```
/************************************************************************************
** Notes on Linux perf tool
**
** Intended audience: Those who would like to learn more about
** Linux perf performance analysis and profiling tool.
**
** Used: CPE 631 Advanced Computer Systems and Architectures
**          CPE 619 Modeling and Analysis of Computer and Communication Systems
**
** ver 0.1, Spring 2012
** ver 0.2, Spring 2018
** ver 0.3, Spring 2021
**
** @Aleksandar Milenkovic, milenkovic@computer.org
************************************************************************************/
```

Perf Tool: Performance Analysis Tool for Linux

1. Introduction

Perf is a profiler tool for Linux 2.6+ based systems that abstracts away CPU hardware differences in Linux performance measurements and presents a simple command line interface. It covers hardware level (CPU/PMU, Performance Monitoring Unit) features and software features (software counters, tracepoints) as well.

To learn more about perf type in man perf.

```
<<~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-bash-4.2$ man perf

-bash-4.2$ $ man perf-stat

-bash-4.2$ $ man perf-top

...
# Note before the start, do the following to enable devtoolset-6
-bash-4.2$ scl enable devtoolset-6 bash
# You can verify that you are using the right environment
-bash-4.2$ bash-4.2$ which gcc
/opt/rh/devtoolset-6/root/usr/bin/gcc
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~>>
```

2. Commands

The perf tool offers a rich set of commands to collect and analyze performance and trace data.
The command line usage is reminiscent of git in that there is a generic tool, perf, which implements a set of commands: stat, record, report, [...].

* To see the list of all options, please type in perf.

```
<<~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-bash-4.2$ $ perf

 usage: perf [--version] [--help] [OPTIONS] COMMAND [ARGS]

 The most commonly used perf commands are:
   annotate        Read perf.data (created by perf record) and display annotated code
   archive         Create archive with object files with build-ids found in perf.data
                    file
   bench           General framework for benchmark suites
   buildid-cache   Manage build-id cache.
   buildid-list    List the buildids in a perf.data file
   data            Data file related processing
   diff            Read perf.data files and display the differential profile
   evlist          List the event names in a perf.data file
   inject          Filter to augment the events stream with additional information
   kmem            Tool to trace/measure kernel memory properties
   kvm             Tool to trace/measure kvm guest os
   list            List all symbolic event types
   lock            Analyze lock events
   mem             Profile memory accesses
   record          Run a command and record its profile into perf.data
   report          Read perf.data (created by perf record) and display the profile
   sched           Tool to trace/measure scheduler properties (latencies)
   script          Read perf.data (created by perf record) and display trace output
   stat            Run a command and gather performance counter statistics
   test            Runs sanity tests.
   timechart       Tool to visualize total system behavior during a workload
   top             System profiling tool.
   probe           Define new dynamic tracepoints
   trace           strace inspired tool

 See 'perf help COMMAND' for more information on a specific command.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~>>
```

* Certain commands require special support in the kernel and may not be available. To obtain the list of options for each command, simply type the command name followed by -h, e.g.:

```
<<~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-bash-4.2$ $ perf stat -h

Usage: perf stat [<options>] [<command>]

    -a, --all-cpus        system-wide collection from all CPUs
    -A, --no-aggr         disable CPU count aggregation
    -B, --big-num         print large numbers with thousands' separators
    -C, --cpu <cpu>       list of cpus to monitor in system-wide
    -c, --scale           scale/normalize counters
    -D, --delay <n>       ms to wait before starting measurement after program start
    -d, --detailed        detailed run - start a lot of events
    -e, --event <event>   event selector. use 'perf list' to list available events
    -G, --cgroup <name>   monitor event in cgroup name only
    -g, --group           put the counters into a counter group
    -I, --interval-print <n>
                          print counts at regular interval in ms (overhead is possible
for values <= 100m
    -i, --no-inherit      child tasks do not inherit counters
    -M, --metrics <metric/metric group list>
                          monitor specified metrics or metric groups (separated by ,)
```

```
    -n, --null            null run - dont start any counters
    -o, --output <file>   output file name
    -p, --pid <pid>       stat events on existing process id
    -r, --repeat <n>      repeat command and print average + stddev (max: 100, 
forever: 0)
    -S, --sync            call sync() before starting a run
    -t, --tid <tid>       stat events on existing thread id
    -T, --transaction     hardware transaction statistics
    -v, --verbose         be more verbose (show counter open errors, etc)
    -x, --field-separator <separator>
                          print counts with custom separator
        --append          append to the output file
        --filter <filter>
                          event filter
        --interval-clear  clear screen in between new interval
        --interval-count <n>
                          print counts for fixed number of times
        --log-fd <n>      log output to fd, instead of stderr
        --metric-only     Only print computed metrics. No raw values
        --no-merge        Do not merge identical named events
        --per-core        aggregate counts per physical processor core
        --per-die         aggregate counts per processor die
        --per-socket      aggregate counts per processor socket
        --per-thread      aggregate counts per thread
        --post <command>  command to run after to the measured command
        --pre <command>   command to run prior to the measured command
        --smi-cost        measure SMI cost
        --table           display details about each run (only with -r option)
        --timeout <n>     stop workload and print counts after a timeout period in ms 
(>= 10ms)
        --topdown         measure topdown level 1 statistics

(END)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~>>
```

