SUSE Linux Enterprise Server

11 SP1

www.novell.com

May 10, 2010

System Analysis and Tuning Guide

SUSE
A NOVELL BUSINESS

System Analysis and Tuning Guide

All information found in this book has been compiled with utmost attention to detail. However, this does not guarantee complete accuracy. Neither Novell, Inc., SUSE LINUX Products GmbH, the authors, nor the translators shall be held liable for possible errors or the consequences thereof.

Contents

About This Guide

SUSE Linux Enterprise Server is used for a broad range of usage scenarios in enterprise and scientific data centers. Novell has ensured SUSE Linux Enterprise Server is set up in a way that it accommodates different operation purposes with optimal performance. However, SUSE Linux Enterprise Server must meet very different demands when employed on a number crunching server compared to a file server, for example. The great versatility of Linux makes it possible to configure it in a way that it brings out the best in each usage scenario.

This manual introduces you to means to monitor and analyze your system. It describes methods to manage system resources and to tune your system. This guide does *not* offer recipes for special scenarios, because each server has got its own different demands. It rather enables you to thoroughly analyze your servers and make the most out of them.

General Notes on System Tuning
: Tuning a system requires a carefully planned proceeding. Learn which steps are necessary to successfully improve your system.

Part II, “System Monitoring” (page 7)
: Linux offers a large variety of tools to monitor almost every aspect of the system. Learn how to use these utilities and how to read and analyze the system log files.

Part III, “Kernel Monitoring” (page 65)
: The Linux kernel itself offers means to examine every nut, bolt and screw of the system. This part introduces you to SystemTap, a scripting language for writing kernel modules that can be used to analyze and filter data. Collect debugging information and find bottlenecks by using kernel probes and use perfmon2 to access the CPU's performance monitoring unit. Last, monitor applications with the help of Oprofile.

Part IV, “Resource Management” (page 105)
: Learn how to set up a tailor-made system fitting exactly the server's need. Get to know how to use power management while at the same time keeping the performance of a system at a level that matches the current requirements.

Part V, "Kernel Tuning" (page 139)
: The Linux kernel can be optimized either by using sysctl or via the `/proc` file system. This part describes how to optimize the kernel's task handling and storage handling and how to tune I/O and network performance.

Part VI, "Handling System Dumps" (page 161)
: This part enables you to analyze and handle application or system crashes. It introduces tracing tools such as strace or ltrace and describes how to handle system crashes using Kexec and Kdump.

TIP: Getting the SUSE Linux Enterprise SDK

Some programs or packages mentioned in this guide are only available from the SUSE Linux Enterprise SDK. The SDK is an add-on product for SUSE Linux Enterprise Server and is available for download from `http://developer.novell.com/wiki/index.php/SUSE_LINUX_SDK`.

Many chapters in this manual contain links to additional documentation resources. This includes additional documentation that is available on the system as well as documentation available on the Internet.

For an overview of the documentation available for your product and the latest documentation updates, refer to `http://www.novell.com/documentation` or to the following section:

1 Available Documentation

We provide HTML and PDF versions of our books in different languages. The following manuals for users and administrators are available on this product:

Deployment Guide (↑*Deployment Guide*)
: Shows how to install single or multiple systems and how to exploit the product inherent capabilities for a deployment infrastructure. Choose from various approaches, ranging from a local installation or a network installation server to a mass deployment using a remote-controlled, highly-customized, and automated installation technique.

Administration Guide (↑*Administration Guide*)
Covers system administration tasks like maintaining, monitoring and customizing an initially installed system.

Security Guide (↑*Security Guide*)
Introduces basic concepts of system security, covering both local and network security aspects. Shows how to make use of the product inherent security software like Novell AppArmor (which lets you specify per program which files the program may read, write, and execute) or the auditing system that reliably collects information about any security-relevant events.

System Analysis and Tuning Guide (page 1)
An administrator's guide for problem detection, resolution and optimization. Find how to inspect and optimize your system by means of monitoring tools and how to efficiently manage resources. Also contains an overview of common problems and solutions and of additional help and documentation resources.

Virtualization Guide (↑*Virtualization Guide*)
Offers an introduction to virtualization technology of your product. It features an overview of the various fields of application and installation types of each of the platforms supported by SUSE Linux Enterprise Server as well as a short description of the installation procedure.

Storage Administration Guide
Provides information about how to manage storage devices on a SUSE Linux Enterprise Server.

In addition to the comprehensive manuals, several quick start guides are available:

Installation Quick Start (↑*Installation Quick Start*)
Lists the system requirements and guides you step-by-step through the installation of SUSE Linux Enterprise Server from DVD, or from an ISO image.

Linux Audit Quick Start
Gives a short overview how to enable and configure the auditing system and how to execute key tasks such as setting up audit rules, generating reports, and analyzing the log files.

Novell AppArmor Quick Start
Helps you understand the main concepts behind Novell® AppArmor.

Find HTML versions of most product manuals in your installed system under `/usr/share/doc/manual` or in the help centers of your desktop. Find the latest documentation updates at http://www.novell.com/documentation where you can download PDF or HTML versions of the manuals for your product.

2 Feedback

Several feedback channels are available:

Bugs and Enhancement Requests

> For services and support options available for your product, refer to http://www.novell.com/services/.
>
> To report bugs for a product component, please use http://support.novell.com/additional/bugreport.html.
>
> Submit enhancement requests at https://secure-www.novell.com/rms/rmsTool?action=ReqActions.viewAddPage&return=www.

User Comments

> We want to hear your comments and suggestions about this manual and the other documentation included with this product. Use the User Comments feature at the bottom of each page in the online documentation or go to http://www.novell.com/documentation/feedback.html and enter your comments there.

3 Documentation Conventions

The following typographical conventions are used in this manual:

- `/etc/passwd`: directory names and filenames
- *`placeholder`*: replace *`placeholder`* with the actual value
- `PATH`: the environment variable PATH
- `ls`, `--help`: commands, options, and parameters

- `user`: users or groups

- Alt, Alt + F1: a key to press or a key combination; keys are shown in uppercase as on a keyboard

- *File*, *File* > *Save As*: menu items, buttons

- ► **amd64 em64t ipf:** This paragraph is only relevant for the specified architectures. The arrows mark the beginning and the end of the text block. ◄

  ► **ipseries zseries:** This paragraph is only relevant for the specified architectures. The arrows mark the beginning and the end of the text block. ◄

- *Dancing Penguins* (Chapter *Penguins*, ↑Another Manual): This is a reference to a chapter in another manual.

Part I. Basics

General Notes on System Tuning

1

This manual discusses how to find the reasons for performance problems and provides means to solve these problems. Before you start tuning your system, you should make sure you have ruled out common problems and have found the cause (bottleneck) for the problem. You should also have a detailed plan on how to tune the system, because applying random tuning tips will not help (and could make things worse).

Procedure 1.1 *General Approach When Tuning a System*

1. Be sure what problem to solve

2. Rule out common problems

3. Find the bottleneck

   3a Monitor the system and/or application

   3b Analyze the data

4. Step-by-step tuning

1.1 Be Sure What Problem to Solve

Before you start tuning your system, try to describe the problem as exactly as possible. Obviously, a simple and general “The system is too slow!” is no helpful problem description. If you plan to tune your Web server for faster delivery of static pages, for example, it makes a difference whether you need to generally improve the speed or whether it only needs to be improved at peak times.

Furthermore, make sure you can apply a measurement to your problem, otherwise you will not be able to control if the tuning was a success or not. You should always be able to compare “before” and “after”.

1.2 Rule Out Common Problems

A performance problem often is caused by network or hardware problems, bugs, or configuration issues. Make sure to rule out problems such as the ones listed below before attempting to tune your system:

- Check `/var/log/warn` and `/var/log/messages` for unusual entries.

- Check (using `top` or `ps`) whether a certain process misbehaves by eating up unusual amounts of CPU time or memory.

- Check for network problems by inspecting `/proc/net/dev`.

- In case of I/O problems with physical disks, make sure it is not caused by hardware problems (check the disk with the `smartmontools`) or by a full disk.

- Ensure that background jobs are scheduled to be carried out in times the server load is low. Those jobs should also run with low priority (set via `nice`).

- If the machine runs several services using the same resources, consider moving services to another server.

- Last, make sure your software is up-to-date.

1.3 Finding the Bottleneck

Finding the bottleneck very often is the hardest part when tuning a system. SUSE Linux Enterprise Server offers a lot of tools helping you with this task. See Part II, “System Monitoring” (page 7) for detailed information on general system monitoring applications and log file analysis. If the problem requires a long-time in-depth analysis, the Linux kernel offers means to perform such analysis. See Part III, “Kernel Monitoring” (page 65) for coverage.

Once you have collected the data, it needs to be analyzed. First, inspect if the server's hardware (memory, CPU, bus) and its I/O capacities (disk, network) are sufficient. If these basic conditions are met, the system might benefit from tuning.

1.4 Step-by-step Tuning

Make sure to carefully plan the tuning itself. It is of vital importance to only do one step at a time. Only by doing so you will be able to measure if the change provided an improvement or even had a negative impact. Each tuning activity should be measured over a sufficient time period in order to ensure you can do an analysis based on significant data. If you cannot measure a positive effect, do not make the change permanent. Chances are, that it might have a negative effect in the future.

Part II. System Monitoring

2 System Monitoring Utilities

There are number of programs, tools, and utilities which you can use to examine the status of your system. This chapter introduces some of them and describes their most important and frequently used parameters.

For each of the described commands, examples of the relevant outputs are presented. In the examples, the first line is the command itself (after the > or # sign prompt). Omissions are indicated with square brackets (`[...]`) and long lines are wrapped where necessary. Line breaks for long lines are indicated by a backslash (\).

```
# command -x -y
output line 1
output line 2
output line 3 is annoyingly long, so long that \
    we have to break it
output line 3
[...]
output line 98
output line 99
```

The descriptions have been kept short so that we can include as many utilities as possible. Further information for all the commands can be found in the manual pages. Most of the commands also understand the parameter `--help`, which produces a brief list of possible parameters.

2.1 Multi-Purpose Tools

While most of the Linux system monitoring tools are specific to monitor a certain aspect of the system, there are a few “swiss army knife” tools showing various aspects of the system at a glance. Use these tools first in order to get an overview and find out which part of the system to examine further.

2.1.1 vmstat

vmstat collects information about processes, memory, I/O, interrupts and CPU. If called without a sampling rate, it displays average values since the last reboot. When called with a sampling rate, it displays actual samples:

Example 2.1 *vmstat Output on a Lightly Used Machine*

```
tux@mercury:~> vmstat -a 2
procs -----------memory---------- ---swap-- -----io---- -system-- -----cpu-------
 r  b   swpd   free  inact active   si   so    bi    bo   in   cs us sy  id wa st
 0  0      0 750992 570648 548848    0    0     0     1    8    9  0  0 100  0  0
 0  0      0 750984 570648 548912    0    0     0     0   63   48  1  0 99   0  0
 0  0      0 751000 570648 548912    0    0     0     0   55   47  0  0 100  0  0
 0  0      0 751000 570648 548912    0    0     0     0   56   50  0  0 100  0  0
 0  0      0 751016 570648 548944    0    0     0     0   57   50  0  0 100  0  0
```

Example 2.2 *vmstat Output on a Heavily Used Machine (CPU bound)*

```
tux@mercury:~> vmstat 2
procs -----------memory----------- ---swap-- -----io---- -system-- -----cpu------
 r  b   swpd   free   buff   cache   si   so    bi    bo   in   cs us sy id wa st
32  1  26236 459640 110240 6312648    0    0  9944     2 4552 6597 95  5  0  0  0
23  1  26236 396728 110336 6136224    0    0  9588     0 4468 6273 94  6  0  0  0
35  0  26236 554920 110508 6166508    0    0  7684 27992 4474 4700 95  5  0  0  0
28  0  26236 518184 110516 6039996    0    0 10830     4 4446 4670 94  6  0  0  0
21  5  26236 716468 110684 6074872    0    0  8734 20534 4512 4061 96  4  0  0  0
```

TIP

The first line of the vmstat output always displays average values since the last reboot.

The columns show the following:

r

Shows the amount of processes in the run queue. These processes are waiting for a free CPU slot to be executed. If the number of processes in this column is constantly higher than the number of CPUs available, this is an indication for insufficient CPU power.

b

Shows the amount of processes waiting for a resource other than a CPU. A high number in this column may indicate an I/O problem (network or disk).

swpd

The amount of swap space currently used.

free

The amount of unused memory.

inact

Recently unused memory that can be reclaimed. This column is only visible when calling `vmstat` with the parameter `-a` (recommended).

active

Recently used memory that normally does not get reclaimed. This column is only visible when calling `vmstat` with the parameter `-a` (recommended).

buff

File buffer cache in RAM. This column is not visible when calling `vmstat` with the parameter `-a` (recommended).

cache

Page cache in RAM. This column is not visible when calling `vmstat` with the parameter `-a` (recommended).

si

Amount of data that is moved from RAM to swap per second. High values over a longer period of time in this column are an indication that the machine would benefit from more RAM.

so

Amount of data that is moved from swap to RAM per second. High values over a longer period of time in this column are an indication that the machine would benefit from more RAM.

bi

Number of blocks per second received from a block device (e.g. a disk read). Note that swapping also impacts the values shown here.

bo

Number of blocks per second sent to a block device (e.g. a disk write). Note that swapping also impacts the values shown here.

in

Interrupts per second. A high value indicates a high I/O level (network and/or disk).

cs

Number of context switches per second. Simplified this means that the kernel has to replace executable code of one program in memory with that of another program.

us

Percentage of CPU usage from user processes.

sy

Percentage of CPU usage from system processes.

id

Percentage of CPU time spent idling. If this value is zero over a longer period of time, your CPU(s) are working to full capacity. This is not necessarily a bad sign—rather refer to the values in columns *r* and *b* to determine if your machine is equipped with sufficient CPU power.

wa

Time spent waiting for IO. If this value is significantly higher than zero over a longer period of time, there is a bottleneck in the I/O system (network or hard disk).

st

Percentage of CPU time used by virtual machines.

See `vmstat --help` for more options.

2.1.2 System Activity Information: sar and sadc

`sar` can generate extensive reports on almost all important system activities, among them CPU, memory, IRQ usage, IO, or networking. It can either generate reports on the fly or query existing reports gathered by the system activity data collector (`sadc`). `sar` and `sadc` both gather all their data from the `/proc` file system.

NOTE: sysstat Package

sar and sadc are part of sysstat package. You need to install the package either with YaST, or with zypper in sysstat.

Automatically Collecting Daily Statistics With sadc

If you want to monitor your system about a longer period of time, use sadc to automatically collect the data. You can read this data at any time using sar. To start sadc, simply run /etc/init.d/boot.sysstat start. This will add a link to /etc/cron.d/ that calls sadc with the following default configuration:

- All available data will be collected.
- Data is written to /var/log/sa/sa*DD*, where *DD* stands for the current day. If a file already exists, it will be archived.
- The summary report is written to /var/log/sa/sar*DD*, where *DD* stands for the current day. Already existing files will be archived.
- Data is collected every ten minutes, a summary report is generated every 6 hours (see /etc/sysstat/sysstat.cron).
- The data is collected by the /usr/lib64/sa/sa1 script (or /usr/lib/sa/sa1 on 32bit systems)
- The summaries are generated by the script /usr/lib64/sa/sa2 (or /usr/lib/sa/sa2 on 32bit systems)

If you need to customize the configuration, copy the sa1 and sa2 scripts and adjust them according to your needs. Replace the link /etc/cron.d/sysstat with a customized copy of /etc/sysstat/sysstat.cron calling your scripts.

Generating reports with sar

To generate reports on the fly, call sar with an interval (seconds) and a count. To generate reports from files specify a filename with the option -f instead of interval and count. If filename, interval and count are not specified, sar attempts to generate a report from `/var/log/sa/saDD`, where *DD* stands for the current day. This is the default location to where `sadc` writes its data. Query multiple files with multiple -f options.

```
sar 2 10                        # on-the-fly report, 10 times every 2 seconds
sar -f ~/reports/sar_2010_05_03  # queries file sar_2010_05_03
sar                              # queries file from today in /var/log/sa/
cd /var/log/sa &&\
sar -f sa01 -f sa02              # queries files /var/log/sa/0[12]
```

Find examples for useful `sar` calls and their interpretation below. For detailed information on the meaning of each column, please refer to the `man (1)` of `sar`. Also refer to the man page for more options and reports—`sar` offers plenty of them.

CPU Utilization Report: sar

When called with no options, `sar` shows a basic report about CPU usage. On multiprocessor machines, results for all CPUs are summarized. Use the option `-P ALL` to also see statistics for individual CPUs.

```
mercury:~ # sar 10 5
Linux 2.6.31.12-0.2-default (mercury) 03/05/10   _x86_64_   (2 CPU)

14:15:43   CPU     %user   %nice   %system   %iowait    %steal     %idle
14:15:53   all     38.55    0.00      6.10      0.10      0.00     55.25
14:16:03   all     12.59    0.00      4.90      0.33      0.00     82.18
14:16:13   all     56.59    0.00      8.16      0.44      0.00     34.81
14:16:23   all     58.45    0.00      3.00      0.00      0.00     38.55
14:16:33   all     86.46    0.00      4.70      0.00      0.00      8.85
Average:   all     49.94    0.00      5.38      0.18      0.00     44.50
```

If the value for *%iowait* (percentage of the CPU being idle while waiting for I/O) is significantly higher than zero over a longer period of time, there is a bottleneck in the I/O system (network or hard disk). If the *%idle* value is zero over a longer period of time, your CPU(s) are working to full capacity.

Memory Usage Report: sar -r

Generate an overall picture of the system memory (RAM) by using the option `-r`:

```
mercury:~ # sar -r 10 5
Linux 2.6.31.12-0.2-default (mercury) 03/05/10   _x86_64_   (2 CPU)

16:12:12 kbmemfree kbmemused %memused kbbuffers kbcached kbcommit %commit
16:12:22    548188   1507488    73.33     20524    64204  2338284   65.10
16:12:32    259320   1796356    87.39     20808    72660  2229080   62.06
```

```
16:12:42    381096   1674580    81.46     21084    75460  2328192   64.82
16:12:52    642668   1413008    68.74     21392    81212  1938820   53.98
16:13:02    311984   1743692    84.82     21712    84040  2212024   61.58
Average:    428651   1627025    79.15     21104    75515  2209280   61.51
```

The last two columns (*kbcommit* and *%commit*) show an approximation of the total amount of memory (RAM plus swap) the current workload would need in the worst case (in kilobyte or percent respectively).

Paging Statistics Report: sar `-B`

Use the option `-B` to display the kernel paging statistics.

```
mercury:~ # sar -B 10 5
Linux 2.6.31.12-0.2-default (mercury) 03/05/10   _x86_64_   (2 CPU)

16:11:43 pgpgin/s pgpgout/s   fault/s majflt/s  pgfree/s pgscank/s pgscand/s pgsteal/s  %vmeff
16:11:53   225.20    104.00  91993.90     0.00  87572.60      0.00      0.00      0.00    0.00
16:12:03   718.32    601.00  82612.01     2.20  99785.69    560.56    839.24   1132.23   80.89
16:12:13  1222.00   1672.40 103126.00     1.70 106529.00   1136.00    982.40   1172.20   55.33
16:12:23   112.18     77.84 113406.59     0.10  97581.24     35.13    127.74    159.38   97.86
16:12:33   817.22     81.28 121312.91     9.41 111442.44      0.00      0.00      0.00    0.00
Average:   618.72    507.20 102494.86     2.68 100578.98    346.24    389.76    492.60   66.93
```

The *majflt/s* (major faults per second) column shows how many pages are loaded from disk (swap) into memory. A large number of major faults slows down the system and is an indication of insufficient main memory. The *%vmeff* column shows the number of pages scanned (*pgscand/s*) in relation to the ones being reused from the main memory cache or the swap cache (*pgsteal/s*). It is a measurement of the efficiency of page reclaim. Healthy values are either near 100 (every inactive page swapped out is being reused) or 0 (no pages have been scanned). The value should not drop below 30.

Block Device Statistics Report: sar `-d`

Use the option `-d` to display the block device (hdd, optical drive, USB storage device, ...). Make sure to use the additional option `-p` (pretty-print) to make the *DEV* column readable.

```
mercury:~ # sar -d -p 10 5
Linux 2.6.31.12-0.2-default (neo)  03/05/10  _x86_64_ (2 CPU)

16:28:31  DEV    tps  rd_sec/s  wr_sec/s  avgrq-sz  avgqu-sz  await  svctm  %util
16:28:41  sdc  11.51     98.50    653.45     65.32      0.10   8.83   4.87   5.61
16:28:41 scd0   0.00      0.00      0.00      0.00      0.00   0.00   0.00   0.00

16:28:41  DEV    tps  rd_sec/s  wr_sec/s  avgrq-sz  avgqu-sz  await  svctm  %util
16:28:51  sdc  15.38    329.27    465.93     51.69      0.10   6.39   4.70   7.23
16:28:51 scd0   0.00      0.00      0.00      0.00      0.00   0.00   0.00   0.00
```

```
16:28:51  DEV    tps rd_sec/s  wr_sec/s  avgrq-sz  avgqu-sz  await  svctm  %util
16:29:01  sdc  32.47   876.72    647.35     46.94      0.33  10.20   3.67  11.91
16:29:01 scd0   0.00     0.00      0.00      0.00      0.00   0.00   0.00   0.00

16:29:01  DEV    tps rd_sec/s  wr_sec/s  avgrq-sz  avgqu-sz  await  svctm  %util
16:29:11  sdc  48.75  2852.45    366.77     66.04      0.82  16.93   4.91  23.94
16:29:11 scd0   0.00     0.00      0.00      0.00      0.00   0.00   0.00   0.00

16:29:11  DEV    tps rd_sec/s  wr_sec/s  avgrq-sz  avgqu-sz  await  svctm  %util
16:29:21  sdc  13.20   362.40    412.00     58.67      0.16  12.03   6.09   8.04
16:29:21 scd0   0.00     0.00      0.00      0.00      0.00   0.00   0.00   0.00

Average:  DEV    tps rd_sec/s  wr_sec/s  avgrq-sz  avgqu-sz  await  svctm  %util
Average:  sdc  24.26   903.52    509.12     58.23      0.30  12.49   4.68  11.34
Average: scd0   0.00     0.00      0.00      0.00      0.00   0.00   0.00   0.00
```

If your machine uses multiple disks, you will receive the best performance, if I/O requests are evenly spread over all disks. Compare the *Average* values for *tps*, *rd_sec/s*, and *wr_sec/s* of all disks. Constantly high values in the *svctm* and *%util* columns could be an indication that the amount of free space on the disk is insufficient.

Network Statistics Reports: sar `-n` ***`KEYWORD`***

The option `-n` lets you generate multiple network related reports. Specify one of the following keywords along with the `-n`:

- *DEV*: Generates a statistic report for all network devices
- *EDEV*: Generates an error statistics report for all network devices
- *NFS*: Generates a statistic report for an NFS client
- *NFSD*: Generates a statistic report for an NFS server
- *SOCK*: Generates a statistic report on sockets
- *ALL*: Generates all network statistic reports

Visualizing sar Data

`sar` reports are not always easy to parse for humans. kSar, a Java application visualizing your `sar` data, creates easy-to-read graphs. It can even generate PDF reports. kSar takes data generated on the fly as well as past data from a file. kSar is licensed under the BSD license and is available from `http://ksar.atomique.net/`.

2.2 System Information

2.2.1 Device Load Information: iostat

`iostat` monitors the system device loading. It generates reports that can be useful for better balancing the load between physical disks attached to your system.

The first `iostat` report shows statistics collected since the system was booted. Subsequent reports cover the time since the previous report.

```
tux@mercury:~> iostat
Linux 2.6.32.7-0.2-default (geeko@buildhost)  02/24/10  _x86_64_

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0,49    0,01    0,10    0,31    0,00   99,09

Device:            tps   Blk_read/s   Blk_wrtn/s   Blk_read   Blk_wrtn
sda               1,34         5,59        25,37    1459766    6629160
sda1              0,00         0,01         0,00       1519          0
sda2              0,87         5,11        17,83    1335365    4658152
sda3              0,47         0,47         7,54     122578    1971008
```

When invoked with the `-n` option, `iostat` adds statistics of network file systems (NFS) load. The option `-x` shows extended statistics information.

You can also specify which device should be monitored at what time intervals. For example, `iostat -p sda 3 5` will display five reports at three second intervals for device sda.

NOTE: sysstat Package

`iostat` is part of sysstat package. To use it, install the package with `zypper in sysstat`

2.2.2 Processor Activity Monitoring: mpstat

The utility `mpstat` examines activities of each available processor. If your system has one processor only, the global average statistics will be reported.

With the -P option, you can specify the number of processors to be reported (note that 0 is the first processor). The timing arguments work the same way as with the iostat command. Entering mpstat -P 1 2 5 prints five reports for the second processor (number 1) at 2 second intervals.

```
tux@mercury:~> mpstat -P 1 2 5
Linux 2.6.32.7-0.2-default (geeko@buildhost)  02/24/10  _x86_64_

08:57:10  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  \
 %guest   %idle
08:57:12    1    4.46    0.00    5.94    0.50    0.00    0.00    0.00  \
 0.00   89.11
08:57:14    1    1.98    0.00    2.97    0.99    0.00    0.99    0.00  \
 0.00   93.07
08:57:16    1    2.50    0.00    3.00    0.00    0.00    1.00    0.00  \
 0.00   93.50
08:57:18    1   14.36    0.00    1.98    0.00    0.00    0.50    0.00  \
 0.00   83.17
08:57:20    1    2.51    0.00    4.02    0.00    0.00    2.01    0.00  \
 0.00   91.46
Average:    1    5.17    0.00    3.58    0.30    0.00    0.90    0.00  \
 0.00   90.05
```

2.2.3 Task Monitoring: pidstat

If you need to see what load a particular task applies to your system, use pidstat command. It prints activity of every selected task or all tasks managed by Linux kernel if no task is specified. You can also set the number of reports to be displayed and the time interval between them.

For example, pidstat -C top 2 3 prints the load statistic for tasks whose command name includes the string “top”. There will be three reports printed at two second intervals.

```
tux@mercury:~> pidstat -C top 2 3
Linux 2.6.27.19-5-default (geeko@buildhost)  03/23/2009  _x86_64_

09:25:42 AM       PID    %usr %system  %guest    %CPU   CPU  Command
09:25:44 AM     23576   37.62   61.39    0.00   99.01     1  top

09:25:44 AM       PID    %usr %system  %guest    %CPU   CPU  Command
09:25:46 AM     23576   37.00   62.00    0.00   99.00     1  top

09:25:46 AM       PID    %usr %system  %guest    %CPU   CPU  Command
09:25:48 AM     23576   38.00   61.00    0.00   99.00     1  top
```

```
Average:          PID    %usr %system  %guest    %CPU   CPU  Command
Average:        23576   37.54   61.46    0.00   99.00     -  top
```

2.2.4 Kernel Ring Buffer: dmesg

The Linux kernel keeps certain messages in a ring buffer. To view these messages, enter the command `dmesg`:

```
tux@mercury:~> dmesg
[...]
end_request: I/O error, dev fd0, sector 0
subfs: unsuccessful attempt to mount media (256)
e100: eth0: e100_watchdog: link up, 100Mbps, half-duplex
NET: Registered protocol family 17
IA-32 Microcode Update Driver: v1.14 <tigran@veritas.com>
microcode: CPU0 updated from revision 0xe to 0x2e, date = 08112004
IA-32 Microcode Update Driver v1.14 unregistered
bootsplash: status on console 0 changed to on
NET: Registered protocol family 10
Disabled Privacy Extensions on device c0326ea0(lo)
IPv6 over IPv4 tunneling driver
powernow: This module only works with AMD K7 CPUs
bootsplash: status on console 0 changed to on
```

Older events are logged in the files `/var/log/messages` and `/var/log/warn`.

2.2.5 List of Open Files: lsof

To view a list of all the files open for the process with process ID *`PID`*, use `-p`. For example, to view all the files used by the current shell, enter:

```
tux@mercury:~> lsof -p $$
COMMAND  PID   USER   FD   TYPE DEVICE  SIZE/OFF NODE NAME
bash    5552 tux  cwd    DIR    3,3    1512 117619 /home/tux
bash    5552 tux  rtd    DIR    3,3     584      2 /
bash    5552 tux  txt    REG    3,3  498816  13047 /bin/bash
bash    5552 tux  mem    REG    0,0              0 [heap] (stat: No such
bash    5552 tux  mem    REG    3,3  217016 115687 /var/run/nscd/passwd
bash    5552 tux  mem    REG    3,3  208464  11867 /usr/lib/locale/en_GB.
[...]
bash    5552 tux  mem    REG    3,3     366   9720 /usr/lib/locale/en_GB.
bash    5552 tux  mem    REG    3,3   97165   8828 /lib/ld-2.3.6.so
bash    5552 tux    0u   CHR  136,5              7 /dev/pts/5
bash    5552 tux    1u   CHR  136,5              7 /dev/pts/5
bash    5552 tux    2u   CHR  136,5              7 /dev/pts/5
bash    5552 tux  255u   CHR  136,5              7 /dev/pts/5
```

The special shell variable `$$`, whose value is the process ID of the shell, has been used.

