openEuler内核编程技术

实训指导书

第四章 第2讲

《proc文件系统》

软件所制

**术语表**

|  |  |  |  |
| --- | --- | --- | --- |
| 英文 | 中文 | 含义 | 别名 |
|  |  |  |  |
|  |  |  |  |

第四章 第2讲 proc文件系统

# 任务1：编写一个proc\_parser.c文件，以内核模块方式加载，实现读取系统CPU负载和内存占用信息、获取当前进程的ID及其他信息的功能，形成持续的打印输出。（60min）

## 相关知识

1. 介绍

前面我们学习了proc文件系统的基本概念。/proc文件系统是一个接口，用户与内核交互的接口，用户从/proc文件系统中读取很多内核释放出来的信息（包括内核各个管理模块的动态信息，CPU信息，硬件驱动释放出来的信息等等）；同时内核也可以在必要的时候从用户得到输入，进而改变内核的变量，或者运行状态。

/proc文件系统中主要包含两方面的文件（或者说主要有两个大的用途）：一是只读文件，用于读取系统信息，或者内核配置信息；二是可写文件，用于向内核传递参数。

/proc文件系统是一个虚拟文件系统，它只存在内存当中，而不占用外存空间。它以文件系统的方式为访问系统内核数据的操作提供接口，通过文件系统接口实现，用于输出系统运行状态。它以文件的形式，为操作系统本身和应用程序直接的通信提供了一个界面，用户和应用程序可以安全、方便地通过/proc得到系统的信息，并可以改变内核的某些参数。

由于系统的信息，如进程，是动态改变的，所以用户或应用程序读取/proc文件时，/proc文件系统是动态从系统内核读出所需信息并提交的。

1. 相关接口函数

内核编程、调试中，经常需要内核释放出一些状态信息给应用层，或者用于调试，或者只是让用户了解内核或驱动的工作状态，这时候我们就需要在/proc文件系统下面添加文件。

那么，怎么样使用proc文件系统创建我们自己的文件和目录呢？首先，包含这个头文件：

#include <linux/proc\_fs.h>

1. 创建普通proc文件

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

参数说明：

name: 要创建的文件名

mode: 要创建的文件的属性

parent: 这个文件的父目录

proc\_fops: 此文件的一系列操作函数，如读写操作

使用该函数创建一个普通文件，文件名是name，文件属性mode，所在的父目录是parent。如果要在/proc文件系统的根目录下创建这个文件，那么parent为NULL。如果创建成功，该函数返回一个指向新建的proc\_dir\_entry结构的指针，否则返回NULL。

name参数可以包含多级目录，比如create\_proc\_entry）"foo/bar/test"），调用这个函数会自动创建文件之前的目录（如果需要），这些目录的属性是默认属性0755。

2. 创建目录

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

参数说明：

name: 要创建的目录名

parent: 这个目录的父目录

该函数创建在父目录parent下创建一个目录name。

通常情况下假如父目录为/proc我们可以将此参数用NULL来代替，当然，在选择父目录的时候可以借助一些快捷变量。

|  |  |
| --- | --- |
| proc\_dir\_entry | 在文件系统中的位置 |
| proc\_root\_fs | /proc |
| proc\_net | /proc/net |
| proc\_bus | /proc/bus |
| proc\_root\_driver | /proc/driver |

3. 删除文件或目录

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

参数说明：

name: 要删除的文件或目录名

parent: 所在的父目录

这个函数从proc文件系统中删除一个文件或目录。可能跟大多数人编程的习惯不一样，proc文件系统中文件的删除是通过参数name，而不是通过创建那个文件时所返回的指向proc\_dir\_entry的指针。另外一点需要注意的是，该函数不会递归删除目录下的文件。如果在proc\_dir\_entry中的data变量保存了分配的内存，也请先释放对应的内存，然后再删除该文件。

1. 文件操作的系列回调函数

static const struct file\_operations proc\_fops = {

.open = proc\_open, //打开文件时的回调函数

.read = seq\_read,//读文件时调用的回调函数

.write = seq\_write,//写文件时的回调函数

.llseek = seq\_lseek,

.release = single\_release,

};

对于打开一个文件一般用，proc\_open(struct inode \*inode, struct file \*file)，然后在其函数的实现中再用 single\_open(file, callback, NULL);见下面例子。

