Prof2 – A Profiler

Frequently there is a problem with a program running slow and you do not know why. You need a thumbnail analysis of where the program is spending most of its time without all of the overwhelming details. This is when a Profiler comes in handy.

There are many different profilers available – this particular one does not do anything special and is fairly ordinary in that it just characterizes frequency of execution of various regions in the program. However, it has some pretty good strengths as listed next.

1) it takes very little or no preconditioning, i.e. setup
2) the effect it has on the running program is minimal
3) it runs almost at full speed, which is really nice
4) it is not particularly thread aware
5) reports where it is executing at intervals of 100 times a second

Setup

This profiler works on x86_64 systems and you must be root to compile and run it. Also, you must install your operating system's "iberty" – for example, which would be binutils-devel for Fedora or libiberty-dev for Ubuntu 18.

Go to the top level Cinelerra directory.
Key in: cd prof2
Key in: make clean all install
Later, if you want to remove this from the system, key in: make uninstall

How to use

The program you are profiling should be compiled debuggable with stack frame and symbols enabled.

To see help, key in: prof -h

```
usage: -h [-o path] [-d] [-e] [-p libpath] [-012] [-u #] cmd args...
  -o  profile output pathname, -=stdout, --=stderr
  -d  debug output enabled
  -e  child debug output enabled
  -p  specify path for libprofile.so
  -0  usr+sys cpu timer intervals (sigprof)
  -1  usr only cpu timer intervals (sigvtalrm)
  -2  real time timer intervals (sigalrm)
  -u  profile timer interval in usecs
```

To execute the profiler, key in: prof -o /tmp/prof_list.txt ./ci

where /tmp/prof_list.txt is the output file and in this case "cin" is the Cinelerra binary file.

The pid of this command will be displayed on the startup window. This comes in handy in the use case where there is a lot of initial load and possible configuration setup inside of Cinelerra and you want to profile plugins and not necessarily all of the setup steps. Then you can use the following command in another window to continue running Cinelerra and obtain the more useful information:

kill -USR1 pid

Running this command refreshes the memory maps used for profiling. When you are profiling a plugin, you want to run this AFTER the plugin loads.

Results

There are 3 sections in the resulting output file of this stochastic quick analysis.

1) The first section is a histogram of the timer intervals of that sample set. Each function occupies a region of addresses. One hundred times a second, the profiler samples the program address in the region of execution.

2) In the second section, there is another histogram of the cumulative frequency of all things in the call stack and this should point out who is the culprit. For example, a codec calls a bunch of subroutines, the cost of the subroutines is accumulated to the codec parent caller. This makes the actual cpu user much more apparent.

3) The last section is for the library spaces. Each library occupies a region and the profiler adds up the time spent in each of the libraries by frequency.

On the very bottom is a 1 line summary which shows you if there is a bad guy and who it is.