The command `lsof` lists all the files currently open when used without any parameters. There are often thousands of open files, therefore, listing all of them is rarely useful. However, the list of all files can be combined with search functions to generate useful lists. For example, list all used character devices:

```
tux@mercury:~> lsof | grep CHR
bash      3838     tux    0u      CHR  136,0                 2 /dev/pts/0
bash      3838     tux    1u      CHR  136,0                 2 /dev/pts/0
bash      3838     tux    2u      CHR  136,0                 2 /dev/pts/0
bash      3838     tux  255u      CHR  136,0                 2 /dev/pts/0
bash      5552     tux    0u      CHR  136,5                 7 /dev/pts/5
bash      5552     tux    1u      CHR  136,5                 7 /dev/pts/5
bash      5552     tux    2u      CHR  136,5                 7 /dev/pts/5
bash      5552     tux  255u      CHR  136,5                 7 /dev/pts/5
X         5646       root  mem       CHR    1,1              1006 /dev/mem
lsof      5673     tux    0u      CHR  136,5                 7 /dev/pts/5
lsof      5673     tux    2u      CHR  136,5                 7 /dev/pts/5
grep      5674     tux    1u      CHR  136,5                 7 /dev/pts/5
grep      5674     tux    2u      CHR  136,5                 7 /dev/pts/5
```

When used with `-i`, `lsof` lists currently open Internet files as well:

```
tux@mercury:~> lsof -i
[...]
pidgin     4349 tux   17r  IPv4  15194      0t0  TCP \
 jupiter.example.com:58542->www.example.net:https (ESTABLISHED)
pidgin     4349 tux   21u  IPv4  15583      0t0  TCP \
 jupiter.example.com:37051->aol.example.org:aol (ESTABLISHED)
evolution  4578 tux   38u  IPv4  16102      0t0  TCP \
 jupiter.example.com:57419->imap.example.com:imaps (ESTABLISHED)
npviewer.  9425 tux   40u  IPv4  24769      0t0  TCP \
 jupiter.example.com:51416->www.example.com:http (CLOSE_WAIT)
npviewer.  9425 tux   49u  IPv4  24814      0t0  TCP \
 jupiter.example.com:43964->www.example.org:http (CLOSE_WAIT)
ssh       17394 tux    3u  IPv4  40654      0t0  TCP \
 jupiter.example.com:35454->saturn.example.com:ssh (ESTABLISHED)
```

2.2.6 Kernel and udev Event Sequence Viewer: udevadm monitor

`udevadm monitor` listens to the kernel uevents and events sent out by a udev rule and prints the device path (DEVPATH) of the event to the console. This is a sequence of events while connecting a USB memory stick:

NOTE: Monitoring udev Events

Only root user is allowed to monitor udev events by running the `udevadm` command.

```
UEVENT[1138806687] add@/devices/pci0000:00/0000:00:1d.7/usb4/4-2/4-2.2
UEVENT[1138806687] add@/devices/pci0000:00/0000:00:1d.7/usb4/4-2/4-2.2/4-2.2
UEVENT[1138806687] add@/class/scsi_host/host4
UEVENT[1138806687] add@/class/usb_device/usbdev4.10
UDEV  [1138806687] add@/devices/pci0000:00/0000:00:1d.7/usb4/4-2/4-2.2
UDEV  [1138806687] add@/devices/pci0000:00/0000:00:1d.7/usb4/4-2/4-2.2/4-2.2
UDEV  [1138806687] add@/class/scsi_host/host4
UDEV  [1138806687] add@/class/usb_device/usbdev4.10
UEVENT[1138806692] add@/devices/pci0000:00/0000:00:1d.7/usb4/4-2/4-2.2/4-2.2
UEVENT[1138806692] add@/block/sdb
UEVENT[1138806692] add@/class/scsi_generic/sg1
UEVENT[1138806692] add@/class/scsi_device/4:0:0:0
UDEV  [1138806693] add@/devices/pci0000:00/0000:00:1d.7/usb4/4-2/4-2.2/4-2.2
UDEV  [1138806693] add@/class/scsi_generic/sg1
UDEV  [1138806693] add@/class/scsi_device/4:0:0:0
UDEV  [1138806693] add@/block/sdb
UEVENT[1138806694] add@/block/sdb/sdb1
UDEV  [1138806694] add@/block/sdb/sdb1
UEVENT[1138806694] mount@/block/sdb/sdb1
UEVENT[1138806697] umount@/block/sdb/sdb1
```

2.2.7 Information on Security Events: audit

The Linux audit framework is a complex auditing system that collects detailed information about all security related events. These records can be consequently analyzed to discover if, for example, a violation of security policies occurred. For more information on audit, see Part "*The Linux Audit Framework*" (↑*Security Guide*).

2.3 Processes

2.3.1 Interprocess Communication: ipcs

The command `ipcs` produces a list of the IPC resources currently in use:

```
------ Shared Memory Segments --------
key        shmid      owner     perms       bytes      nattch     status
0x00000000 58261504   tux    600        393216     2          dest
0x00000000 58294273   tux    600        196608     2          dest
```

```
0x00000000 83886083   tux    666        43264      2
0x00000000 83951622   tux    666        192000     2
0x00000000 83984391   tux    666        282464     2
0x00000000 84738056   root      644        151552     2          dest

------ Semaphore Arrays --------
key        semid      owner     perms      nsems
0x4d038abf 0          tux    600        8

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
```

2.3.2 Process List: ps

The command `ps` produces a list of processes. Most parameters must be written without a minus sign. Refer to `ps --help` for a brief help or to the man page for extensive help.

To list all processes with user and command line information, use `ps axu`:

```
tux@mercury:~> ps axu
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0    696   272 ?        S    12:59   0:01 init [5]
root         2  0.0  0.0      0     0 ?        SN   12:59   0:00 [ksoftirqd
root         3  0.0  0.0      0     0 ?        S<   12:59   0:00 [events
[...]
tux    4047  0.0  6.0 158548 31400 ?        Ssl  13:02   0:06 mono-best
tux    4057  0.0  0.7   9036  3684 ?        Sl   13:02   0:00 /opt/gnome
tux    4067  0.0  0.1   2204   636 ?        S    13:02   0:00 /opt/gnome
tux    4072  0.0  1.0  15996  5160 ?        Ss   13:02   0:00 gnome-scre
tux    4114  0.0  3.7 130988 19172 ?        SLl  13:06   0:04 sound-juic
tux    4818  0.0  0.3   4192  1812 pts/0    Ss   15:59   0:00 -bash
tux    4959  0.0  0.1   2324   816 pts/0    R+   16:17   0:00 ps axu
```

To check how many `sshd` processes are running, use the option `-p` together with the command `pidof`, which lists the process IDs of the given processes.

```
tux@mercury:~> ps -p $(pidof sshd)
  PID TTY      STAT   TIME COMMAND
 3524 ?        Ss     0:00 /usr/sbin/sshd -o PidFile=/var/run/sshd.init.pid
 4813 ?        Ss     0:00 sshd: tux [priv]
 4817 ?        R      0:00 sshd: tux@pts/0
```

The process list can be formatted according to your needs. The option `-L` returns a list of all keywords. Enter the following command to issue a list of all processes sorted by memory usage:

```
tux@mercury:~> ps ax --format pid,rss,cmd --sort rss
  PID   RSS CMD
    2     0 [ksoftirqd/0]
    3     0 [events/0]
    4     0 [khelper]
    5     0 [kthread]
   11     0 [kblockd/0]
   12     0 [kacpid]
  472     0 [pdflush]
  473     0 [pdflush]
[...]
 4028 17556 nautilus --no-default-window --sm-client-id default2
 4118 17800 ksnapshot
 4114 19172 sound-juicer
 4023 25144 gnome-panel --sm-client-id default1
 4047 31400 mono-best --debug /usr/lib/beagle/Best.exe --autostarted
 3973 31520 mono-beagled --debug /usr/lib/beagle/BeagleDaemon.exe --bg --aut
```

Useful `ps` Calls

`ps aux --sort` *`column`*

Sort the output by *`column`*. Replace *`column`* with

`pmem` for physical memory ratio
`pcpu` for CPU ratio
`rss` for resident set size (non-swapped physical memory)

`ps axo pid,%cpu,rss,vsz,args,wchan`

Shows every process, their PID, CPU usage ratio, memory size (resident and virtual), name, and their syscall.

`ps axfo pid,args`

Show a process tree.

2.3.3 Process Tree: pstree

The command `pstree` produces a list of processes in the form of a tree:

```
tux@mercury:~> pstree
init-+-NetworkManagerD
     |-acpid
     |-3*[automount]
     |-cron
     |-cupsd
     |-2*[dbus-daemon]
     |-dbus-launch
     |-dcopserver
     |-dhcpcd
     |-events/0
     |-gpg-agent
     |-hald-+-hald-addon-acpi
     |      `-hald-addon-stor
     |-kded
     |-kdeinit-+-kdesu---su---kdesu_stub---yast2---y2controlcenter
     |         |-kio_file
     |         |-klauncher
     |         |-konqueror
     |         |-konsole-+-bash---su---bash
     |         |         `-bash
     |         `-kwin
     |-kdesktop---kdesktop_lock---xmatrix
     |-kdesud
     |-kdm-+-X
     |     `-kdm---startkde---kwrapper
[...]
```

The parameter `-p` adds the process ID to a given name. To have the command lines displayed as well, use the `-a` parameter:

2.3.4 Table of Processes: top

The command `top`, which stands for `table of processes`, displays a list of processes that is refreshed every two seconds. To terminate the program, press Q. The parameter `-n 1` terminates the program after a single display of the process list. The following is an example output of the command `top -n 1`:

```
tux@mercury:~> top -n 1
top - 17:06:28 up  2:10,  5 users,  load average: 0.00, 0.00, 0.00
Tasks:  85 total,   1 running,  83 sleeping,   1 stopped,   0 zombie
Cpu(s):  5.5% us,  0.8% sy,  0.8% ni, 91.9% id,  1.0% wa,  0.0% hi,  0.0% si
Mem:    515584k total,   506468k used,     9116k free,    66324k buffers
Swap:   658656k total,        0k used,   658656k free,   353328k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
    1 root      16   0   700  272  236 S  0.0  0.1   0:01.33 init
    2 root      34  19     0    0    0 S  0.0  0.0   0:00.00 ksoftirqd/0
    3 root      10  -5     0    0    0 S  0.0  0.0   0:00.27 events/0
    4 root      10  -5     0    0    0 S  0.0  0.0   0:00.01 khelper
    5 root      10  -5     0    0    0 S  0.0  0.0   0:00.00 kthread
   11 root      10  -5     0    0    0 S  0.0  0.0   0:00.05 kblockd/0
   12 root      20  -5     0    0    0 S  0.0  0.0   0:00.00 kacpid
  472 root      20   0     0    0    0 S  0.0  0.0   0:00.00 pdflush
  473 root      15   0     0    0    0 S  0.0  0.0   0:00.06 pdflush
  475 root      11  -5     0    0    0 S  0.0  0.0   0:00.00 aio/0
  474 root      15   0     0    0    0 S  0.0  0.0   0:00.07 kswapd0
  681 root      10  -5     0    0    0 S  0.0  0.0   0:00.01 kseriod
  839 root      10  -5     0    0    0 S  0.0  0.0   0:00.02 reiserfs/0
  923 root      13  -4  1712  552  344 S  0.0  0.1   0:00.67 udevd
 1343 root      10  -5     0    0    0 S  0.0  0.0   0:00.00 khubd
 1587 root      20   0     0    0    0 S  0.0  0.0   0:00.00 shpchpd_event
 1746 root      15   0     0    0    0 S  0.0  0.0   0:00.00 w1_control
 1752 root      15   0     0    0    0 S  0.0  0.0   0:00.00 w1_bus_master1
 2151 root      16   0  1464  496  416 S  0.0  0.1   0:00.00 acpid
 2165 messageb  16   0  3340 1048  792 S  0.0  0.2   0:00.64 dbus-daemon
 2166 root      15   0  1840  752  556 S  0.0  0.1   0:00.01 syslog-ng
 2171 root      16   0  1600  516  320 S  0.0  0.1   0:00.00 klogd
 2235 root      15   0  1736  800  652 S  0.0  0.2   0:00.10 resmgrd
 2289 root      16   0  4192 2852 1444 S  0.0  0.6   0:02.05 hald
 2403 root      23   0  1756  600  524 S  0.0  0.1   0:00.00 hald-addon-acpi
 2709 root      19   0  2668 1076  944 S  0.0  0.2   0:00.00 NetworkManagerD
 2714 root      16   0  1756  648  564 S  0.0  0.1   0:00.56 hald-addon-stor
```

By default the output is sorted by CPU usage (column *%CPU*, shortcut Shift + P). Use following shortcuts to change the sort field:

Shift + M: Resident Memory (*RES*)
Shift + N: Process ID (*PID*)
Shift + T: Time (*TIME+*)

To use any other field for sorting, press F and select a field from the list. To toggle the sort order, Use Shift + R.

The parameter `-U UID` monitors only the processes associated with a particular user. Replace *`UID`* with the user ID of the user. Use `top -U $(id -u)` to show processes of the current user

2.3.5 Modify a process' niceness: nice and renice

The kernel determines which processes require more CPU time than others by the process' nice level, also called niceness. The higher the “nice” level of a process is, the less CPU time it will take from other processes. Nice levels range from -20 (the least “nice” level) to 19. Negative values can only be set by `root`.

Adjusting the niceness level is useful when running a non time-critical process that lasts long and uses large amounts of CPU time, such as compiling a kernel on a system that also performs other tasks. Making such a process “nicer”, ensures that the other tasks, for example a Web server, will have a higher priority.

Calling `nice` without any parameters prints the current niceness:

```
tux@mercury:~> nice
0
```

Running `nice` *`command`* increments the current nice level for the given command by 10. Using `nice -n` *`level command`* lets you specify a new niceness relative to the current one.

To change the niceness of a running process, use `renice` *`priority`* `-p` *`process id`*, for example:

```
renice +5 3266
```

To renice all processes owned by a specific user, use the option `-u` *`user`*. Process groups are reniced by the option `-g` *`process group id`*.

2.4 Memory

2.4.1 Memory Usage: free

The utility `free` examines RAM and swap usage. Details of both free and used memory and swap areas are shown:

```
tux@mercury:~> free
            total       used       free     shared    buffers     cached
Mem:       2062844    2047444      15400          0     129580     921936
-/+ buffers/cache:     995928    1066916
Swap:      2104472          0    2104472
```

The options -b, -k, -m, -g show the output in bytes, KB, MB, or GB, respectively. The parameter -d delay ensures that the display is refreshed every *delay* seconds. For example, free -d 1.5 produces an update every 1.5 seconds.

2.4.2 Detailed Memory Usage: /proc/meminfo

Use /proc/meminfo to get more detailed information on memory usage than with free. Actually free uses some of the data from this file. See an example output from a 64bit system below. Note that it slightly differs on 32bit systems due to different memory management):

```
tux@mercury:~> cat /proc/meminfo
MemTotal:        8182956 kB
MemFree:         1045744 kB
Buffers:          364364 kB
Cached:          5601388 kB
SwapCached:         1936 kB
Active:          4048268 kB
Inactive:        2674796 kB
Active(anon):     663088 kB
Inactive(anon):   107108 kB
Active(file):    3385180 kB
Inactive(file):  2567688 kB
Unevictable:           4 kB
Mlocked:               4 kB
SwapTotal:       2096440 kB
SwapFree:        2076692 kB
Dirty:                44 kB
Writeback:             0 kB
AnonPages:        756108 kB
Mapped:           147320 kB
Slab:             329216 kB
SReclaimable:     300220 kB
SUnreclaim:        28996 kB
PageTables:        21092 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:     6187916 kB
Committed_AS:    1388160 kB
```

```
VmallocTotal:   34359738367 kB
VmallocUsed:      133384 kB
VmallocChunk:   34359570939 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
DirectMap4k:     2689024 kB
DirectMap2M:     5691392 kB
```

The most important entries are:

MemTotal

Total amount of usable RAM

MemFree

Total amount of unused RAM

Buffers

File buffer cache in RAM

Cached

Page cache in RAM

SwapCached

Page cache in swap

Active

Recently used memory that normally is not reclaimed. This value is the sum of memory claimed by anonymous pages (listed as *Active(anon)*) and file-backed pages (listed as *Active(file)*)

Inactive

Recently unused memory that can be reclaimed. This value is the sum of memory claimed by anonymous pages (listed as *Inactive(anon)*) and file-backed pages (listed as *Inactive(file)*).

SwapTotal

Total amount of swap space

SwapFree

Total amount of unused swap space

Dirty
: Amount of memory that will be written to disk

Writeback
: Amount of memory that currently is written to disk

Mapped
: Memory claimed with the nmap command

Slab
: Kernel data structure cache

Committed_AS
: An approximation of the total amount of memory (RAM plus swap) the current workload needs in the worst case.

2.4.3 Process Memory Usage: smaps

Exactly determining how much memory a certain process is consuming is not possible with standard tools like `top` or `ps`. Use the smaps subsystem, introduced in Kernel 2.6.14, if you need exact data. It can be found at `/proc/`*`pid`*`/smaps` and shows you the number of clean and dirty memory pages the process with the ID *`PID`* is using at that time. It differentiates between shared and private memory, so you are able to see how much memory the process is using without including memory shared with other processes.

2.5 Networking

2.5.1 Show the Network Status: netstat

`netstat` shows network connections, routing tables (`-r`), interfaces (`-i`), masquerade connections (`-M`), multicast memberships (`-g`), and statistics (`-s`).

```
tux@mercury:~> netstat -r
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
192.168.2.0     *               255.255.254.0   U         0 0          0 eth0
link-local      *               255.255.0.0     U         0 0          0 eth0
```

```
loopback        *               255.0.0.0      U        0 0          0 lo
default         192.168.2.254   0.0.0.0        UG       0 0          0 eth0
```

```
tux@mercury:~> netstat -i
Kernel Interface table
Iface   MTU Met   RX-OK RX-ERR RX-DRP RX-OVR  TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0   1500   0 1624507 129056      0      0   7055      0      0      0 BMNRU
lo    16436   0   23728      0      0      0  23728      0      0      0 LRU
```

When displaying network connections or statistics, you can specify the socket type to display: TCP (`-t`), UDP (`-u`), or raw (`-r`). The `-p` option shows the PID and name of the program to which each socket belongs.

The following example lists all TCP connections and the programs using these connections.

```
mercury:~ # netstat -t -p
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address  Foreign Address       State       PID/Pro
[...]
tcp      0      0 mercury:33513    www.novell.com:www-http ESTABLISHED 6862/fi
tcp      0    352 mercury:ssh      mercury2.:trc-netpoll     ESTABLISHED
19422/s
tcp      0      0 localhost:ssh  localhost:17828         ESTABLISHED -
```

In the following, statistics for the TCP protocol are displayed:

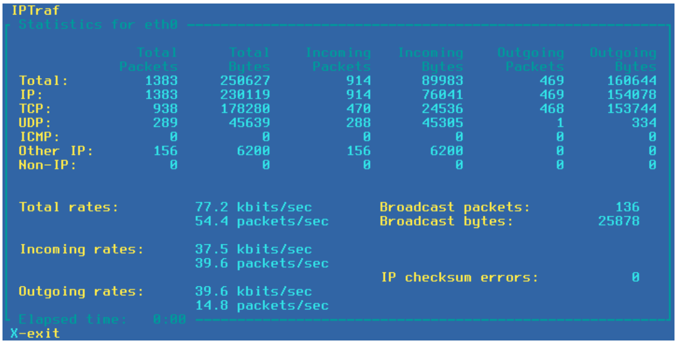
```
tux@mercury:~> netstat -s -t
Tcp:
    2427 active connections openings
    2374 passive connection openings
    0 failed connection attempts
    0 connection resets received
    1 connections established
    27476 segments received
    26786 segments send out
    54 segments retransmited
    0 bad segments received.
    6 resets sent
[...]
    TCPAbortOnLinger: 0
    TCPAbortFailed: 0
    TCPMemoryPressures: 0
```

2.5.2 Interactive Network Monitor: iptraf

The `iptraf` utility is a menu based Local Area Network (LAN) monitor. It generates network statistics, including TCP and UDP counts, Ethernet load information, IP checksum errors and others.

If you enter the command without any option, it runs in an interactive mode. You can navigate through graphical menus and choose the statistics that you want `iptraf` to report. You can also specify which network interface to examine.

Figure 2.1 *iptraf Running in Interactive Mode*



```
IPTraf
┌ Statistics for eth0 ─────────────────────────────────────────────────────────┐
│                                                                              │
│               Total      Total   Incoming   Incoming   Outgoing   Outgoing   │
│             Packets      Bytes    Packets      Bytes    Packets      Bytes   │
│ Total:         1383     250627        914      89983        469     160644   │
│ IP:            1383     230119        914      76041        469     154078   │
│ TCP:            938     178280        470      24536        468     153744   │
│ UDP:            289      45639        288      45305          1        334   │
│ ICMP:             0          0          0          0          0          0   │
│ Other IP:       156       6200        156       6200          0          0   │
│ Non-IP:           0          0          0          0          0          0   │
│                                                                              │
│                                                                              │
│ Total rates:         77.2 kbits/sec        Broadcast packets:         136    │
│                      54.4 packets/sec      Broadcast bytes:         25878    │
│                                                                              │
│ Incoming rates:      37.5 kbits/sec                                          │
│                      39.6 packets/sec                                        │
│                                            IP checksum errors:          0    │
│ Outgoing rates:      39.6 kbits/sec                                          │
│                      14.8 packets/sec                                        │
└ Elapsed time:   0:00 ──────────────────────────────────────────────────────────┘
X-exit
```

The command `iptraf` understands several options and can be run in a batch mode as well. The following example will collect statistics for network interface eth0 (`-i`) for 1 minute (`-t`). It will be run in the background (`-B`) and the statistics will be written to the `iptraf.log` file in your home directory (`-L`).

```
tux@mercury:~> iptraf -i eth0 -t 1 -B -L ~/iptraf.log
```

You can examine the log file with the `more` command:

```
tux@mercury:~> more ~/iptraf.log
Mon Mar 23 10:08:02 2010; ******** IP traffic monitor started ********
Mon Mar 23 10:08:02 2010; UDP; eth0; 107 bytes; from 192.168.1.192:33157 to
\
 239.255.255.253:427
Mon Mar 23 10:08:02 2010; VRRP; eth0; 46 bytes; from 192.168.1.252 to \
 224.0.0.18
Mon Mar 23 10:08:03 2010; VRRP; eth0; 46 bytes; from 192.168.1.252 to \
 224.0.0.18
Mon Mar 23 10:08:03 2010; VRRP; eth0; 46 bytes; from 192.168.1.252 to \
 224.0.0.18
```

```
[...]
Mon Mar 23 10:08:06 2010; UDP; eth0; 132 bytes; from 192.168.1.54:54395 to \
 10.20.7.255:111
Mon Mar 23 10:08:06 2010; UDP; eth0; 46 bytes; from 192.168.1.92:27258 to \
 10.20.7.255:8765
Mon Mar 23 10:08:06 2010; UDP; eth0; 124 bytes; from 192.168.1.139:43464 to
\
 10.20.7.255:111
Mon Mar 23 10:08:06 2010; VRRP; eth0; 46 bytes; from 192.168.1.252 to \
 224.0.0.18
--More--(7%)
```

2.6 The /proc File System

The `/proc` file system is a pseudo file system in which the kernel reserves important information in the form of virtual files. For example, display the CPU type with this command:

```
tux@mercury:~> cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 15
model           : 4
model name      : Intel(R) Pentium(R) 4 CPU 3.40GHz
stepping        : 3
cpu MHz         : 2800.000
cache size      : 2048 KB
physical id     : 0
[...]
```

Query the allocation and use of interrupts with the following command:

```
tux@mercury:~> cat /proc/interrupts
           CPU0
  0:    3577519          XT-PIC  timer
  1:        130          XT-PIC  i8042
  2:          0          XT-PIC  cascade
  5:     564535          XT-PIC  Intel 82801DB-ICH4
  7:          1          XT-PIC  parport0
  8:          2          XT-PIC  rtc
  9:          1          XT-PIC  acpi, uhci_hcd:usb1, ehci_hcd:usb4
 10:          0          XT-PIC  uhci_hcd:usb3
 11:      71772          XT-PIC  uhci_hcd:usb2, eth0
 12:     101150          XT-PIC  i8042
 14:      33146          XT-PIC  ide0
 15:     149202          XT-PIC  ide1
NMI:          0
LOC:          0
```

```
ERR:          0
MIS:          0
```

Some of the important files and their contents are:

`/proc/devices`
    Available devices

`/proc/modules`
    Kernel modules loaded

`/proc/cmdline`
    Kernel command line

`/proc/meminfo`
    Detailed information about memory usage

`/proc/config.gz`
    `gzip`-compressed configuration file of the kernel currently running

Further information is available in the text file `/usr/src/linux/Documentation/filesystems/proc.txt` (this file is available when the package `kernel-source` is installed). Find information about processes currently running in the `/proc/`*NNN* directories, where *NNN* is the process ID (PID) of the relevant process. Every process can find its own characteristics in `/proc/self/`:

```
tux@mercury:~> ls -l /proc/self
lrwxrwxrwx 1 root root 64 2007-07-16 13:03 /proc/self -> 5356
tux@mercury:~> ls -l /proc/self/
total 0
dr-xr-xr-x 2 tux users 0 2007-07-16 17:04 attr
-r-------- 1 tux users 0 2007-07-16 17:04 auxv
-r--r--r-- 1 tux users 0 2007-07-16 17:04 cmdline
lrwxrwxrwx 1 tux users 0 2007-07-16 17:04 cwd -> /home/tux
-r-------- 1 tux users 0 2007-07-16 17:04 environ
lrwxrwxrwx 1 tux users 0 2007-07-16 17:04 exe -> /bin/ls
dr-x------ 2 tux users 0 2007-07-16 17:04 fd
-rw-r--r-- 1 tux users 0 2007-07-16 17:04 loginuid
-r--r--r-- 1 tux users 0 2007-07-16 17:04 maps
-rw------- 1 tux users 0 2007-07-16 17:04 mem
-r--r--r-- 1 tux users 0 2007-07-16 17:04 mounts
-rw-r--r-- 1 tux users 0 2007-07-16 17:04 oom_adj
-r--r--r-- 1 tux users 0 2007-07-16 17:04 oom_score
lrwxrwxrwx 1 tux users 0 2007-07-16 17:04 root -> /
-rw------- 1 tux users 0 2007-07-16 17:04 seccomp
-r--r--r-- 1 tux users 0 2007-07-16 17:04 smaps
```

```
-r--r--r-- 1 tux users 0 2007-07-16 17:04 stat
[...]
dr-xr-xr-x 3 tux users 0 2007-07-16 17:04 task
-r--r--r-- 1 tux users 0 2007-07-16 17:04 wchan
```

The address assignment of executables and libraries is contained in the `maps` file:

```
tux@mercury:~> cat /proc/self/maps
08048000-0804c000 r-xp 00000000 03:03 17753      /bin/cat
0804c000-0804d000 rw-p 00004000 03:03 17753      /bin/cat
0804d000-0806e000 rw-p 0804d000 00:00 0          [heap]
b7d27000-b7d5a000 r--p 00000000 03:03 11867      /usr/lib/locale/en_GB.utf8/
b7d5a000-b7e32000 r--p 00000000 03:03 11868      /usr/lib/locale/en_GB.utf8/
b7e32000-b7e33000 rw-p b7e32000 00:00 0
b7e33000-b7f45000 r-xp 00000000 03:03 8837       /lib/libc-2.3.6.so
b7f45000-b7f46000 r--p 00112000 03:03 8837       /lib/libc-2.3.6.so
b7f46000-b7f48000 rw-p 00113000 03:03 8837       /lib/libc-2.3.6.so
b7f48000-b7f4c000 rw-p b7f48000 00:00 0
b7f52000-b7f53000 r--p 00000000 03:03 11842      /usr/lib/locale/en_GB.utf8/
[...]
b7f5b000-b7f61000 r--s 00000000 03:03 9109       /usr/lib/gconv/gconv-module
b7f61000-b7f62000 r--p 00000000 03:03 9720       /usr/lib/locale/en_GB.utf8/
b7f62000-b7f76000 r-xp 00000000 03:03 8828       /lib/ld-2.3.6.so
b7f76000-b7f78000 rw-p 00013000 03:03 8828       /lib/ld-2.3.6.so
bfd61000-bfd76000 rw-p bfd61000 00:00 0          [stack]
ffffe000-fffff000 ---p 00000000 00:00 0          [vdso]
```

2.6.1 procinfo

Important information from the `/proc` file system is summarized by the command `procinfo`:

```
tux@mercury:~> procinfo
Linux 2.6.32.7-0.2-default (geeko@buildhost) (gcc 4.3.4) #1 2CPU

Memory:      Total        Used        Free      Shared     Buffers
Mem:       2060604     2011264       49340           0      200664
Swap:      2104472         112     2104360

Bootup: Wed Feb 17 03:39:33 2010    Load average: 0.86 1.10 1.11 3/118 21547

user  :       2:43:13.78   0.8%  page in :   71099181  disk 1:  2827023r 968
nice  :   1d 22:21:27.87  14.7%  page out:  690734737
system:      13:39:57.57   4.3%  page act:  138388345
IOwait:      18:02:18.59   5.7%  page dea:   29639529
hw irq:       0:03:39.44   0.0%  page flt: 9539791626
sw irq:       1:15:35.25   0.4%  swap in :         69
idle  :   9d 16:07:56.79  73.8%  swap out:        209
uptime:   6d 13:07:11.14         context :  542720687
```

```
irq  0: 141399308 timer          irq 14:   5074312 ide0
irq  1:     73784 i8042          irq 50:   1938076 uhci_hcd:usb1, ehci_
irq  4:         2                irq 58:         0 uhci_hcd:usb2
irq  6:         5 floppy [2]     irq 66:    872711 uhci_hcd:usb3, HDA I
irq  7:         2                irq 74:        15 uhci_hcd:usb4
irq  8:         0 rtc            irq 82: 178717720 0         PCI-MSI  e
irq  9:         0 acpi           irq169:  44352794 nvidia
irq 12:         3                irq233:   8209068 0         PCI-MSI  l
```

To see all the information, use the parameter `-a`. The parameter `-nN` produces updates of the information every *N* seconds. In this case, terminate the program by pressing q.

By default, the cumulative values are displayed. The parameter `-d` produces the differential values. `procinfo -dn5` displays the values that have changed in the last five seconds:

2.7 Hardware Information

2.7.1 PCI Resources: lspci

NOTE: Accessing PCI configuration.

Most operating systems require root user privileges to grant access to the computer's PCI configuration.

The command `lspci` lists the PCI resources:

```
mercury:~ # lspci
00:00.0 Host bridge: Intel Corporation 82845G/GL[Brookdale-G]/GE/PE \
    DRAM Controller/Host-Hub Interface (rev 01)
00:01.0 PCI bridge: Intel Corporation 82845G/GL[Brookdale-G]/GE/PE \
    Host-to-AGP Bridge (rev 01)
00:1d.0 USB Controller: Intel Corporation 82801DB/DBL/DBM \
    (ICH4/ICH4-L/ICH4-M) USB UHCI Controller #1 (rev 01)
00:1d.1 USB Controller: Intel Corporation 82801DB/DBL/DBM \
    (ICH4/ICH4-L/ICH4-M) USB UHCI Controller #2 (rev 01)
00:1d.2 USB Controller: Intel Corporation 82801DB/DBL/DBM \
    (ICH4/ICH4-L/ICH4-M) USB UHCI Controller #3 (rev 01)
00:1d.7 USB Controller: Intel Corporation 82801DB/DBM \
    (ICH4/ICH4-M) USB2 EHCI Controller (rev 01)
00:1e.0 PCI bridge: Intel Corporation 82801 PCI Bridge (rev 81)
00:1f.0 ISA bridge: Intel Corporation 82801DB/DBL (ICH4/ICH4-L) \
    LPC Interface Bridge (rev 01)
```

```
00:1f.1 IDE interface: Intel Corporation 82801DB (ICH4) IDE \
    Controller (rev 01)
00:1f.3 SMBus: Intel Corporation 82801DB/DBL/DBM (ICH4/ICH4-L/ICH4-M) \
    SMBus Controller (rev 01)
00:1f.5 Multimedia audio controller: Intel Corporation 82801DB/DBL/DBM \
    (ICH4/ICH4-L/ICH4-M) AC'97 Audio Controller (rev 01)
01:00.0 VGA compatible controller: Matrox Graphics, Inc. G400/G450 (rev 85)
02:08.0 Ethernet controller: Intel Corporation 82801DB PRO/100 VE (LOM) \
    Ethernet Controller (rev 81)
```

Using `-v` results in a more detailed listing:

```
mercury:~ # lspci -v
[...]
00:03.0 Ethernet controller: Intel Corporation 82540EM Gigabit Ethernet \
Controller (rev 02)
  Subsystem: Intel Corporation PRO/1000 MT Desktop Adapter
  Flags: bus master, 66MHz, medium devsel, latency 64, IRQ 19
  Memory at f0000000 (32-bit, non-prefetchable) [size=128K]
  I/O ports at d010 [size=8]
  Capabilities: [dc] Power Management version 2
  Capabilities: [e4] PCI-X non-bridge device
  Kernel driver in use: e1000
  Kernel modules: e1000
```

Information about device name resolution is obtained from the file `/usr/share/pci.ids`. PCI IDs not listed in this file are marked “Unknown device.”

