<table>
<thead>
<tr>
<th>Times:</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0, 1]</td>
<td>0</td>
</tr>
<tr>
<td>[2, 4]</td>
<td>0</td>
</tr>
<tr>
<td>[4, 8]</td>
<td>0</td>
</tr>
<tr>
<td>[8, 16]</td>
<td>0</td>
</tr>
<tr>
<td>[16, 32]</td>
<td>0</td>
</tr>
<tr>
<td>[32, 64]</td>
<td>0</td>
</tr>
<tr>
<td>[64, 128]</td>
<td>0</td>
</tr>
<tr>
<td>[128, 256]</td>
<td>0</td>
</tr>
<tr>
<td>[256, 512]</td>
<td>25</td>
</tr>
<tr>
<td>[512, 1k]</td>
<td>578</td>
</tr>
<tr>
<td>[1k, 2k]</td>
<td>826</td>
</tr>
<tr>
<td>[2k, 4k]</td>
<td>1969</td>
</tr>
<tr>
<td>[4k, 8k]</td>
<td>1335</td>
</tr>
<tr>
<td>[8k, 16k]</td>
<td>512</td>
</tr>
<tr>
<td>[16k, 32k]</td>
<td>240</td>
</tr>
<tr>
<td>[32k, 64k]</td>
<td>46</td>
</tr>
<tr>
<td>[64k, 128k]</td>
<td>4</td>
</tr>
<tr>
<td>[128k, 256k]</td>
<td>1</td>
</tr>
<tr>
<td>[256k, 512k]</td>
<td>4</td>
</tr>
<tr>
<td>[512k, 1M]</td>
<td>0</td>
</tr>
<tr>
<td>[1M, 2M]</td>
<td>0</td>
</tr>
<tr>
<td>[2M, 4M]</td>
<td>0</td>
</tr>
<tr>
<td>[4M, 8M]</td>
<td>1</td>
</tr>
</tbody>
</table>
Berkeley Packet Filter

# tcpdump -d port 80
(000) ldh [12]
(001) jeq #0x86dd           jt 2 jf 10
(002) ldb [20]
(003) jeq #0x84              jt 6 jf 4
(004) jeq #0x6               jt 6 jf 5
(005) jeq #0x11              jt 6 jf 23
(006) ldh [54]
(007) jeq #0x50              jt 22 jf 8
(008) ldh [56]
(009) jeq #0x50              jt 22 jf 23
(010) jeq #0x800             jt 11 jf 23
...
BPF

32-bit registers

Accumulator

X

16 x 32-bit memory

Scratch pad

Maps

eBPF

64-bit registers

r0

r1

r10

Stack

512 bytes
Linux Requirements

Kprobes (4.1)  kprobe:vfs_read { ... }
Uprobes (4.3)  uprobe:/bin/bash:readline { ... }
USDT (4.3)    Stack traces, per-cpu maps (4.6)
Tracepoints (4.7)  tracepoint:sched:sched_switch { ... }
Timers (4.9)     profile:hz:99 { ... }
Software events (4.9)  software:fauluts: { ... }
Hardware events (4.9)  hardware:cache-references: { ... }
tracepoint:syscalls:sys_enter_read
{
    @mymap = count();
}

tracepoint:syscalls:sys_enter_
{
    @mymap[probe] = count();
}

profile:hz:99
{
    @[stack] = count();
}

kprobe:blk_account_io_start
{
    @start[arg0] = nsecs;
}

kprobe:blk_account_io_completion /@start[arg0]/
{
    $diff = (nsecs - @start[arg0]) / 1000;
    @usecs = hist($diff);
    delete(@start[arg0]);
}
struct Conference
{
    int rating;
    char name[32];
}

uprobe:../a.out:get_rating
{
    $foo = (Conference*)arg0;
    printf("%s gets %d out of 10\n", $foo->name, $foo->rating);
}
tracepoint:syscalls:sys_enter_open
{
    printf("%s %s\n", comm, str(args->filename));
}
Builtins

- `pid` - Process ID (kernel tgid)
- `tid` - Thread ID (kernel pid)
- `cgroup` - Cgroup ID of the current process
- `uid` - User ID
- `gid` - Group ID
- `nsecs` - Nanosecond timestamp
- `cpu` - Processor ID
- `comm` - Process name
- `stack` - Kernel stack trace
- `ustack` - User stack trace
- `arg0, arg1, ... etc.` - Arguments to the function being traced
- `retval` - Return value from function being traced
- `func` - Name of the function currently being traced
- `probes` - Full name of the probe
- `curtask` - Current task_struct as a u64
- `rand` - Random number of type u32

