# 实验三(Guide)

## 任务一：创建并运行内核线程（15min）

源码参考了优秀的指导书

kthread.c

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

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

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

MODULE\_LICENSE("*GPL*");

#*define* BUF\_SIZE 20

*static* *struct* task\_struct *\**myThread = *NULL*;

*static* *int* print(*void* *\*data*)

{

*while*(!kthread\_should\_stop()){

    printk("*New kthread is running.*");

    msleep(2000);

  }

*return* 0;

}

*static* *int* \_\_init kthread\_init(*void*)

{

  printk("*Create kernel thread!\n*");

  myThread = kthread\_run(print, *NULL*, "*new\_kthread*");

*return* 0;

}

*static* *void* \_\_exit kthread\_exit(*void*)

{

  printk("*Kill new kthread.\n*");

*if*(myThread)

    kthread\_stop(myThread);

}

module\_init(kthread\_init);

module\_exit(kthread\_exit);

Makefile

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

  obj-m := kthread.o

*else*

  KERNELDIR ?= /root/raspberrypi-kernel

  PWD := $(*shell pwd*)

default:

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

*endif*

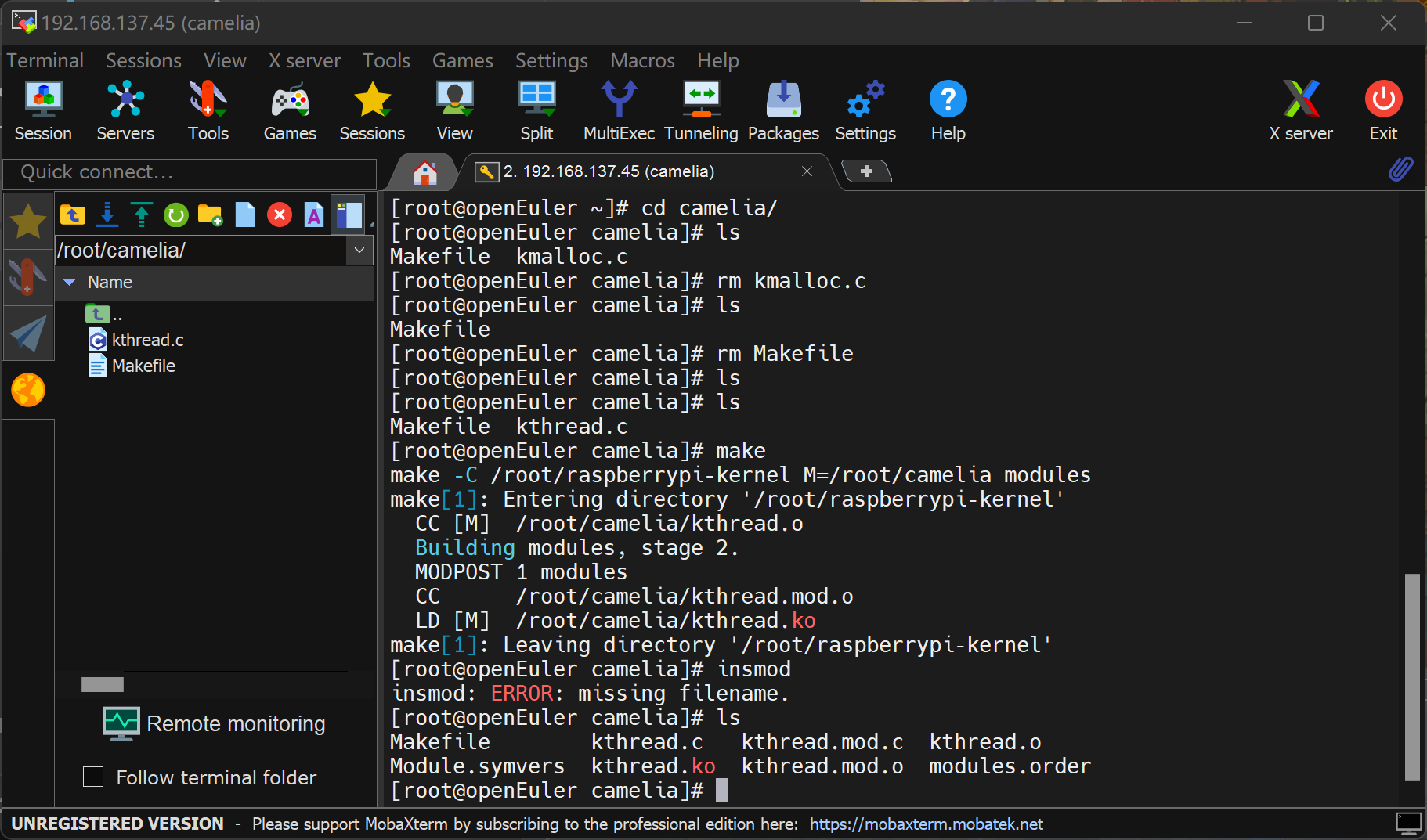
.PHONY:clean

clean:

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

Make编译

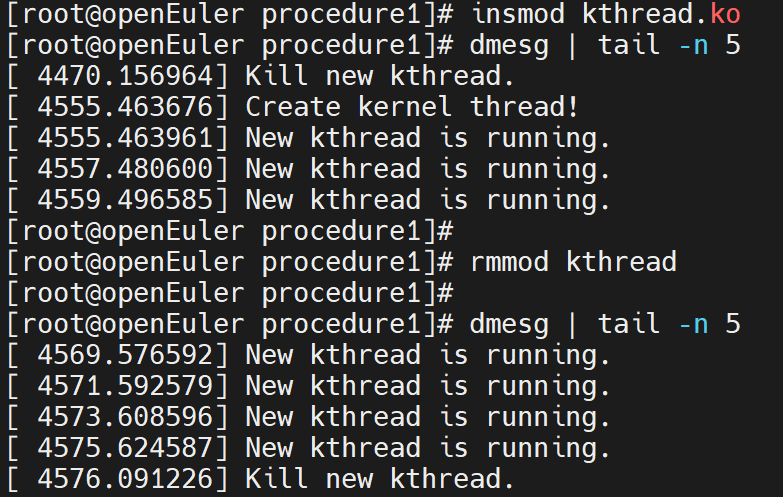
获取到.ko文件



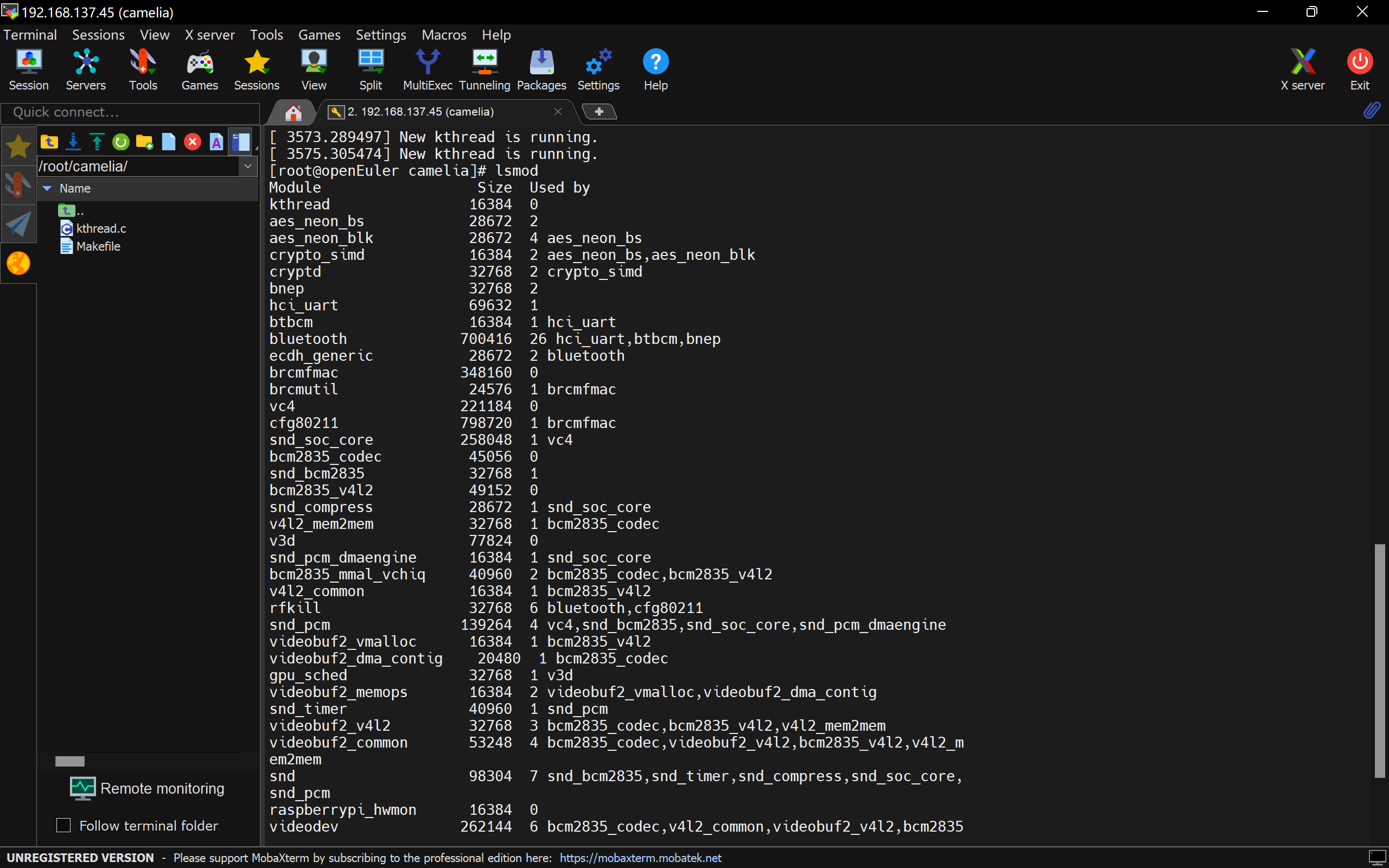
模块加载进内核insmod kthread.ko

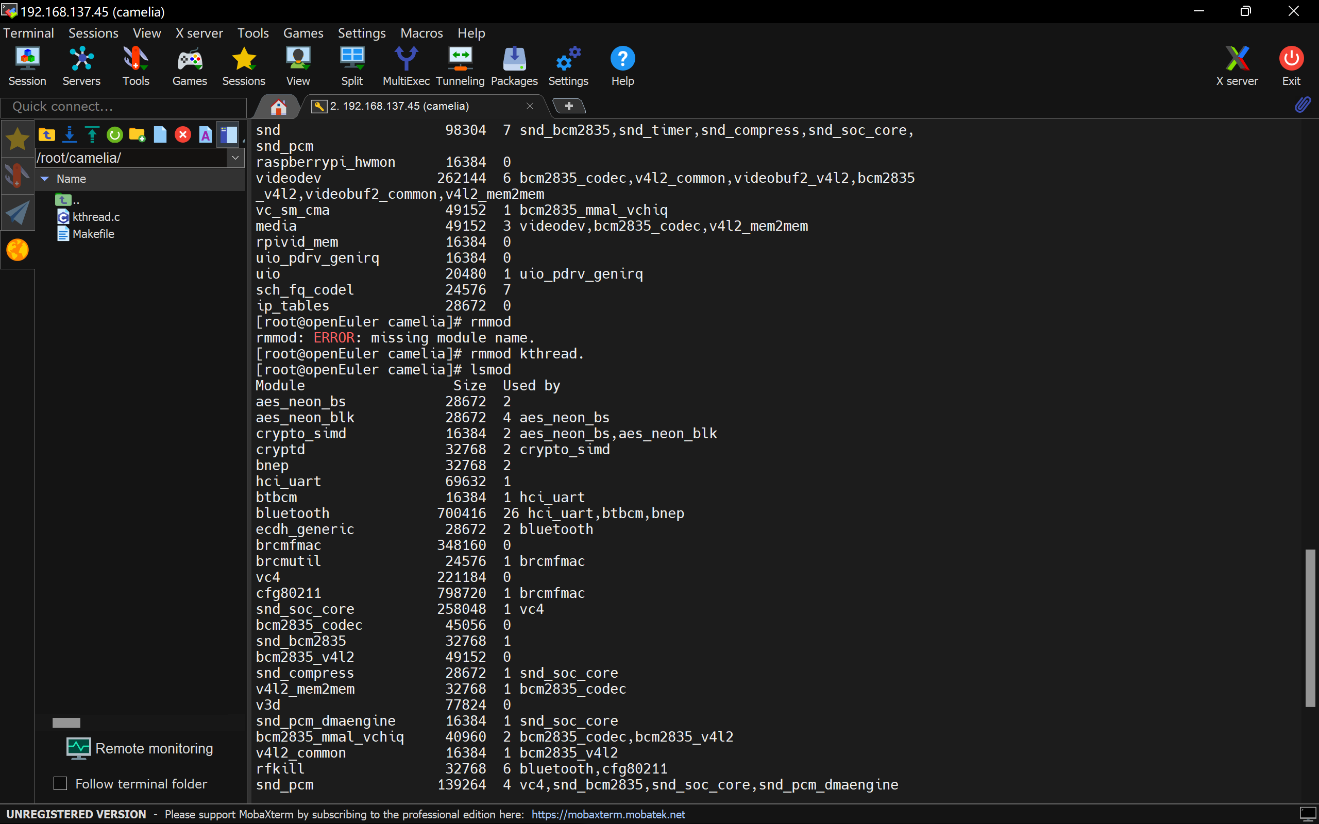
查看加载内容 -- 创建新内核线程（第一项输出是上次没清除的）

卸载内核模块 再度查看内容 成功在模块退出时杀死线程

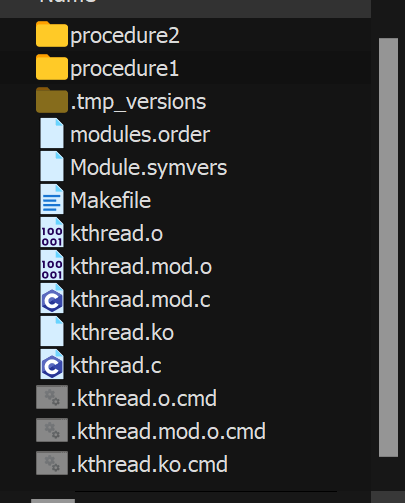


lsmod查看内核模块（加载与卸载模块前后对比图）





最终文件结构：



## 任务二：打印输出当前系统 CPU 负载情况

后续过程与任务一基本一致，不再赘述，仅提供源码与内核功能展示。

cpu\_loadavg.c

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

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

MODULE\_LICENSE("*GPL*");

*char* tmp\_cpu\_load[5] = {'*\0*'};

*static* *int* get\_loadavg(*void*)

{

*struct* file *\**fp\_cpu;

  loff\_t pos = 0;

*char* buf\_cpu[10];

  fp\_cpu = filp\_open("*/proc/loadavg*", O\_RDONLY, 0);

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

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

*return* *-*1;

  }

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

  strncpy(tmp\_cpu\_load, buf\_cpu, 4);

  filp\_close(fp\_cpu, *NULL*);

*return* 0;

}

*static* *int* \_\_init cpu\_loadavg\_init(*void*)

{

  printk("*Start cpu\_loadavg!\n*");

*if*(0 != get\_loadavg()){

    printk("*Failed to read loadarvg file!\n*");

*return* *-*1;

  }

  printk("*The cpu loadavg in one minute is: %s\n*", tmp\_cpu\_load);

*return* 0;

}

*static* *void* \_\_exit cpu\_loadavg\_exit(*void*)

{

  printk("*Exit cpu\_loadavg!\n*");

}

module\_init(cpu\_loadavg\_init);

module\_exit(*cpu\_loadavg\_exit*);

Makefile

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

  obj-m := cpu\_loadavg.o

*else*

  KERNELDIR ?= /root/raspberrypi-kernel

  PWD := $(*shell pwd*)

default:

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

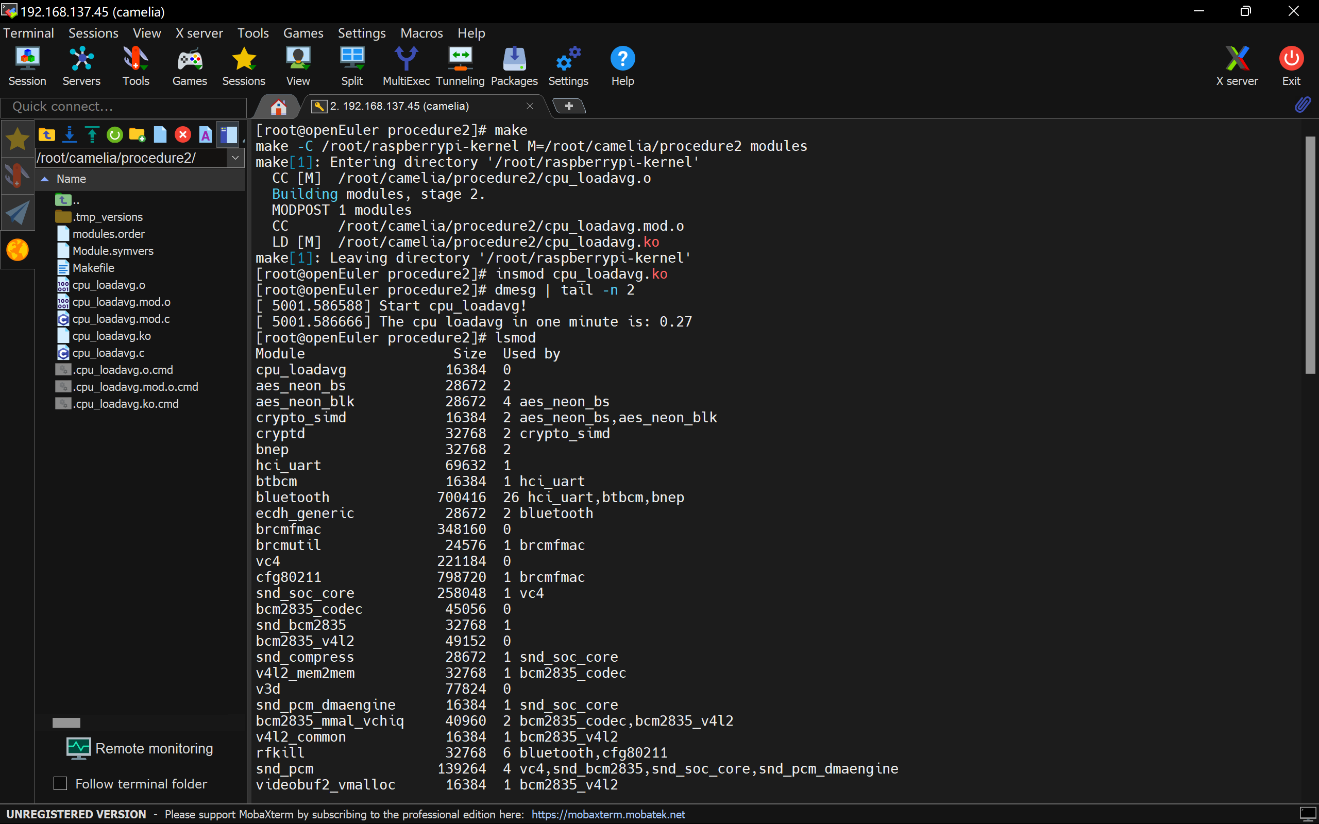
*endif*

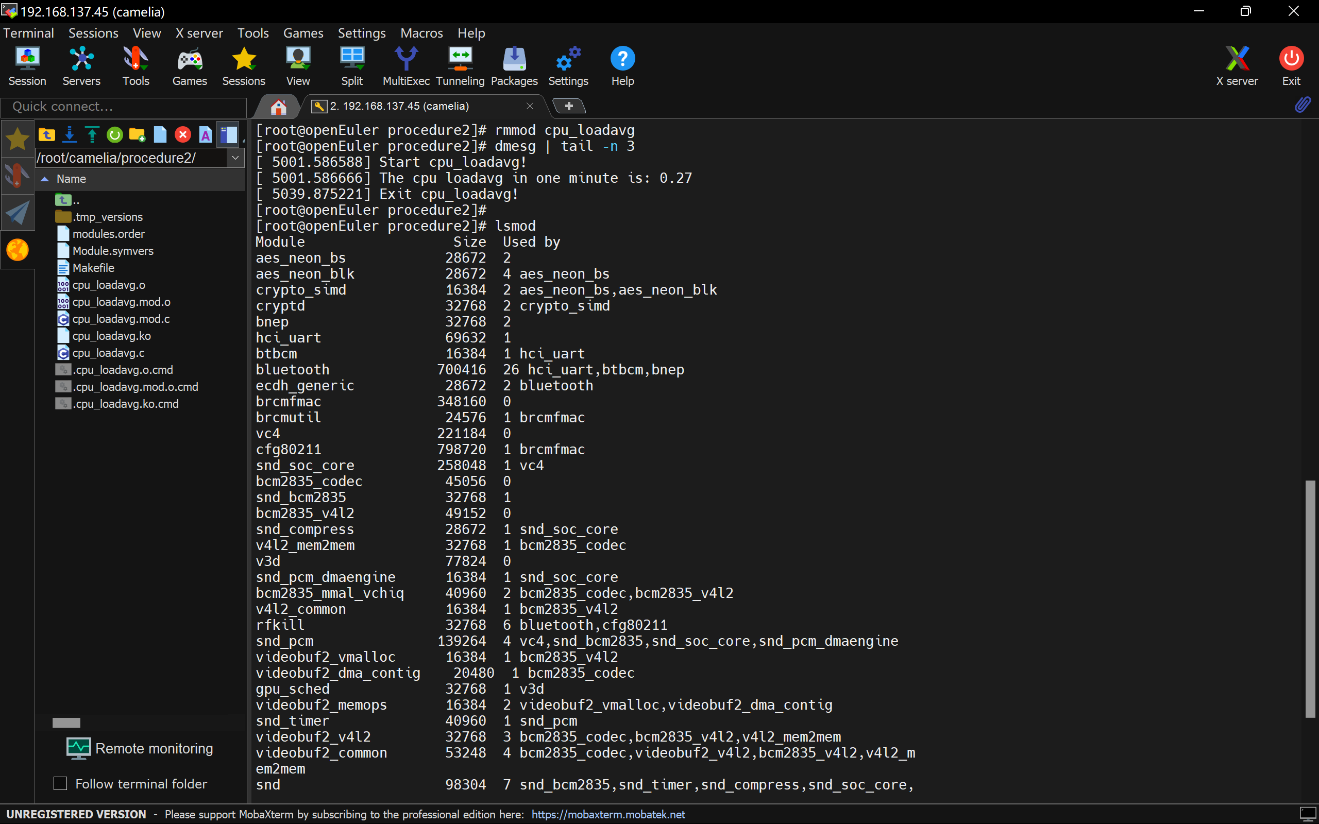
.PHONY:clean

clean:

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

两张分别为加载后与退出后





### 任务三：打印输出当前处于运行状态的进程的 PID 和名字

process\_info.c

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

#*include* <*linux/sched/signal.h*>

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

MODULE\_LICENSE("*GPL*");

*struct* task\_struct *\**p;

*static* *int* \_\_init process\_info\_init(*void*)

{

  printk("*Start process\_info!\n*");

  for\_each\_process(p){

*if*(p->state == 0)

      printk("*1)name:%s  2)pid:%d  3)state:%ld\n*", p->comm, p->pid, p->state);

  }

*return* 0;

}

*static* *void* \_\_exit process\_info\_exit(*void*)

{

  printk("*Exit process\_info!\n*");

}

module\_init(process\_info\_init);

module\_exit(*process\_info\_exit*);

Makefile

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

  obj-m := process\_info.o

*else*

  KERNELDIR ?= /root/raspberrypi-kernel

  PWD := $(*shell pwd*)

default:

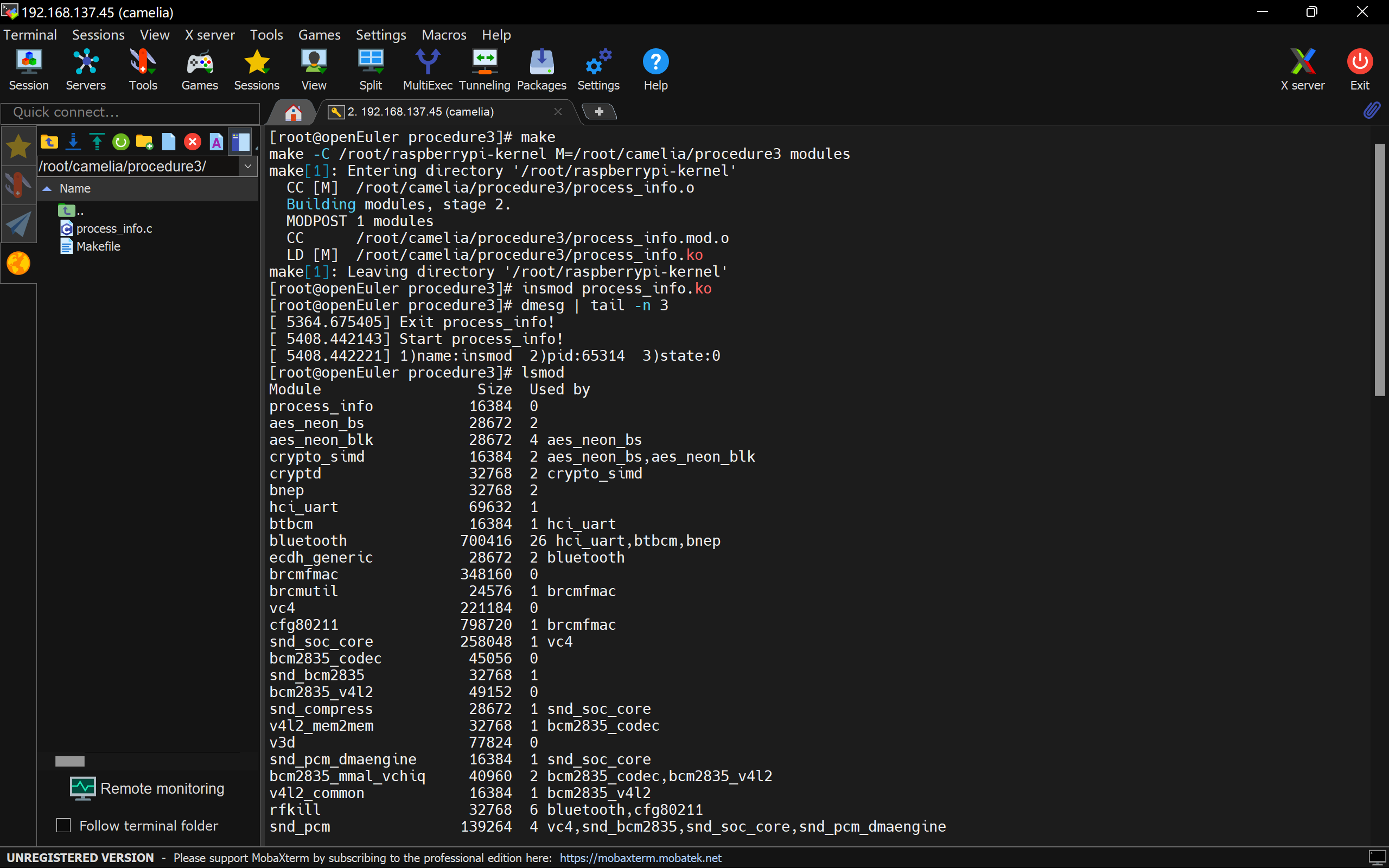
  $(*MAKE*) -C $(*KERNELDIR*) M=$(*PWD*) modules

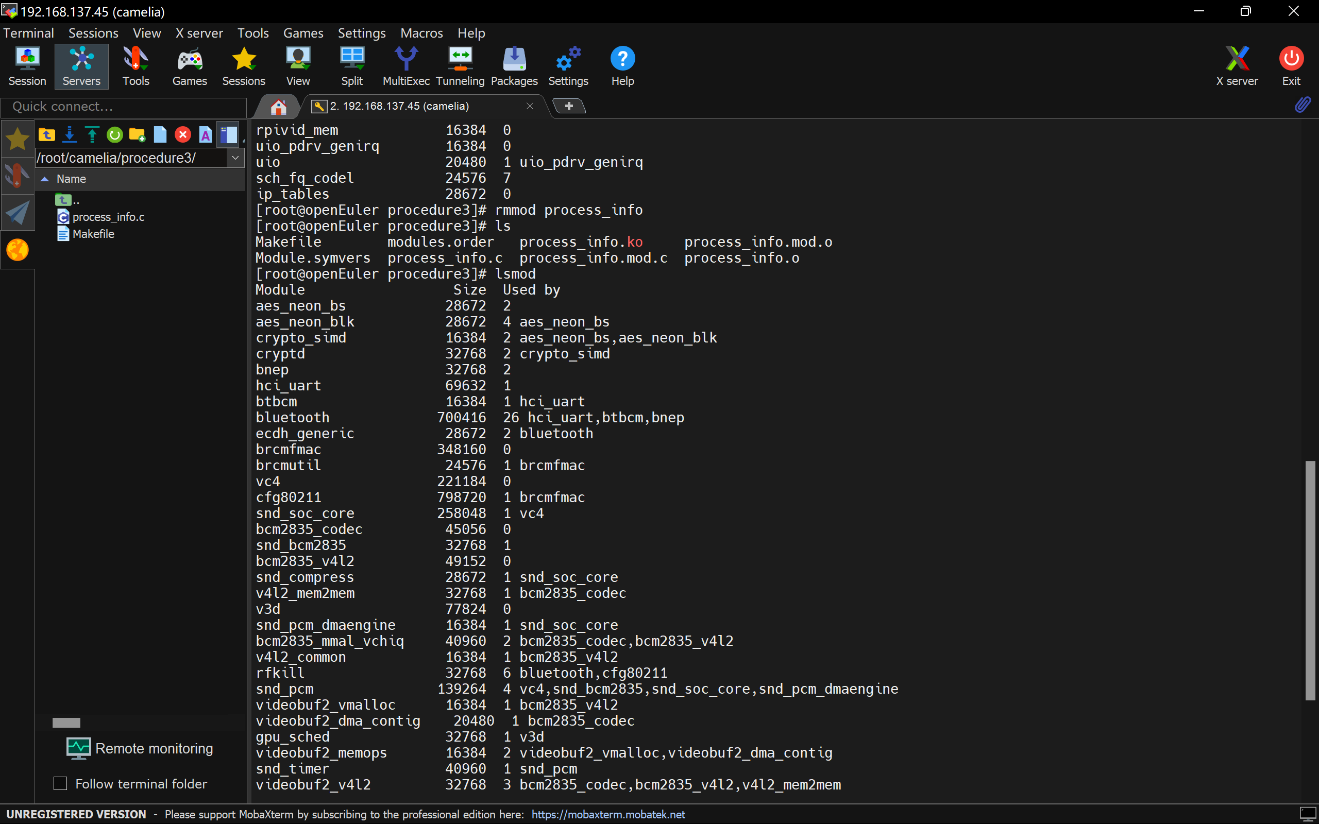
*endif*

.PHONY:clean

clean:

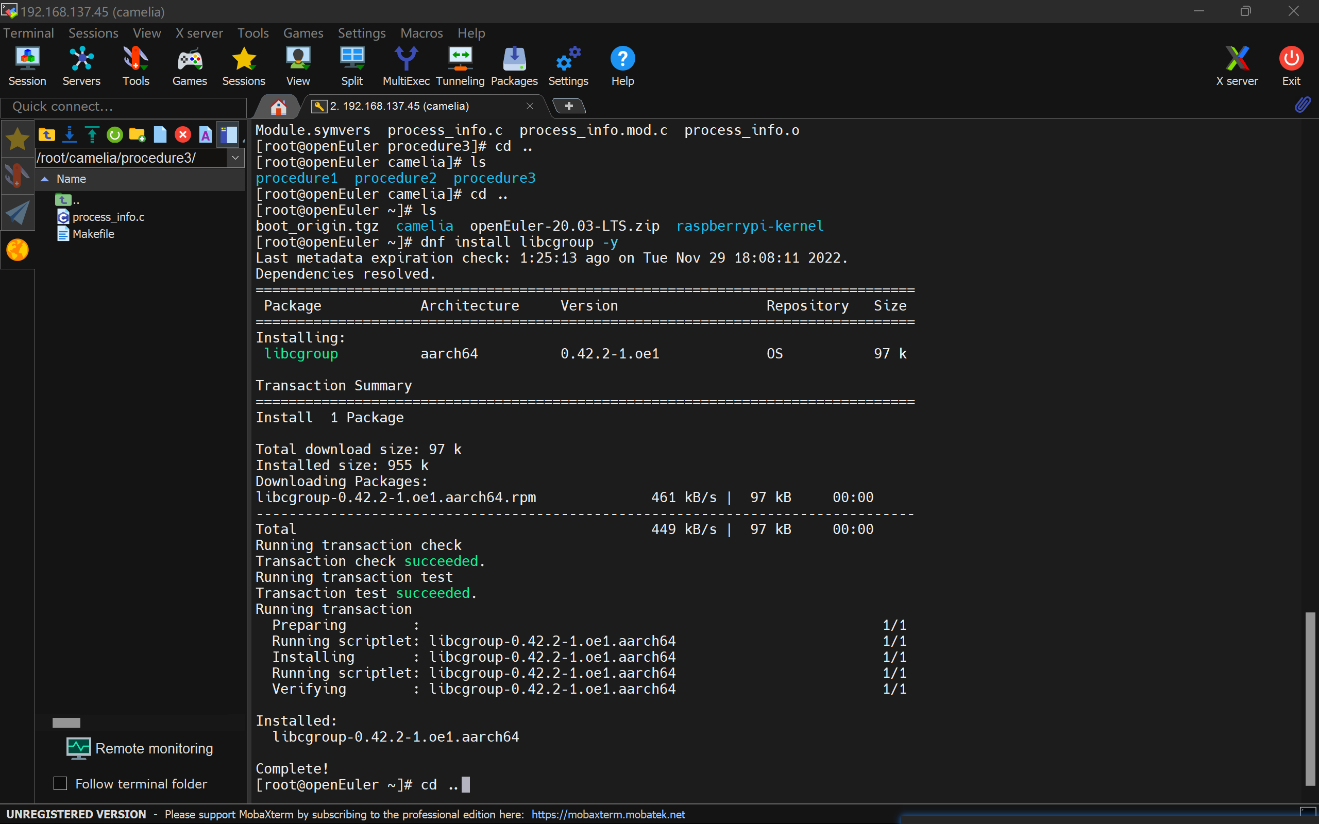
*-*rm \*.mod.c \*.o \*.order \*.symvers \*.ko

