Overhead Evaluation about Kprobes and Djprobe (Direct Jump Probe)
Masami Hiramatsu <hiramatu@sdl.hitachi.co.jp>
Hitachi, Ltd., SDL
Jul. 13. 2005

1. Abstract
To implement “flight recorder” system, the overhead of probes should be as small as possible. It could be estimated that the overhead of kprobes and jprobe would be heavy, because those used int3 interruption. To reduce this overhead, I developed a new probe method called “djprobe”. I evaluated the performance of kprobes, jprobe and djprobe on various IA32 processors. The results indicated that the djprobe was 10 times or more as fast as other probes. And I also evaluated the performance of probing and recording by using LKST′s recording function. The results indicated that the djprobe was as light as LKST′s hook method.

2. New probe method -- Djprobe (Direct jump probe)
I developed the new light weight probe that uses ‘jmp’ instruction instead of ‘int3’. Djprobe provides the non-locking and low overhead probe function. It uses the relative ‘jmp’ opcode instead of ‘int3’ breakpoint opcode. It can reduce overheads of probing by the interruption, single-stepping and locking.

As is well known, it is not assured that both a ‘jmp’ instruction and a destination address are inserted atomically. To avoid this atomic operation problem, djprobe bypasses the section under 'construction' by using kprobes.

The detailed documentation about djprobe is in the README text file included in the source package (tar ball) of djprobe. You can get it from http://lkst.sourceforge.net.

3. Benchmark Program
I developed a small micro benchmark test program described below:

- gtodbench:

  Micro benchmark program that repeats gettimeofday system call for 10 seconds. It counts the number of execution and calculates average processing time. This program is distributed with the djprobe.

To measure performance by using gtodbench, I inserted probe function into the head of the sys_gettimeofday() function that is called once on each gettimeofday system call.

---

1 LKST: Linux Kernel State Tracer (http://lkst.sourceforge.net)
4. Evaluation 1: Measurement overheads of probes

I measured the overhead of probes on the latest Linux kernel 2.6.12.

4.1. Probe handlers

To measure just the overhead of probing, I used the non-operation probing functions shown in List 1, List 2 and List 3. Each function does nothing. In the case of kprobe, I used only pre-handler.

**List 1 kprobe's no-op handler**

```c
int kprobe_func(struct kprobe *kp, struct pt_regs *regs)
{
    return 0;
}
```

**List 2 jprobe's no-op handler**

```c
long jprobe_func(struct timeval __user *tv, struct timezone __user *tz)
{
    jprobe_return();
    return 0;
}
```

**List 3 djprobe's no-op handler**

```c
void djprobe_func(struct djprobe *djp, struct pt_regs *regs)
{
    return ;
}
```
4.2. Target machine:

Specifications of machine used for evaluation:
CPU: Pentium4 3.06GHz 512KB cache (with Hyper Threading)
Memory: 1GB
Distribution: Fedora Core 3
Kernel Version: 2.6.12 (SMP kernel)

4.3. Benchmark Results

Benchmarking results are shown in List 4 and Figure 1.

**List 4 Results of Measurement by using no-op handler**

<table>
<thead>
<tr>
<th></th>
<th>linux-2.6.12</th>
<th>kprobes</th>
<th>jprobe</th>
<th>djprobe</th>
</tr>
</thead>
<tbody>
<tr>
<td>gtodbench(sample1)</td>
<td>262</td>
<td>1326</td>
<td>1876</td>
<td>290</td>
</tr>
<tr>
<td>gtodbench(sample2)</td>
<td>262</td>
<td>1327</td>
<td>1874</td>
<td>290</td>
</tr>
<tr>
<td>gtodbench(sample3)</td>
<td>262</td>
<td>1326</td>
<td>1877</td>
<td>290</td>
</tr>
<tr>
<td>gtodbench(sample4)</td>
<td>262</td>
<td>1328</td>
<td>1875</td>
<td>290</td>
</tr>
<tr>
<td>average(nsec)</td>
<td>262</td>
<td>1326.75</td>
<td>1875.5</td>
<td>290</td>
</tr>
<tr>
<td>overhead(nsec)</td>
<td>0</td>
<td>1064.75</td>
<td>1613.5</td>
<td>28</td>
</tr>
</tbody>
</table>

**Figure 1 Overheads of probes**
4.4. Summary of Evaluation 1

We can see followings about each probe.
- kprobe consumes more than 1 micro second to probe.
- jprobe consumes more than 1.5 micro seconds.
- djprobe consumes less than 0.03 micro seconds.