The parameter `-vv` produces all the information that could be queried by the program. To view the pure numeric values, use the parameter `-n`.

2.7.2 USB Devices: lsusb

The command `lsusb` lists all USB devices. With the option `-v`, print a more detailed list. The detailed information is read from the directory `/proc/bus/usb/`. The following is the output of `lsusb` with these USB devices attached: hub, memory stick, hard disk and mouse.

```
mercury:/ # lsusb
Bus 004 Device 007: ID 0ea0:2168 Ours Technology, Inc. Transcend JetFlash \
    2.0 / Astone USB Drive
Bus 004 Device 006: ID 04b4:6830 Cypress Semiconductor Corp. USB-2.0 IDE \
    Adapter
Bus 004 Device 005: ID 05e3:0605 Genesys Logic, Inc.
Bus 004 Device 001: ID 0000:0000
Bus 003 Device 001: ID 0000:0000
Bus 002 Device 001: ID 0000:0000
Bus 001 Device 005: ID 046d:c012 Logitech, Inc. Optical Mouse
Bus 001 Device 001: ID 0000:0000
```

2.8 Files and File Systems

2.8.1 Determine the File Type: file

The command `file` determines the type of a file or a list of files by checking `/usr/share/misc/magic`.

```
tux@mercury:~> file /usr/bin/file
/usr/bin/file: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), \
    for GNU/Linux 2.6.4, dynamically linked (uses shared libs), stripped
```

The parameter `-f` *`list`* specifies a file with a list of filenames to examine. The `-z` allows `file` to look inside compressed files:

```
tux@mercury:~> file /usr/share/man/man1/file.1.gz
usr/share/man/man1/file.1.gz: gzip compressed data, from Unix, max compression
tux@mercury:~> file -z /usr/share/man/man1/file.1.gz
/usr/share/man/man1/file.1.gz: troff or preprocessor input text \
    (gzip compressed data, from Unix, max compression)
```

The parameter `-i` outputs a mime type string rather than the traditional description.

```
tux@mercury:~> file -i /usr/share/misc/magic
/usr/share/misc/magic: text/plain charset=utf-8
```

2.8.2 File Systems and Their Usage: mount, df and du

The command `mount` shows which file system (device and type) is mounted at which mount point:

```
tux@mercury:~> mount
/dev/sda3 on / type reiserfs (rw,acl,user_xattr)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
udev on /dev type tmpfs (rw)
devpts on /dev/pts type devpts (rw,mode=0620,gid=5)
/dev/sda1 on /boot type ext2 (rw,acl,user_xattr)
```

```
/dev/sda4 on /local type reiserfs (rw,acl,user_xattr)
/dev/fd0 on /media/floppy type subfs (rw,nosuid,nodev,noatime,fs=floppyfss,p
```

Obtain information about total usage of the file systems with the command `df`. The parameter `-h` (or `--human-readable`) transforms the output into a form understandable for common users.

```
tux@mercury:~> df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/sda3              11G  3.2G  6.9G  32% /
udev                  252M  104K  252M   1% /dev
/dev/sda1              16M  6.6M  7.8M  46% /boot
/dev/sda4              27G   34M   27G   1% /local
```

Display the total size of all the files in a given directory and its subdirectories with the command `du`. The parameter `-s` suppresses the output of detailed information and gives only a total for each argument. `-h` again transforms the output into a human-readable form:

```
tux@mercury:~> du -sh /opt
192M    /opt
```

2.8.3 Additional Information about ELF Binaries

Read the content of binaries with the `readelf` utility. This even works with ELF files that were built for other hardware architectures:

```
tux@mercury:~> readelf --file-header /bin/ls
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x402540
  Start of program headers:          64 (bytes into file)
  Start of section headers:          95720 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         9
  Size of section headers:           64 (bytes)
  Number of section headers:         32
  Section header string table index: 31
```

2.8.4 File Properties: stat

The command `stat` displays file properties:

```
tux@mercury:~> stat /etc/profile
  File: `/etc/profile'
  Size: 9662       Blocks: 24         IO Block: 4096   regular file
Device: 802h/2050d Inode: 132349      Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2009-03-20 07:51:17.000000000 +0100
Modify: 2009-01-08 19:21:14.000000000 +0100
Change: 2009-03-18 12:55:31.000000000 +0100
```

The parameter `--file-system` produces details of the properties of the file system in which the specified file is located:

```
tux@mercury:~> stat /etc/profile --file-system
  File: "/etc/profile"
    ID: d4fb76e70b4d1746 Namelen: 255     Type: ext2/ext3
Block size: 4096       Fundamental block size: 4096
Blocks: Total: 2581445    Free: 1717327    Available: 1586197
Inodes: Total: 655776     Free: 490312
```

2.9 User Information

2.9.1 User Accessing Files: fuser

It can be useful to determine what processes or users are currently accessing certain files. Suppose, for example, you want to unmount a file system mounted at `/mnt`. `umount` returns "device is busy." The command `fuser` can then be used to determine what processes are accessing the device:

```
tux@mercury:~> fuser -v /mnt/*

                     USER        PID ACCESS COMMAND
/mnt/notes.txt       tux    26597 f....  less
```

Following termination of the `less` process, which was running on another terminal, the file system can successfully be unmounted. When used with `-k` option, `fuser` will kill processes accessing the file as well.

2.9.2 Who Is Doing What: w

With the command `w`, find out who is logged onto the system and what each user is doing. For example:

```
tux@mercury:~> w
 14:58:43 up 1 day,  1:21,  2 users,  load average: 0.00, 0.00, 0.00
USER     TTY        LOGIN@   IDLE   JCPU   PCPU WHAT
tux      :0        12:25   ?xdm?   1:23   0.12s /bin/sh /usr/bin/startkde
root     pts/4     14:13    0.00s  0.06s  0.00s w
```

If any users of other systems have logged in remotely, the parameter `-f` shows the computers from which they have established the connection.

2.10 Time and Date

2.10.1 Time Measurement with time

Determine the time spent by commands with the `time` utility. This utility is available in two versions: as a shell built-in and as a program (`/usr/bin/time`).

```
tux@mercury:~> time find . > /dev/null

real    0m4.051s
user    0m0.042s
sys     0m0.205s
```

2.11 Graph Your Data: RRDtool

There are a lot of data in the world around you, which can be easily measured in time. For example, changes in the temperature, or the number of data sent or received by your computer's network interface. RRDtool can help you store and visualize such data in detailed and customizable graphs.

RRDtool is available for most UNIX platforms and Linux distributions. SUSE® Linux Enterprise Server ships RRDtool as well. Install it either with YaST or by entering

`zypper install` *`rrdtool`* in the command line as `root`.

TIP

There are Perl, Python, Ruby, or PHP bindings available for RRDtool, so that you can write your own monitoring scripts with your preferred scripting language.

2.11.1 How RRDtool Works

RRDtool is a shortcut of *Round Robin Database tool*. *Round Robin* is a method for manipulating with a constant amount of data. It uses the principle of a circular buffer, where there is no end nor beginning to the data row which is being read. RRDtool uses Round Robin Databases to store and read its data.

As mentioned above, RRDtool is designed to work with data that change in time. The ideal case is a sensor which repeatedly reads measured data (like temperature, speed etc.) in constant periods of time, and then exports them in a given format. Such data are perfectly ready for RRDtool, and it is easy to process them and create the desired output.

Sometimes it is not possible to obtain the data automatically and regularly. Their format needs to be pre-processed before it is supplied to RRDtool, and often you need to manipulate RRDtool even manually.

The following is a simple example of basic RRDtool usage. It illustrates all three important phases of the usual RRDtool workflow: *creating* a database, *updating* measured values, and *viewing* the output.

2.11.2 Simple Real Life Example

Suppose we want to collect and view information about the memory usage in the Linux system as it changes in time. To make the example more vivid, we measure the currently free memory for the period of 40 seconds in 4-second intervals. During the measuring, the three hungry applications that usually consume a lot of system memory have been started and closed: the Firefox Web browser, the Evolution e-mail client, andthe Eclipse development framework.

Collecting Data

RRDtool is very often used to measure and visualize network traffic. In such case, Simple Network Management Protocol (SNMP) is used. This protocol can query network devices for relevant values of their internal counters. Exactly these values are to be stored with RRDtool. For more information on SNMP, see http://www.net-snmp.org/.

Our situation is different - we need to obtain the data manually. A helper script `free_mem.sh` repetitively reads the current state of free memory and writes it to the standard output.

```
tux@mercury:~> cat free_mem.sh
INTERVAL=4
for steps in {1..10}
do
    DATE=`date +%s`
    FREEMEM=`free -b | grep "Mem" | awk '{ print $4 }'`
    sleep $INTERVAL
    echo "rrdtool update free_mem.rrd $DATE:$FREEMEM"
done
```

Points to Notice

- The time interval is set to 4 seconds, and is implemented with the `sleep` command.

- RRDtool accepts time information in a special format - so called *Unix time*. It is defined as the number of seconds since the midnight of January 1, 1970 (UTC). For example, 1272907114 represents 2010-05-03 17:18:34.

- The free memory information is reported in bytes with `free -b`. Prefer to supply basic units (bytes) instead of multiple units (like kilobytes).

- The line with the `echo ...` command contains the future name of the database file (`free_mem.rrd`), and together creates a command line for the purpose of updating RRDtool values.

After running `free_mem.sh`, you see an output similar to this:

```
tux@mercury:~> sh free_mem.sh
rrdtool update free_mem.rrd 1272974835:1182994432
rrdtool update free_mem.rrd 1272974839:1162817536
rrdtool update free_mem.rrd 1272974843:1096269824
rrdtool update free_mem.rrd 1272974847:1034219520
rrdtool update free_mem.rrd 1272974851:909438976
rrdtool update free_mem.rrd 1272974855:832454656
rrdtool update free_mem.rrd 1272974859:829120512
rrdtool update free_mem.rrd 1272974863:1180377088
rrdtool update free_mem.rrd 1272974867:1179369472
rrdtool update free_mem.rrd 1272974871:1181806592
```

It is convenient to redirect the command's output to a file with

```
sh free_mem.sh > free_mem_updates.log
```

to ease its future execution.

Creating Database

Create the initial Robin Round database for our example with the following command:

```
rrdtool create free_mem.rrd --start 1272974834 --step=4 \
DS:memory:GAUGE:600:U:U RRA:AVERAGE:0.5:1:24
```

Points to Notice

- This command creates a file called `free_mem.rrd` for storing our measured values in a Round Robin type database.

- The `--start` option specifies the time (in Unix time) when the first value will be added to the database. In this example, it is one less than the first time value of the `free_mem.sh` output (1272974835).

- The `--step` specifies the time interval in seconds with which the measured data will be supplied to the database.

- The `DS:memory:GAUGE:600:U:U` part introduces a new data source for the database. It is called *memory*, its type is *gauge*, the maximum number between two updates is 600 seconds, and the *minimal* and *maximal* value in the measured range are unknown (U).

- `RRA:AVERAGE:0.5:1:24` creates Round Robin archive (RRA) whose stored data are processed with the *consolidation functions* (CF) that calculates the *average* of data points. 3 arguments of the consolidation function are appended to the end of the line .

If no error message is displayed, then `free_mem.rrd` database is created in the current directory:

```
tux@mercury:~> ls -l free_mem.rrd
-rw-r--r-- 1 tux users 776 May  5 12:50 free_mem.rrd
```

Updating Database Values

After the database is created, you need to fill it with the measured data. In Section “Collecting Data” (page 42), we already prepared the file `free_mem_updates.log` which consists of `rrdtool update` commands. These commands do the update of database values for us.

```
tux@mercury:~> sh free_mem_updates.log; ls -l free_mem.rrd
-rw-r--r--  1 tux users  776 May  5 13:29 free_mem.rrd
```

As you can see, the size of `free_mem.rrd` remained the same even after updating its data.

Viewing Measured Values

We have already measured the values, created the database, and stored the measured value in it. Now we can play with the database, and retrieve or view its values.

To retrieve all the values from our database, enter the following on the command line:

```
tux@mercury:~> rrdtool fetch free_mem.rrd AVERAGE --start 1272974830 \
--end 1272974871
          memory
1272974832: nan
1272974836: 1.1729059840e+09
1272974840: 1.1461806080e+09
1272974844: 1.0807572480e+09
1272974848: 1.0030243840e+09
1272974852: 8.9019289600e+08
1272974856: 8.3162112000e+08
1272974860: 9.1693465600e+08
1272974864: 1.1801251840e+09
1272974868: 1.1799787520e+09
1272974872: nan
```

Points to Notice

- `AVERAGE` will fetch average value points from the database, because only one data source is defined (Section “Creating Database” (page 43)) with `AVERAGE` processing and no other function is available.

- The first line of the output prints the name of the data source as defined in Section “Creating Database” (page 43).

- The left results column represents individual points in time, while the right one represents corresponding measured average values in scientific notation.

- The `nan` in the last line stands for “not a number”.

Now a graph representing representing the values stored in the database is drawn:

```
tux@mercury:~> rrdtool graph free_mem.png \
--start 1272974830 \
--end 1272974871 \
--step=4 \
DEF:free_memory=free_mem.rrd:memory:AVERAGE \
LINE2:free_memory#FF0000 \
--vertical-label "GB" \
--title "Free System Memory in Time" \
--zoom 1.5 \
--x-grid SECOND:1:SECOND:4:SECOND:10:0:%X
```

Points to Notice

- `free_mem.png` is the file name of the graph to be created.

- `--start` and `--end` limit the time range within which the graph will be drawn.

- `--step` specifies the time resolution (in seconds) of the graph.

- The `DEF:...` part is a data definition called *free_memory*. Its data are read from the `free_mem.rrd` database and its data source called *memory*. The *average* value points are calculated, because no others were defined in Section “Creating Database” (page 43).

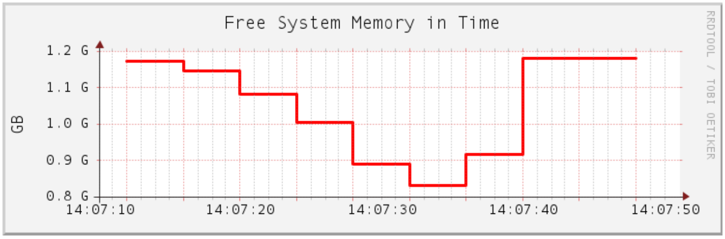
- The `LINE...` part specifies properties of the line to be drawn into the graph. It is 2 pixels wide, its data come from the *free_memory* definition, and its color is red.

- `--vertical-label` sets the label to be printed along the *y* axis, and `--title` sets the main label for the whole graph.

- `--zoom` specifies the zoom factor for the graph. This value must be greater than zero.

- `--x-grid` specifies how to draw grid lines and their labels into the graph. Our example places them every second, while major grid lines are placed every 4 seconds. Labels are placed every 10 seconds under the major grid lines.

***Figure 2.2** Example Graph Created with RRDtool*



2.11.3 For More Information

RRDtool is a very complex tool with a lot of sub-commands and command line options. Some of them are easy to understand, but you have to really *study* RRDtool to make it produce the results you want and fine-tune them according to your liking.

Apart form RRDtool's man page (`man 1 rrdtool`) which gives you only basic information, you should have a look at the RRDtool homepage [`http://oss.oetiker.ch/rrdtool/`]. There is a detailed documentation [`http://oss.oetiker.ch/rrdtool/doc/index.en.html`] of the `rrdtool` command and all its subcommands. There are also several tutorials [`http://oss.oetiker.ch/rrdtool/tut/index.en.html`] to help you understand the common RRDtool workflow.

If you are interested in monitoring network traffic, have a look at MRTG [`http://oss.oetiker.ch/mrtg/`]. It stands for Multi Router Traffic Grapher and can graph the activity of all sorts of network devices. It can easily make use of RRDtool.

3 Monitoring with Nagios

Nagios is a stable, scalable and extensible enterprise-class network and system monitoring tool which allows administrators to monitor network and host resources such as HTTP, SMTP, POP3, disk usage and processor load. Originally Nagios was designed to run under Linux, but it can also be used on several UNIX operating systems. This chapter covers the installation and parts of the configuration of Nagios (`http://www.nagios.org/`).

3.1 Features of Nagios

The most important features of Nagios are:

- Monitoring of network services (SMTP, POP3, HTTP, NNTP, etc.).
- Monitoring of host resources (processor load, disk usage, etc.).
- Simple plug-in design that allows administrators to develop further service checks.
- Support for redundant Nagios servers.

3.2 Installing Nagios

Install Nagios either with `zypper` or using YaST.

For further information on how to install packages see:

- Section “Using Zypper” (Chapter 4, *Managing Software with Command Line Tools*, ↑*Administration Guide*)

- Section “Installing and Removing Packages or Patterns” (Chapter 9, *Installing or Removing Software*, ↑*Deployment Guide*)

Both methods install the packages `nagios` and `nagios-www`. The later RPM package contains a Web interface for Nagios which allows, for example, to view the service status and the problem history. However, this is not absolutely necessary.

Nagios is modular designed and, thus, uses external check plug-ins to verify whether a service is available or not. It is recommended to install the `nagios-plugin` RPM package that contains ready-made check plug-ins. However, it is also possible to write your own, custom check plug-ins.

3.3 Nagios Configuration Files

Nagios organizes the configuration files as follows:

`/etc/nagios/nagios.cfg`
: Main configuration file of Nagios containing a number of directives which define how Nagios operates. See `http://nagios.sourceforge.net/docs/3_0/configmain.html` for a complete documentation.

`/etc/nagios/resource.cfg`
: Containing path to all Nagios plug-ins (default: `/usr/lib/nagios/plugins`).

`/etc/nagios/command.cfg`
: Defining the programs to be used to determine the availability of services or the commands which are used to send e-mail notifications.

`/etc/nagios/cgi.cfg`
: Contains options regarding the Nagios Web interface.

`/etc/nagios/objects/`
: A directory containing object definition files. See Section 3.3.1, “Object Definition Files” (page 49) for a more complete documentation.

3.3.1 Object Definition Files

In addition to those configuration files Nagios comes with very flexible and highly customizable configuration files called *Object Definition* configuration files. Those configuration files are very important since they define the following objects:

- Hosts
- Services
- Contacts

The flexibility lies in the fact that objects are easily enhanceable. Imagine you are responsible for a host with only one service running. However, you want to install another service on the same host machine and you want to monitor that service as well. It is possible to add another service object and assign it to the host object without huge efforts.

Right after the installation, Nagios offers default templates for object definition configuration files. They can be found at `/etc/nagios/objects`. In the following see a description on how hosts, services and contacts are added:

Example 3.1 *A Host Object Definition*

```
define host {
 name                    SRV1
 host_name               SRV1
 address                 192.168.0.1
 use                     generic-host
 check_period            24x7
 check_interval          5
 retry_interval          1
 max_check_attempts      10
 notification_period     workhours
 notification_interval   120
 notification_options    d,u,r
}
```

The `host_name` option defines a name to identify the host that has to be monitored. `address` is the IP address of this host. The `use` statement tells Nagios to inherit other configuration values from the generic-host template. `check_period` defines whether the machine has to be monitored 24x7. `check_interval` makes Nagios checking the service every 5 minutes and `retry_interval` tells Nagios to schedule host check retries at 1 minute intervals. Nagios tries to execute the checks multiple times when they do not pass. You can define how many attempts Nagios should do with the max_check_attempts directive. All configuration flags beginning with notification handle how Nagios should behave when a failure of a monitored service occurs. In the host definition above, Nagios notifies the administrators only on working hours. However, this can be adjusted with notification_period. According to notification_interval notifications will be resend every two hours. notification_options contains four different flags: d, u, r and n. They control in which state Nagios should notify the administrator. d stands for a down state, u for unreachable and r for recoveries. n does not send any notifications anymore.

Example 3.2 *A Service Object Definition*

```
define service {
 use                    generic-service
 host_name              SRV1
 service_description    PING
 contact_groups         router-admins
 check_command          check_ping!100.0,20%!500.0,60%
}
```

The first configuration directive use tells Nagios to inherit from the generic-service template. host_name is the name that assigns the service to the host object. The host itself is defined in the host object definition. A description can be set with service_description. In the example above the description is just PING. Within the contact_groups option it is possible to refer to a group of people who will be contacted on a failure of the service. This group and its members are later defined in a contact group object definition. check_command sets the program that checks whether the service is available, or not.

Example 3.3 *A Contact and Contactgroup Definition*

```
define contact {
 contact_name           admins
 use                    generic-contact
 alias                  Nagios Admin
 email                  nagios@localhost
}

define contactgroup {
 contactgroup_name      router-admins
 alias                  Administrators
 members                admins
}
```

The example listing above shows the direct contact definition and its proper contactgroup. The contact definition contains the e-mail address and the name of the person who is contacted on a failure of a service. Usually this is the responsible administrator. use inherits configuration values from the generic-contact definition.

An overview of all Nagios objects and further information about them can be found at: http://nagios.sourceforge.net/docs/3_0/objectdefinitions.html.

3.4 Configuring Nagios

Learn step-by-step how to configure Nagios to monitor different things like remote services or remote host-resources.

3.4.1 Monitoring Remote Services with Nagios

This section explains how to monitor remote services with Nagios. Proceed as follows to monitor a remote service:

Procedure 3.1 *Monitoring a Remote HTTP Service with Nagios*

1. Create a directory inside /etc/nagios/objects using mkdir. You can use any desired name for it.

2. Open /etc/nagios/nagios.conf and set cfg_dir (configuration directory) to the directory you have created in the first step.

3. Change to the configuration directory created in the first step and create the following files: hosts.cfg, services.cfg and contacts.cfg

4. Insert a host object in hosts.cfg:

```
define host {
 name                     host.name.com
 host_name                host.name.com
 address                  192.168.0.1
 use                      generic-host
 check_period             24x7
 check_interval         5
 retry_interval         1
 max_check_attempts     10
 contact_groups         admins
 notification_interval  60
 notification_options   d,u,r
}
```

5 Insert a service object in `services.cfg`:

```
define service {
 use                    generic-service
 host_name              host.name.com
 service_description    HTTP
 contact_groups         router-admins
 check_command          check_http
}
```

6 Insert a contact and contactgroup object in `contacts.cfg`:

```
define contact {
 contact_name           max-mustermann
 use                    generic-contact
 alias                  Webserver Administrator
 email                  mmustermann@localhost
}

define contactgroup {
 contactgroup_name      admins
 alias                  Administrators
 members                max-mustermann
}
```

7 Execute `rcnagios restart` to (re)start Nagios.

8 Execute `cat /var/log/nagios/nagios.log` and verify whether the following content appears:

```
[1242115343] Nagios 3.0.6 starting... (PID=10915)
[1242115343] Local time is Tue May 12 10:02:23 CEST 2009
[1242115343] LOG VERSION: 2.0
[1242115343] Finished daemonizing... (New PID=10916)
```

If you need to monitor a different remote service, it is possible to adjust `check_command` in step Step 5 (page 52). A full list of all available check programs can be obtained by executing `ls /usr/lib/nagios/plugins/check_*`

See Section 3.5, “Troubleshooting” (page 55) if an error occurred.

3.4.2 Monitoring Remote Host-Resources with Nagios

This section explains how to monitor remote host resources with Nagios.

Proceed as follows on the Nagios server:

Procedure 3.2 *Monitoring a Remote Host Resource with Nagios (Server)*

1 Install `nagios-nsca` (for example, `zypper in nagios-nsca`).

2 Set the following options in `/etc/nagios/nagios.cfg`:

```
check_external_commands=1
 accept_passive_service_checks=1
 accept_passive_host_checks=1
 command_file=/var/spool/nagios/nagios.cmd
```

3 Set the `command_file` option in `/etc/nagios/nsca.conf` to the same file defined in `/etc/nagios/nagios.conf`.

4 Add another host and service object:

```
define host {
 name                             foobar
 host_name                        foobar
 address                          10.10.4.234
 use                              generic-host
 check_period                     24x7
 check_interval                   0
 retry_interval                   1
 max_check_attempts               1
 active_checks_enabled            0
 passive_checks_enabled           1
 contact_groups                   router-admins
 notification_interval            60
 notification_options             d,u,r
}
```

```
define service {
 use                              generic-service
 host_name                        foobar
 service_description              diskcheck
 active_checks_enabled            0
 passive_checks_enabled           1
 contact_groups                   router-admins
 check_command                    check_ping
}
```

5. Execute `rcnagios restart` and `rcnsca restart`.

Proceed as follows on the client you want to monitor:

Procedure 3.3 *Monitoring a Remote Host Resource with Nagios (client)*

1. Install `nagios-nsca-client` on the host you want to monitor.

2. Write your test scripts (for example a script that checks the disk usage) like this:

```
#!/bin/bash

 NAGIOS_SERVER=10.10.4.166
 THIS_HOST=foobar

 #
 # Write own test algorithm here
 #

 # Execute On SUCCESS:
 echo "$THIS_HOST;diskcheck;0;OK: test ok" \
          | send_nsca -H $NAGIOS_SERVER -p 5667 -c
/etc/nagios/send_nsca.cfg -d ";"

 # Execute On Warning:
 echo "$THIS_HOST;diskcheck;1;Warning: test warning"  \
          | send_nsca -H $NAGIOS_SERVER -p 5667 -c
/etc/nagios/send_nsca.cfg -d ";"

 # Execute On FAILURE:
 echo "$THIS_HOST;diskcheck;2;CRITICAL: test critical"  \
          | send_nsca -H $NAGIOS_SERVER -p 5667 -c
/etc/nagios/send_nsca.cfg -d ";"
```

3. Insert a new cron entry with `crontab -e`. A typical cron entry could look like this:

```
*/5 * * * * /directory/to/check/program/check_diskusage
```

3.5 Troubleshooting

`Error: ABC 'XYZ' specified in ... '...' is not defined anywhere!`

Make sure that you have defined all necessary objects correctly. Be careful with the spelling.

`(Return code of 127 is out of bounds - plugin may be missing)`

Make sure that you have installed `nagios-plugins`.

E-mail notification does not work

Make sure that you have installed and configured a mail server like `postfix` or `exim` correctly. You can verify if your mail server works with `echo "Mail Server Test!" | mail foo@bar.com` which sends an e-mail to foo@bar.com. If this e-mail arrives, your mail server is working correctly. Otherwise, check the log files of the mail server.

3.6 For More Information

The complete Nagios documentation

`http://nagios.sourceforge.net/docs/3_0/toc.html`

Object Configuration Overview

`http://nagios.sourceforge.net/docs/3_0/configobject.html`

Object Definitions

`http://nagios.sourceforge.net/docs/3_0/objectdefinitions.html`

Nagios Plugins

`http://nagios.sourceforge.net/docs/3_0/plugins.html`

Analyzing and Managing System Log Files

4

System log file analysis is one of the most important tasks when analyzing the system. In fact, looking at the system log files should be the first thing to do when maintaining or troubleshooting a system. SUSE Linux Enterprise Server automatically logs almost everything that happens on the system in detail. Normally system log files are written in plain text and therefore can be easily read using an editor or pager. They are also parsable by scripts, allowing you to easily filter their content.

4.1 System Log Files in /var/log/

System log files are always located under the `/var/log` directory. The following list presents an overview of all system log files from SUSE Linux Enterprise Server present after a default installation. Depending on your installation scope, `/var/log` also contains log files from other services and applications not listed here. Some files and directories described below are “placeholders” and are only used, when the corresponding application is installed. Most log files are only visible for the user `root`.

`acpid`
: Log of the advanced configuration and power interface event daemon (`acpid`), a daemon to notify user-space programs of ACPI events. `acpid` will log all of it's activities, as well as the `STDOUT` and `STDERR` of any actions to syslog.

`apparmor`
: Novell AppArmor log files. See Part “Confining Privileges with Novell AppArmor” (↑*Security Guide*) for details an AppArmor.

`audit`
: Logs from the audit framework. See Part “*The Linux Audit Framework*” (↑*Security Guide*) for details.

`boot.msg`
: Log of the system init process - this file contains all boot messages from the kernel, from the boot scripts and from the services started during the boot sequence.

  Check this file to find out whether your hardware has been correctly initialized or all services have been successfully started.

`boot.omsg`
: Log of the system shutdown process - this file contains all messages issued on the last shutdown or reboot.

`ConsoleKit/*`
: Logs of the `ConsoleKit` daemon (daemon for tracking what users are logged in and how they interact with the computer.

`cups/`
: Access and error logs of the common UNIX printing system (`cups`).

`faillog`
: Database file that contains all login failures. Use the `faillog` command to view. See `man 8 faillog` for more information.

`firewall`
: Firewall logs.

`gdm/*`
: Log files from the GNOME display manager.

`krb5`
: Log files from the Kerberos network authentication system.

`lastlog`
: The lastlog file is a database which contains info on the last login of each user. Use the command `lastlog` to view. See `man 8 lastlog` for more information.

`localmessages`
Log messages of some boot scripts, for example the log of the DHCP client.

`mail*`
Mail server (`postfix`, `sendmail`) logs.

`messages`
This is the default place where all kernel and system log messages go and should be the first place (along with `/var/log/warn`) to look at in case of problems.

`NetworkManager`
NetworkManager log files

`news/*`
Log messages from a news server.

Logs from the Network Time Protocol daemon (`ntpd`).

`pk_backend_zypp`
PackageKit (with `libzypp` backend) log files.

`puppet/*`
Log files from the data center automation tool puppet.

`samba/*`
Log files from samba, the Windows SMB/CIFS file server.

`SaX.log`
Logs from sax2, the SuSE advanced X11 configuration tool.

`scpm`
Logs from the system configuration profile management (`scpm`).

`warn`
Log of all system warnings and errors. This should be the first place (along with `/var/log/messages`) to look at in case of problems.

`wtmp`
Database of all login/logout activities, runlevel changes and remote connections. Use the command `last` to view. See `man 1 last` for more information.

`xinetd.log`
Log files from the extended Internet services daemon (`xinetd`).

`Xorg.0.log`
X startup logfile. Refer to it in case you have problems starting X. Copies from previous X starts are numbered Xorg. `?`.log.

`YaST2/*`
All YaST log files.

`zypp/*`
`libzypp` log files. refer to these files for the package installation history.

`zypper.log`
Logs from the command line installer `zypper`.

4.2 Viewing and Parsing Log Files

To view log files, you can use your favorite text editor. There is also a simple YaST module for viewing `/var/log/messages`, available in the YaST Control Center under *Miscellaneous* > *System Log*.

For viewing log files in a text console, use the commands `less` or `more`. Use `head` and `tail` to view the beginning or end of a log file. To view entries appended to a log file in real time use `tail -f`. For information about how to use this tools, see their man pages.

