# 广州大学学生实验报告

**开课学院及实验室：**计算机科学与教育软件学院计算机软件实验室 2020**年**4**月**16**日**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **学院** | 计算机科学与教育软件学院 | **年级/专业/班** | 软件171 | **姓名** | 谢金宏 | **学号** | 1706300001 |
| **实验课**  **程名称** | Linux系统实验 | | | | | **成绩** |  |
| **实验项**  **目名称** | 文件操作算法 | | | | | **指导**  **老师** | 张艳玲 |

## 实验目的

1. 理解Linux中虚拟文件系统的内容；
2. 学习编写内核模块的方法；
3. 在虚拟文件系统/proc中实现文件操作算法。

## 实验内容

编写一个内核模块，在/proc文件系统中增加一个目录hello，并在这个目录中增加一个文件world，文件的内容为hello world。

## 实验涉及的系统调用以及内核函数

1. **在proc中创建目录函数proc\_mkdir**

该函数原型为：

*struct proc\_dir\_entry \*proc\_mkdir(const char \*name, struct proc\_dir\_entry \*parent)；*

name：要创建的目录的名称

parent：指向该目录的父目录的指针

返回指向当前目录的指针结构体。

1. **创建version文件的函数为proc\_create，该函数原型为：**

*static inline struct proc\_dir\_entry \*proc\_create(const char \*name, umode\_t mode, struct proc\_dir\_entry \*parent, const struct file\_operations \*proc\_fops)；*

第一个参数name为文件名，

第二个参数mode为文件的读写权限，

第三个参数parent为其父目录的结构体指针，第四个参数proc\_fops为文件的读写操作结构体。

1. **实现内核空间与用户空间的数据传递函数**

copy\_to\_user()和copy\_from\_user()这两个函数。(旧内核是raw\_copy\_to\_user() 和 raw\_copy\_from\_user()这两个函数。)

copy\_to\_user()函数的完整形态为：

*unsigned long copy\_to\_user(void \*to, const void \*from, unsigned long count);*

函数的作用是将内核空间的数据复制到用户空间，其中：

to：目标地址（用户空间）

from：源地址（内核空间）

count：将要拷贝数据的字节数

返回：成功返回0，失败返回没有拷贝成功的数据字节数

copy\_from\_user()函数的完整形态为

*unsigned long copy\_from\_user(void \*to, const void \*from, unsigned long count);*

函数的作用是将用户空间的数据复制到内核空间，其中：

to：目标地址（内核空间）

from：源地址（用户空间）

count：将要拷贝数据的字节数

返回：成功返回0，失败返回没有拷贝成功的数据字节数

1. **从 /proc 中删除一个文件函数 remove\_proc\_entry()，函数的原型是：**

*remove\_proc\_entry(const char \*name, struct proc\_dir\_entry \*parent)*

name：要删除的文件名

parent ：文件在 /proc 文件系统中的位置

## 实验原理与重点

/proc文件系统是Linux上的一种虚拟文件系统，存储的是当前内核运行状态的一系列特殊文件，用户可以通过这些文件查看有关系统硬件及当前正在运行进程的信息。

最初开发 /proc 文件系统是为了提供有关系统中进程的信息。但是由于这个文件系统非常有用，因此内核中的很多元素也开始使用它来报告信息或启用动态运行时配置。

用户一般不可以随便在/proc 文件系统中添加信息。

1. **/proc 目录下文件详解**
   1. 内存

/proc/buddyinfo 伙伴系统的信息

/proc/pagetypeinfo 伙伴系统进一步细分信息

/proc/zoneinfo 内存区域使用情况

/proc/slabinfo

/proc/meminfo 当前内存信息

/proc/vmstat 虚拟内存统计信息

/proc/vmallocinfo 虚拟内存分配信息

/proc/swaps swap分区使用情况

/proc/mtd 内存设备分区表信息

/proc/dma DMA（直接内存访问）通道的列表

/proc/mtrr 系统使用的Memory Type Range Registers (MTRRs)