The new djprobe method is about 30 times as fast as the kprobe method. I guess the overheads of kprobe and jprobe mainly come from int3 and trap interruptions and its software architecture (reference of probe table, single-step execution, etc.). The kprobe uses int3 once, and the jprobe uses it twice. The results seem to reflect that.

5. Evaluation 2: Evaluate overheads on various IA32 processors

There are some kinds of processors which support IA32. There are also the differences of number of pipe-line stages, width of memory bus, and size of cache at each processor. Such differences of implementation may cause different effects on performance. Thus, I measured these effects for the overhead of kprobe, jprobe and djprobe. To expect impartiality, I measured overheads on Uni-Processor environment.
5.1. Processor 1: Pentium 4 (UP)

5.1.1. Machine Specifications
- Processor: Pentium 4 3.06GHz 512KB cache HT-off (UP)
- Memory Size: 1024MB
- OS: Fedora Core 3
- Kernel Version: 2.6.12 (UP kernel)

5.1.2. Benchmark Results
Benchmarking results are shown in List 5 and Figure 2. Both overheads of kprobes and jprobe are over 1 micro-sec. And the overhead of djprobe is about 0.03 micro-sec.

List 5 Results of Measurement on Pentium 4

<table>
<thead>
<tr>
<th></th>
<th>linux-2.6.12</th>
<th>kprobes</th>
<th>jprobe</th>
<th>djprobe</th>
</tr>
</thead>
<tbody>
<tr>
<td>gtodbench(sample1)</td>
<td>230</td>
<td>1247</td>
<td>1808</td>
<td>260</td>
</tr>
<tr>
<td>gtodbench(sample2)</td>
<td>230</td>
<td>1246</td>
<td>1809</td>
<td>261</td>
</tr>
<tr>
<td>gtodbench(sample3)</td>
<td>230</td>
<td>1245</td>
<td>1806</td>
<td>260</td>
</tr>
<tr>
<td>gtodbench(sample4)</td>
<td>230</td>
<td>1247</td>
<td>1809</td>
<td>260</td>
</tr>
<tr>
<td>average(nsec)</td>
<td>230</td>
<td>1246.25</td>
<td>1808</td>
<td>260.25</td>
</tr>
<tr>
<td>overhead(nsec)</td>
<td>0</td>
<td>1016.25</td>
<td>1578</td>
<td>30.25</td>
</tr>
</tbody>
</table>

![overhead of probes](attachment:image.png)

Figure 2 Overheads of probes on Pentium4 (UP)
5.2. Processor 2: Athlon64 (Legacy 32bits mode)

5.2.1. Machine Specifications
Processor: Athlon64 2400+ (32bits) 512KB cache
Memory Size: 1024MB
OS: Fedora Core 4
Kernel Version: 2.6.12 (UP kernel)

5.2.2. Benchmark Results
Benchmarking results are shown in List 6 and Figure 3. The overhead of kprobes is over 1.7 micro-secs. And the overhead of jprobe is about 0.63 micro-secs. And the overhead of djprobe is less than 0.03 micro-secs.

List 6 Results of Measurement on Athlon64 (Legacy mode)