To search for strings or regular expressions in log files use `grep`. `awk` is useful for parsing and rewriting log files.

4.3 Managing Log Files with logrotate

Logfiles under `/var/log` grow on a daily basis and quickly become very big. `logrotate` helps control the growth of these files. It is a tool for managing large amount of log files. It allows automatic rotation, removal, compression, and mailing of log files. Log files can be handled periodically (daily, weekly, or monthly) or when they exceed a particular size.

`logrotate` is usually run as a daily cron job. It does not modify any log files more than once a day, unless unless the log is to be modified because of its size, `logrotate` is being run multiple times a day, or unless the `--force` option is used.

The main configuration file of `logrotate` is `/etc/logrotate.conf`. System packages as well as programs that produce logfiles (for example `apache2`) put their own configuration files in the `/etc/logrotate.d/` directory. The content of `/etc/logrotate.d/` is included via `/etc/logrotate.conf`.

Example 4.1 *Example for /etc/logrotate.conf*

```
# see "man logrotate" for details
# rotate log files weekly
weekly

# keep 4 weeks worth of backlogs
rotate 4

# create new (empty) log files after rotating old ones
create

# use date as a suffix of the rotated file
dateext

# uncomment this if you want your log files compressed
#compress

# comment these to switch compression to use gzip or another
# compression scheme
compresscmd /usr/bin/bzip2
uncompresscmd /usr/bin/bunzip2

# RPM packages drop log rotation information into this directory
include /etc/logrotate.d
```

IMPORTANT

The `create` option pays heed to the the modes and ownerships of files specified in `/etc/permissions*`. If you modify these settings, make sure no conflicts arise.

logrotate is controlled through cron and is called daily by /etc/cron.daily/logrotate. Use /var/lib/logrotate.status to find out when a particular file was last rotated.

4.4 Monitoring Log Files with logwatch

logwatch is a customizable, pluggable log-monitoring script. It parses system logs, extracts the important information and present them in a human readable manner. To use logwatch, install the logwatch package.

logwatch can either be used at the command-line to generate on the fly reports, or via cron to regularly create custom reports. Reports can either be printed on the screen, saved to a file, or be mailed to an address specified. The latter is especially useful when automatically generating reports via cron.

The command-line syntax is easy. You basically tell logwatch for which service, time span and to which detail level to generate a report:

```
# Detailed report on all kernel messages from yesterday
logwatch --service kernel --detail High --range Yesterday --print

# Low detail report on all sshd events recorded (incl. archived logs)
logwatch --service sshd --detail Low --range All --archives --print

# Mail a report on all smartd messages from May 5th to May 7th to root@localhost
logwatch --service smartd --range 'between 5/5/2005 and 5/7/2005' \
--mailto root@localhost --print
```

The --range option has got a complex syntax—see logwatch --range help for details. A list of all services that can be queried is available with the following command:

```
ls /usr/share/logwatch/default.conf/services/ | sed 's/\.conf//g'
```

logwatch can be customized to great detail. However, the default configuration should be sufficient in most cases. The default configuration files are located under /usr/share/logwatch/default.conf/. Never change these (because they would get overwritten again with the next update), but rather place custom configuration in /etc/logwatch/conf/ (you may use the default configuration file as a template, though). A detailed HOWTO on customizing logwatch is available at /usr/share/doc/packages/logwatch/HOWTO-Customize-LogWatch. The following config files exist:

logwatch.conf
: The main configuration file. The default version is extensively commented. Each configuration option can be overwritten on the command line.

ignore.conf
: Filter for all lines that should globally be ignored by logwatch.

services/*.conf
: The service directory holds configuration files for each service you can generate a report for.

logfiles/*.conf
: Specifications on what logfiles should be parsed for each service.

Part III. Kernel Monitoring

SystemTap—Filtering and Analyzing System Data

5

SystemTap provides a command line interface and a scripting language to examine the activities of a running Linux system, particularly the kernel, in fine detail. SystemTap scripts are written in the SystemTap scripting language, are then compiled to C-code kernel modules and inserted into the kernel. The scripts can be designed to extract, filter and summarize data, thus allowing the diagnosis of complex performance problems or functional problems. SystemTap provides information similar to the output of tools like `netstat`, `ps`, `top`, and `iostat`. However, more filtering and analysis options can be used for the collected information.

5.1 Installation

There are two different setups for using SystemTap: You can either compile the SystemTap script and insert the resulting kernel module on the same machine (this requires the machine to have the kernel debugging information installed) or you can make use of the SystemTap client on one machine and the SystemTap compile server on another machine. This allows you to compile a SystemTap module on a machine other than the one on which it will be run, as long as client and server are compatible.

Depending on your setup, check the sections below for an overview of the packages you need. To get access to the man pages and to a helpful collection of example SystemTap scripts for various purposes, additionally install the `systemtap-doc` package.

5.1.1 SystemTap Client

For this setup, you need:

- `systemtap`

- `systemtap-client`

5.1.2 SystemTap Compile Server

For a SystemTap compile server, you need the following SystemTap packages:

- `systemtap`

- `systemtap-server`

As SystemTap needs information about the kernel, some kernel-related packages must be installed on the SystemTap compile server in addition to the SystemTap packages. For each kernel you want to probe with SystemTap, you need to install a set of the following packages that exactly matches the kernel version and flavor (indicated by `*` in the list below):

- `kernel-*-debuginfo`

- `kernel-*-devel`

- `kernel-source-*`

- `gcc`

IMPORTANT: Repository for Packages with Debugging Information

If you subscribed your system for online updates, you can find “debuginfo” packages in the `*-Debuginfo-Updates` online installation repository relevant for SUSE Linux Enterprise Server 11 SP1. Use YaST to enable the repository.

To check if all needed packages are installed and if SystemTap is ready to use, perform the quick test run described in Section 5.2, “Initial Test” (page 69).

5.2 Initial Test

For a quick test run that probes the currently used kernel, execute the following command as `root`:

```
stap -v -e 'probe vfs.read {printf("read performed\n"); exit()}'
```

It runs a script and returns an output. If the output is similar to the following, SystemTap is successfully deployed and ready to use:

```
Pass 1: ❶ parsed user script and 59 library script(s) in 80usr/0sys/214real
ms.
Pass 2: ❷ analyzed script: 1 probe(s), 11 function(s), 2 embed(s), 1 global(s)
 in 140usr/20sys/412real ms.
Pass 3: ❸ translated to C
into"/tmp/stapDwEk76/stap_1856e21ea1c246da85ad8c66b4338349_4970.c" in
160usr/0sys/408real ms.
Pass 4: ❹ compiled C into "stap_1856e21ea1c246da85ad8c66b4338349_4970.ko" in
 2030usr/360sys/10182real ms.
Pass 5: ❺ starting run.
read performed
Pass 5: run completed in 10usr/20sys/257real ms.
```

❶ Checks the script against the existing tapset library in `/usr/share/systemtap/tapset/` for any tapsets used. Tapsets are scripts that form a library of pre-written probes and functions that can be used in SystemTap scripts.

❷ Examines the script for its components.

❸ Translates the script to C. Runs the system C compiler to create a kernel module from it. Both the resulting C code (`*.c`) and the kernel module (`*.ko`) are stored in the SystemTap cache, `~/.systemtap`.

❹ Loads the module and enables all the probes (events and handlers) in the script by hooking into the kernel. The event being probed is a Virtual File System (VFS) read. As the event occurs on any processor, a valid handler is executed (prints the text `read performed`) and closed with no errors.

❺ After the SystemTap session is terminated, the probes are disabled, and the kernel module is unloaded.

In case any error messages appear during the test, check the output for hints about any missing packages and make sure they are installed correctly. Rebooting and loading the appropriate kernel may also be needed.

5.3 Conceptual Overview

Each time you run a SystemTap script, a SystemTap session is started. A number of passes are done on the script before it is allowed to run, at which point the script is compiled into a kernel module and loaded. In case the script has already been executed before and no changes regarding any components have occurred (for example, regarding compiler version, kernel version, library path, script contents), SystemTap does not compile the script again, but uses the `*.c` and `*.ko` data stored in the SystemTap cache (`~/.systemtap`). The module is unloaded when the tap has finished running. For an example, see the test run in Section 5.1, “Installation” (page 67) and the respective explanation.

5.3.1 SystemTap Scripts

SystemTap usage is based on SystemTap scripts (`*.stp`). They tell SystemTap which type of information to collect, and what to do once that information is collected. The scripts are written in the SystemTap scripting language that is similar to AWK and C. For the language definition, see `http://sourceware.org/systemtap/langref/`.

The essential idea behind a SystemTap script is to name `events`, and to give them `handlers`. When SystemTap runs the script, it monitors for certain events. When an event occurs, the Linux kernel runs the handler as a sub-routine, then resumes. Thus, events serve as the triggers for handlers to run. Handlers can record specified data and print it in a certain manner.

The SystemTap language only uses a few data types (integers, strings, and associative arrays of these), and full control structures (blocks, conditionals, loops, functions). It has a lightweight punctuation (semicolons are optional) and does not need detailed declarations (types are inferred and checked automatically).

For more information about SystemTap scripts and their syntax, refer to Section 5.4, “Script Syntax” (page 73) and to the `stapprobes` and `stapfuncs` man pages, that are available with the `systemtap-doc` package.

5.3.2 Tapsets

Tapsets are a library of pre-written probes and functions that can be used in SystemTap scripts. When a user runs a SystemTap script, SystemTap checks the script's probe events and handlers against the tapset library. SystemTap then loads the corresponding probes and functions before translating the script to C. Like SystemTap scripts themselves, tapsets use the filename extension `*.stp`.

However, unlike SystemTap scripts, tapsets are not meant for direct execution—they constitute the library from which other scripts can pull definitions. Thus, the tapset library is an abstraction layer designed to make it easier for users to define events and functions. Tapsets provide useful aliases for functions that users may want to specify as an event (knowing the proper alias is mostly easier than remembering specific kernel functions that might vary between kernel versions).

5.3.3 Commands and Privileges

The main commands associated with SystemTap are `stap` and `staprun`. To execute them, you either need `root` privileges or must be a member of the `stapdev` or `stapusr` group.

`stap`

SystemTap front-end. Runs a SystemTap script (either from file, or from standard input). It translates the script into C code, compiles it, and loads the resulting kernel module into a running Linux kernel. Then, the requested system trace or probe functions are performed.

`staprun`

SystemTap back-end. Loads and unloads kernel modules produced by the SystemTap front-end.

For a list of options for each command, use `--help`. For details, refer to the `stap` and the `staprun` man pages.

There is a particular reason why SystemTap is split into a front-end and a back-end: This allows you to compile a SystemTap script on a (development) machine that has the kernel debugging information (needed to compile the script) and then transfer the resulting kernel module to a (production) machine that does not have any development tools or kernel debugging information installed.

To avoid giving `root` access to users just for running SystemTap, you can make use of the following SystemTap groups. They are not available by default on SUSE Linux Enterprise, but you can create the groups and modify the access rights accordingly.

`stapdev`
: Members of this group can run SystemTap scripts with `stap`, or run SystemTap instrumentation modules with `staprun`. As running `stap` involves compiling scripts into kernel modules and loading them into the kernel, members of this group still have effective `root` access.

`stapusr`
: Members of this group are only allowed to run SystemTap instrumentation modules with `staprun`. In addition, they can only run those modules from `/lib/modules/`*`kernel_version`*`/systemtap/`. This directory must be owned by `root` and must only be writable for the `root` user.

5.3.4 Important Files and Directories

The following list gives an overview of the SystemTap main files and directories.

`/lib/modules/`*`kernel_version`*`/systemtap/`
: Holds the SystemTap instrumentation modules.

`/usr/share/systemtap/tapset/`
: Holds the standard library of tapsets.

`/usr/share/doc/packages/systemtap/examples`
: Holds a number of example SystemTap scripts for various purposes. Only available if the `systemtap-doc` package is installed.

`~/.systemtap`
: Data directory for cached SystemTap files.

`/tmp/stap*`
: Temporary directory for SystemTap files, including translated C code and kernel object.

5.4 Script Syntax

SystemTap scripts consist of the following two components:

SystemTap Events (Probe Points) (page 75)
: Name the kernel events at the associated handler should be executed. Examples for events are entering or exiting a certain function, a timer expiring, or starting or terminating a session.

SystemTap Handlers (Probe Body) (page 76)
: Series of script language statements that specify the work to be done whenever a certain event occurs. This normally includes extracting data from the event context, storing them into internal variables, or printing results.

An event and its corresponding handler is collectively called a `probe`. SystemTap events are also called `probe points`. A probe's handler is also referred to as `probe body`.

Comments can be inserted anywhere in the SystemTap script in various styles: using either `#`, `/* */`, or `//` as marker.

5.4.1 Probe Format

A SystemTap script can have multiple probes. They must be written in the following format:

```
probe event {statements}
```

Each probe has a corresponding statement block. This statement block must be enclosed in `{ }` and contains the statements to be executed per event.

***Example 5.1** Simple SystemTap Script*

The following example shows a simple SystemTap script.

```
probe❶ begin❷
{❸
   printf❹ ("hello world\n")❺
   exit ()❻
}❼
```

❶ Start of the probe.

❷ Event `begin` (the start of the SystemTap session).

❸ Start of the handler definition, indicated by `{`.

❹ First function defined in the handler: the `printf` function.

❺ String to be printed by the `printf` function, followed by a line break (`/n`).

❻ Second function defined in the handler: the `exit()` function. Note that the SystemTap script will continue to run until the `exit()` function executes. If you want to stop the execution of the script before, stop it manually by pressing Ctrl + C.

❼ End of the handler definition, indicated by `}`.

The event `begin` ❷ (the start of the SystemTap session) triggers the handler enclosed in `{ }`, in this case the `printf` function ❹ which prints `hello world` followed by a new line ❺, then exits.

If your statement block holds several statements, SystemTap executes these statements in sequence—you do not need to insert special separators or terminators between multiple statements. A statement block can also be nested within another statement blocks. Generally, statement blocks in SystemTap scripts use the same syntax and semantics as in the C programming language.

5.4.2 SystemTap Events (Probe Points)

SystemTap supports a number of built-in events.

The general event syntax is a dotted-symbol sequence. This allows a breakdown of the event namespace into parts. Each component identifier may be parametrized by a string or number literal, with a syntax like a function call. A component may include a `*` character, to expand to other matching probe points. A probe point may be followed by a `?` character, to indicate that it is optional, and that no error should result if it fails to expand. Alternately, a probe point may be followed by a `!` character to indicate that it is both optional and sufficient.

SystemTap supports multiple events per probe—they need to be separated by a comma (`,`). If multiple events are specified in a single probe, SystemTap will execute the handler when any of the specified events occur.

In general, events can be classified into the following categories:

- Synchronous events: Occur when any process executes an instruction at a particular location in kernel code. This gives other events a reference point (instruction address) from which more contextual data may be available.

  An example for a synchronous event is `vfs.`*`file_operation`*: The entry to the *`file_operation`* event for Virtual File System (VFS). For example, in Section 5.2, “Initial Test” (page 69), `read` is the *`file_operation`* event used for VFS.

- Asynchronous events: Not tied to a particular instruction or location in code. This family of probe points consists mainly of counters, timers, and similar constructs.

  Examples for asynchronous events are: `begin` (start of a SystemTap session—as soon as a SystemTap script is run, `end` (end of a SystemTap session), or timer events. Timer events specify a handler to be executed periodically, like `example timer.s(`*`seconds`*`)`, or `timer.ms(`*`milliseconds`*`)`.

  When used in conjunction with other probes that collect information, timer events allow you to print out periodic updates and see how that information changes over time.

Example 5.2 *Probe with Timer Event*

For example, the following probe would print the text "hello world" every 4 seconds:

```
probe timer.s(4)
{
   printf("hello world\n")
}
```

For detailed information about supported events, refer to the `stapprobes` man page. The *See Also* section of the man page also contains links to other man pages that discuss supported events for specific subsystems and components.

5.4.3 SystemTap Handlers (Probe Body)

Each SystemTap event is accompanied by a corresponding handler defined for that event, consisting of a statement block.

Functions

If you need the same set of statements in multiple probes, you can place them in a function for easy reuse. Functions are defined by the keyword `function` followed by a name. They take any number of string or numeric arguments (by value) and may return a single string or number.

```
function function_name(arguments) {statements}
probe event {function_name(arguments)}
```

The statements in *`function_name`* are executed when the probe for *`event`* executes. The *`arguments`* are optional values passed into the function.

Functions can be defined anywhere in the script. They may take any

One of the functions needed very often was already introduced in Example 5.1, "Simple SystemTap Script" (page 74): the `printf` function for printing data in a formatted way. When using the `printf` function, you can specify how arguments should be printed by using a format string. The format string is included in quotation marks and can contain further format specifiers, introduced by a `%` character.

Which format strings to use depends on your list of arguments. Format strings can have multiple format specifiers—each matching a corresponding argument. Multiple arguments can be separated by a comma.

***Example 5.3** printf Function with Format Specifiers*

```
printf ("❶%s❷(%d❸) open\n❹", execname(), pid())
```

❶ Start of the format string, indicated by ".

❷ String format specifier.

❸ Integer format specifier.

❹ End of the format string, indicated by ".

The example above would print the current executable name (`execname()`) as string and the process ID (`pid()`) as integer in brackets, followed by a space, then the word `open` and a line break:

```
[...]
vmware-guestd(2206) open
hald(2360) open
[...]
```

Apart from the two functions `execname()` and `pid()`) used in Example 5.3, "`printf` Function with Format Specifiers" (page 77), a variety of other functions can be used as `printf` arguments.

Among the most commonly used SystemTap functions are the following:

tid()
: ID of the current thread.

pid()
: Process ID of the current thread.

uid()
: ID of the current user.

cpu()
: Current CPU number.

execname()
: Name of the current process.

gettimeofday_s()
: Number of seconds since UNIX epoch (January 1, 1970).

ctime()
: Convert time into a string.

pp()
: String describing the probe point currently being handled.

thread_indent()
: Useful function for organizing print results. It (internally) stores an indentation counter for each thread (`tid()`). The function takes one argument, an indentation delta, indicating how many spaces to add or remove from the thread's indentation counter. It returns a string with some generic trace data along with an appropriate number of indentation spaces. The generic data returned includes a timestamp (number of microseconds since the initial indentation for the thread), a process name, and the thread ID itself. This allows you to identify what functions were called, who called them, and how long they took.

  Call entries and exits often do not immediately precede each other (otherwise it would be easy to match them). In between a first call entry and its exit, usually a number of other call entries and exits are made. The indentation counter helps you match an entry with its corresponding exit as it indents the next function call in case it is *not* the exit of the previous one. For an example SystemTap script using `thread_indent()` and the respective output, refer to the *SystemTap Tutorial*: `http://sourceware.org/systemtap/tutorial/Tracing.html#fig:socket-trace`.

For more information about supported SystemTap functions, refer to the `stapfuncs` man page.

Other Basic Constructs

Apart from functions, you can use several other common constructs in SystemTap handlers, including variables, conditional statements (like `if`/`else`, `while` loops, `for` loops, arrays or command line arguments.

5.5 Example Scripts

If you have installed the `systemtap-doc` package, you can find a number of useful SystemTap example scripts in `/usr/share/doc/packages/systemtap/examples`.

This section describes a rather simple example script in more detail: `/usr/share/doc/packages/systemtap/examples/network/tcp_connections.stp`.

Example 5.4 *Monitoring Incoming TCP Connections with tcp_connections.stp*

```
#! /usr/bin/env stap

probe begin {
  printf("%6s %16s %6s %6s %16s\n",
         "UID", "CMD", "PID", "PORT", "IP_SOURCE")
}

probe kernel.function("tcp_accept").return?,
      kernel.function("inet_csk_accept").return? {
  sock = $return
  if (sock != 0)
    printf("%6d %16s %6d %6d %16s\n", uid(), execname(), pid(),
           inet_get_local_port(sock), inet_get_ip_source(sock))
}
```

This SystemTap script monitors the incoming TCP connections and helps to identify unauthorized or unwanted network access requests in real time. It shows the following information for each new incoming TCP connection accepted by the computer:

- User ID (UID)
- Command accepting the connection (CMD)
- Process ID of the command (PID)
- Port used by the connection (PORT)
- IP address from which the TCP connection originated (IP_SOUCE)

To run the script, execute

```
stap /usr/share/doc/packages/systemtap/examples/network/tcp_connections.stp
```

and follow the output on the screen. To manually stop the script, press Ctrl + C.

5.6 For More Information

This chapter only provides a short SystemTap overview. Refer to the following links for more information about SystemTap:

`http://sourceware.org/systemtap/`
SystemTap project home page.

`http://sourceware.org/systemtap/wiki/`
Huge collection of useful information about SystemTap, ranging from detailed user and developer documentation to reviews and comparisons with other tools, or Frequently Asked Questions and tips. Also contains collections of SystemTap scripts, examples and usage stories and lists recent talks and papers about SystemTap.

`http://sourceware.org/systemtap/documentation.html`
Features a *SystemTap Tutorial*, a *SystemTap Beginner's Guide*, a *Tapset Developer's Guide*, and a *SystemTap Language Reference* in PDF and HTML format. Also lists the relevant man pages.

You can also find the SystemTap language reference and SystemTap tutorial in your installed system under `/usr/share/doc/packages/systemtap`. Example SystemTap scripts are available from the `example` subdirectory.

6 Kernel Probes

Kernel probes are a set of tools to collect Linux kernel debugging and performance information. Developers and system administrators usually use them either to debug the kernel, or to find system performance bottlenecks. The reported data can then be used to tune the system for better performance.

You can insert these probes into any kernel routine, and specify a handler to be invoked after a particular break-point is hit. The main advantage of kernel probes is that you no longer need to rebuild the kernel and reboot the system after you make changes in a probe.

To use kernel probes, you typically need to write or obtain a specific kernel module. Such module includes both the *init* and the *exit* function. The init function (such as `register_kprobe()`) registers one or more probes, while the exit function unregisters them. The registration function defines *where* the probe will be inserted and *which handler* will be called after the probe is hit. To register or unregister a group of probes at one time, you can use relevant `register_`*`<probe_type>`*`probes()` or `unregister_`*`<probe_type>`*`probes()` functions.

Debugging and status messages are typically reported with the `printk` kernel routine. `printk` is a kernel-space equivalent of a user-space `printf` routine. For more information on `printk`, see Logging kernel messages [`http://www.win.tue.nl/~aeb/linux/lk/lk-2.html#ss2.8`]. Normally, you can view these messages by inspecting `/var/log/messages` or `/var/log/syslog`. Commenting, file not ready For more information on log files, see Chapter 4, *Analyzing and Managing System Log Files* (page 57).

6.1 Supported Architectures

Kernel probes are *fully* implemented on the following architectures:

- i386
- x86_64 (AMD-64, EM64T)
- ppc64
- arm
- ppc

Kernel probes are *partially* implemented on the following architectures:

- ia64 (does not support probes on instruction `slot1`)
- sparc64 (return probes not yet implemented)

6.2 Types of Kernel Probes

There are three types of kernel probes: *kprobes*, *jprobes*, and *kretprobes*. Kretprobes are sometimes referred to as *return probes*. You can find vivid source code examples of all three type of kernel probes in the `/usr/src/linux/samples/kprobes/` directory (package `kernel-source`).

6.2.1 Kprobe

Kprobe can be attached to any instruction in the Linux kernel. When it is registered, it inserts a break-point at the first bytes of the probed instruction. When the processor hits this break-point, the processor registers are saved, and the processing passes to kprobes. First, a *pre-handler* is executed, then the probed instruction is stepped, and, finally a *post-handler* is executed. The control is then passed to the instruction following the probe point.

6.2.2 Jprobe

Jprobe is implemented through the kprobe mechanism. It is inserted on a function's entry point and allows direct access to the arguments of the function which is being probed. Its handler routine must have the same argument list and return value as the probed function. It also has to end by calling the `jprobe_return()` function.

When jprobe is hit, the processor registers are saved, and the instruction pointer is directed to the jprobe handler routine. The control then passes to the handler with the same register contents as the function being probed. Finally, the handler calls the `jprobe_return()` function, and switches the control back to the control function.

In general, you can insert multiple probes on one function. Jprobe is, however, limited to only one instance per function.

6.2.3 Return Probe

Return probes are also implemented through kprobes. When the `register_kretprobe()` function is called, a kprobe is attached to the entry of the probed function. After hitting the probe, the Kernel probes mechanism saves the probed function return address and calls a user-defined return handler. The control is then passed back to the probed function.

Before you call `register_kretprobe()`, you need to set a `maxactive` argument, which specifies how many instances of the function can be probed at the same time. If set too low, you will miss a certain number of probes.

6.3 Kernel probes API

Kprobe's programming interface consists of functions, which are used to register and unregister all used kernel probes, and associated probe handlers. For a more detailed description of these functions and their arguments, see the information sources in Section 6.5, “For More Information” (page 85).

`register_kprobe()`
: Inserts a break-point on a specified address. When the break-point is hit, the `pre_handler` and `post_handler` are called.

`register_jprobe()`
: Inserts a break-point in the specified address. The address has to be the address of the first instruction of the probed function. When the break-point is hit, the specified handler is run. The handler should have the same argument list and return type as the probed.

`register_kretprobe()`
: Inserts a return probe for the specified function. When the probed function returns, a specified handler is run. This function returns 0 on success, or a negative error number on failure.

`unregister_kprobe()`, `unregister_jprobe()`, `unregister_kretprobe()`
: Removes the specified probe. You can use it any time after the probe has been registered.

`register_kprobes()`, `register_jprobes()`, `register_kretprobes()`
: Inserts each of the probes in the specified array.

`unregister_kprobes()`, `unregister_jprobes()`, `unregister_kretprobes()`
: Removes each of the probes in the specified array.

`disable_kprobe()`, `disable_jprobe()`, `disable_kretprobe()`
: Disables the specified probe temporarily.

`enable_kprobe()`, `enable_jprobe()`, `enable_kretprobe()`
: Enables temporarily disabled probes.

6.4 Debugfs Interface

With recent Linux kernels, the Kernel probes instrumentation uses the kernel debugfs interface. It helps you list all registered probes and globally switch all the probes on or off.

6.4.1 How to List Registered Kernel Probes

The list of all currently registered kprobes is in the `/sys/kernel/debug/kprobes/list` file.

```
saturn.example.com:~ # cat /sys/kernel/debug/kprobes/list
c015d71a  k  vfs_read+0x0   [DISABLED]
c011a316  j  do_fork+0x0
c03dedc5  r  tcp_v4_rcv+0x0
```

The first column lists the address in the kernel where the probe is inserted. The second column prints the type of the probe: `k` for kprobe, `j` for jprobe, and `r` for return probe. The third column specifies the symbol, offset and optional module name of the probe. The following optional columns include the status information of the probe. If the probe is inserted on a virtual address which is not valid anymore, it is marked with `[GONE]`. If the probe is temporarily disabled, it is marked with `[DISABLED]`.

6.4.2 How to Switch All Kernel Probes On or Off

The `/sys/kernel/debug/kprobes/enabled` file represents a switch with which you can globally and forcibly turn on or off all the registered kernel probes. To turn them off, simply enter

```
echo "0" > /sys/kernel/debug/kprobes/enabled
```

on the command line as `root`. To turn them on again, enter

```
echo "1" > /sys/kernel/debug/kprobes/enabled
```

Note that this way you do not change the status of the probes. If a probe is temporarily disabled, it will not be enabled automatically but will remain in the `[DISABLED]` state after entering the latter command.

6.5 For More Information

To learn more about kernel probes, look at the following sources of information:

- Thorough but more technically oriented information about kernel probes is in /usr/src/linux/Documentation/kprobes.txt (package kenrel-source).

- Examples of all three types of probes (together with related Makefile) are in the /usr/src/linux/samples/kprobes/ directory (package kenrel-source).

- In-depth information about Linux kernel modules and printk kernel routine is in The Linux Kernel Module Programming Guide [http://tldp.org/LDP/lkmpg/2.6/html/lkmpg.html]

- Practical but slightly outdated information about practical use of kernel probes is in Kernel debugging with Kprobes [http://www.ibm.com/developerworks/library/l-kprobes.html]

Perfmon2—Hardware-Based Performance Monitoring 7

Perfmon2 is a standardized, generic interface to access the performance monitoring unit (PMU) of a processor. It is portable across all PMU models and architectures, supports system-wide and per-thread monitoring, counting and sampling.

7.1 Conceptual Overview

The following subsections give you a brief overview about Perfmon.

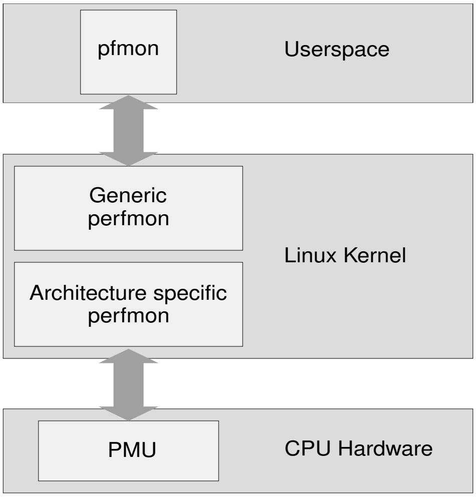
7.1.1 Perfmon2 Structure

Performance monitoring is “the action of collecting information related to how an application or system performs”. The information can be obtained from the code or the CPU/chipset.

Modern processors contain a performance monitoring unit (PMU). The design and functionality of a PMU is CPU specific: for example, the number of registers, counters and features supported will vary by CPU implementation.

The Perfmon interface is designed to be generic, flexible and extensible. It can monitor at the program (thread) or system levels. In either mode, it is possible to count or sample your profile information. This uniformity makes it easier to write portable tools. Figure 7.1, “Architecture of perfmon2” (page 88) gives an overview.

Figure 7.1 *Architecture of perfmon2*



Each PMU model consists of a set of registers: the performance monitor configuration (PMC) and the performance monitor data (PMD). PMCs can only be write, but both can be read. These registers store configuration information and data.

7.1.2 Sampling and Counting

Perfmon2 supports two modes where you can run your profiling: sampling or couting.

Sampling is usually expressed by an interval of time (time based) or a occurance of a definied number of events (event based). Perfmon indirectly support time based sampling by using an event based sample with constant correlation to time (for example, `unhalted_reference_cycles`.)

In contrast, *Counting* is expressed in terms of a number of occurances of an event.

Both methods store their information into a *sample*. This sample contains information about, for example, where a thread was or instruction pointers.

