# 实验六(Guide)

**任务一：编写USB设备驱动程序**



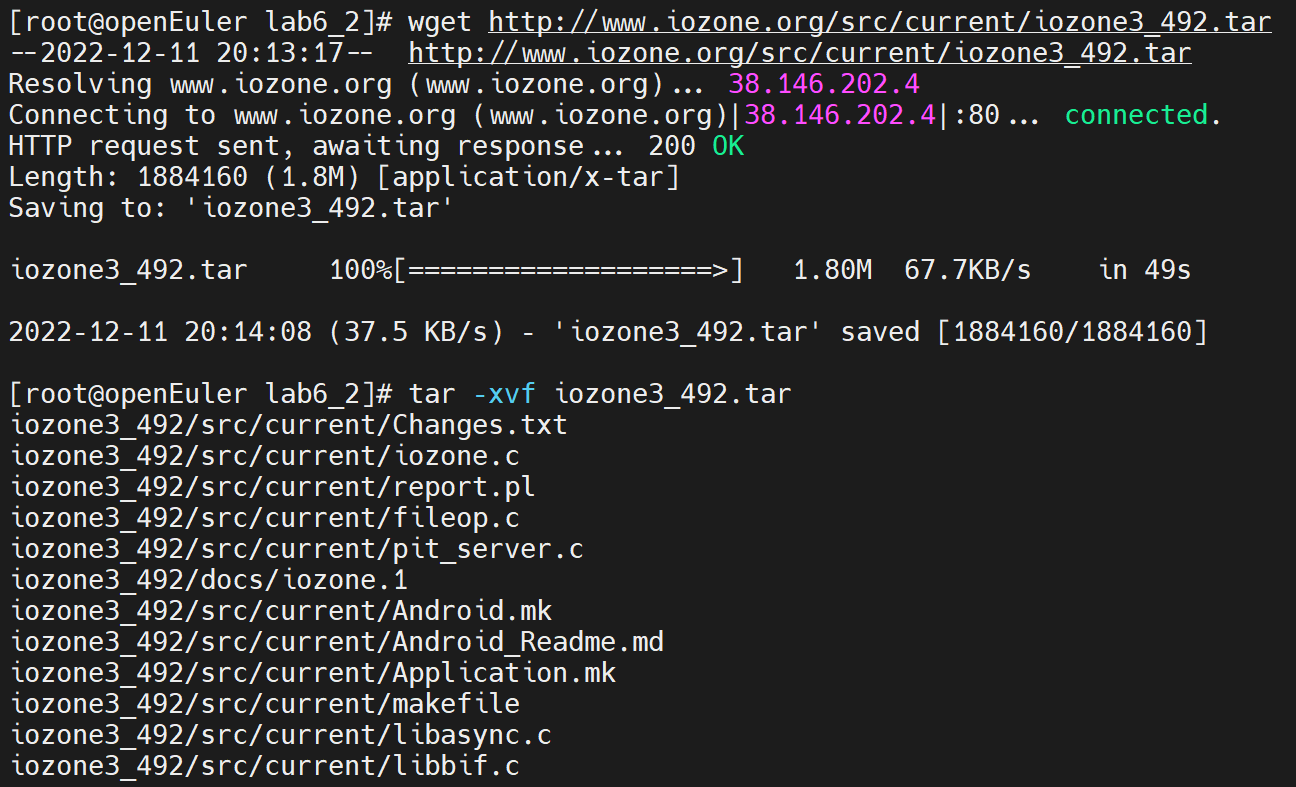
无法完成

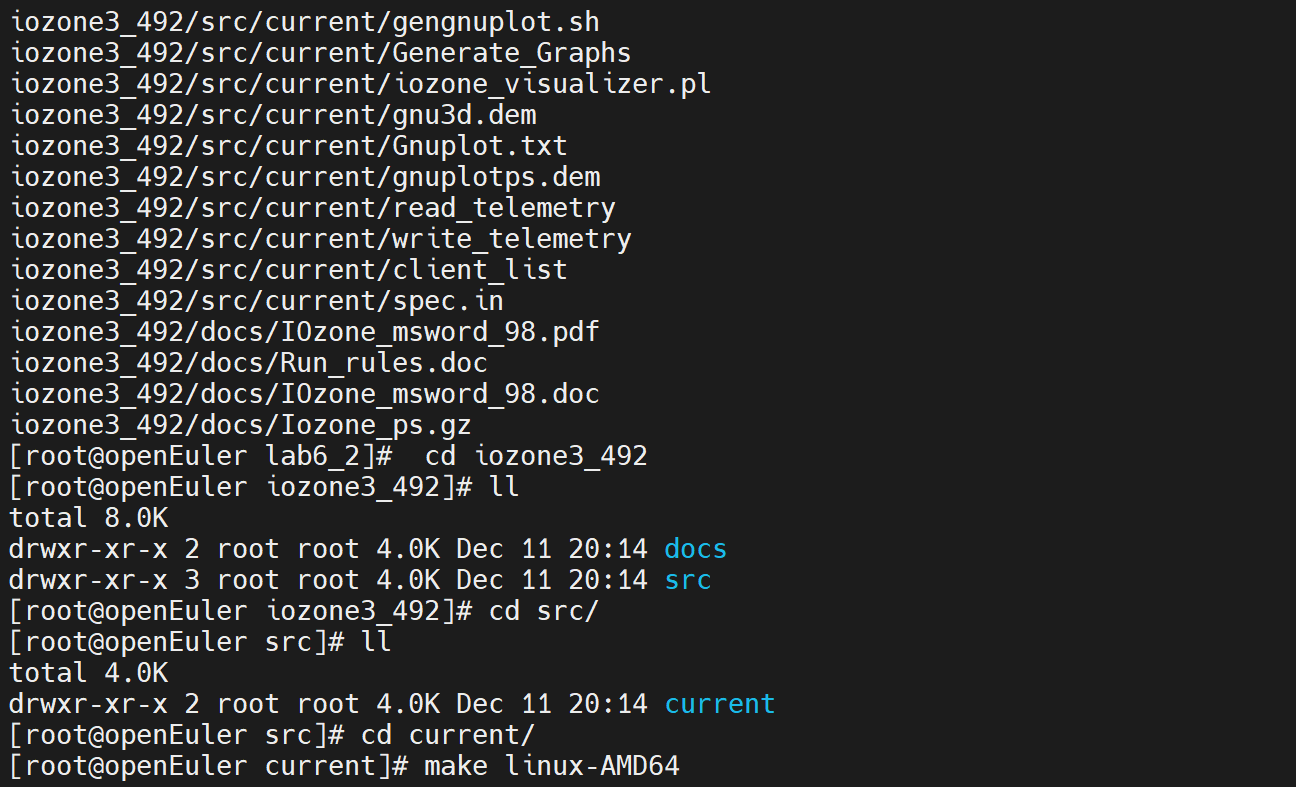


