信息与软件工程学院

语言类项目实践中期报告

课程名称：语言类项目实践

课题名称：Linux 系统计费

指导教师：文军

所在系别：嵌入式

执行学期：第四学期

学生信息：

序号	学号	姓名
1（组长）	2016220304002	许奕腾
2	2016220302034	刁瑞琪
3	2016220302028	王铁举
4	2016220301008	杨浩铨
5	2015220302008	徐智林
6	2016220304031	邓巧

目 录

说明:

1、报告要求 2000 字以上。

2、本模板仅为基本参考，请各位同学根据个人情况进行目录结构扩展。

3、报告正文必须双面打印。

第一章 综合设计的进展情况

1.1 需求分析与建模

在 Linux 系统下，监控显示每一个进程的 CPU 使用量、内存使用量、交换内存、缓存大小、缓冲区大小、流程 PID、用户、命令等信息，用程序实现针对上面指标的显示列表、计费，用于控制特别进程的使用，为云计算调度，提供支持。

整个 LINUX 系统：

- 物理内存总量
- 使用的物理内存总量
- 空闲内存总量
- 用作内核缓存的内存量
- 交换区总量
- 使用的交换区总量
- 空闲交换区总量
- 缓冲的交换区总量

每一个进程的 CPU 使用量：

- 上次更新到现在的 CPU 时间占用百分比
- 进程使用的 CPU 时间总计，单位秒

每一个进程的内存使用量：

- 进程使用的物理内存百分比
- 进程使用的、未被换出的物理内存大小，单位 kb。
- 进程使用的虚拟内存总量，单位 kb
- 可执行代码占用的物理内存大小，单位 kb
- 可执行代码以外的部分(数据段+栈)占用的物理内存大小，单位 kb