The following example demonstrates the counting of the event `CPU_OP_CYCLES` and also the sampling of this event, generating a sample per 100000 occurances of the event:

```
pfmon --no-cmd-output -e CPU_OP_CYCLES_ALL /bin/ls
1306604 CPU_OP_CYCLES_ALL
```

The following command gives the count of a specific function and the procentual amount of the total cycles:

```
pfmon --no-cmd-output --short-smpl-periods=100000   -e CPU_OP_CYCLES_ALL
/bin/ls
# results for [28119:28119<-[28102]] (/bin/ls)
# total samples          : 12
# total buffer overflows : 0
#
#                   event00
#     counts   %self    %cum          code addr
           1   8.33%   8.33% 0x2000000000007180
           1   8.33%  16.67% 0x20000000000195a0
           1   8.33%  25.00% 0x2000000000019260
           1   8.33%  33.33% 0x2000000000014e60
           1   8.33%  41.67% 0x20000000001f38c0
           1   8.33%  50.00% 0x20000000001ea481
           1   8.33%  58.33% 0x200000000020b260
           1   8.33%  66.67% 0x2000000000203490
           1   8.33%  75.00% 0x2000000000203360
           1   8.33%  83.33% 0x2000000000203440
           1   8.33%  91.67% 0x4000000000002690
           1   8.33% 100.00% 0x20000000001cfdf1
```

7.2 Installation

In order to use Perfmon2, first check the following preconditions:

SUSE Linux Enterprise 11

Supported architectures are IA64, x86_64. The package `perf` (Performance Counters for Linux) is now the supported tools for x86 and PPC64

SUSE Linux Enterprise 11 SP1

Supported architecture is IA64 only

The `pfmon` on SUSE Linux Enterprise11 supports the following processors (taken from `/usr/share/doc/packages/pfmon/README`):

Table 7.1 *Supported Processors*

Model	Processor
Intel IA-64	Itanium (Merced), Itanium 2 (McKinley, Madison, Deerfield), Itanium 2 9000/9100 (Montecito, Montvale) and Generic
AMD X86	Opteron (K8, fam 10h)
Intel X86	Intel P6 (Pentium II, Pentium Pro, Pentium III, Pentium M); Yonah (Core Duo, Core Solo); Netburst (Pentium 4, Xeon); Core (Merom, Penryn, Dunnington) Core 2 and Quad; Atom; Nehalem; architectural perfmon v1, v2, v3

Install the following packages depending on your architecture:

Table 7.2 *Needed Packages*

Architecture	Packages
ia64	`pfmon`

7.3 Using Perfmon

In order to use Perfmon, use the command line tool `pfmon` to get all your information.

NOTE: Mutual Exclusion of Perfmon and OProfile Sessions

On x86 architectures it is not possible to start a Perfmon session and a OProfile session. Only one can be run at the same time.

7.3.1 Getting Event Information

To get a list of supported events, use the option `-l` from `pfmon` to list them. Keep in mind, this list depends on the host PMU:

```
pfmon -l
ALAT_CAPACITY_MISS_ALL
ALAT_CAPACITY_MISS_FP
ALAT_CAPACITY_MISS_INT
BACK_END_BUBBLE_ALL
BACK_END_BUBBLE_FE
BACK_END_BUBBLE_L1D_FPU_RSE
...
CPU_CPL_CHANGES_ALL
CPU_CPL_CHANGES_LVL0
CPU_CPL_CHANGES_LVL1
CPU_CPL_CHANGES_LVL2
CPU_CPL_CHANGES_LVL3
CPU_OP_CYCLES_ALL
CPU_OP_CYCLES_QUAL
CPU_OP_CYCLES_HALTED
DATA_DEBUG_REGISTER_FAULT
DATA_DEBUG_REGISTER_MATCHES
DATA_EAR_ALAT
...
```

Get an explanation of these entries with the option `-i` and the event name:

```
pfmon -i CPU_OP_CYCLES_ALL
Name     : CPU_OP_CYCLES_ALL
Code     : 0x12
Counters : [ 4 5 6 7 8 9 10 11 12 13 14 15 ]
Desc     : CPU Operating Cycles -- All CPU cycles counted
Umask    : 0x0
EAR      : None
ETB      : No
MaxIncr  : 1  (Threshold 0)
Qual     : None
Type     : Causal
Set      : None
```

7.3.2 Enabling System Wide Sessions

Use the `--system-wide` option to enable monitoring all processes that execute on a specific CPU or sets of CPUs. You do not have to be `root` to do so; per default, user level is turned on for all events (option `-u`).

It is possible that one system wide session can run concurrently with another system wide sessions as long as they do not monitor the same set of CPUs. However, you can not run a system wide session with any per-thread sessions together.

The following examples are taken from a Itanium IA64 Montecito processor. To execute a system-wide session, perform the following procedure:

1. Detect your CPU set:

   ```
   pfmon -v --system-wide
   ...
   selected CPUs (2 CPU in set, 2 CPUs online): CPU0 CPU1
   ```

2. Delimit your session. The following list describes options which are used in the examples below (refer to the man page for more details):

   `-e/--events`
      Profile only selected events. See Section 7.3.1, “Getting Event Information” (page 90) for how to get a list.

   `--cpu-list`
      Specifies the list of processors to monitor. Without this options, all available processors are monitored.

   `-t/--session-timeout`
      Specifies the duration of the monitor session expressed in seconds.

   Use one of the three methods to start your profile session.

   - Use the default events:

     ```
     pfmon --cpu-list=0-2 --system-wide -k -e
     CPU_OP_CYCLES_ALL,IA64_INST_RETIRED
     <press ENTER to stop session>
     CPU0                       7670609 CPU_OP_CYCLES_ALL
     CPU0                       4380453 IA64_INST_RETIRED
     CPU1                       7061159 CPU_OP_CYCLES_ALL
     CPU1                       4143020 IA64_INST_RETIRED
     CPU2                       7194110 CPU_OP_CYCLES_ALL
     CPU2                       4168239 IA64_INST_RETIRED
     ```

   - Use a timeout expressed in seconds:

     ```
     pfmon --cpu-list=0-2 --system-wide --session-timeout=10 -k -e
     CPU_OP_CYCLES_ALL,IA64_INST_RETIRED
     <session to end in 10 seconds>
     CPU0                      69263547 CPU_OP_CYCLES_ALL
     CPU0                      38682141 IA64_INST_RETIRED
     CPU1                      87189093 CPU_OP_CYCLES_ALL
     CPU1                      54684852 IA64_INST_RETIRED
     ```

```
CPU2                      64441287 CPU_OP_CYCLES_ALL
CPU2                      37883915 IA64_INST_RETIRED
```

- Execute a command. The session is automatically started when the program starts and automatically stopped when the program is finished:

  ```
  pfmon --cpu-list=0-1 --system-wide -u -e
  CPU_OP_CYCLES_ALL,IA64_INST_RETIRED -- ls -l /dev/null
  crw-rw-rw- 1 root root 1, 3 27. Mär 03:30 /dev/null
  CPU0                         38925 CPU_OP_CYCLES_ALL
  CPU0                          7510 IA64_INST_RETIRED
  CPU1                          9825 CPU_OP_CYCLES_ALL
  CPU1                          1676 IA64_INST_RETIRED
  ```

3 Press the Enter key to stop a session:

4 If you want to aggregate counts, use the -aggr option after the previous command:

```
pfmon --cpu-list=0-1 --system-wide -u -e
CPU_OP_CYCLES_ALL,IA64_INST_RETIRED --aggr
<press ENTER to stop session>

                     52655 CPU_OP_CYCLES_ALL
                     53164 IA64_INST_RETIRED
```

7.3.3 Monitoring Running Tasks

Perfmon can also monitor an existing thread. This is useful for monitoring system daemons or programs which take a long time to start. First determine the process ID you wish to monitor:

```
ps ax | grep foo
10027  pts/1    R     2:23  foo
```

Use the found PID for the --attach-task option of pfmon:

```
pfmon --attach-task=10027
3682190 CPU_OP_CYCLES_ALL
```

7.4 Retrieving Metrics From DebugFS

Perfmon can collect statistics which are exported through the debug interface. The metrics consists of mostly aggregated counts and durations.

Access the data through mounting the debug file system as `root` under `/sys/kernel/debug`

The data is located under `/sys/kernel/debug/perfmon/` and organized per CPU. Each CPU contains a set of metrics, accessible as ASCII file. The following data is taken from the `/usr/src/linux/Documentation/perfmon2-debugfs.txt`:

Table 7.3 Read-Only Files in /sys/kernel/debug/perfmon/cpu/*

File	Description
`ctxswin_count`	Number of PMU context switch in
`ctxswin_ns`	Number of nanoseconds spent in the PMU context switch in routine `Average cost of the PMU context switch in = ctxswin_ns / ctxswin_count`
`ctxswout_count`	Number of PMU context switch out
`ctxswout_ns`	Number of nanoseconds spend in the PMU context switch out routine `Average cost of the PMU context switch out = ctxswout_ns / ctxswout_count`
`fmt_handler_calls`	Number of calls to the sampling format routine that handles PMU interrupts (typically the routine that recors a sample)
`fmt_handler_ns`	Number of nanoseconds spent in the routine that handle PMU interrupt in the sampling format

File	Description

	`Average time spent in this routine =`
	`  fmt_handler_ns / fmt_handler_calls`

handle_timeout_count	Number of times the pfm_handle_timeout() routine is called (used for timeout-based set switching)
handle_work_count	Number of times pfm_handle_work() routine is called
ovl_intr_all_count	Number of PMU interrupts received by the kernel
ovfl_intr_nmi_count	Number of non maskeable interrupts (NMI) received by the kernel from perfmon (only for X86 hardware)

ovfl_intr_ns	Number of nanoseconds spent in the perfmon2 PMU interrupt handler routine.
	`Average time to handle one PMU interrupt =`
	`  ovfl_intr_ns / ovfl_intr_all_count`

ovfl_intr_regular _count	Number of PMU interrupts which are actually processed by the perfmon interrupt handler
ovfl_intr_replay _count	Number of PMU interrupts which were replayed on the context switch in or on event set switching
perfom_intr_spurious _count, ovfl_intr _spurious_count	Number of PMU interrupts which were dropped because there was no active context
pfm_restart_count	Number of times pfm_restart() is called
reset_pmds_count	Number of times pfm_reset_pmds() is called
set_switch_count	Number of event set switches

File	Description
`set_switch_ns`	Number of nanoseconds spent in the set switching rountine `Average cost of switching sets = set_switch_ns / set_switch_count`

This might be useful to compare your metrics before and after the perfmon run. For example, collect your data first:

```
for i in /sys/kernel/debug/perfmon/cpu0/*; do
   echo "$i:"; cat $i
done >> pfmon-before.txt
```

Run your performance monitoring, maybe restrict it to a specific CPU:

```
pfmon --cpu-list=0 ...
```

Collect your data again:

```
for i in /sys/kernel/debug//perfmon/cpu0/*; do
   echo "$i:"; cat $i
done >> pfmon-after.txt
```

Compare these two files:

```
diff -u pfmon-before.txt pfmon-after.txt
```

7.5 For More Information

This chapter only provides a short overview. Refer to the following links for more information:

http://perfmon2.sourceforge.net/
: The project home page.

http://www.iop.org/EJ/article/1742-6596/119/4/042017/jpconf8_119_042017.pdf
: A good overview as PDF.

Chapter 8, *OProfile—System-Wide Profiler* (page 97)
: Consult this chapter for other performance optimizations.

OProfile—System-Wide Profiler

8

OProfile is a profiler for dynamic program analysis. It investigates the behaviour of a running program and gathers information. This information can be viewed and gives hints for further optimizations.

It is not necessary to recompile or use wrapper libraries in order to use OProfile. Not even a kernel patch is needed. Usually, when you profile an application, a small overhead is expected, depending on work load and sampling frequency.

8.1 Conceptual Overview

OProfile consists of a kernel driver and a daemon for collecting data. It makes use of the hardware performance counters provided on Intel, AMD, and other processors. OProfile is capable of profiling all code including the kernel, kernel modules, kernel interrupt handlers, system shared libraries, and other applications.

Modern processors support profiling through the hardware by performance counters. Depending on the processor, there can be many counters and each of these can be programmed with an event to count. Each counter has a value which determines how often a sample is taken. The lower the value, the more often it is used.

During the post-processing step, all information is collected and instruction addresses are mapped to a function name.

8.2 Installation and Requirements

In order to make use of OProfile, install the `oprofile` package. OProfile works on IA-64, AMD64, s390, and PPC64.

It is useful to install the respective `debuginfo` package for the respective application you want to profile. If you want to profile the Kernel, you need the `debuginfo` package as well.

8.3 Available OProfile Utilities

OProfile contains several utilities to handle the profiling process and its profiled data. The following list is a short summary of processes used in this chapter:

`opannotate`
: Outputs annotated source or assembly listings mixed with profile information.

`opcontrol`
: Controls the profiling sessions (start or stop), dumps profile data, and sets up parameters.

`ophelp`
: Lists available events with short descriptions.

`opimport`
: Converts sample database files from a foreign binary format to the native format.

`opreport`
: Generates reports from profiled data.

8.4 Using OProfile

It is possible with OProfile to profile both kernel and applications. When profiling the kernel, tell OProfile where to find the `vmlinuz*` file. Use the `--vmlinux` option and point it to `vmlinuz*` (usually in `/boot`). If you need to profile kernel modules, OProfile does this by default. However, make sure you read http://oprofile.sourceforge.net/doc/kernel-profiling.html.

Applications usually do not need to profile the kernel, so better use the --no-vmlinux option to reduce the amount of information.

8.4.1 General Steps

In its simplest form, start the daemon, collect data, stop the daemon, and create your report. This method is described in detail in the following procedure:

1. Open a shell and log in as root.

2. Decide if you want to profile with or without the Linux kernel:

   2a Profile With the Linux Kernel Execute the following commands, because the opcontrol command needs an uncompressed image:

   ```
   cp /boot/vmlinux-`uname -r`.gz /tmp
   gunzip /tmp/vmlinux*.gz
   opcontrol --vmlinux=/tmp/vmlinux*
   ```

   2b Profile Without the Linux Kernel Use the following command:

   ```
   opcontrol --no-vmlinux
   ```

   If you want to see which functions call other functions in the output, use additionally the --callgraph option:

   ```
   opcontrol --no-vmlinux --callgraph
   ```

3. Start the OProfile daemon:

   ```
   opcontrol --start
   Using 2.6+ OProfile kernel interface.
   Using log file /var/lib/oprofile/samples/oprofiled.log
   Daemon started.
   Profiler running.
   ```

4. Start your application you want to profile right after the previous step.

5. Stop the OProfile daemon:

   ```
   opcontrol --stop
   ```

6. Dump the collected data to `/var/lib/oprofile/samples`:

   ```
   opcontrol --dump
   ```

7. Create a report:

   ```
   opreport
   Overflow stats not available
   CPU: CPU with timer interrupt, speed 0 MHz (estimated)
   Profiling through timer interrupt
             TIMER:0|
     samples|      %|
   ------------------
       84877 98.3226 no-vmlinux
   ...
   ```

8. Shutdown the OProfile daemon:

   ```
   opcontrol --shutdown
   ```

8.4.2 Getting Event Configurations

The general procedure for event configuration is as follows:

1. Use first the events `CPU-CLK_UNHALTED` and `INST_RETIRED` to find optimization opportunities.

2. Use specific events to find bottlenecks. To list them, use the command `opcontrol --list-events`.

If you need to profile certain events, first check the available events supported by your processor with the `ophelp` command (example output generated from Intel Core i5 CPU):

```
ophelp
oprofile: available events for CPU type "Intel Architectural Perfmon"

See Intel 64 and IA-32 Architectures Software Developer's Manual
Volume 3B (Document 253669) Chapter 18 for architectural perfmon events
This is a limited set of fallback events because oprofile doesn't know your
CPU
```

```
CPU_CLK_UNHALTED: (counter: all))
        Clock cycles when not halted (min count: 6000)
INST_RETIRED: (counter: all))
        number of instructions retired (min count: 6000)
LLC_MISSES: (counter: all))
        Last level cache demand requests from this core that missed the LLC
(min count: 6000)
        Unit masks (default 0x41)
        ----------
        0x41: No unit mask
LLC_REFS: (counter: all))
        Last level cache demand requests from this core (min count: 6000)
        Unit masks (default 0x4f)
        ----------
        0x4f: No unit mask
BR_MISS_PRED_RETIRED: (counter: all))
        number of mispredicted branches retired (precise) (min count: 500)
```

You can get the same output from `opcontrol --list-events`.

Specify the performance counter events with the option `--event`. Multiple options are possible. This option needs an event name (from `ophelp`) and a sample rate, for example:

```
opcontrol --event=CPU_CLK_UNHALTED:100000
```

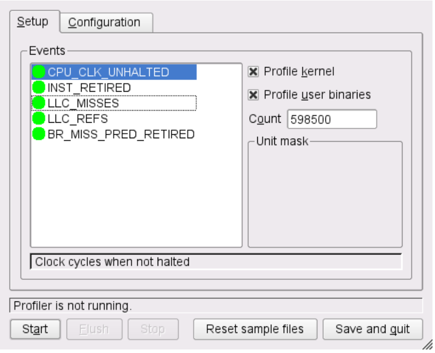
WARNING: Be Careful with Low Sampling Rates with `CPU_CLK_UNHALTED`

Setting sampling rates is dangerous as small rates cause the system to overload and freeze.

8.5 Using OProfile's GUI

The GUI for OProfile can be started as `root` with `oprof_start`, see Figure 8.1, “GUI for OProfile” (page 102). Select your events and change the counter, if necessary. Every highlighted line is added to the list of checked events. Use the *Configuration* tab to set the buffer and CPU size, the verbose option and others. Click on *Start* to execute OProfile.

Figure 8.1 *GUI for OProfile*



- CPU_CLK_UNHALTED
- INST_RETIRED
- LLC_MISSES
- LLC_REFS
- BR_MISS_PRED_RETIRED

☒ Profile kernel

☒ Profile user binaries

Count 598500

Clock cycles when not halted

Profiler is not running.

8.6 Generating Reports

Before generating a report, make sure OProfile has dumped your data to the `/var/lib/oprofile/samples` directory using the command `opcontrol --dump`. A report can be generated with the commands `opreport` or `opannotate`.

Calling `oreport` without any options gives a complete summary. With an executable as an argument, retrieve profile data only from this executable. If you analyze applications written in C++, use the `--demangle smart` option.

The `opannotate` generates output with annotations from source code. Run it with the following options:

```
opannotate --source \
   --base-dirs=BASEDIR \
   --search-dirs= \
   --output-dir=annotated/ \
   /lib/libfoo.so
```

The option `--base-dir` contains a comma separated list of paths which is stripped from debug source files. This paths were searched prior than looking in `--search-dirs`. The `--search-dirs` option is also a comma separated list of directories to search for source files.

NOTE: Inaccuracies in Annotated Source

Due to compiler optimization, code can disappear and appear in a different place. Use the information in `http://oprofile.sourceforge.net/doc/debug-info.html` to fully understand its implications.

8.7 For More Information

This chapter only provides a short overview. Refer to the following links for more information:

`http://oprofile.sourceforge.net`
The project home page.

Manpages
Details descriptions about the options of the different tools.

`/usr/share/doc/packages/oprofile/oprofile.html`
Contains the OProfile manual.

`http://developer.intel.com/`
Architecture reference for Intel processors.

`http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/22007.pdf`
Architecture reference for AMD Athlon/Opteron/Phenom/Turion.

`http://www-01.ibm.com/chips/techlib/techlib.nsf/productfamilies/PowerPC/`
Architecture reference for PowerPC64 processors in IBM iSeries, pSeries, and blade server systems.

Part IV. Resource Management

General System Resource Management 9

Tuning the system is not only about optimizing the kernel or getting the most out of your application, it begins with setting up a lean and fast system. The way you set up your partitions and file systems can influence the servers speed. The number of active services and the way routine tasks are scheduled also affects performance.

9.1 Planning the Installation

A carefully planned installation ensures that the system is basically set up exactly as you need it for the given purpose. It also saves considerable time when fine tuning the system. All changes suggested in this section can be made in the *Installation Settings* step during the installation. See Section “Installation Settings” (Chapter 6, *Installation with YaST*, ↑*Deployment Guide*) for details.

9.1.1 Partitioning

Depending on the server's range of application and the hardware layout the partitioning scheme can influence the machine's performance (although to a lesser extend only). It is beyond the scope of this manual to suggest different partion schemes for particular workloads, however, the following rules will positively affect performance. Of course they do not apply when using an external storage system.

- Make sure there always is some free space available on the disk, since a full disk has got inferior performance

- Disperse simultaneous read and write access onto different disks by, for example:
  - using separate disks for the operating system, the data, and the log files
  - placing a mail server's spool directory on a separate disk
  - distributing the user directories of a home server between different disks

9.1.2 Installation Scope

Actually, the installation scope has no direct influence on the machine's performance, but a carefully chosen scope of packages nevertheless has got advantages. It is recommended to install the minimum of packages needed to run the server. A system with a minimum set of packages is easier to mainatain and has got less potential security issues. Furthermore, a tailor made installation scope also ensures no unnecessary services are started by default.

SUSE Linux Enterprise Server lets you customize the installation scope on the Installation Summary screen. By default, you can select or remove preconfigured patterns for specific tasks, but it is also possible to start the YaST Software Manager for a fine-grained package based selection.

One or more of the following patterns selected for installation by default may not be needed in all cases:

GNOME Desktop Environment

A server seldomly needs a full-blown desktop environment. In case a graphical environment is needed, a more economical solution such as as icewm or fvwm may also be sufficient.

X Window System

When solely administrating the server and its applications via command line, consider to not install this pattern. However, keep in mind that it is needed to run GUI applications from a remote machine. If your application is managed by a GUI or if you prefer the GUI version of YaST, keep this pattern.

Print Server

This pattern is only needed when you want to print from the machine.

9.1.3 Default Runlevel

A running X Window system eats up many ressources and is seldomly needed on a server. It is strongly recommended to start the system in runlevel 3 (Full multi-user with network, no X). You will still be able to start graphical applications from remote or use the `startx` command to start a local graphical desktop.

9.2 Disabling Unnecessary Services

The default installation starts a number of services (the number varies with the installation scope). Since each service consumes ressources, it is recommended to disable the ones not needed. Start *YaST* > *System* > *System Services (Runlevel)* > *Expert Mode* to start the services management module. When using the graphical version of YaST you can click on the column headlines to sort the service list. Use this to get an overview of which services are currently running or which services are started in the server's default runlevel. Mark a service with the mouse to see its description. Use the *Start/Stop/Refresh* dropdown to disable the service for the running session. To permanently disable it, use the *Set/Reset* drop-down.

The following list shows services started after a default installation of SUSE Linux Enterprise Server that may not be needed:

alsasound
: Loads the Advanced Linux Sound System. Disable if you do not need sound.

auditd
: A daemon for the audit system (see Part “*The Linux Audit Framework*” (↑*Security Guide*) for details). Disable if you do not use Audit.

bluez-coldplug
: Handles coldplugging of bluetooth dongles. Disable if you do not have bluetooth.

cups
: A printer daemon. Disable if you do not have acces to a printer.

java.binfmt_misc
: Enables the execution of *.class or *.jar Java programs. Disable if you do not run Java applications.

nfs

Services needed to mount NFS file systems. Disable if not needed.

smbfs

Services needed to mount SMB/CIFS file systems from a Windows server. Disable if not needed.

splash / splash_early

Shows the splash screen on start-up. Usually not needed on a server

9.3 File Systems and Disk Access

Hard disks are the slowest components in a computer system and therefore often the cause for a bottleneck. Using the file system that best suits your workload helps to improve performance. Using special mount options or prioritizing a process' I/O priority are further means to speed up the system.

9.3.1 File Systems

SUSE Linux Enterprise Server ships with a number of different file systems, including Ext3, Ext2, ReiserFS, and XFS. Each file system has its own advantages and disadvantages. Please refer to Chapter 1, *Overview of File Systems in Linux* (↑SLES 11 SP1: Storage Administration Guide) for detailed information.

NFS

NFS (Version 3) tuning is covered in detail in the NFS Howto at `http://nfs.sourceforge.net/nfs-howto/`. The first thing you should experiment with when mounting NFS shares is increasing the read write blocksize to 32768 by using the mount options `wsize` and `rsize`.

9.3.2 Disabling Access Time (atime) Updates

Whenever a file is read on a Linux file system, its access time (atime) is updated. As a result, each read-only file access in fact causes a write. On a journaling file system it is even two write operations since the journal will be updated, too. It is recommended to turn this feature off when you do not need to keep track of access times. This is possibly true for file and Web servers as well as for a netwok storage.

To turn off access time updates, mount the file system with the `noatime` option. To do so, either edit `/etc/fstab` directly, or use the *Fstab Options* dialog when editing or adding a partition with the YaST Partitioner.

9.3.3 Prioritizing Disk Access with ionice

The `ionice` command lets you prioritize disk access for single processes. This enables you to give less I/O priority to non time-critical background processes with heavy disk access such as backup jobs. On the other hand `ionice` lets you raise I/O priority for a specific process to make sure this process has always immediate access to the disk. You may set the following three scheduling classes:

Idle

A process from the idle scheduling class is only granted disk access, when no other process has asked for disk I/O.

Best effort

By default, every process will be granted I/O priority from this class. Priority within this class can be adjusted to a level from 0 to 7 (with 0 being the highest priority). By default, a process will be granted a priority corresponding to their CPU nice level.

Real-time

Processes in this class are always granted disk access first. Fine-tune the priority level from 0 to 7 (with 0 being the highest priority). Use with care, since it can starve other processes.

For more details and the exact command syntax refer to the `ionice(1)` man page for ionice.

10 Kernel Control Groups

Kernel Control Groups (abbreviated known as “cgroups”) are a kernel feature that allows aggregating or partitioning tasks (processes) and all their children into hierarchical organized groups. These hierarchical groups can be configured to show a specialized behavior that helps with tuning the system to make best use of available hardware and network resources.

10.1 Technical Overview and Definitions

The following terms are used in this chapter:

- “cgroup” is another name for Control Groups.

- In a cgroup there is a set of tasks (processes) associated with a set of subsystems that act as parameters constituting an environment for the tasks.

- Subsystems provide the parameters that can be assigned and define CPU sets, freezer, or—more general—“resource controllers” for memory, disk I/O, etc.

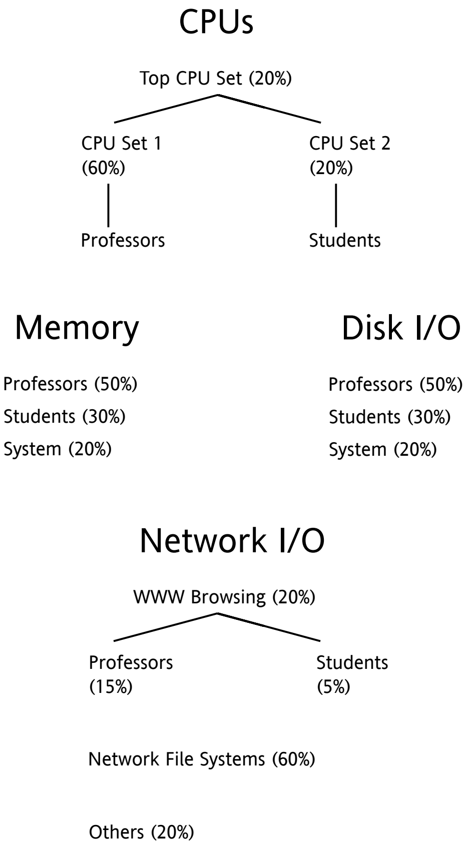
- cgroups are organized in a tree-structured hierarchy. There can be more than one hierarchy in the system. You use a different or alternate hierarchy to cope with specific situations.

- Every task running in the system is in exactly one of the cgroups in the hierarchy.

10.2 Scenario

See the following resource planning scenario for a better understanding (source: `/usr/src/linux/Documentation/cgroups/cgroups.txt`):

Figure 10.1 *Resource Planning*



Web browser such as Firefox will be part of the Web network class, while the NFS daemons such as (k)nfsd will be part of the NFS network class. On the other side, Firefox will share appropriate CPU and memory classes depending on whether a professor or student started it.

10.3 Control Group Subsystems

The following subsystems are available and can be classified as two types:

Isolation and Special Controllers
: cpuset, namespace, freezer, device, checkpoint/restart

Resource Controllers
: cpu(scheduler), memory, disk I/O, network

Either mount each subsystem separately:

```
mount -t cgroup -o cpu none /cpu
mount -t cgroup -o cpuset none /cpuset
```

or all subsystems in one go:

```
mount -t cgroup none /cgroups
```

Some additional information on available subsystems:

Cpuset (Isolation)
: Use cpuset to tie processes to system subsets of CPUs and memory ("memory nodes"). For an example, see Section 10.4.3, "Example: Cpusets" (page 118).

Namespace (Isolation)
: Namespace is for showing private view of system to processes in cgroup. It is mainly used for OS-level virtualization. This subsystem itself has no special functions and just tracks changes in namespace.

Freezer (Control)
: The Freezer subsystem is useful for high-performance computing clusters (HPC clusters). Use it to freeze (stop) all tasks in a group or to stop tasks, if they reach

a defined checkpoint. For more information, see /usr/src/linux/Documentation/cgroups/freezer-subsystem.txt.

Here are basic commands, how you can use the freezer subsystem:

```
mount -t cgroup freezer  /freezer -o freezer
# Create a child cgroup:
mkdir /freezer/0
# Put a task into this cgroup:
echo $task_pid > /freezer/0/tasks
# Freeze it:
echo FROZEN > /freezer/0/freezer.state
# Unfreeze (thaw) it:
echo THAWED > /freezer/0/freezer.state
```

Device (Isoloation)

A system administrator can provide a list of devices that can be accessed by processes under cgroups.

It limits access to a device or a file system on a device to only tasks that belong to the specified cgroup. For more information, see /usr/src/linux/Documentation/cgroups/devices.txt.

Checkpoint/Restart (Control)

Save the state of all processes in a cgroup to a dump file. Restart it later (or just save the state and continue).

Allows to move “saved container” between physical machines (as VM can do).

Dump all process's image to a file.

Cpuacct (Control)

The CPU accounting controller groups tasks using cgroups and accounts the CPU usage of these groups. For more information, see /usr/src/linux/Documentation/cgroups/cpuacct.txt.

CPU (Resource Control)

Share CPU bandwidth between groups with the group scheduling function of CFS (the scheduler). Mechanically complicated.

Memory (Resource Control)

- Limits memory usage of user space processes.

- Limit LRU (Least Recently Used) pages.

- Anonymous and file cache.

- No limits for kernel memory.

- Maybe in another subsystem if needed.

For more information, see `/usr/src/linux/Documentation/cgroups/memory.txt`.

Disk I/O (Resource Control) (Draft)
Three proposals are currently being discussed: dm-ioband, io-throttle, and io-controller.

Network I/O (Resource Control) (Draft)
Still under discussion.

10.4 Using Controller Groups

10.4.1 Prerequisites

To use cgroups, install the following additional packages:

- `libcgroup1` contains basic user space tools to simplify resource management.