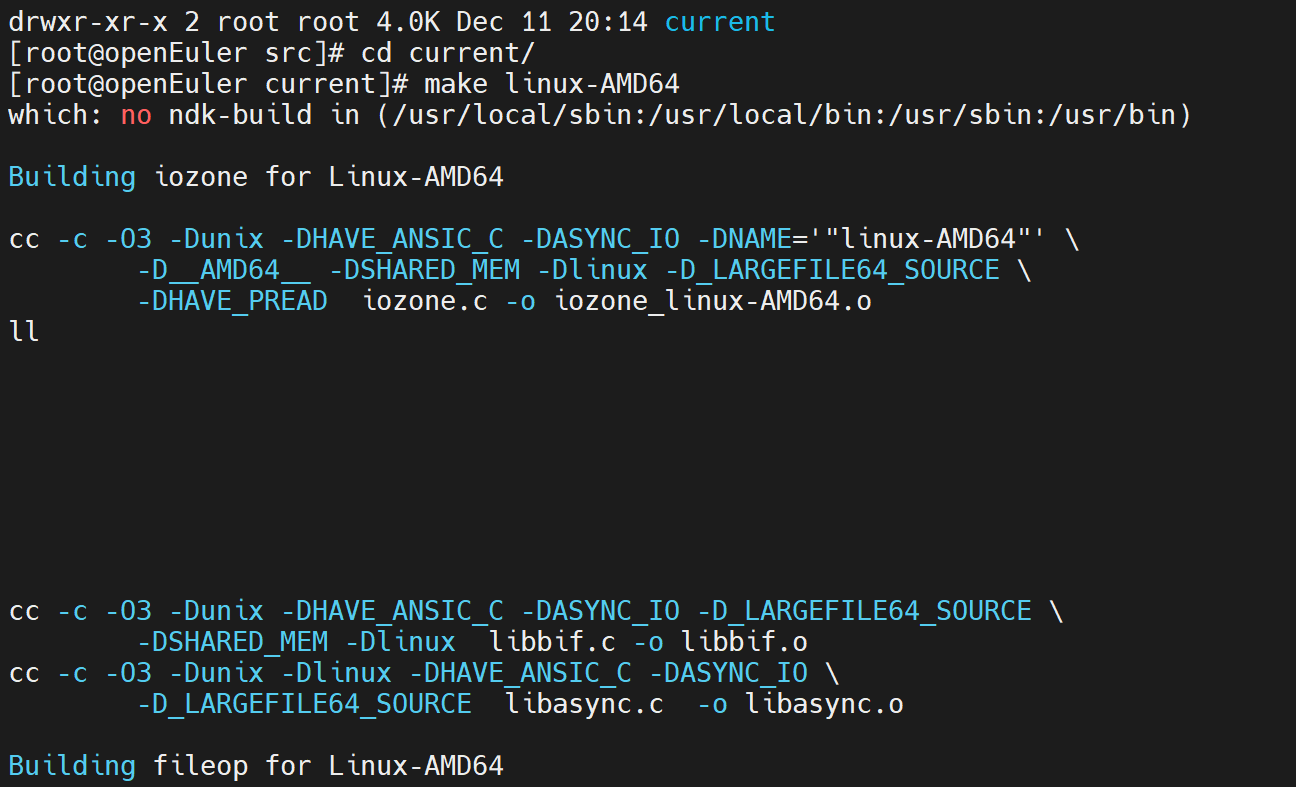
具体详情参见实验三任务5部分

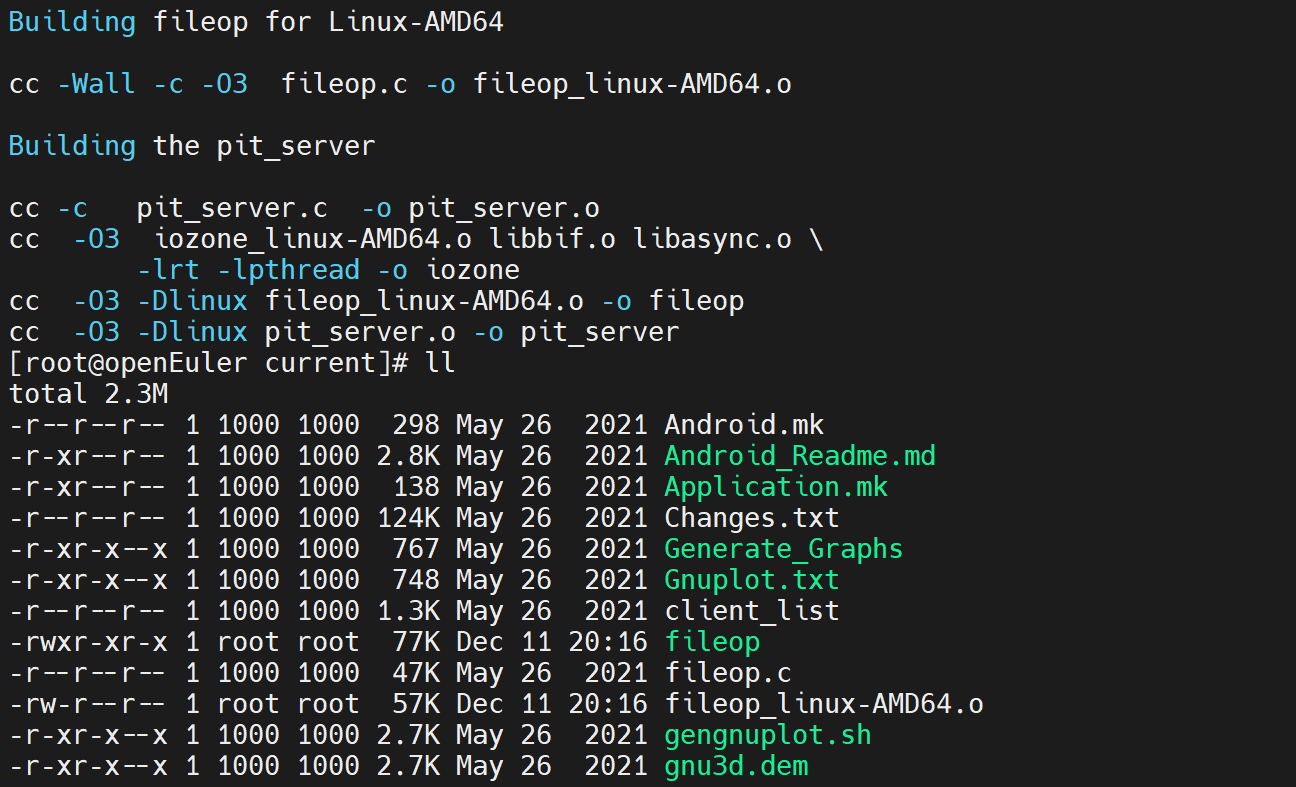
## 任务二：编写内核模块测试硬盘的读写速率，并与 iozone工具的测试结果比较

