Tracing with Perf tools

Namhyung Kim

2013-11-13 Wed

Outline

1. Introduction
2. Other technologies
3. Kernel-level tracing with perf tools
4. User-level tracing with perf tools

tracing

- tracking program execution
- similar to logging
  - lower-level, more-frequently
- code instrumentation

```
func() {
  start_time = gettime();
  ...
  end_time = gettime();
  runtime = end_time - start_time;
  printf("%s: %lu\n", __func__, runtime);
}
```

perf tool basics

- Linux system performance monitoring tool
  - modern, actively developed
  - command line (stdio, tui) or GUI (gtk)
- lives in kernel source tree
  - easy to keep up with kernel-level change
- supports various hw/sw performance events
  - works for both of user-level and kernel-level
  - counting, sampling and tracing

perf event

- hardware PMU events
- kernel software events
- kernel tracepoint events
- hardware raw events
- dynamic software events

perf target

- existing process or thread (–pid, –tid)
- user (–uid)
  - all existing process/thread belong to the user
- command line argument
  - fork & exec new child
- system-wide (–all-cpus)
- cpu (–cpu)

perf commands

- perf top
  - live system profiling
- perf stat
  - read performance counter
- perf record
  - recode profiling events
- perf report
  - report performance bottle-neck

tracing feature in perf tools

- tracepoint events
  - kernel has various (static) tracing information
- perf probe
  - add trace-able events dynamically
- perf script
  - show every traced record
  - also can run various scripts on it
  - currently python and perl supported

manual instrumentation

- what, when, how to record trace info
- users also want to know the timing info
- need precise clock source
  - gettimeofday()
  - clock_gettime()
  - rdtsc or similar

aspect-oriented programming

- weaving
  - combine concerns at compile/run-time
  - primary concern (main logic)
  - cross-cutting concern (secondary logic)
  - join point / pointcut
- http://en.wikipedia.org/wiki/Aspect-oriented_programming

compiler-guided instrumentation

- gcc supports following options
- -pg (-mfentry)
  - mcount()
- -finstrument-functions
  - void __cyg_profile_func_enter()
  - void __cyg_profile_func_exit()
- -finstrument-functions-exclude-{file,function}-list
  - __attribute__((no_instrument_function))

gprof

- profiling tool included in binutils
- use -pg flag when compile binary (gcc)
  - only work for the binary
- using function tracing and sampling
- https://sourceware.org/binutils/docs/gprof/

```
$ gcc -o foo -pg foo.c
$ ./foo
$ gprof foo
```

gcov

- coverage testing tool included in gcc
- compiler generates coverage information
  - count basic block execution
- use without optimization
- generate plain-text output file
- http://gcc.gnu.org/onlinedocs/gcc/Gcov.html

```
$ gcc -o foo --coverage foo.c
$ ./foo
$ gcov foo.c
$ cat foo.c.gcov
```

ptrace

- system call to trace process execution
  - trace new child or attach to existing process
- stopped by single step, syscall or signal
  - check WIFSTOPPED() and WSTOPSIG()
- access to address space and registers
- used by debuggers
- man ptrace(2)

strace

- command line utility to trace syscalls (and signals)
  - using ptrace
- no re-compilation is needed
- can filter specific syscalls
- very comprehensive argument parsing
  - string and structures are dereferenced
- http://sourceforge.net/projects/strace/

ltrace

- command line utility to trace library calls
  - using ptrace
- usage is almost same as strace
- add breakpoint at plt entries
  - and get arguments
- filtering as "function@library"
- also support syscalls and signals
- http://ltrace.org/

LD_PRELOAD

- glibc rtld supports library symbol hooking
- intercept library call to other implementation
  - find symbol before default search path
- wrap and record necessary info
  - save original function with dlsym(RTLD_NEXT, sym)

LD_AUDIT

- glibc rtld supports library auditing
- defines set of interfaces
  - can alter dynamic loader's behavior
  - library search, opening
  - symbol binding, enter/exit
- man rtld-audit(7)

dyninst

- user-level dynamic instrumentation API
  - using ptrace + code patching
- patch binary at runtime
  - mutator process injects snippet code
  - can call function inside the target process
  - need trampoline code to setup arguments
- support x86 and ppc
- http://www.dyninst.org/

ftrace

- function tracer for kernel
  - CONFIG_DYNAMIC_FTRACE
  - using mcount()
- can enable/disable dynamically
- also support other kinds of tracers
- function filtering
  - event trigger
- trace marker
- trace-cmd: front-end tool for ftrace

kprobes

- insert probes to kernel code dynamically
  - pre- and post-handler
- add a breakpoint basically
  - execute original instruction as single-step
  - can be optimized to jump instructions
- jprobes and kretprobes
  - access to arguments and return value

uprobes

- insert probes to user code dynamically
  - handler runs in kernel space
- add a breakpoint basically
  - execute original instruction as single-step
- uretprobes
  - access to return value

systemtap

- use custom script language (tapset)
- supports kernel and user level tracing
  - using kprobes and uprobes
- compile kernel module and load at runtime
  - convert script to C source file
  - compile it with system compiler (gcc)
- SDT defines user-level tracepoints
- debuginfo-based dynamic instrumentation
- http://sourceware.org/systemtap/