回调函数就是一个通过函数指针调用的函数。如果你把函数的指针（地址）作为参数传递给另一个函数，当这个指针被用来调用其所指向的函数时，我们就说这是回调函数。回调函数不是由该函数的实现方直接调用，而是在特定的事件或条件发生时由另外的一方调用的，用于对该事件或条件进行响应。

1. proc\_dir\_entry结构体

在创建proc目录和文件时都会返回一个表示proc文件结构的结构体proc\_dir\_entry.

struct proc\_dir\_entry {

unsigned int low\_ino;

umode\_t mode; // 文件访问权限模式

nlink\_t nlink;

kuid\_t uid; // 文件的用户ID

kgid\_t gid; // 文件的组ID

loff\_t size;

const struct inode\_operations \*proc\_iops; // 文件Inode操作函数

const struct file\_operations \*proc\_fops; // 文件操作函数

struct proc\_dir\_entry \*next, \*parent, \*subdir;

void \*data;

atomic\_t count; /\* use count \*/

atomic\_t in\_use; /\* number of callers into module in progress; \*/

/\* negative -> it's going away RSN \*/

struct completion \*pde\_unload\_completion;

struct list\_head pde\_openers; /\* who did ->open, but not ->release \*/

spinlock\_t pde\_unload\_lock; /\* proc\_fops checks and pde\_users bumps \*/

u8 namelen;

char name[];

};

1. 将内容写入seq\_file文件中

int seq\_printf(struct seq\_file \*, const char \*, ...)\_\_attribute\_\_ ((format(printf,2,3)));

函数seq\_printf是最常用的输出函数，它用于把给定参数按照给定的格式输出到seq\_file文件。

**Seq\_file的作用:**

内核通过在procfs文件系统下建立文件来向用户空间提供输出信息，用户空间可以通过任何文本阅读应用查看该文件信息，但是procfs 有一个缺陷，如果输出内容大于1个内存页，需要多次读，因此处理起来很难，另外，如果输出太大，速度比较慢，有时会出现一些意想不到的情况。因此就出现seq\_file功能，seq\_file可以完成原有procfs能完成的所有功能，并且可以轻松的处理大的文件。

## 任务描述

1、以内核方式加载一个hello module程序

2、以内核模式加载创建一个文件，路径为/home/kernel\_file，写入“你好”，读出并打印。

3、读取cpu的负载；

4、读取内存的占用信息；

5、获取当前进程的PID和名字；

6、用内核线程将获取的信息持续打印；

## 审核要求

1. 正确编写满足功能的源文件，正确编译。
2. 正常加载、卸载内核模块；且内核模块功能满足任务所述。
3. 提交相关源码与运行截图。

## 参考答案

1、环境准备

（1）内核源码准备

查看系统下是否存在当前内核版本的源码：

|  |
| --- |
| # ls |

如果不存在，则下载对应版本的内核源码：

此处示例的内核版本对应的源码是：https://gitee.com/openeuler/raspberrypi-kernel/repository/archive/openEuler-20.03-LTS-raspi.zip

下载完解压之后，拷贝到上述系统目录下（或者其他指定目录下，目录需与Makefile中所述一致）。

|  |
| --- |
| # wget https://gitee.com/openeuler/raspberrypi-kernel/repository/archive/openEuler-20.03-LTS-raspi.zip # 下载  # unzip openeuler-raspberrypi-kernel-openEuler-20.03-LTS-raspi.zip # 解压 |

（2）内核源码编译

|  |
| --- |
| # cd raspberrypi-kernel  # make openeuler-raspi\_defconfig # 加载内核配置  # make prepare && make scripts # 内核预编译（耗时短） |