每一个进程的交换内存：

- 进程使用的虚拟内存中，被换出的大小，单位 kb

每个进程的流程 PID：

- 进程 ID

- 父进程 ID

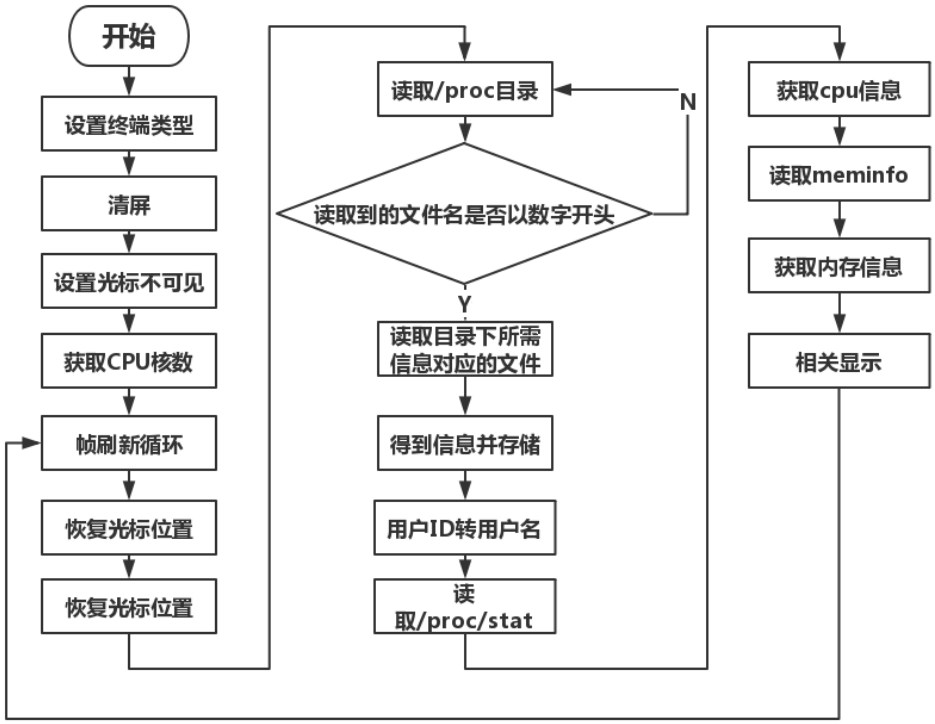
每个进程的用户：

- Real user name
- 进程所有者的用户 id
- 进程所有者的用户名
- 进程所有者的组名

每个进程的命令：

- 命令名/命令行

本项目需要用到的工具暂时不多，基础工具为 linux 系统（可用虚拟机实现）提供系统 api 与 gcc，编辑器（选用 vscode 与 vim）。



流程图

1.2 复杂问题归纳

需要解决的工程问题有：

1. 对于进程相关的数据的读出。
2. 对于数据的统计与计费方式与算法。

1.3 针对复杂工程问题的方案实现

Linux 为每一个进程维护一个 task_struct 结构，在结构中可以读取到进程状态，进程标识符，标记符号，优先级等。

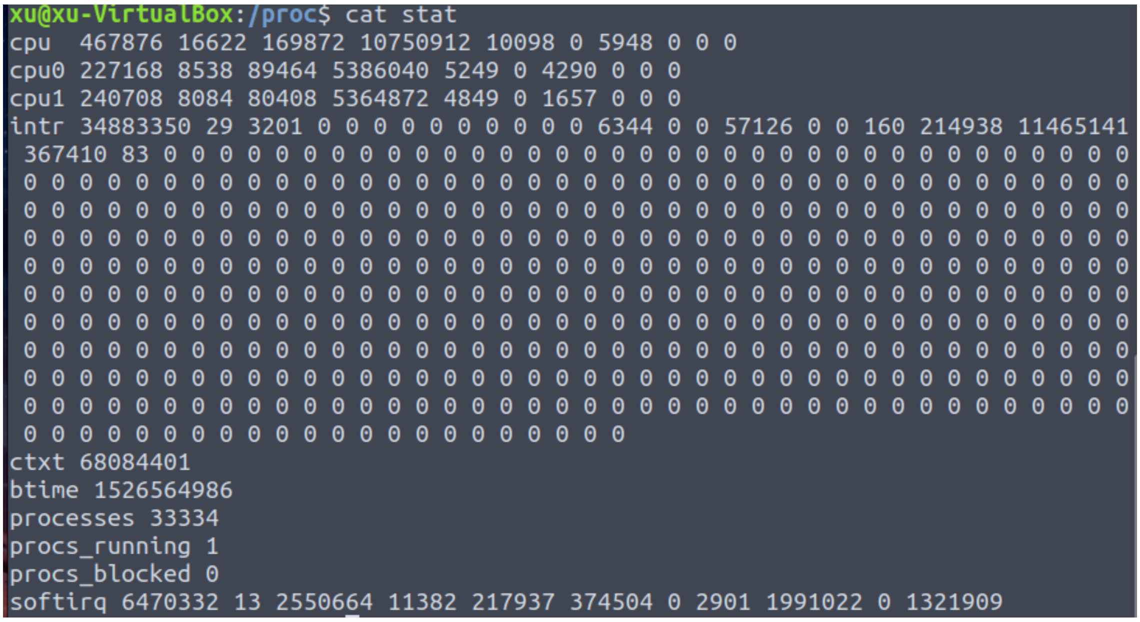
```
pid_t pid;      //进程的标识符
pid_t tgid;     //线程组标识符

int prio, static_prio, normal_prio;
unsigned int rt_priority;
```

但由于全部的进程信息无法由简单的通过进程控制块维护的双向链表遍历得来，所以考虑查询 linux 操作系统是否有一个具体的文件来维护总的进程数量。

Linux 系统维护了一个伪文件系统：/proc 文件系统，这个系统只存在内存中，不占用外存储空间，他以文件的方式为内核和进程提供通讯的接口。值得一提的是这个文件系统中的文件在读取的时候是动态变化的，也就是说这个系统会动态的从内核中取出我们需要的相关信息。

在/proc 下的 stat 中，维护了系统当前运行的全部进程的信息：



```
xu@xu-VirtualBox:/proc$ cat stat
cpu  467876 16622 169872 10750912 10098 0 5948 0 0 0
cpu0 227168 8538 89464 5386040 5249 0 4290 0 0 0
cpu1 240708 8084 80408 5364872 4849 0 1657 0 0 0
intr 34883350 29 3201 0 0 0 0 0 0 0 0 0 0 6344 0 0 57126 0 0 160 214938 11465141
 367410 83 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ctxt 68084401
btime 1526564986
processes 33334
procs_running 1
procs_blocked 0
softirq 6470332 13 2550664 11382 217937 374504 0 2901 1991022 0 1321909
```