LTTng

- using custom kernel module for kernel tracing
- trace kernel and userspace
  - use MONOTONIC clock for co-relate kernel & user space
  - kernel: tracepoint, function, PMU event, kprobes, timer
  - UST: SDT (need recompile), function, syscall
- use CTF (common trace format)
  - also provides converter and graphical viewer
- http://lttng.org/

ktap

- use custom script language
  - based on Lua
- integrated interpreter in kernel
  - no compiler required
  - something like systemtap w/o gcc
- tracepoint, function, kprobes, uprobes, timer
- support x86, arm, ppc, mips
- safety in sandbox
- http://www.ktap.org/

perf kmem

- kernel slap allocator stat

```
usage: perf kmem [<options>] {record|stat}

    -i, --input <file>    input file name
        --caller          show per-callsite statistics
        --alloc           show per-allocation statistics
    -s, --sort <key[,key2...]>
                          sort by keys: ptr, call_site, bytes, hit,
                                        pingpong, frag
    -l, --line <num>      show n lines
        --raw-ip          show raw ip instead of symbol
```

perf sched

- various scheduler events
  - fork, sleep, wakeup, switch, migrate
- finding latency

```
usage: perf sched [<options>] {record|latency|map|replay|script}

    -i, --input <file>    input file name
    -v, --verbose         be more verbose (show symbol address, etc)
    -D, --dump-raw-trace  dump raw trace in ASCII
```

perf lock

- in-kernel locking event
- CONFIG_LOCK_STAT, CONFIG_LOCKDEP required

```
usage: perf lock [<options>] {record|report|script|info}

    -i, --input <file>    input file name
    -v, --verbose         be more verbose (show symbol address, etc)
    -D, --dump-raw-trace  dump raw trace in ASCII
```

perf timechart

- system power events
  - cpufreq, cpuidle, scheduler events
- generate SVG image file to see events

```
usage: perf timechart [<options>] {record}

    -i, --input <file>    input file name
    -o, --output <file>   output file name
    -w, --width <n>       page width
    -P, --power-only      output power data only
    -p, --process <process>
                          process selector. Pass a pid or name.
        --symfs <directory>
                          Look for files with symbols relative to
                          this directory
```

perf ftrace

- front-end to kernel ftrace (WIP)

```
usage: perf ftrace {live|record|show|report} [<options>] [<command>]
   or: perf ftrace {live|record|show|report} [<options>] --
                                             <command> [<options>]

   -t, --tracer <tracer>
                         tracer to use: function_graph or function
   -l, --filter <function[,function,...]>
                         show only these functions in the trace
   -p, --pid <pid>       trace on existing process id
   -v, --verbose         be more verbose
   -a, --all-cpus        system-wide collection from all CPUs
   -C, --cpu <cpu>       list of cpus to monitor
   -c, --clock <clock>   clock to be used for tracer
```

perf trace

- syscall tracing
  - with kernel (page fault and sched) information
  - without hurting performance too much
- depends on audit library (CONFIG_AUDIT)
- argument beautifying
  - get filename from probe:vfs_getname (WIP)

```
usage: perf trace [<options>] [<command>]
   or: perf trace [<options>] -- [<command>] [<options>]
   or: perf trace record [<options>] [<command>].
   or: perf trace record [<options>] -- <command> [<options>]
```

perf probe

- add dynamic probe at runtime
  - using kprobes and uprobes
- depends on libelf and libdw(arf)
- specify specific address or symbol
- SDT and argument fetching is in progress

```
usage: perf probe [<options>] 'PROBEDEF' ['PROBEDEF' ...]
   or: perf probe [<options>] --add 'PROBEDEF' [--add 'PROBEDEF' ...
   or: perf probe [<options>] --del '[GROUP:]EVENT' ...
   or: perf probe --list
   or: perf probe [<options>] --line 'LINEDESC'
   or: perf probe [<options>] --vars 'PROBEPOINT'
```

perf probe event definition

```
-a, --add <[EVENT=]FUNC[@SRC][+OFF|%return|:RL|;PT]|SRC:AL|SRC;PT
                                                  [[NAME=]ARG ...]>
                        probe point definition, where
        GROUP:  Group name (optional)
        EVENT:  Event name
        FUNC:   Function name
        OFF:    Offset from function entry (in byte)
        %return:Put the probe at function return
        SRC:    Source code path
        RL:     Relative line number from function entry.
        AL:     Absolute line number in file.
        PT:     Lazy expression of line code.
        ARG:    Probe argument (local variable name or
                kprobe-tracer argument format.)
```

Q & A

thanks!