Oracle Linux 8

Profiling For Performance Analysis With Gprofng

F77421-04
January 2024

Oracle Linux 8 Profiling For Performance Analysis With Gprofng,

F77421-04

Contents

Preface

Oracle Linux 8: Profiling For Performance Analysis With Gprofng describes how to install and use the `gprofng` tool to find performance bottlenecks in executable programs.

Documentation License

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at https://www.oracle.com/corporate/accessibility/.

For information about the accessibility of the Oracle Help Center, see the Oracle Accessibility Conformance Report at https://www.oracle.com/corporate/accessibility/templates/t2-11535.html.

Access to Oracle Support for Accessibility

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit https://www.oracle.com/corporate/accessibility/learning-support.html#support-tab.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

1
About `gprofng` and Profiling

Gprofng is a next generation application profiling tool that can be used to diagnose performance bottlenecks in software applications.

The tool can be used to profile programs compiled with toolchains released by Oracle Linux. These programs can be written using the C, C++, Java and Scala programming languages for the x86_64 and aarch64 processor architectures. The full extent of the data that can be collected differs between CPU models and types.

Oracle developed this tool and contributed it back upstream to the `binutils` project so that it's now part of the GNU binutils tools suite.

For more information, see https://sourceware.org/binutils/wiki/gprofng and the `gprofng(1)` manual page.

2
How to Install `gprofng`

Installing the `binutils-gprofng` package on Oracle Linux 8.

Before installing the `binutils-gprofng` package, enable the `ol8_addons` repository:

```
sudo dnf config-manager --enable ol8_addons

sudo dnf update -y
```

For more information, see Oracle Linux: Managing Software on Oracle Linux.

The `binutils-gprofng` package provides the gprofng profiling tool and its prerequisites on Oracle Linux systems.

1. Install the `binutils-gprofng` package.

   Use the `dnf` command to install the package:

   ```
   sudo dnf install -y binutils-gprofng
   ```

2. Verify that the `binutils-gprofng` package has installed successfully.

   Use the `gprofng` command to verify its presence:

   ```
   gprofng --version
   ```

The `binutils-gprofng` package is installed.

3
Getting Started

Creating an experiment directory, capturing performance data, and inspecting the results.

Install the `binutils-gprofng` package. For more information, see How to Install `gprofng`.

The gprofng profiling tool can be used to assist development teams seeking to optimize their code and improve application performance.

1. Set up the experiment directory.

   You can run the `gprofng` command inside any directory because it generates the necessary directory structure automatically.

   Consider creating a separate directory for the performance experiments. That directory can be stored anywhere, for example in the user home directory or as an unversioned subdirectory within a code project folder.

2. Collect performance data for a program.

   Use the `gprofng collect app` command to start the application and collect performance data while it runs:

   ```
   gprofng collect app /path/to/application -options
   ```

3. Review the performance data that has been captured.

   Use the `gprofng display text` command to analyze the performance data. By default, experiment results are stored in an experiment directory that follows the `test.n.er` naming pattern, where `n` is a numerical identifier for the test and `.er` is a required suffix.

   For example, to review the performance data stored in the `test.1.er` directory, run the following command:

   ```
   gprofng display text -functions test.1.er
   ```

An experiment directory has been created, performance data was captured, and the test directory in which that performance data is stored can be analyzed by using the `gprofng display` command.

4

gprofng Command Reference

This table provides information about the `gprofng` command.

Action	Command	Description
Collect performance data.	`gprofng collect app`	Collects performance data about a running application and stores it in the experiment directory.
Review performance results in a terminal.	`gprofng display text`	Displays performance data from the specified experiment directories in ASCII plain-text format.
Review performance data in a web browser.	`gprofng display html`	Generates a HTML structure from the specified experiment directories.
Review the source and disassembly code.	`gprofng display src`	Displays the source code interleaved with instructions.
Archive an experiment directory.	`gprofng archive`	Copies shared libraries, object files and source code to the experiment directory for later analysis.

When collecting performance data, it's possible to specify the experiment directory by using the `-O` option:

```
gprofng collect app -O experiment-directory-name.er /path/to/application -options
```

Note that `.er` is a required suffix for any experiment directory name.

