------------------------------------------------------------------------------

# T H E  /proc   F I L E S Y S T E M

------------------------------------------------------------------------------

/proc/sys         Terrehon Bowden <terrehon@pacbell.net>        October 7 1999
                  Bodo Bauer <bb@ricochet.net>

2.4.x update      Jorge Nerin <comandante@zaralinux.com>      November 14 2000
move /proc/sys    Shen Feng <shen@cn.fujitsu.com>                  April 1 2009

------------------------------------------------------------------------------

Version 1.3                                              Kernel version 2.2.12
                                            Kernel version 2.4.0-test11-pre4

------------------------------------------------------------------------------

fixes/update part 1.1  Stefani Seibold <stefani@seibold.net>       June 9 2009

## Table of Contents



------------------------------------------------------------------------------

## Preface

------------------------------------------------------------------------------

### 0.1 Introduction/Credits

This documentation is  part of a soon (or  so we hope) to be  released book on the SuSE  Linux distribution. As  there is  no complete documentation  for the /proc file system and we've used  many freely available sources to write these chapters, it  seems only fair  to give the work  back to the  Linux community. This work is  based on the 2.2.*  kernel version and the  upcoming 2.4.*. I'm afraid it's still far from complete, but we  hope it will be useful. As far as we know, it is the first 'all-in-one' document about the /proc file system. It is focused on the Intel x86 hardware, so if you are looking for PPC, ARM, SPARC, AXP, etc., features, you probably won't find what you are looking for. It also only covers IPv4 networking, not IPv6 nor other protocols - sorry. But additions and patches are welcome and will be added to this document if you mail them to Bodo.

We'd like to thank Alan Cox, Rik van Riel, and Alexey Kuznetsov and a lot of other people for help compiling this documentation. We'd also like to extend a special thank you to Andi Kleen for documentation, which we relied on heavily to create this document, as well as the additional information he provided. Thanks to everybody else who contributed source or docs to the Linux kernel and helped create a great piece of software... :)

If you have any comments, corrections or additions, please don't hesitate to contact Bodo Bauer at bb@ricochet.net. We'll be happy to add them to this document.

The latest version of this document is available online at http://skaro.nightcrawler.com/~bb/Docs/Proc as HTML version.

If the above direction does not works for you, ypu could try the kernel mailing list at linux-kernel@vger.kernel.org and/or try to reach me at comandante@zaralinux.com.

## 0.2 Legal Stuff

We don't guarantee the correctness of this document, and if you come to us complaining about how you screwed up your system because of incorrect documentation, we won't feel responsible...

------------------------------------------------------------------------------

# CHAPTER 1: COLLECTING SYSTEM INFORMATION

------------------------------------------------------------------------------

------------------------------------------------------------------------------

## In This Chapter

------------------------------------------------------------------------------

* Investigating the properties of the pseudo file system /proc and its ability to provide information on the running Linux system
* Examining /proc's structure
* Uncovering various information about the kernel and the processes running on the system

------------------------------------------------------------------------------

The proc file system acts as an interface to internal data structures in the kernel. It can be used to obtain information about the system and to change certain kernel parameters at runtime (sysctl).

First, we'll take a look at the read-only parts of /proc. In Chapter 2, we show you how you can use /proc/sys to change settings.

## 1.1 Process-Specific Subdirectories

The directory /proc contains (among other things) one subdirectory for each process running on the system, which is named after the process ID (PID).

The link self points to the process reading the file system. Each process subdirectory has the entries listed in Table 1-1.

Table 1-1: Process specific entries in /proc

```
..............................................................................
 File           Content
 clear_refs     Clears page referenced bits shown in smaps output
 cmdline        Command line arguments
 cpu            Current and last cpu in which it was executed   (2.4)(smp)
 cwd            Link to the current working directory
 environ        Values of environment variables
 exe            Link to the executable of this process
 fd             Directory, which contains all file descriptors
 maps           Memory maps to executables and library files    (2.4)
 mem            Memory held by this process
 root           Link to the root directory of this process
 stat           Process status
 statm          Process memory status information
 status         Process status in human readable form
 wchan          If CONFIG_KALLSYMS is set, a pre-decoded wchan
 stack          Report full stack trace, enable via CONFIG_STACKTRACE
 smaps          a extension based on maps, showing the memory consumption of
                each mapping
..............................................................................
```

For example, to get the status information of a process, all you have to do is read the file /proc/PID/status:

```
  >cat /proc/self/status
  Name:   cat
  State:  R (running)
  Tgid:   5452
  Pid:    5452
  PPid:   743
  TracerPid:      0                                               (2.4)
  Uid:    501     501     501     501
  Gid:    100     100     100     100
  FDSize: 256
  Groups: 100 14 16
  VmPeak:     5004 kB
  VmSize:     5004 kB
  VmLck:         0 kB
  VmHWM:       476 kB
  VmRSS:       476 kB
  VmData:      156 kB
  VmStk:        88 kB
  VmExe:        68 kB
  VmLib:      1412 kB
  VmPTE:        20 kb
  VmSwap:        0 kB
  Threads:        1
```

```
  SigQ:   0/28578
  SigPnd: 0000000000000000
  ShdPnd: 0000000000000000
  SigBlk: 0000000000000000
  SigIgn: 0000000000000000
  SigCgt: 0000000000000000
  CapInh: 00000000fffffeff
  CapPrm: 0000000000000000
  CapEff: 0000000000000000
  CapBnd: ffffffffffffffff
  voluntary_ctxt_switches:        0
  nonvoluntary_ctxt_switches:     1
```

This shows you nearly the same information you would get if you viewed it with the ps command. In fact, ps uses the proc file system to obtain its information. But you get a more detailed view of the process by reading the file /proc/PID/status. It fields are described in table 1-2.

The statm file contains more detailed information about the process memory usage. Its seven fields are explained in Table 1-3. The stat file contains details information about the process itself. Its fields are explained in Table 1-4.

(for SMP CONFIG users)
For making accounting scalable, RSS related information are handled in asynchronous manner and the vaule may not be very precise. To see a precise snapshot of a moment, you can see /proc/<pid>/smaps file and scan page table. It's slow but very precise.

