经映射到某一个物理（内存或盘区）页面，更不保证该页面就在内存中。当一个未经映射的页面受到访问时，就会产生一个 `page fault` 异常（也称缺页异常、缺页中断）。

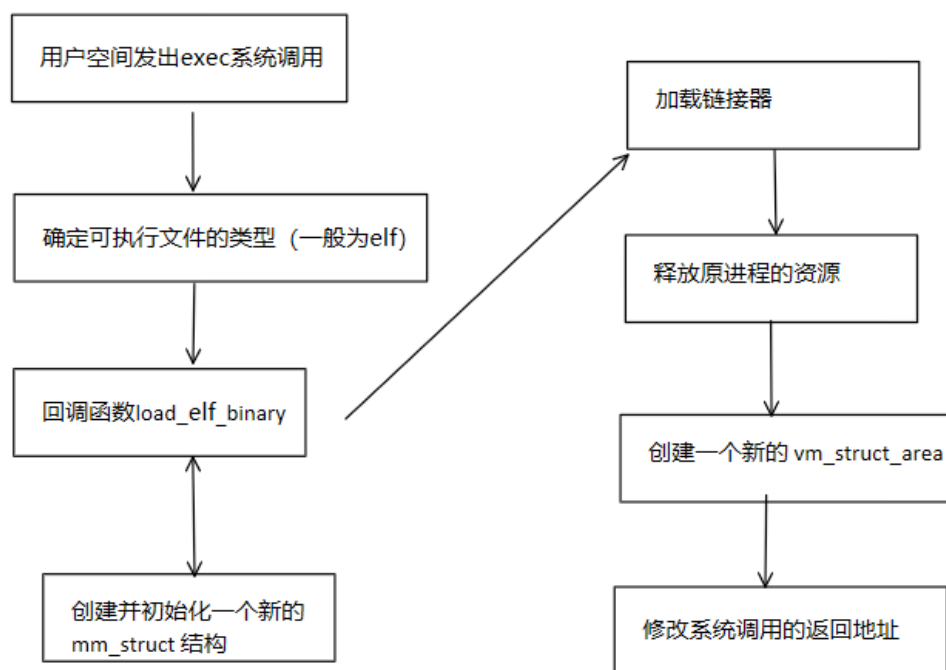


图 16: 可执行文件加载过程

如图 16 是可执行文件的加载过程，需要注意的是这个过程只是建立了映射，但是并不分配实际的物理内存，只有在访问时才会触发缺页异常分配对应的物理页面。了解运行的时候分配物理页面的过程之前要先了解物理内存的结构。