Sample output

```
---- profile start ----
 1020 ticks 43 modules 81412 syms
   0.010s   0.1% Autos::copy(long, long, FileXML*, int, int) /mnt0/build5/cinelerra-5.1/bin/cin
   0.010s   0.1% BinFolders::copy_from(BinFolders*) /mnt0/build5/cinelerra-5.1/bin/cin
   0.010s   0.1% cstrdup(char const*)     /mnt0/build5/cinelerra-5.1/bin/cin
   0.010s   0.1% XMLBuffer::encode_data(char*, char const*, int) /mnt0/build5/cinelerra-5.1/bin/cin
   0.010s   0.1% XMLBuffer::encoded_length(char const*, int) /mnt0/build5/cinelerra-5.1/bin/cin
   0.010s   0.1% PluginClient::send_configure_change() /mnt0/build5/cinelerra-5.1/bin/cin
   0.010s   0.1% UndoVersion::scan_lines(UndoHashTable*, char*, char*) /mnt0/.../cin
   0.010s   0.1% UndoStackItem::set_data(char*) /mnt0/build5/cinelerra-5.1/bin/cin
   0.010s   0.1% UndoStack::load(_IO_FILE*) /mnt0/build5/cinelerra-5.1/bin/cin
   0.010s   0.1% BC_Bitmap::cur_bfr()     /mnt0/build5/cinelerra-5.1/bin/cin
   0.010s   0.1% YUV::init_tables(int, int*, int*, int*, int*, int*, int*, int*, int*, int*, int*, int*,
int*, int*) /mnt0/build5/cinelerra-5.1/bin/cin
…
…
…
---- profile calls ----
   0.010s   0.1% AutoConf::save_xml(FileXML*)   1.0 /mnt0/build5/cinelerra-5.1/bin/cin
   0.010s   0.1% Automation::copy(long, long, FileXML*, int, int)   1.0 /mnt0/.../cin
   0.010s   0.1% AWindow::run()            1.0 /mnt0/build5/cinelerra-5.1/bin/cin
   0.010s   0.1% Canvas::stop_single()      1.0 /mnt0/build5/cinelerra-5.1/bin/cin
   0.010s   0.1% ColorPicker::new_gui()     1.0 /mnt0/build5/cinelerra-5.1/bin/cin
   0.010s   0.1% ColorWindow::create_objects()   1.0 /mnt0/build5/cinelerra-5.1/bin/cin
```

```
   0.010s   0.1% PaletteWheel::draw(float, float)   1.0 /mnt0/build5/cinelerra-5.1/bin/cin
   0.010s   0.1% PaletteHex::update()       1.0 /mnt0/build5/cinelerra-5.1/bin/cin
   0.010s   0.1% CWindowGUI::draw_status(int)   1.0 /mnt0/build5/cinelerra-5.1/bin/cin
   0.010s   0.1% CWindowCanvas::status_event()   1.0 /mnt0/build5/cinelerra-5.1/bin/cin
…
…
…
   0.990s   9.7% BC_Xfer::xfer_slices(int)   1.0 /mnt0/build5/cinelerra-5.1/bin/cin
   1.880s  18.4% DirectUnit::process_package(LoadPackage*)   1.0 /mnt0/build5/cinelerra-5.1/bin/cin
   1.880s  18.4% DirectUnit::rgba8888()     1.0 /mnt0/build5/cinelerra-5.1/bin/cin
   3.910s  38.3% __init_array_end           1.1 /mnt0/build5/cinelerra-5.1/bin/cin
   5.450s  53.4% LoadClient::run()          1.0 /mnt0/build5/cinelerra-5.1/bin/cin
   7.890s  77.4% Thread::entrypoint(void*)   1.0 /mnt0/build5/cinelerra-5.1/bin/cin
   7.890s  77.4% start_thread               1.0 /lib64/libpthread-2.28.so
----
   0.010s   0.1%/  0.0% /lib64/libm-2.28.so
   0.010s   0.1%/  0.0% /lib64/libexpat.so.1.6.8
   0.020s   0.2%/  0.1% /lib64/libXext.so.6.4.0
   0.020s   0.2%/  0.1% /lib64/libXft.so.2.3.2
   0.020s   0.2%/  0.1% /lib64/libxcb.so.1.1.0
   0.040s   0.4%/  0.2% /lib64/ld-2.28.so
   0.050s   0.5%/  0.2% /lib64/libpng16.so.16.34.0
   0.130s   1.3%/  0.6% /lib64/libX11.so.6.3.0
   0.180s   1.8%/  0.8% /lib64/libz.so.1.2.11
   0.200s   2.0%/  0.9% /lib64/libfontconfig.so.1.12.0
   0.380s   3.7%/  1.8% /lib64/libpthread-2.28.so
   1.660s  16.3%/  7.7% /lib64/libc-2.28.so
   7.480s  73.3%/ 34.7% /mnt0/build5/cinelerra-5.1/bin/cin
  10.200t 0.001u+0.000s 21.566r  47.3%
---- profile end ----
```

The summary line above in Bold represents the User time, System time, Real time and the percentage is how much Timer time elapsed over Real time so in this case the measurement covers 47.3% of time.

So why use a Profiler? Because it is the “ls” for executable functions!!