Table 1-2: Contents of the status files (as of 2.6.30-rc7)

| Field | Content |
|---|---|
| Name | filename of the executable |
| State | state (R is running, S is sleeping, D is sleeping in an uninterruptible wait, Z is zombie, T is traced or stopped) |
| Tgid | thread group ID |
| Pid | process id |
| PPid | process id of the parent process |
| TracerPid | PID of process tracing this process (0 if not) |
| Uid | Real, effective, saved set, and file system UIDs |
| Gid | Real, effective, saved set, and file system GIDs |
| FDSize | number of file descriptor slots currently allocated |
| Groups | supplementary group list |
| VmPeak | peak virtual memory size |
| VmSize | total program size |
| VmLck | locked memory size |
| VmHWM | peak resident set size ("high water mark") |
| VmRSS | size of memory portions |
| VmData | size of data, stack, and text segments |
| VmStk | size of data, stack, and text segments |
| VmExe | size of text segment |
| VmLib | size of shared library code |
| VmPTE | size of page table entries |
| VmSwap | size of swap usage (the number of referred swapents) |

```
 Threads                     number of threads
 SigQ                        number of signals queued/max. number for queue
 SigPnd                      bitmap of pending signals for the thread
 ShdPnd                      bitmap of shared pending signals for the process
 SigBlk                      bitmap of blocked signals
 SigIgn                      bitmap of ignored signals
 SigCgt                      bitmap of catched signals
 CapInh                      bitmap of inheritable capabilities
 CapPrm                      bitmap of permitted capabilities
 CapEff                      bitmap of effective capabilities
 CapBnd                      bitmap of capabilities bounding set
 Cpus_allowed                mask of CPUs on which this process may run
 Cpus_allowed_list           Same as previous, but in "list format"
 Mems_allowed                mask of memory nodes allowed to this process
 Mems_allowed_list           Same as previous, but in "list format"
 voluntary_ctxt_switches     number of voluntary context switches
 nonvoluntary_ctxt_switches  number of non voluntary context switches
..............................................................................

Table 1-3: Contents of the statm files (as of 2.6.8-rc3)
..............................................................................
 Field    Content
 size     total program size (pages)            (same as VmSize in status)
 resident size of memory portions (pages)       (same as VmRSS in status)
 shared   number of pages that are shared       (i.e. backed by a file)
 trs      number of pages that are 'code'       (not including libs; broken,
                                                        includes data segment)
 lrs      number of pages of library            (always 0 on 2.6)
 drs      number of pages of data/stack         (including libs; broken,
                                                        includes library text)
 dt       number of dirty pages                 (always 0 on 2.6)
..............................................................................


Table 1-4: Contents of the stat files (as of 2.6.30-rc7)
..............................................................................
 Field          Content
  pid           process id
  tcomm         filename of the executable
  state         state (R is running, S is sleeping, D is sleeping in an
                uninterruptible wait, Z is zombie, T is traced or stopped)
  ppid          process id of the parent process
  pgrp          pgrp of the process
  sid           session id
  tty_nr        tty the process uses
  tty_pgrp      pgrp of the tty
  flags         task flags
  min_flt       number of minor faults
  cmin_flt      number of minor faults with child's
  maj_flt       number of major faults
  cmaj_flt      number of major faults with child's
  utime         user mode jiffies
  stime         kernel mode jiffies
  cutime        user mode jiffies with child's
  cstime        kernel mode jiffies with child's
  priority      priority level
```

```
  nice          nice level
  num_threads   number of threads
  it_real_value (obsolete, always 0)
  start_time    time the process started after system boot
  vsize         virtual memory size
  rss           resident set memory size
  rsslim        current limit in bytes on the rss
  start_code    address above which program text can run
  end_code      address below which program text can run
  start_stack   address of the start of the stack
  esp           current value of ESP
  eip           current value of EIP
  pending       bitmap of pending signals
  blocked       bitmap of blocked signals
  sigign        bitmap of ignored signals
  sigcatch      bitmap of catched signals
  wchan         address where process went to sleep
  0             (place holder)
  0             (place holder)
  exit_signal   signal to send to parent thread on exit
  task_cpu      which CPU the task is scheduled on
  rt_priority   realtime priority
  policy        scheduling policy (man sched_setscheduler)
  blkio_ticks   time spent waiting for block IO
  gtime         guest time of the task in jiffies
  cgtime        guest time of the task children in jiffies
..............................................................................
```

The /proc/PID/maps file containing the currently mapped memory regions and their access permissions.

The format is:

```
address           perms offset  dev   inode      pathname

08048000-08049000 r-xp 00000000 03:00 8312       /opt/test
08049000-0804a000 rw-p 00001000 03:00 8312       /opt/test
0804a000-0806b000 rw-p 00000000 00:00 0          [heap]
a7cb1000-a7cb2000 ---p 00000000 00:00 0
a7cb2000-a7eb2000 rw-p 00000000 00:00 0
a7eb2000-a7eb3000 ---p 00000000 00:00 0
a7eb3000-a7ed5000 rw-p 00000000 00:00 0
a7ed5000-a8008000 r-xp 00000000 03:00 4222       /lib/libc.so.6
a8008000-a800a000 r--p 00133000 03:00 4222       /lib/libc.so.6
a800a000-a800b000 rw-p 00135000 03:00 4222       /lib/libc.so.6
a800b000-a800e000 rw-p 00000000 00:00 0
a800e000-a8022000 r-xp 00000000 03:00 14462      /lib/libpthread.so.0
a8022000-a8023000 r--p 00013000 03:00 14462      /lib/libpthread.so.0
a8023000-a8024000 rw-p 00014000 03:00 14462      /lib/libpthread.so.0
a8024000-a8027000 rw-p 00000000 00:00 0
a8027000-a8043000 r-xp 00000000 03:00 8317       /lib/ld-linux.so.2
a8043000-a8044000 r--p 0001b000 03:00 8317       /lib/ld-linux.so.2
a8044000-a8045000 rw-p 0001c000 03:00 8317       /lib/ld-linux.so.2
aff35000-aff4a000 rw-p 00000000 00:00 0          [stack]
ffffe000-fffff000 r-xp 00000000 00:00 0          [vdso]
```

where "address" is the address space in the process that it occupies, "perms" is a set of permissions:

```
 r = read
 w = write
 x = execute
 s = shared
 p = private (copy on write)
```

"offset" is the offset into the mapping, "dev" is the device (major:minor), and "inode" is the inode on that device. 0 indicates that no inode is associated with the memory region, as the case would be with BSS (uninitialized data). The "pathname" shows the name associated file for this mapping. If the mapping is not associated with a file:

```
 [heap]                  = the heap of the program
 [stack]                 = the stack of the main process
 [vdso]                  = the "virtual dynamic shared object",
                           the kernel system call handler
```

 or if empty, the mapping is anonymous.

The /proc/PID/smaps is an extension based on maps, showing the memory consumption for each of the process's mappings. For each of mappings there is a series of lines such as the following:

```
08048000-080bc000 r-xp 00000000 03:02 13130      /bin/bash
Size:               1084 kB
Rss:                 892 kB
Pss:                 374 kB
Shared_Clean:        892 kB
Shared_Dirty:          0 kB
Private_Clean:         0 kB
Private_Dirty:         0 kB
Referenced:          892 kB
Swap:                  0 kB
KernelPageSize:        4 kB
MMUPageSize:           4 kB
```

The first of these lines shows the same information as is displayed for the mapping in /proc/PID/maps. The remaining lines show the size of the mapping, the amount of the mapping that is currently resident in RAM, the "proportional set size" (divide each shared page by the number of processes sharing it), the number of clean and dirty shared pages in the mapping, and the number of clean and dirty private pages in the mapping. The "Referenced" indicates the amount of memory currently marked as referenced or accessed.

This file is only present if the CONFIG_MMU kernel configuration option is enabled.