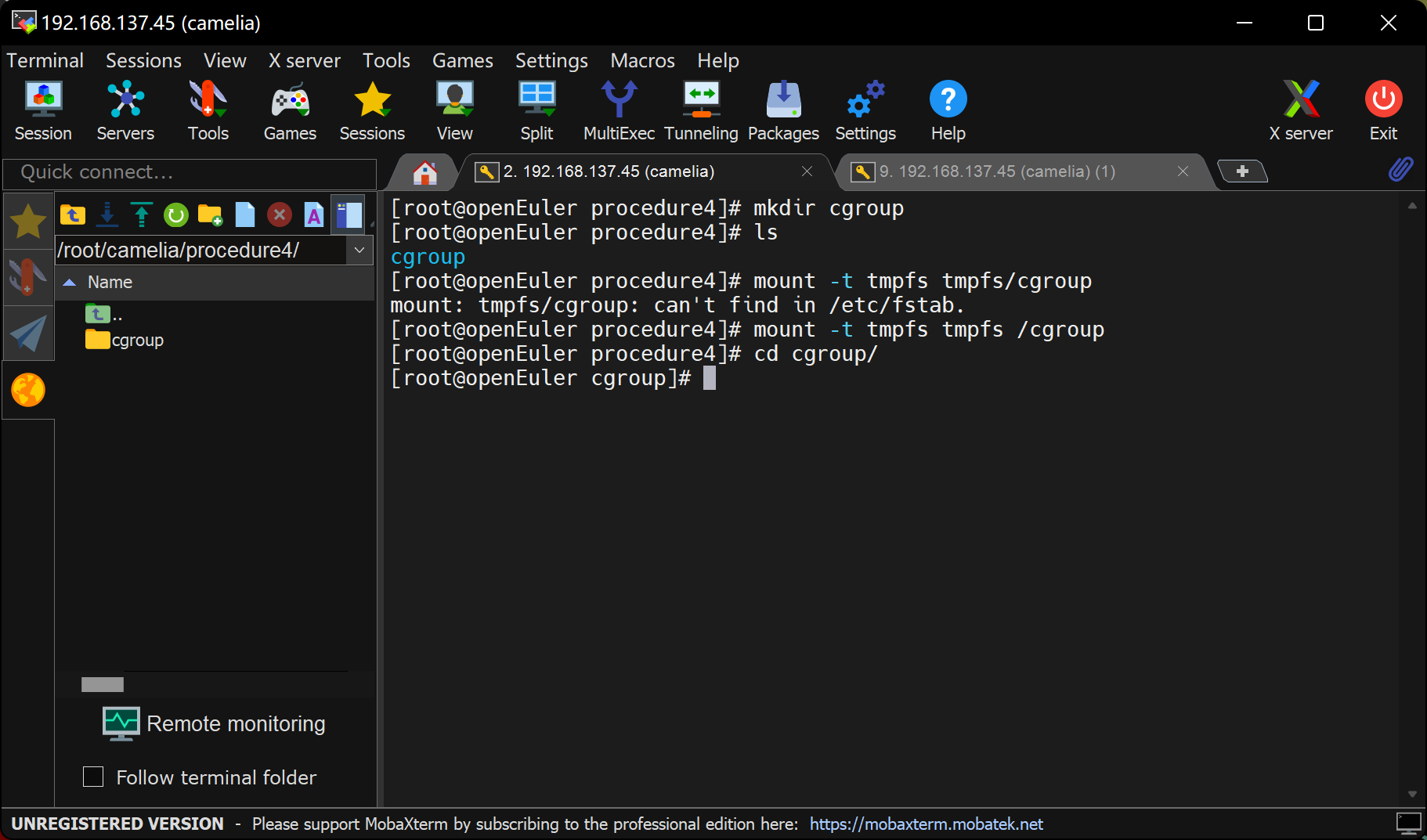


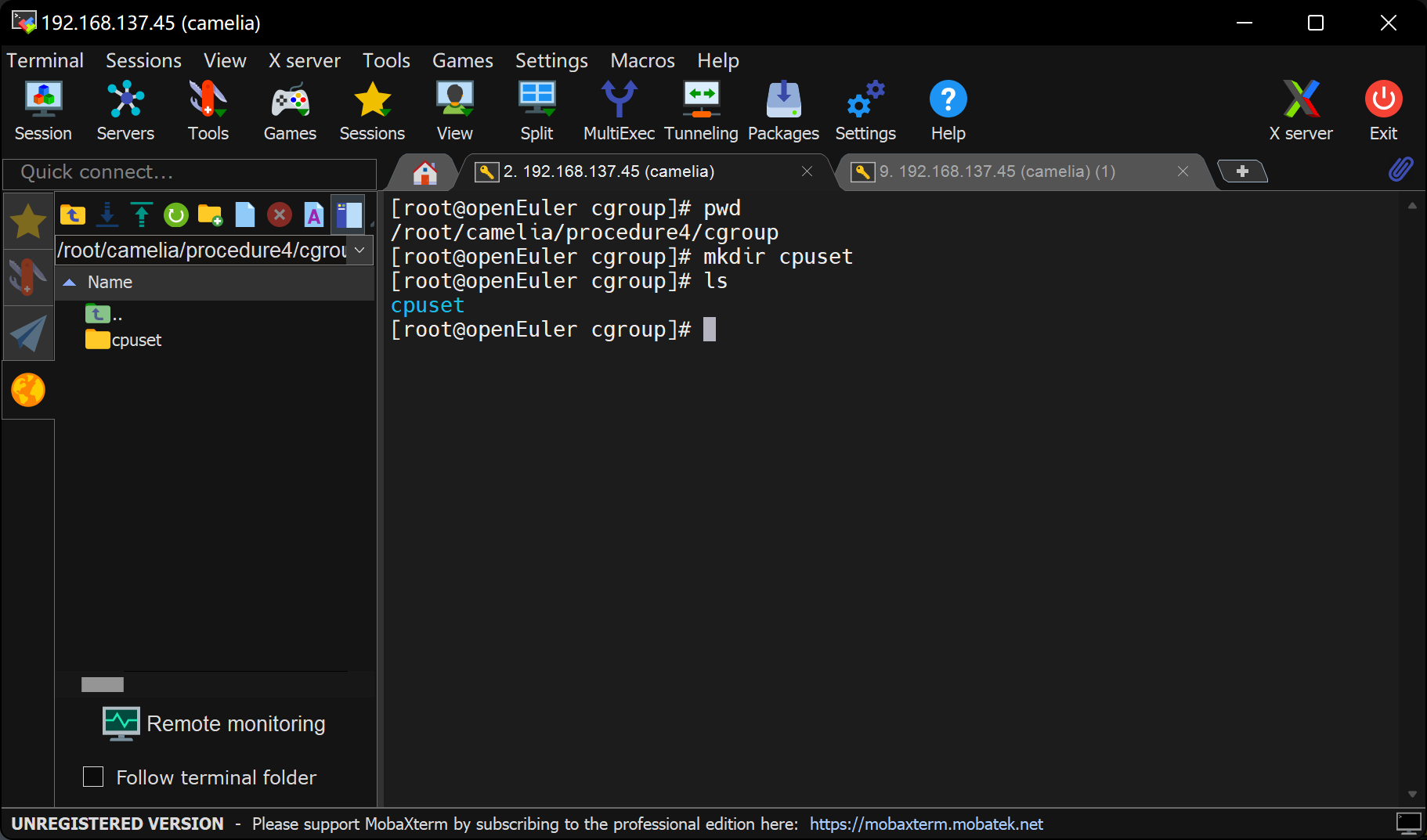


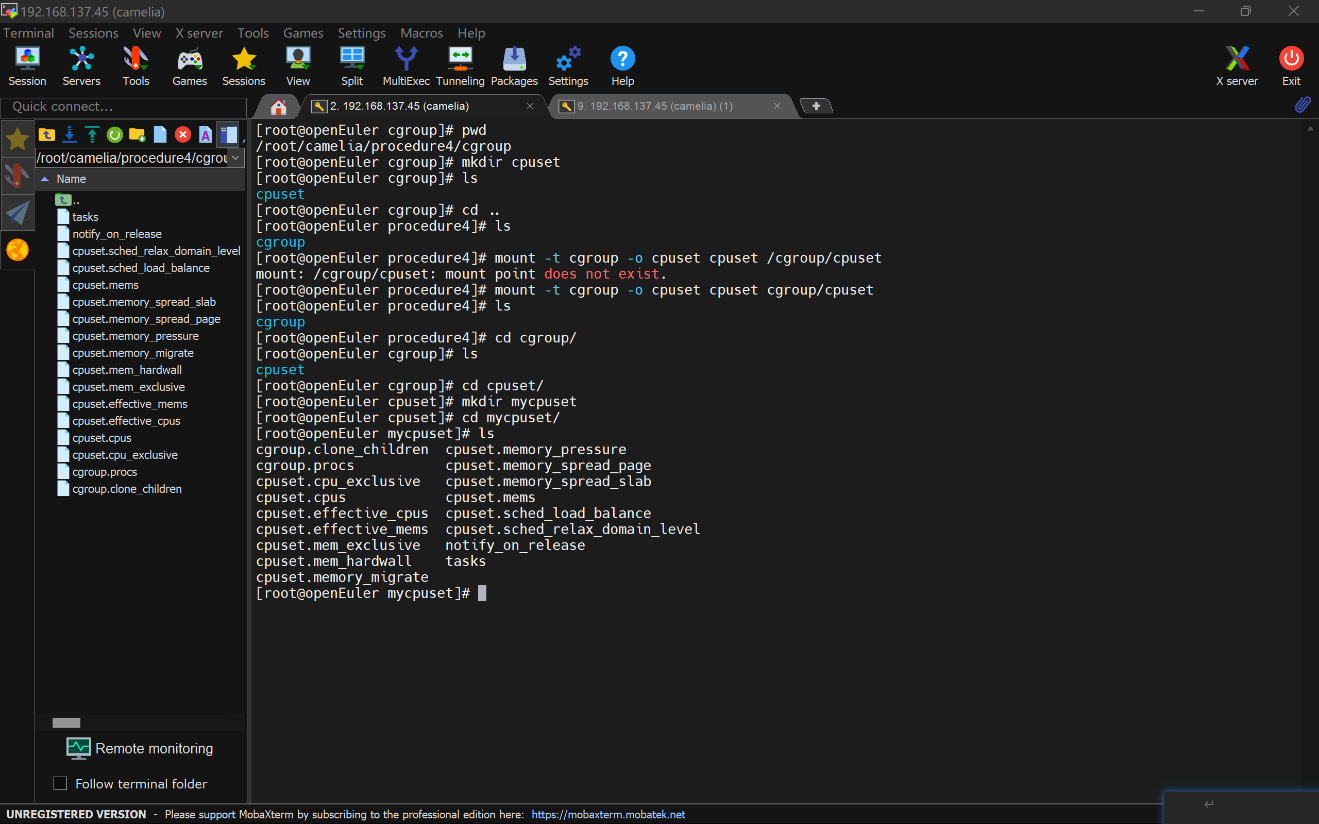
**任务4：使用 cgroup 实现限制 