/proc/kpagecount

/proc/kpageflags

* 1. I/O

/proc/filesystems 目前系统支持的文件系统

/proc/diskstats 磁盘设备的统计信息

/proc/ioports 当前系统硬件设备使用的IO端口列表

/proc/iomem I/O 内存映射

/proc/locks 当前被内核锁定的文件

/proc/mounts 当前挂载信息

* 1. CPU

/proc/cpuinfo cpu相关信息

/proc/loadavg 当前系统负载

/proc/softirqs 系统软中断信息

/proc/schedstat 调度器信息

/proc/sched\_debug 调度器debug信息

* 1. kernel

/proc/cmdline 在引导启动时传递给Linux内核的参数

/proc/crypto 内核支持的加密方式

/proc/modules 当前系统已经加载的模块（lsmod）

/proc/version 内核版本信息

/proc/stat 系统和内核的统计信息

/proc/fb 内核编译期间帧缓冲信息

/proc/kmsg 内核日志信息

/proc/kcore 表示系统物理内存，可以用gdb检查内核数据结构的

/proc/kallsyms 内核符号信息，主要用于调试

/proc/timer\_list 内核各种计时器信息

/proc/timer\_stats

/proc/sysrq-trigger 内核触发器（危险！！！）

/proc/execdomains Linux内核当前支持的execution domains

* 1. other

/proc/interrupts 中断表

/proc/uptime 系统运行时间

/proc/devices 设备信息（主设备号等）

/proc/mdstat 虚拟设备信息（软raid等）

/proc/misc 其他的主要设备(设备号为10)上注册的驱动

/proc/cgroup cgroup相关信息

/proc/consoles

/proc/keys 证书相关

/proc/key-users

1. **/proc目录下的目录**

acpi

bus

driver

fs

ipmi

irq

scsi

sys

sysvipc

tty

1. **（重点）本实验是编写一个内核模块，在/proc文件系统中增加一个目录hello，并在这个目录中增加一个文件world，文件的内容为hello world。**

实验步骤如下：

* 1. 利用proc\_mkdir（）函数创建目录：

*proc\_parent = proc\_mkdir("hello", NULL);*

* 1. 在/proc目录下有version和softirqs等文件，本实验利用proc\_create函数创建一个version文件world：

*proc\_create("world", 0, proc\_parent, &proc\_fops);*

其中，world文件的父目录指针即为先前创建hello目录所返回的指针，proc\_fops定义了该文件所能执行的操作，由于实验只要求读取文件内容，可只实现read属性：

*static const struct file\_operations proc\_fops = {*

*.read = read\_proc*

*};*

其中，read\_proc为实现读取文件内容的函数指针，其实现如下：

*static ssize\_t read\_proc(struct file \*filp, char \_\_user \*buf, size\_t count, loff\_t \*offp)*

*{*

*if (count > temp)*

*count = temp;*

*temp = temp - count;*

*raw\_copy\_to\_user(buf, msg, count);*

*if (count == 0)*

*temp = len;*

*return count;*

*}*

该函数将文件内容通过msg复制给buf，以此实现文件内容的读取。

* 1. 定义模块的初始化和清理函数

其实现如下：

*int proc\_init(void)*

*{*

*create\_new\_proc\_entry();*

*return 0;*

*}*

*void proc\_cleanup(void)*

*{*

*printk(KERN\_INFO "Inside cleanup\_module\n");*

*remove\_proc\_entry("hello", proc\_parent);*

*remove\_proc\_entry("world", NULL);*

*}*

*MODULE\_LICENSE("GPL");*

*module\_init(proc\_init);*

*module\_exit(proc\_cleanup);*

* 1. 编写Makefile文件，编译内核文件，并插入内核，观察结果。

## 实验原始数据、结果与分析

内核模块的代码如下：

*// proc\_hello\_world.c*

#include <linux/module.h>

#include <linux/kernel.h>

#include <linux/sched.h>

#include <linux/uaccess.h>

#include <linux/proc\_fs.h>

#define PRINTK\_TAG KERN\_INFO "proc\_hello\_world: "

static struct proc\_dir\_entry \*proc\_parent;

static const char \*message = "Hello, world!\n";

static ssize\_t proc\_read(struct file \*filp, char \_\_user \*buf, size\_t count, loff\_t \*offp)

{

    ssize\_t message\_len = strlen(message);

    if (\*offp >= message\_len) {

        \*offp = message\_len;

        return 0;

    }

    if (\*offp + count > message\_len) {

        count = message\_len - \*offp;

    }

    copy\_to\_user(buf, message + \*offp, count);

    \*offp += count;

    return count;

}

static const struct file\_operations proc\_fops = {

    .read = proc\_read

};

int proc\_init(void)

{

*// Create a directory named "hello".*

    proc\_parent = proc\_mkdir("hello", NULL);

    if (proc\_parent == NULL)

    {

        printk(PRINTK\_TAG "[%s] Error creating proc entry!\n", \_\_func\_\_);

        return -ENOMEM;

    }

*// Create a file named "world" and add read attribute to it.*

*// With file mode 444, everyone can read this file.*

    if (proc\_create("world", 0444, proc\_parent, &proc\_fops) == NULL) {

        printk(PRINTK\_TAG "[%s] Error creating proc file!\n", \_\_func\_\_);

        remove\_proc\_entry("hello", NULL);

        return -ENOMEM;

    }

    printk(PRINTK\_TAG "[%s] Successfully initialized.\n", \_\_func\_\_);

    return 0;

}

void proc\_cleanup(void)

