Useful links:
Freecode
IceWalkers

The Linux Foundation
Linux Kernel Archives
Linux Kernel Mailing List
The Linux Documentation Project

Sysstat tutorial

You will find here a tutorial describing a few use cases for some sysstat commands. The first section below concerns the **sa** commands. The second one concerns the **pidstat** command. Of course, you should *really* have a look at the manual pages features and how these commands can help you to monitor your system (follow the Documentation link above for that).

1. Section 1: Using sar and sadf
2. Section 2: Using pidstat

Section 1: Using sar and sadf

sar is the system activity reporter. By interpreting the reports that sar produces, you can locate system bottlenecks and sugg possible solutions to those annoying performance problems.
The Linux kernel maintains internal counters that keep track of requests, completion times, I/O block counts, etc. From this a information, sar calculates rates and ratios that give insight into where the bottlenecks are.

The key to understanding sar is that it reports on system activity over a period of time. You must take care to collect sar data appropriate time (not at lunch time or on weekends, for example). Here is one way to invoke sar:

```
$ sar -u -o datafile 2 3
```

The -u option specifies our interest in the CPU subsystem. The -o option will create an output file that contains binary data. F take 3 samples at two-second intervals. Upon completion of the sampling, sar will report the results to the screen. This provi snapshot of current system activity.
The above example uses sar in interactive mode. You can also invoke sar from cron. In this case, cron would run the /usr/lib script and create a daily log file. The /usr/lib/sa/sa2 shell script is run to format the log into human-readable form. These scrip invoked by a crontab run by root (although I prefer to use adm). Here is the crontab, located in /etc/cron.d directory and using syntax, that makes this happen:

```
# Run system activity accounting tool every 10 minutes
*/10 * * * * root /usr/lib/sa/sa1 -d 1 1
# 0 * * * * root /usr/lib/sa/sa1 -d 600 6 &
# Generate a daily summary of process accounting at 23:53
53 23 * * * root /usr/lib/sa/sa2 -A
```

In reality, the sa1 script initiates a related utility called sadc. sa1 gives sadc several arguments to specify the amount of time samples, the number of samples, and the name of a file into which the binary results should be written.
A new file is created each day so that we can easily interpret daily results. The sa2 script calls sar, which formats the binary human-readable form.

Let's think of our system as being composed of three interdependant subsystems: CPU, disk and memory. Our goal is to find subsystem is responsible for any performance bottleneck. By analyzing sar's output, we can achieve that goal.
Listing below represents the report produced by initiating the sar -u command. Initiating sar in this manner produces a report log file produced by sadc.

```
Linux 2.6.8.1-27mdkcustom (localhost)   03/29/2006

09:00:00 PM       CPU     %user     %nice   %system   %iowait    %steal     %idle
09:10:00 PM       all     96.18      0.00      0.42      0.00      0.00      3.40
09:20:00 PM       all     97.99      0.00      0.36      0.00      0.00      1.65
09:30:00 PM       all     97.59      0.00      0.38      0.00      0.00      2.03
...
```

The %user and %system columns simply specify the amount of time the CPU spends in user and system mode. The %iowai columns are of interest to us when doing performance analysis. The %iowait column specifies the amount of time the CPU s for I/O requests to complete. The %idle column tells us how much useful work the CPU is doing. A %idle time near zero indi bottleneck, while a high %iowait value indicates unsatisfactory disk performance.
Additional information can be obtained by the sar -q command, which displays the run queue length, total number of process load averages for the past one, five and fifteen minutes:

```
Linux 2.6.8.1-27mdkcustom (localhost)   03/29/2006

09:00:00 PM   runq-sz  plist-sz   ldavg-1   ldavg-5  ldavg-15
09:10:00 PM         2       121      2.22      2.17      1.45
09:20:00 PM         6       137      2.79      2.48      1.73
09:30:00 PM         5       129      3.31      2.83      1.95
...
```

This example shows that the system is busy (since more than one process is runnable at any given time) and rather overloa sar also lets you monitor memory utilization. Have a look at the following example produced by sar -r:

```
Linux 2.6.8.1-27mdkcustom (localhost)   03/29/2006

09:00:00 PM kbmemfree kbmemused  %memused kbbuffers  kbcached kbswpfree kbswpused  %swpused  kbswpcad
09:10:00 PM    591468    444388     42.90     19292    227412   1632920         0      0.00         0
09:20:00 PM    546860    488996     47.21     21844    243900   1632920         0      0.00         0
09:30:00 PM    538268    497588     48.04     25308    267228   1632920         0      0.00         0
...
```

