



TASK: VMSTATで得られる値 IOWAITの出身を探索せよ

上川純一

Linuxカーネル初心者



vmstat

- システムの負荷の概要をおしえてくれるツール
- ディスクIOでつらいのか、CPU負荷がつらいのか、メモリ負荷がつらいのか大まかな部分を教えてくれる
- 疑問：さて、このツールででてくる「cpu wa」っていったい何の数字を教えてくれているの？

```
dancer@atoron: /home/dancer/shared/2.6.11
ファイル(F) 編集(E) 表示(V) 端末(T) タブ(T) ヘルプ(H)
dancer@atoron: ... dancer@atoron: ... dancer@atoron: ... dancer@atoron: ... dancer@atoron: ...
procs -----memory----- --swap-- --io-- --system-- ----cpu----
r b swpd free buff cache si so bi bo in cs us sy id wa
7 1 7856 6228 4116 103600 0 0 12 1132 1752 1773 62 7 0 31
4 0 7856 6608 4076 103396 0 0 84 1152 1774 1586 53 6 0 41
1 2 7856 6292 4104 103708 0 0 180 820 1642 1007 40 4 0 56
0 3 7852 5796 4124 103964 0 0 108 1176 1658 921 49 6 0 46
1 2 7848 6572 4152 103480 96 12 272 780 1559 1140 41 7 0 52
4 1 7848 5996 4160 103592 416 0 428 240 1625 715 96 4 0 0
3 2 7848 5792 4176 103876 32 0 108 2024 1889 890 56 5 0 39
0 2 7844 6260 4188 103028 32 0 128 1120 1780 1102 59 6 0 35
0 2 7840 5940 4256 103304 0 0 156 1420 1747 1068 51 7 0 42
1 2 7840 6004 4292 103540 0 0 80 1132 1862 996 48 6 0 45
4 0 7836 6468 4328 102996 0 0 48 1244 1885 1019 50 6 0 44
3 1 7828 6148 4360 103116 0 0 20 1536 1824 855 68 6 0 26
0 1 7828 6148 4360 103116 0 0 20 1536 1824 855 68 6 0 26
```

2005年3月26日カー





ユーザランドの追求(1) : vmstatコマンドはどこだ

- Debianでの調査方法
 - dpkg -S コマンドで検索
 - auto-apt searchコマンドで検索
 - どうやらprocpsパッケージにはいっているらしい

```
$ auto-apt search bin/vmstat
usr/bin/vmstat base/procps
$ dpkg -S bin/vmstat
procps: /usr/bin/vmstat
```



ユーザランドの追求(2) : procps ソースを探求

- パッケージ名がわかったらソースコードを取得
 - apt-get source procps
 - このパッケージはパッチ管理システムを使っているみたいなので、debian/rules patch

```
$ apt-get source procps
Reading Package Lists... Done
Building Dependency Tree... Done
Need to get 303kB of source archives.
Get:1 http://ring.asahi-net.or.jp unstable/main procps 1:3.2.5-1 (dsc) [616B]
Get:2 http://ring.asahi-net.or.jp unstable/main procps 1:3.2.5-1 (tar) [277kB]
Get:3 http://ring.asahi-net.or.jp unstable/main procps 1:3.2.5-1 (diff) [25.0kB]
Fetched 303kB in 0s (1261kB/s)
dpkg-source: extracting procps in procps-3.2.5
$ cd procps-3.2.5/
$ debian/rules patch
test -d debian/patched || install -d debian/patched
dpatch apply-all
applying patch 10_sysctl.8 to ./ ... ok.
applying patch 20_kill.1 to ./ ... ok.
applying patch 20_top_manpage to ./ ... ok.
applying patch 30_library_map_freeproc to ./ ... ok.
applying patch 30_pgrep_start_time to ./ ... ok.
applying patch 30_readproc_c to ./ ... ok.
applying patch 30_tload_no_optargs to ./ ... ok.
dpatch cat-all >>patch-stampT
mv -f patch-stampT patch-stamp
```



脱線： GONZUIを導入

- WEBサーバになっていてWEB
インタフェースがある
 - ./gonzui-server
- ソースコードを検索するツール
 - ./gonzui-import -apt procps
 - パッチをapplyしてくれない
のでパッチを書いていた
 - ./gonzui-import procps*.dsc
 - なんかできないのでパッチ
を書いていた



