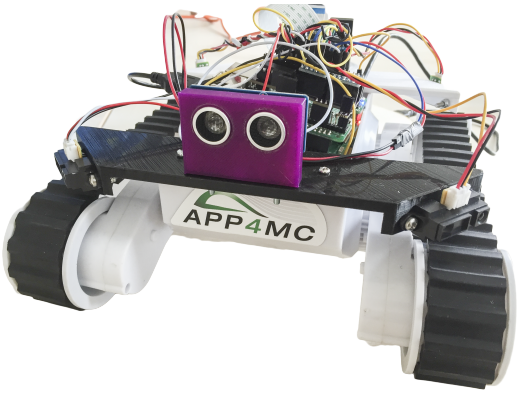
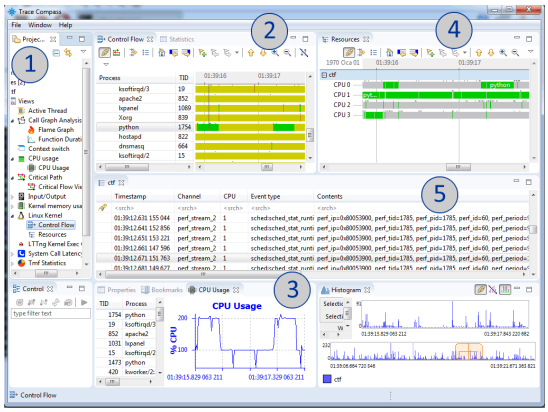
Rover Documentation

Tracing with **Perf**, Conversion to **CTF**, and analysis with **TraceCompass**

Version	*June 16, 2017*	
Implementation	*Mustafa Özcelikörs*	*mozcelikors@gmail.com*
Supervision & revision	*Robert Höttger*	*robert.hoettger@fh-dortmund.de*

University of Applied Sciences and Arts Dortmund
IDiAL Institute, Project AMALTHEA4public
BMBF Fund.Nb. 01—S14029K





1 Scope

This documentation describes (a) the generation of **`Perf`** traces [3] on a Linux system, (b) an approach to convert `perf` traces to the **CTF** (Common Tracing Format) format [2], and (c) the **CTF** import to an analysis tool e.g. TraceCompass [1].

2 Installing BabelTrace

The perf module is already included with the Linux kernel but requires a LibBabelTrace specific build for the conversion support to CTF. Hence, BabelTrace has to be installed first. Therefore, the repository list of the Raspberry Pi has to be up to date:

```
1  sudo apt-get update
```

The following command installs the LibBabelTrace module:

```
1  sudo apt-get install libbabeltrace-ctf-dev libbabeltrace-ctf1 libbabeltrace1
     libbabeltrace-dev python3-babeltrace
```

In addition, the module's dependencies must be installed as well via:

```
1  sudo apt-get install dh-autoreconf bison libdw-dev libelf-dev flex uuid-dev
    libpopt-dev
```

Afterwards, the following commands should be used in order to clone, configure, build, and install LibBabel-Trace:

```
cd /home/pi
sudo git clone git://git.efficios.com/babeltrace.git
cd babeltrace
sudo ./bootstrap
sudo ./configure --prefix=/opt/libbabeltrace LDFLAGS=-L/usr/local/lib
sudo make -j4 prefix=/opt/libbabeltrace
sudo make install prefix=/opt/libbabeltrace
```

3 Building Perf with BabelTrace support

After LibBabelTrace is installed, perf should be rebuilt from the Linux kernel with the babeltrace support. By using the following command, the dependencies and features regarding the new perf build can be installed:

```
sudo apt-get install libnewt-dev binutils-arm-none-eabi libcrypto++-dev
 binutils-multiarch-dev libunwind-dev systemtap-sdt-dev libssl-dev libperl-
 dev libiberty-dev
```

Having all the dependencies, one should clone, build, and install the following linux repository with a LibBabelTrace adapted perf that supports data conversion via:

```
cd /home/pi
sudo git clone git://git.kernel.org/pub/scm/linux/kernel/git/acme/linux.git
cd linux/tools/perf/
sudo git checkout perf/core
sudo LIBBABELTRACE=1 LIBBABELTRACE_DIR=/opt/libbabeltrace/ make
sudo LIBBABELTRACE=1 LIBBABELTRACE_DIR=/opt/libbabeltrace/ make install
```

This will create a perf executable located at the `/home/pi/linux/tools/perf/` directory.