The /proc/PID/clear_refs is used to reset the PG_Referenced and ACCESSED/YOUNG bits on both physical and virtual pages associated with a process.
To clear the bits for all the pages associated with the process

```
    > echo 1 > /proc/PID/clear_refs
```

To clear the bits for the anonymous pages associated with the process
    > echo 2 > /proc/PID/clear_refs

To clear the bits for the file mapped pages associated with the process
    > echo 3 > /proc/PID/clear_refs
Any other value written to /proc/PID/clear_refs will have no effect.

## 1.2 Kernel data

Similar to the process entries, the kernel data files give information about the running kernel. The files used to obtain this information are contained in /proc and are listed in Table 1-5. Not all of these will be present in your system. It depends on the kernel configuration and the loaded modules, which files are there, and which are missing.

Table 1-5: Kernel info in /proc

| File | Content | |
|---|---|---|
| apm | Advanced power management info | |
| buddyinfo | Kernel memory allocator information (see text) | (2.5) |
| bus | Directory containing bus specific information | |
| cmdline | Kernel command line | |
| cpuinfo | Info about the CPU | |
| devices | Available devices (block and character) | |
| dma | Used DMS channels | |
| filesystems | Supported filesystems | |
| driver | Various drivers grouped here, currently rtc (2.4) | |
| execdomains | Execdomains, related to security | (2.4) |
| fb | Frame Buffer devices | (2.4) |
| fs | File system parameters, currently nfs/exports | (2.4) |
| ide | Directory containing info about the IDE subsystem | |
| interrupts | Interrupt usage | |
| iomem | Memory map | (2.4) |
| ioports | I/O port usage | |
| irq | Masks for irq to cpu affinity | (2.4)(smp?) |
| isapnp | ISA PnP (Plug&Play) Info | (2.4) |
| kcore | Kernel core image (can be ELF or A.OUT(deprecated in 2.4)) | |
| kmsg | Kernel messages | |
| ksyms | Kernel symbol table | |
| loadavg | Load average of last 1, 5 & 15 minutes | |
| locks | Kernel locks | |
| meminfo | Memory info | |
| misc | Miscellaneous | |
| modules | List of loaded modules | |
| mounts | Mounted filesystems | |
| net | Networking info (see text) | |
| pagetypeinfo | Additional page allocator information (see text) | (2.5) |
| partitions | Table of partitions known to the system | |
| pci | Deprecated info of PCI bus (new way -> /proc/bus/pci/, decoupled by lspci | (2.4) |
| rtc | Real time clock | |
| scsi | SCSI info (see text) | |
| slabinfo | Slab pool info | |
| softirqs | softirq usage | |

```
 stat        Overall statistics
 swaps       Swap space utilization
 sys         See chapter 2
 sysvipc     Info of SysVIPC Resources (msg, sem, shm)          (2.4)
 tty         Info of tty drivers
 uptime      System uptime
 version     Kernel version
 video       bttv info of video resources                       (2.4)
 vmallocinfo Show vmalloced areas
..............................................................................
```

You can, for example, check which interrupts are currently in use and what they are used for by looking in the file /proc/interrupts:

```
  > cat /proc/interrupts
             CPU0
    0:    8728810          XT-PIC  timer
    1:        895          XT-PIC  keyboard
    2:          0          XT-PIC  cascade
    3:     531695          XT-PIC  aha152x
    4:    2014133          XT-PIC  serial
    5:      44401          XT-PIC  pcnet_cs
    8:          2          XT-PIC  rtc
   11:          8          XT-PIC  i82365
   12:     182918          XT-PIC  PS/2 Mouse
   13:          1          XT-PIC  fpu
   14:    1232265          XT-PIC  ide0
   15:          7          XT-PIC  ide1
  NMI:          0
```

In 2.4.* a couple of lines where added to this file LOC & ERR (this time is the output of a SMP machine):

```
  > cat /proc/interrupts

             CPU0       CPU1
    0:    1243498    1214548    IO-APIC-edge  timer
    1:       8949       8958    IO-APIC-edge  keyboard
    2:          0          0          XT-PIC  cascade
    5:      11286      10161    IO-APIC-edge  soundblaster
    8:          1          0    IO-APIC-edge  rtc
    9:      27422      27407    IO-APIC-edge  3c503
   12:     113645     113873    IO-APIC-edge  PS/2 Mouse
   13:          0          0          XT-PIC  fpu
   14:      22491      24012    IO-APIC-edge  ide0
   15:       2183       2415    IO-APIC-edge  ide1
   17:      30564      30414   IO-APIC-level  eth0
   18:        177        164   IO-APIC-level  bttv
  NMI:    2457961    2457959
  LOC:    2457882    2457881
  ERR:       2155
```

NMI is incremented in this case because every timer interrupt generates a NMI (Non Maskable Interrupt) which is used by the NMI Watchdog to detect lockups.

LOC is the local interrupt counter of the internal APIC of every CPU.

ERR is incremented in the case of errors in the IO-APIC bus (the bus that connects the CPUs in a SMP system. This means that an error has been detected, the IO-APIC automatically retry the transmission, so it should not be a big problem, but you should read the SMP-FAQ.

In 2.6.2* /proc/interrupts was expanded again. This time the goal was for /proc/interrupts to display every IRQ vector in use by the system, not just those considered 'most important'. The new vectors are:

THR -- interrupt raised when a machine check threshold counter (typically counting ECC corrected errors of memory or cache) exceeds a configurable threshold. Only available on some systems.

TRM -- a thermal event interrupt occurs when a temperature threshold has been exceeded for the CPU. This interrupt may also be generated when the temperature drops back to normal.

SPU -- a spurious interrupt is some interrupt that was raised then lowered by some IO device before it could be fully processed by the APIC. Hence the APIC sees the interrupt but does not know what device it came from. For this case the APIC will generate the interrupt with a IRQ vector of 0xff. This might also be generated by chipset bugs.

RES, CAL, TLB -- rescheduling, call and TLB flush interrupts are sent from one CPU to another per the needs of the OS. Typically, their statistics are used by kernel developers and interested users to determine the occurrence of interrupts of the given type.

The above IRQ vectors are displayed only when relevent. For example, the threshold vector does not exist on x86_64 platforms. Others are suppressed when the system is a uniprocessor. As of this writing, only i386 and x86_64 platforms support the new IRQ vector displays.

Of some interest is the introduction of the /proc/irq directory to 2.4. It could be used to set IRQ to CPU affinity, this means that you can "hook" an IRQ to only one CPU, or to exclude a CPU of handling IRQs. The contents of the irq subdir is one subdir for each IRQ, and two files; default_smp_affinity and prof_cpu_mask.

For example

```
  > ls /proc/irq/
  0  10  12  14  16  18  2  4  6  8  prof_cpu_mask
  1  11  13  15  17  19  3  5  7  9  default_smp_affinity
  > ls /proc/irq/0/
  smp_affinity
```

smp_affinity is a bitmask, in which you can specify which CPUs can handle the IRQ, you can set it by doing:

```
  > echo 1 > /proc/irq/10/smp_affinity
```

This means that only the first CPU will handle the IRQ, but you can also echo 5 which means that only the first and fourth CPU can handle the IRQ.