|  |
| --- |
| [root@openEuler build]#  [root@openEuler build]# make openeuler-raspi\_defconfig  HOSTCC scripts/basic/fixdep  HOSTCC scripts/kconfig/conf.o  YACC scripts/kconfig/zconf.tab.c  LEX scripts/kconfig/zconf.lex.c  HOSTCC scripts/kconfig/zconf.tab.o  HOSTLD scripts/kconfig/conf  #  # configuration written to .config  #  [root@openEuler build]#  [root@openEuler build]# make prepare  UPD include/config/kernel.release  WRAP arch/arm64/include/generated/uapi/asm/errno.h  WRAP arch/arm64/include/generated/uapi/asm/ioctl.h  WRAP arch/arm64/include/generated/uapi/asm/ioctls.h  WRAP arch/arm64/include/generated/uapi/asm/ipcbuf.h  WRAP arch/arm64/include/generated/uapi/asm/kvm\_para.h  WRAP arch/arm64/include/generated/uapi/asm/mman.h  WRAP arch/arm64/include/generated/uapi/asm/msgbuf.h  WRAP arch/arm64/include/generated/uapi/asm/poll.h  WRAP arch/arm64/include/generated/uapi/asm/resource.h  WRAP arch/arm64/include/generated/uapi/asm/sembuf.h  WRAP arch/arm64/include/generated/uapi/asm/shmbuf.h  WRAP arch/arm64/include/generated/uapi/asm/socket.h  WRAP arch/arm64/include/generated/uapi/asm/sockios.h  WRAP arch/arm64/include/generated/uapi/asm/swab.h  WRAP arch/arm64/include/generated/uapi/asm/termbits.h  WRAP arch/arm64/include/generated/uapi/asm/termios.h  WRAP arch/arm64/include/generated/uapi/asm/types.h  WRAP arch/arm64/include/generated/asm/bugs.h  WRAP arch/arm64/include/generated/asm/delay.h  WRAP arch/arm64/include/generated/asm/div64.h  WRAP arch/arm64/include/generated/asm/dma.h  WRAP arch/arm64/include/generated/asm/dma-contiguous.h  WRAP arch/arm64/include/generated/asm/early\_ioremap.h  WRAP arch/arm64/include/generated/asm/emergency-restart.h  WRAP arch/arm64/include/generated/asm/hw\_irq.h  WRAP arch/arm64/include/generated/asm/irq\_regs.h  WRAP arch/arm64/include/generated/asm/kdebug.h  WRAP arch/arm64/include/generated/asm/kmap\_types.h  WRAP arch/arm64/include/generated/asm/local.h  WRAP arch/arm64/include/generated/asm/local64.h  WRAP arch/arm64/include/generated/asm/mcs\_spinlock.h  WRAP arch/arm64/include/generated/asm/mm-arch-hooks.h  WRAP arch/arm64/include/generated/asm/msi.h  WRAP arch/arm64/include/generated/asm/preempt.h  WRAP arch/arm64/include/generated/asm/qrwlock.h  WRAP arch/arm64/include/generated/asm/rwsem.h  WRAP arch/arm64/include/generated/asm/segment.h  WRAP arch/arm64/include/generated/asm/serial.h  WRAP arch/arm64/include/generated/asm/set\_memory.h  WRAP arch/arm64/include/generated/asm/sizes.h  WRAP arch/arm64/include/generated/asm/switch\_to.h  WRAP arch/arm64/include/generated/asm/trace\_clock.h  WRAP arch/arm64/include/generated/asm/unaligned.h  WRAP arch/arm64/include/generated/asm/user.h  WRAP arch/arm64/include/generated/asm/vga.h  UPD include/generated/uapi/linux/version.h  UPD include/generated/utsrelease.h  CC kernel/bounds.s  UPD include/generated/bounds.h  UPD include/generated/timeconst.h  CC arch/arm64/kernel/asm-offsets.s  UPD include/generated/asm-offsets.h  CALL scripts/checksyscalls.sh  LDS arch/arm64/kernel/vdso/vdso.lds  CC arch/arm64/kernel/vdso/gettimeofday.o  AS arch/arm64/kernel/vdso/note.o  AS arch/arm64/kernel/vdso/sigreturn.o  VDSOL arch/arm64/kernel/vdso/vdso.so.dbg  VDSOSYM include/generated/vdso-offsets.h  [root@openEuler build]# |

