



中山大學

SUN YAT-SEN UNIVERSITY

实 验 报 告

课程名称： 操作系统

姓 名： 方桂安

学 号： 20354027

专业班级： 2020 级智能科学与技术

任课教师： 吴贺俊

2022 年 12 月 2 日

## 实验报告成绩评定表

评定项目	内                容	满    分	评    分	总    分
实验态度	态度端正、遵守纪律、出勤情况	10		
实验过程	按要求完成算法设计、代码书写、注释清晰、运行结果正确	30		
实验记录	展示讲解清楚、任务解决良好、实验结果准确	20		
报告撰写	报告书写规范、内容条理清楚、表达准确规范、上交及时、无抄袭，抄袭记 0 分，提供报告供抄袭者扣分。	40		
<b>评语：</b>				
<div style="height: 180px;"></div>				
<b>指导老师签字：_____年   月   日</b>				

## 实验七 虚拟内存管理（内核源码）实验

### 一、 实验目的

1. 复习虚拟内存技术和缺页中断知识。
2. 统计一段时间内发生的缺页中断次数。
3. 掌握 Linux 内核函数的使用。

### 二、 实验内容

#### 1. 任务描述

##### 1) Linux 内核

Linux 是当今流行的操作系统之一。由于其源码的开放性，现代操作系统设计的思想和技术能够不断运用于它的新版本中。因此，读懂并修改 Linux 内核源代码无疑是学习操作系统设计技术的有效方法。

**Linux 内核：**内核指的是一个提供设备驱动、文件系统、进程管理、网络通信等功能的系统软件，内核并不是一套完整的操作系统，它只是操作系统的核心。

**Linux 发行版本：**一些组织或厂商将 Linux 内核与各种软件和文档包装起来，并提供系统安装界面和系统配置、设定与管理工具，就构成了 Linux 的发行版本。

## 2) 设计任务简介

编译 Linux 的内核，包括如下几个关键步骤：

### 1. 下载内核源代码

Linux 受 GNU 通用公共许可证（GPL）保护，内核源代码是完全开放的，可在 Linux 的官方网站下载。

### 2. 配置内核源代码

配置的作用是精确控制新内核的功能，即控制哪些功能需要编译到内核的二进制映像中。

### 3. 编译内核和模块

### 4. 安装和启动 Linux 内核

## 2. 实验说明

虚拟内存是操作系统内存管理的一种技术，它使得进程不必完全处于内存。

进程认为它拥有连续的可用的内存（一个连续完整的地址空间），而实际上，它通常是被分隔成多个物理内存碎片，还有部分暂时存储在外部磁盘存储器上，在需要进行数据交换。

这种方案的优点是程序可以大于物理内存。目前，大多数操作系统都使用了虚拟内存技术。

由于进程线性地址空间里的页面不必常驻内存，在执行一条指令时，如果要访问的页没有在内存中，那么停止该指令的执行，并产生一个页不存在的异常，对应的故障处理程序可通过从外存加载该页的方法来排除故障，即缺页中断处理。中断处理之后，原先引起的异常的指令就可以继续执行。

### 三、 实验记录

#### 1. 实施步骤

##### 1、查看内核版本

```
# uname -r
```

使用该命令查看当前内核的版本。

```
enderfga@Enderfga-PC:~$ uname -r
5.4.72-microsoft-standard-WSL2
enderfga@Enderfga-PC:~$
```

我使用的虚拟机是 ws12 的 5.4.72 版本。

由于使用的是 ws12，具体操作与 PPT 上有少许不同，步骤如下：



##### 2、下载所需内核版本

通过 Linux 官方网站下载内核，也可以到国内的某些网站进行下载。

这里我使用 wget 命令下载，使用的是 5.9.9 版本

```
enderfga@Enderfga-PC:~$ wget https://mirrors.edge.kernel.org/pub/linux/kernel/v5.x/linux-5.9.9.tar.gz
--2022-12-11 16:17:26-- https://mirrors.edge.kernel.org/pub/linux/kernel/v5.x/linux-5.9.9.tar.gz
Resolving mirrors.edge.kernel.org (mirrors.edge.kernel.org)... 147.75.48.161, 2604:1380:40f1:3f00::1
Connecting to mirrors.edge.kernel.org (mirrors.edge.kernel.org)[147.75.48.161]:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 181068934 (173M) [application/x-gzip]
Saving to: 'linux-5.9.9.tar.gz'

linux-5.9.9.tar.gz      17%[=====>
```