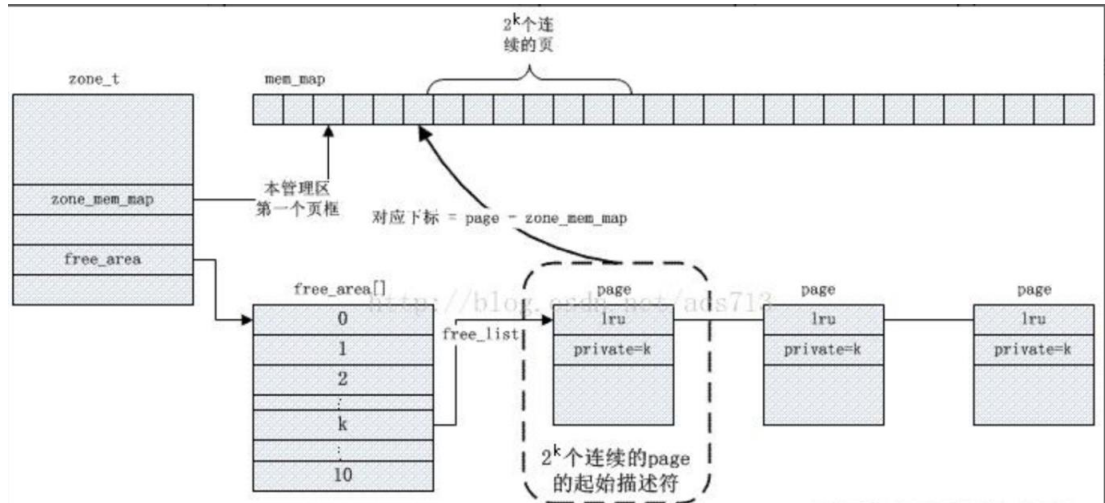


图 14: Linux 物理内存管理

如图 14 为 Linux 物理内存的管理示意图，Linux 系统使用伙伴系统算法以减少外部碎片，把所有的空闲页框分组为  $\text{MAX\_ORDER}$  (ICE 为 11) 个块链表，每个块链表分别包含大小为  $2^n$  ( $n$  为 0~10 的整数) 个连续的页框。比如说我们要申请一个  $b$  阶大小的页块，那么系统会直接在  $b$  阶块中查找看这个链表是否为空，如果不为空则说明恰好有这么大的页可以用于分配。如果该链表为空，则会在  $b+1$  阶中寻找，如过  $b+1$  阶链表不为空，则将  $b+1$  中页块一分为二，一半用于分配，另一半加入  $b$  阶链表中。如果  $b+1$  阶链表也为空那么就继续向上寻找，如果都没找到空闲地址，就只能返回 NULL。以上的过程是分配空间的过程，在释放页的时候正好是分配的一个逆过程

当可执行文件正式执行的时候会发生缺页中断 page fault, `do_page_fault` 是缺页中断的核心函数，主要工作交给 `__do_page_fault` 处理，然后进行一些异常处理 `__do_kernel_fault` 和 `__do_user_fault`。 `do_page_fault` 调用 `__do_page_fault`，`__do_page_fault` 首先根据 `addr` 查找 VMA，然后交给 `handle_mm_fault` 进行处理。`handle_mm_fault` 调用 `__handle_mm_fault`，`__handle_mm_fault` 进行从

PGD-->PUD-->PMD-->PTE 的处理。对于二级映射来说，最主要的 PTE 的处理交给 `handle_pte_fault`。缺页中断处理会将页面加载到页框中。

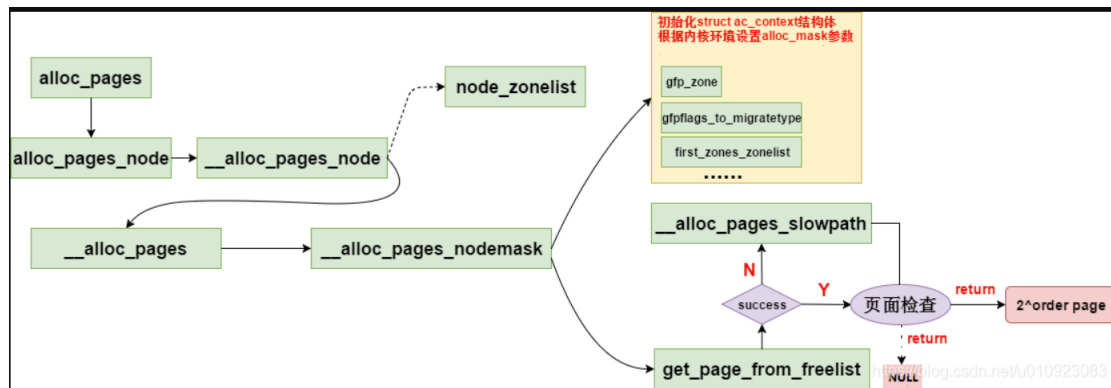


图 15: Linux 分配页面主要函数

如图 15 是 Linux 伙伴系统分配页面的时候用到的主要函数，我重新修改编译的内核是 Linux-6.3.8 版本，其中外层使用的函数是 `__get_free_pages`，`__get_free_pages()` 函数以 `gfp_mask` 分配方式分配 2 的 `order` 次方 ( $1 \leq order$ ) 个连续的物理页。该函数的实现主要是调用了 `alloc_pages()` 函数，它返回所分配的连续物理页中第一个页的逻辑地址。`__alloc_pages_nodemask` 是伙伴系统的核心，处理实质的内存分配工作。`__alloc_pages_nodemask()` 分配内存页面的关键函数是：`get_page_from_freelist()` 和 `__alloc_pages_slowpath()`，其中所谓的快速分配就是直接从现有的内存中去分配，如果不成功，再去尝试慢速分配。慢速分配会进行内存压缩，回收，然后再去尝试分配内存。