To analyze performance data collected for each function, use the `-functions` option with the `gprofng display text` command:

```
gprofng display text -functions experiment-directory-name.er

Functions sorted by metric: Exclusive Total CPU Time

Excl. Total   Incl. Total    Name
CPU           CPU
 sec.      %   sec.      %
5.554 100.00  5.554 100.00   <Total>
5.274  94.95  5.274  94.95   mxv_core
0.140   2.52  0.270   4.86   init_data
0.090   1.62  0.110   1.98   erand48_r
0.020   0.36  0.020   0.36   __drand48_iterate
0.020   0.36  0.130   2.34   drand48
0.010   0.18  0.010   0.18   _int_malloc
0.      0.    0.280   5.05   __libc_start_main
0.      0.    0.010   0.18   allocate_data
0.      0.    5.274  94.95   collector_root
0.      0.    5.274  94.95   driver_mxv
```

```
0.      0.    0.280   5.05   main
0.      0.    0.010   0.18   malloc
0.      0.    5.274  94.95   start_thread
```

To limit the number of functions displayed, use the `-limit` option as follows:

```
gprofng display text -limit 5 -functions experiment-directory-name.er

Print limit set to 5
Functions sorted by metric: Exclusive Total CPU Time

Excl. Total   Incl. Total    Name
CPU           CPU
 sec.      %   sec.      %
5.775 100.00  5.775 100.00   <Total>
5.494  95.15  5.494  95.15   mxv_core
0.126   2.18  0.267   4.63   init_data
0.068   1.17  0.104   1.80   erand48_r
0.038   0.66  0.142   2.45   drand48
```

To list all the metrics that have been collected in an experiment directory, use the `-metric_list` option without any other options:

```
gprofng display text -metric_list experiment-directory-name.er

Current metrics: e.%totalcpu:i.%totalcpu:name
Current Sort Metric: Exclusive Total CPU Time ( e.%totalcpu )
Available metrics:
Exclusive Total CPU Time: e.%totalcpu
Inclusive Total CPU Time: i.%totalcpu
                    Size: size
              PC Address: address
                    Name: name
```

After you have established which metrics have been collected in an experiment directory, select the displayed metrics by using the `-metrics` options, separating each with a `:` character:

```
gprofng display text -metrics name:i.%totalcpu:e.%totalcpu -limit 10 -functions
experiment-directory-name.er

Current metrics: name:i.%totalcpu:e.%totalcpu
Current Sort Metric: Exclusive Total CPU Time ( e.%totalcpu )
Print limit set to 10
Functions sorted by metric: Exclusive Total CPU Time

Name                Incl. Total   Excl. Total
                    CPU           CPU
                     sec.      %   sec.      %
 <Total>            5.775 100.00  5.775 100.00
 mxv_core           5.494  95.15  5.494  95.15
 init_data          0.267   4.63  0.126   2.18
 erand48_r          0.104   1.80  0.068   1.17
 drand48            0.142   2.45  0.038   0.66
 __drand48_iterate  0.036   0.62  0.036   0.62
 _int_malloc        0.013   0.22  0.008   0.14
 sysmalloc          0.005   0.09  0.003   0.05
 brk                0.002   0.03  0.002   0.03
 __default_morecore 0.002   0.03  0.      0.
```

To sort the performance data according to a specific metric, for example `name`, add the `-sort` option:

```
gprofng display text -metrics name:i.%totalcpu:e.%totalcpu -sort name -limit 10 -
functions experiment-directory-name.er

Current metrics: name:i.%totalcpu:e.%totalcpu
Current Sort Metric: Exclusive Total CPU Time ( e.%totalcpu )
Current Sort Metric: Name ( name )
Print limit set to 10
Functions sorted by metric: Name

Name                Incl. Total   Excl. Total
                    CPU           CPU
                     sec.      %   sec.      %
 <Total>            5.775 100.00  5.775 100.00
 __default_morecore 0.002   0.03  0.      0.
 __drand48_iterate  0.036   0.62  0.036   0.62
 __libc_start_main  0.280   4.85  0.      0.
 _int_malloc        0.013   0.22  0.008   0.14
 allocate_data      0.013   0.22  0.      0.
 brk                0.002   0.03  0.002   0.03
 collector_root     5.494  95.15  0.      0.
 drand48            0.142   2.45  0.038   0.66
 driver_mxv         5.494  95.15  0.      0.
```

Use the `-disasm` option and specify the function name to review metrics at an instruction or assembly level:

```
gprofng display text -metrics e.totalcpu -disasm function-name experiment-directory-
name.er
```

For more information, see https://sourceware.org/binutils/wiki/gprofng and the `gprofng(1)` manual page.