图 1-1 stat 文件的内容

在这个内容中：第一行与 cpu 有关的数据分别是

1．正常的进程在用户态下执行的时间积累

2. NICED 的进程在用户态下执行的时间列

3. 进程在内核态的执行时间积累

4. 空闲时间积累

5. 等待 i/o 完成的时间积累

6. 硬中断时间

7. 软中断时间

再往下的内容中记录了如下的数据

“intr”： 这行给出自系统启动以来的所有中断信息。第一个数字记录所有的中断的次数；然后每个数对应一个特定的中断自系统启动以来所发生的次数。

“ctxt”： 给出了自系统启动以来 CPU 发生的上下文交换的次数。

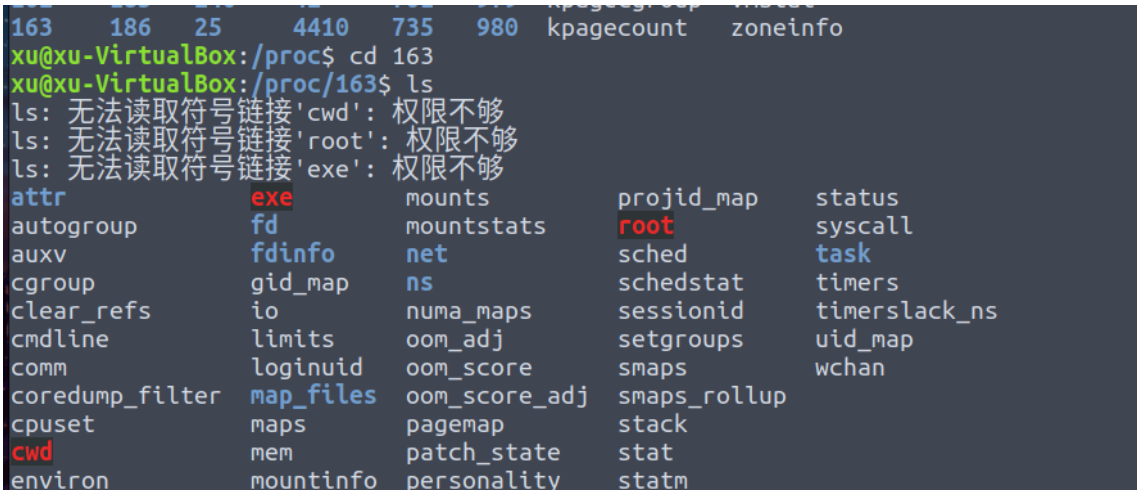
“btime”： 给出了从系统启动到现在为止的时间，单位为秒。

“processes ”： 自系统启动以来所创建的任务的个数目。

“procs_running”：当前运行队列的任务的数目。

“procs_blocked”：当前被阻塞的任务的数目，等待 I/O 完成次数。

在/proc 目录下，有一些一数字命名的目录，这些目录被称之为进程目录，同样在



这个进程目录下我们可以看到有很多的进程相关的信息。

图 1-2 进程目录里的信息

在进程文件夹里的 stat 文件中可以看到某一个进程里的更多信息

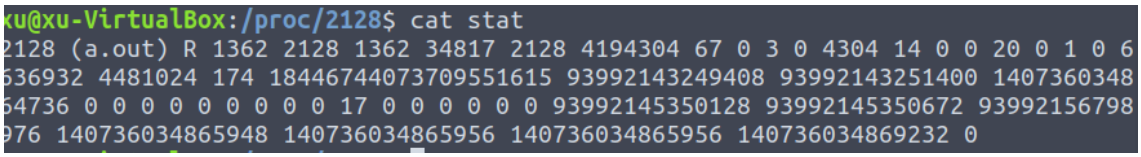


图 1-3 进程 stat 信息

比如刚刚测试运行了一个 a.out，后面一长串信息，查询后面这一大串的字符信息如下。

pid=2128 进程(包括轻量级进程，即线程)号

comm=a.out 应用程序或命令的名字

task_state=R 任务的状态，R:runnign, S:sleeping (TASK_INTERRUPTIBLE), D:disk sleep (TASK_UNINTERRUPTIBLE), T: stopped, T:tracing stop,Z:zombie, X:dead