安装过程：

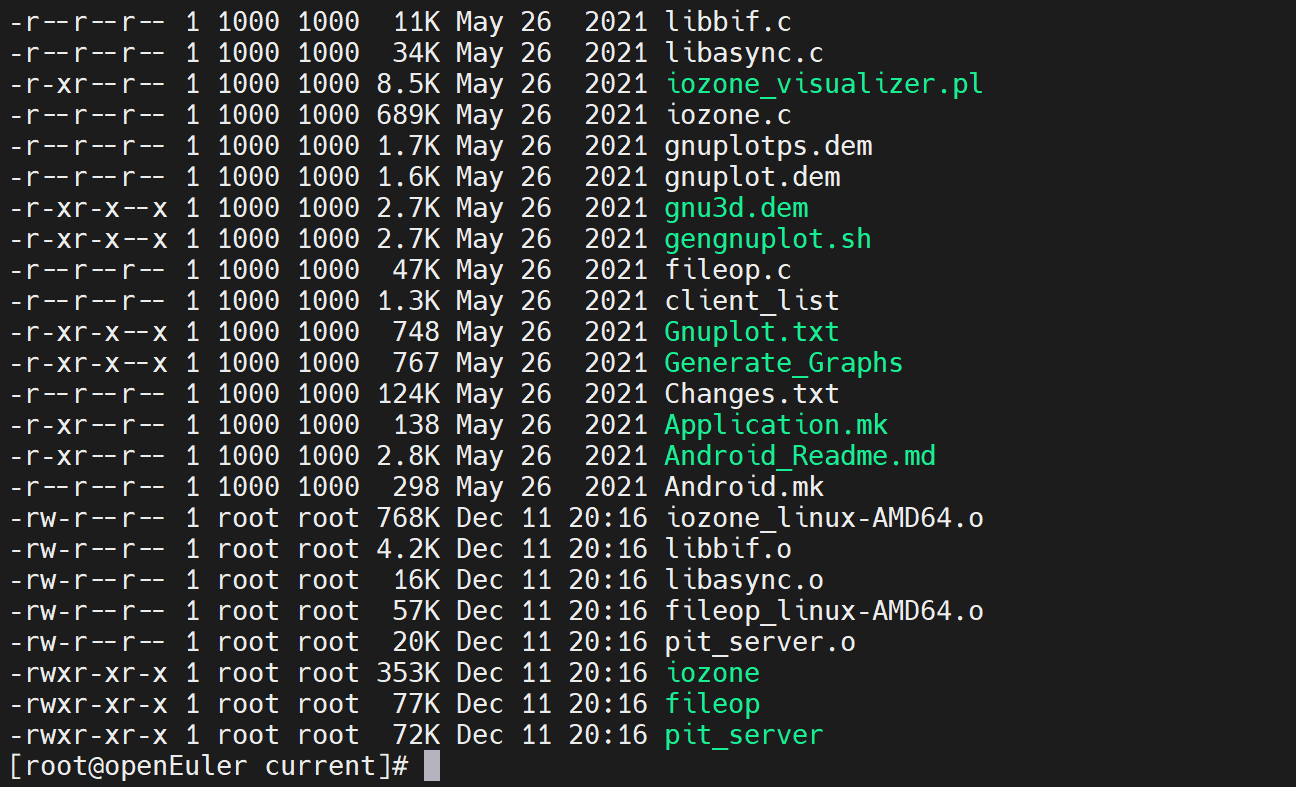




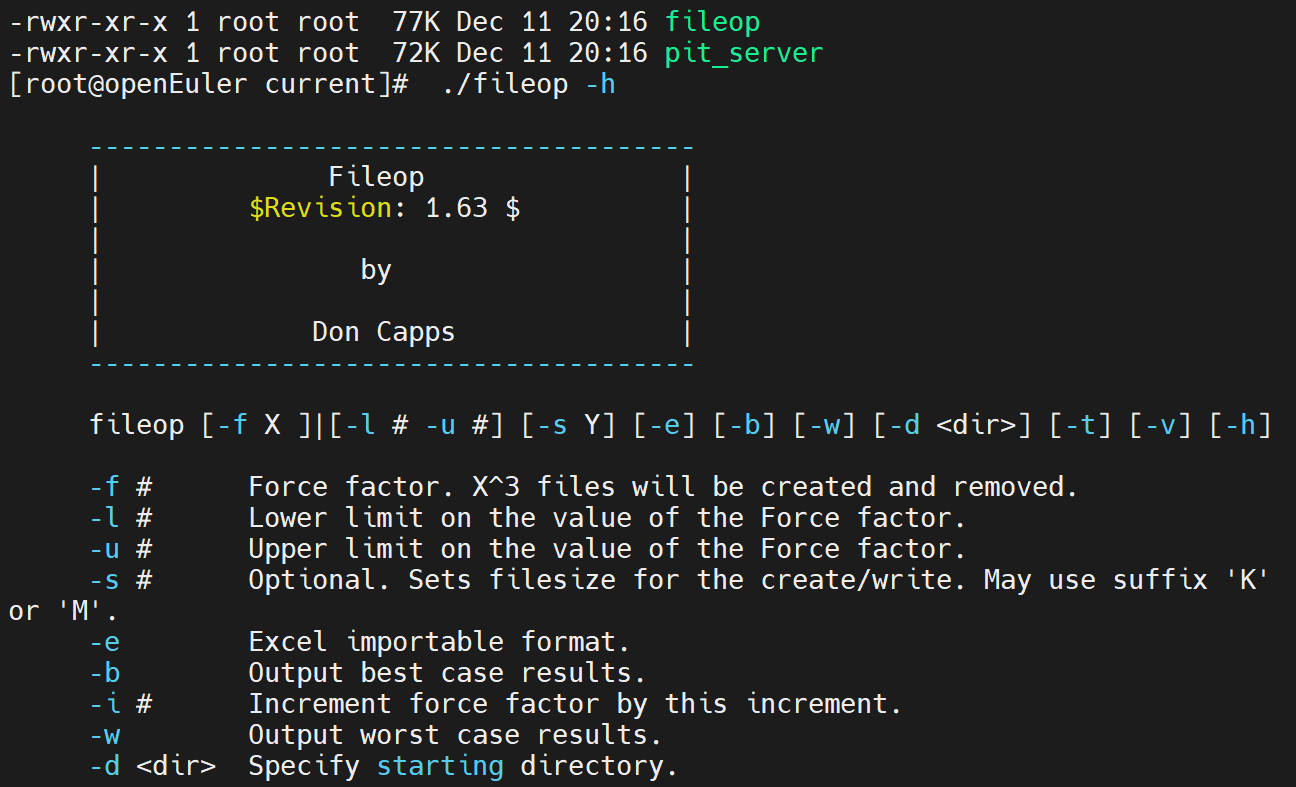


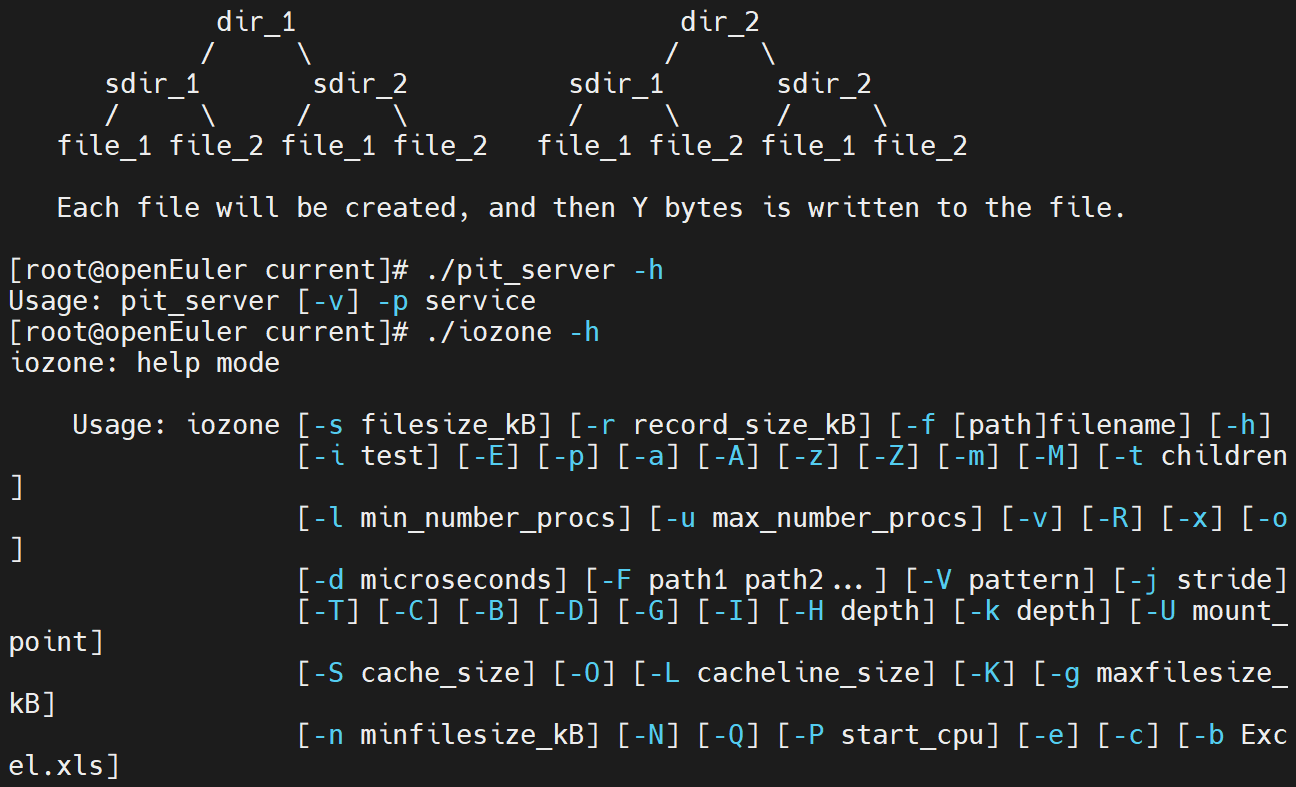


找到三个可执行文件



查看帮助验证安装正确





源码参考了优秀的指导书

写文件

write\_to\_disk.c

#*include* <*linux/module.h*>

#*include* <*linux/kernel.h*>

#*include* <*linux/fs.h*>

#*include* <*linux/rtc.h*>

#*define* buf\_size 1024

#*define* write\_times 524288

MODULE\_LICENSE("*GPL*");

*struct* timeval tv;

*static* *int* \_\_init write\_disk\_init(*void*)

{

*struct* file *\**fp\_write;

*char* buf[buf\_size];

*int* i;

*int* write\_start\_time;

*int* write\_start\_time\_u;

*int* write\_end\_time;

*int* write\_end\_time\_u;

