PROFILING DESKTOP APPS
SPARE THE ROD
SPOIL THE APP

wmealing@redhat.com  eteo@redhat.com
PROFILING IS A MIX OF SCIENCE AND DETECTIVE WORK
ENGAGE THE COMMUNITY
Who Profiles Applications?

- Software developers
- System architects
- Benchmarkers
Profiling life cycle

1. Prepare Environment
2. Track down and solve
3. Upstream acceptance
Stage 1. Preparing environment

- Simulate "production" environment
- Reliable hardware
- Eliminate Variables
- Disable disruptive services
  - CRON
  - Log Rotation
  - CPUSPEED
Stage 2.
Track down and Solve

1. Define a goal
2. Create and automate a test case
3. Build a hypothesis
4. Attempt a solution
5. Improvement?
6. Run test case
   Compare stats
Stage 2.
Track down and Solve

• Hints:
  – Work with community members
  – Consistency – RESET BETWEEN TESTS
  – 80/20 rule
Stage 3.
Upstream Acceptance

- Present objective case to dev community
- Share test case
- Share code
- Accept criticism
- Accept failure
- Try, try again
General tools of the trade

- Traditional monitoring tools:
  - top, ps, /proc interface
  - systat (vmstat, iostat)
  - strace, ltrace
  - free
- Not fine grained or “immediately” accurate.
- Problems may not be readily exposed
Tool of the trade - valgrind

- Memory misuse
- Thread misuse
- Cache Profiler
Tool of the trade - oprofile

- Sample based
- Uses hardware performance counters
- Profile application and kernel code
- Generate instruction level profiles
- Pinpoint functions that need to be optimized
**Tool of the trade - SystemTap**

- Trace, monitor, and observe
- Able to probe kernel-space applications
- Supports dynamic and static probing
- User-space instrumentation in the works
- Free/Open Source Software (GPL)
Tool of the trade - SystemTap

Real time Performance Analysis
Low overhead and safe for production systems
Functional problem analysis
On demand probing
Continuous Performance Monitoring
Customize monitoring tools
Tool of the trade - SystemTap

1. Parse
2. Elaborate
3. Translate
4. Compile
5. Run

Probe Script
Tapset Library
Probe LKM
Probe Output
Get output
Tool of the trade - SystemTap

global reads
probe begin {
    printf("probe begins\n")
}
probe syscall.read {
    reads[execname()] <<< count
}
probe end {
    foreach (programe in reads) {
        printf("%s reads: %d,\n", programe,
            @count(reads[programe]))
        printf("total bytes: %d, avg: %d\n", 
            @sum(reads[programe]),
            @avg(reads[programe]))
    }
}

- Global variables
- Built-in functions
- Associative arrays
- Aggregation operations and functions
- Pre-defined tapsets
- Probe entry and termination call-backs
Tool of the trade - SystemTap

```c
#define GLOBALGLOBALS

int main() {
  printf("probe begins\n")
}

int probe_syscall_read {
  reads[execname()] <<< count
}

int probe_end {
  foreach (program in reads) {
    printf("%s reads: %d, \n", program,
           @count(reads[program]))
    printf("total bytes: %d, avg: %d\n",,
           @sum(reads[program]),
           @avg(reads[program]))
  }
}
```
Tool of the trade - SystemTap

```c
global reads
probe begin {
   printf("probe begins\n")
}
probe syscall.read {
   reads[execname()] <<< count
}
probe end {
   foreach (progname in reads) {
      printf("%s reads: %d, ", progname,
       @count(reads[progname]))
      printf("total bytes: %d, avg: %d\n", 
       @sum(reads[progname]),
       @avg(reads[progname]))
   }
}
```

- Global variables
- Built-in functions
- Associative arrays
- Aggregation operations and functions
- Pre-defined tapsets
- Probe entry and termination call-backs
Tool of the trade - SystemTap

```c
global reads
probe begin {
    printf("probe begins\n")
}
probe syscall.read {
    reads[execname()] <<< count
}
probe end {
    foreach (progname in reads) {
        printf("%s reads: %d, ", progname,
                @count(reads[progname]))
        printf("total bytes: %d, avg: %d\n", 
                @sum(reads[progname]),
                @avg(reads[progname]))
    }
}
```

- Global variables
- Built-in functions
- Associative arrays
- Aggregation operations and functions
- Pre-defined tapsets
- Probe entry and termination call-backs
global reads
probe begin {
    printf("probe begins\n")
}
probe syscall.read {
    reads[execname()] <<< count
}
probe end {
    foreach (progname in reads) {
        printf("%s reads: %d, ", progname,
            @count(reads[progname]))
        printf("total bytes: %d, avg: %d\n", 
            @sum(reads[progname]),
            @avg(reads[progname]))
    }
}

- Global variables
- Built-in functions
- Associative arrays
- Aggregation operations and functions
- Pre-defined tapsets
- Probe entry and termination call-backs
Tools that we avoid - dtrace

- Similar to SystemTap but implemented differently; has its own D language
- Trace, monitor, and observe
- Predefined probe points
- Able to probe both user/kernel-space apps
- CDDL != GPL
- Solaris && !Linux
Spyglass

- Spyglass \(\text{Spy\textsuperscript{"g}lass}\) (-gl[.a]s`), n.
  A small telescope for viewing distant terrestrial objects. [1913 Webster]
- Consists of a profiler and graphical plotting tool
- Profiler can call SystemTap, shell script, vmstat, etc, save logs, and visualize it with Spyglass
- Very early alpha; still in development
Spyglass