#### 3、解压

将压缩包解压到当前目录，然后切换到对应目录中

```
enderfga@Enderfga-PC:~$ ls
linux-5.9.9  linux-5.9.9.tar.gz
enderfga@Enderfga-PC:~$ cd linux-5.9.9/
enderfga@Enderfga-PC:~/linux-5.9.9$
```

## 4、配置内核

从微软开源的 github 仓库上获取到 config 文件的具体内容，位置在 WSL2-Linux-Kernel/Microsoft/config-wsl。

```
# CONFIG_X86_DEBUG_FPU is not set
# CONFIG_PUNIT_ATOM_DEBUG is not set
CONFIG_UNWINDER_ORC=y
# CONFIG_UNWINDER_FRAME_POINTER is not set
# CONFIG_UNWINDER_GUESS is not set
# end of x86 Debugging

#
# Kernel Testing and Coverage
#
# CONFIG_KUNIT is not set
# CONFIG_NOTIFIER_ERROR_INJECTION is not set
CONFIG_FUNCTION_ERROR_INJECTION=y
# CONFIG_FAULT_INJECTION is not set
CONFIG_ARCH_HAS_KCOV=y
CONFIG_CC_HAS_SANCOV_TRACE_PC=y
# CONFIG_KCOV is not set
# CONFIG_RUNTIME_TESTING_MENU is not set
CONFIG_ARCH_USE_MMEMTEST=y
# CONFIG_MMEMTEST is not set
# CONFIG_HIPRV_TESTING is not set
# end of Kernel Testing and Coverage
# end of Kernel hacking
".config" 4176L, 109543C                                     4176, 23      Bot
```

然后还需要安装构建依赖项：

```
enderfga@Enderfga-PC:~/linux-5.9.9$ sudo apt install build-essential flex bison dwarves libssl-dev libelf-dev
[sudo] password for enderfga:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  dpkg-dev fakeroot libalgorithms-diff-perl libalgorithms-diff-xs-perl libalgorithms-merge-perl libdpkg-perl libfakeroot libfile-fcntllock-perl libfl-dev
  libssl1.1 m4 make zlib1g zlib1g-dev
Suggested packages:
  bison-doc debian-keyring flex-doc bzip libssl-doc m4-doc make-doc
The following NEW packages will be installed:
  bison build-essential dpkg-dev dwarves fakeroot flex libalgorithms-diff-perl libalgorithms-diff-xs-perl libalgorithms-merge-perl libdpkg-perl libelf-dev
  libfakeroot libfile-fcntllock-perl libfl-dev libssl-dev m4 make zlib1g-dev
The following packages will be upgraded:
  libssl1.1 zlib1g
2 upgraded, 18 newly installed, 0 to remove and 166 not upgraded.
Need to get 5977 kB of archives.
After this operation, 21.3 MB of additional disk space will be used.
Do you want to continue? [Y/n]
```

## 5、编译内核，编译和安装模块，安装内核

```
# make -j8;make modules -j8;make modules_install -j8;make install -j8
```

利用 make 命令开始编译内核。使用 -j 选项来多线程处理，可以更有效的利用 CPU 资源。一台双核的机器上，可以用 make -j4，让 make 最多允许 4 个编译命令同时执行；四核的机器上，可以用 make -j8，让 make 最多允许 8 个编译命令同时执行。

```
Setup is 14108 bytes (padded to 14336 bytes).
System is 12542 kB
CRC 2d35b34f
Kernel: arch/x86/boot/bzImage is ready (#1)
DESCEND objtool
DESCEND bpf/resolve_btfids
CALL scripts/atomic/check-atomics.sh
CALL scripts/checksyscalls.sh
CHK include/generated/compile.h
INSTALL fs/nfs/flexfilelayout/nfs_layout_flexfiles.ko
DEPMOD 5.9.9-SYSU-Enderfga-20354027-WSL2
sh ./arch/x86/boot/install.sh 5.9.9-SYSU-Enderfga-20354027-WSL2 arch/x86/boot/bzImage \
System.map "/boot"
```

这里我所有命令一起执行了。

## 6、重新启动，检查新内核

编译完成。这里显示的 Kernel: arch/x86/boot/bzImage is ready (#1)

就是编译后生成的内核的位置

显然，和虚拟机不同，wsl2 并不是使用 grub 启动的，我们需要的是替换在 Windows 某目录下的内核文件：

```
C:\WINDOWS\System32\lxss\tools
```

```
# wsl --shutdown
```

重启，以开启新的内核。

注意：可能出现短暂死机情况。

```
# uname -r
```

再次查看内核版本，检查是否成功。



The screenshot shows a terminal window titled "enderfga@Enderfga-PC: .". The user enters the command `uname -r`, and the output is `5.9.9-SYSU-Enderfga-20354027-WSL2`. The terminal also displays Chinese text annotations: "这里我所有命令一起执行了。" (I executed all my commands together here.), "6、重新启动，检查新内核" (6. Restart and check the new kernel), "编译完成。这里显示的 Kernel: arch/x86/boot/bzImage is ready (#1)" (Compilation complete. The kernel displayed here is: arch/x86/boot/bzImage is ready (#1)), and "就是编译后生成的内核的位置" (This is the location of the kernel generated after compilation).