- `hist(int n)` - Produce a log2 histogram of values of n
- `lhist(int n, int min, int max, int step)` - Produce a linear histogram of values of n
- `count()` - Count the number of times this function is called
- `sum(int n)` - Sum this value
- `min(int n)` - Record the minimum value seen
- `max(int n)` - Record the maximum value seen
- `avg(int n)` - Average this value
- `stats(int n)` - Return the count, average, and total for this value
- `delete(@x)` - Delete the map element passed in as an argument
- `str(char *s)` - Returns the string pointed to by s
- `printf(char *fmt, ...)` - Print formatted to stdout
- `print(@x[, int top [, int div]])` - Print a map, with optional top entry count and divisor
- `clear(@x)` - Delete all key/values from a map
- `sym(void *p)` - Resolve kernel address
- `usym(void *p)` - Resolve user space address
- `kaddr(char *name)` - Resolve kernel symbol name
- `uaddr(char *name)` - Resolve user space symbol name
- `reg(char *name)` - Returns the value stored in the named register
- `join(char *arr[])` - Prints the string array
- `time(char *fmt)` - Print the current time
- `system(char *fmt)` - Execute shell command
- `exit()` - Quit bpftrace
BEGIN
{
    printf("Tracing bash commands... Hit Ctrl-C to end.\n");
    printf("%-9s %-6s %s\n", "TIME", "PID", "COMMAND");
}

uretprobe:/bin/bash:readline
{
    time("%H:%M:%S ");
    printf("%-6d %s\n", pid, str(retval));
}

from __future__ import print_function
from bcc import BPF
from time import strftime
import ctypes as ct

# load BPF program
bpf_text = """"#include <uapi/linux/ptrace.h>
struct str_t {
    u64 pid;
    char str[80];
};
BPF_PERF_OUTPUT(events);
int printret(struct pt_regs *ctx) {
    struct str_t data = {};
    u32 pid;
    if (!PT_REGS_RC(ctx))
        return 0;
    pid = bpf_get_current_pid_tgid();
    data.pid = pid;
    bpf_probe_read(&data.str, sizeof(data.str), (void *)PT_REGS_RC(ctx));
    events.perf_submit(ctx, &data, sizeof(data));
    return 0;
};
"

STR_DATA = 80
class Data(ct.Structure):
    _fields_ = [('pid', ct.c_ulonglong),
                ('str', ct.c_char * STR_DATA)]

b = BPF(text=bpf_text)
b.attach_uretprobe(name="/bin/bash", sym="readline", fn_name="printret")

# header
print("%-9s %-6s %s\n" (%(TIME", "PID", "COMMAND")))

def print_event(cpu, data, size):
    event = ct.cast(data, ct.POINTER(Data)).contents
    print("%-9s %-6d %s\n" % (strftime("%H:%M:%S"), event.pid,
        event.str.decode('utf-8', 'replace')))
### Future Work

<table>
<thead>
<tr>
<th>Issue</th>
<th>Title</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>#211</td>
<td>Building With Multiple Versions Of Clang Installed</td>
<td>2 hours ago by att'la</td>
</tr>
<tr>
<td>#209</td>
<td><code>sym()</code> resolves symbols in aggregations, but not in print</td>
<td>a day ago by Birch-san</td>
</tr>
<tr>
<td>#207</td>
<td>cannot run opensnoop.bt</td>
<td>3 days ago by yujinjui</td>
</tr>
<tr>
<td>#206</td>
<td>Error loading program, with no details (kern version)</td>
<td>4 days ago by brendangregg</td>
</tr>
<tr>
<td>#201</td>
<td>bpftrace crashes during finalization with specific one-liner</td>
<td>5 days ago by carin!</td>
</tr>
<tr>
<td>#199</td>
<td>str() to accept optional length</td>
<td>5 days ago by brendangregg</td>
</tr>
<tr>
<td>#198</td>
<td>strings in predicates failures</td>
<td>5 days ago by brendangregg</td>
</tr>
<tr>
<td>#197</td>
<td>bash command completions</td>
<td>6 days ago by brendangregg</td>
</tr>
<tr>
<td>#194</td>
<td>Use after free from BPFtrace::get_arg_values</td>
<td>7 days ago by ajor</td>
</tr>
<tr>
<td>#192</td>
<td>if statement semicolon</td>
<td>7 days ago by brendangregg</td>
</tr>
<tr>
<td>#191</td>
<td>Tidy USAGE message</td>
<td>7 days ago by brendangregg</td>
</tr>
<tr>
<td></td>
<td>Fix codegen tests</td>
<td></td>
</tr>
</tbody>
</table>