

**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

第一章 综合设计的进展情况 1

1.1 需求分析与建模 1

1.2 复杂工程问题归纳 1

1.3 实施方案与可行性研究 1

第二章 存在问题与解决方案 2

2.1 存在的主要问题 2

2.2 解决方案 2

第三章 前期任务完成度与后续实施计划 3

参考文献 4

**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

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

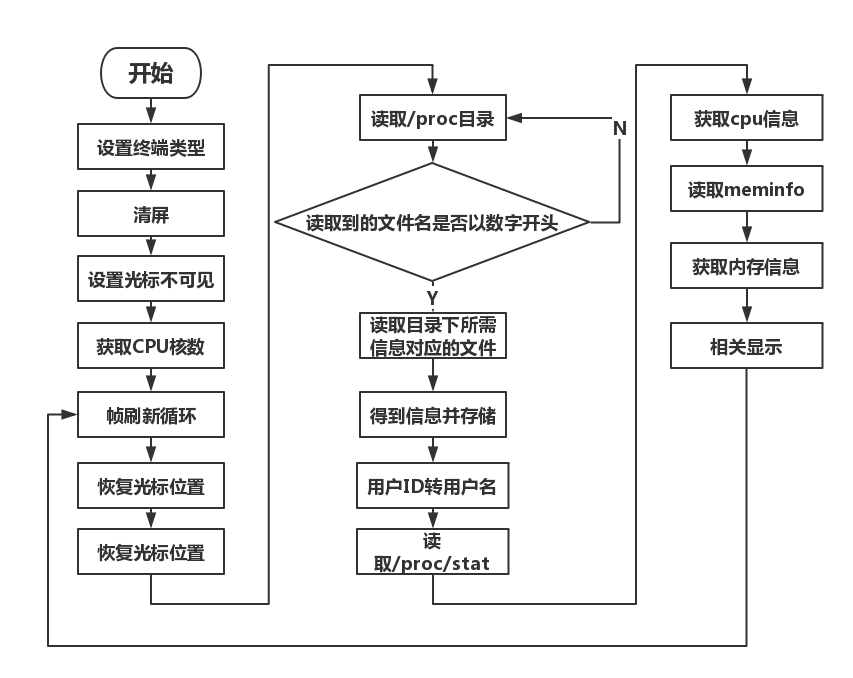
·进程所有者的用户id

·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

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。

针对复杂工程问题的方案实现

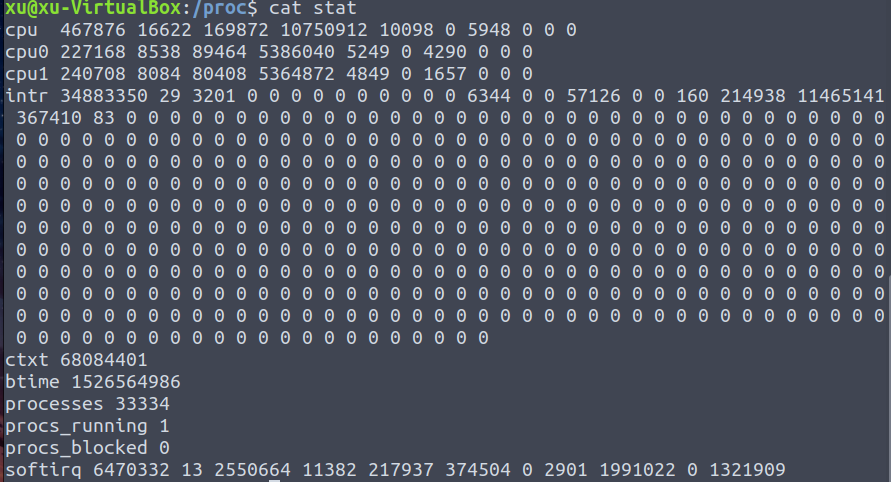
Linux为每一个进程维护一个task\_struct结构，在结构中可以读取到进程状态，进程标识符，标记符号，优先级等。

