<https://www.cnblogs.com/cxjchen/archive/2013/03/30/2990548.html>

# [Linux下计算进程的CPU占用和内存占用的编程方法](http://www.cnblogs.com/cxjchen/archive/2013/03/30/2990548.html)

Linux下没有直接可以调用系统函数知道CPU占用和内存占用。那么如何知道CPU和内存信息呢。只有通过proc伪文件系统来实现。

proc伪文件就不介绍了，只说其中4个文件。一个是/proc/stat,/proc/meminfo,/proc/<pid>/status,/proc/<pid>/stat

摘自：<http://www.blogjava.net/fjzag/articles/317773.html>

/proc/stat:存放系统的CPU时间信息

该文件包含了所有CPU活动的信息，该文件中的所有值都是从系统启动开始累计到当前时刻。不同内核版本中该文件的格式可能不大一致，以下通过实例来说明数据该文件中各字段的含义。

实例数据：2.6.24-24版本上的

fjzag@fjzag-desktop:~$ cat /proc/stat

cpu 38082 627 27594 893908 12256 581 895 0 0

cpu0 22880 472 16855 430287 10617 576 661 0 0

cpu1 15202 154 10739 463620 1639 4 234 0 0

intr 120053 222 2686 0 1 1 0 5 0 3 0 0 0 47302 0 0 34194 29775 0 5019 845 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 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 1434984

btime 1252028243

processes 8113

procs\_running 1

procs\_blocked 0

第一行的数值表示的是CPU总的使用情况，所以我们只要用第一行的数字计算就可以了。下表解析第一行各数值的含义：

**参数 解析（单位：jiffies）**

(jiffies是内核中的一个全局变量，用来记录自系统启动一来产生的节拍数，在linux中，一个节拍大致可理解为操作系统进程调度的最小时间片，不同linux内核可能值有不同，通常在1ms到10ms之间)

**user** (38082) 从系统启动开始累计到当前时刻，处于用户态的运行时间，不包含 nice值为负进程。

**nice** (627) 从系统启动开始累计到当前时刻，nice值为负的进程所占用的CPU时间

**system** (27594) 从系统启动开始累计到当前时刻，处于核心态的运行时间

**idle** (893908) 从系统启动开始累计到当前时刻，除IO等待时间以外的其它等待时间**iowait** (12256) 从系统启动开始累计到当前时刻，IO等待时间(since 2.5.41)

**irq** (581) 从系统启动开始累计到当前时刻，硬中断时间(since 2.6.0-test4)

**softirq** (895) 从系统启动开始累计到当前时刻，软中断时间(since 2.6.0-test4)**stealstolen**(0) which is the time spent in other operating systems when running in a virtualized environment(since 2.6.11)

**guest**(0) which is the time spent running a virtual CPU for guest operating systems under the control of the Linux kernel(since 2.6.24)

**结论2**：总的cpu时间totalCpuTime = user + nice + system + idle + iowait + irq + softirq + stealstolen + guest

可以利用scanf,sscanf,fscanf读取这些信息，具体可以查man proc.我的程序中只取了前4个。

/proc/meminfo：存放系统的内存信息

[ubuntu@root ~]#cat /proc/meminfo   
MemTotal:        2061616 kB   
MemFree:         1093608 kB   
Buffers:          151140 kB   
Cached:           479372 kB   
SwapCached:            0 kB   
Active:           516964 kB   
Inactive:         374672 kB   
Active(anon):     261412 kB   
Inactive(anon):     5604 kB   
Active(file):     255552 kB   
Inactive(file):   369068 kB

……

别的就不说了，主要看第一个MemTotal,系统总的物理内存，它比真实的物理内存要小一点

/proc/<pid>/status：存放进程的CPU时间信息以及一些综合信息

[ubuntu@root ~]#cat /proc/889/status   
Name:    Xorg   
State:    S (sleeping)   
Tgid:    889   
Pid:    889   
PPid:    881   
TracerPid:    0   
Uid:    0    0    0    0   
Gid:    0    0    0    0   
FDSize:    256   
Groups:      
VmPeak:       99036 kB   
VmSize:       52424 kB   
VmLck:           0 kB   
VmHWM:       57004 kB   
VmRSS:       45508 kB   
VmData:       35668 kB   
VmStk:         136 kB   
VmExe:        1660 kB   
VmLib:        6848 kB   
VmPTE:         120 kB   
VmPeak是占用虚拟内存的峰值，也就是最高的一个值，而且是虚拟内存，所以有时候会比物理内存要大。PS和TOP指令都是利用VmPeak计算内存占用的。

VmRSS是进程所占用的实际物理内存。

/proc/<pid>/stat：保存着进程的CPU信息。

[ubuntu@root ~]#cat /proc/889/stat   
889 (Xorg) S 881 889 889 1031 889 4202752 5307477 0 0 0 34943 12605 0 0 20 0 1 0 8146 89399296 11377 4294967295 134512640 136211844 3221201472 3221200460 5456930 0 0 3149824 1367369423 3223423286 0 0 17 0 0 0 0 0 0

pid=889 进程号

utime=34943 该任务在用户态运行的时间，单位为jiffies

stime=12605 该任务在核心态运行的时间，单位为jiffies

cutime=0 所有已死线程在用户态运行的时间，单位为jiffies

cstime=0 所有已死在核心态运行的时间，单位为jiffies

可以利用scanf,sscanf,fscanf读取这些信息，具体可以查man proc.