The contents of each smp_affinity file is the same by default:

```
  > cat /proc/irq/0/smp_affinity
  ffffffff
```

The default_smp_affinity mask applies to all non-active IRQs, which are the IRQs which have not yet been allocated/activated, and hence which lack a /proc/irq/[0-9]* directory.

The node file on an SMP system shows the node to which the device using the IRQ reports itself as being attached. This hardware locality information does not include information about any possible driver locality preference.

prof_cpu_mask specifies which CPUs are to be profiled by the system wide profiler. Default value is ffffffff (all cpus).

The way IRQs are routed is handled by the IO-APIC, and it's Round Robin between all the CPUs which are allowed to handle it. As usual the kernel has more info than you and does a better job than you, so the defaults are the best choice for almost everyone.

There are three more important subdirectories in /proc: net, scsi, and sys. The general rule is that the contents, or even the existence of these directories, depend on your kernel configuration. If SCSI is not enabled, the directory scsi may not exist. The same is true with the net, which is there only when networking support is present in the running kernel.

The slabinfo file gives information about memory usage at the slab level. Linux uses slab pools for memory management above page level in version 2.2. Commonly used objects have their own slab pool (such as network buffers, directory cache, and so on).

..............................................................................

```
> cat /proc/buddyinfo

Node 0, zone      DMA      0      4      5      4      4      3 ...
Node 0, zone   Normal      1      0      0      1    101      8 ...
Node 0, zone  HighMem      2      0      0      1      1      0 ...
```

External fragmentation is a problem under some workloads, and buddyinfo is a useful tool for helping diagnose these problems.  Buddyinfo will give you a clue as to how big an area you can safely allocate, or why a previous allocation failed.

Each column represents the number of pages of a certain order which are available.  In this case, there are 0 chunks of 2^0*PAGE_SIZE available in ZONE_DMA, 4 chunks of 2^1*PAGE_SIZE in ZONE_DMA, 101 chunks of 2^4*PAGE_SIZE available in ZONE_NORMAL, etc...

More information relevant to external fragmentation can be found in pagetypeinfo.

```
> cat /proc/pagetypeinfo
Page block order: 9
Pages per block:  512
```

```
Free pages count per migrate type at order       0      1      2      3      4
   5      6      7      8      9     10
Node    0, zone      DMA, type    Unmovable      0      0      0      1      1
   1      1      1      1      1      0
Node    0, zone      DMA, type  Reclaimable      0      0      0      0      0
   0      0      0      0      0      0
Node    0, zone      DMA, type      Movable      1      1      2      1      2
   1      1      0      1      0      2
Node    0, zone      DMA, type      Reserve      0      0      0      0      0
   0      0      0      0      1      0
Node    0, zone      DMA, type      Isolate      0      0      0      0      0
   0      0      0      0      0      0
Node    0, zone    DMA32, type    Unmovable    103     54     77      1      1
   1     11      8      7      1      9
Node    0, zone    DMA32, type  Reclaimable      0      0      2      1      0
   0      0      0      1      0      0
Node    0, zone    DMA32, type      Movable    169    152    113     91     77
  54     39     13      6      1    452
Node    0, zone    DMA32, type      Reserve      1      2      2      2      2
   0      1      1      1      1      0
Node    0, zone    DMA32, type      Isolate      0      0      0      0      0
   0      0      0      0      0      0

Number of blocks type     Unmovable  Reclaimable      Movable      Reserve
Isolate
Node 0, zone      DMA            2            0            5            1
    0
Node 0, zone    DMA32           41            6          967            2
    0
```

Fragmentation avoidance in the kernel works by grouping pages of different migrate types into the same contiguous regions of memory called page blocks. A page block is typically the size of the default hugepage size e.g. 2MB on X86-64. By keeping pages grouped based on their ability to move, the kernel can reclaim pages within a page block to satisfy a high-order allocation.

The pagetypinfo begins with information on the size of a page block. It then gives the same type of information as buddyinfo except broken down by migrate-type and finishes with details on how many page blocks of each type exist.

If min_free_kbytes has been tuned correctly (recommendations made by hugeadm from libhugetlbfs http://sourceforge.net/projects/libhugetlbfs/), one can make an estimate of the likely number of huge pages that can be allocated at a given point in time. All the "Movable" blocks should be allocatable unless memory has been mlock()'d. Some of the Reclaimable blocks should also be allocatable although a lot of filesystem metadata may have to be reclaimed to achieve this.

..............................................................................

meminfo:

Provides information about distribution and utilization of memory.  This varies by architecture and compile options.  The following is from a 16GB PIII, which has highmem enabled.  You may not have all of these fields.

```
> cat /proc/meminfo


MemTotal:     16344972 kB
MemFree:      13634064 kB
Buffers:          3656 kB
Cached:        1195708 kB
SwapCached:          0 kB
Active:         891636 kB
Inactive:      1077224 kB
HighTotal:    15597528 kB
HighFree:     13629632 kB
LowTotal:       747444 kB
LowFree:          4432 kB
SwapTotal:           0 kB
SwapFree:            0 kB
Dirty:             968 kB
Writeback:           0 kB
AnonPages:      861800 kB
Mapped:         280372 kB
Slab:           284364 kB
SReclaimable:   159856 kB
SUnreclaim:     124508 kB
PageTables:      24448 kB
NFS_Unstable:        0 kB
Bounce:              0 kB
WritebackTmp:        0 kB
CommitLimit:   7669796 kB
Committed_AS:   100056 kB
VmallocTotal:   112216 kB
VmallocUsed:       428 kB
VmallocChunk:   111088 kB

    MemTotal: Total usable ram (i.e. physical ram minus a few reserved
              bits and the kernel binary code)
     MemFree: The sum of LowFree+HighFree
     Buffers: Relatively temporary storage for raw disk blocks
              shouldn't get tremendously large (20MB or so)
      Cached: in-memory cache for files read from the disk (the
              pagecache).  Doesn't include SwapCached
  SwapCached: Memory that once was swapped out, is swapped back in but
              still also is in the swapfile (if memory is needed it
              doesn't need to be swapped out AGAIN because it is already
              in the swapfile. This saves I/O)
      Active: Memory that has been used more recently and usually not
              reclaimed unless absolutely necessary.
    Inactive: Memory which has been less recently used.  It is more
              eligible to be reclaimed for other purposes
   HighTotal:
    HighFree: Highmem is all memory above ~860MB of physical memory
              Highmem areas are for use by userspace programs, or
              for the pagecache.  The kernel must use tricks to access
              this memory, making it slower to access than lowmem.
    LowTotal:
     LowFree: Lowmem is memory which can be used for everything that
```

```
              highmem can be used for, but it is also available for the
              kernel's use for its own data structures.  Among many
              other things, it is where everything from the Slab is
              allocated.  Bad things happen when you're out of lowmem.
   SwapTotal: total amount of swap space available
    SwapFree: Memory which has been evicted from RAM, and is temporarily
              on the disk
       Dirty: Memory which is waiting to get written back to the disk
   Writeback: Memory which is actively being written back to the disk
   AnonPages: Non-file backed pages mapped into userspace page tables
      Mapped: files which have been mmaped, such as libraries
        Slab: in-kernel data structures cache
SReclaimable: Part of Slab, that might be reclaimed, such as caches
  SUnreclaim: Part of Slab, that cannot be reclaimed on memory pressure
  PageTables: amount of memory dedicated to the lowest level of page
              tables.
NFS_Unstable: NFS pages sent to the server, but not yet committed to stable
              storage
      Bounce: Memory used for block device "bounce buffers"
WritebackTmp: Memory used by FUSE for temporary writeback buffers
 CommitLimit: Based on the overcommit ratio ('vm.overcommit_ratio'),
              this is the total amount of  memory currently available to
              be allocated on the system. This limit is only adhered to
              if strict overcommit accounting is enabled (mode 2 in
              'vm.overcommit_memory').
              The CommitLimit is calculated with the following formula:
              CommitLimit = ('vm.overcommit_ratio' * Physical RAM) + Swap
              For example, on a system with 1G of physical RAM and 7G
              of swap with a `vm.overcommit_ratio` of 30 it would
              yield a CommitLimit of 7.3G.
              For more details, see the memory overcommit documentation
              in vm/overcommit-accounting.
Committed_AS: The amount of memory presently allocated on the system.
              The committed memory is a sum of all of the memory which
              has been allocated by processes, even if it has not been
              "used" by them as of yet. A process which malloc()'s 1G
              of memory, but only touches 300M of it will only show up
              as using 300M of memory even if it has the address space
              allocated for the entire 1G. This 1G is memory which has
              been "committed" to by the VM and can be used at any time
              by the allocating application. With strict overcommit
              enabled on the system (mode 2 in 'vm.overcommit_memory'),
              allocations which would exceed the CommitLimit (detailed
              above) will not be permitted. This is useful if one needs
              to guarantee that processes will not fail due to lack of
              memory once that memory has been successfully allocated.
VmallocTotal: total size of vmalloc memory area
 VmallocUsed: amount of vmalloc area which is used
VmallocChunk: largest contiguous block of vmalloc area which is free

..............................................................................

vmallocinfo:

Provides information about vmalloced/vmaped areas. One line per area,
containing the virtual address range of the area, size in bytes,
```

caller information of the creator, and optional information depending
on the kind of area :

```
 pages=nr    number of pages
 phys=addr   if a physical address was specified
 ioremap     I/O mapping (ioremap() and friends)
 vmalloc     vmalloc() area
 vmap        vmap()ed pages
 user        VM_USERMAP area
 vpages      buffer for pages pointers was vmalloced (huge area)
 N<node>=nr  (Only on NUMA kernels)
             Number of pages allocated on memory node <node>
```

```
> cat /proc/vmallocinfo
0xffffc20000000000-0xffffc20000201000 2101248 alloc_large_system_hash+0x204 ...
  /0x2c0 pages=512 vmalloc N0=128 N1=128 N2=128 N3=128
0xffffc20000201000-0xffffc20000302000 1052672 alloc_large_system_hash+0x204 ...
  /0x2c0 pages=256 vmalloc N0=64 N1=64 N2=64 N3=64
0xffffc20000302000-0xffffc20000304000    8192 acpi_tb_verify_table+0x21/0x4f...
  phys=7fee8000 ioremap
0xffffc20000304000-0xffffc20000307000   12288 acpi_tb_verify_table+0x21/0x4f...
  phys=7fee7000 ioremap
0xffffc2000031d000-0xffffc2000031f000    8192 init_vdso_vars+0x112/0x210
0xffffc2000031f000-0xffffc2000032b000   49152 cramfs_uncompress_init+0x2e ...
  /0x80 pages=11 vmalloc N0=3 N1=3 N2=2 N3=3
0xffffc2000033a000-0xffffc2000033d000   12288 sys_swapon+0x640/0xac0      ...
  pages=2 vmalloc N1=2
0xffffc20000347000-0xffffc2000034c000   20480 xt_alloc_table_info+0xfe ...
  /0x130 [x_tables] pages=4 vmalloc N0=4
0xffffffffa0000000-0xffffffffa000f000   61440 sys_init_module+0xc27/0x1d00 ...
   pages=14 vmalloc N2=14
0xffffffffa000f000-0xffffffffa0014000   20480 sys_init_module+0xc27/0x1d00 ...
   pages=4 vmalloc N1=4
0xffffffffa0014000-0xffffffffa0017000   12288 sys_init_module+0xc27/0x1d00 ...
   pages=2 vmalloc N1=2
0xffffffffa0017000-0xffffffffa0022000   45056 sys_init_module+0xc27/0x1d00 ...
   pages=10 vmalloc N0=10
```

..............................................................................

softirqs:

Provides counts of softirq handlers serviced since boot time, for each cpu.

```
> cat /proc/softirqs
                CPU0       CPU1       CPU2       CPU3
      HI:          0          0          0          0
   TIMER:      27166      27120      27097      27034
  NET_TX:          0          0          0         17
  NET_RX:         42          0          0         39
   BLOCK:          0          0        107       1121
 TASKLET:          0          0          0        290
   SCHED:      27035      26983      26971      26746
 HRTIMER:          0          0          0          0
     RCU:       1678       1769       2178       2250
```

## 1.3 IDE devices in /proc/ide

The subdirectory /proc/ide contains information about all IDE devices of which the kernel is aware. There is one subdirectory for each IDE controller, the file drivers and a link for each IDE device, pointing to the device directory in the controller specific subtree.

The file drivers contains general information about the drivers used for the IDE devices:

```
  > cat /proc/ide/drivers
  ide-cdrom version 4.53
  ide-disk version 1.08
```

More detailed information can be found in the controller specific subdirectories. These are named ide0, ide1 and so on. Each of these directories contains the files shown in table 1-6.

Table 1-6: IDE controller info in /proc/ide/ide?

| File | Content |
|---|---|
| channel | IDE channel (0 or 1) |
| config | Configuration (only for PCI/IDE bridge) |
| mate | Mate name |
| model | Type/Chipset of IDE controller |

Each device connected to a controller has a separate subdirectory in the controllers directory. The files listed in table 1-7 are contained in these directories.