- `cpuset`

- `libcpuset1`

- `kernel-source` (for documentation purposes only)

- `lcx`

10.4.2 Checking the Environment

The kernel shipped with SUSE Linux Enterprise Server supports cgroups. There is no need to apply additional patches. Execute `lxc-checkconfig` to see a cgroups environment similar to the following output:

```
--- Namespaces ---
Namespaces: enabled
Utsname namespace: enabled
Ipc namespace: enabled
Pid namespace: enabled
User namespace: enabled
Network namespace: enabled
Multiple /dev/pts instances: enabled

--- Control groups ---
Cgroup: enabled
Cgroup namespace: enabled
Cgroup device: enabled
Cgroup sched: enabled
Cgroup cpu account: enabled
Cgroup memory controller: enabled
Cgroup cpuset: enabled

--- Misc ---
Veth pair device: enabled
Macvlan: enabled
Vlan: enabled
File capabilities: enabled
```

To find out which subsystems are available, proceed as follows:

```
mkdir /cgroups
mount -t cgroup none /cgroups
grep cgroup /proc/mounts
```

The following subsystems are available: rw, freezer, devices, cpuacct, cpu, ns, cpuset, memory. Disk and network subsystem controllers may become available during SUSE Linux Enterprise Server 11 lifetime.

10.4.3 Example: Cpusets

With the command line proceed as follows:

1 To determine the number of CPUs and memory nodes see `/proc/cpuinfo` and `/proc/zoneinfo`.

2 Create the cpuset hierarchy as a virtual file system (source: /usr/src/linux/Documentation/cgroups/cgroups.txt):

```
mkdir /dev/cpuset
mount -t cpuset cpuset /dev/cpuset
cd /dev/cpuset
mkdir Charlie
cd Charlie
# List of CPUs in this cpuset:
/bin/echo 2-3 > cpus
# List of memory nodes in this cpuset:
/bin/echo 1 > mems
/bin/echo $$ > tasks
# The current shell is now running in the Charlie cpuset
# The next line should display '/Charlie'
cat /proc/self/cpuset
```

3 Remove the cpuset using shell commands:

```
rmdir /dev/cpuset/Charlie
```

This fails as long as this cpuset is in use. First, you have to remove the inside cpusets or tasks (processes) that belong to it. Check this with:

```
cat /dev/cpuset/Charlie/tasks
```

For background information and additional configuration flags, see `/usr/src/linux/Documentation/cgroups/cpusets.txt`.

With the `cset` tool, proceed as follows:

```
# Determine the number of CPUs and memory nodes
cset set --list
# Creating the cpuset hierarchy
cset set --cpu=2-3 --mem=1 --set=Charlie
# Starting processes in a cpuset
cset proc --set Charlie --exec -- stress -c 1 &
# Moving existing processes to a cpuset
cset proc --move --pid PID --toset=Charlie
# List task in a cpuset
cset proc --list --set Charlie
# Removing a cpuset
cset set --destroy Charlie
```

10.4.4 Example: cgroups

Using shell commands, proceed as follows:

1. Create the cgroups hierarchy:

   ```
   mkdir /dev/cgroup
   mount -t cgroup cgroup /dev/cgroup
   cd /dev/cgroup
   mkdir priority
   cd priority
   cat cpu.shares
   ```

2. Understanding cpu.shares:

   - 1024 is the default (for more information, see `sched-design-CFS.txt`) = 50% utilization
   - 1524 = 60% utilization
   - 2048 = 67% utilization
   - 512 = 40% utilization

3. Changing cpu.shares

   ```
   /bin/echo 1024 > cpu.shares
   ```

10.5 For More Information

- Kernel documentation (package `kernel-source`): files in `/usr/src/linux/Documentation/cgroups`:
  - `/usr/src/linux/Documentation/cgroups/cgroups.txt`
  - `/usr/src/linux/Documentation/cgroups/cpuacct.txt`
  - `/usr/src/linux/Documentation/cgroups/cpusets.txt`
  - `/usr/src/linux/Documentation/cgroups/devices.txt`

- /usr/src/linux/Documentation/cgroups/freezer-subsystem.txt
  - /usr/src/linux/Documentation/cgroups/memcg_test.txt
  - /usr/src/linux/Documentation/cgroups/memory.txt
  - /usr/src/linux/Documentation/cgroups/resource_counter.txt
- http://lwn.net/Articles/243795/—Corbet, Jonathan: Controlling memory use in containers (2007).
- http://lwn.net/Articles/236038/—Corbet, Jonathan: Process containers (2007).

Power Management 11

Power management aims at reducing operating costs for energy and cooling systems while at the same time keeping the performance of a system at a level that matches the current requirements. Thus, power management is always a matter of balancing the actual performance needs and power saving options for a system. Power management can be implemented and used at different levels of the system. A set of specifications for power management functions of devices and the operating system interface to them has been defined in the Advanced Configuration and Power Interface (ACPI). As power savings in server environments can primarily be achieved on processor level, this chapter introduces some of the main concepts and highlights some tools for analyzing and influencing relevant parameters.

11.1 Power Management at CPU Level

At CPU level, you can control power usage in various ways: for example, by using idling power states (C-states), changing CPU frequency (P-states), and throttling the CPU (T-states). The following sections give a short introduction to each approach and its significance for power savings. Detailed specifications can be found at `http://www.acpi.info/spec.htm`.

11.1.1 C-States (Processor Operating States)

Modern processors have several power saving modes called `C-states`. They reflect the capability of an idle processor to turn off unused components in order to save power. Whereas C-states have been available for laptops for some time, they are a rather recent trend in the server market (for example, with Intel* processors, C-modes are only available since Nehalem).

When a processor runs in the `C0` state, it is executing instructions. A processor running in any other C-state is idle. The higher the C number, the deeper the CPU sleep mode: more components are shut down to save power. Deeper sleep states save more power, but the downside is that they have higher latency (the time the CPU needs to go back to `C0`).

Some states also have submodes with different power saving latency levels. Which C-states and submodes are supported depends on the respective processor. However, `C1` is always available.

Table 11.1, “C-States” (page 124) gives an overview of the most common C-states.

Table 11.1 *C-States*

Mode	Definition
C0	Operational state. CPU fully turned on.
C1	First idle state. Stops CPU main internal clocks via software. Bus interface unit and APIC are kept running at full speed.
C2	Stops CPU main internal clocks via hardware. State where the processor maintains all software-visible states, but may take longer to wake up through interrupts.
C3	Stops all CPU internal clocks. The processor does not need to keep its cache coherent, but maintains other states. Some processors have variations of the C3 state that differ in how long it takes to wake the processor through interrupts.

11.1.2 P-States (Processor Performance States)

While a processor operates (in C0 state), it can be in one of several CPU performance states (P-states). Whereas C-states are idle states (all but C0), P-states are operational states that relate to CPU frequency and voltage.

The higher the P-state, the lower the frequency and voltage at which the processor runs. The number of P-states is processor-specific and the implementation differs across the various types. However, P0 is always the highest-performance state. Higher P-state numbers represent slower processor speeds and lower power consumption. For example, a processor in P3 state runs more slowly and uses less power than a processor running at P1 state. To operate at any P-state, the processor must be in the C0 state where the processor is working and not idling. The CPU P-states are also defined in the Advanced Configuration and Power Interface (ACPI) specification, see http://www.acpi.info/spec.htm.

C-states and P-states can vary independently of one another.

11.1.3 T-States (Processor Throttling States)

T-states refer to throttling the processor clock to lower frequencies in order to reduce thermal effects. This means that the CPU is forced to be idle a fixed percentage of its cycles per second. Throttling states range from T1 (the CPU has no forced idle cycles) to T*n*, with the percentage of idle cycles increasing the greater *n* is.

This differs from changing the frequency (which makes the CPU have fewer cycles per second), and from running in a C-state other than C1. Note that throttling does not reduce voltage and since the CPU is forced to idle part of the time, processes will take longer to finish and will consume more power instead of saving any power.

T-states are a concept from the times when dynamic frequency scaling and C-states did not exist. With the implementation of the latter, T-states are only useful if reducing thermal effects is the primary goal. Since T-states can interfere with C-states (preventing the CPU from reaching higher C-states), they can even increase power consumption in a modern CPU capable of C-states.

11.2 The Linux Kernel CPUfreq Infrastructure

Processor performance states (P-states) and processor operating states (C-states) are the capability of a processor to switch between different supported operating frequencies and voltages to modulate power consumption.

In order to dynamically scale processor frequencies at runtime, you can use the CPUfreq infrastructure to set a static or dynamic power policy for the system. Its main components are the CPUfreq subsystem (providing a common interface to the various low-level technologies and high-level policies) , the in-kernel governors (policy governors that can change the CPU frequency based on different criteria) and CPU-specific drivers that implement the technology for the specific processor. Apart from that, user-space daemons may be available.

The dynamic scaling of the clock speed helps to consume less power and generate less heat when not operating at full capacity.

11.2.1 In-Kernel Governors

You can think of the in-kernel governors as a sort of pre-configured power schemes for the CPU. The CPUfreq governors use P-states to change frequencies and lower power consumption. The dynamic governors can switch between CPU frequencies, based on CPU utilization to allow for power savings while not sacrificing performance. These governors also allow for some tuning so you can customize and change the frequency scaling.

The following governors are available with the CPUfreq subsystem:

Performance Governor

The CPU frequency is statically set to the highest possible for maximum performance. Consequently, saving power is not the focus of this governor.

Tuning options: The range of maximum frequencies available to the governor can be adjusted. For details, see Section 11.3.2, “Modifying Current Settings with `cpufreq-set`” (page 129).

Powersave Governor

The CPU frequency is statically set to the lowest possible. This can have severe impact on the performance, as the system will never rise above this frequency no matter how busy the processors are.

However, using this governor often does not lead to the expected power savings as the highest savings can usually be achieved at idle through entering C-states. Due to running processes at the lowest frequency with the powersave governor, processes will take longer to finish, thus prolonging the time for the system to enter any idle C-states.

Tuning options: The range of minimum frequencies available to the governor can be adjusted. For details, see Section 11.3.2, "Modifying Current Settings with `cpufreq-set`" (page 129).

On-demand Governor

The kernel implementation of a dynamic CPU frequency policy: The governor monitors the processor utilization. As soon as it exceeds a certain threshold, the governor will set the frequency to the highest available. If the utilization is less than the threshold, the next lowest frequency is used. If the system continues to be underutilized, the frequency is again reduced until the lowest available frequency is set.

Tuning options: The range of available frequencies, the rate at which the governor checks utilization, and the utilization threshold can be adjusted.

Conservative Governor

Similar to the on-demand implementation, this governor also dynamically adjusts frequencies based on processor utilization, except that it allows for a more gradual increase in power. If processor utilization exceeds a certain threshold, the governor does not immediately switch to the highest available frequency (as the on-demand governor does), but only to next higher frequency available.

Tuning options: The range of available frequencies, the rate at which the governor checks utilization, the utilization thresholds, and the frequency step rate can be adjusted.

11.2.2 Related Files and Directories

If the CPUfreq subsystem in enabled on your system (which it is by default with SUSE Linux Enterprise Server), you can find the relevant files and directories under `/sys/devices/system/cpu/`. If you list the contents of this directory, you will find a `cpu{0..x}` subdirectory for each processor, and several other files and directories. You will find a `cpufreq` subdirectory in each processor directory, holding a number of files and directories that define the parameters for CPUfreq. Some of them are writable (for `root`), some of them are read-only. If your system currently uses the on-demand or conservative governor, you will see a separate subdirectory for those governors in `cpufreq`, containing the parameters for the governors.

NOTE: Different Processor Settings

The settings under the `cpufreq` directory can be different for each processor. If you want to use the same policies across all processors, you need to adjust the parameters for each processor.

11.3 Tuning Options for P-states

The CPUfreq subsystem offers several tuning options for P-states: You can switch between the different governors or change individual governor parameters.

Though you can view or adjust the current settings manually (in `/sys/devices/system/cpu/cpufreq` or in `/sys/devices/system/cpu/cpu*/cpufreq` for machines with multiple cores), we advise to use the tools provided by `cpufrequtils` for that. After you have installed the `cpufrequtils` package, you can make use of the `cpufreq-info` and `cpufreq-set` command line tools as described below.

11.3.1 Viewing Current Settings with cpufreq-info

The `cpufreq-info` command helps you to retrieve CPUfreq kernel information. Run without any options, it collects the information available for your system and shows an output similar to the following:

```
cpufrequtils 004: cpufreq-info (C) Dominik Brodowski 2004-2006
Report errors and bugs to http://bugs.opensuse.org, please.
analyzing CPU 0:
  driver: acpi-cpufreq
  CPUs which need to switch frequency at the same time: 0
  hardware limits: 2.80 GHz - 3.40 GHz
  available frequency steps: 3.40 GHz, 2.80 GHz
  available cpufreq governors: conservative, userspace, powersave, ondemand, performance
  current policy: frequency should be within 2.80 GHz and 3.40 GHz.
                  The governor "performance" may decide which speed to use
                  within this range.
  current CPU frequency is 3.40 GHz.
analyzing CPU 1:
  driver: acpi-cpufreq
  CPUs which need to switch frequency at the same time: 1
  hardware limits: 2.80 GHz - 3.40 GHz
  available frequency steps: 3.40 GHz, 2.80 GHz
  available cpufreq governors: conservative, userspace, powersave, ondemand, performance
  current policy: frequency should be within 2.80 GHz and 3.40 GHz.
                  The governor "performance" may decide which speed to use
                  within this range.
  current CPU frequency is 3.40 GHz.
```

Using the appropriate options, you can view the current CPU frequency, the minimum and maximum CPU frequency allowed, show the currently used CPUfreq policy, the available CPUfreq governors, or determine the CPUfreq kernel driver used. For more details and the available options, refer to the `cpufreq-info` man page or run `cpufreq-info --help`.

11.3.2 Modifying Current Settings with cpufreq-set

To modify CPUfreq settings, use the `cpufreq-set` command as `root`. It allows you set values for the minimum or maximum CPU frequency the governor may select or to create a new governor. With the `-c` option, you can also specify for which of the processors the settings should be modified. That makes it easy to use a consistent policy across all processors without adjusting the settings for each processor individually. For more details and the available options, refer to the cpufreq-set man page or run cpufreq-set --help.

You can switch to another governor at runtime with the -g option. For example, the following command will activate the on-demand governor:

```
cpufreq-set -g ondemand
```

If you want the change in governor to persist also after a reboot or shutdown, use the pm-profiler as described in Section 11.5, “Creating and Using Power Management Profiles” (page 132).

11.3.3 Modifying Further Settings

Apart from the governor settings that can be influenced with cpufreq-set (like minimum or maximum CPU frequency to be used), you can also tune further governor parameters manually, for example, Ignoring Nice Values in Processor Utilization (page 130).

Another parameter that significantly impacts the performance loss caused by dynamic frequency scaling is the sampling rate (rate at which the governor checks the current CPU load and adjusts the processor's frequency accordingly). Its default value depends on a BIOS value and it should be as low as possible. However, in modern systems, an appropriate sampling rate is set by default and does not need manual intervention.

Procedure 11.1 *Ignoring Nice Values in Processor Utilization*

One parameter you might want to change for the on-demand or conservative governor is ignore_nice_load.

Each process has a niceness value associated with it. This value is used by the kernel to determine which processes require more processor time than others. The higher the nice value, the lower the priority of the process. Or: the “nicer” a process, the less CPU it will try to take from other processes.

If the ignore_nice_load parameter for the on-demand or conservative governor is set to 1, any processes with a nice value will not be counted toward the overall processor utilization. When ignore_nice_load is set to 0 (default value), all processes are counted toward the utilization. Adjusting this parameter can be useful if you are running something that requires a lot of processor capacity but you do not care about the runtime.

1. Change to the subdirectory of the governor whose settings you want to modify, for example:

   ```
   cd /sys/devices/system/cpu/cpu0/cpufreq/conservative/
   ```

2. Show the current value of `ignore_nice_load` with:

   ```
   cat ignore_nice_load
   ```

3. To set the value to `1`, execute:

   ```
   echo 1 > ignore_nice_load
   ```

When setting the `ignore_nice_load` value for `cpu0`, the same value is automatically used for all cores. In this case, you do not need to repeat the steps above for each of the processors where you want to modify this governor parameter.

11.4 Tuning Options for C-states

By default, SUSE Linux Enterprise Server uses C-states appropriately. The only parameter you might want to touch for optimization is the `sched_mc_power_savings` scheduler. Instead of distributing a work load across all cores with the effect that all cores are utilized only at a minimum level, the kernel can try to schedule processes on as few cores as possible so that the others can go idle. This helps to save power as it allows some processors to be idle for a longer time so they can reach a higher C-state. However, the actual savings depend on a number of factors, for example how many processors are available and which C-states are supported by them (especially deeper ones such as C3 to C6).

If `sched_mc_power_savings` is set to `0` (default value), no special scheduling is done. If it is set to `1`, the scheduler tries to consolidate the work onto the fewest number of processors possible in the case that all processors are a little busy. To modify this parameter, proceed as follows:

Procedure 11.2 *Scheduling Processes on Cores*

1 Change to the subdirectory where the scheduler is located:

```
cd /sys/devices/system/cpu/
```

2 Show the current value of `sched_mc_power_savings` with:

```
cat sched_mc_power_savings
```

3 To set the value to `1`, execute:

```
echo 1 > sched_mc_power_savings
```

11.5 Creating and Using Power Management Profiles

SUSE Linux Enterprise Server includes pm-profiler, intended for server use. It is a script infrastructure to enable or disable certain power management functions via configuration files. It allows you to define different profiles, each having a specific configuration file for defining different settings. A configuration template for new profiles can be found at `/usr/share/doc/packages/pm-profiler/config.template`. The template contains a number of parameters you can use for your profile, including comments on usage and links to further documentation. The individual profiles are stored in `/etc/pm-profiler/`. The profile that will be activated on system start, is defined in `/etc/pm-profiler.conf`.

Procedure 11.3 *Creating and Switching Power Profiles*

To create a new profile, proceed as follows:

1 Create a directory in `/etc/pm-profiler/`, containing the profile name, for example:

```
mkdir /etc/pm-profiler/testprofile
```

2 To create the configuration file for the new profile, copy the profile template to the newly created directory:

```
cp /usr/share/doc/packages/pm-profiler/config.template \
     /etc/pm-profiler/testprofile/config
```

3 Edit the settings in `/etc/pm-profiler/testprofile/config` and save the file. You can also remove variables that you do not need—they will be handled like empty variables, the settings will not be touched at all.

4 Edit `/etc/pm-profiler.conf`. The `PM_PROFILER_PROFILE` variable defines which profile will be activated on system start. If it has no value, the default system or kernel settings will be used. To set the newly created profile:

```
PM_PROFILER_PROFILE="testprofile"
```

The profile name you enter here must match the name you used in the path to the profile configuration file (`/etc/pm-profiler/testprofile/config`), not necessarily the `NAME` you used for the profile in the `/etc/pm-profiler/testprofile/config`.

5 To activate the profile, run

```
rcpm-profiler start
```

or

```
/usr/lib/pm-profiler/enable-profile testprofile
```

Though you have to manually create or modify a profile by editing the respective profile configuration file, you can use YaST to switch between different profiles. Start YaST and select *System* > *Power Management* to open the *Power Management Settings*. Alternatively, become `root` and execute `yast2 power-management` on a command line. The drop-down list shows the available profiles. `Default` means that the system default settings will be kept. Select the profile to use and click *Finish*.

11.6 Monitoring Power Consumption with powerTOP

A useful tool for monitoring system power consumption is powerTOP. It helps you to identify the reasons for unnecessary high power consumption (for example, processes that are mainly responsible for waking up a processor from its idle state) and to optimize your system settings to avoid these. It supports both Intel and AMD processors. The powertop package is available from the SUSE Linux Enterprise SDK. For information how to access the SDK, refer to About This Guide (page vii).

powerTOP combines various sources of information (analysis of programs, device drivers, kernel options, amounts and sources of interrupts waking up processors from sleep states) and shows them in one screen. Example 11.1, “Example powerTOP Output” (page 135) shows which information categories are available:

Example 11.1 *Example powerTOP Output*

```
Cn               Avg  residency       P-states   (frequencies)
❶                ❷         ❸              ❹           ❺
C0 (cpu running)        (11.6%)       2.00 Ghz       0.1%
polling         0.0ms   ( 0.0%)       2.00 Ghz       0.0%
C1              4.4ms   (57.3%)       1.87 Ghz       0.0%
C2             10.0ms   (31.1%)       1064 Mhz      99.9%


Wakeups-from-idle per second : 11.2     interval: 5.0s ❻
no ACPI power usage estimate available ❼


Top causes for wakeups: ❽
96.2% (826.0)       <interrupt> : extra timer interrupt
 0.9% (   8.0)    <kernel core> : usb_hcd_poll_rh_status (rh_timer_func)
 0.3% (   2.4)      <interrupt> : megasas
 0.2% (   2.0)    <kernel core> : clocksource_watchdog (clocksource_watchdog)
 0.2% (   1.6)      <interrupt> : eth1-TxRx-0
 0.1% (   1.0)      <interrupt> : eth1-TxRx-4

[...]

Suggestion: ❾ Enable SATA ALPM link power management via:
echo min_power > /sys/class/scsi_host/host0/link_power_management_policy
or press the S key.
```

❶ The column shows the C-states. When working, the CPU is in state `0`, when resting it is in some state greater than `0`, depending on which C-states are available and how deep the CPU is sleeping.

❷ The column shows average time in milliseconds spent in the particular C-state.

❸ The column shows the percentages of time spent in various C-states. For considerable power savings during idle, the CPU should be in deeper C-states most of the time. In addition, the longer the average time spent in these C-states, the more power is saved.

❹ The column shows the frequencies the processor and kernel driver support on your system.

❺ The column shows the amount of time the CPU cores stayed in different frequencies during the measuring period.

❻ Shows how often the CPU is awoken per second (number of interrupts). The lower the number the better. The `interval` value is the powerTOP refresh interval which can be controlled with the `-t` option. The default time to gather data is 5 seconds.

❼ When running powerTOP on a laptop, this line displays the ACPI information on how much power is currently being used and the estimated time until discharge of the battery. On servers, this information is not available.

❽ Shows what is causing the system to be more active than needed. powerTOP displays the top items causing your CPU to awake during the sampling period.

❾ Suggestions on how to improve power usage for this machine.

For more information, refer to the powerTOP project page at `http://www.lesswatts.org/projects/powertop/`. It also provides tips and tricks and an informative FAQ section.

11.7 Troubleshooting

BIOS options enabled?

In order to make use of C-states or P-states, check your BIOS options:

- To use C-states, make sure to enable `CPU C State` or similar options to benefit from power savings at idle.

- To use P-states and the CPUfreq governors, make sure to enable `Processor Performance States` options or similar.

In case of a CPU upgrade, make sure to upgrade your BIOS, too. The BIOS needs to know the new CPU and its valid frequencies steps in order to pass this information on to the operating system.

CPUfreq subsystem enabled?

In SUSE Linux Enterprise Server, the CPUfreq subsystem is enabled by default. To find out if the subsystem is currently enabled, check for the following path in your system: `/sys/devices/system/cpu/cpufreq` (or `/sys/devices/system/cpu/cpu*/cpufreq` for machines with multiple cores). If the `cpufreq` subdirectory exists, the subsystem is enabled.

Log file information?

Check syslog (usually `/var/log/messages`) for any output regrading the CPUfreq subsystem. Only severe errors are reported there.

If you suspect problems with the CPUfreq subsystem on your machine, you can also enable additional debug output. To do so, either use `cpufreq.debug=7` as boot parameter or execute the following command as `root`:

```
echo 7 > /sys/module/cpufreq/parameters/debug
```

This will cause CPUfreq to log more information to `dmesg` on state transitions, which is useful for diagnosis. But as this additional output of kernel messages can be rather comprehensive, use it only if you are fairly sure that a problem exists.

11.8 For More Information

- A threepart, comprehensive article about tuning components with regards to power efficiency is available at the following URLs:

  - *Reduce Linux power consumption, Part 1: The CPUfreq subsystem*, available at http://www.ibm.com/developerworks/linux/library/l-cpufreq-1/?ca=dgr-lnxw03ReduceLXPWR-P1dth-LX&S_TACT=105AGX59&S_CMP=grlnxw03

  - *Reduce Linux power consumption, Part 2: General and governor-specific settings*, available at http://www.ibm.com/developerworks/linux/library/l-cpufreq-2/?ca=dgr-lnxw03ReduceLXPWR-P1dth-LX&S_TACT=105AGX59&S_CMP=grlnxw03

  - *Reduce Linux power consumption, Part 3: Tuning results*, available athttp://www.ibm.com/developerworks/linux/library/l-cpufreq-3/?ca=dgr-lnxw03ReduceLXPWR-P1dth-LX&S_TACT=105AGX59&S_CMP=grlnxw03

- The LessWatts.org project deals with how to save power, reduce costs and increase efficiency on Linux systems. Find the project home page at http://www.lesswatts.org/. The project page also holds an informative FAQs section at http://www.lesswatts.org/documentation/faq/index.php and provides useful tips and tricks. For tips dealing with the CPU level, refer to http://www.lesswatts.org/tips/cpu.php. For more information about powerTOP, refer to http://www.lesswatts.org/projects/powertop/.

- There is also platform-specific power saving information available, for example: *HP ProLiant Server Power Management on SUSE Linux Enterprise Server 11—Integration Note* , available from `http://h18004.www1.hp.com/products/servers/technology/whitepapers/os-techwp.html`

Part V. Kernel Tuning

Installing Multiple Kernel Versions 12

SUSE Linux Enterprise Server supports the parallel installation of multiple kernel versions. When installing a second kernel, a boot entry and an initrd are automatically created, so no further manual configuration is needed. When rebooting the machine, the newly added kernel is available as an additional boot option.

Using this functionality, you can safely test kernel updates while being able to always fall back to the proven former kernel. To do so, do not use the update tools (such as the YaST Online Update or the updater applet), but instead follow the process described in this chapter.

WARNING: Support Entitlement

Please be aware that you loose your entire support entitlement for the machine when installing a self-compiled or a third-party kernel. Only kernels shipped with SUSE Linux Enterprise Server and kernels delivered via the official update channels for SUSE Linux Enterprise Server are supported.

TIP: Check Your Bootloader Configuration Kernel

It is recommended to check your bootloader config after having installed another kernel in order to set the default boot entry of your choice. See Section “Configuring the Boot Loader with YaST” (Chapter 8, *The Boot Loader GRUB*, ↑*Administration Guide*) for more information. To change the default append line for new kernel installations, adjust `/etc/sysconfig/bootloader` prior to installing a new kernel. For more information refer to

Section "The File /etc/sysconfig/bootloader" (Chapter 8, *The Boot Loader GRUB*, ↑*Administration Guide*).

12.1 Enabling Multiversion Support

Installing multiple versions of a software package (multiversion support) is not enabled by default. To enable this feature, proceed as follows:

1. Open /etc/zypp/zypp.conf with the editor of your choice as root, for example

   ```
   sudo vi /etc/zypp/zypp.conf
   ```

2. Search for the string multiversion. To enable multiversion for all kernel packages capable of this feature, uncomment the following line

   ```
   # multiversion = provides:multiversion(kernel)
   ```

   by removing the # character.

   To restrict multiversion support to certain kernel flavors, specify the package names as a comma-separated list, for example

   ```
   multiversion = kernel-default,kernel-default-base,kernel-source
   ```

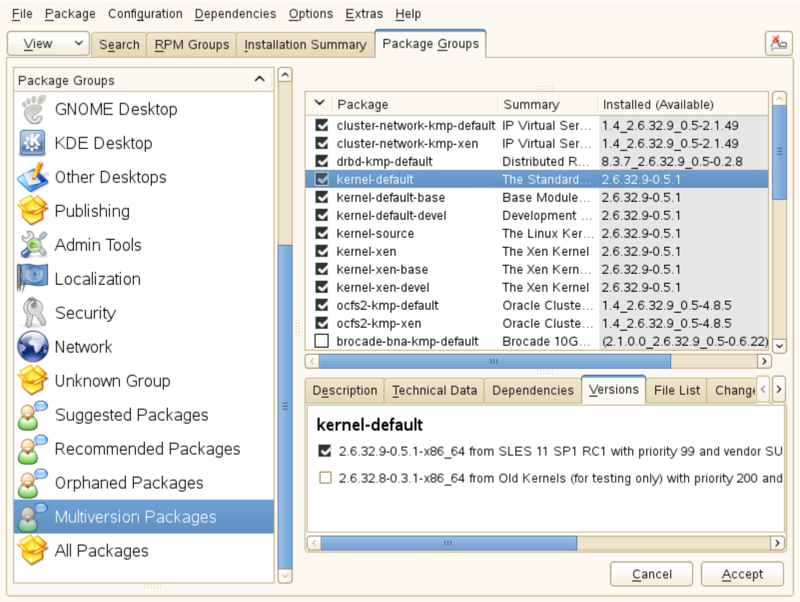
3. Save your changes.

12.2 Installing/Removing Multiple Kernel Versions with YaST

1. Start YaST and open the software manager via *Software* > *Software Mannagment*.

2. List all packages capable of providing multiple versions by choosing *View* > *Package Groups* > *Multiversion Packages*.

Figure 12.1 The YaST Software Manager - Multiversion View



	Package	Summary	Installed (Available)
☑	cluster-network-kmp-default	IP Virtual Ser…	1.4_2.6.32.9_0.5-2.1.49
☑	cluster-network-kmp-xen	IP Virtual Ser…	1.4_2.6.32.9_0.5-2.1.49
☑	drbd-kmp-default	Distributed R…	8.3.7_2.6.32.9_0.5-0.2.8
☑	kernel-default	The Standard…	2.6.32.9-0.5.1
☑	kernel-default-base	Base Module…	2.6.32.9-0.5.1
☑	kernel-default-devel	Development …	2.6.32.9-0.5.1
☑	kernel-source	The Linux Ker…	2.6.32.9-0.5.1
☑	kernel-xen	The Xen Kernel	2.6.32.9-0.5.1
☑	kernel-xen-base	The Xen Kern…	2.6.32.9-0.5.1
☑	kernel-xen-devel	The Xen Kernel	2.6.32.9-0.5.1
☑	ocfs2-kmp-default	Oracle Cluste…	1.4_2.6.32.9_0.5-4.8.5
☑	ocfs2-kmp-xen	Oracle Cluste…	1.4_2.6.32.9_0.5-4.8.5
□	brocade-bna-kmp-default	Brocade 10G…	(2.1.0.0_2.6.32.9_0.5-0.6.22)

kernel-default

☑ 2.6.32.9-0.5.1-x86_64 from SLES 11 SP1 RC1 with priority 99 and vendor SU

□ 2.6.32.8-0.3.1-x86_64 from Old Kernels (for testing only) with priority 200 and

3 Select a package and open its *Version* tab in the bottom pane on the left.

4 To install a package, click its checkbox. A green checkmark indicates it is selected for installation.

To remove an already installed package (marked with a white checkmark), click its checkbox until a red `X` indicates it is selected for removal.

5 Click *Accept* to start the installation.

12.3 Installing/Removing Multiple Kernel Versions with zypper

1 Use the command `zypper se -s 'kernel*'` to display a list of all kernel packages available:

```
S | Name           | Type       | Version         | Arch   | Repository
--+----------------+------------+-----------------+--------+-------------------
v | kernel-default | package    | 2.6.32.10-0.4.1 | x86_64 | Alternative Kernel
i | kernel-default | package    | 2.6.32.9-0.5.1  | x86_64 | (System Packages)
  | kernel-default | srcpackage | 2.6.32.10-0.4.1 | noarch | Alternative Kernel
i | kernel-default | package    | 2.6.32.9-0.5.1  | x86_64 | (System Packages)
...
```

2. Specify the exact version when installing:

   ```
   zypper in kernel-default-2.6.32.10-0.4.1
   ```

3. When uninstalling a kernel, use the commands `zypper se -si 'kernel*'` to list all kernels installed and `zypper rm` *`PACKAGENAME-VERSION`* to remove the package.