ppid=1362 父进程 ID

pgid=2128 线程组号

sid=1362 该任务所在的会话组 ID

tty_nr=34817(pts/3) 该任务的 tty 终端的设备号，INT（34817/256）=主设备号，（34817-主设备号）=次设备号

tty_pgrp=2128 终端的进程组号，当前运行在该任务所在终端的前台任务(包括 shell 应用程序)的 PID。

task->flags=4194303 进程标志位，查看该任务的特性

min_flt=67 该任务不需要从硬盘拷数据而发生的缺页（次缺页）的次数

cmin_flt=0 累计的该任务的所有的 waited-for 进程曾经发生的次缺页的次数目

maj_flt=3 该任务需要从硬盘拷数据而发生的缺页（主缺页）的次数

cmaj_flt=0 累计的该任务的所有的 waited-for 进程曾经发生的主缺页的次数目

utime=4304 该任务在用户态运行的时间，单位为 jiffies

stime=14 该任务在核心态运行的时间，单位为 jiffies

cutime=0 累计的该任务的所有的 waited-for 进程曾经在用户态运行的时间，单位为 jiffies

cstime=0 累计的该任务的所有的 waited-for 进程曾经在核心态运行的时间，单位为 jiffies

priority=20 任务的动态优先级

nice=0 任务的静态优先级

num_threads=1 该任务所在的线程组里线程的个数

it_real_value=0 由于计时间隔导致的下一个 SIGALRM 发送进程的时延，以 jiffy 为单位.
start_time=6636932 该任务启动的时间，单位为 jiffies
vsize=4481024（page） 该任务的虚拟地址空间大小
rss=174 (page) 该任务当前驻留物理地址空间的大小
Number of pages the process has in real memory,minu 3 for administrative purpose.
这些页可能用于代码，数据和栈。
rlim=18446744073709551615（bytes） 该任务能驻留物理地址空间的最大值
start_code=93992143249408 该任务在虚拟地址空间的代码段的起始地址
end_code=93992143251400 该任务在虚拟地址空间的代码段的结束地址
start_stack=140736034864736 该任务在虚拟地址空间的栈的结束地址
kstkesp=0 esp(32 位堆栈指针) 的当前值, 与在进程的内核堆栈页得到的一致.
kstkeip=2097798 指向将要执行的指令的指针, EIP(32 位指令指针)的当前值.
pendingsig=0 待处理信号的位图，记录发送给进程的普通信号
block_sig=0 阻塞信号的位图
sigign=0 忽略的信号的位图
sigcatch=0 被俘获的信号的位图
wchan=0 如果该进程是睡眠状态，该值给出调度的调用点
nswap 被 swapped 的页数，当前没用
cnswap 所有子进程被 swapped 的页数的和，当前没用
exit_signal=17 该进程结束时，向父进程所发送的信号
task_cpu(task)=0 运行在哪个 CPU 上
task_rt_priority=0 实时进程的相对优先级别
task_policy=0 进程的调度策略，0=非实时进程，1=FIFO 实时进程；2=RR 实时进程

以上的信息我们重点提取出 utime=4304，stime=14，cutime=0，cstime=0 四条，计算进程总的 cpu 时间为 utime+stime+cutime+cstime。

总的 cpu 使用时间可以使用如下算法。
1、采样两个足够短的时间间隔的 Cpu 快照，分别记作 t1,t2，其中 t1、t2 的结构

均为：(user、nice、system、idle、iowait、irq、softirq、stealstolen、guest)的 9 元组;

2、计算总的 Cpu 时间片 totalCpuTime

a)把第一次的所有 cpu 使用情况求和，得到 s1;

b)把第二次的所有 cpu 使用情况求和，得到 s2;

c)s2 - s1 得到这个时间间隔内的所有时间片，即 totalCpuTime = j2 - j1 ;