而释放页面的时候，Linux-6.3.8 使用的 `__free_pages` 函数，该函数有两个参数，从给定的页面 `page` 开始，释放的页面块个数为 2 的 `order` 次方 ( $1 \leq order$ ) 个

## 2.3 内核跟踪和调试程序设计

运行环境：Linux-6.3.8

实验思路：

1. 在可执行文件在建立映射之后从物理内存中加载页面的时候判断执行的文件是否为所监测的文件。

内核的可执行文件运行读取页面在内核中的/fs/exec.c 文件中，其中有一个函数 `static struct page *get_arg_page(struct linux_binprm *bprm, unsigned long pos,int write)`，该函数用于获取从可执行二进制文件中加载到内存中的一个页面。

```
if(strcmp(bprm->filename, ". /HustSSE_zz") == 0){
    printk("调用get_arg_page:获取页面，并映射到物理内存:%lu\n", (unsigned long) page_address(page));
    unsigned long pfn = page_to_pfn(page);
    printk(KERN_INFO "调用get_arg_page:pfn: %lu\n", pfn);
}
```

图 16: 打印物理页面

在该函数中添加如图所示的代码，判断当前的可执行文件的名称，确定后打印映射到物理内存中的页面地址。

2. 可执行文件运行之后，释放页框的时候调用的函数也在/fs/exec.c 文件中，其中有两个函数用于释放页框，`static void free_arg_page(struct linux_binprm *bprm, int i)`这个函数是释放单个页框，`static void free_arg_pages(struct linux_binprm *bprm)`这个函数是释放连续的内存，在这两个函数中可用于监测指定文件什么时候占用的页框和内存被释放。
3. 在执行可执行程序是会发生缺页中断，那么可以找到寻找空闲页框的函数中添加监测代码，linux-6.3.8 内核是使用的 `unsigned long __get_free_pages(gfp_t gfp_mask, unsigned int order)`函数，该函数上边已经介绍过了，其定义是在内核中的/mm/pages\_alloc.c 文件中。

```

unsigned long prn = page_to_prn(page);
if (strcmp(current->comm, "HustSSE_zz") == 0) {

    printk(KERN_INFO "HustSSE: Allocating page frame %lu\n", pfn);
    printk("page:");
    printk("%lu\n", (unsigned long) page_address(page));
}

```

图 17: 打印找到的空闲页框

如图 17 在 `__get_free_pages` 函数中添加上述代码, 就可以监测到特定文件运行时找到的空闲页框。

4. Linux 内核中也有相应的释放页框的函数 `void __free_pages(struct page *page, unsigned int order)`, 该函数释放指定的页框, 我们可以添加类似图 17 的代码到其函数体中, 这样可以监测到释放了哪些页框。

## 2.4 运行和测试过程

编写一个 C 语言程序, 通过 `gcc` 编译成 `elf` 类型的名为 `HustSSE_zz` 的可执行文件, 运行该可执行文件, 通过查看日志中的内容可以看到修改的内核中 `printk` 的信息

```

zhaozhan@zhaozhan-virtual-machine:~$ gcc HustSSE_zz.c -o HustSSE_zz
zhaozhan@zhaozhan-virtual-machine:~$ ./HustSSE_zz
hello HustSSE_zz
zhaozhan@zhaozhan-virtual-machine:~$