Table 1-7: IDE device information

| File | Content |
|---|---|
| cache | The cache |
| capacity | Capacity of the medium (in 512Byte blocks) |
| driver | driver and version |
| geometry | physical and logical geometry |
| identify | device identify block |
| media | media type |
| model | device identifier |
| settings | device setup |
| smart_thresholds | IDE disk management thresholds |
| smart_values | IDE disk management values |

The most interesting file is settings. This file contains a nice overview of the drive parameters:

```
  # cat /proc/ide/ide0/hda/settings
  name                    value           min             max             mode
  ----                    -----           ---             ---             ----
```

| | | | | |
|---|---|---|---|---|
| bios_cyl | 526 | 0 | 65535 | rw |
| bios_head | 255 | 0 | 255 | rw |
| bios_sect | 63 | 0 | 63 | rw |
| breada_readahead | 4 | 0 | 127 | rw |
| bswap | 0 | 0 | 1 | r |
| file_readahead | 72 | 0 | 2097151 | rw |
| io_32bit | 0 | 0 | 3 | rw |
| keepsettings | 0 | 0 | 1 | rw |
| max_kb_per_request | 122 | 1 | 127 | rw |
| multcount | 0 | 0 | 8 | rw |
| nice1 | 1 | 0 | 1 | rw |
| nowerr | 0 | 0 | 1 | rw |
| pio_mode | write-only | 0 | 255 | w |
| slow | 0 | 0 | 1 | rw |
| unmaskirq | 0 | 0 | 1 | rw |
| using_dma | 0 | 0 | 1 | rw |

## 1.4 Networking info in /proc/net

The subdirectory /proc/net follows the usual pattern. Table 1-8 shows the additional values you get for IP version 6 if you configure the kernel to support this. Table 1-9 lists the files and their meaning.

Table 1-8: IPv6 info in /proc/net

| File | Content |
|---|---|
| udp6 | UDP sockets (IPv6) |
| tcp6 | TCP sockets (IPv6) |
| raw6 | Raw device statistics (IPv6) |
| igmp6 | IP multicast addresses, which this host joined (IPv6) |
| if_inet6 | List of IPv6 interface addresses |
| ipv6_route | Kernel routing table for IPv6 |
| rt6_stats | Global IPv6 routing tables statistics |
| sockstat6 | Socket statistics (IPv6) |
| snmp6 | Snmp data (IPv6) |

Table 1-9: Network info in /proc/net

| File | Content |
|---|---|
| arp | Kernel ARP table |
| dev | network devices with statistics |
| dev_mcast | the Layer2 multicast groups a device is listening too (interface index, label, number of references, number of bound addresses). |
| dev_stat | network device status |
| ip_fwchains | Firewall chain linkage |
| ip_fwnames | Firewall chain names |
| ip_masq | Directory containing the masquerading tables |
| ip_masquerade | Major masquerading table |
| netstat | Network statistics |
| raw | raw device statistics |

```
 route          Kernel routing table
 rpc            Directory containing rpc info
 rt_cache       Routing cache
 snmp           SNMP data
 sockstat       Socket statistics
 tcp            TCP  sockets
 tr_rif         Token ring RIF routing table
 udp            UDP sockets
 unix           UNIX domain sockets
 wireless       Wireless interface data (Wavelan etc)
 igmp           IP multicast addresses, which this host joined
 psched         Global packet scheduler parameters.
 netlink        List of PF_NETLINK sockets
 ip_mr_vifs     List of multicast virtual interfaces
 ip_mr_cache    List of multicast routing cache
..............................................................................
```

You can  use  this  information  to see which network devices are available in your system and how much traffic was routed over those devices:

```
  > cat /proc/net/dev
  Inter-|Receive                                                   |[...
   face |bytes    packets errs drop fifo frame compressed multicast|[...
      lo:  908188   5596     0    0    0     0          0         0 [...

    ppp0:15475140  20721   410    0    0   410          0         0 [...
    eth0:  614530   7085     0    0    0     0          0         1 [...

  ...] Transmit
  ...] bytes    packets errs drop fifo colls carrier compressed
  ...]  908188     5596    0    0    0     0       0          0
  ...] 1375103    17405    0    0    0     0       0          0
  ...] 1703981     5535    0    0    0     3       0          0
```

In addition, each Channel Bond interface has its own directory.  For example, the bond0 device will have a directory called /proc/net/bond0/. It will contain information that is specific to that bond, such as the current slaves of the bond, the link status of the slaves, and how many times the slaves link has failed.

## 1.5 SCSI info

If you  have  a  SCSI  host adapter in your system, you'll find a subdirectory named after  the driver for this adapter in /proc/scsi. You'll also see a list of all recognized SCSI devices in /proc/scsi:

```
  >cat /proc/scsi/scsi
  Attached devices:
  Host: scsi0 Channel: 00 Id: 00 Lun: 00
    Vendor: IBM      Model: DGHS09U          Rev: 03E0
    Type:   Direct-Access                    ANSI SCSI revision: 03
  Host: scsi0 Channel: 00 Id: 06 Lun: 00
    Vendor: PIONEER  Model: CD-ROM DR-U06S   Rev: 1.04
    Type:   CD-ROM                           ANSI SCSI revision: 02
```

The directory named after the driver has one file for each adapter found in the system. These files contain information about the controller, including the used IRQ and the IO address range. The amount of information shown is dependent on the adapter you use. The example shows the output for an Adaptec AHA-2940 SCSI adapter:

```
  > cat /proc/scsi/aic7xxx/0

  Adaptec AIC7xxx driver version: 5.1.19/3.2.4
  Compile Options:
    TCQ Enabled By Default : Disabled
    AIC7XXX_PROC_STATS     : Disabled
    AIC7XXX_RESET_DELAY    : 5
  Adapter Configuration:
             SCSI Adapter: Adaptec AHA-294X Ultra SCSI host adapter
                             Ultra Wide Controller
      PCI MMAPed I/O Base: 0xeb001000
   Adapter SEEPROM Config: SEEPROM found and used.
        Adaptec SCSI BIOS: Enabled
                      IRQ: 10
                     SCBs: Active 0, Max Active 2,
                           Allocated 15, HW 16, Page 255
               Interrupts: 160328
        BIOS Control Word: 0x18b6
     Adapter Control Word: 0x005b
     Extended Translation: Enabled
  Disconnect Enable Flags: 0xffff
       Ultra Enable Flags: 0x0001
   Tag Queue Enable Flags: 0x0000
  Ordered Queue Tag Flags: 0x0000
  Default Tag Queue Depth: 8
      Tagged Queue By Device array for aic7xxx host instance 0:
        {255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255}
      Actual queue depth per device for aic7xxx host instance 0:
        {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}
  Statistics:
  (scsi0:0:0:0)
    Device using Wide/Sync transfers at 40.0 MByte/sec, offset 8
    Transinfo settings: current(12/8/1/0), goal(12/8/1/0), user(12/15/1/0)
    Total transfers 160151 (74577 reads and 85574 writes)
  (scsi0:0:6:0)
    Device using Narrow/Sync transfers at 5.0 MByte/sec, offset 15
    Transinfo settings: current(50/15/0/0), goal(50/15/0/0), user(50/15/0/0)
    Total transfers 0 (0 reads and 0 writes)
```

## 1.6 Parallel port info in /proc/parport
---------------------------------------

The directory /proc/parport contains information about the parallel ports of your system. It has one subdirectory for each port, named after the port number (0,1,2,...).

These directories contain the four files shown in Table 1-10.