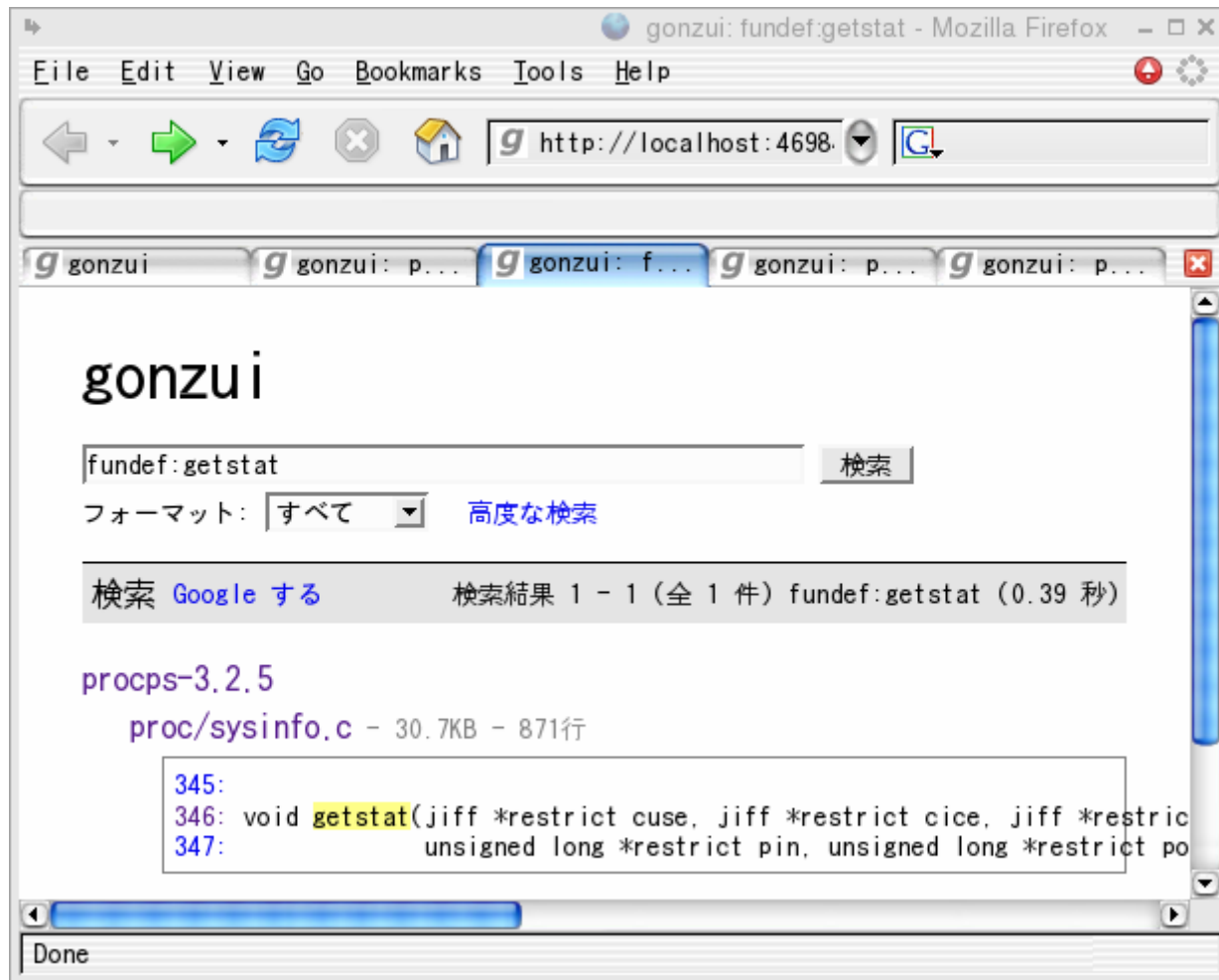


ユーザランドの追求(3) vmstat.cの該当部分を見つける

```
187:
188:     sleep_half=(sleep_time/2);
189:     new_header();
190:     meminfo();
191:
192:     getstat(cpu_use,cpu_nic,cpu_sys,cpu_idl,cpu_iow,cpu_xxx,cpu_yyy,
193:             ppggin,pgpgout,pswpin,pswpout,
194:             intr,ctxt,
195:             &running,&blocked,
196:             &dummy_1, &dummy_2);
197:
198:     duse= *cpu_use + *cpu_nic;
199:     dsys= *cpu_sys + *cpu_xxx + *cpu_yyy;
200:     didl= *cpu_idl;
201:     diow= *cpu_iow;
202:     Div= duse+dsys+didl+diow;
203:     divo2= Div/2UL;
204:     printf(format,
205:            running, blocked,
```



ユーザランドの追求(4) : getstatの関数定義元を探す





ユーザランドの追求(5) :

getstatは/proc/statを見ているだけ
だという事実に驚愕する

```
346: void getstat(jiff *restrict cuse, jiff *restrict cice, jiff *restrict csys)
347:             unsigned long *restrict pin, unsigned long *restrict pout, u
348:             unsigned *restrict intr, unsigned *restrict cxtx,
349:             unsigned int *restrict running, unsigned int *restrict block
350:             unsigned int *restrict btime, unsigned int *restrict process
351:     static int fd;
352:     unsigned long long llbuf = 0;
353:     int need_vmstat_file = 0;
354:     int need_proc_scan = 0;
355:     const char* b;
356:     buff[BUFSIZE-1] = 0; /* ensure null termination in buffer */
357:
358:     if(fd){
359:         lseek(fd, 0L, SEEK_SET);
360:     }else{
361:         fd = open("/proc/stat", O_RDONLY, 0);
362:         if(fd == -1) crash("/proc/stat");
363:     }
364:     read(fd, buff, BUFSIZE-1);
365:     *intr = 0;
```




