<https://www.cnblogs.com/bozhicheng/p/6216667.html>

在实际工作中有时需要程序打印出某个进程的内存占用情况以作参考, 下面介绍一种通过Linux下的伪文件系统/proc 计算某进程内存占用的程序实现方法.

首先, 为什么会有所谓的 **伪文件** 呢. Linux系统的文件类型大致可分为三类: 普通文件, 目录文件和伪文件. 伪文件不是用来存储数据的, 因此这些文件不占用磁盘空间, 只是存在于内存中. /proc 让你可以与内核内部数据进行交互, 获取有关进程的有用信息.

下面主要介绍一下 /proc 下面的四个文件: /proc/stat, /proc/meminfo, /proc/<pid>/stat, /proc/<pid>/status.

/proc/stat 存放系统的cpu时间, 该文件包含了所有cpu活动的信息.

cpu 72389 2891 16811 1148664 31374 0 67 0 0 0

cpu0 17608 452 3786 288899 6210 0 30 0 0 0

cpu1 18724 926 4598 285844 8911 0 15 0 0 0

cpu2 16803 658 3726 288710 7220 0 7 0 0 0

cpu3 19254 855 4700 285209 9032 0 13 0 0 0

...

...

...

/proc/meminfo 存放系统的内存信息, 通过文件中各个变量的名字便可知其代表的信息.

MemTotal: 4046236 kB

MemFree: 1054440 kB

MemAvailable: 2460060 kB

Buffers: 359688 kB

Cached: 1158056 kB

SwapCached: 0 kB

Active: 2020096 kB

Inactive: 677948 kB

Active(anon): 1181376 kB

...

...

...

/proc/<pid>/stat 存放某个进程的cpu信息

2476 (firefox) S 1773 1910 1910 0 -1 4210688 3413511 1712 757 1 45466 4629 2 7 20 0 57 0 20381 1774743552 150565 18446744073709551615 94844693012480 94844693126372 140732961864784 140732961858304 139747170914269 0 0 4096 33572079 0 0 0 17 2 0 0 1178 0 0 94844695226592 94844695228536 94844713955328 140732961867643 140732961867668 140732961867668 140732961869791 0

/proc/<pid>/status 存放某个进程的cpu信息以及一些综合信息

Name: firefox

State: S (sleeping)

Tgid: 2476

Ngid: 0

Pid: 2476

PPid: 1773

TracerPid: 0

Uid: 1000 1000 1000 1000

Gid: 1000 1000 1000 1000

FDSize: 256

Groups: 4 24 27 30 46 108 124 1000

NStgid: 2476

NSpid: 2476

NSpgid: 1910

NSsid: 1910

VmPeak: 1722812 kB

VmSize: 1690920 kB

VmLck: 0 kB

VmPin: 0 kB

VmHWM: 684048 kB

VmRSS: 600324 kB

VmData: 993040 kB

VmStk: 192 kB

...

...

...

以上数据都可以通过文件读取的方式来获取. 根据自己实验的需要可以计算相应的数据, 比如 pmem = VmRSS/MemTotal\*100 等等.

下面只是贴出一个简单的获取某进程当前时刻所占用的实际内存的c代码实现例子.

//get\_mem.h

#include <stdlib.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <assert.h>

#define VMRSS\_LINE 21//VMRSS所在行, 注:根据不同的系统,位置可能有所区别.

#define pid\_t int

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

{

char file[64] = {0};//文件名

FILE \*fd; //定义文件指针fd

char line\_buff[256] = {0}; //读取行的缓冲区

sprintf(file,"/proc/%d/status",p);

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

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

//获取vmrss:实际物理内存占用

int i;

char name[32];//存放项目名称

int vmrss;//存放内存

//读取VmRSS这一行的数据

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

{

char\* ret = fgets (line\_buff, sizeof(line\_buff), fd);

}

char\* ret1 = fgets (line\_buff, sizeof(line\_buff), fd);

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

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

fclose(fd); //关闭文件fd

return vmrss;

}

int get\_rmem(pid\_t p)

{

return get\_phy\_mem(p);

}

int get\_total\_mem()

{

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

FILE \*fd; //定义文件指针fd

char line\_buff[256] = {0}; //读取行的缓冲区

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

//获取memtotal:总内存占用大小

int i;

char name[32];//存放项目名称

int memtotal;//存放内存峰值大小

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

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

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

fclose(fd); //关闭文件fd

return memtotal;

}

测试文件:

#include "get\_mem.h"

int main()

{

int list[1024];

for(int i = 0; i < 1024; i++)

list[i] = i;

int mem = get\_rmem(getpid());

}

分类: [Linux基础](http://www.cnblogs.com/bozhicheng/category/885664.html)