4 Tracing processes and threads

4.1 Option 1: Tracing Processes and Threads Manually

In order to record a system trace for e.g. 15 seconds, the following command can be used while the applications are running:

```
sudo ./home/pi/linux/tools/perf/perf sched record -- sleep 15
```

This command creates a perf trace file namely `perf.data`. Since the Eclipse TraceCompass tool does not support the perf format directly, the perf trace can be converted to the CTF format via:

```
sudo LD_LIBRARY_PATH=/opt/libbabeltrace/lib ./home/pi/linux/tools/perf/perf
 data convert --to-ctf=./ctf
```

The command above should be called within the path where `perf.data` is located. In order to avoid having to do this, the input argument can also be used to specify the location of the perf trace e.g.:

```
sudo LD_LIBRARY_PATH=/opt/libbabeltrace/lib ./home/pi/linux/tools/perf/perf
 data convert -i=/path/to/perf.data --to-ctf=./ctf
```

In order to use the trace in Eclipse TraceCompass on a Windows operating system, the CTF trace can be archived by using the following command:

```
sudo tar -czvf trace.tar.gz ctf/
```

4.2 Option 2: Tracing processes and threads using a script

In order to trace and manually convert the trace, the following shell script `TraceLinuxProcesses.sh` (located at src/tracing) can be used. The script should have the following content:

```
#!/bin/bash
args=("$@")
trace_name=${args[0]}
seconds=${args[1]}
perf_directory=${args[2]}

if [ "$#" -ne 3 ]; then
  echo "Entered arguments seem to be incorrect"
  echo "Right usage: sudo TraceLinuxProcesses.sh <trace_name> <period> <path_to_perf>"
  echo "e.g. sudo TraceLinuxProcesses.sh APP4MC_Trace 15 /home/pi/linux/tools/perf"
else
  echo "### Creating directory.."
  sudo mkdir out_$trace_name/
  echo "### Writing out process names.."
  ps -aux >> out_$trace_name/Processes_List.txt
  echo "### Tracing with perf for $seconds seconds.."
  sudo $perf_directory/./perf sched record -o out_$trace_name/perf.data -- sleep $seconds
  echo "### Converting to data to CTF (Common Tracing Format).."
  sudo LD_LIBRARY_PATH=/opt/libbabeltrace/lib $perf_directory/./perf data convert -i out_$trace_name/perf.data --to-ctf=./ctf
  sudo tar -czvf out_$trace_name/trace.tar.gz ctf/
  sudo rm -rf ctf/

  echo "### Process IDs are written to out_$trace_name/Processes_List.txt"
  echo "### Trace in Perf format is written to out_$trace_name/perf.data"
  echo "### Trace in CTF format is written to out_$trace_name/trace.tar.gz"
  echo "### Exiting.."
fi
```

To run the scrip with root privileges, execute:

```
sudo chmod 777 TraceLinuxProcesses.sh
```

The provided script takes three arguments: trace name, amount of time to trace, installed perf directory. An example way to call this script is shown below which traces the system for 15 seconds and creates a folder out_rover_trace1 using the perf object located at `/home/pi/linux/tools/perf/` directory:

```
sudo TraceLinuxProcesses.sh out_rover_trace1 15 /home/pi/linux/tools/perf
```

The script produces an output directory which contains perf format data, ctf format and the process id list to interpret the output.

5 Visualization and Interpretation of the Trace

By using Eclipse TraceCompass, which can be downloaded from *http://www.tracecompass.org*, traces in CTF format can be interpreted. By choosing the trace file by selecting *File* → *Import* menu items, the trace can be imported into Eclipse TraceCompass seen in Figure 1.

TraceCompass enables users to analyze a system regarding call graphs, threads, context switches, cpu usage, critical path, I/O, control flow, and resources which can be seen as (1) in the Figure 1. The control flow window (2) shows each process state with respect to time including the transitions along all the processes. System-wide CPU usage and individual processes' CPU usages are shown in the CPU Usage window which is shown as (3) in the figure. Therefore, if the system has 4 cores, the CPU usage of up to 400 percent could be observed. The resources window (shown as (4)) depicts how processes are distributed amongst the existing cores with respect to time. Therefore, the load balancing could be roughly observed from this view by simply looking at each of the cores. Moreover, using the resources window, one can measure and estimate the timing properties of the scheduling done by the kernel. Finally, the trace event list (shown as (5)) can be used to see the exact events that occurred in a specific time frame selected in another view.