Some screenshots:
War Stories

- UDP datagram loss
- SCSI request size mismatch
- Top I/O users by userid
UDP Datagram Loss

• Problem:
  − Customer wanted to see UDP statistics for both sending and receiving sides and how many UDP datagrams were dropped.
  − `netstat -su` don't show how many datagrams are dropped when sending.
  − `Iptraf` don't show statistics on datagram loss

• Solution:
  − Write a simple SystemTap script to find out
UDP Datagram Loss

# Thanks to Eugene Teo from Red Hat

global udp_out, udp_outerr, udp_in, udp_inerr, udp_noport
probe begin {
    printf("%11s %10s %10s %10s %10s
",
        "UDP_out", "UDP_outErr", "UDP_in", "UDP_inErr", "UDP_noPort")
}
probe kernel.function("udp_sendmsg").return {
    $return >= 0 ? udp_out++ : udp_outerr++
}
probe kernel.function("udp_queue_rcv_skb").return {
    $return == 0 ? udp_in++ : udp_inerr++
}
probe kernel.function("icmp_send") {
    /* icmp_send(skb, ICMP_DEST_UNREACH, ICMP_PORT_UNREACH, 0); */
    if ($type == 3 && $code == 3) {
        if ($skb_in->nh->iph->protocol == 17) /* UDP */
            udp_noport++
    }
}
probe timer.ms(1000) {
    printf("%11s %10s %10s %10s %10s
",
        "UDP_out", "UDP_outErr", "UDP_in", "UDP_inErr", "UDP_noPort")
}
## UDP Datagram Loss

```
$ ./udpstat.stp

<table>
<thead>
<tr>
<th>UDP_out</th>
<th>UDP_outErr</th>
<th>UDP_in</th>
<th>UDP_inErr</th>
<th>UDP_noPort</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>19</td>
<td>1</td>
<td>9</td>
<td>0</td>
<td>6</td>
</tr>
</tbody>
</table>
```
SCSI Request Sizes

• Problem:
  – In a benchmark run, we observed a mismatch between expected and actual SCSI I/O counts

• Solution:
  – Create a simple SystemTap script to track the counts and sizes of SCSI requests to a specific device
# Thanks to Allan Brunelle from HP

global rqss, host_no, channel, id, lun, direction

probe begin {
    host_no = 0
    channel = 1
    id = 1
    lun = 0
    direction = 1 /* write */
}

probe scsi.iодиспачинг {
    if (data_direction != direction) next
    if (lun != lun) next
    if (id != dev_id) next
    if (channel != channel) next
    if (host_no != host_no) next
    rqss[req_buflen / 1024]++
}

probe end {
    printf("ReqSz(KB)\t#Rqs\n")
    foreach (rec+ in rqss)
        printf("%8d\t%5d\n", rec, rqss[rec])
}
## SCSI Request Sizes

$ ./scsi_req.stp

<table>
<thead>
<tr>
<th>ReqSz (KB)</th>
<th>#Reqs</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>28</td>
<td>1</td>
</tr>
<tr>
<td>44</td>
<td>1</td>
</tr>
<tr>
<td>88</td>
<td>1</td>
</tr>
<tr>
<td>164</td>
<td>1</td>
</tr>
<tr>
<td>204</td>
<td>1</td>
</tr>
<tr>
<td>216</td>
<td>1</td>
</tr>
<tr>
<td>308</td>
<td>1</td>
</tr>
<tr>
<td>448</td>
<td>1</td>
</tr>
<tr>
<td>508</td>
<td>1</td>
</tr>
<tr>
<td>512</td>
<td>36</td>
</tr>
</tbody>
</table>
Top I/O Users by Userid

• Problem:
  - Which user is doing the most I/O on the system? Iostat does not provide statistics on a per user basis

• Solution:
  - Write a simple SystemTap script that probes file system read() and write() and records the bytes of I/O for each user
# Thanks to Mike Grundy and Mike Mason from IBM

global reads, writes
function print_top () {
    cnt=0
    printf("%-10s\t%10s\t%15s\n", "User ID", "KB Read", "KB Written")
    foreach (id in reads-) {
        printf("%-10s\t%10d\t%15d\n", id, reads[id]/1024, writes[id]/1024)
        if (cnt++ == 5)
            break
    }
    delete reads
    delete writes
} 

probe kernel.function("vfs_read") {
    reads[sprintf("%d", uid())] += $count
}

probe kernel.function("vfs_write") {
    writes[sprintf("%d", uid())] += $count
}

probe timer.ms(5000) {
    print_top ()
}
Top I/O Users by Userid

$ ./uid-iotop.stp

<table>
<thead>
<tr>
<th>User ID</th>
<th>KB Read</th>
<th>KB Written</th>
</tr>
</thead>
<tbody>
<tr>
<td>504</td>
<td>14237</td>
<td>3163</td>
</tr>
<tr>
<td>505</td>
<td>11208</td>
<td>929</td>
</tr>
<tr>
<td>502</td>
<td>11175</td>
<td>889</td>
</tr>
<tr>
<td>503</td>
<td>12469</td>
<td>866</td>
</tr>
<tr>
<td>0</td>
<td>1778</td>
<td>1831</td>
</tr>
</tbody>
</table>
More War Stories

- http://sourceware.org/systemtap/wiki/WarStories
To find out more

- Eugene Teo – eteo@redhat.com
- Wade Mealing - wmealing@redhat.com
- Or come talk to us afterwards!