{

*// Remove file "world" and its parent directory "world".*

    remove\_proc\_entry("world", proc\_parent);

    remove\_proc\_entry("hello", NULL);

    printk(PRINTK\_TAG "[%s] Cleanup.\n", \_\_func\_\_);

}

MODULE\_LICENSE("GPL");

module\_init(proc\_init);

module\_exit(proc\_cleanup);

在内核模块的入口函数中，先使用proc\_mkdir在系统的proc目录下创建hello目录，然后使用proc\_create在hello目录下创建world文件，将该文件设为可读（文件模式0444），并将文件的read操作与proc\_read函数相关联。

在proc\_read函数中，需要注意读文件操作的实现，该函数的返回值是成功读取的字节数。读文件操作的偏移量参数\*offp不能超过预设消息“Hello, world!\n”的长度。若该偏移量已经超过或等于预设消息长度，则将偏移量修改为消息长度，并返回0，表示没有后续数据可读取。偏移量与欲读取的字节数总长不能超过消息长度，否则欲读取的字节数修改为最多能读取的字节数。最后，使用copy\_to\_user完成到用户空间数据的传递，将\*offp相应地向前移动，并返回成功读取的字节数。

编写Makefile，内容如下：

obj-m := proc\_hello\_world.o

KERNELDIR := /lib/modules/$(shell uname -r)/build

PWD := $(shell pwd)

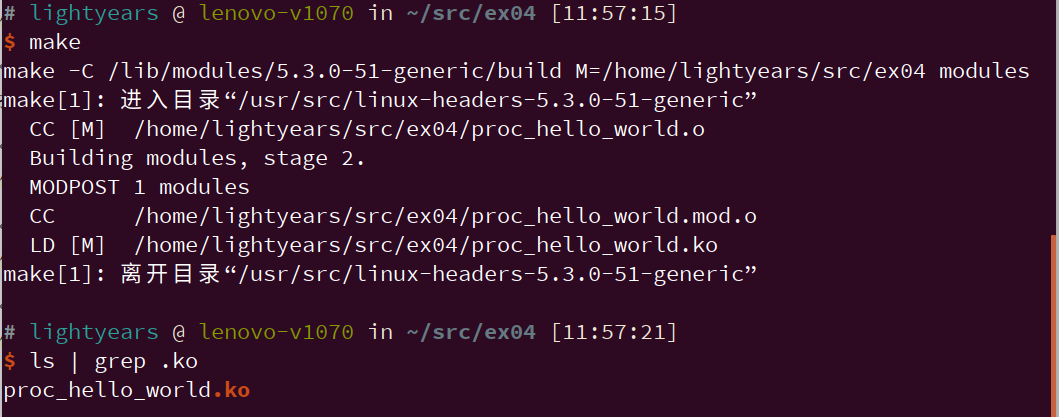
default:

    make -C $(KERNELDIR) M=$(PWD) modules

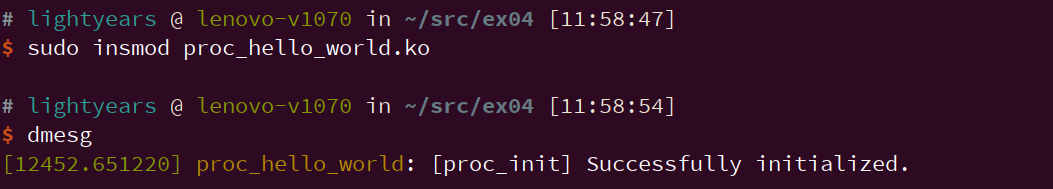
clean:

    make -C $(KERNELDIR) M=$(PWD) clean

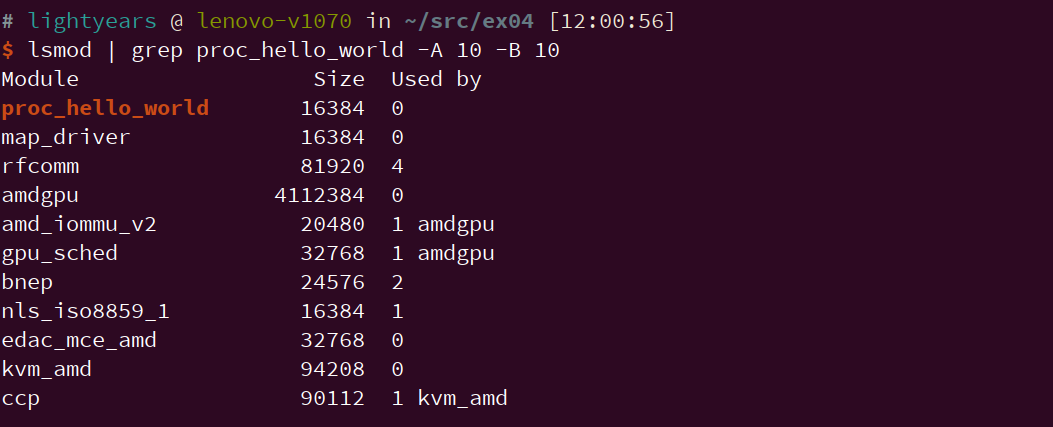
使用make编译内核模块。

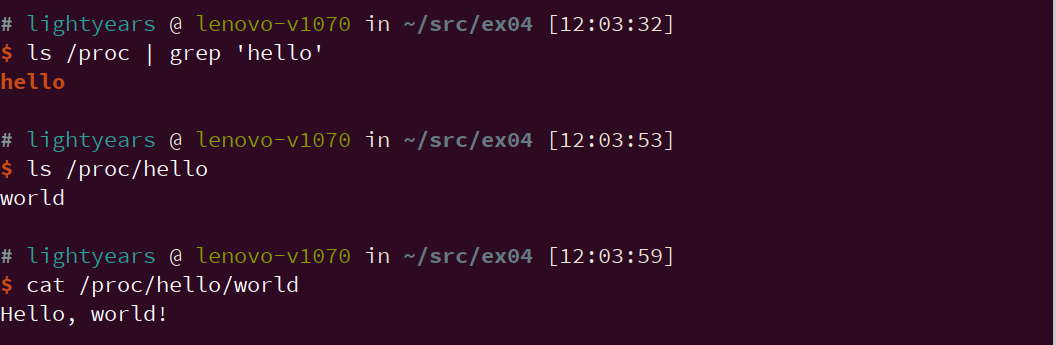


插入内核模块，并使用dmesg查看内核日志。

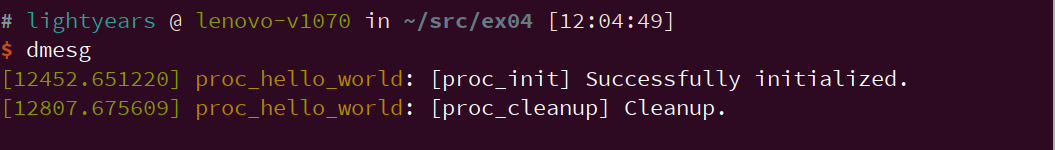


使用ls列出/proc/hello目录的内容，使用cat读取/proc/hello/world文件。

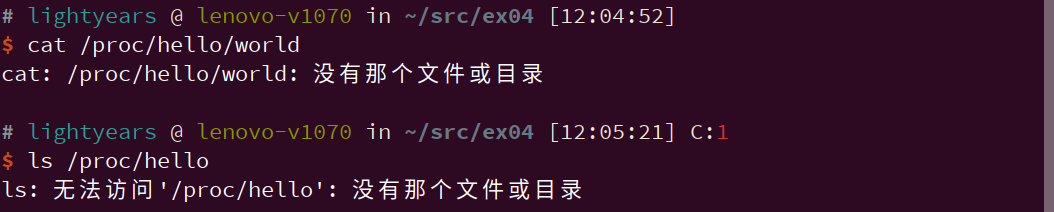




使用rmmod移除内核模块，并使用dmesg查看内核日志。



使用ls和cat验证内核模块在/proc目录下创建的目录和文件均已移除。



## 实验思考题

**编写一个有两个进程并发运行的程序，并在结尾设定一个死循环，编译完成后让其在后台运行，然后在前台用cat命令查看/proc目录下的文件/proc/meminfo（当前内存信息）、/proc/vmstat（虚拟内存统计信息）、/proc/vmallocinfo（虚拟内存分配息）、/proc/filesystems（目前系统支持的文件系统）、/proc/locks（当前被内核锁定的文件）、/proc/cpuinfo（cpu相关信息）和/proc/modules（当前系统已经加载的模块（lsmod））中信息，分析其含义。**

不难写出思考题所要求的程序，使两个进程并发空转。在运行程序的指令后使用“&”符号，可使得程序在后台运行。

*// demo1.c*

#include <stdbool.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

int main(void)

{

    pid\_t pid = fork();

    while (true) {

        continue;