This listing shows that the system has plenty of free memory. Swap space is not used. So memory is not a problem here. Yo
check this by using sar -W to get swapping statistics:

```
Linux 2.6.8.1-27mdkcustom (localhost)   03/29/2006

09:00:00 PM  pswpin/s pswpout/s
09:10:00 PM      0.00      0.00
09:20:00 PM      0.00      0.00
09:30:00 PM      0.00      0.00
...
```

sar can also help you to monitor disk activity. sar -b displays I/O and transfer rate statistics grouped for all block devices:

```
Linux 2.6.8.1-27mdkcustom (localhost)   03/29/2006

09:00:00 PM       tps      rtps      wtps   bread/s   bwrtn/s
09:10:00 PM      6.37      2.32      4.05    126.84     61.41
09:20:00 PM      4.03      0.74      3.29     54.49     46.04
09:30:00 PM      6.71      3.11      3.59     80.13     49.18
...
```

sar -d enables you to get more detailed information on a per device basis. It displays statistics data similar to those displayed

```
Linux 2.6.8.1-27mdkcustom  (localhost)   03/29/2006

09:00:00 AM       DEV       tps  rd_sec/s  wr_sec/s  avgrq-sz  avgqu-sz     await     svctm     %util
09:10:00 AM       sda      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
09:10:00 AM       sdb     18.09      0.00    160.80      8.89      0.01      0.67      0.19      0.35
09:20:00 AM       sda      2.51      0.00     52.26     20.80      0.00      0.60      0.40      0.10
09:20:00 AM       sdb     18.91      0.00    141.29      7.47      0.02      0.92      0.21      0.40
09:30:00 AM       sda     26.87     11.94    291.54     11.30      0.12      4.33      1.07      2.89
09:30:00 AM       sdb      7.00      0.00     54.00      7.71      0.00      0.50      0.14      0.10
...
```

sar has numerous other options that enable you to gather statistics for every part of your system. You will find useful informat
in the manual page.
OK. As a last example, let's show how the sadf command can help us to produce some graphs.
We use the command sar -B to display paging statistics from daily data file sa29 (see example below).

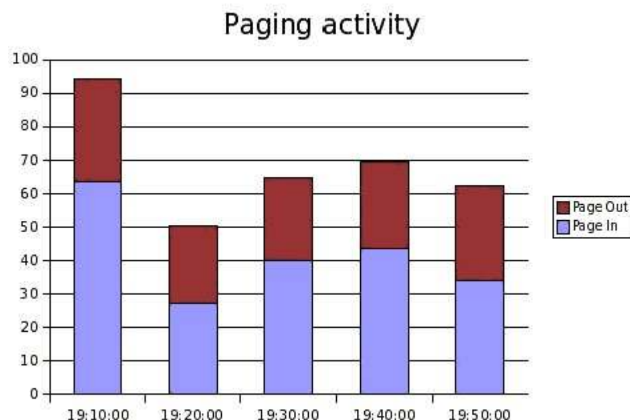
```
# sar -B -f /var/log/sa/sa29
Linux 2.6.8.1-27mdkcustom (localhost)   03/29/2006

09:00:00 PM  pgpgin/s pgpgout/s   fault/s  majflt/s
09:10:00 PM     63.42     30.71    267.35      0.45
09:20:00 PM     27.25     23.02    281.88      0.26
09:30:00 PM     40.06     24.59    246.51      0.32
09:40:00 PM     43.58     26.11    265.25      0.34
09:50:00 PM     34.12     28.38    271.54      0.37
Average:        41.69     26.56    266.51      0.35
```

sadf -d extracts data in a format that can be easily ingested by a relational database:

```
# sadf -d /var/log/sa/sa29 -- -B
localhost;601;2006-03-29 19:10:00 UTC;63.42;30.71;267.35;0.45
localhost;600;2006-03-29 19:20:00 UTC;27.25;23.02;281.88;0.26
localhost;600;2006-03-29 19:30:00 UTC;40.06;24.59;246.51;0.32
localhost;600;2006-03-29 19:40:00 UTC;43.58;26.11;265.25;0.34
localhost;600;2006-03-29 19:50:00 UTC;34.12;28.38;271.54;0.37
```

If we saw this as a text file, both Excel and Open Office will allow us to specify a semicolon as a field delimiter. Then we can
performance report and graph.



Section 2: Using pidstat

The **pidstat** command is used to monitor processes and threads currently being managed by the Linux kernel. It can also mo
children of those processes and threads.

