NAME

Devel::DProf - a Perl code profiler

SYNOPSIS

```
perl -d:DProf test.pl
```

DESCRIPTION

The Devel::DProf package is a Perl code profiler. This will collect information on the execution time of a Perl script and of the subs in that script. This information can be used to determine which subroutines are using the most time and which subroutines are being called most often. This information can also be used to create an execution graph of the script, showing subroutine relationships.

To profile a Perl script run the perl interpreter with the **-d** debugging switch. The profiler uses the debugging hooks. So to profile script *test.pl* the following command should be used:

```
perl -d:DProf test.pl
```

When the script terminates (or when the output buffer is filled) the profiler will dump the profile information to a file called *tmon.out*. A tool like *dprofpp* can be used to interpret the information which is in that profile. The following command will print the top 15 subroutines which used the most time:

```
dprofpp
```

To print an execution graph of the subroutines in the script use the following command:

```
dprofpp -T
```

Consult *dprofpp* for other options.

PROFILE FORMAT

The old profile is a text file which looks like this:

```
#fOrTyTwO
$hz=100;
$XS_VERSION='DProf 19970606';
# All values are given in HZ
$rrun_utime=2; $rrun_stime=0; $rrun_rtime=7
PART2
+ 26 28 566822884 DynaLoader::import
- 26 28 566822884 DynaLoader::import
+ 27 28 566822885 main::bar
- 27 28 566822886 main::bar
+ 27 28 566822886 main::baz
+ 27 28 566822887 main::bar
- 27 28 566822888 main::bar
[....]
```

The first line is the magic number. The second line is the hertz value, or clock ticks, of the machine where the profile was collected. The third line is the name and version identifier of the tool which created the profile. The fourth line is a comment. The fifth line contains three variables holding the user time, system time, and realtime of the process while it was being profiled. The sixth line indicates the beginning of the sub entry/exit profile section.

The columns in **PART2** are:

```
sub entry(+)/exit(-) mark
app's user time at sub entry/exit mark, in ticks
app's system time at sub entry/exit mark, in ticks
app's realtime at sub entry/exit mark, in ticks
fully-qualified sub name, when possible
```

With newer perls another format is used, which may look like this:

```
#fOrTyTwO
$hz=10000;
$XS_VERSION='DProf 19971213';
# All values are given in HZ
$over_utime=5917; $over_stime=0; $over_rtime=5917;
$over_tests=10000;
$rrun_utime=1284; $rrun_stime=0; $rrun_rtime=1284;
$total_marks=6;

PART2
@ 406 0 406
& 2 main bar
+ 2
@ 456 0 456
- 2
@ 1 0 1
& 3 main baz
+ 3
@ 141 0 141
+ 2
@ 141 0 141
- 2
@ 1 0 1
& 4 main foo
+ 4
@ 142 0 142
+ & Devel::DProf::write
@ 5 0 5
- & Devel::DProf::write
```

(with high value of $ENV{PERL_DPROF_TICKS}).

New `$over_*` values show the measured overhead of making $over_tests calls to the profiler These values are used by the profiler to subtract the overhead from the runtimes.

Lines starting with @ mark the amount of time passed since the previous @ line. The numbers following the @ are integer tick counts representing user, system, and real time. Divide these numbers by the $hz value in the header to get seconds.

Lines starting with & map subroutine identifiers (an integer) to subroutine packages and names. These should only occur once per subroutine.

Lines starting with + or - mark normal entering and exit of subroutines. The number following is a reference to a subroutine identifier.

Lines starting with * mark where subroutines are entered by `goto &subr`, but note that the return will still be marked as coming from the original sub. The sequence might look like this:

```
+ 5
```

```
 * 6
 - 5
```

Lines starting with `/` is like `-` but mark where subroutines are exited by dying. Example:

```
 + 5
 + 6
 / 6
 / 5
```

Finally you might find `@` time stamp marks surrounded by `+ & Devel::DProf::write` and `- & Devel::DProf::write` lines. These 3 lines are outputted when printing of the mark above actually consumed measurable time.

AUTOLOAD

When Devel::DProf finds a call to an `&AUTOLOAD` subroutine it looks at the `$AUTOLOAD` variable to find the real name of the sub being called. See *"Autoloading" in perlsub*.

ENVIRONMENT

`PERL_DPROF_BUFFER` sets size of output buffer in words. Defaults to 2**14.

`PERL_DPROF_TICKS` sets number of ticks per second on some systems where a replacement for times() is used. Defaults to the value of `HZ` macro.

`PERL_DPROF_OUT_FILE_NAME` sets the name of the output file. If not set, defaults to tmon.out.

BUGS

Builtin functions cannot be measured by Devel::DProf.

With a newer Perl DProf relies on the fact that the numeric slot of $DB::sub contains an address of a subroutine. Excessive manipulation of this variable may overwrite this slot, as in

```
$DB::sub = 'current_sub';
...
$addr = $DB::sub + 0;
```

will set this numeric slot to numeric value of the string `current_sub`, i.e., to `0`. This will cause a segfault on the exit from this subroutine. Note that the first assignment above does not change the numeric slot (it will *mark* it as invalid, but will not write over it).

Another problem is that if a subroutine exits using goto(LABEL), last(LABEL) or next(LABEL) then perl may crash or Devel::DProf will die with the error:

```
panic: Devel::DProf inconsistent subroutine return
```

For example, this code will break under Devel::DProf:

```
sub foo {
  last FOO;
}
FOO: {
  foo();
}
```

A pattern like this is used by Test::More's skip() function, for example. See *perldiag* for more details.

Mail bug reports and feature requests to the perl5-porters mailing list at <*perl5-porters@perl.org*>.

SEE ALSO

perl, *dprofpp*, times(2)