3、计算空闲时间 idle

idle 对应第四列的数据，用第二次的 idle - 第一次的 idle 即可

idle=第二次的 idle - 第一次的 idle

4、计算 cpu 使用率

pcpu =100* (total-idle)/total

为了便于使用，我们为每个进程构造一个结构体，用于存储相关信息：

```
typedef struct proc_t {
int
tid,
ppid;
unsigned
pcpu;
char
                                                    state,
                                                    pad_1,
                                                    pad_2,
                                                    pad_3;

unsigned long long
utime,
stime,

cutime,
cstime,
start_time;
```

```
#ifdef SIGNAL_STRING
char
signal[18],
blocked[18],
sigignore[18],
sigcatch[18],
_sigpnd[18];
#else
long long
signal,
blocked,
sigignore,
sigcatch,
_sigpnd;
#endif
unsigned long long
start_code,
end_code,
start_stack,
kstk_esp,
kstk_eip,
wchan;
long
priority,
nice,
rss,
alarm,
size,
resident,
share,
trs,
lrs,
```

```
drs,
dt;
unsigned long
vm_size,
vm_lock,
vm_rss,
vm_data,
vm_stack,
vm_exe,
vm_lib,
rtprio,
sched,
vsize,
rss_rlim,
flags,
min_flt,
maj_flt,
cmin_flt,
cmaj_flt;
char
**environ,
**cmdline;
char
                                                    euser[20],
                                                    ruser[20],
                                                    suser[20],
                                                    fuser[20],
                                                    rgroup[20],
                                                    egroup[20]
                                                    sgroup[20],
                                                    fgroup[20],
                                                    cmd[16];
```

```
struct proc_t
*ring,
*next;
int
pgrp,
session,
nlwp,
tgid,
tty,
        euid, egid,
        ruid, rgid,
        suid, sgid,
        fuid, fgid,
tpgid,
exit_signal,
processor;
}
```

文件信息获取的实现:

这里对于文件的打开和读取，我们没有采用文件指针，而是使用了更为稳健的文件描述符。

利用 linux 提供的 open 和 read 函数，我们可以分别（打开文件）获得文件描述符和通过文件描述符读取文件内容。

这里以读取/proc/#/stat 为例

```
fd = open(filename, O_RDONLY, 0);
if(fd == -1) return -1;
num_read = read(fd, ret, cap - 1);
close(fd);
```

先将 filename 文件的内容读入到 ret 中。

然后利用 sscanf 将内容赋给对应的变量

```
num = sscanf(S,
    "%c "
```

```
    "%d %d %d %d %d "
    "%lu %lu %lu %lu %lu "
    "%Lu %Lu %Lu %Lu "
    "%ld %ld "
    "%d "
    "%ld "
    "%Lu "
    "%lu "
    "%ld "
    "%lu %"KLF"u %"KLF"u %"KLF"u %"KLF"u %"KLF"u "
    "%*s %*s %*s %*s " /* discard, no RT signals & Linux 2.1 used hex */
    "%"KLF"u %*lu %*lu "
    "%d %d "
    "%lu %lu",
    &P->state,
    &P->ppid, &P->pgrp, &P->session, &P->tty, &P->tpgid,
    &P->flags, &P->min_flt, &P->cmin_flt, &P->maj_flt, &P->cmaj_flt,
    &P->utime, &P->stime, &P->cutime, &P->cstime,
    &P->priority, &P->nice,
    &P->nlwp,
    &P->alarm,
    &P->start_time,
    &P->vsize,
    &P->rss,
    &P->rss_rlim, &P->start_code, &P->end_code, &P->start_stack, &P->kstk_esp,
&P->kstk_eip,
    &P->wchan,
    &P->exit_signal,
    &P->processor,
    &P->rtprio,
    &P->sched
);
```

第二章 存在问题与解决方案

2.1 存在的主要问题

（分析、总结和归纳综合设计过程中尚未解决的主要工程问题）

在输出结果到命令行的时候，显示的计费数据滚动，可视性与交互性都非常的差。这里希望能实现一个刷新屏幕的功能。

2.2 解决方案

（针对发现的问题，通过分析文献寻求可替代的解决方案）

无疑我们要处理的是字符终端，linux 提供给字符终端处理的有 ncurses 库。

```
sudo apt-get install libncurses5-dev
```

ncurses 库中，提供了函数”setupterm”用于设置终端类型，这里我们使用了

```
setupterm(NULL, STDOUT_FILENO, NULL);
```

当第一个参数为 NULL 时，使用环境变量 TERM 的值，终端类型用来查找相应的数据库以获得信息。

我们可以使用 string capabilities 来对终端进行控制，如

```
putp(clear_screen);
putp(cursor_invisible);
```