Table 1-10: Files in /proc/parport

```
..............................................................................
 File      Content
 autoprobe Any IEEE-1284 device ID information that has been acquired.
 devices   list of the device drivers using that port. A + will appear by the
           name of the device currently using the port (it might not appear
           against any).
 hardware  Parallel port's base address, IRQ line and DMA channel.
 irq       IRQ that parport is using for that port. This is in a separate
           file to allow you to alter it by writing a new value in (IRQ
           number or none).
..............................................................................
```

1.7 TTY info in /proc/tty
-------------------------

Information about the available and actually used tty's can be found in the directory /proc/tty.You'll find entries for drivers and line disciplines in this directory, as shown in Table 1-11.

Table 1-11: Files in /proc/tty

```
..............................................................................
 File          Content
 drivers       list of drivers and their usage
 ldiscs        registered line disciplines
 driver/serial usage statistic and status of single tty lines
..............................................................................
```

To see which tty's are currently in use, you can simply look into the file /proc/tty/drivers:

```
  > cat /proc/tty/drivers
  pty_slave            /dev/pts      136   0-255 pty:slave
  pty_master           /dev/ptm      128   0-255 pty:master
  pty_slave            /dev/ttyp       3   0-255 pty:slave
  pty_master           /dev/pty        2   0-255 pty:master
  serial               /dev/cua        5   64-67 serial:callout
  serial               /dev/ttyS       4   64-67 serial
  /dev/tty0            /dev/tty0       4       0 system:vtmaster
  /dev/ptmx            /dev/ptmx       5       2 system
  /dev/console         /dev/console    5       1 system:console
  /dev/tty             /dev/tty        5       0 system:/dev/tty
  unknown              /dev/tty        4    1-63 console
```

1.8 Miscellaneous kernel statistics in /proc/stat
-------------------------------------------------

Various pieces of information about kernel activity are available in the /proc/stat file. All of the numbers reported in this file are aggregates since the system first booted. For a quick look, simply cat the file:

```
  > cat /proc/stat
  cpu  2255 34 2290 22625563 6290 127 456 0 0
```

```
  cpu0 1132 34 1441 11311718 3675 127 438 0 0
  cpu1 1123 0 849 11313845 2614 0 18 0 0
  intr 114930548 113199788 3 0 5 263 0 4 [... lots more numbers ...]
  ctxt 1990473
  btime 1062191376
  processes 2915
  procs_running 1
  procs_blocked 0
  softirq 183433 0 21755 12 39 1137 231 21459 2263
```

The very first "cpu" line aggregates the numbers in all of the other "cpuN" lines. These numbers identify the amount of time the CPU has spent performing different kinds of work. Time units are in USER_HZ (typically hundredths of a second). The meanings of the columns are as follows, from left to right:

- user: normal processes executing in user mode
- nice: niced processes executing in user mode
- system: processes executing in kernel mode
- idle: twiddling thumbs
- iowait: waiting for I/O to complete
- irq: servicing interrupts
- softirq: servicing softirqs
- steal: involuntary wait
- guest: running a normal guest
- guest_nice: running a niced guest

The "intr" line gives counts of interrupts serviced since boot time, for each of the possible system interrupts. The first column is the total of all interrupts serviced; each subsequent column is the total for that particular interrupt.

The "ctxt" line gives the total number of context switches across all CPUs.

The "btime" line gives the time at which the system booted, in seconds since the Unix epoch.

The "processes" line gives the number of processes and threads created, which includes (but is not limited to) those created by calls to the fork() and clone() system calls.

The "procs_running" line gives the total number of threads that are running or ready to run (i.e., the total number of runnable threads).

The "procs_blocked" line gives the number of processes currently blocked, waiting for I/O to complete.

The "softirq" line gives counts of softirqs serviced since boot time, for each of the possible system softirqs. The first column is the total of all softirqs serviced; each subsequent column is the total for that particular softirq.

## 1.9 Ext4 file system parameters

Information about mounted ext4 file systems can be found in

/proc/fs/ext4.  Each mounted filesystem will have a directory in
/proc/fs/ext4 based on its device name (i.e., /proc/fs/ext4/hdc or
/proc/fs/ext4/dm-0).   The files in each per-device directory are shown
in Table 1-12, below.

Table 1-12: Files in /proc/fs/ext4/<devname>

| File | Content |
|---|---|
| mb_groups | details of multiblock allocator buddy cache of free blocks |

------------------------------------------------------------------------------

Summary

------------------------------------------------------------------------------

The /proc file system serves information about the running system. It not only allows access to process data but also allows you to request the kernel status by reading files in the hierarchy.

The directory structure of /proc reflects the types of information and makes it easy, if not obvious, where to look for specific data.

------------------------------------------------------------------------------

------------------------------------------------------------------------------

CHAPTER 2: MODIFYING SYSTEM PARAMETERS

------------------------------------------------------------------------------

------------------------------------------------------------------------------

In This Chapter

------------------------------------------------------------------------------

* Modifying kernel parameters by writing into files found in /proc/sys
* Exploring the files which modify certain parameters
* Review of the /proc/sys file tree

------------------------------------------------------------------------------

A very interesting part of /proc is the directory /proc/sys. This is not only a source of information, it also allows you to change parameters within the kernel. Be very careful when attempting this. You can optimize your system, but you can also cause it to crash. Never alter kernel parameters on a production system. Set up a development machine and test to make sure that everything works the way you want it to. You may have no alternative but to reboot the machine once an error has been made.

To change a value, simply echo the new value into the file. An example is given below in the section on the file system data. You need to be root to do this. You can create your own boot script to perform this every time your system boots.

The files in /proc/sys can be used to fine tune and monitor miscellaneous and general things in the operation of the Linux kernel. Since some of the files can inadvertently disrupt your system, it is advisable to read both documentation and source before actually making adjustments. In any case, be very careful when writing to any of these files. The entries in /proc may change slightly between the 2.1.* and the 2.2 kernel, so if there is any doubt review the kernel documentation in the directory /usr/src/linux/Documentation.

This chapter is heavily based on the documentation included in the pre 2.2 kernels, and became part of it in version 2.2.1 of the Linux kernel.

Please see: Documentation/sysctls/ directory for descriptions of these entries.

---

## Summary

---

Certain aspects of kernel behavior can be modified at runtime, without the need to recompile the kernel, or even to reboot the system. The files in the /proc/sys tree can not only be read, but also modified. You can use the echo command to write value into these files, thereby changing the default settings of the kernel.

---

---

## CHAPTER 3: PER-PROCESS PARAMETERS

---

### 3.1 /proc/<pid>/oom_adj - Adjust the oom-killer score

This file can be used to adjust the score used to select which processes should be killed in an out-of-memory situation. Giving it a high score will increase the likelihood of this process being killed by the oom-killer. Valid values are in the range -16 to +15, plus the special value -17, which disables oom-killing altogether for this process.

The process to be killed in an out-of-memory situation is selected among all others
based on its badness score. This value equals the original memory size of the process
and is then updated according to its CPU time (utime + stime) and the
run time (uptime - start time). The longer it runs the smaller is the score.
Badness score is divided by the square root of the CPU time and then by
the double square root of the run time.