结论3：进程的总Cpu时间processCpuTime = utime + stime + cutime + cstime，该值包括其所有线程的cpu时间。

以上这些数据都可以通过文件读取的方式，可以按照一行一行的读取，然后采用scanf,sscanf,fscanf获取信息。

占用内存的计算方法：

pmem = VmRSS / MemTotal \* 100;

计算CPU占用的方法：

取一次processCpuTime1和totalCpuTime1;

间隔一段时间;

再取一次processCpuTime2和totalCpuTime2;

pcpu = 100 \* (processCpuTime2 – processCpuTime1)/(totalCpuTime2 - totalCpuTime1);

代码

[复制代码](javascript:void(0);)

1 get\_cpu.h

2

3 #ifdef \_\_cplusplus

4 extern "C"{

5 #endif

6

7 #define VMRSS\_LINE 15//VMRSS所在行

8 #define PROCESS\_ITEM 14//进程CPU时间开始的项数

9

10 typedef struct //声明一个occupy的结构体

11 {

12 unsigned int user; //从系统启动开始累计到当前时刻，处于用户态的运行时间，不包含 nice值为负进程。

13 unsigned int nice; //从系统启动开始累计到当前时刻，nice值为负的进程所占用的CPU时间

14 unsigned int system;//从系统启动开始累计到当前时刻，处于核心态的运行时间

15 unsigned int idle; //从系统启动开始累计到当前时刻，除IO等待时间以外的其它等待时间iowait (12256) 从系统启动开始累计到当前时刻，IO等待时间(since 2.5.41)

16 }total\_cpu\_occupy\_t;

17

18 typedef struct

19 {

20 pid\_t pid;//pid号

21 unsigned int utime; //该任务在用户态运行的时间，单位为jiffies

22 unsigned int stime; //该任务在核心态运行的时间，单位为jiffies

23 unsigned int cutime;//所有已死线程在用户态运行的时间，单位为jiffies

24 unsigned int cstime; //所有已死在核心态运行的时间，单位为jiffies

25 }process\_cpu\_occupy\_t;

26

27 int get\_phy\_mem(const pid\_t p);//获取占用物理内存

28 int get\_total\_mem();//获取系统总内存

29 unsigned int get\_cpu\_total\_occupy();//获取总的CPU时间

30 unsigned int get\_cpu\_process\_occupy(const pid\_t p);//获取进程的CPU时间

31 const char\* get\_items(const char\* buffer,int ie);//取得缓冲区指定项的起始地址

32

33 extern float get\_pcpu(pid\_t p);//获取进程CPU占用

34 extern float get\_pmem(pid\_t p);//获取进程内存占用

35 extern int get\_rmem(pid\_t p);//获取真实物理内存

36

37

38 #ifdef \_\_cplusplus

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

1 get\_cpu.c

2

3 #include <stdio.h>

4 #include <stdlib.h>

5 #include <unistd.h> //头文件

6 #include <assert.h>

7 #include "get\_cpu.h"

8

9 int get\_phy\_mem(const pid\_t p)

10 {

11 char file[64] = {0};//文件名

12

13 FILE \*fd; //定义文件指针fd

14 char line\_buff[256] = {0}; //读取行的缓冲区

15 sprintf(file,"/proc/%d/status",p);//文件中第11行包含着

16

17 fprintf (stderr, "current pid:%d\n", p);

18 fd = fopen (file, "r"); //以R读的方式打开文件再赋给指针fd

19

20 //获取vmrss:实际物理内存占用

21 int i;

22 char name[32];//存放项目名称

23 int vmrss;//存放内存峰值大小

24 for (i=0;i<VMRSS\_LINE-1;i++)

25 {

26 fgets (line\_buff, sizeof(line\_buff), fd);

27 }//读到第15行

28 fgets (line\_buff, sizeof(line\_buff), fd);//读取VmRSS这一行的数据,VmRSS在第15行

29 sscanf (line\_buff, "%s %d", name,&vmrss);

30 fprintf (stderr, "====%s：%d====\n", name,vmrss);

31 fclose(fd); //关闭文件fd

32 return vmrss;

33 }

34

35 int get\_rmem(pid\_t p)

36 {

37 return get\_phy\_mem(p);

38 }

39

40

41 int get\_total\_mem()