ユーザランドの追求： vmstatのソースコードの結論

- どうやらvmstatはvmstat.c から生成されているらしい
- そこでは、値を得るためにgetstat()という関数を活用しているようだ
- proc/sysstat.cでgetstat()関数が定義されているようだ
- そこを見ると、getstat ()関数は/proc/statの内容を見ているだけのようだ
 - カーネルソース読め



カーネル側の追求： /proc/statはどこで提供しているのか

- 過去の経験でfs/proc/proc_misc.cあたりだろうと見る
- proc/statファイルを開くとproc/stat ファイルはproc_misc_init()のcreate_seq_entry("stat", 0, &proc_stat_operations);で作成されているようだ
- proc_stat_operations構造体で指定してあるstat_open()関数がどうやら呼ばれるらしい
- stat_openはshow_stat()を呼び出し、そこでcpustat情報を返しているらしい

```
319: },
320:
321: static int show_stat(struct seq_file *p, void *v)
322: {
323:     int i;
324:     unsigned long jif;
```



カーネル側の追求： cpustatとは何者か

- include/linux/kernel_stat.h: kstat_cpu = per_cpu で、kernel_stat構造体がCPU毎にあるらしい。
- struct cpu_usage_stat cpustat; を参照しているらしい。
 - kstat_cpu().cpustat.user/nice/などを更新しているのはどこだろう。

The screenshot shows a Mozilla Firefox browser window with the following content:

- Address bar: gonzui: path:linux-2.6.11/kernel/sched.c cpustat - Mozilla Firefox
- Search bar: path: linux-2.6.11/kernel/sched.c cpustat
- Search results: 検索結果 1 - 1 (全 1 件) cpustat from linux-2.6.11/kernel/sched.c (0.04 秒)
- Code snippet from kernel/sched.c:

```
2330: {
2331:     struct cpu_usage_stat *cpustat = &kstat_this_cpu.cpustat;
2332:     cputime64_t tmp;
2340:
2341:     /* Add user time to cpustat. */
2342:     tmp = cputime_to_cputime64(cputime);
2343:     if (TASK_NICE(p) > 0)
```





カーネル側の追求： cpustatの更新している場所

- kernel/sched.cのscheduler_tickがそれっぽい
 - パラメータのuser_ticksとsys_ticksが与えられている。
 - kernel/timer.c: update_process_timesから、user_ticksかsys_ticksのどちらかが1になるように呼び出されている。forkからは両方0で呼び出されている
- ロジックを見ると現在のプロセスがidleの場合にIO待ちの状態であるプロセスが一つでもあれば (nr_iowait>0) iowait の値が増加するらしい、そうでないならidleとみなす
- nr_iowaitはio_schedule_timeout/io_scheduleでのみ変更されているようだ



カーネル側の追求： io_scheduleを呼ぶ場合って？

- gonzuiで検索
 - どうやらio_scheduleはファイル名から眺めるとFS関係、ディスク書き込み関連っぽいところでのみ利用されているようだ
 - io_schedule_timeoutはblk_congestion_wait(int rw, long timeout)のみから呼ばれている
- どうやらディスクアクセスを中心に呼ばれており、ネットワーク関連では呼ばれないようだ

```
drivers/md/dm-io.c - 17.8KB - 631行
540:
541:     io_schedule();
542: }

drivers/md/dm.c - 29.2KB - 1,177行
1078:
1079:     io_schedule();
1080: }

fs/buffer.c - 101.5KB - 3,096行
64:     blk_run_address_space(bd->bd_inode->i_mapping);
65:     io_schedule();
66:     return 0;

fs/direct-io.c - 46.0KB - 1,255行
371:     blk_run_address_space(dio->inode->i_mapping);
372:     io_schedule();
373:     spin_lock_irqsave(&dio->bio_lock, flags);

direct-io.c 内の検索結果
include/linux/sched.h - 41.8KB - 1,219行
168:
169: void io_schedule(void);
170: long io_schedule_timeout(long timeout);

kernel/sched.c - 158.7KB - 5,053行
```



念のためにネットワーク関連の待ち では呼ばれていないことを確認

- なんとなくsys_selectあたりを調べてみる
- sys_selectはdo_selectを呼ぶだけのようだ
- fs/select.cのdo_selectは待ちのためにschedule_timeoutを呼んでいる

The screenshot shows two browser windows. The left window is at 'http://localhost:46984/advsearch/' and shows the search interface for 'gonzui'. The search term 'sys_select' is entered, and 10 items are listed. The right window is at 'http://localhost:46984/search?q=' and shows the search results for 'fundef:sys_select'. The results show a file 'fs/select.c' with 532 lines, and a code snippet for 'sys_select' function definition.

```
293: asmlinkage long
294: sys_select(int n, fd_set __user *inp, fd_set __user *outp, fd_set __user *exp, str
295: {
```



まとめ

- Debian GNU/Linuxでアプリケーションからカーネルの部分まで追跡する方法について説明しました
- 追跡にgonzuiを活用してみました
- ディスク関連の操作で"cpu wa"の値が増加するらしいということがわかりました。ネットワークの通信ではどうやら増加しないっぽい雰囲気です。