*int* write\_time;

  loff\_t pos;

  printk("*Start write\_to\_disk module...\n*");

*for*(i = 0; i < buf\_size; i++)

  {

    buf[i] = i *+* '*0*';

  }

  fp\_write = filp\_open("*/home/tmp\_file*", O\_RDWR *|* O\_CREAT,*0*644);

*if* (IS\_ERR(fp\_write)) {

    printk("*Failed to open file...\n*");

*return* *-*1;

  }

  pos = 0;

  do\_gettimeofday(*&*tv);

  write\_start\_time = (*int*)tv.tv\_sec;

  write\_start\_time\_u = (*int*)tv.tv\_usec;

*for*(i = 0; i < write\_times; i++) {

    kernel\_write(fp\_write, buf, buf\_size, *&*pos);

  }

  do\_gettimeofday(*&*tv);

  write\_end\_time = (*int*)tv.tv\_sec;

  write\_end\_time\_u = (*int*)tv.tv\_usec;

  filp\_close(*fp\_write*, *NULL*);

  write\_time = (write\_end\_time *-* write\_start\_time)  *\** 1000000 *+* (write\_end\_time\_u *-* write\_start\_time\_u);

  printk(KERN\_ALERT "*Writing to file costs %d us\n*", write\_time);

  printk("*Writing speed is %d M/s\n*", buf\_size *\** write\_times */* write\_time);

*return* 0;

}

*static* *void* \_\_exit write\_disk\_exit(*void*)

{

  printk("*Exit write\_to\_disk module...\n*");

}

module\_init(write\_disk\_init);

module\_exit(*write\_disk\_exit*);

Makefile

*ifneq* ($(*KERNELRELEASE*),)

  obj-m := write\_to\_disk.o

*else*

  KERNELDIR ?= /root/raspberrypi-kernel

  PWD := $(*shell pwd*)

default:

  $(*MAKE*) -C $(*KERNELDIR*) M=$(*PWD*) modules

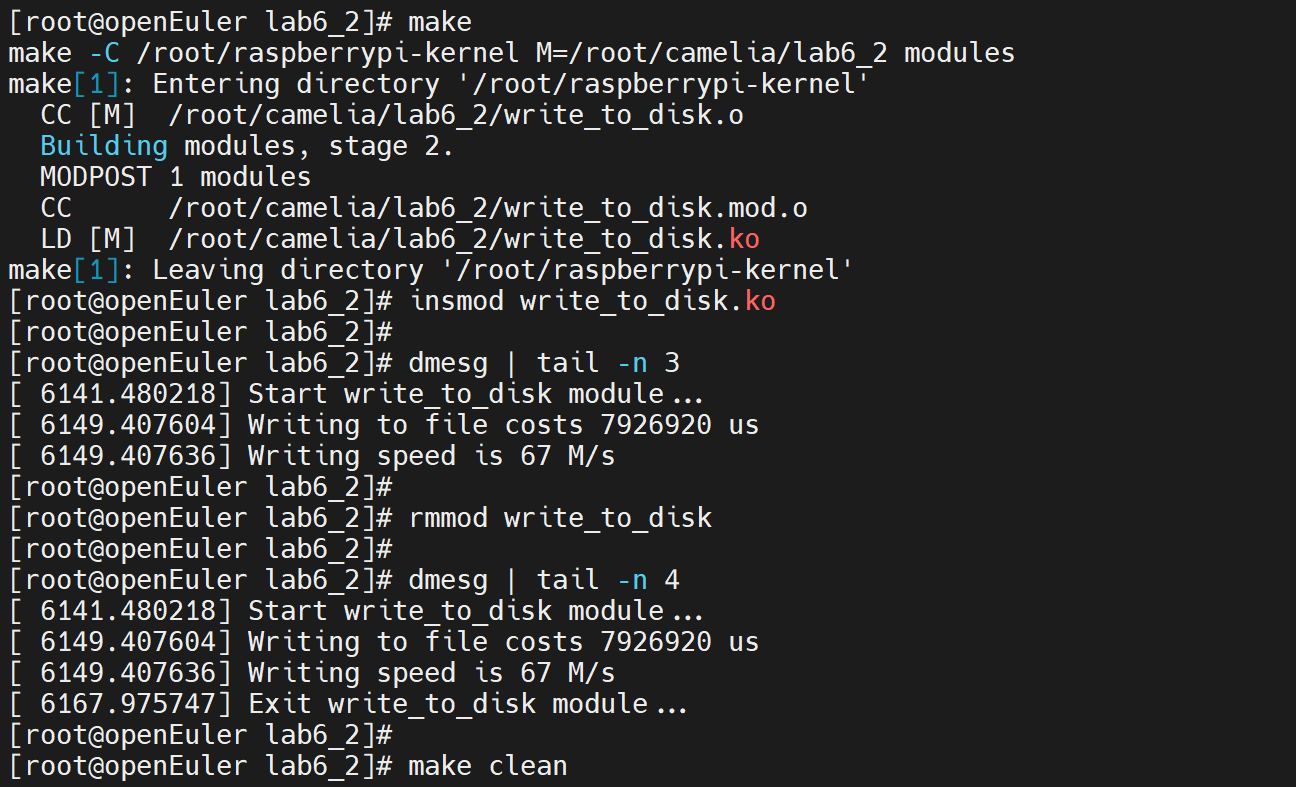
*endif*

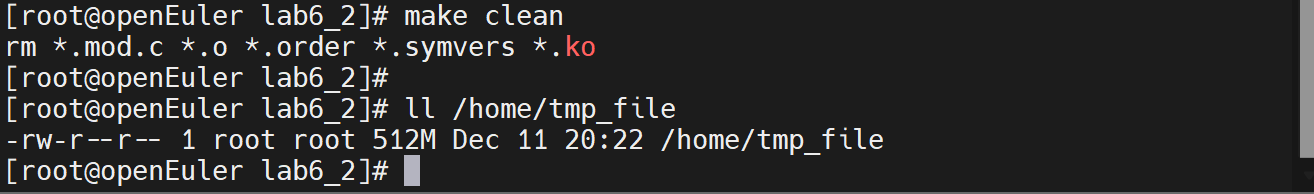
.PHONY:clean

clean:

*-*rm \*.mod.c \*.o \*.order \*.symvers \*.ko

效果流程：





读文件

read\_from\_disk.c

#*include* <*linux/module.h*>

#*include* <*linux/fs.h*>

#*include* <*linux/rtc.h*>

#*define* buf\_size 1024

#*define* read\_times 524288

MODULE\_LICENSE("*GPL*");

*struct* timeval tv;

*static* *int* \_\_init read\_disk\_init(*void*)

{

*struct* file *\**fp\_read;

*char* buf[buf\_size];

*int* i;

*int* read\_start\_time;

*int* read\_start\_time\_u;

*int* read\_end\_time;

*int* read\_end\_time\_u;

*int* read\_time;

  loff\_t pos;

  printk("*Start read\_from\_disk module...\n*");

  fp\_read = filp\_open("*/home/tmp\_file*", O\_RDONLY, 0);

*if* (IS\_ERR(fp\_read)) {

    printk("*Failed to open file...\n*");

*return* *-*1;

  }

  do\_gettimeofday(*&*tv);

  read\_start\_time = (*int*)tv.tv\_sec;

  read\_start\_time\_u = (*int*)tv.tv\_usec;

  pos = 0;

*for*(i = 0; i < read\_times; i++) {

    kernel\_read(fp\_read, buf, buf\_size, *&*pos);

  }

  do\_gettimeofday(*&*tv);

  read\_end\_time = (*int*)tv.tv\_sec;

  read\_end\_time\_u = (*int*)tv.tv\_usec;

  filp\_close(*fp\_read*, *NULL*);

  read\_time = (read\_end\_time *-* read\_start\_time)  *\** 1000000 *+* (read\_end\_time\_u *-* read\_start\_time\_u);

  printk(KERN\_ALERT "*Read file costs %d us\n*", read\_time);

  printk("*Reading speed is %d M/s\n*", buf\_size *\** read\_times */* read\_time);

*return* 0;

}

*static* *void* \_\_exit read\_disk\_exit(*void*)

{

  printk("*Exit read\_from\_disk module...\n*");

}

module\_init(read\_disk\_init);

module\_exit(*read\_disk\_exit*);

Makefile

*ifneq* ($(*KERNELRELEASE*),)

  obj-m := read\_from\_disk.o

*else*

  KERNELDIR ?= /root/raspberrypi-kernel

  PWD := $(*shell pwd*)

default:

  $(*MAKE*) -C $(*KERNELDIR*) M=$(*PWD*) modules

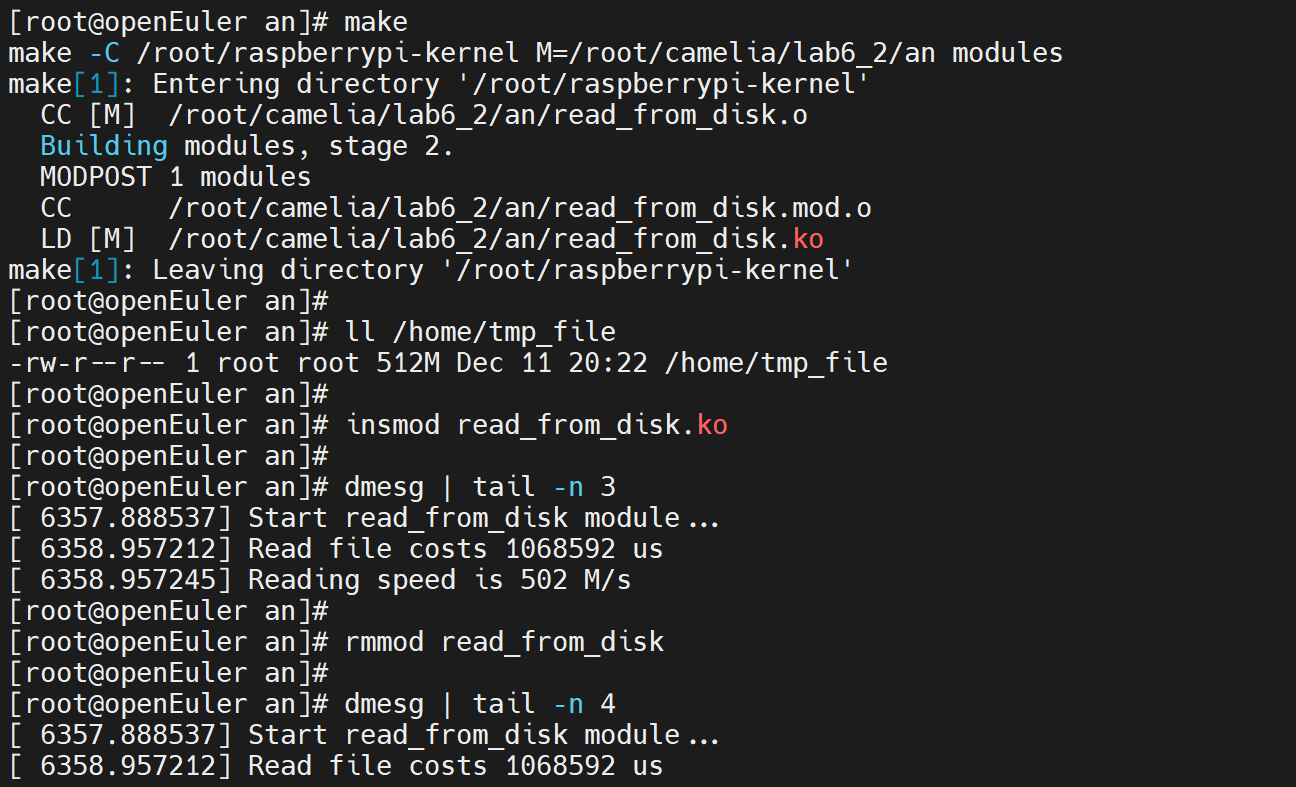
*endif*

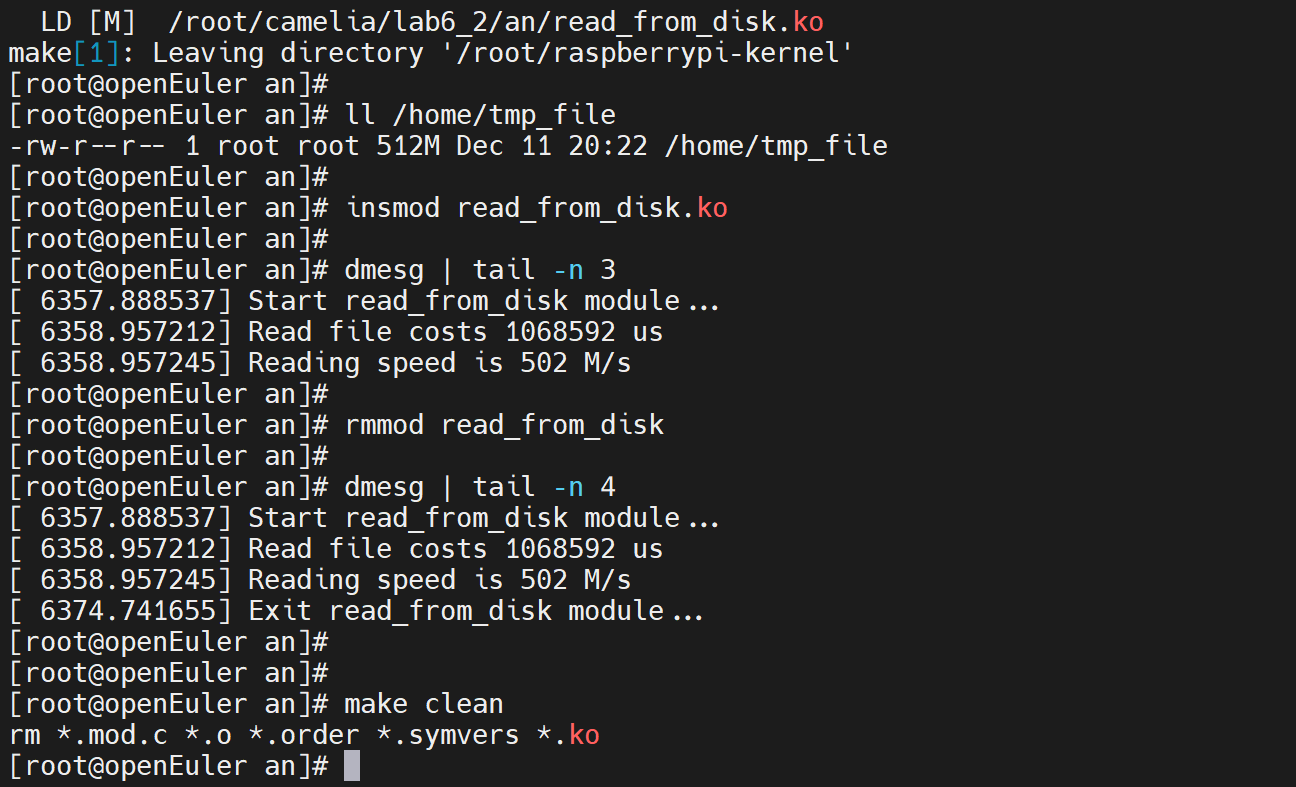
.PHONY:clean

clean:

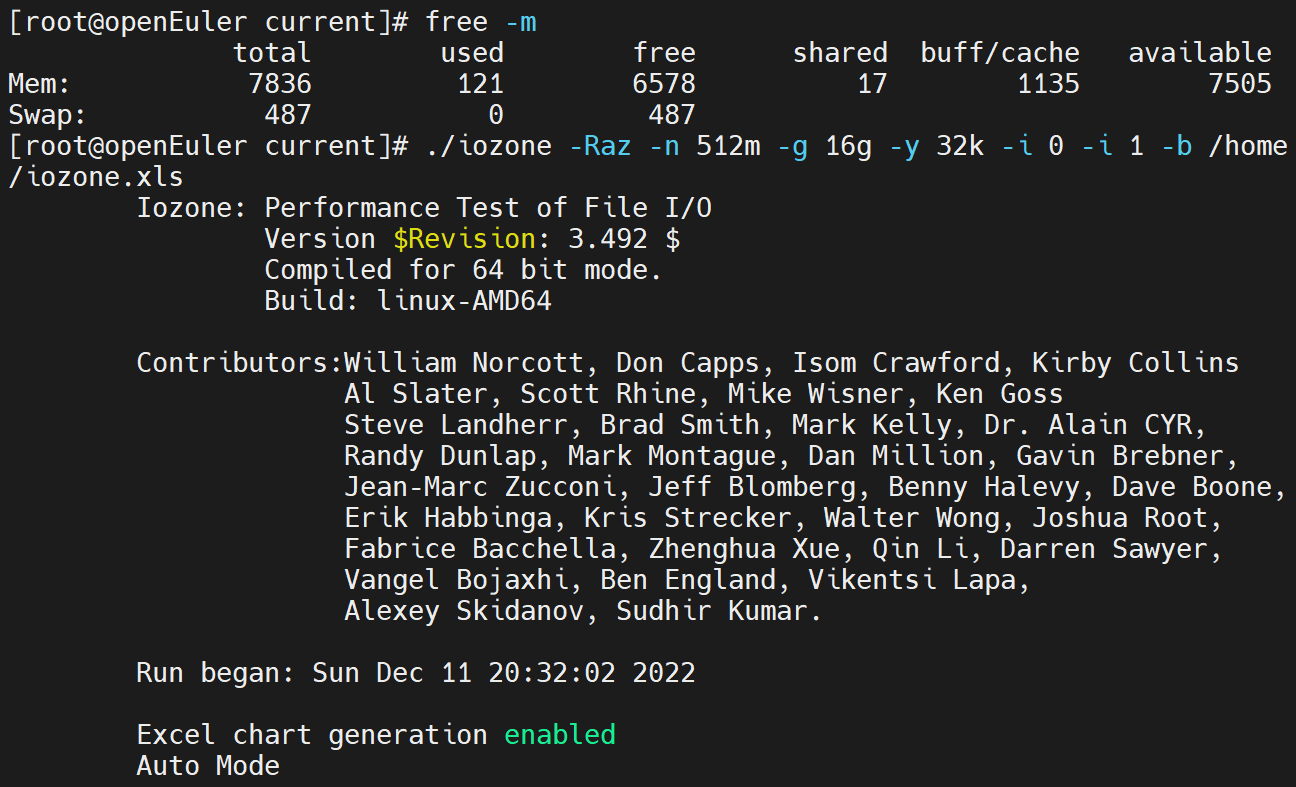
*-*rm \*.mod.c \*.o \*.order \*.symvers \*.ko