CPU 核数**

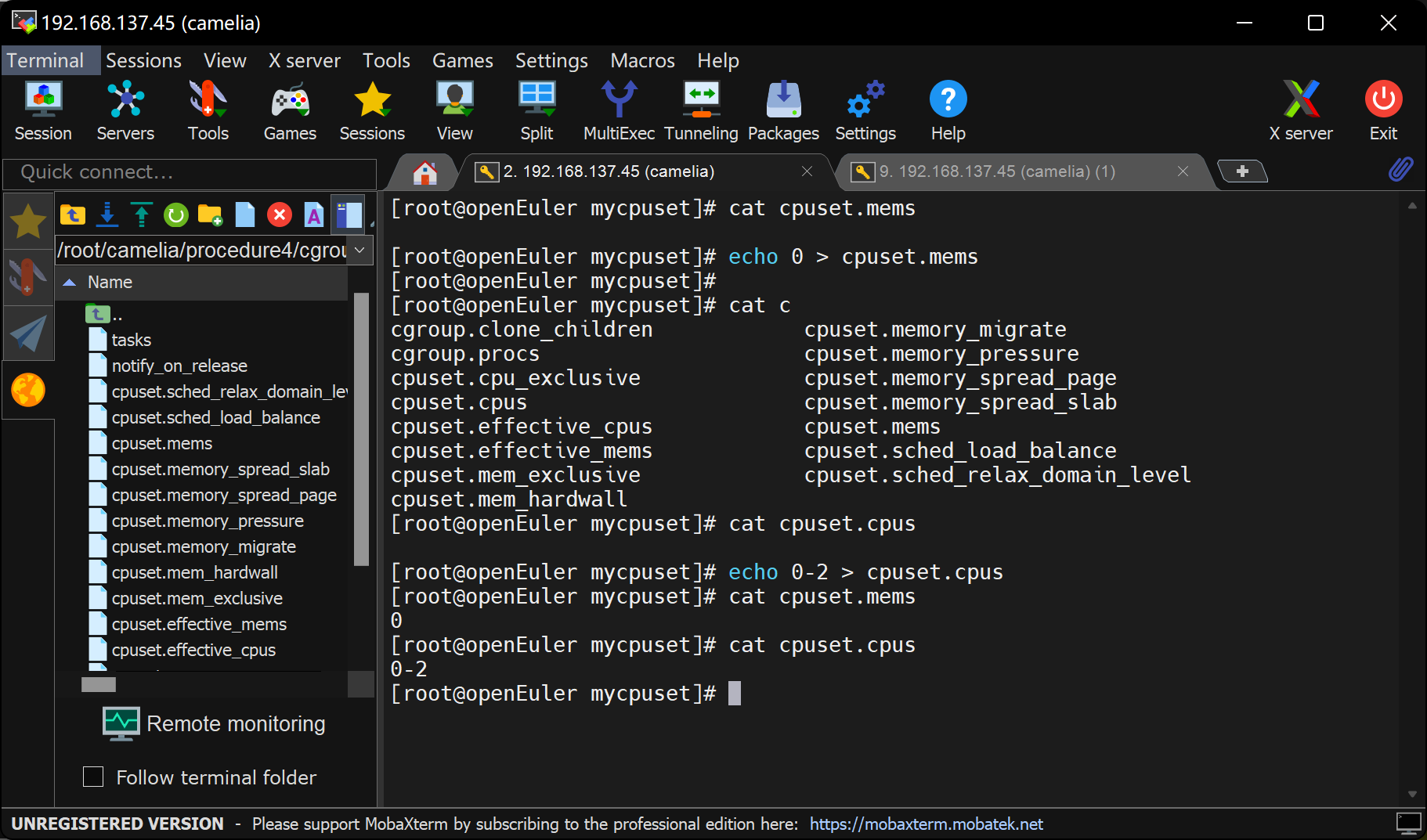
见下方流程截图

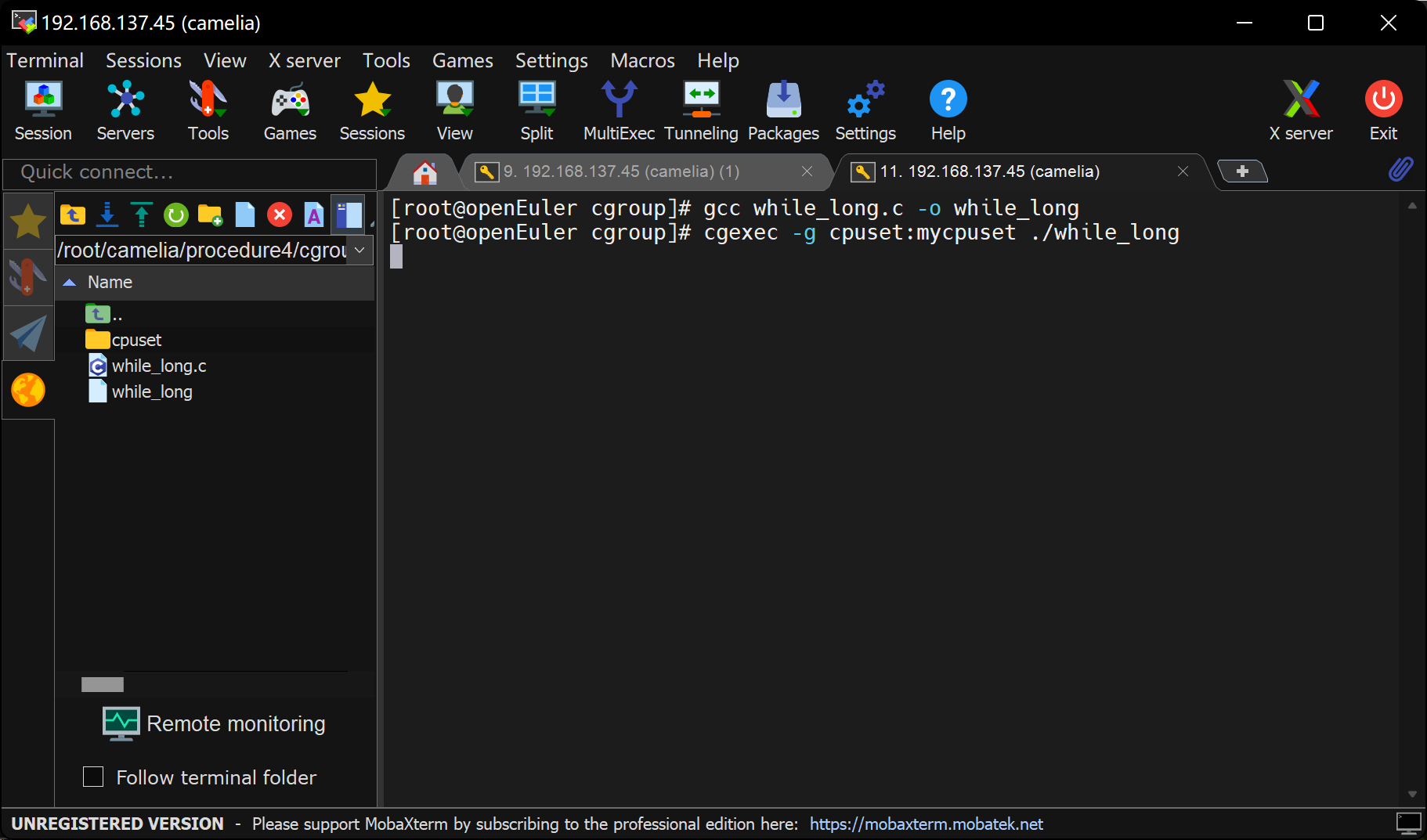
****

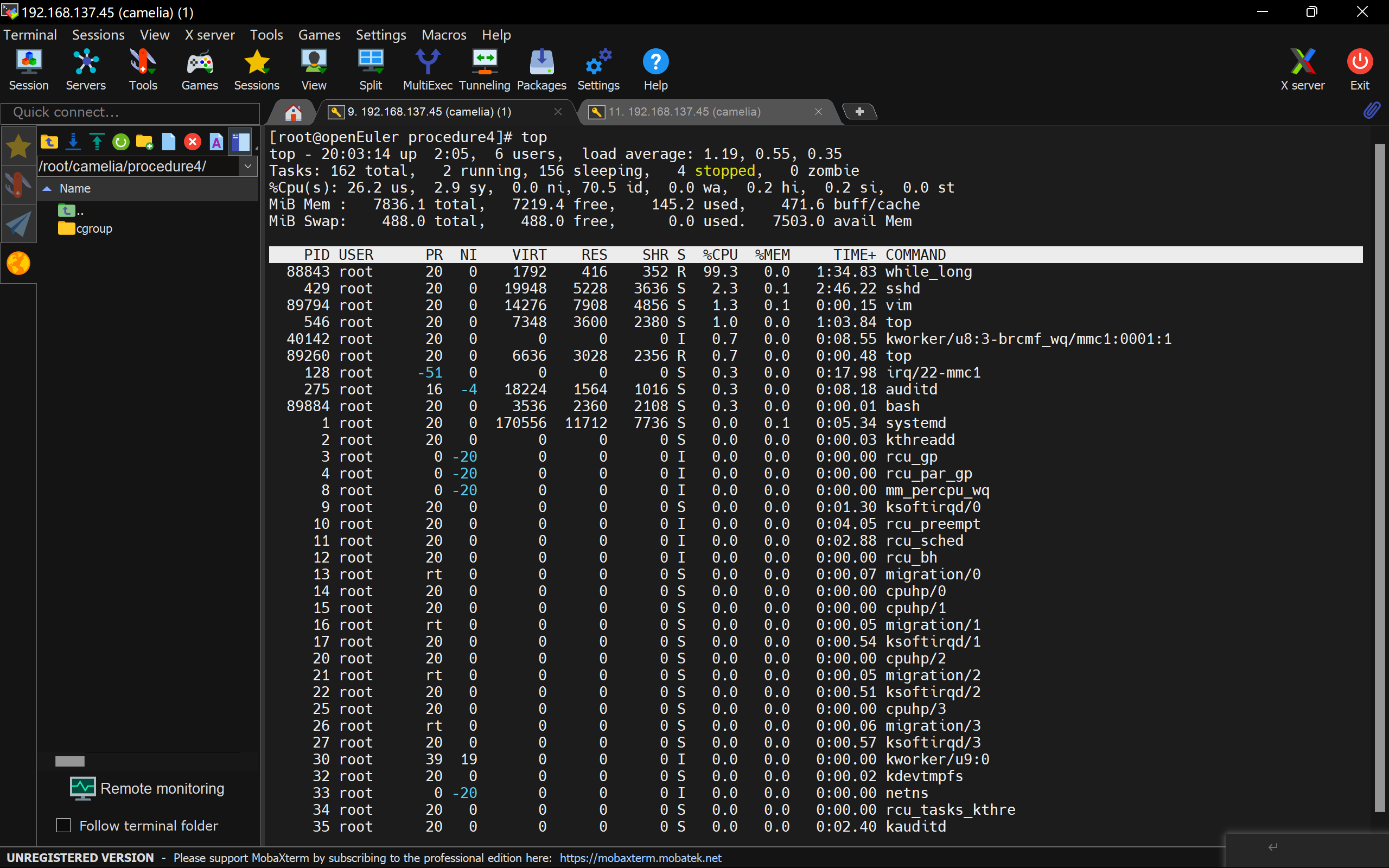