## 2. 实验结果

本实验采用修改内核源代码的方法来统计系统缺页次数，因此，涉及到相关内核源代码的修改、内核的重新编译、统计缺页次数的输出等内容。

具体步骤如下：

1. 在内核源码中找到 `include/linux/mm.h` 文件，声明变量 `pfcount`，用于统计缺页次数。

```

80     extern void * high_memory;
81     extern int page_cluster;
82
83     // 声明变量
84     extern unsigned long volatile pfcoun;

```

2. 在 /arch/x86/mm/fault.c 文件中定义变量 pfcoun，并在 do\_page\_fault() 函数中找到 good\_area，让变量 pfcoun 递增 1，实现了缺页次数的统计。

```

37     /*
38      * Returns 0 if mmiotrace is disabled, or if the fault is not
39      * handled by mmiotrace:
40      */
41     unsigned long volatile pfcoun;
42
43
44     static nokprobe_inline int

```

```

1356     good_area:
1357         if (unlikely(access_error(hw_error_code, vma))) {
1358             bad_area_access_error(regs, hw_error_code, address, vma);
1359             return;
1360         }
1361         pfcoun++;

```

4. 修改 kernel/kallsyms.c 文件，在文件最后插入

```
EXPORT_SYMBOL(pfcoun);
```

该步骤的作用是，EXPORT\_SYMBOL 标签内定义的函数或者符号对全部内核代码公开。

- ✓ 可以用文本编辑器编辑直接修改文件
- ✓ 也可使用命令

echo 'EXPORT\_SYMBOL (pfcoun);' >>kernel/kallsyms.c 完成修改。

```

enderfga@Enderfga-PC:~/linux-5.9.9$ cat kernel/kallsyms.c | grep pfcoun
EXPORT_SYMBOL(pfcoun);
enderfga@Enderfga-PC:~/linux-5.9.9$

```



4. 重新编译内核，前面已经提到了。
5. 编写测试程序和 Makefile，请参见提供的示例代码。

```
# cd ~ && mkdir source && cd source
```

完成 readpfcount.c 和 Makefile 的编写。

测试代码的功能是以内核模块的形式读取 pfcount 的值，并输出。

```
static int my_proc_show(struct seq_file* m, void* v){

    seq_printf(m, "The pfcount is %ld and jiffies is %ld!\n",

pfcount, jiffies);

    return 0;

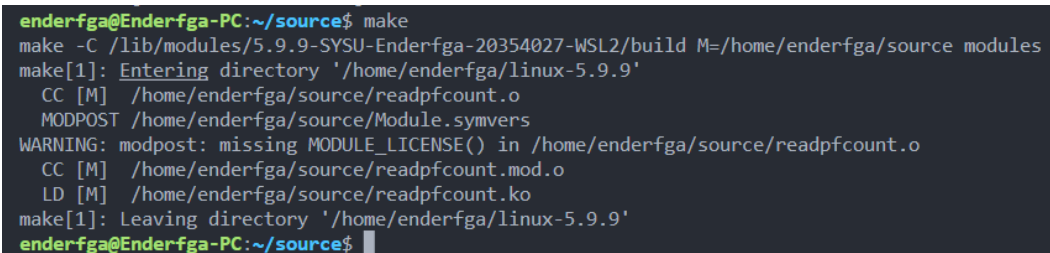
}

struct proc_dir_entry* file = proc_create("readpfcount", 0x0644,

NULL, &my_fops);
```

6. 编译测试程序，并加载内核

```
# make
```



```
enderfga@Enderfga-PC:~/source$ make
make -C /lib/modules/5.9.9-SYSU-Enderfga-20354027-WSL2/build M=/home/enderfga/source modules
make[1]: Entering directory '/home/enderfga/linux-5.9.9'
  CC [M]  /home/enderfga/source/readpfcount.o
  MODPOST /home/enderfga/source/Module.symvers
WARNING: modpost: missing MODULE_LICENSE() in /home/enderfga/source/readpfcount.o
  CC [M]  /home/enderfga/source/readpfcount.mod.o
  LD [M]  /home/enderfga/source/readpfcount.ko
make[1]: Leaving directory '/home/enderfga/linux-5.9.9'
enderfga@Enderfga-PC:~/source$
```

```
# insmod readpfcount.ko
```