2、编写任务所述内核模块

**一、参考答案源码**

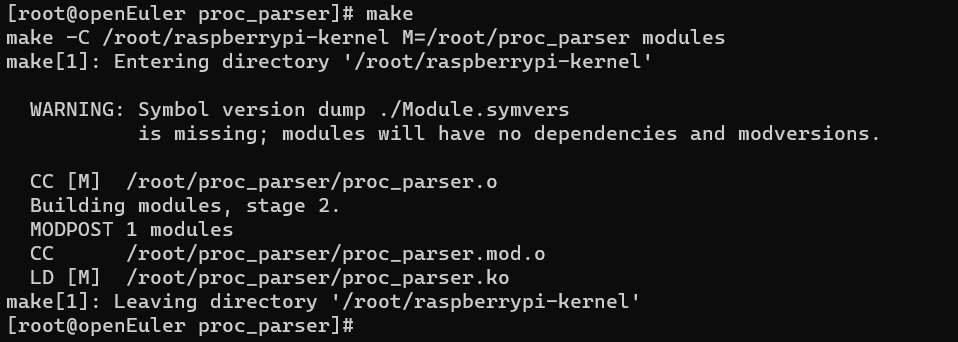
（1）proc\_parser.c

|  |
| --- |
| #include <linux/module.h>  #include <linux/init.h>  #include <linux/kernel.h>  #include <linux/fs.h>  #include <linux/sched/signal.h>  #include <linux/string.h>  #include <linux/sched.h>  #include <linux/kthread.h>  static struct task\_struct \*cycleprint\_kthread = NULL;  struct task\_struct \*tmp\_p;  char tmp\_cpu\_load[5] = "";  char tmp\_mem\_total[20] = "";  char tmp\_mem\_free[20] = "";  static char\* info = "你好\n";  static int write\_file(void)  {  char buf[20] = "";  struct file\* fp;  loff\_t pos;  fp = filp\_open("/home/kernel\_file", O\_RDWR | O\_CREAT, 0644);  if (IS\_ERR(fp))  {  printk("Create file error\n");  return -1;  }  pos = 0;  kernel\_write(fp, info, strlen(info), &pos);  pos = 0;  kernel\_read(fp, buf, strlen(info), &pos);  printk("%s", buf);  filp\_close(fp, NULL);  return 0;  }  static void sleep\_milli\_sec(int n\_milli\_sec) {  long timeout = (n\_milli\_sec)\*HZ/1000;  do {  while(timeout > 0) {  timeout = schedule\_timeout(timeout);  }  } while(0);  }  static int atoi(char a[], int n)  {  int ans = a[0] - '0';  int i = 1;  for(;i < n; i++)  ans = ans \* 10 + (a[i] - '0');  return ans;  }  static int load\_kenerl\_info(void)  {    struct file \*fp\_cpu, \*fp\_mem;  loff\_t pos;  char buf\_cpu[10];  char buf\_mem[100];  int i = 0;  int j = 0;  fp\_cpu = filp\_open("/proc/loadavg", O\_RDONLY, 0);  if (IS\_ERR(fp\_cpu))  {  printk("open load file error\n");  return -1;  }  fp\_mem = filp\_open("/proc/meminfo", O\_RDONLY, 0);  if (IS\_ERR(fp\_mem))  {  printk("open mem file error\n");  return -1;  }  pos = 0;  kernel\_read(fp\_cpu, buf\_cpu, sizeof(buf\_cpu), &pos);  pos = 0;  kernel\_read(fp\_mem, buf\_mem, sizeof(buf\_mem), &pos);  strncpy(tmp\_cpu\_load, buf\_cpu, 4);  for(; i <= 99; i++)  {  if (buf\_mem[i] >= '0' && buf\_mem[i] <= '9')  {  tmp\_mem\_total[j] = buf\_mem[i];  j++;  }  if (buf\_mem[i] == '\n')  {  i++;  break;  }  }  for(j = 0; i <= 99; i++)  {  if (buf\_mem[i] >= '0' && buf\_mem[i] <= '9')  {  tmp\_mem\_free[j] = buf\_mem[i];  j++;  }  if (buf\_mem[i] == '\n')  break;  }  filp\_close(fp\_cpu, NULL);  filp\_close(fp\_mem, NULL);  return 0;  }  static int print\_fun(void \*data)  {  write\_file();  while(!kthread\_should\_stop())  {  if(load\_kenerl\_info() != 0)  {  printk("read meminfo or loadarvg file error!\n");  return -1;  }  for\_each\_process(tmp\_p)  {  if(tmp\_p->state == 0)  printk("p name:%s, pid is:%d\n", tmp\_p->comm, tmp\_p->pid);  }  printk("the cpu load is: %s\n", tmp\_cpu\_load);  printk("total memory is: %s KB\n", tmp\_mem\_total);  printk("free memory is: %s KB\n", tmp\_mem\_free);  printk("occupy memory is: %d KB\n", atoi(tmp\_mem\_total, strlen(tmp\_mem\_total)) - atoi(tmp\_mem\_free, strlen(tmp\_mem\_free)));  sleep\_milli\_sec(3000);  }  return 0;  }  static int \_\_init proc\_parser\_init(void)  {  cycleprint\_kthread = kthread\_run(print\_fun, NULL, "cycle\_print\_kthread");  return 0;  }  static void \_\_exit proc\_parser\_exit(void)  {  if(cycleprint\_kthread)  {  printk("stop thread: cycle\_print\_kthread\n");  kthread\_stop(cycleprint\_kthread);  }  printk("proc\_parser exit\n");  }    module\_init(proc\_parser\_init);  module\_exit(proc\_parser\_exit);  MODULE\_LICENSE("GPL"); |