效果流程：

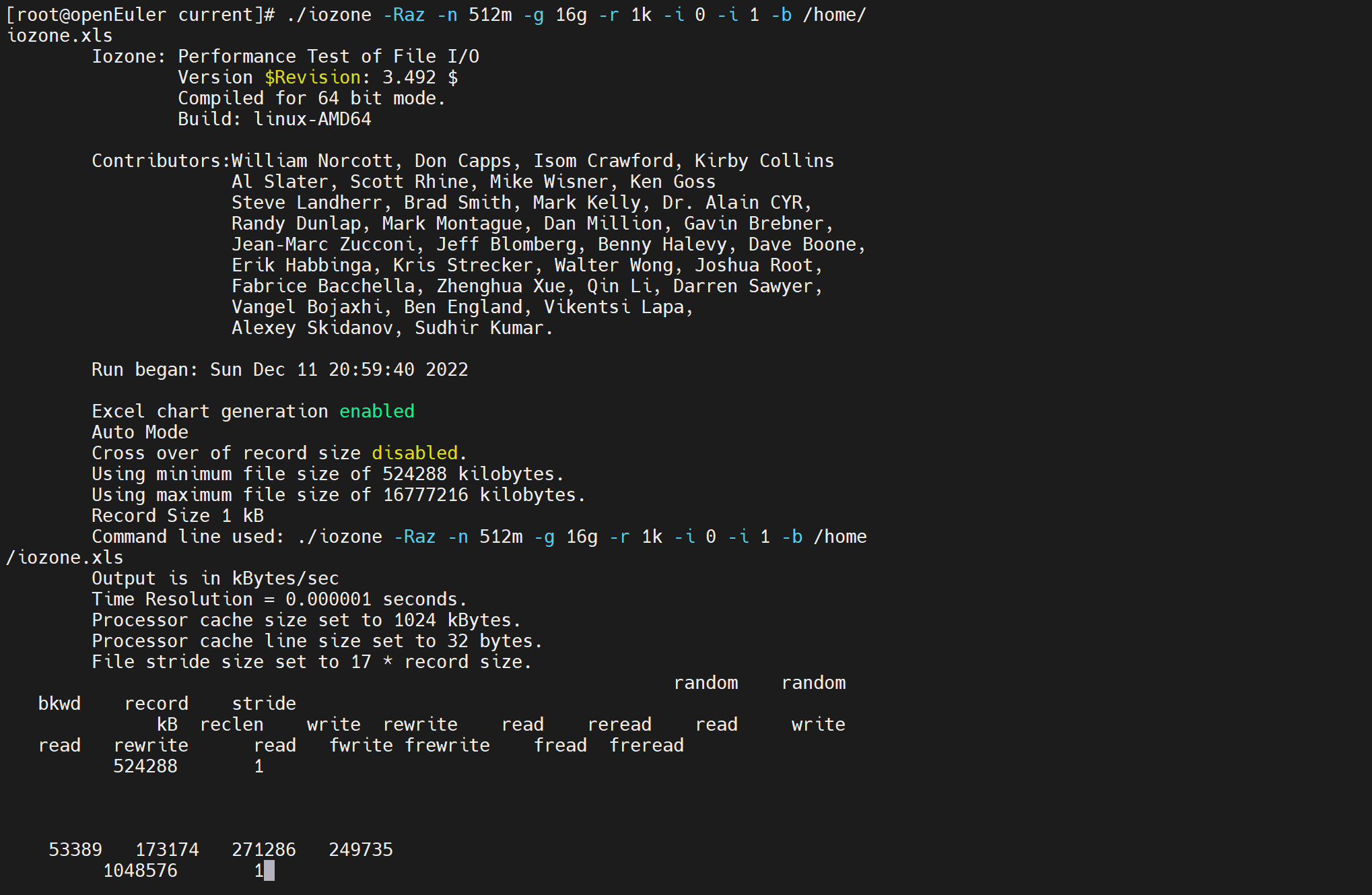




Iozone测试



部分效果展示：



Write: 53389 Read: 271286

换算单位 M/s

52.1376953125

264.927734375

比较内核态

结果分析： 第一列是测试文件的大小（kbytes）第二列的1是记录块大小，即1Kb；第三列是测试的传输速度。 比如图中“53389”，意思是测试文件大小为512M，以大小为1K（1024byte）的记录块进行传输，它的传输速度为53389/s，即约52.1376953125M/s。 对于内核模块程序中，写入的文件大小是512M，读取的文件是之前写入的大小为512M的文件；且读写的记录块都是1024byte；对应于iozone测试的文件大小为512M、记录块为1024byte的写速率是 53389 kb/s（即52.1376953125M/s），读速率是 271286 kb/s（即264.927734375M/s）。

对比分析用户态和内核态下测试的读写速率：在iozone工具的最终测试结果中，将传输速率与前文编写的内核模块的测试结果比较（注意单位一致性：内核模块单位Mb/s；iozone输出单位Kb/s）： 通过内核模块测试的写速率是：67M/s；读速率是502M/s。 通过iozone工具测试相同条件下的写速率是：52.1376953125M/s；读速率是 264.927734375M/s。 比较后可知：iozone所测试的读写传输速率，均小于内核模块的测试速率；即：用户态的读写速率比内核态更慢。

原因说明： 内核态：CPU可以访问内存所有数据, 包括外围设备, 例如硬盘、网卡；CPU也可以将自己从一个程序切换到另一个程序 用户态：只能受限的访问内存, 且不允许访问外围设备；占用CPU的能力被剥夺，CPU资源可以被其他程序获取。 也就是说：内核态的程序可以直接读写硬盘；而用户态的程序需要先切换至内核态，再由内核态进行系统调用来读写磁盘。因此，用户态的硬盘读写速率比内核态更耗时。