    }

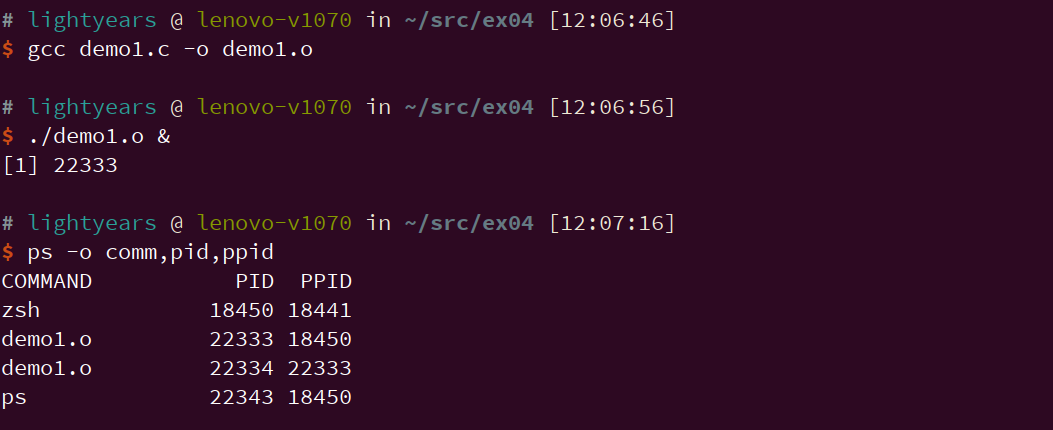
    if (pid > 0) {

        wait(NULL);

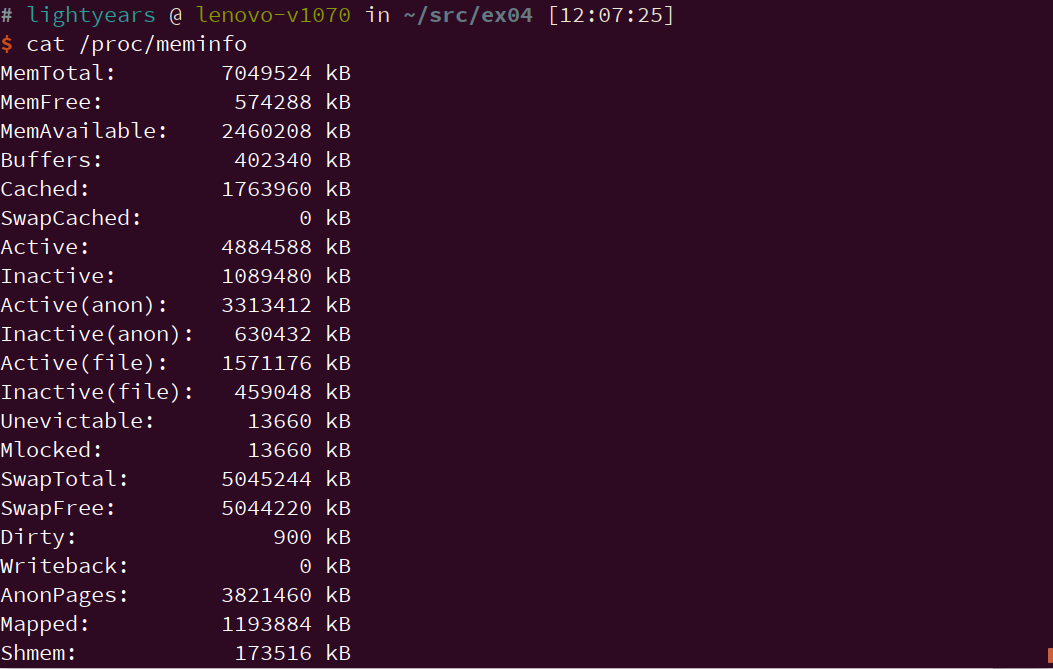
    }

    return 0;