With its -d option, pidstat can report I/O statistics, providing that you have a recent Linux kernel (2.6.20+) with the option CONFIG_TASK_IO_ACCOUNTING compiled in. So imagine that your system is undergoing heavy I/O and you want to kno
are generating them. You could then enter the following command:

```
$ pidstat -d 2
Linux 2.6.20 (localhost)    09/26/2007

10:13:31 AM       PID   kB_rd/s   kB_wr/s kB_ccwr/s  Command
10:13:31 AM     15625      1.98  16164.36      0.00  dd

10:13:33 AM       PID   kB_rd/s   kB_wr/s kB_ccwr/s  Command
10:13:33 AM     15625      4.00  20556.00      0.00  dd

10:13:35 AM       PID   kB_rd/s   kB_wr/s kB_ccwr/s  Command
10:13:35 AM     15625      0.00  10642.00      0.00  dd
...
```

This report tells us that there is only one task (a "dd" command with PID 15625) which is responsible for these I/O.

When no PID's are explicitly selected on the command line (as in the case above), the pidstat command examines all the tas
the system but displays only those whose statistics are varying during the interval of time. But you can also indicate which ta
monitor. The following example reports CPU statistics for PID 8197 and all its threads:

```
$ pidstat -t -p 8197 1 3
Linux 2.6.8.1-27mdkcustom (localhost)    09/26/2007

10:40:05 AM       PID       TID   %user %system    %CPU   CPU  Command
10:40:06 AM      8197         -   71.29     1.98   73.27     0  procthread
10:40:06 AM         -      8197   71.29     1.98   73.27     0  |__procthread
10:40:06 AM         -      8198    0.00     0.99    0.99     0  |__procthread

10:40:06 AM       PID       TID   %user %system    %CPU   CPU  Command
10:40:07 AM      8197         -   67.00     2.00   69.00     0  procthread
10:40:07 AM         -      8197   67.00     2.00   69.00     0  |__procthread
10:40:07 AM         -      8198    1.00     1.00    2.00     0  |__procthread

10:40:07 AM       PID       TID   %user %system    %CPU   CPU  Command
10:40:08 AM      8197         -   56.00     6.00   62.00     0  procthread
10:40:08 AM         -      8197   56.00     6.00   62.00     0  |__procthread
10:40:08 AM         -      8198    2.00     1.00    3.00     0  |__procthread

Average:          PID       TID   %user %system    %CPU   CPU  Command
Average:         8197         -   64.78     3.32   68.11     -  procthread
Average:            -      8197   64.78     3.32   68.11     -  |__procthread
Average:            -      8198    1.00     1.00    1.99     -  |__procthread
```

As a last example, let me show you how pidstat helped me to detect a memory leak in the pidstat command itself. At that tim
the very first version of pidstat I wrote for sysstat 7.1.4 and fixing the last remaining bugs. Here is the command I entered on t
line and the output I got:

```
$ pidstat -r 2
Linux 2.6.8.1-27mdkcustom (localhost)    09/26/2007

10:59:03 AM       PID  minflt/s  majflt/s     VSZ    RSS   %MEM  Command
10:59:05 AM     14364    113.66      0.00    2480   1540   0.15  pidstat

10:59:05 AM       PID  minflt/s  majflt/s     VSZ    RSS   %MEM  Command
10:59:07 AM      7954    150.00      0.00   27416  19448   1.88  net_applet
10:59:07 AM     14364    120.00      0.00    3048   2052   0.20  pidstat

10:59:07 AM       PID  minflt/s  majflt/s     VSZ    RSS   %MEM  Command
10:59:09 AM     14364    116.00      0.00    3488   2532   0.24  pidstat

10:59:09 AM       PID  minflt/s  majflt/s     VSZ    RSS   %MEM  Command
10:59:11 AM      7947      0.50      0.00   27044  18356   1.77  mdkapplet
10:59:11 AM     14364    116.00      0.00    3928   3012   0.29  pidstat

10:59:11 AM       PID  minflt/s  majflt/s     VSZ    RSS   %MEM  Command
10:59:13 AM      7954    155.50      0.00   27416  19448   1.88  net_applet
10:59:13 AM     14364    115.50      0.00    4496   3488   0.34  pidstat
...
```

I noticed that pidstat had a memory footprint (VSZ and RSS fields) that was constantly increasing as the time went by. I quicl
had forgotten to close a file descriptor in a function of my code and that was responsible for the memory leak...!