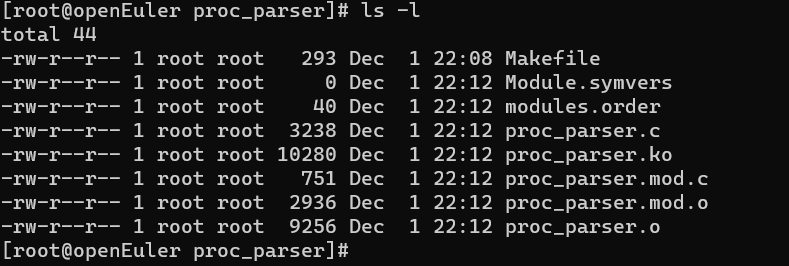
（2）Makefile

|  |
| --- |
| ifneq ($(KERNELRELEASE),)  obj-m := proc\_parser.o  else  KERNELDIR ?=/root/raspberrypi-kernel  PWD := $(shell pwd)  default:  $(MAKE) -C $(KERNELDIR) M=$(PWD) modules  endif  install:  insmod proc\_parser.ko  uninstall:  rmmod proc\_parser  .PHONY:clean  clean:  -rm \*.mod.c \*.o \*.order \*.symvers \*.ko |

执行make编译源码：

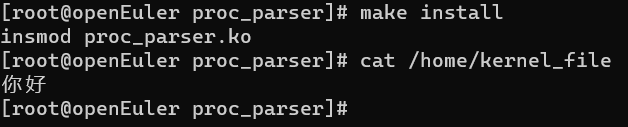


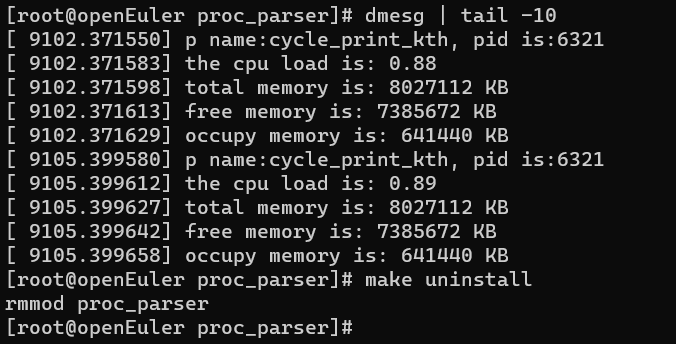
编译完成后的文件列表：



进行模块加载、查看、卸载：

执行插入模块make install后，程序将在/home目录生成文件kernel\_file，打开将显示你好，同时查看打印信息dmesg | tail -10可以得到读取cpu的负载、读取内存的占用信息、获取当前进程的PID和名字





# 任务2：编写一个proc\_time.c程序，以内核模块加载，在/proc目录下生成一个目录，在此目录下生成一个文件，读取时可以显示系统的当前时间（60min）

## 相关知识

1. 内核时钟介绍

Linux系统时钟以读取的硬件时钟为起始点，根据系统启动后的滴答数来计算时间，系统内的所有计时均基于它。系统用一个全局变量jiffies表示，该变量每个时钟周期更新一次，即表示系统自启动以来的时钟滴答数目。在这里我们需要借助一些内核函数来完成系统内核时间获取。