Swapped out tasks are killed first. Half of each child's memory size is added to the parent's score if they do not share the same memory. Thus forking servers are the prime candidates to be killed. Having only one 'hungry' child will make parent less preferable than the child.

/proc/<pid>/oom_score shows process' current badness score.

The following heuristics are then applied:

```
 * if the task was reniced, its score doubles
 * superuser or direct hardware access tasks (CAP_SYS_ADMIN, CAP_SYS_RESOURCE
        or CAP_SYS_RAWIO) have their score divided by 4
 * if oom condition happened in one cpuset and checked process does not belong
        to it, its score is divided by 8
 * the resulting score is multiplied by two to the power of oom_adj, i.e.
        points <<= oom_adj when it is positive and
        points >>= -(oom_adj) otherwise
```

The task with the highest badness score is then selected and its children are killed, process itself will be killed in an OOM situation when it does not have children or some of them disabled oom like described above.

## 3.2 /proc/<pid>/oom_score - Display current oom-killer score

This file can be used to check the current score used by the oom-killer is for any given <pid>. Use it together with /proc/<pid>/oom_adj to tune which process should be killed in an out-of-memory situation.

## 3.3 /proc/<pid>/io - Display the IO accounting fields

This file contains IO statistics for each running process

### Example

```
test:/tmp # dd if=/dev/zero of=/tmp/test.dat &
[1] 3828

test:/tmp # cat /proc/3828/io
rchar: 323934931
wchar: 323929600
syscr: 632687
syscw: 632675
read_bytes: 0
write_bytes: 323932160
cancelled_write_bytes: 0
```

### Description

#### rchar

I/O counter: chars read
The number of bytes which this task has caused to be read from storage. This is simply the sum of bytes which this process passed to read() and pread(). It includes things like tty IO and it is unaffected by whether or not actual physical disk IO was required (the read might have been satisfied from pagecache)

#### wchar

I/O counter: chars written
The number of bytes which this task has caused, or shall cause to be written to disk. Similar caveats apply here as with rchar.

#### syscr

I/O counter: read syscalls
Attempt to count the number of read I/O operations, i.e. syscalls like read() and pread().

syscw
-----

I/O counter: write syscalls
Attempt to count the number of write I/O operations, i.e. syscalls like write() and pwrite().

read_bytes
----------

I/O counter: bytes read
Attempt to count the number of bytes which this process really did cause to be fetched from the storage layer. Done at the submit_bio() level, so it is accurate for block-backed filesystems. <please add status regarding NFS and CIFS at a later time>

write_bytes
-----------

I/O counter: bytes written
Attempt to count the number of bytes which this process caused to be sent to the storage layer. This is done at page-dirtying time.

cancelled_write_bytes
---------------------

The big inaccuracy here is truncate. If a process writes 1MB to a file and then deletes the file, it will in fact perform no writeout. But it will have been accounted as having caused 1MB of write.
In other words: The number of bytes which this process caused to not happen, by truncating pagecache. A task can cause "negative" IO too. If this task truncates some dirty pagecache, some IO which another task has been accounted for (in its write_bytes) will not be happening. We _could_ just subtract that from the truncating task's write_bytes, but there is information loss in doing that.

Note
----

At its current implementation state, this is a bit racy on 32-bit machines: if process A reads process B's /proc/pid/io while process B is updating one of those 64-bit counters, process A could see an intermediate result.

More information about this can be found within the taskstats documentation in Documentation/accounting.

## 3.4 /proc/<pid>/coredump_filter - Core dump filtering settings

When a process is dumped, all anonymous memory is written to a core file as long as the size of the core file isn't limited. But sometimes we don't want to dump some memory segments, for example, huge shared memory. Conversely, sometimes we want to save file-backed memory segments into a core file, not only the individual files.

/proc/<pid>/coredump_filter allows you to customize which memory segments will be dumped when the <pid> process is dumped. coredump_filter is a bitmask of memory types. If a bit of the bitmask is set, memory segments of the corresponding memory type are dumped, otherwise they are not dumped.

The following 7 memory types are supported:
- (bit 0) anonymous private memory
- (bit 1) anonymous shared memory
- (bit 2) file-backed private memory
- (bit 3) file-backed shared memory
- (bit 4) ELF header pages in file-backed private memory areas (it is effective only if the bit 2 is cleared)
- (bit 5) hugetlb private memory
- (bit 6) hugetlb shared memory

Note that MMIO pages such as frame buffer are never dumped and vDSO pages are always dumped regardless of the bitmask status.

Note bit 0-4 doesn't effect any hugetlb memory. hugetlb memory are only effected by bit 5-6.

Default value of coredump_filter is 0x23; this means all anonymous memory segments and hugetlb private memory are dumped.

If you don't want to dump all shared memory segments attached to pid 1234, write 0x21 to the process's proc file.

```
  $ echo 0x21 > /proc/1234/coredump_filter
```

When a new process is created, the process inherits the bitmask status from its parent. It is useful to set up coredump_filter before the program runs.
For example:

```
  $ echo 0x7 > /proc/self/coredump_filter
  $ ./some_program
```

## 3.5 /proc/<pid>/mountinfo - Information about mounts

This file contains lines of the form:

```
36 35 98:0 /mnt1 /mnt2 rw,noatime master:1 - ext3 /dev/root rw,errors=continue
(1)(2)(3)   (4)   (5)      (6)      (7)   (8) (9)   (10)         (11)
```

(1) mount ID:  unique identifier of the mount (may be reused after umount)
(2) parent ID:  ID of parent (or of self for the top of the mount tree)
(3) major:minor:  value of st_dev for files on filesystem

```
(4) root:  root of the mount within the filesystem
(5) mount point:  mount point relative to the process's root
(6) mount options:  per mount options
(7) optional fields:  zero or more fields of the form "tag[:value]"
(8) separator:  marks the end of the optional fields
(9) filesystem type:  name of filesystem of the form "type[.subtype]"
(10) mount source:  filesystem specific information or "none"
(11) super options:  per super block options

Parsers should ignore all unrecognised optional fields.  Currently the
possible optional fields are:

shared:X  mount is shared in peer group X
master:X  mount is slave to peer group X
propagate_from:X  mount is slave and receives propagation from peer group X (*)
unbindable  mount is unbindable

(*) X is the closest dominant peer group under the process's root.  If
X is the immediate master of the mount, or if there's no dominant peer
group under the same root, then only the "master:X" field is present
and not the "propagate_from:X" field.

For more information on mount propagation see:

  Documentation/filesystems/sharedsubtree.txt


3.6	/proc/<pid>/comm  & /proc/<pid>/task/<tid>/comm
--------------------------------------------------------
These files provide a method to access a tasks comm value. It also allows for
a task to set its own or one of its thread siblings comm value. The comm value
is limited in size compared to the cmdline value, so writing anything longer
then the kernel's TASK_COMM_LEN (currently 16 chars) will result in a truncated
comm value.
```