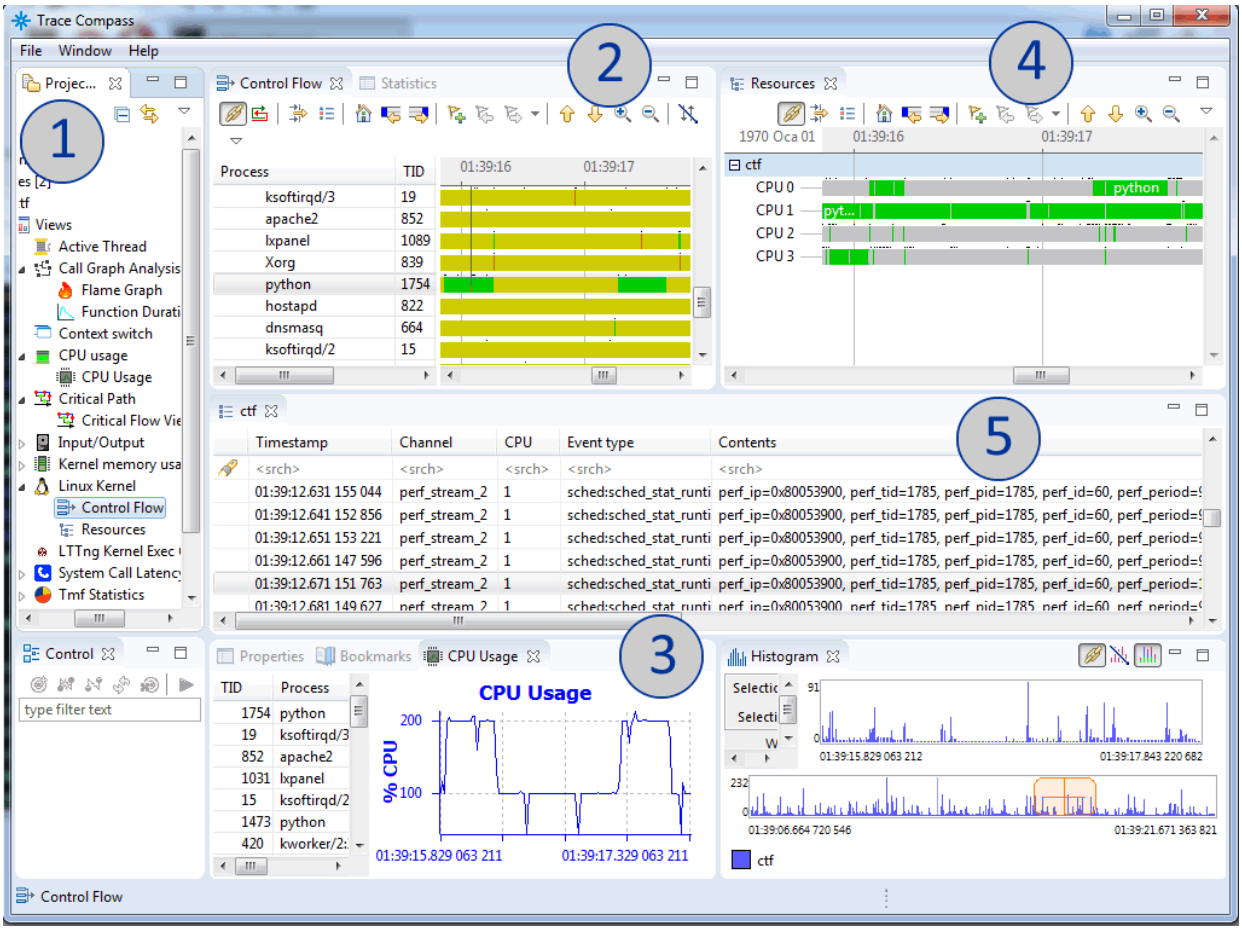


Figure 1: Eclipse TraceCompass

6 Making Thread Names Visible to TraceCompass and Monitoring Core Utilization of Threads

In order to monitor core utilization of each thread using Linux shell, first the thread names of each thread should be registered in user application. That is, using the `pthread_setname_np()` function after each thread is created, which is already implemented in the Rover project. Once this function is used in the implementation, the Linux command name of the thread will be overwritten. This helps Linux tools and TraceCompass recognize the thread name and be able to show the thread name when tracing.

Implemented scripts help to monitor threads of a process easily. The scripts are located in the rover project repository and are briefly explained below.