1. 在内核中获取系统时间接口

获取时间

void do\_gettimeofday (struct timeval \*tv)

此函数获取从1970-1-1 0:0:0到现在的时间值，存在timeval的结构体里边。

变量 tv 以秒和微妙表示当前系统时间，tv 指定获取当前系统时间的结构体变量地址。

struct timeval 结构体：

struct timeval {

time\_t tv\_sec;

suseconds\_t tv\_usec;

};

struct timeval 为设定时间或获取时间时使用的结构体，tv\_sec 变量把当前时间换算为秒，tv\_usec 值指定或获取 tv\_usec 无法表示的 us 单位经过的时间。

头文件：#include <linux/timer.h>

时间格式转换

void rtc\_time\_to\_tm(unsigned long time, struct rtc\_time \*tm)

此内核函数将系统实时时钟时间转换为格林尼治标准时间（GMT）。如果要得到北京时间需。需要将此时间处理（年份加上1900，月份加上1，小时加上8）。具体参看内核中rtc.h中rtc\_time\_to\_tm代码实现。

头文件：#include <linux/rtc.h>

## 任务描述

1、创建proc文件hello，编写文件对应的回调函数。在用户空间打开时显示"hello proc！"

2、在/proc目录下生成一个目录show\_time，在此目录下生成一个文件，在内核中调用时钟接口，获取内核时间，转化为GMT时间后，再转为北京时间并写入这个文件。