3. Events

The perf tool supports a list of measurable events. The tool and underlying kernel interface can measure events coming from different sources. For instance, some events are pure kernel counters; in this case they are called software events. Examples include: context-switches, minor-fault.
Another source of events is the processor itself and its Performance Monitoring Unit (PMU). It provides a list of events to measure micro-architectural events such as the number of cycles, instructions retired, L1 cache misses and so on. Those events are called PMU hardware events or hardware events for short. They vary with each processor type and model. The perf_events interface also provides a small set of common hardware events monikers.
On each processor, those events get mapped onto actual events provided by the CPU, if they exist, otherwise the event cannot be used. Somewhat confusingly, these are also called hardware events and hardware cache events.
Finally, there are also tracepoint events which are implemented by the kernel ftrace infrastructure. Those are only available with the 2.6.3x and newer kernels.

* To obtain a list of supported events type in perf list.

```
<<~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-bash-4.2$ $ perf list
```

```
List of pre-defined events (to be used in -e):

  branch-instructions OR branches                    [Hardware event]
  branch-misses                                      [Hardware event]
  bus-cycles                                         [Hardware event]
  cache-misses                                       [Hardware event]
  cache-references                                   [Hardware event]
  cpu-cycles OR cycles                               [Hardware event]
  instructions                                       [Hardware event]
  ref-cycles                                         [Hardware event]

  alignment-faults                                   [Software event]
  bpf-output                                         [Software event]
  context-switches OR cs                             [Software event]
  cpu-clock                                          [Software event]
  cpu-migrations OR migrations                       [Software event]
  dummy                                              [Software event]
  emulation-faults                                   [Software event]
  major-faults                                       [Software event]
  minor-faults                                       [Software event]
  page-faults OR faults                              [Software event]
  task-clock                                         [Software event]

  L1-dcache-load-misses                              [Hardware cache event]
  L1-dcache-loads                                    [Hardware cache event]
  L1-dcache-stores                                   [Hardware cache event]
  L1-icache-load-misses                              [Hardware cache event]
  LLC-load-misses                                    [Hardware cache event]
  LLC-loads                                          [Hardware cache event]
  LLC-store-misses                                   [Hardware cache event]
  LLC-stores                                         [Hardware cache event]
  branch-load-misses                                 [Hardware cache event]
  branch-loads                                       [Hardware cache event]
  dTLB-load-misses                                   [Hardware cache event]
  dTLB-loads                                         [Hardware cache event]
  dTLB-store-misses                                  [Hardware cache event]
  dTLB-stores                                        [Hardware cache event]
  iTLB-load-misses                                   [Hardware cache event]
  iTLB-loads                                         [Hardware cache event]
  node-load-misses                                   [Hardware cache event]
  node-loads                                         [Hardware cache event]
  node-store-misses                                  [Hardware cache event]
  node-stores                                        [Hardware cache event]

  branch-instructions OR cpu/branch-instructions/    [Kernel PMU event]
  branch-misses OR cpu/branch-misses/                [Kernel PMU event]
  bus-cycles OR cpu/bus-cycles/                      [Kernel PMU event]
  cache-misses OR cpu/cache-misses/                  [Kernel PMU event]
  cache-references OR cpu/cache-references/          [Kernel PMU event]
  cpu-cycles OR cpu/cpu-cycles/                      [Kernel PMU event]
  cycles-ct OR cpu/cycles-ct/                        [Kernel PMU event]
  cycles-t OR cpu/cycles-t/                          [Kernel PMU event]
  el-abort OR cpu/el-abort/                          [Kernel PMU event]
  el-capacity OR cpu/el-capacity/                    [Kernel PMU event]
  el-commit OR cpu/el-commit/                        [Kernel PMU event]