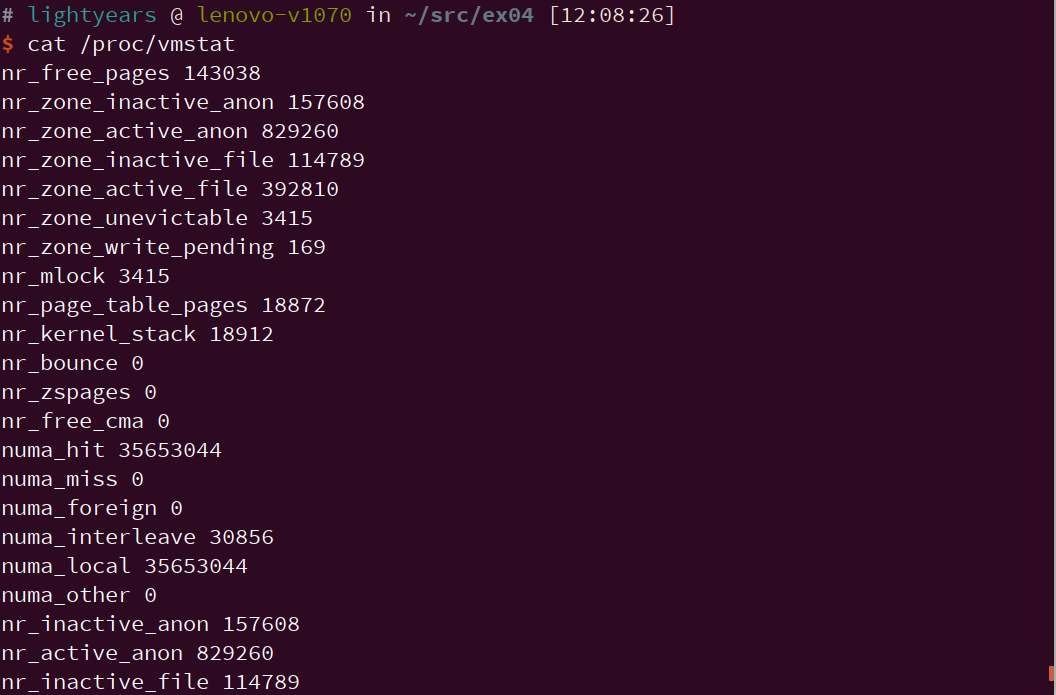
}



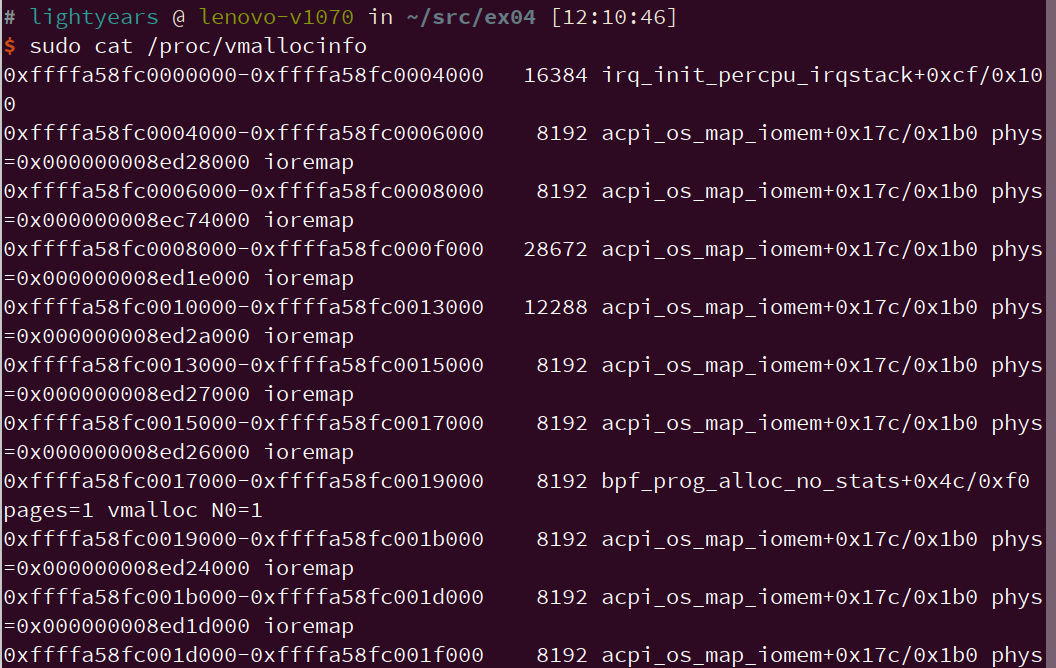
在前台通过cat读取/proc目录下的信息：



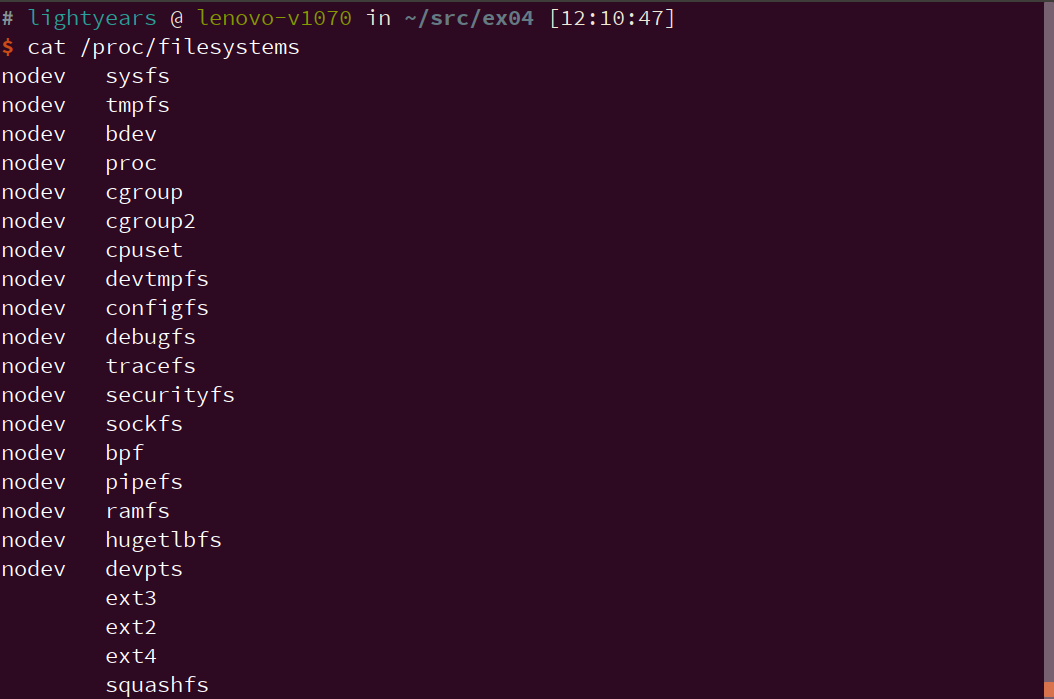
Meminfo汇报系统中关于内存的相关信息。Meminfo显示的信息单位是KB(=1000B)，但实际上是KiB（=1024B），只是处于兼容原因没有将单位更正过来。汇报的信息包括内存总量、可用内存总量、缓存的内存总量、用于页面交换的内存总量等等。