42 {

43 char\* file = "/proc/meminfo";//文件名

44

45 FILE \*fd; //定义文件指针fd

46 char line\_buff[256] = {0}; //读取行的缓冲区

47 fd = fopen (file, "r"); //以R读的方式打开文件再赋给指针fd

48

49 //获取memtotal:总内存占用大小

50 int i;

51 char name[32];//存放项目名称

52 int memtotal;//存放内存峰值大小

53 fgets (line\_buff, sizeof(line\_buff), fd);//读取memtotal这一行的数据,memtotal在第1行

54 sscanf (line\_buff, "%s %d", name,&memtotal);

55 fprintf (stderr, "====%s：%d====\n", name,memtotal);

56 fclose(fd); //关闭文件fd

57 return memtotal;

58 }

59

60 float get\_pmem(pid\_t p)

61 {

62 int phy = get\_phy\_mem(p);

63 int total = get\_total\_mem();

64 float occupy = (phy\*1.0)/(total\*1.0);

65 fprintf(stderr,"====process mem occupy:%.6f\n====",occupy);

66 return occupy;

67 }

68

69 unsigned int get\_cpu\_process\_occupy(const pid\_t p)

70 {

71 char file[64] = {0};//文件名

72 process\_cpu\_occupy\_t t;

73

74 FILE \*fd; //定义文件指针fd

75 char line\_buff[1024] = {0}; //读取行的缓冲区

76 sprintf(file,"/proc/%d/stat",p);//文件中第11行包含着

77

78 fprintf (stderr, "current pid:%d\n", p);

79 fd = fopen (file, "r"); //以R读的方式打开文件再赋给指针fd

80 fgets (line\_buff, sizeof(line\_buff), fd); //从fd文件中读取长度为buff的字符串再存到起始地址为buff这个空间里

81

82 sscanf(line\_buff,"%u",&t.pid);//取得第一项

83 char\* q = get\_items(line\_buff,PROCESS\_ITEM);//取得从第14项开始的起始指针

84 sscanf(q,"%u %u %u %u",&t.utime,&t.stime,&t.cutime,&t.cstime);//格式化第14,15,16,17项

85

86 fprintf (stderr, "====pid%u:%u %u %u %u====\n", t.pid, t.utime,t.stime,t.cutime,t.cstime);

87 fclose(fd); //关闭文件fd

88 return (t.utime + t.stime + t.cutime + t.cstime);

89 }

90

91

92 unsigned int get\_cpu\_total\_occupy()

93 {

94 FILE \*fd; //定义文件指针fd

95 char buff[1024] = {0}; //定义局部变量buff数组为char类型大小为1024

96 total\_cpu\_occupy\_t t;

97

98 fd = fopen ("/proc/stat", "r"); //以R读的方式打开stat文件再赋给指针fd

99 fgets (buff, sizeof(buff), fd); //从fd文件中读取长度为buff的字符串再存到起始地址为buff这个空间里

100 /\*下面是将buff的字符串根据参数format后转换为数据的结果存入相应的结构体参数 \*/

101 char name[16];//暂时用来存放字符串

102 sscanf (buff, "%s %u %u %u %u", name, &t.user, &t.nice,&t.system, &t.idle);

103

104

105 fprintf (stderr, "====%s:%u %u %u %u====\n", name, t.user, t.nice,t.system, t.idle);

106 fclose(fd); //关闭文件fd

107 return (t.user + t.nice + t.system + t.idle);

108 }

109

110

111 float get\_pcpu(pid\_t p)

112 {

113 unsigned int totalcputime1,totalcputime2;

114 unsigned int procputime1,procputime2;

115 totalcputime1 = get\_cpu\_total\_occupy();

116 procputime1 = get\_cpu\_process\_occupy(p);

117 usleep(500000);//延迟500毫秒

118 totalcputime2 = get\_cpu\_total\_occupy();

119 procputime2 = get\_cpu\_process\_occupy(p);

120 float pcpu = 100.0\*(procputime2 - procputime1)/(totalcputime2 - totalcputime1);

121 fprintf(stderr,"====pcpu:%.6f\n====",pcpu);

122 return pcpu;

123 }

124

125 const char\* get\_items(const char\* buffer,int ie)

126 {

127 assert(buffer);

128 char\* p = buffer;//指向缓冲区

129 int len = strlen(buffer);

130 int count = 0;//统计空格数

131 if (1 == ie || ie < 1)

132 {

133 return p;

134 }

135 int i;

136

137 for (i=0; i<len; i++)

138 {

139 if (' ' == \*p)

140 {

141 count++;

142 if (count == ie-1)

143 {

144 p++;

145 break;

146 }

147 }

148 p++;

149 }

150

151 return p;

152 }

[复制代码](javascript:void(0);)

标签: [linux](http://www.cnblogs.com/cxjchen/tag/linux/), [proc](http://www.cnblogs.com/cxjchen/tag/proc/), [CPU](http://www.cnblogs.com/cxjchen/tag/CPU/), [内存](http://www.cnblogs.com/cxjchen/tag/%E5%86%85%E5%AD%98/)