## 审核要求

1. 正确编写满足功能的源文件，正确编译。
2. 正常加载、卸载内核模块；且内核模块功能满足任务所述。
3. 提交相关源码与运行截图。

## 参考答案

一、参考答案源码

1、proc\_time.c

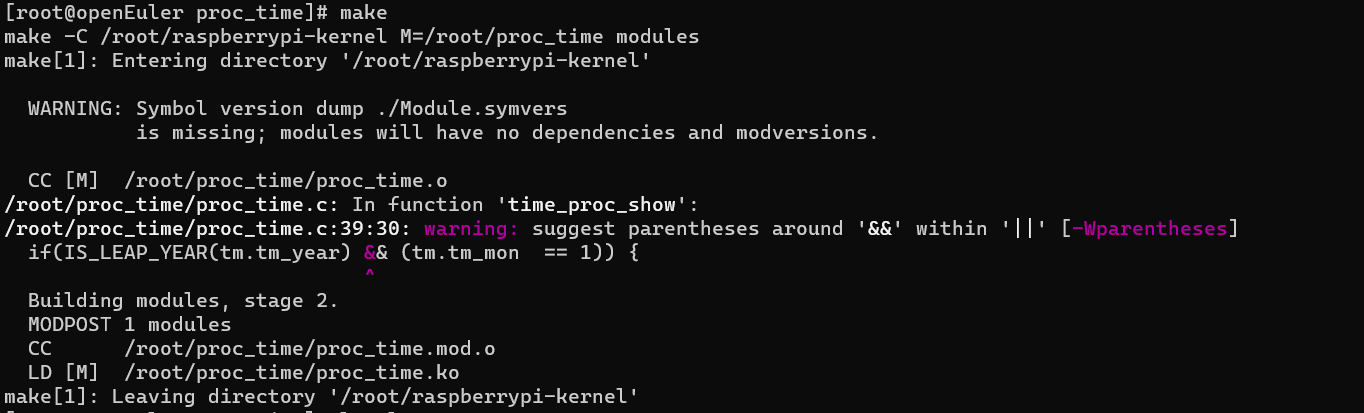
|  |
| --- |
| #include <linux/module.h>  #include <linux/proc\_fs.h>  #include <linux/seq\_file.h>  #include <linux/timer.h>  #include <linux/rtc.h>  #define IS\_LEAP\_YEAR(year) !(year%400)||(year%100&&!(year%4))  struct proc\_dir\_entry \*time\_dir;  static int mon[12] = {31,28,31,30,31,30,31,31,30,31,30,31};  static int hello\_proc\_show(struct seq\_file \*m, void \*v) {  seq\_printf(m, "Hello proc\n");  return 0;  }  static int hello\_proc\_open(struct inode \*inode, struct file \*file) {  return single\_open(file, hello\_proc\_show, NULL);  }  static const struct file\_operations hello\_proc\_fops = {  .owner = THIS\_MODULE,  .open = hello\_proc\_open,  .read = seq\_read,  .llseek = seq\_lseek,  .release = single\_release,  };  static int time\_proc\_show(struct seq\_file \*m, void \*v) {  struct timeval tv;  struct rtc\_time tm;  int year,month,day,hour;  int carry = 0;  do\_gettimeofday(&tv);  rtc\_time\_to\_tm(tv.tv\_sec,&tm);    hour = (tm.tm\_hour + 8)%24;  carry = (tm.tm\_hour + 8)/24;    if(IS\_LEAP\_YEAR(tm.tm\_year) && (tm.tm\_mon == 1)) {  day = (tm.tm\_mday + carry)%(mon[tm.tm\_mon] + 1);  carry = (tm.tm\_mday + carry)/(mon[tm.tm\_mon] + 2);  }  else {  day = (tm.tm\_mday + carry)%mon[tm.tm\_mon];  carry = (tm.tm\_mday + carry)/(mon[tm.tm\_mon] + 1);  }    month = (tm.tm\_mon + carry)%12 + 1;  carry = (tm.tm\_mon + carry)/12;    year = tm.tm\_year + 1900 + carry;  seq\_printf(m, "UTC time :%d-%d-%d %d:%d:%d \n",year, month, day, hour, tm.tm\_min,tm.tm\_sec);  return 0;  }  static int time\_proc\_open(struct inode \*inode, struct file \*file) {  return single\_open(file, time\_proc\_show, NULL);  }  static const struct file\_operations time\_proc\_fops = {  .owner = THIS\_MODULE,  .open = time\_proc\_open,  .read = seq\_read,  .llseek = seq\_lseek,  .release = single\_release,  };  static int \_\_init time\_proc\_init(void) {  proc\_create("hello", 0, NULL, &hello\_proc\_fops);  time\_dir = proc\_mkdir("show\_time" ,NULL);  proc\_create("proc\_time", 0, time\_dir, &time\_proc\_fops);  return 0;  }  static void \_\_exit time\_proc\_exit(void) {  remove\_proc\_entry("hello", NULL);  remove\_proc\_entry("proc\_time", time\_dir);  remove\_proc\_entry("show\_time", NULL);  }  MODULE\_LICENSE("GPL");  module\_init(time\_proc\_init);  module\_exit(time\_proc\_exit); |

2、Makefile

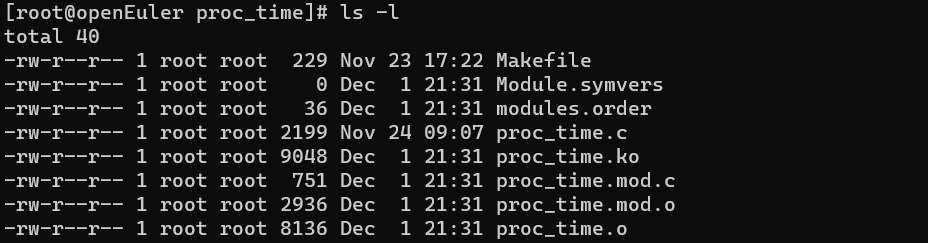
|  |
| --- |
| ifneq ($(KERNELRELEASE),)  obj-m := proc\_time.o  else  KERNELDIR ?=/root/raspberrypi-kernel  PWD := $(shell pwd)  default:  $(MAKE) -C $(KERNELDIR) M=$(PWD) modules  endif  install:  insmod proc\_time.ko  uninstall:  rmmod proc\_time  .PHONY:clean  clean:  -rm \*.mod.c \*.o \*.order \*.symvers \*.ko |

二、运行结果

执行make编译源码：



编译完成后的文件列表：



进行模块加载、查看、卸载：

执行插入模块make install后，程序将在/proc目录生成文件hello，打开hello将打印Hello proc，在/proc生成目录show\_time，打开文件夹show\_time，有一个proc\_time文件，打开该文件将显示当前的北京时间。切换回模块所在文件夹执行make uninstall执行对模块的卸载