Task Scheduler 13

Modern operating systems, such as SUSE® Linux Enterprise Server, normally run many different tasks at the same time. For example, you can be searching in a text file while receiving an e-mail and copying a long file to an external hard drive. These simple tasks require many additional processes to be run by the system. To provide each task with its required system resources, the Linux kernel needs a tool to distribute available system resources to individual tasks. And this is exactly what *task scheduler* does.

The following sections explain the most important terms related to process scheduling. They also introduce information about the task scheduler policy, scheduling algorithm, description of the task scheduler used by SUSE® Linux Enterprise Server, and references to other sources of relevant information.

13.1 Introduction

The Linux kernel controls the way tasks (or processes) are managed in the running system. Task scheduler, sometimes called *process scheduler*, is part of the kernel that decides which task to run next. It is one of the core components of a multitasking operating system (such as Linux) responsible for best utilizing system resources to guarantee that multiple tasks are being executed simultaneously.

13.1.1 Preemption

The theory behind task scheduling is very simple. If there are runnable processes in a system, at least one process must always be running. If there are more runnable processes than processors in a system, not all the processes can be running all the time.

Therefore, some processes need to be stopped temporarily, or *suspended*, so that others can be running again. The scheduler decides what process in the queue will run next.

As already mentioned, Linux, like all other Unix variants, is a *multitasking* operating system. That means that more tasks are processed can be running at the same time. Linux provides so called *preemptive* multitasking. In the world of preemptive multitasking, the scheduler decides when a process is suspended and another process resumed. This forced suspension is called *preemption*. All Unix flavors have been providing preemptive multitasking since the beginning.

13.1.2 Timeslice

The time period for which a process will be running before it is *preempted* is defined in advance. It is called a process' *timeslice* and represents a slice of the processor time that is provided to each process. By assigning timeslices, the scheduler makes global decisions for the running system, and prevents individual processes from dominating over the processor resources.

13.1.3 Process Priority

The scheduler evaluates processes based on their priority. To calculate the current priority of a process, the task scheduler uses complex algorithms. As a result, each process is given a value according to which it is “allowed” to run on a processor.

13.2 Process Classification

Processes are usually classified according to their purpose and behavior. Although the borderline is not always clearly distinct, we generally use two criteria to sort them. These criteria are independent and do not exclude each other.

One approach is to classify a process either *I/O-bound* or *processor-bound*.

I/O-bound
: I/O stands for Input/Output devices, such as keyboards, mice, or optical and hard disks. *I/O-bound processes* spend the majority of time submitting and waiting for requests. They are run very frequently, but for short time intervals, not to block other processes waiting for I/O requests.

processor-bound
: On the other hand, *processor-bound* tasks use their time to execute a code, and usually run until they are preempted by the scheduler. They do not block processes waiting for I/O requests, and, therefore, can be run less frequently but for longer time intervals.

Another approach is to divide processes to either *interactive*, *batch*, or *real-time* ones.

- *Interactive* processes spend a lot of time waiting for I/O requests, such as keyboard or mouse operations. The scheduler must wake up such process quickly on user request, or the user will find the environment unresponsive. The typical delay is approximately 100 ms. Office applications, text editors or image manipulation programs represent typical interactive processes.

- *Batch* processes often run in the background and do not need to be responsive. They usually receive lower priority from the scheduler. Multimedia converters, database search engines, or log files analyzers are typical examples of batch processes.

- *Real-time* processes must never be blocked by low-priority processes, and the scheduler guarantees a short response time to them. Applications for editing multimedia content are a good example here.

13.3 O(1) Scheduler

The Linux kernel version 2.6 introduced a new task scheduler. It is called O(1) scheduler (see Big O notation [`http://en.wikipedia.org/wiki/Big_O_notation`]), and it used to be the default scheduler up to the Linux kernel version 2.6.22. Its main task is to schedule tasks within a fixed amount of time, no matter how many runnable processes there are in the system.

The scheduler calculates the timeslices dynamically. However, to determine the appropriate timeslice is a complex task: Too long timeslices cause the system to be less interactive and responsive, while too short ones make the processor waste a lot of time on the overhead of switching the processes too frequently. The default timeslice is usually rather low, for example 20ms. The scheduler determines the timeslice based on priority of a process, which allows the processes with higher priority to run more often and for a longer time.

A process does not have to utilize all its timeslice at once. For instance, a process with a timeslice of 150ms does not have to be running for 150ms in one go. It can be running in five different schedule slots for 30ms instead. Interactive tasks typically benefit from this approach because they do not need such a large timeslice at once while they need to be responsive as long as possible.

The scheduler also assigns process priorities dynamically. It monitors the processes' behavior and, if needed, adjusts its priority. For example, a process which is being suspended for a long time is brought up by increasing its priority.

13.4 Completely Fair Scheduler

Since the Linux kernel version 2.6.23, a new approach has been taken to the scheduling of runnable processes. Completely Fair Scheduler (CFS) became the default Linux kernel scheduler. Since then, important changes and improvements have been made. The information in this chapter applies to SUSE Linux Enterprise Server with kernel version 2.6.32. The scheduler environment was divided into several parts, and three main new features were introduced:

Modular Scheduler Core

> The core of the scheduler was enhanced with *scheduling classes*. These classes are modular and represent scheduling policies.

Completely Fair Scheduler

> Introduced in kernel 2.6.23 and extended in 2.6.24, CFS tries to assure that each process obtains its “fair” share of the processor time.

Group Scheduling

> For example, if you split processes into groups according to which user is running them, CFS tries to provide each of these groups with the same amount of processor time.

As a result, CFS brings more optimized scheduling for both servers and desktops.

13.4.1 How CFS Works

CFS tries to guarantee a fair approach to each runnable task. To find the most balanced way of task scheduling, it uses the concept of *red-black tree*. A red-black tree is a type of self-balancing data search tree which provides inserting and removing entries in a reasonable way so that it remains well balanced. For more information, see the wiki pages of Red-black tree [http://en.wikipedia.org/wiki/Red_black_tree].

When a task enters into the *runqueue* (a planned time line of processes to be executed next), the scheduler records the current time. While the process waits for processor time, its "wait" value gets incremented by an amount derived from the total number of tasks currently in the runqueue and the process priority. As soon as the processor runs the task, its "wait" value gets decremented. If the value drops below a certain level, the task is preempted by the scheduler and other tasks get closer to the processor. By this algorithm, CFS tries to reach the ideal state where the "wait" value is always zero.

13.4.2 Grouping Processes

Since the Linux kernel version 2.6.24, CFS can be tuned to be fair to users or groups rather than to tasks only. Runnable tasks are then grouped to form entities, and CFS tries to be fair to these entities instead of individual runnable tasks. The scheduler also tries to be fair to individual tasks within these entities.

Tasks can be grouped in two mutually exclusive ways:

- By user IDs
- By kernel control groups.

The way the kernel scheduler lets you group the runnable tasks depends on setting the kernel compile-time options `CONFIG_FAIR_USER_SCHED` and `CONFIG_FAIR_CGROUP_SCHED`. The default setting in SUSE® Linux Enterprise Server 11 SP1 is to use control groups, which lets you create groups as needed. For more information, see Chapter 10, *Kernel Control Groups* (page 113).

13.4.3 Kernel Configuration Options

Basic aspects of the task scheduler behavior can be set through the kernel configuration options. Setting these options is part of the kernel compilation process. Because kernel compilation process is a complex task and out of this document's scope, refer to relevant source of information (for example `http://en.opensuse.org/Configure,_Build_and_Install_a_Custom_Linux_Kernel`).

WARNING: Kernel Compilation

If you run SUSE Linux Enterprise Server on a kernel that was not shipped with it, for example on a self-compiled kernel, you lose the entire support entitlement.

13.4.4 Terminology

Documents regarding task scheduling policy often use several technical terms which you need to know to understand the information correctly. Here are some of them:

Latency

Delay between the time a process is scheduled to run and the actual process execution.

Granularity

The relation between granularity and latency can be expressed by the following equation:

```
gran = ( lat / rtasks ) - ( lat / rtasks / rtasks )
```

where *gran* stands for granularity, *lat* stand for latency, and *rtasks* is the number of running tasks.

SCHED_BATCH

Scheduling policy designed for CPU-intensive tasks.

SCHED_OTHER

Default Linux time-sharing scheduling policy.

13.4.5 Runtime Tuning

The `sysctl` interface for examining and changing kernel parameters at runtime introduces important variables by means of which you can change the default behavior of the task scheduler. The syntax of the `sysctl` is simple, and all the following commands must be entered on the command line as `root`.

To read a value from a kernel variable, enter

```
sysctl variable
```

To assign a value, enter

```
sysctl variable=value
```

To get a list of all scheduler related `sysctl` variables, enter

```
sysctl -A | grep "sched" | grep -v "domain"
```

```
saturn.example.com:~ # sysctl -A | grep "sched" | grep -v "domain"
kernel.sched_child_runs_first = 0
kernel.sched_min_granularity_ns = 1000000
kernel.sched_latency_ns = 5000000
kernel.sched_wakeup_granularity_ns = 1000000
kernel.sched_shares_ratelimit = 250000
kernel.sched_tunable_scaling = 1
kernel.sched_shares_thresh = 4
kernel.sched_features = 15834238
kernel.sched_migration_cost = 500000
kernel.sched_nr_migrate = 32
kernel.sched_time_avg = 1000
kernel.sched_rt_period_us = 1000000
kernel.sched_rt_runtime_us = 950000
kernel.sched_compat_yield = 0
```

Note that variables ending with '_ns' accept values in nanoseconds.

According to Ingo Molnar, the author of CFS, there is only one important tuning option for CFS: `kernel.sched_min_granularity_ns`. It can be used to fine-tune the scheduler from low-latency response suitable for desktops, to server deployment giving more priority to batch tasks.

A list of the most important task scheduler `sysctl` tuning variables with a short description follows:

`sched_child_runs_first`
: Specifies whether a child process gets scheduled next after the `fork` call. This is the default value. If set to 0, the parent process is prioritized.

`sched_min_granularity_ns`
: Specifies the minimum granularity for preempting processor-bound tasks.

`sched_latency_ns`
: Specifies the preferred latency for preempting processor-bound tasks.

`sched_wakeup_granularity_ns`
: Specifies the wake-up granularity for SCHED_OTHER scheduling policy.

`sched_batch_wakeup_granularity_ns`
: Specifies the wake-up granularity for SCHED_BATCH scheduling policy.

`sched_features`
: Provides information about specific debugging features.

`sched_compat_yield`
: Moves the yielding task to the last position of the red-blue tree. The default is 0.

`sched_stat_granularity_ns`
: Specifies the granularity for collecting task scheduler statistics.

`sched_nr_migrate`
: Controls how many tasks can be moved across processors through migration software interrupts (softirq). If a large number of tasks is created by SCHED_OTHER policy, they will all be run on the same processor. The default is 32. Increasing this value gives a performance boost to large SCHED_OTHER threads at the expense of increased latencies for real-time tasks.

13.4.6 Debugging Interface and Scheduler Statistics

CFS comes with a new improved debugging interface, and provides runtime statistics information. Relevant files were added to the `/proc` file system, which can be examined simply with the `cat` or `less` command. A list of the related `/proc` files follows with their short description:

`/proc/sched_debug`

Contains the current values of all tunable variables (see Section 13.4.5, “Runtime Tuning” (page 151)) that affect the task scheduler behavior, CFS statistics, and information about the runqueue on all available processors.

```
saturn.example.com:~ # less /proc/sched_debug
Sched Debug Version: v0.09, 2.6.32.8-0.3-default #1
now at 2413026096.408222 msecs
  .jiffies                                 : 4898148820
  .sysctl_sched_latency                    : 5.000000
  .sysctl_sched_min_granularity            : 1.000000
  .sysctl_sched_wakeup_granularity         : 1.000000
  .sysctl_sched_child_runs_first           : 0.000000
  .sysctl_sched_features                   : 15834238
  .sysctl_sched_tunable_scaling            : 1 (logaritmic)

cpu#0, 1864.411 MHz
  .nr_running                    : 1
  .load                          : 1024
  .nr_switches                   : 37539000
  .nr_load_updates               : 22950725
[...]
cfs_rq[0]:/
  .exec_clock                    : 52940326.803842
  .MIN_vruntime                  : 0.000001
  .min_vruntime                  : 54410632.307072
  .max_vruntime                  : 0.000001
[...]
rt_rq[0]:/
  .rt_nr_running                 : 0
  .rt_throttled                  : 0
  .rt_time                       : 0.000000
  .rt_runtime                    : 950.000000

runnable tasks:
  task  PID   tree-key    switches prio exec-runtime    sum-exec sum-sleep
--------------------------------------------------------------------------
R cat 16884 54410632.307072    0   120  54410632.307072 13.836804 0.000000
```

`/proc/schedstat`

Displays statistics relevant to the current runqueue. Also domain-specific statistics for SMP systems are displayed for all connected processors. Because the output format is not user-friendly, read the contents of `/usr/src/linux/Documentation/scheduler/sched-stats.txt` for more information.

`/proc/`*`PID`*`/sched`

Displays scheduling information on the process with id *`PID`*.

```
saturn.example.com:~ # cat /proc/`pidof nautilus`/sched
 nautilus (4009, #threads: 1)
 ---------------------------------------------------------
 se.exec_start                      :    2419575150.560531
 se.vruntime                        :      54549795.870151
 se.sum_exec_runtime                :       4867855.829415
 se.avg_overlap                     :             0.401317
 se.avg_wakeup                      :             3.247651
 se.avg_running                     :             0.323432
 se.wait_start                      :             0.000000
 se.sleep_start                     :    2419575150.560531
 [...]
 nr_voluntary_switches              :               938552
 nr_involuntary_switches            :                71872
 se.load.weight                     :                 1024
 policy                             :                    0
 prio                               :                  120
 clock-delta                        :                  109
```

13.5 For More Information

To get a compact knowledge about Linux kernel task scheduling, you need to explore several information sources. Here are some of them:

- For task scheduler System Calls description, see the relevant manual page (for example `man 2 sched_setaffinity`).
- General information on scheduling is described in Scheduling [`http://en.wikipedia.org/wiki/Scheduling_(computing)`] wiki page.
- General information on Linux task scheduling is described in Inside the Linux scheduler [`http://www.ibm.com/developerworks/linux/library/l-scheduler/`].

- Information specific to Completely Fair Scheduler is available in Multiprocessing with the Completely Fair Scheduler [`http://www.ibm.com/developerworks/linux/library/l-cfs/?ca=dgr-lnxw06CFC4Linux`]

- Information specific to tuning Completely Fair Scheduler is available in Tuning the Linux Kernel's Completely Fair Scheduler [`http://www.hotaboutlinux.com/2010/01/tuning-the-linux-kernels-completely-fair-scheduler/`]

- A useful lecture on Linux scheduler policy and algorithm is available in `http://www.inf.fu-berlin.de/lehre/SS01/OS/Lectures/Lecture08.pdf`.

- A good overview of Linux process scheduling is given in *Linux Kernel Development* by Robert Love (ISBN-10: 0-672-32512-8). See `http://www.informit.com/articles/article.aspx?p=101760`.

- A very comprehensive overview of the Linux kernel internals is given in *Understanding the Linux Kernel* by Daniel P. Bovet and Marco Cesati (ISBN 978-0-596-00565-8).

- Technical information about task scheduler is covered in files under `/usr/src/linux/Documentation/scheduler`.

Network Tuning 14

The network subsystem is rather complex and its tuning highly depends on the system use scenario and also on external factors such as software clients or hardware components (switches, routers, or gateways) in your network. The Linux kernel aims more at reliability and low latency than low overhead and high throughput. Other settings can mean less security, but better performance.

14.1 Configurable Kernel Socket Buffers

Networking is largely based on the TCP/IP protocol and a socket interface for communication; for more information about TCP/IP, see Chapter 18, *Basic Networking* (↑*Administration Guide*). The Linux kernel handles data it receives or sends via the socket interface in socket buffers. These kernel socket buffers are tunable.

IMPORTANT: TCP Autotuning

Since kernel version 2.6.17 full autotuning with 4 MB maximum buffer size exists. This means that manual tuning in most cases will not improve networking performance considerably. It is often the best not to touch the following variables, or, at least, to check the outcome of tuning efforts carefully.

If you update from an older kernel, it is recommended to remove manual TCP tunings in favor of the autotuning feature.

The special files in the /proc file system can modify the size and behavior of kernel socket buffers; for general information about the /proc file system, see Section 2.6, “The /proc File System” (page 32). Find networking related files in:

```
/proc/sys/net/core
/proc/sys/net/ipv4
/proc/sys/net/ipv6
```

General net variables are explained in the kernel documentation (linux/Documentation/networking/sysctl/net.txt). Special ipv4 variables are explained in linux/Documentation/networking/ip-sysctl.txt and linux/Documentation/networking/ipvs-sysctl.txt.

In the /proc file system, for example, it is possible to either set the Maximum Socket Receive Buffer and Maximum Socket Send Buffer for all protocols, or both these options for the TCP protocol only (in ipv4) and, thu,s overriding the setting for all protocols (in core).

/proc/sys/net/ipv4/tcp_moderate_rcvbuf
: If /proc/sys/net/ipv4/tcp_moderate_rcvbuf is set to 1, autotuning is active and buffer size is adjusted dynamically.

/proc/sys/net/ipv4/tcp_rmem
: The three values setting the minimum, initial, and maximum size of the Memory Receive Buffer per connection. They define the actual memory usage, not just TCP window size.

/proc/sys/net/ipv4/tcp_wmem
: The same as tcp_rmem, but just for Memory Send Buffer per connection.

/proc/sys/net/core/rmem_max
: Set to limit the maximum receive buffer size that applications can request.

/proc/sys/net/core/wmem_max
: Set to limit the maximum send buffer size that applications can request.

Via /proc it is possible to disable TCP features that you do not need (all TCP features are switched on by default). For example, check the following files:

/proc/sys/net/ipv4/tcp_timestamps
: TCP timestamps are defined in RFC1323.

`/proc/sys/net/ipv4/tcp_window_scaling`
TCP window scaling is also defined in RFC1323.

`/proc/sys/net/ipv4/tcp_sack`
Select acknowledgments (SACKS).

Use `sysctl` to read or write variables of the `/proc` file system. `sysctl` is preferable to `cat` (for reading) and `echo` (for writing), because it also reads settings from `/etc/sysctl.conf` and, thus, those settings survive reboots reliably. With `sysctl` you can read all variables and their values easily; as `root` use the following command to list TCP related settings:

```
sysctl -a | grep tcp
```

NOTE: Side-Effects of Tuning Network Variables

Tuning network variables can affect other system resources such as CPU or memory use.

14.2 Detecting Network Bottlenecks and Analyzing Network Traffic

Before starting with network tuning, it is important to isolate network bottlenecks and network traffic patterns. There are some tools that can help you with detecting those bottlenecks.

The following tools can help analyzing your network traffic: `netstat`, `tcpdump`, and `wireshark`. Wireshark is a network traffic analyser.

14.3 Netfilter

The Linux firewall and masquerading features are provided by the Netfilter kernel modules. This is a highly configurable rule based framework. If a rule matches a packet, Netfilter accepts or denies it or takes special action (“target”) as defined by rules such as address translation.

There are quite some properties, Netfilter is able to take into account. Thus, the more rules are defined, the longer packet processing may last. Also advanced connection tracking could be rather expensive and, thus, slowing down overall networking.

For more information, see the home page of the Netfilter and iptables project, http://www.netfilter.org

14.4 For More Information

- Eduardo Ciliendo, Takechika Kunimasa: “Linux Performance and Tuning Guidelines” (2007), esp. sections 1.5, 3.5, and 4.7: http://www.redbooks.ibm.com/redpapers/abstracts/redp4285.html

- John Heffner, Matt Mathis: “Tuning TCP for Linux 2.4 and 2.6” (2006): http://www.psc.edu/networking/projects/tcptune/#Linux

Part VI. Handling System Dumps

Tracing Tools 15

SUSE Linux Enterprise Server comes with a number of tools that help you obtain useful information about your system. You can use the information for various purposes, for example, to debug and find problems in your program, to discover places causing performance drops, or to trace a running process to find out what system resources it uses. The tools are mostly part of the installation media, otherwise you can install them from the downloadable SUSE Software Development Kit.

NOTE: Tracing and Impact on Performance

While a running process is being monitored for system or library calls, the performance of the process is heavily reduced. You are advised to use tracing tools only for the time you need to collect the data.

15.1 Tracing System Calls with strace

The `strace` command traces system calls of a process and signals received by the process. `strace` can either run a new command and trace its system calls, or you can attach `strace` to an already running command. Each line of the command's output contains the system call name, followed by its arguments in parenthesis and its return value.

To run a new command and start tracing its system calls, enter the command to be monitored as you normally do, and add `strace` at the beginning of the command line:

```
tux@mercury:~> strace ls
execve("/bin/ls", ["ls"], [/* 52 vars */]) = 0
brk(0)                                  = 0x618000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) \
 = 0x7f9848667000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) \
 = 0x7f9848666000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT \
(No such file or directory)
open("/etc/ld.so.cache", O_RDONLY)      = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=200411, ...}) = 0
mmap(NULL, 200411, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f9848635000
close(3)                                = 0
open("/lib64/librt.so.1", O_RDONLY)     = 3
[...]
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) \
= 0x7fd780f79000
write(1, "Desktop\nDocuments\nbin\ninst-sys\n", 31Desktop
Documents
bin
inst-sys
) = 31
close(1)                                = 0
munmap(0x7fd780f79000, 4096)            = 0
close(2)                                = 0
exit_group(0)                           = ?
```

To attach `strace` to an already running process, you need to specify the `-p` with the process ID (`PID`) of the process that you want to monitor:

```
tux@mercury:~> strace -p `pidof mysqld`
Process 2868 attached - interrupt to quit
select(15, [13 14], NULL, NULL, NULL)   = 1 (in [14])
fcntl(14, F_SETFL, O_RDWR|O_NONBLOCK)   = 0
accept(14, {sa_family=AF_FILE, NULL}, [2]) = 31
fcntl(14, F_SETFL, O_RDWR)              = 0
getsockname(31, {sa_family=AF_FILE, path="/var/run/mysql"}, [28]) = 0
fcntl(31, F_SETFL, O_RDONLY)            = 0
fcntl(31, F_GETFL)                      = 0x2 (flags O_RDWR)
fcntl(31, F_SETFL, O_RDWR|O_NONBLOCK)   = 0
[...]
setsockopt(31, SOL_IP, IP_TOS, [8], 4)  = -1 EOPNOTSUPP (Operation \
not supported)
clone(child_stack=0x7fd1864801f0, flags=CLONE_VM|CLONE_FS|CLONE_ \
FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_ \
PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tidptr=0x7fd1864809e0, \
tls=0x7fd186480910, child_tidptr=0x7fd1864809e0) = 21993
select(15, [13 14], NULL, NULL, NULL
```

The `-e` option understands several sub-options and arguments. For example, to trace all attempts to open or write to a particular file, use the following:

```
tux@mercury:~> strace -e trace=open,write ls ~
open("/etc/ld.so.cache", O_RDONLY)       = 3
open("/lib64/librt.so.1", O_RDONLY)      = 3
open("/lib64/libselinux.so.1", O_RDONLY) = 3
open("/lib64/libacl.so.1", O_RDONLY)     = 3
open("/lib64/libc.so.6", O_RDONLY)       = 3
open("/lib64/libpthread.so.0", O_RDONLY) = 3
[...]
open("/usr/lib/locale/cs_CZ.utf8/LC_CTYPE", O_RDONLY) = 3
open(".", O_RDONLY|O_NONBLOCK|O_DIRECTORY|O_CLOEXEC) = 3
write(1, "addressbook.db.bak\nbin\ncxoffice\n"..., 311) = 311
```

To trace only network related system calls, use `-e trace=network`:

```
tux@mercury:~> strace -e trace=network -p 26520
Process 26520 attached - interrupt to quit
socket(PF_NETLINK, SOCK_RAW, 0)         = 50
bind(50, {sa_family=AF_NETLINK, pid=0, groups=00000000}, 12) = 0
getsockname(50, {sa_family=AF_NETLINK, pid=26520, groups=00000000}, \
[12]) = 0
sendto(50, "\24\0\0\0\26\0\1\3~p\315K\0\0\0\0\0\0\0\0", 20, 0,
{sa_family=AF_NETLINK, pid=0, groups=00000000}, 12) = 20
[...]
```

The `-c` calculates the time the kernel spent on each system call:

```
tux@mercury:~> strace -c find /etc -name xorg.conf
/etc/X11/xorg.conf
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 32.38    0.000181         181         1           execve
 22.00    0.000123           0       576           getdents64
 19.50    0.000109           0       917        31 open
 19.14    0.000107           0       888           close
  4.11    0.000023           2        10           mprotect
  0.00    0.000000           0         1           write
[...]
  0.00    0.000000           0         1           getrlimit
  0.00    0.000000           0         1           arch_prctl
  0.00    0.000000           0         3         1 futex
  0.00    0.000000           0         1           set_tid_address
  0.00    0.000000           0         4           fadvise64
  0.00    0.000000           0         1           set_robust_list
------ ----------- ----------- --------- --------- ----------------
100.00    0.000559                  3633        33 total
```

To trace all child processes of a process, use `-f`:

```
tux@mercury:~> strace -f rcapache2 status
execve("/usr/sbin/rcapache2", ["rcapache2", "status"], [/* 81 vars */]) = 0
brk(0)                                  = 0x69e000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) \
= 0x7f3bb553b000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) \
= 0x7f3bb553a000
[...]
[pid  4823] rt_sigprocmask(SIG_SETMASK, [],  <unfinished ...>
[pid  4822] close(4 <unfinished ...>
[pid  4823] <... rt_sigprocmask resumed> NULL, 8) = 0
[pid  4822] <... close resumed> )       = 0
[...]
[pid  4825] mprotect(0x7fc42cbbd000, 16384, PROT_READ) = 0
[pid  4825] mprotect(0x60a000, 4096, PROT_READ) = 0
[pid  4825] mprotect(0x7fc42cde4000, 4096, PROT_READ) = 0
[pid  4825] munmap(0x7fc42cda2000, 261953) = 0
[...]
[pid  4830] munmap(0x7fb1fff10000, 261953) = 0
[pid  4830] rt_sigprocmask(SIG_BLOCK, NULL, [], 8) = 0
[pid  4830] open("/dev/tty", O_RDWR|O_NONBLOCK) = 3
[pid  4830] close(3)
[...]
read(255, "\n\n# Inform the caller not only v"..., 8192) = 73
rt_sigprocmask(SIG_BLOCK, NULL, [], 8)  = 0
rt_sigprocmask(SIG_BLOCK, NULL, [], 8)  = 0
exit_group(0)
```

If you need to analyze the output of `strace` and the output messages are too long to be inspected directly in the console window, use `-o`. In that case, unnecessary messages, such as information about attaching and detaching processes, are suppressed. You can also suppress these messages (normally printed on the standard output) with `-q`. To optionally prepend timestamps to each line with a system call, use `-t`:

```
tux@mercury:~> strace -t -o strace_sleep.txt sleep 1; more strace_sleep.txt
08:44:06 execve("/bin/sleep", ["sleep", "1"], [/* 81 vars */]) = 0
08:44:06 brk(0)                         = 0x606000
08:44:06 mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, \
-1, 0) = 0x7f8e78cc5000
[...]
08:44:06 close(3)                       = 0
08:44:06 nanosleep({1, 0}, NULL)        = 0
08:44:07 close(1)                       = 0
08:44:07 close(2)                       = 0
08:44:07 exit_group(0)                  = ?
```

The behavior and output format of strace can be largely controlled. For more information, see the relevant manual page (man 1 strace).

15.2 Tracing Library Calls with ltrace

`ltrace` traces dynamic library calls of a process. It is used in a similar way to `strace`, and most of their parameters have a very similar or identical meaning. By default, `ltrace` uses `/etc/ltrace.conf` or `~/.ltrace.conf` configuration files. You can, however, specify an alternative one with the `-F` *`config_file`* option.

In addition to library calls, `ltrace` with the `-S` option can trace system calls as well:

```
tux@mercury:~> ltrace -S -o ltrace_find.txt find /etc -name \
xorg.conf; more ltrace_find.txt
SYS_brk(NULL)                                              = 0x00628000
SYS_mmap(0, 4096, 3, 34, 0xffffffff)                       = 0x7f1327ea1000
SYS_mmap(0, 4096, 3, 34, 0xffffffff)                       = 0x7f1327ea0000
[...]
fnmatch("xorg.conf", "xorg.conf", 0)                       = 0
free(0x0062db80)                                           = <void>
__errno_location()                                         = 0x7f1327e5d698
__ctype_get_mb_cur_max(0x7fff25227af0, 8192, 0x62e020, -1, 0) = 6
__ctype_get_mb_cur_max(0x7fff25227af0, 18, 0x7f1327e5d6f0, 0x7fff25227af0,
0x62e031) = 6
__fprintf_chk(0x7f1327821780, 1, 0x420cf7, 0x7fff25227af0, 0x62e031
<unfinished ...>
SYS_fstat(1, 0x7fff25227230)                               = 0
SYS_mmap(0, 4096, 3, 34, 0xffffffff)                       = 0x7f1327e72000
SYS_write(1, "/etc/X11/xorg.conf\n", 19)                   = 19
[...]
```

You can change the type of traced events with the `-e` option. The following example prints library calls related to `fnmatch` and `strlen` functions:

```
tux@mercury:~> ltrace -e fnmatch,strlen find /etc -name xorg.conf
[...]
fnmatch("xorg.conf", "xorg.conf", 0)             = 0
strlen("Xresources")                             = 10
strlen("Xresources")                             = 10
strlen("Xresources")                             = 10
fnmatch("xorg.conf", "Xresources", 0)            = 1
strlen("xorg.conf.install")                      = 17
[...]
```

To display only the symbols included in a specific library, use `-l` *`/path/to/library`*:

```
tux@mercury:~> ltrace -l /lib64/librt.so.1 sleep 1
clock_gettime(1, 0x7fff4b5c34d0, 0, 0, 0)                  = 0
clock_gettime(1, 0x7fff4b5c34c0, 0xffffffffff600180, -1, 0) = 0
+++ exited (status 0) +++
```

You can make the output more readable by indenting each nested call by the specified number of space with the `-n` *`num_of_spaces`*.

15.3 Debugging and Profiling with Valgrind

Valgrind is a set of tools to debug and profile your programs so that they can run faster and with less errors. Valgrind can detect problems related to memory management and threading, or can also serve as a framework for building new debugging tools.

