<http://telyy123.iteye.com/blog/1176850>

程序实现思想：   
Linux 系统的进程号是保存在 /proc 目录下的，一个在 linux 系统上运行得程序，只要是在运行状态下，就必然会再 /proc 目录下写入一批文件，其文件写入的方式如下：   
  
/proc/<进程ID>/文件组   
  
这里面的文件组是指一批文件，不是指一个文件，大致的文件名信息如下：   
  
    dr-xr-xr-x 2 work work 0 09-21 00:51 attr   
    -r-------- 1 work work 0 09-21 00:51 auxv   
    -r--r--r-- 1 work work 0 09-21 00:51 cmdline   
    -rw-r--r-- 1 work work 0 09-21 00:51 coredump\_filter   
    -r--r--r-- 1 work work 0 09-21 00:51 cpuset   
    lrwxrwxrwx 1 work work 0 09-21 00:51 cwd -> /home/work/tmp   
    -r-------- 1 work work 0 09-21 00:51 environ   
    lrwxrwxrwx 1 work work 0 09-21 00:51 exe -> /home/work/tmp/tt   
    dr-x------ 2 work work 0 09-21 00:51 fd   
    dr-x------ 2 work work 0 09-21 00:51 fdinfo   
    -r-------- 1 work work 0 09-21 00:51 io   
    -r--r--r-- 1 work work 0 09-21 00:51 limits   
    -rw-r--r-- 1 work work 0 09-21 00:51 loginuid   
    -r--r--r-- 1 work work 0 09-21 00:51 maps   
    -rw------- 1 work work 0 09-21 00:51 mem   
    -r--r--r-- 1 work work 0 09-21 00:51 mounts   
    -r-------- 1 work work 0 09-21 00:51 mountstats   
    -r--r--r-- 1 work work 0 09-21 00:51 numa\_maps   
    -rw-r--r-- 1 work work 0 09-21 00:51 oom\_adj   
    -r--r--r-- 1 work work 0 09-21 00:51 oom\_score   
    lrwxrwxrwx 1 work work 0 09-21 00:51 root -> /   
    -r--r--r-- 1 work work 0 09-21 00:51 schedstat   
    -r--r--r-- 1 work work 0 09-21 00:51 smaps   
    -r--r--r-- 1 work work 0 09-21 00:51 stat   
    -r--r--r-- 1 work work 0 09-21 00:51 statm   
    -r--r--r-- 1 work work 0 09-21 00:51 status   
    dr-xr-xr-x 3 work work 0 09-21 00:51 task   
    -r--r--r-- 1 work work 0 09-21 00:51 wchan   
  
比如，我们运行了一个程序 tt ，假设其进程号为 31673 ，则在 /proc 目录下会形成 /proc/31673/ 的目录，且在其目录下存在上述的文件组。这个里面，我们需要的文件为 status ，该文件保存的是 tt 当前运行得状态，我们来看看这个文件的内容：   
  
    Name: tt   
    State: S (sleeping)   
    SleepAVG: 98%   
    Tgid: 31673   
    Pid: 31673   
    PPid: 7977   
    TracerPid: 0   
    Uid: 500 500 500 500   
    Gid: 500 500 500 500   
    FDSize: 256   
    Groups: 500   
    VmPeak: 3828 kB   
    VmSize: 3664 kB   
    VmLck: 0 kB   
    VmHWM: 300 kB   
    VmRSS: 300 kB   
    VmData: 44 kB   
    VmStk: 88 kB   
    VmExe: 4 kB   
    VmLib: 1448 kB   
    VmPTE: 32 kB   
    StaBrk: 09e04000 kB   
    Brk: 09e04000 kB   
    StaStk: 7ffff0e895d0 kB   
    Threads: 1   
    SigQ: 0/8192   
    SigPnd: 0000000000000000   
    ShdPnd: 0000000000000000   
    SigBlk: 0000000000000000   
    SigIgn: 0000000000000000   
    SigCgt: 0000000000000000   
    CapInh: 0000000000000000   
    CapPrm: 0000000000000000   
    CapEff: 0000000000000000   
    Cpus\_allowed: 00000000,00000000,00000000,00000000,00000000,00000000,00000000,ffffffff   
    Mems\_allowed: 00000000,00000001   
  