****

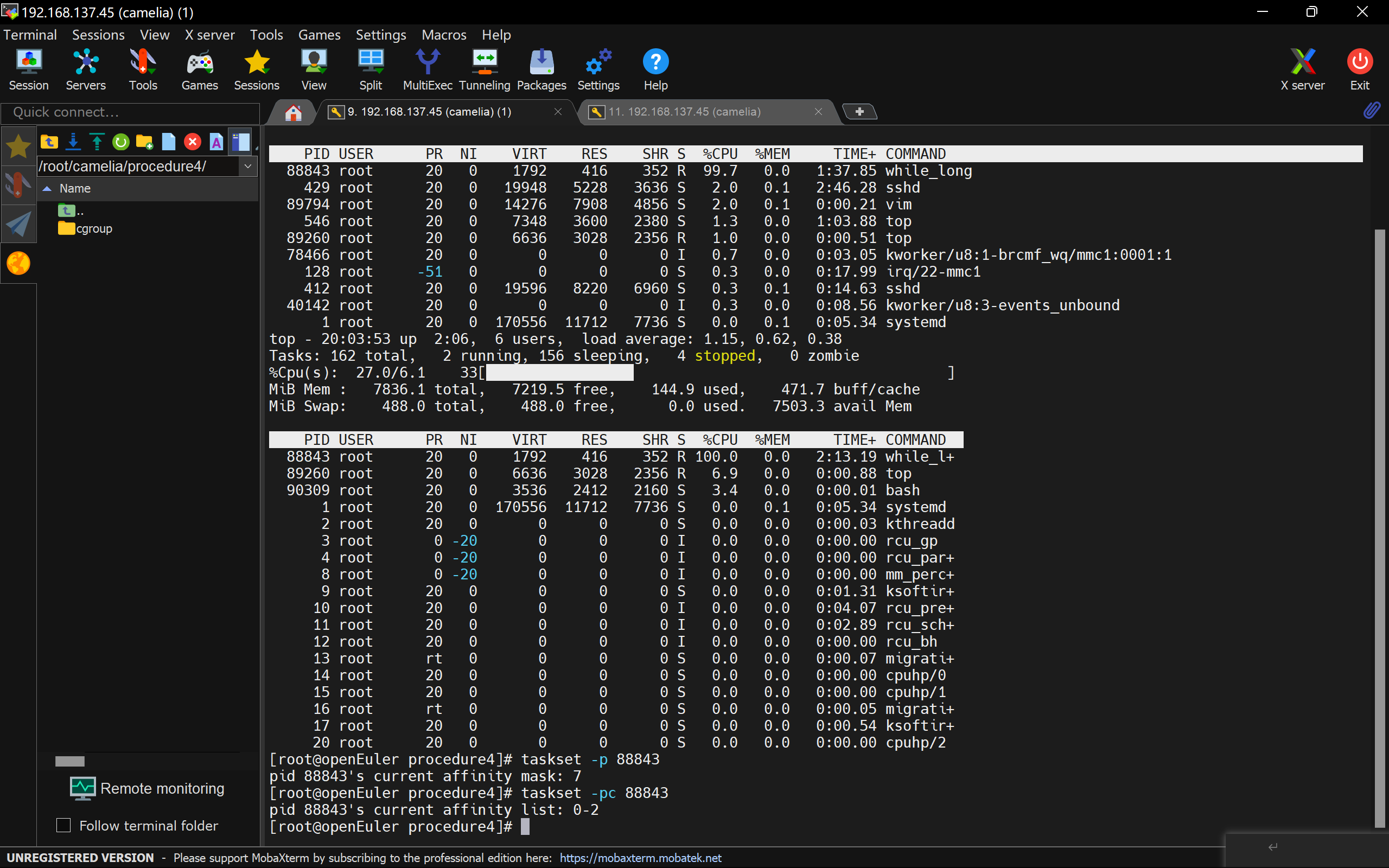
****

****

****

****



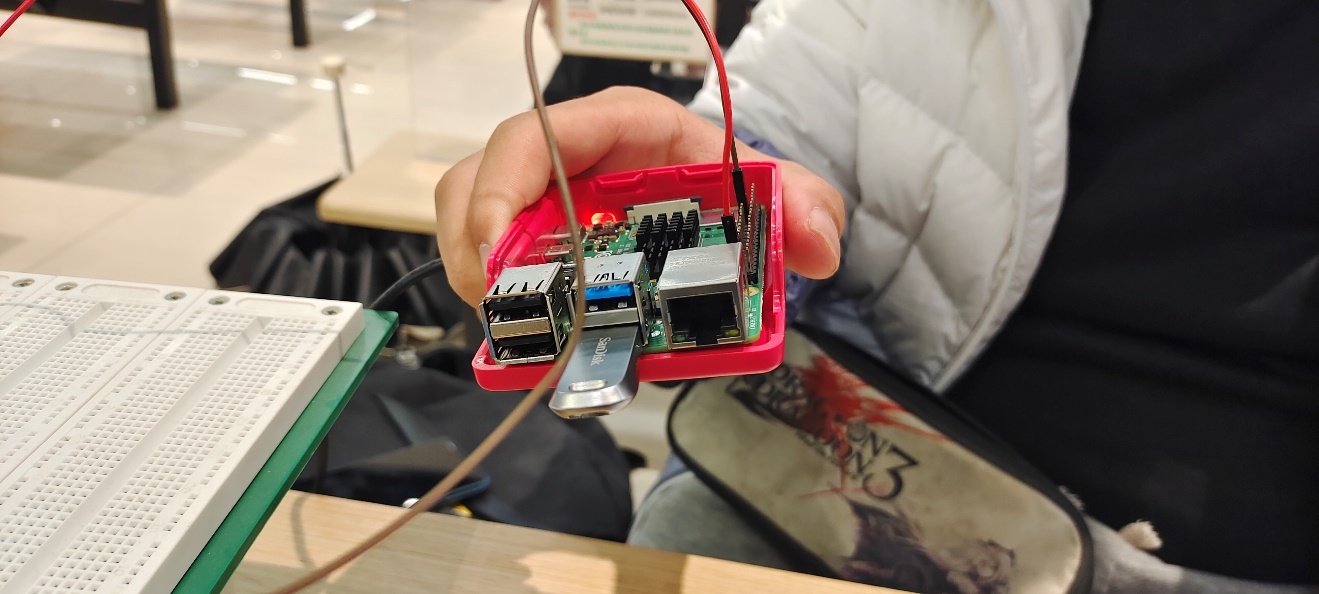
****

**任务5：使用 cgroup 实现不允许访问U盘**

无法完成