|  |
| --- |
| 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：

 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。

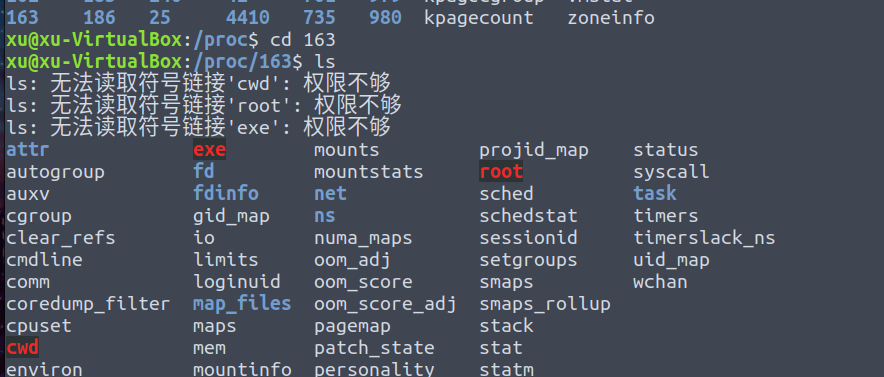
在/proc目录下，有一些一数字命名的目录，这些目录被称之为进程目录，同样在这个进程目录下我们可以看到有很多的进程相关的信息。

图1-2 进程目录里的信息

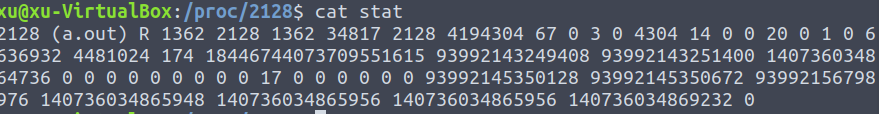
在进程文件夹里的stat文件中可以看到某一个进程里的更多信息

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。

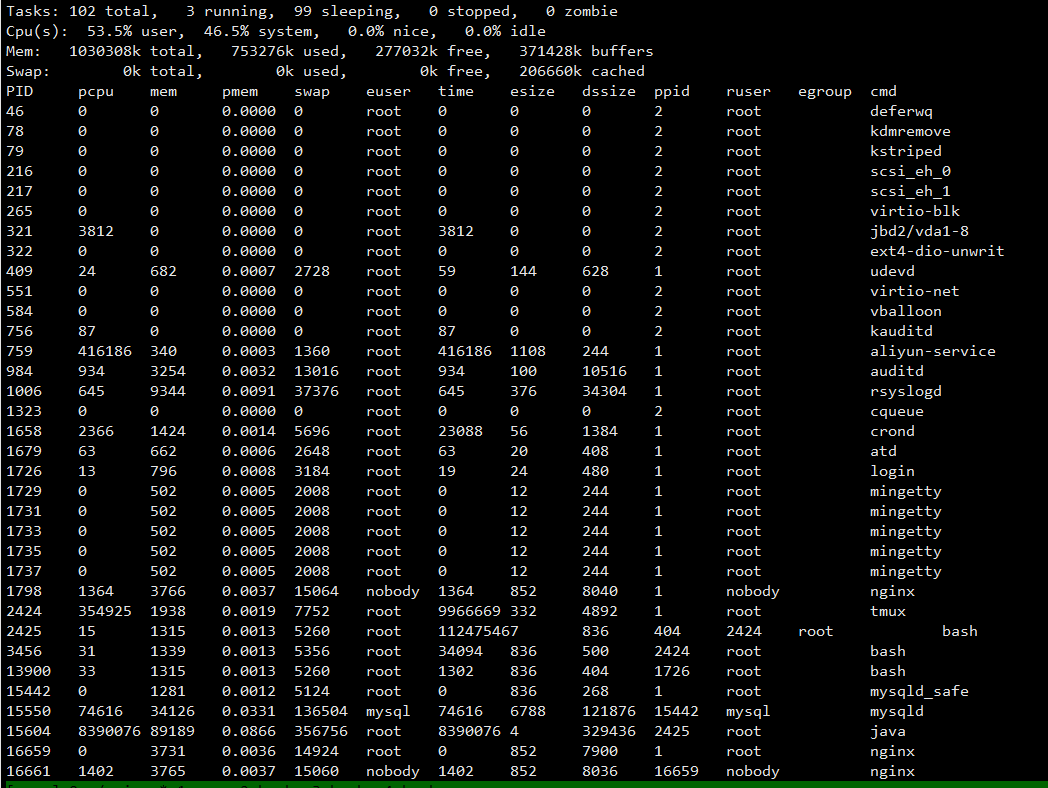
通过ncurses库提供的tgoto函数和cursor\_address我们可以得到控制光标移动的转义序列。

|  |
| --- |
| 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。

# 参考文献

1. Robert Love. Linux Kernel Development[M].
2. Amold Robbins. Essential Linux Device Drivers[M].