我们发现，这个文件的第一行就是文件名，有了这个东西，那我们就有了获取当前正在运行得特定程序的进程号了。   
基本的解决思路是，我们通过遍历 /proc 目录下所有的用数字作为目录的子目录，依次打开这些目录下的 status 文件，获取其中的第一行内容，并从中获取对应的 Name 节的信息，与我们传入的需要查询的程序名字进行比较，如果一致，则该目录的数字就是该程序的进程号。   
  
    /\* find\_pid\_by\_name()   
     \*   
     \* This finds the pid of the specified process.   
     \* Currently, it's implemented by rummaging through   
     \* the proc filesystem.   
     \* Returns a list of all matching PIDs   
     \*/   
  
    #include <unistd.h>   
    #include <sys/types.h>   
    #include <sys/stat.h>   
    #include <unistd.h>   
    #include <dirent.h>   
    #include <string.h>   
    #include <stdlib.h>   
    #include <stdio.h>   
  
    #define READ\_BUF\_SIZE 1024   
  
  
    long\* find\_pid\_by\_name( char\* pidName)   
    {   
            DIR \*dir;   
            struct dirent \*next;   
            long\* pidList=NULL;   
            int i=0;   
  
  
            dir = opendir("/proc");   
            if (!dir)   
            {   
                    //perror\_msg\_and\_die("Cannot open /proc");   
                    fprintf(stderr, "Cannot open /proc\n");   
                    return pidList;   
            }   
  
            while ((next = readdir(dir)) != NULL)   
            {   
                    FILE \*status;   
                    char filename[READ\_BUF\_SIZE];   
                    char buffer[READ\_BUF\_SIZE];   
                    char name[READ\_BUF\_SIZE];   
  
                    /\* Must skip ".." since that is outside /proc \*/   
                    if (strcmp(next->d\_name, "..") == 0)   
                            continue;   
  
                    /\* If it isn't a number, we don't want it \*/   
                    if (!isdigit(\*next->d\_name))   
                            continue;   
                    sprintf(filename, "/proc/%s/status", next->d\_name);   
  
                    if (! (status = fopen(filename, "r")) )   
                    {   
                            continue;   
                    }   
  
                    //Read first line in /proc/?pid?/status   
  
                    if (fgets(buffer, READ\_BUF\_SIZE-1, status) == NULL)   
                    {   
                            fclose(status);   
                            continue;   
                    }   
  
                    fclose(status);   
  
                    // Buffer should contain a string like "Name: binary\_name"a   
  
                    sscanf(buffer, "%\*s %s", name);   
  
                    if ( name != NULL && name[0] != '\0')   
                    {   
                            if (strcmp(name, pidName) == 0)   
                            {   
                                    pidList=realloc( pidList, sizeof(long) \* (i+2));   
                                    pidList[i++]=strtol(next->d\_name, NULL, 0);   
                            }   
  
                    }   
  
            }   
  
            if (pidList)   
            {   
                    pidList[i]=0;   
            }   
            return pidList;   
    }   
  
  
  
    int main ( int argc , char \*\*argv)   
    {   
            long \*pid\_t=NULL;   
            if ( argc != 2 )   
            {   
                    fprintf(stdout,"Usage %s <filename>\n",argv[0]);   
                    return 0;   
            }   
  
            pid\_t = find\_pid\_by\_name( argv[1]);   
  
            while ( pid\_t != 0 && \*pid\_t != 0)   
            {   
                    fprintf(stdout,"\n%s is [ %d]\n",argv[1],\*pid\_t);   
                    pid\_t++;   
            }   
            return 0;   
    }   
  
  
这段程序中特别需要注意的是：   
  
main 函数中的 while(pid\_t != 0 && \*pid\_t != 0) 这一行，开始的时候俺没有注意，一直报段错误，仔细观察后发现，不仅需要判断这个指针的值是否是等于 0 ， 同时还得判断这个指针本身是否等于 0 ，否则会出现查询的程序根本就没有运行时出现段错误，或者是查询的程序运行了，也会出现段错误的情况。   
  
==============================================   
  
此程序在需要限制某些程序在特定的时间内只能一次运行的情况下很有好处，这种情况多数出现在金融系统、保险系统、收费系统等核心业务领域，通常在日终账务处理时，绝对不允许在同一个时间内连续两次运行扎帐程序，否则会造成业务数据紊乱。   
  
因此，我们可以在程序运行得时候，首先获取自身的进程号 pid （用 getpid（）函数获取），然后通过上述的程序代码，获取当前程序的进程号，如果存在两个，则退出当前的程序，从而保证在一个特定的时间内，扎帐程序只有一个运行。