```

图 18: 可执行测试文件

如图 18 新建一个名为 `HustSSE_zz` 的可执行文件, 并进行运行。

```

[ 4990.948761] 调用get_arg_page:获取页面，并映射到物理内存:18446617665779986432
[ 4990.948766] 调用get_arg_page:pfn: 640047
[ 4990.948770] 调用get_arg_page:获取页面，并映射到物理内存:18446617665779986432
[ 4990.948771] 调用get_arg_page:pfn: 640047
[ 4990.948783] 调用get_arg_page:获取页面，并映射到物理内存:18446617665779986432
[ 4990.948784] 调用get_arg_page:pfn: 640047
[ 4990.948955] HustSSE: Allocating page frame 62361
[ 4990.948956] page:
[ 4990.948957] 18446617663413784576
[ 4990.948977] HustSSE: Allocating page frame 58404
[ 4990.948977] page:
[ 4990.948978] 18446617663397576704
[ 4990.950096] HustSSE: Allocating page frame 62357
[ 4990.950098] page:
[ 4990.950099] 18446617663413768192
[ 4990.950104] HustSSE: Freeing page frame 62357
[ 4990.950185] HustSSE: Allocating page frame 52084
[ 4990.950186] page:
[ 4990.950186] 18446617663371689984
[ 4990.950209] HustSSE: Freeing page frame 52084
zhaozhan@zhaozhan-virtual-machine:~/桌面$

```

图 19: 日志信息

如图 19 为 ./HustSSE\_zz 执行后 printk 打印出的信息，可以看出运行时调用了 get\_arg\_page 函数，打印出了物理内存的地址以及对应的页面号，获取空闲页框也是打印出了页框号 and 对应的地址，可以监测到在这个过程中释放了两个页框，页框号分别是：623567 和 52084。

## 2.5 实验心得和建议

编译内核的时候总是出现这样的错误：

make[1]: \*\*\* No rule to make target... 的错误

经查找资料发现输入命令：

scripts/config --disable SYSTEM\_TRUSTED\_KEYS

scripts/config --disable SYSTEM\_REVOCATION\_KEYS

就可以解决，经过几次测试发现必须是输入 sudo make -j8 命令并编译出错之后

之后在输入上述命令才能解决问题，提前输入命令并不能避免报错。

## 2.6 学习和编程实现参考网址

[\(166 条消息\) \[内核内存\] 伙伴系统 4---alloc\\_pages\(内存块分配\) 早起的虫儿有鹰吃的博客-CSDN 博客](#)

[\(165 条消息\) Linux 内存子系统——分配物理页面（alloc\\_pages） 绍兴小贵宁的博客-CSDN 博客](#)

[\(165 条消息\) 【Linux 1.0 内核源码剖析】执行程序——exec.c linux exec.c selfimpr1991 的博客-CSDN 博客](#)

[\(165 条消息\) Linux 内存管理一函数详解 memery 中的 cen\\_chenglian 999 的博客-CSDN 博客](#)

[\(165 条消息\) linux 内存管理-几个重要的数据结构和函数 pte t guoguangwu 的博客-CSDN 博客](#)

[linux 进程加载 - execve 系统调用 - 知乎 \(zhihu.com\)](#)

[深入理解 Linux Kernel 内核架构\(图文详解\) - 知乎 \(zhihu.com\)](#)

[\(166 条消息\) 内核函数 get\\_free\\_page 和 free\\_pages get\\_free\\_pages yldfree 的博客-CSDN 博客](#)

[\(166 条消息\) linux 可执行文件的加载和运行之一\(3\) get\\_arg\\_page yyt7529 的博客-CSDN 博客](#)

[深入浅出 Linux 内核内存管理基础 - 知乎 \(zhihu.com\)](#)

[Linux 内核 API free\\_pages|极客笔记 \(deepinout.com\)](#)

[\(166 条消息\) Linux 内存管理 \(10\)缺页中断处理 weixin\\_30362233 的博客-CSDN 博客](#)

[\(166 条消息\) Linux 内存管理 \(10\)缺页中断处理 weixin\\_30362233 的博客-CSDN 博客](#)

[\(166 条消息\) 编译内核报错 No rule to make target ‘debian/canonical-certs.pem’ 或 ‘canonical-revoked-certs.pem’ 的解决方法 linux 内核编译错误:make\[1\]: \\*\\*\\* 没有规则可制作目标‘debian/canonica\\_lylhw13 的博客-CSDN 博客](#)