见下方详情

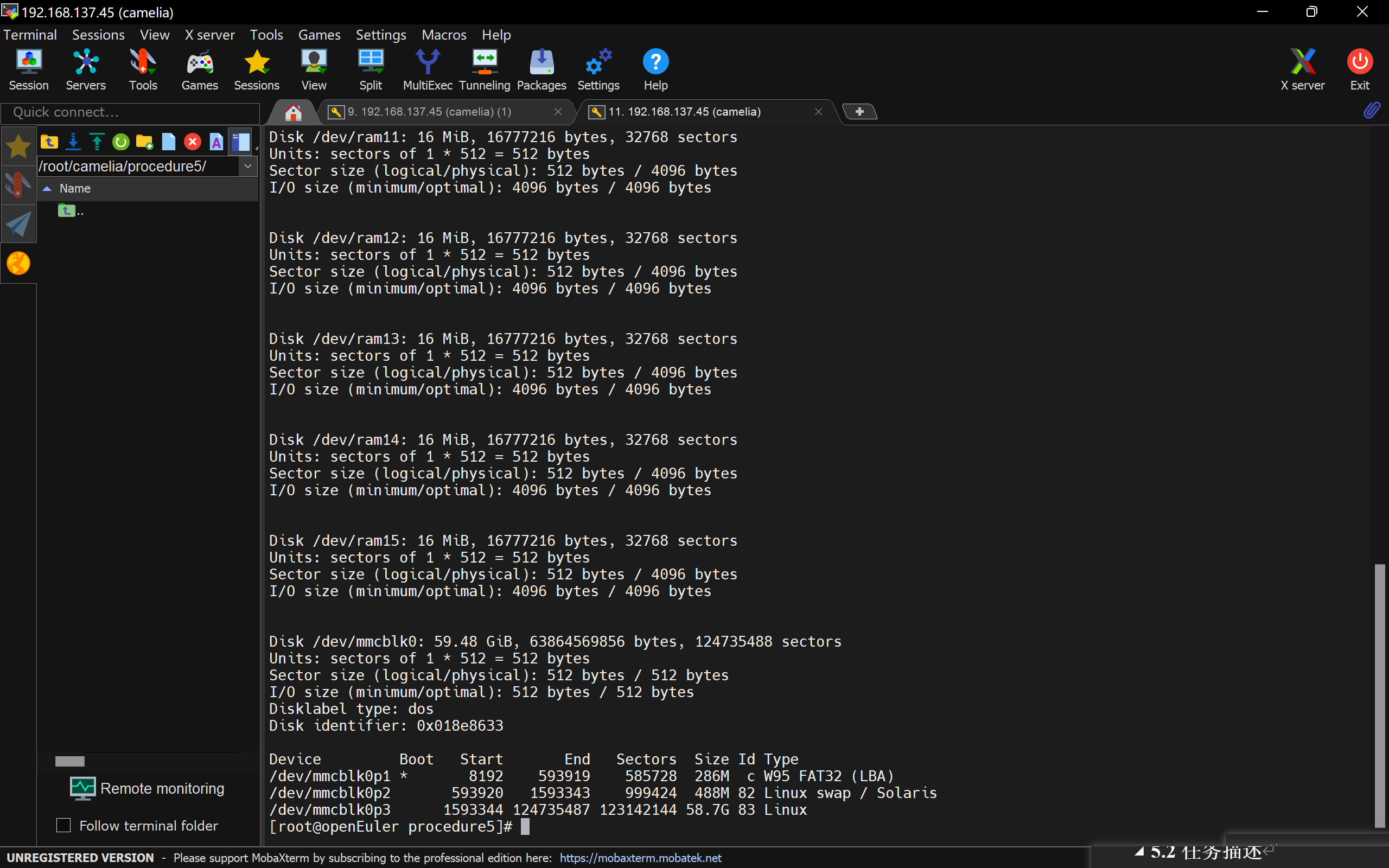
插入U盘

****

执行命令

****

得到分区信息，与插入U盘前并无二致

****

使用lsusb命令查看毫无作用

换U盘，接口亦无作用

据悉为系统对应版本（21.03）驱动所致