7. 加载内核成功后，输入如下命令，测试实验结果。

```
cat /proc/readpfcount
```

```
enderfga@Enderfga-PC:/proc$ ls
1 16 295 77 buddyinfo cpuinfo execdomains ioports kmsg mdstat net self thread-self vmstat
117 196 36 78 bus crypto fb irq kpagecgroup meminfo pagetypeinfo softirqs timer_list zoneinfo
13 21 4518 79 cgroups devices filesystems kallsyms kpagecount misc partitions readpfcount stat tty
14 2239 68 86 cmdline diskstats fs kcore kpageflags modules readpfcount swaps uptime
15 25 69 99 config.gz dma interrupts key-users loadavg mounts sched_debug sys version
156 294 70 acpi consoles driver iomem locks mtrr mtrr schedstat sysvipc vmallocinfo

enderfga@Enderfga-PC:/proc$ cat /proc/readpfcount
The pfcunt is 2722927 and jiffies is 4295043754!
enderfga@Enderfga-PC:/proc$
```

至此，实验顺利完成。

## 四、 总结与讨论

这个实验采用修改内核源代码的方法来统计系统缺页次数，这是一种比较常见的方法。修改内核源代码可以让我们有效地控制和跟踪系统的缺页行为，从而对系统进行优化。

在实现这个实验时，需要注意几个问题：

1. 在修改内核源代码时，要仔细阅读代码，了解代码的含义，以免破坏原有的功能。
2. 在编译内核时，要确保编译环境的正确性，并且注意使用相应的编译选项，以确保编译出的内核能够正常工作。
3. 在统计缺页次数时，要确保统计的方法准确无误，以免得出错误的结论。

## 五、附：程序模块的源代码

readpfcount.c:

```
/*
*****
使用 seq_file 接口实现可读写 proc 文件
参考:
https://elixir.bootlin.com/linux/v5.0/source/include/linux/proc\_fs.h
*****
*/

#include <linux/module.h>
#include <linux/sched.h>
#include <linux/uaccess.h>
#include <linux/proc_fs.h>
#include <linux/fs.h>
#include <linux/mm.h>
#include <linux/seq_file.h>
```

```

#include <linux/slab.h>

extern unsigned long volatile pfcount;

/*5,实现 show 函数
   作用是将内核数据输出到用户空间
   将在 proc file 输出时被调用
   */
static int my_proc_show(struct seq_file *m, void *v)
{
    /*这里不能使用printfk之类的函数
      要使用seq_file输出的一组特殊函数
      */
    seq_printf(m, "The pfcount is %ld and jiffies is %ld!\n",
pfcount,jiffies);
    return 0;
}

static int my_proc_open(struct inode *inode, struct file *file)
{
    /*4,在 open 函数中调用 single_open 绑定 seq_show 函数指针*/
    return single_open(file, my_proc_show, NULL);
}

/*2,填充proc_create函数中调用的flie_operations结构体
   其中my开头的函数为自己实现的函数,
   seq和single开头为内核实现好的函数,直接填充上就行
   open为必须填充函数
   */
static struct proc_ops my_fops={
    .proc_open = my_proc_open,
    .proc_release = single_release,
    .proc_read = seq_read,
    .proc_lseek = seq_lseek
};

static int __init my_init(void)
{
    struct proc_dir_entry *file;
    //创建父级目录,第二个参数NULL表示在/proc下
    struct proc_dir_entry *parent = proc_mkdir("20354027",NULL);

    /*1,
      首先要调用创建proc文件的函数,需要绑定flie_operations

```

```

    参数 1: 要创建的文件
    参数 2: 权限设置
    参数 3: 父级目录, 如果传 NULL, 则在 /proc 下
    参数 4: 绑定 flie_operations
    */
    file = proc_create("readpfcount", 0644, parent, &my_fops);
    if(!file)
        return -ENOMEM;
    return 0;
}

/*6,删除proc 文件*/
static void __exit my_exit(void)
{
    //移除目录及文件
    remove_proc_entry("20354027", NULL);
}

module_init(my_init);
module_exit(my_exit);

```

## Makefile:

```

ifneq ($(KERNELRELEASE),)
    obj-m:=readpfcount.o
else
    KDIR:= /lib/modules/$(shell uname -r)/build
    PWD:= $(shell pwd)

default:
    $(MAKE) -C $(KDIR) M=$(PWD) modules
clean:
    $(MAKE) -C $(KDIR) M=$(PWD) clean
endif

```