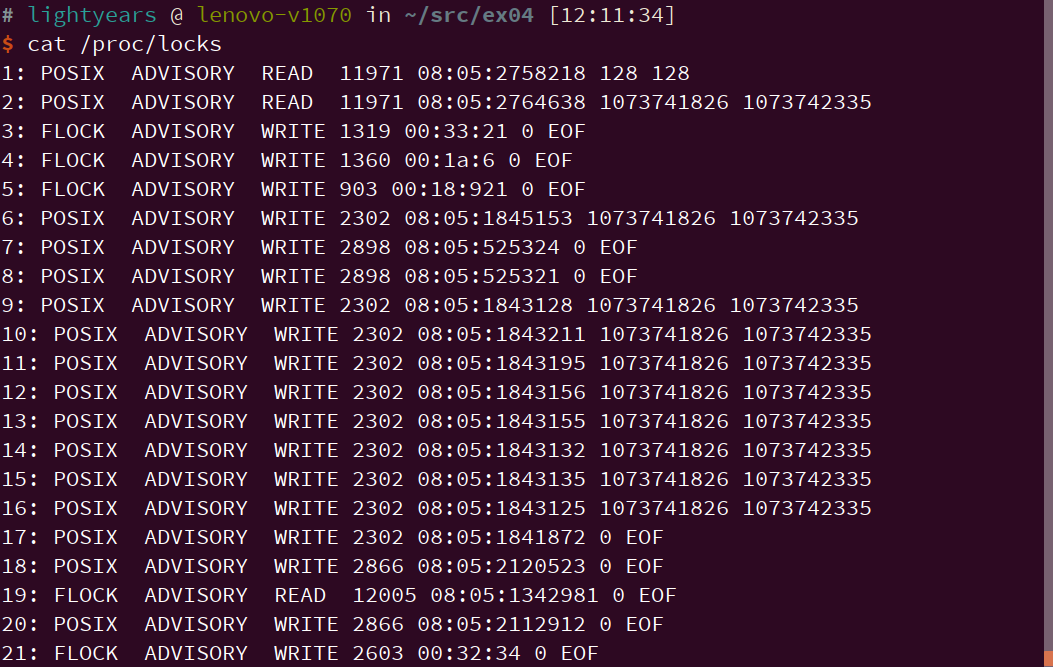
Vmstat汇报的是虚拟内存的统计信息，汇报的信息包括脏页面数量、回写页数、不稳定页数等等。