  el-conflict OR cpu/el-conflict/                    [Kernel PMU event]
  el-start OR cpu/el-start/                          [Kernel PMU event]
  instructions OR cpu/instructions/                  [Kernel PMU event]
  intel_pt//                                         [Kernel PMU event]
  mem-loads OR cpu/mem-loads/                        [Kernel PMU event]
  mem-stores OR cpu/mem-stores/                      [Kernel PMU event]
  msr/aperf/                                         [Kernel PMU event]
  msr/mperf/                                         [Kernel PMU event]
  msr/pperf/                                                [Kernel PMU event]
  msr/smi/                                                  [Kernel PMU event]
  msr/tsc/                                                  [Kernel PMU event]
  power/energy-cores/                                       [Kernel PMU event]
  power/energy-pkg/                                         [Kernel PMU event]
  power/energy-ram/                                         [Kernel PMU event]
  ref-cycles OR cpu/ref-cycles/                             [Kernel PMU event]
  tx-abort OR cpu/tx-abort/                                 [Kernel PMU event]
  tx-capacity OR cpu/tx-capacity/                           [Kernel PMU event]
  tx-commit OR cpu/tx-commit/                               [Kernel PMU event]
  tx-conflict OR cpu/tx-conflict/                           [Kernel PMU event]
  tx-start OR cpu/tx-start/                                 [Kernel PMU event]
  uncore_iio_free_running_0/bw_in_port0/                    [Kernel PMU event]
  uncore_iio_free_running_0/bw_in_port1/                    [Kernel PMU event]
  uncore_iio_free_running_0/bw_in_port2/                    [Kernel PMU event]
  uncore_iio_free_running_0/bw_in_port3/                    [Kernel PMU event]
  uncore_iio_free_running_0/bw_out_port0/                   [Kernel PMU event]
  uncore_iio_free_running_0/bw_out_port1/                   [Kernel PMU event]
  uncore_iio_free_running_0/bw_out_port2/                   [Kernel PMU event]
  uncore_iio_free_running_0/bw_out_port3/                   [Kernel PMU event]
  uncore_iio_free_running_0/ioclk/                          [Kernel PMU event]
  uncore_iio_free_running_0/util_in_port1/                  [Kernel PMU event]
  uncore_iio_free_running_0/util_in_port2/                  [Kernel PMU event]
  uncore_iio_free_running_0/util_in_port3/                  [Kernel PMU event]
  uncore_iio_free_running_0/util_out_port0/                 [Kernel PMU event]
  uncore_iio_free_running_0/util_out_port1/                 [Kernel PMU event]
  uncore_iio_free_running_0/util_out_port2/                 [Kernel PMU event]
  uncore_iio_free_running_0/util_out_port3/                 [Kernel PMU event]
  . . . (list goes on and on, review it carefully)

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~>>
```

4. Counting with perf stat

For any of the supported events, perf can keep a running count during process execution. In counting modes, the occurrences of events are simply aggregated and presented on standard output at the end of an application run.
To generate these statistics, use the stat command of perf. For instance:

* Perform perf stat on a program arrsum from the time measurement tutorial.

```
<<~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# do the following if you have skipped the time measurement tutorial
-bash-4.2$ rm -rf exetimemeasure/
-bash-4.2$ cp -r /apps/arch/arch.tut/exetimemeasure/ .
-bash-4.2$ cd exetimemeasure/
-bash-4.2$ ls
arrsum.c  arrsum_clock.c
-bash-4.2$ gcc -O2 arrsum.c -o arrsum.exe
-bash-4.2$ ls
arrsum.c  arrsum_clock.c  arrsum.exe

# start from here if you already have the executables

-bash-4.2$ perf stat ./arrsum.exe 16384
array sum is 17488290749289

 Performance counter stats for './arrsum.exe 16384':
```

```
              0.64 msec task-clock:u                #    0.639 CPUs utilized
                 0      context-switches:u          #    0.000 K/sec
                 0      cpu-migrations:u            #    0.000 K/sec
               146      page-faults:u               #    0.228 M/sec
           753,803      cycles:u                    #    1.178 GHz
         1,056,762      instructions:u              #    1.40  insn per cycle
           299,367      branches:u                  #  467.849 M/sec
             2,093      branch-misses:u             #    0.70% of all branches

       0.001001550 seconds time elapsed

       0.001088000 seconds user
       0.000000000 seconds sys
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~>>
```

* With no events specified, perf stat collects the common events listed above.
Some are software events, such as context-switches, others are generic hardware events such as cycles.
After the hash sign, derived metrics may be presented, such as 'IPC' (instructions per cycle).
Increase the size of the processed array to 32,768. Observe perf output. What changes do you notice?