其中 clear_screen 用于清屏，cursor_ivisible 用于设置光标不可见，其余的 cap 可在 man 5 terminfo 中查到。

通过 ncurses 库提供的 tgoto 函数和 cursor_address 我们可以得到控制光标移动的转义序列。

```
putp(tgoto(cursor_address, 0, 3));
```

上述函数可以将光标移动到 3 行 0 列处。

于是通过光标移动转义序列、清空行剩余列、清空行等转义序列，我们便可以达到不滚动同时刷新终端的目的。

第三章 前期任务完成度与后续实施计划

前期任务完成进度：完成了进程总数目，各状态进程数目，cpu 占有量，内存总量、使用量、剩余量，交换区总量、使用量、剩余量，用于内核缓冲的内存总量和用于缓存的交换区总量，进程 CPU 使用量、内存使用量、内存使用率、交换内存、PID、父进程 ID、real user、所有者、real user grou、所有者 group 等的读取。完成了字符终端页面的绘制、刷新等。

```
Tasks: 102 total,   3 running,  99 sleeping,   0 stopped,   0 zombie
Cpu(s):  53.5% user,  46.5% system,   0.0% nice,   0.0% idle
Mem:   1030308k total,   753276k used,   277032k free,   371428k buffers
Swap:        0k total,        0k used,        0k free,   206660k cached
PID     pcpu    mem     pmem    swap    euser   time    esize   dssize  ppid    ruser   egroup  cmd
46      0       0       0.0000  0       root    0       0       0       2       root            deferwq
78      0       0       0.0000  0       root    0       0       0       2       root            kdmremove
79      0       0       0.0000  0       root    0       0       0       2       root            kstriped
216     0       0       0.0000  0       root    0       0       0       2       root            scsi_eh_0
217     0       0       0.0000  0       root    0       0       0       2       root            scsi_eh_1
265     0       0       0.0000  0       root    0       0       0       2       root            virtio-blk
321     3812    0       0.0000  0       root    3812    0       0       2       root            jbd2/vda1-8
322     0       0       0.0000  0       root    0       0       0       2       root            ext4-dio-unwrit
409     24      682     0.0007  2728    root    59      144     628     1       root            udevd
551     0       0       0.0000  0       root    0       0       0       2       root            virtio-net
584     0       0       0.0000  0       root    0       0       0       2       root            vballoon
756     87      0       0.0000  0       root    87      0       0       2       root            kauditd
759     416186  340     0.0003  1360    root    416186  1108    244     1       root            aliyun-service
984     934     3254    0.0032  13016   root    934     100     10516   1       root            auditd
1006    645     9344    0.0091  37376   root    645     376     34304   1       root            rsyslogd
1323    0       0       0.0000  0       root    0       0       0       2       root            cqueue
1658    2366    1424    0.0014  5696    root    23088   56      1384    1       root            crond
1679    63      662     0.0006  2648    root    63      20      408     1       root            atd
1726    13      796     0.0008  3184    root    19      24      480     1       root            login
1729    0       502     0.0005  2008    root    0       12      244     1       root            mingetty
1731    0       502     0.0005  2008    root    0       12      244     1       root            mingetty
1733    0       502     0.0005  2008    root    0       12      244     1       root            mingetty
1735    0       502     0.0005  2008    root    0       12      244     1       root            mingetty
1737    0       502     0.0005  2008    root    0       12      244     1       root            mingetty
1798    1364    3766    0.0037  15064   nobody  1364    852     8040    1       nobody          nginx
2424    354925  1938    0.0019  7752    root    9966669 332     4892    1       root            tmux
2425    15      1315    0.0013  5260    root    112475467       836     404     2424    root            bash
3456    31      1339    0.0013  5356    root    34094   836     500     2424    root            bash
13900   33      1315    0.0013  5260    root    1302    836     404     1726    root            bash
15442   0       1281    0.0012  5124    root    0       836     268     1       root            mysqld_safe
15550   74616   34126   0.0331  136504  mysql   74616   6788    121876  15442   mysql           mysqld
15604   8390076 89189   0.0866  356756  root    8390076 4       329436  2425    root            java
16659   0       3731    0.0036  14924   root    0       852     7900    1       root            nginx
16661   1402    3765    0.0037  15060   nobody  1402    852     8036    16659   nobody          nginx
```

参考文献

[1] Robert Love. Linux Kernel Development[M].

[2] Amold Robbins. Essential Linux Device Drivers[M].