5
How to Store `gprofng` Options for Reuse

Use Scripts to save `gprofng display` options.

There can be many ways to customize the output of the `gprofng display text` command, and the `-script` option has been provided so that you can supply a text file containing options for use with other experiment directories. That can be a more straightforward way to reproduce performance views.

For example, to get a table with the inclusive and exclusive total CPU times with percentages, limited to the first 10 lines, create a script with the following content:

```
# Command to define the metrics
metrics name:i.%totalcpu:e.%totalcpu
# Limit the views to 10 lines
limit 10
# Display the function overview
functions
```

Note that each option has its own line in the script and no leading dash - character.

That script can then be used with the `gprofng display text` command as follows:

```
gprofng display text -script script-name experiment-directory-name.er

# Command to define the metrics
Current metrics: name:i.%totalcpu:e.%totalcpu
Current Sort Metric: Exclusive Total CPU Time ( e.%totalcpu )
# Limit the views to 10 lines
Print limit set to 10
# Display the function overview
Functions sorted by metric: Exclusive Total CPU Time

Name                Incl. Total   Excl. Total
                    CPU           CPU
                     sec.      %   sec.      %
 <Total>            5.775 100.00  5.775 100.00
 mxv_core           5.494  95.15  5.494  95.15
 init_data          0.267   4.63  0.126   2.18
 erand48_r          0.104   1.80  0.068   1.17
 drand48            0.142   2.45  0.038   0.66
 __drand48_iterate  0.036   0.62  0.036   0.62
 _int_malloc        0.013   0.22  0.008   0.14
 sysmalloc          0.005   0.09  0.003   0.05
 brk                0.002   0.03  0.002   0.03
 __default_morecore 0.002   0.03  0.      0.
```

6

How to Work With `gprofng` and Threaded Applications

Collect performance data for multithreaded applications.

By default, the performance data for a multithreaded application is aggregated over all threads. If the data for individual threads, or a set of threads, is needed, then filters can be used to analyze that information.

To list all the threads for which performance data has been captured, use the `-thread_list` and `-threads` options:

```
gprofng display text -thread_list -threads experiment-directory-name.er

Exp Sel Total
=== === =====
  1 all     3
Objects sorted by metric: Exclusive Total CPU Time

Excl. Total    Name
CPU
 sec.      %
5.775 100.00   <Total>
2.773  48.01   Process 1, Thread 3
2.722  47.13   Process 1, Thread 2
0.280   4.85   Process 1, Thread 1
```

In the example output you can see that the CPU times for three threads were used during that experiment. The thread number can be used to review information for both a specific thread or a group of threads.

To review the performance data for a specific thread, use the `-thread_select` option with a thread number:

```
gprofng display text -limit 5 -thread_select 1 -functions experiment-directory-name.er

Print limit set to 5
Exp Sel Total
=== === =====
  1 1       3
Functions sorted by metric: Exclusive Total CPU Time

Excl. Total   Incl. Total    Name
CPU           CPU
 sec.      %   sec.      %
0.280 100.00  0.280 100.00   <Total>
0.126  44.84  0.267  95.37   init_data
0.068  24.20  0.104  37.01   erand48_r
0.038  13.52  0.142  50.53   drand48
0.036  12.81  0.036  12.81   __drand48_iterate
```

7

Known Issues

The following sections describe known issues in the latest gprofng release. For more information, see https://sourceware.org/binutils/wiki/gprofng#Known_Limitations.

Incorrect Source and Disassembly Percentages

The source and disassembly listings display all percentages as zero. A future release will resolve this problem.

Internal gprofng Function Displayed in Function View

The `collector_root` function might be displayed in function view. That's an internal gprofng function that doesn't consume additional hardware resources, but to avoid any potential confusion, a future release will hide that function by default.