* We can specify specific events to monitor for both user and kernel level code (uk):

```
<<~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-bash-4.2$ perf stat -e cycles:uk ./arrsum.exe 16384
array sum is 17488294945123

 Performance counter stats for './arrsum.exe 16384':

           693,018      cycles:uku

       0.000919692 seconds time elapsed

       0.001185000 seconds user
       0.000000000 seconds sys
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~>>
```

```
<<~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-bash-4.2$ perf stat -e cycles:u ./arrsum.exe 16384
array sum is 17488294945123

 Performance counter stats for './arrsum.exe 16384':

           663,946      cycles:u

       0.000908855 seconds time elapsed

       0.001163000 seconds user
       0.000000000 seconds sys
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~>>
```

* It is possible to use perf stat to run the same test workload multiple times and get for each count, the standard deviation from the mean.

```
<<~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-bash-4.2$ perf stat -r 5 -e cycles ./arrsum.exe 16384
array sum is 17488294945123
array sum is 17488294945123
```

```
array sum is 17488294945123
array sum is 17488294945123
array sum is 17488294945123

 Performance counter stats for './arrsum.exe 16384' (5 runs):

           670,888      cycles:u
( +-  2.12% )

         0.0008779 +- 0.0000299 seconds time elapsed  ( +-  3.41% )

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~>>
```

5. Sampling with perf record

The perf tool can be used to collect profiles on per-thread, per-process and per-cpu basis.
There are several commands associated with sampling: record, report, annotate.
You must first collect the samples using perf record. This generates an output file called perf.data.
That file can then be analyzed, possibly on another machine, using the perf report and perf annotate commands.
The model is fairly similar to that of OProfile.

```
<<~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-bash-4.2$ perf record ./arrsum.exe 16384
array sum is 17488294945123
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.002 MB perf.data (9 samples) ]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~>>
```

6. Sample analysis with perf report

Samples collected by perf record are saved into a binary file called, by default, perf.data. The perf report command reads this file and generates a concise execution profile. By default, samples are sorted by functions with the most samples first. It is possible to customize the sorting order and therefore to view the data differently.

```
<<~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-bash-4.2$ perf report

Samples: 9  of event 'cycles:uppp', Event count (approx.): 993651
Overhead  Command     Shared Object     Symbol
  42.58%  arrsum.exe  libc-2.17.so      [.] __random
  41.63%  arrsum.exe  ld-2.17.so        [.] check_match.9525
  14.65%  arrsum.exe  ld-2.17.so        [.] _dl_sysdep_start
   1.14%  arrsum.exe  [unknown]         [k] 0xffffffff8518a4ef

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~>>
```

The column 'Overhead' indicates the percentage of the overall samples collected in the corresponding function.
The second column reports the process from which the samples were collected.

In per-thread/per-process mode, this is always the name of the monitored command.
But in cpu-wide mode, the command can vary.
The third column shows the name of the ELF image where the samples came from.
If a program is dynamically linked, then this may show the name of a shared library.
When the samples come from the kernel, then the pseudo ELF image name [kernel.kallsyms] is used.
The fourth column indicates the privilege level at which the sample was taken,
i.e. when the program was running when it was interrupted:
[.] : user level
[k]: kernel level
[g]: guest kernel level (virtualization)
[u]: guest os user space
[H]: hypervisor
The final column shows the symbol name.

```
<<~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-bash-4.2$ $ perf report --sort comm,dso
amples: 10  of event 'cycles:uppp', Event count (approx.): 988532
Overhead  Command     Shared Object
  59.85%  arrsum.exe  libc-2.17.so
  39.28%  arrsum.exe  ld-2.17.so
   0.87%  arrsum.exe  [unknown]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~>>
```

7. Source level analysis with perf annotate

It is possible to drill down to the instruction level with perf annotate. For that, you need to invoke perf annotate with the name of the command to annotate. Perf annotate can generate source code level information if the application is compiled with -ggdb.