The following script can be used in order to monitor the core utilization of every registered thread (`MonitorThreads.sh`):

```
1  args=("$@")
2  process_name=${args[0]}
3
4  if [ "$#" -ne 1 ]; then
5    echo "Entered arguments seem to be incorrect"
6    echo "Right usage: MonitorThreads.sh <process_name>"
7    echo "e.g. MonitorThreads.sh <process_name>"
8  else
9    pid=$(pgrep -f $process_name -o)
10   top H -p $pid
11 fi
```

Similarly, by using the `ListThreads.sh` given below, one can list thread names and thread ID's of all the threads of a process given its process name.

```
1  args=("$@")
2  process_name=${args[0]}
3  if [ "$#" -ne 1 ]; then
4    echo "Entered arguments seem to be incorrect"
5    echo "Right usage: ListThreads.sh <process_name>"
6    echo "e.g. ListThreads.sh <process_name>"
7  else
8    pid=$(pgrep -f $process_name -o)
9    ps H -p $pid -o 'pid tid cmd comm'
10 fi
```

The Figure 2 depicts the output from the `MonitorThreads.sh`.

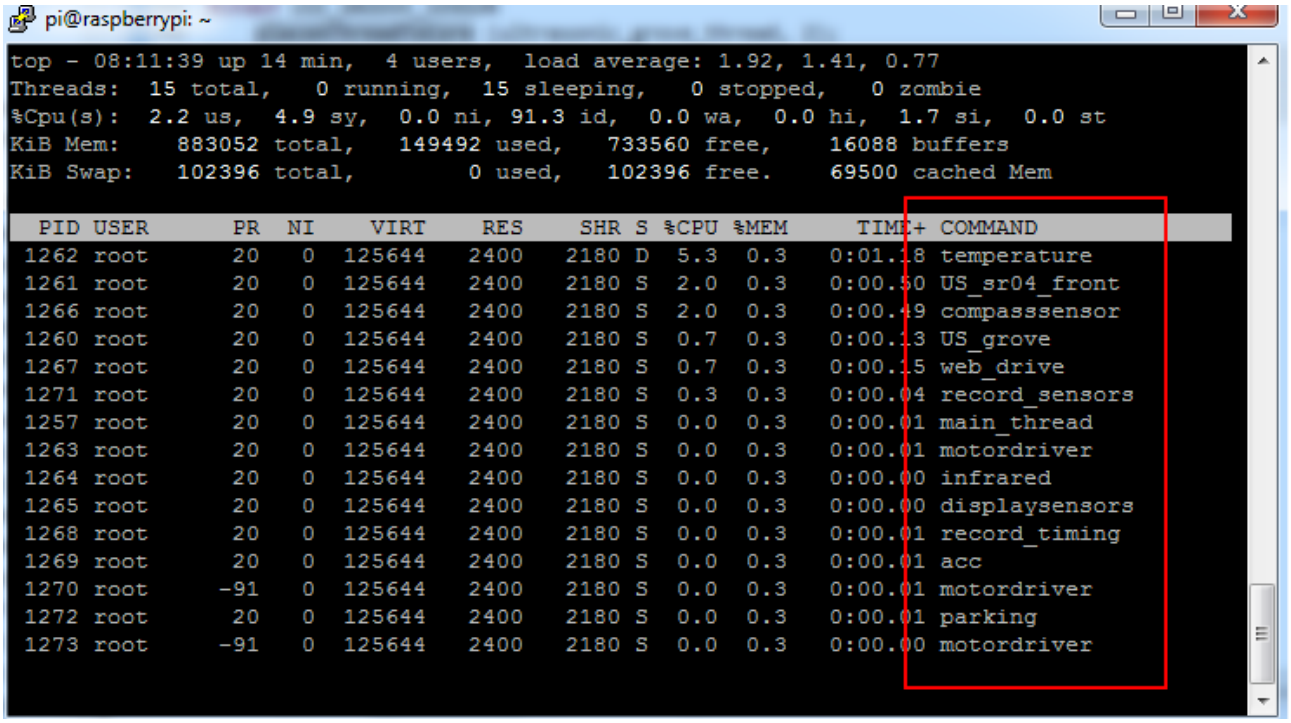


Figure 2: MonitorThreads.sh output

References

[1] TraceCompass consortium. Eclipse Trace Compass. `https://projects.eclipse.org/projects/tools.tracecompass`. Accessed: 2017-06-01.

[2] Various contributors. Babeltrace - an open source trace format converter. `http://diamon.org/babeltrace/`. Accessed: 2017-06-01.

[3] Various contributors. perf: Linux profiling with performance counters. `https://perf.wiki.kernel.org/index.php/Main_Page`. Accessed: 2017-06-01.