15.3.1 Installation

Valgrind is not shipped with standard SUSE Linux Enterprise Server distribution. To install it on your system, you need to obtain SUSE Software Development Kit, and either install it as an Add-On product and run

```
zypper install valgrind
```

or browse through the SUSE Software Development Kit directory tree, locate the Valgrind package and install it with

```
rpm -i valgrind-version_architecture.rpm
```

15.3.2 Supported Architectures

Valgrind runs on the following architectures:

- i386
- x86_64 (AMD-64)
- ppc
- ppc64

15.3.3 General Information

The main advantage of Valgrind is that it works with existing compiled executables. You do not have to recompile or modify your programs to make use of it. Run Valgrind like this:

```
valgrind valgrind_options your-prog your-program-options
```

Valgrind consists of several tools, and each provides specific functionality. Information in this section is general and valid regardless of the used tool. The most important configuration option is `--tool` . This option tells Valgrind which tool to run. If you omit this option, `memcheck` is selected by default. For example, if you want to run `find ~ -name .bashrc` with Valgrind's `memcheck` tools, enter the following in the command line:

```
valgrind --tool=memcheck find ~ -name .bashrc
```

A list of standard Valgrind tools with a brief description follows:

`memcheck`
: Detects memory errors. It helps you tune your programs to behave correctly.

`cachegrind`
: Profiles cache prediction. It helps you tune your programs to run faster.

`callgrind`
: Works in a similar way to `cachegrind` but also gathers additional cache-profiling information.

`exp-drd`
: Detects thread errors. It helps you tune your multi-threaded programs to behave correctly.

`helgrind`
: Another thread error detector. Similar to `exp-drd` but uses different techniques for problem analysis.

`massif`
A heap profiler. Heap is an area of memory used for dynamic memory allocation. This tool helps you tune your program to use less memory.

`lackey`
An example tool showing instrumentation basics.

15.3.4 Default Options

Valgrind can read options at start-up. There are three places which Valgrind checks:

1. The file `.valgrindrc` in the home directory of the user who runs Valgrind.

2. The environment variable `$VALGRIND_OPTS`

3. The file `.valgrindrc` in the current directory where Valgrind is runned from.

These resources are parsed exactly in this order, while later given options take precedence over earlier processed options. Options specific to a particular Valgrind tool must be prefixed with the tool name and a colon. For example, if you want `cachegrind` to always write profile data to the `/tmp/cachegrind_`*`PID`*`.log`, add the following line to the `.valgrindrc` file in your home directory:

```
--cachegrind:cachegrind-out-file=/tmp/cachegrind_%p.log
```

15.3.5 How Valgrind Works

Valgrind takes control of your executable before it starts. It reads debugging information from the executable and related shared libraries. The executable's code is redirected to the selected Valgrind tool, and the tool adds its own code to handle its debugging. Then the code is handed back to the Valgrind core and the execution continues.

For example, `memcheck` adds its code, which checks every memory access. As a consequence, the program runs much slower than in the native execution environment.

Valgrind simulates every instruction of your program. Therefore, it not only checks the code of your program, but also all related libraries (including the C library), libraries used for graphical environment, and so on. If you try to detect errors with Valgrind, it also detects errors in associated libraries (like C, X11, or Gtk libraries). Because you probably do not need these errors, Valgrind can selectively, suppress these error messages to suppression files. The `--gen-suppressions=yes` tells Valgrind to report these suppressions which you can copy to a file.

Note that you should supply a real executable (machine code) as an Valgrind argument. Therefore, if your application is run, for example, from a shell or a Perl script you will by mistake get error reports related to `/bin/sh` (or `/usr/bin/perl`). In such case, you can use `--trace-children=`*`yes`* or, which is better, supply a real executable to avoid any processing confusion.

15.3.6 Messages

During its runtime, Valgrind reports messages with detailed errors and important events. The following example explains the messages:

```
tux@mercury:~> valgrind --tool=memcheck find ~ -name .bashrc
[...]
==6558== Conditional jump or move depends on uninitialised value(s)
==6558==    at 0x400AE79: _dl_relocate_object (in /lib64/ld-2.11.1.so)
==6558==    by 0x4003868: dl_main (in /lib64/ld-2.11.1.so)
[...]
==6558== Conditional jump or move depends on uninitialised value(s)
==6558==    at 0x400AE82: _dl_relocate_object (in /lib64/ld-2.11.1.so)
==6558==    by 0x4003868: dl_main (in /lib64/ld-2.11.1.so)
[...]
==6558== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
==6558== malloc/free: in use at exit: 2,228 bytes in 8 blocks.
==6558== malloc/free: 235 allocs, 227 frees, 489,675 bytes allocated.
==6558== For counts of detected errors, rerun with: -v
==6558== searching for pointers to 8 not-freed blocks.
==6558== checked 122,584 bytes.
==6558==
==6558== LEAK SUMMARY:
==6558==    definitely lost: 0 bytes in 0 blocks.
==6558==      possibly lost: 0 bytes in 0 blocks.
==6558==    still reachable: 2,228 bytes in 8 blocks.
==6558==         suppressed: 0 bytes in 0 blocks.
==6558== Rerun with --leak-check=full to see details of leaked memory.
```

The `==6558==` introduces Valgrind's messages and contains the process ID number (PID). You can easily distinguish Valgrind's messages from the output of the program itself, and decide which messages belong to a particular process.

To make Valgrind's messages more detailed, use `-v` or even `-v -v`.

Basically, you can make Valgrind send its messages to three different places:

1. By default, Valgrind sends its messages to the file descriptor 2, which is the standard error output. You can tell Valgrind to send its messages to any other file descriptor with the `--log-fd=`*`file_descriptor_number`* option.

2. The second and probably more useful way is to send Valgrind's messages to a file with `--log-file=`*`filename`*. This option accepts several variables, for example, `%p` gets replaced with the PID of the currently profiled process. This way you can send messages to different files based on their PID. `%q{env_var}` is replaced with the value of the related `env_var` environment variable.

   The following example checks for possible memory errors during the Apache Web server restart, while following children processes and writing detailed Valgrind's messages to separate files distinguished by the current process PID:

   ```
   tux@mercury:~> valgrind -v --tool=memcheck --trace-children=yes \
   --log-file=valgrind_pid_%p.log rcapache2 restart
   ```

   This process created 52 log files in the testing system, and took 75 seconds instead of the usual 7 seconds needed to run `rcapache2 restart` without Valgrind, which is approximately 10 times more.

   ```
   tux@mercury:~> ls -1 valgrind_pid_*log
   valgrind_pid_11780.log
   valgrind_pid_11782.log
   valgrind_pid_11783.log
   [...]
   valgrind_pid_11860.log
   valgrind_pid_11862.log
   valgrind_pid_11863.log
   ```

3. You may also prefer to send the Valgrind's messages over the network. You need to specify the `aa.bb.cc.dd` IP address and `port_num` port number of the network socket with the `--log-socket=`*`aa.bb.cc.dd:port_num`* option. If you omit the port number, 1500 will be used.

   It is useless to send Valgrind's messages to a network socket if no application is capable of receiving them on the remote machine. That is why `valgrind-listener`, a simple listener, is shipped together with Valgrind. It accepts connections on the specified port and copies everything it receives to the standard output.

15.3.7 Error Messages

Valgrind remembers all error messages, and if it detects a new error, the error is compared against old error messages. This way Valgrind checks for duplicate error messages. In case of a duplicate error, it is recorded but no message is shown. This mechanism prevents you from being overwhelmed by millions of duplicate errors.

The `-v` option will add a summary of all reports (sorted by their total count) to the end of the Valgrind's execution output. Moreover, Valgrind stops collecting errors if it detects either 1000 different errors, or 10 000 000 errors in total. If you want to suppress this limit and wish to see all error messages, use `--error-limit=no`.

Some errors usually cause other ones. Therefore, fix errors in the same order as they appear and re-check the program continuously.

15.4 For More Information

- For a complete list of options related to the described tracing tools, see the corresponding man page (`man 1 strace`, `man 1 ltrace`, and `man 1 valgrind`).

- To describe advanced usage of Valgrind is beyond the scope of this document. It is very well documented, see Valgrind User Manual [http://valgrind.org/docs/manual/manual.html]. These pages are indispensable if you need more advanced information on Valgrind or the usage and purpose of its standard tools.

Kexec and Kdump 16

Kexec is a tool to boot to another kernel from the currently running one. You can perform faster system reboots without any hardware initialization. You can also prepare the system to boot to another kernel if the system crashes.

16.1 Introduction

With Kexec, you can replace the running kernel with another without a hard reboot. The tool is useful for several reasons:

- Faster system rebooting

  If, for any reasons, you have to reboot the system frequently, Kexec can save you significant time.

- Avoiding unreliable firmware and hardware

  Nowadays, computer hardware is complex and serious problems may occur during the system start-up. You cannot always replace unreliable hardware immediately. Kexec boots the kernel to a controlled environment with the hardware already initialized. The risk of unsuccessful system start is minimized.

- Saving the dump of a crashed kernel

  Kexec preserves the contents of the physical memory. After the production kernel fails, the capture kernel, which runs in a reserved memory, saves the state of the failed kernel. The saved image can help you with the subsequent analysis.

- Booting without GRUB or LILO configuration

  When the system boots a kernel with Kexec, it skips the boot loader stage. Normal booting procedure can fail due to an error in the boot loader configuration. With Kexec, you do not depend on a working boot loader configuration.

16.2 Required Packages

If you aim to use Kexec on SUSE® Linux Enterprise Server to speed up reboots or avoid potential hardware problems, you need to install the `kexec-tools` package.

The package `kexec-tools` contains a script called `kexec-bootloader`. It reads the boot loader configuration and runs Kexec with the same kernel options as the normal boot loader does. `kexec-bootloader -h` gives you the list of possible options.

To set up an environment that helps you obtain useful debug information in case of a kernel crash, you need to install `makedumpfile` in addition.

The preferred method to use Kdump in the SUSE environment is through the YaST Kdump module. Install the package `yast2-kdump` by entering `zypper install yast2-kdump` in the command line as `root`.

16.3 Kexec Internals

The most important component of Kexec is the `/sbin/kexec` command. You can load a kernel with Kexec in two different ways:

- With `kexec -l` *`kernel_image`* to load the kernel to the address space of a production kernel for regular reboot. You can later boot to this kernel with `kexec -e`.

- With `kexec -p` *`kernel_image`* to load the kernel to a reserved area of memory. This kernel will be booted automatically when the system crashes.

If you want to boot another kernel and preserve the data of the production kernel when the system crashes, you need to reserve a dedicated area of the system memory. The production kernel never loads to this area because it must be available at all times. It is used for the capture kernel so that the memory pages of the production kernel can be preserved. You reserve the area with `crashkernel=`*`size@offset`* as a command line parameter of the production kernel. Note that this is not a parameter of the capture kernel. The capture kernel does not use Kexec at all.

The capture kernel is loaded to the reserved area and waits for the kernel to crash. Then Kdump tries to invoke the capture kernel in the most simple way because the production kernel is no longer reliable at this stage. This means that even Kdump can fail.

To load the capture kernel, you need to include the kernel boot parameters. Usually, the initial RAM file system is used for booting. You can specify it with `--initrd=`*`filename`*. With `--append=`*`cmdline`*, you append options to the command line of the kernel to boot. It is helpful to include the command line of the production kernel if these options are necessary for the kernel to boot. You can simply copy the command line with `--append=`*`"$(cat /proc/cmdline)"`* or add more options with `--append=`*`"$(cat /proc/cmdline) more_options"`*.

You can always unload the previously loaded kernel. To unload a kernel that was loaded with the `-l` option, use the `kexec -u` command. To unload a crash kernel loaded with the `-p` option, use `kexec -p -u` command.

16.4 Basic Kexec Usage

To verify if your Kexec environment works properly, follow these steps:

1. Make sure no users are currently logged in and no important services are running on the system.

2. Log in as `root`.

3. Switch to runlevel 1 with `telinit 1`

4. Load the new kernel to the address space of the production kernel with the following command:

   ```
   kexec -l /boot/vmlinuz --append="$(cat /proc/cmdline)"
   --initrd=/boot/initrd
   ```

5. Unmount all mounted file systems except the root file system with `umount -a`

   IMPORTANT: Unmounting Root Filesystem

   Unmounting all file systems will most likely produce a `device is busy` warning message. The root file system cannot be unmounted if the system is running. Ignore the warning.

6. Remount the root file system in read-only mode:

   ```
   mount -o remount,ro /
   ```

7. Initiate the reboot of the kernel that you loaded in Step 4 (page 177) with `kexec -e`

It is important to unmount the previously mounted disk volumes in read-write mode. The reboot syscall acts immediately upon calling. Hard drive volumes mounted in read-write mode neither synchronize nor unmount automatically. The new kernel may find them “dirty”. Read-only disk volumes and virtual file systems do not need to be unmounted. Refer to `/etc/mtab` to determine which file systems you need to unmount.

The new kernel previously loaded to the address space of the older kernel rewrites it and takes control immediately. It displays the usual start-up messages. When the new kernel boots, it skips all hardware and firmware checks. Make sure no warning messages appear. All the file systems are supposed to be clean if they had been unmounted.

16.5 How to Configure Kexec for Routine Reboots

Kexec is often used for frequent reboots. For example, if it takes a long time to run through the hardware detection routines or if the start-up is not reliable.

NOTE: Rebooting with Kexec

In previous versions of SUSE® Linux Enterprise Server, you had to manually edit the configuration file `/etc/sysconfig/shutdown` and the init script

`/etc/init.d/halt` to use Kexec to reboot the system. You no longer need to edit any system files, since version 11 is already configured for Kexec reboots.

Note that firmware as well as the boot loader are not used when the system reboots with Kexec. Any changes you make to the boot loader configuration will be ignored until the computer performs a hard reboot.

16.6 Basic Kdump Configuration

You can use Kdump to save kernel dumps. If the kernel crashes, it is useful to copy the memory image of the crashed environment on the file system. You can then debug the dump file to find the cause of the kernel crash. This is called “core dump” .

Kdump works similar to Kexec (seeChapter 16, *Kexec and Kdump* (page 175)). The capture kernel is executed after the running production kernel crashes. The difference is that Kexec replaces the production kernel with the capture kernel. With Kdump, you still have access to the memory space of the crashed production kernel. You can save the memory snapshot of the crashed kernel in the environment of the Kdump kernel.

You can either configure Kdump manually or with YaST.

16.6.1 Manual Kdump Configuration

Kdump reads its configuration from the `/etc/sysconfig/kdump` file. To make sure that Kdump works on your system, its default configuration is sufficient. To use Kdump with the default settings,follow these steps:

1. Append the following kernel command line option to your boot loader configuration, and reboot the system:

   ```
   crashkernel=size@offset
   ```

   You can find the corresponding values for *`size`* and *`offset`* in the following table:

Table 16.1 Recommended Values for Additional Kernel Command Line Parameters

Architecture	Recommended value
i386 and x86-64	crashkernel=64M@16M
IA64	crashkernel=256M (small systems) or crashkernel=512M (larger systems)
ppc64	crashkernel=128M or crashkernel=256M (larger systems)

2 Enable Kdump init script:

```
chkconfig boot.kdump on
```

3 You can edit the options in `/etc/sysconfig/kdump`. Reading the comments will help you understand the meaning of individual options.

4 Execute the init script once with `rckdump start`, or reboot the system.

After configuring Kdump with the default values, check if it works as expected. Make sure that no users are currently logged in and no important services are running on your system. Then follow these steps:

1 Switch to runlevel 1 with `telinit 1`

2 Unmount all the disk file systems except the root file system with `umount -a`

3 Remount the root file system in read-only mode: `mount -o remount,ro /`

4 Invoke “kernel panic” with the `procfs` interface to Magic SysRq keys:

```
echo c >/proc/sysrq-trigger
```

IMPORTANT: The Size of Kernel Dumps

The KDUMP_KEEP_OLD_DUMPS option controls the number of preserved kernel dumps (default is 5). Without compression, the size of the dump can take up to the size of the physical RAM memory. Make sure you have sufficient space on the /var partition.

The capture kernel boots and the crashed kernel memory snapshot is saved to the file system. The save path is given by the KDUMP_SAVEDIR option and it defaults to /var/crash. If KDUMP_IMMEDIATE_REBOOT is set to yes , the system automatically reboots the production kernel. Log in and check that the dump has been created under /var/crash.

WARNING: Screen Freezes in X11 Session

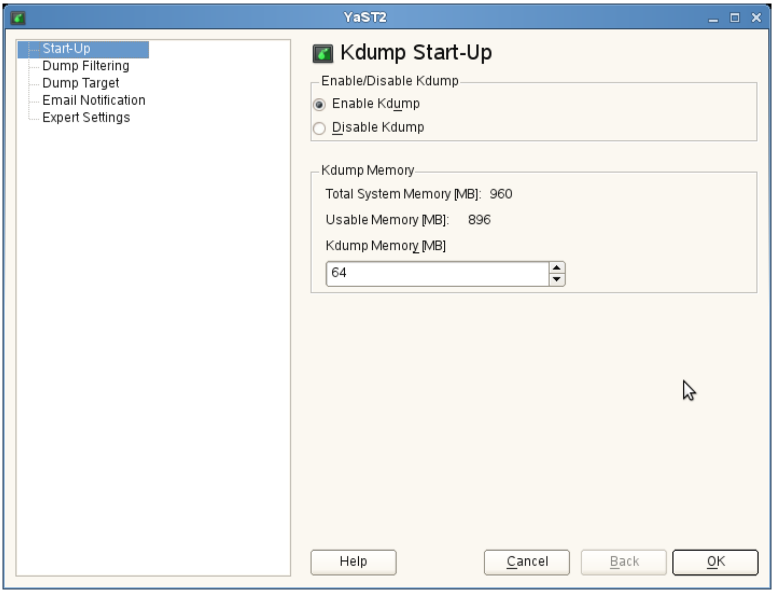
When Kdump takes control and you are logged in an X11 session, the screen will freeze without any notice. Some Kdump activity can be still visible (for example, deformed messages of a booting kernel on the screen).

Do not reset the computer because Kdump always needs some time to complete its task.

16.6.2 YaST Configuration

In order to configure Kdump with YaST, you need to install the yast2-kdump package. Then either start the *Kernel Kdump* module in the *System* category of YaST Control Center, or enter yast2 kdump in the command line as root.

Figure 16.1 *YaST2 Kdump Module - Start-Up Page*



Kdump Start-Up

Enable/Disable Kdump

● Enable Kdump

○ Disable Kdump

Kdump Memory

Total System Memory [MB]: 960

Usable Memory [MB]: 896

Kdump Memory [MB]

64

In the *Start-Up* window, select *Enable Kdump*. The default value for Kdump memory is sufficient on most systems.

Click *Dump Filtering* in the left pane, and check what pages to include in the dump. You do not need to include the following memory content to be able to debug kernel problems:

- Pages filled with zero

- Cache pages

- User data pages

- Free pages

In the *Dump Target* window, select the type of the dump target and the URL where you want to save the dump. If you selected a network protocol, such as FTP or SSH, you need to enter relevant access information as well.

Fill the *Email Notification* window information if you want Kdump to inform you about its events via E-mail and confirm your changes with *OK* after fine tuning Kdump in the *Expert Settings* window. Kdump is now configured.

16.7 Analyzing the Crash Dump

After you obtain the dump, it is time to analyze it. There are several options.

The original tool to analyze the dumps is GDB. You can even use it in the latest environments, although it has several disadvantages and limitations:

- GDB was not specifically designed to debug kernel dumps.
- GDB does not support ELF64 binaries on 32-bit platforms.
- GDB does not understand other formats than ELF dumps (it cannot debug compressed dumps).

That is why the *crash* utility was implemented. It analyzes crash dumps and debugs the running system as well. It provides functionality specific to debugging the Linux kernel and is much more suitable for advanced debugging.

If you want to debug the Linux kernel, you need to install its debugging information package in addition. Check if the package is installed on your system with `zypper se kernel | grep debug`.

IMPORTANT: Repository for Packages with Debugging Information

If you subscribed your system for online updates, you can find “debuginfo” packages in the `*-Debuginfo-Updates` online installation repository relevant for SUSE Linux Enterprise Server 11 SP1. Use YaST to enable the repository.

To open the captured dump in `crash` on the machine that produced the dump, use a command like this:

```
crash /boot/vmlinux-2.6.32.8-0.1-default.gz
/var/crash/2010-04-23-11\:17/vmcore
```

The first parameter represents the kernel image. The second parameter is the dump file captured by Kdump. You can find this file under `/var/crash` by default.

16.7.1 Kernel Binary Formats

The Linux kernel comes in Executable and Linkable Format (ELF). This file is usually called `vmlinux` and is directly generated in the compilation process. Not all boot loaders, especially on x86 (i386 and x86_64) architecture, support ELF binaries. The following solutions exist on different architectures supported by SUSE® Linux Enterprise Server.

x86 (i386 and x86_64)

Mostly for historic reasons, the Linux kernel consists of two parts: the Linux kernel itself (`vmlinux`) and the setup code run by the boot loader.

These two parts are linked together in a file called `bzImage`, which can be found in the kernel source tree. The file is now called `vmlinuz` (note `z` vs. `x`) in the kernel package.

The ELF image is never directly used on x86. Therefore, the main kernel package contains the `vmlinux` file in compressed form called `vmlinux.gz`.

To sum it up, an x86 SUSE kernel package has two kernel files:

- `vmlinuz` which is executed by the boot loader.

- `vmlinux.gz`, the compressed ELF image that is required by *crash* and GDB.

IA64

The `elilo` boot loader, which boots the Linux kernel on the IA64 architecture, supports loading ELF images (even compressed ones) out of the box. The IA64 kernel package contains only one file called `vmlinuz`. It is a compressed ELF image. `vmlinuz` on IA64 is the same as `vmlinux.gz` on x86.

PPC and PPC64

The yaboot boot loader on PPC also supports loading ELF images, but not compressed ones. In the PPC kernel package, there is an ELF Linux kernel file vmlinux. Considering *crash*, this is the easiest architecture.

If you decide to analyze the dump on another machine, you must check both the architecture of the computer and the files necessary for debugging.

You can analyze the dump on another computer only if it runs a Linux system of the same architecture. To check the compatibility, use the command uname -i on both computers and compare the outputs.

If you are going to analyze the dump on another computer, you also need the appropriate files from the kernel and kernel debug packages.

1 Put the kernel dump, the kernel image from /boot, and its associated debugging info file from /usr/lib/debug/boot into a single empty directory.

2 Additionally, copy the kernel modules from /lib/modules/$(uname -r)/kernel/ and the associated debug info files from /usr/lib/debug/lib/modules/$(uname -r)/kernel/ into a subdirectory named modules.

3 In the directory with the dump, the kernel image, its debug info file, and the modules subdirectory, launch the crash utility: crash *vmlinux-version vmcore*.

NOTE: Support for Kernel Images

Compressed kernel images (gzip, not the bzImage file) are supported by SUSE packages of crash since SUSE® Linux Enterprise Server 11. For older versions, you have to extract the vmlinux.gz (x86) or the vmlinuz (IA64) to vmlinux.

Regardless of the computer on which you analyze the dump, the crash utility will produce an output similar to this:

```
tux@mercury:~> crash /boot/vmlinux-2.6.32.8-0.1-default.gz
/var/crash/2010-04-23-11\:17/vmcore

crash 4.0-7.6
```

```
Copyright (C) 2002, 2003, 2004, 2005, 2006, 2007, 2008  Red Hat, Inc.
Copyright (C) 2004, 2005, 2006  IBM Corporation
Copyright (C) 1999-2006  Hewlett-Packard Co
Copyright (C) 2005, 2006  Fujitsu Limited
Copyright (C) 2006, 2007  VA Linux Systems Japan K.K.
Copyright (C) 2005  NEC Corporation
Copyright (C) 1999, 2002, 2007  Silicon Graphics, Inc.
Copyright (C) 1999, 2000, 2001, 2002  Mission Critical Linux, Inc.
This program is free software, covered by the GNU General Public License,
and you are welcome to change it and/or distribute copies of it under
certain conditions.  Enter "help copying" to see the conditions.
This program has absolutely no warranty.  Enter "help warranty" for details.

GNU gdb 6.1
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "x86_64-unknown-linux-gnu"...

      KERNEL: /boot/vmlinux-2.6.32.8-0.1-default.gz
   DEBUGINFO: /usr/lib/debug/boot/vmlinux-2.6.32.8-0.1-default.debug
    DUMPFILE: /var/crash/2009-04-23-11:17/vmcore
        CPUS: 2
        DATE: Thu Apr 23 13:17:01 2010
      UPTIME: 00:10:41
LOAD AVERAGE: 0.01, 0.09, 0.09
       TASKS: 42
    NODENAME: eros
     RELEASE: 2.6.32.8-0.1-default
     VERSION: #1 SMP 2010-03-31 14:50:44 +0200
     MACHINE: x86_64  (2999 Mhz)
      MEMORY: 1 GB
       PANIC: "SysRq : Trigger a crashdump"
         PID: 9446
     COMMAND: "bash"
        TASK: ffff88003a57c3c0  [THREAD_INFO: ffff880037168000]
         CPU: 1
       STATE: TASK_RUNNING (SYSRQ)
crash>
```

The command output prints first useful data: There were 42 tasks running at the moment of the kernel crash. The cause of the crash was a SysRq trigger invoked by the task with PID 9446. It was a Bash process because the `echo` that has been used is an internal command of the Bash shell.

The *crash* utility builds upon GDB and provides many useful additional commands. If you enter `bt` without any parameters, the backtrace of the task running at the moment of the crash is printed:

```
crash> bt
PID: 9446   TASK: ffff88003a57c3c0  CPU: 1   COMMAND: "bash"
 #0 [ffff880037169db0] crash_kexec at ffffffff80268fd6
 #1 [ffff880037169e80] __handle_sysrq at ffffffff803d50ed
 #2 [ffff880037169ec0] write_sysrq_trigger at ffffffff802f6fc5
 #3 [ffff880037169ed0] proc_reg_write at ffffffff802f068b
 #4 [ffff880037169f10] vfs_write at ffffffff802b1aba
 #5 [ffff880037169f40] sys_write at ffffffff802b1c1f
 #6 [ffff880037169f80] system_call_fastpath at ffffffff8020bfbb
    RIP: 00007fa958991f60  RSP: 00007fff61330390  RFLAGS: 00010246
    RAX: 0000000000000001  RBX: ffffffff8020bfbb  RCX: 0000000000000001
    RDX: 0000000000000002  RSI: 00007fa959284000  RDI: 0000000000000001
    RBP: 0000000000000002   R8: 00007fa9592516f0   R9: 00007fa958c209c0
    R10: 00007fa958c209c0  R11: 0000000000000246  R12: 00007fa958c1f780
    R13: 00007fa959284000  R14: 0000000000000002  R15: 00000000595569d0
    ORIG_RAX: 0000000000000001  CS: 0033  SS: 002b
crash>
```

Now it is clear what happened: The internal `echo` command of Bash shell sent a character to `/proc/sysrq-trigger`. After the corresponding handler recognized this character, it invoked the `crash_kexec()` function. This function called `panic()` and Kdump saved a dump.

In addition to the basic GDB commands and the extended version of `bt`, the crash utility defines many other commands related to the structure of the Linux kernel. These commands understand the internal data structures of the Linux kernel and present their contents in a human readable format. For example, you can list the tasks running at the moment of the crash with `ps`. With `sym`, you can list all the kernel symbols with the corresponding addresses, or inquire an individual symbol for its value. With `files`, you can display all the open file descriptors of a process. With `kmem`, you can display details about the kernel memory usage. With `vm`, you can inspect the virtual memory of a process, even at the level of individual page mappings. The list of useful commands is very long and many of these accept a wide range of options.

The commands that we mentioned reflect the functionality of the common Linux commands, such as `ps` and `lsof`. If you would like to find out the exact sequence of events with the debugger, you need to know how to use GDB and to have strong debugging skills. Both of these are out of the scope of this document. In addition, you need to understand the Linux kernel. Several useful reference information sources are given at the end of this document.

16.8 Advanced Kdump Configuration

The configuration for Kdump is stored in `/etc/sysconfig/kdump`. You can also use YaST to configure it. Kdump configuration options are available under *System* > *Kernel Kdump* in YaST Control Center. The following Kdump options may be useful for you:

You can change the directory for the kernel dumps with the `KDUMP_SAVEDIR` option. Keep in mind that the size of kernel dumps can be very large. Kdump will refuse to save the dump if the free disk space, subtracted by the estimated dump size, drops below the value specified by the `KDUMP_FREE_DISK_SIZE` option. Note that KDUMP_SAVEDIR understands URL format *`protocol://specification`*, where *`protocol`* is one of `file`, `ftp`, `sftp`, `nfs` or `cifs`, and `specification` varies for each protocol. For example, to save kernel dump on an FTP server, use the following URL as a template: `ftp://username:password@ftp.example.com:123/var/crash`.

Kernel dumps are usually huge and contain many pages that are not necessary for analysis. With `KDUMP_DUMPLEVEL` option, you can omit such pages. The option understands numeric value between 0 and 31. If you specify *`0`*, the dump size will be largest. If you specify *`31`*, it will produce the smallest dump. For a complete table of possible values, see the manual page of `kdump` (`man 7 kdump`).

Sometimes it is very useful to make the size of the kernel dump smaller. For example, if you want to transfer the dump over the network, or if you need to save some disk space in the dump directory. This can be done with `KDUMP_DUMPFORMAT` set to *`compressed`*. The `crash` utility supports dynamic uncompression of the compressed dumps.

IMPORTANT: Changes to Kdump Configuration File

You always need to execute `rckdump restart` after you make manual changes to `/etc/sysconfig/kdump`. Otherwise these changes will take effect next time you reboot the system.

16.9 For More Information

Since there is no single comprehensive reference to Kexec and Kdump usage, you have to explore several resources to get the information you need. Here are some of them:

- For the Kexec utility usage, see the manual page of Kexec.
- You can find general information about Kexec at `http://www.ibm.com/developerworks/linux/library/l-kexec.html` . Might be slightly outdated.
- For more details on Kdump specific to SUSE Linux, see `http://ftp.suse.com/pub/people/tiwai/kdump-training/kdump-training.pdf` .
- An in-depth description of Kdump internals can be found at `http://lse.sourceforge.net/kdump/documentation/ols2oo5-kdump-paper.pdf` .

For more details on *crash* dump analysis and debugging tools, use the following resources:

- Very useful information about kernel dump debugging with `crash` can be found at `http://en.opensuse.org/Crashdump_Debugging` .
- In addition to the info page of GDB (`info gdb`), you might want to read the printable guides at `http://sourceware.org/gdb/documentation/` .
- A white paper with a comprehensive description of the crash utility usage can be found at `http://people.redhat.com/anderson/crash_whitepaper/` .
- The crash utility also features a comprehensive online help. Just write `help` *`command`* to display the online help for `command`.
- If you have the necessary Perl skills, you can use Alicia to make the debugging easier. This Perl-based front end to the crash utility can be found at `http://alicia.sourceforge.net/` .

- If you prefer Python instead, you may want to install Pykdump. This package helps you control GDB through Python scripts and can be downloaded from `http://sf.net/projects/pykdump`.

- A very comprehensive overview of the Linux kernel internals is given in *Understanding the Linux Kernel* by Daniel P. Bovet and Marco Cesati (ISBN 978-0-596-00565-8).