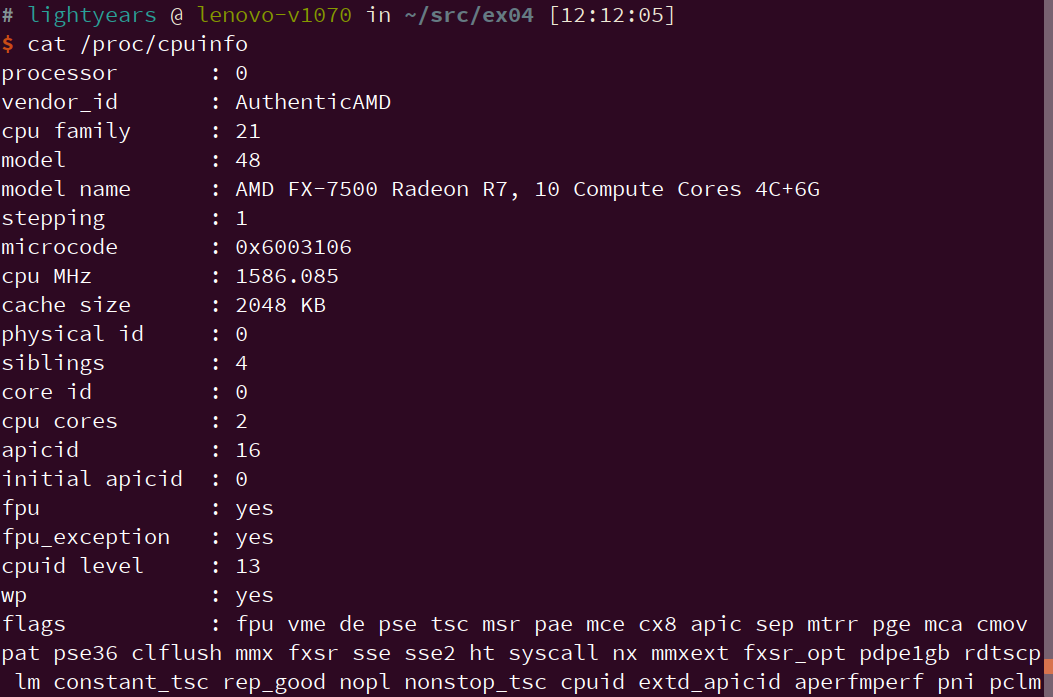
Vmallocinfo汇报内核中的虚存分配信息，在调试内核（模块）时很有用。第一列表示虚存的起止地址，第二列表示虚存的大小，第三列表示相应虚存是在哪一个函数中被分配的，第四列表示虚存对应的物理内存，第五列表示分配类型。



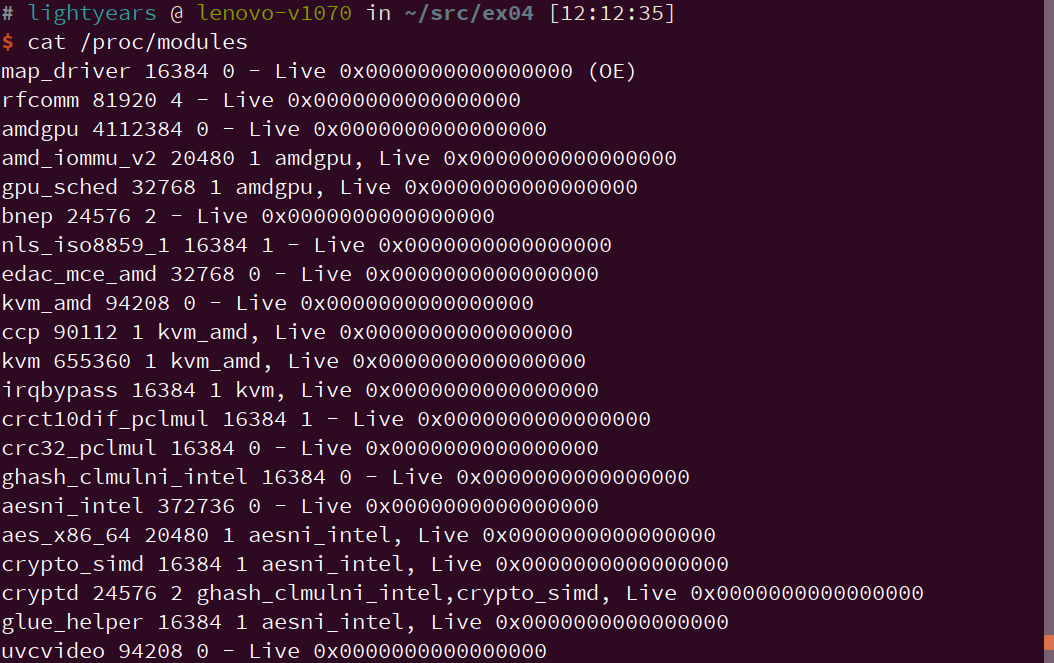
Filesystems汇报系统中已经安装的文件系统。Nodev表示其后的文件系统并不需要与特定的物理设备关联。



Locks汇报当前由内核锁定的文件的相关信息，包含内核内部的调试数据；每个锁定占据一行，且具有一个惟一的编号；输出信息中每行的第二列表示当前锁定使用的锁定类别，POSIX表示目前较新类型的文件锁，由lockf系统调用产生，FLOCK是传统的UNIX文件锁，由flock系统调用产生。



Cpuinfo显示与当前系统安装的CPU相关的信息。汇报的信息包括处理器的序号、CPU的型号、CPU核心数量等等。



Modules汇报系统中已经安装的内核模块。第一列是模块名称，第二列是模块占用的内存大小（以字节为单位），第三列是模块被加载的次数，第四列显示模块依赖，第五列是当前模块在内核中的偏移位置。