<table>
<thead>
<tr>
<th></th>
<th>linux-2.6.12</th>
<th>kprobes</th>
<th>jprobe</th>
<th>djprobe</th>
</tr>
</thead>
<tbody>
<tr>
<td>gtodbench(sample1)</td>
<td>2232</td>
<td>3995</td>
<td>2921</td>
<td>2254</td>
</tr>
<tr>
<td>gtodbench(sample2)</td>
<td>2231</td>
<td>4008</td>
<td>2844</td>
<td>2255</td>
</tr>
<tr>
<td>gtodbench(sample3)</td>
<td>2229</td>
<td>4004</td>
<td>2842</td>
<td>2255</td>
</tr>
<tr>
<td>gtodbench(sample4)</td>
<td>2229</td>
<td>3998</td>
<td>2844</td>
<td>2254</td>
</tr>
<tr>
<td>average(nsec)</td>
<td>2230.25</td>
<td>4001.25</td>
<td>2862.75</td>
<td>2254.5</td>
</tr>
<tr>
<td>overhead(nsec)</td>
<td>0</td>
<td>1771</td>
<td>632.5</td>
<td>24.25</td>
</tr>
</tbody>
</table>

Figure 3 Overheads of probes on Athlon64 (Legacy mode)
5.3. Processor 3: Pentium M

5.3.1. Machine Specifications
Processor: Pentium M 1600MHz 1MB cache
Memory Size: 758MB
OS: Fedora Core 4
Kernel Version: 2.6.12 (UP kernel)

5.3.2. Benchmark Results
Benchmarking results are shown in List 7 and Figure 4. The overhead of kprobes is over 2 micro-secs. And the overhead of jprobe is about 0.78 micro-secs. And the overhead of djprobe is about -0.12 micro-secs.

List 7 Results of Measurement on Pentium M

<table>
<thead>
<tr>
<th></th>
<th>linux-2.6.12</th>
<th>kprobes</th>
<th>jprobe</th>
<th>djprobe</th>
</tr>
</thead>
<tbody>
<tr>
<td>gtodbench(sample1)</td>
<td>2497</td>
<td>4699</td>
<td>3293</td>
<td>2402</td>
</tr>
<tr>
<td>gtodbench(sample2)</td>
<td>2497</td>
<td>4694</td>
<td>3297</td>
<td>2406</td>
</tr>
<tr>
<td>gtodbench(sample3)</td>
<td>2500</td>
<td>4699</td>
<td>3298</td>
<td>2402</td>
</tr>
<tr>
<td>gtodbench(sample4)</td>
<td>2562</td>
<td>4730</td>
<td>3295</td>
<td>2386</td>
</tr>
<tr>
<td>average(nsec)</td>
<td>2514</td>
<td>4705.5</td>
<td>3295.75</td>
<td>2399</td>
</tr>
<tr>
<td>overhead(nsec)</td>
<td>0</td>
<td>2191.5</td>
<td>781.75</td>
<td>-115</td>
</tr>
</tbody>
</table>

Figure 4 Overheads of probes on Pentium M
5.4. Summary of Evaluation 2

We could see there are big differences depended on the processor implementations. In each case, the djprobe method was 10 times or more as fast as other methods. Especially, on Pentium M processor, it seemed that djprobe accelerates the gettimeofday system call. I conjectured that depended on the cache implementation of Pentium M. And jprobe method has been used kprobes, but it was faster than kprobes in Pentium M and Athlon64.

6. Evaluation 3: Evaluate overheads of probing and recording

It will be shown that the overheads given by new flight recorder come of probing and recording. Thus I measured the overheads of the combination of each probe method and LKST’s recording function (buffer). For comparison, I also measured the overhead of LKST handler.

6.1. Probe handlers

I used the recording handlers shown in List 8, List 9 and List 10 for benchmarking. Each function calls a LKST’s recording function and it records 64 bytes information of event (it is including tsc, pid, event-id, arguments, etc..). In the case of kprobe, I used only pre-handler.

List 8 kprobe’s recording handler

```c
int kprobe_func(struct kprobe *kp, struct pt_regs *regs)
{
    lkst_evhandlerprim_entry_log(LKST_ETYPE_KPROBE_INFO,
        LKST_ARGP(kp->addr),
        LKST_ARG32(regs->eax,regs->edx),
        LKST_ARG32(regs->ecx,((unsigned long*)regs->esp)[1]),
        LKST_ARGP(((unsigned long*)regs->esp)[0]));
    return 0;
}
```
List 9 jprobe's recording handler

```c
long jprobe_func(struct timeval __user *tv, struct timezone __user *tz)
{
    lkst_evhandlerprim_entry_log(LKST_ETYPE_JPROBE_INFO,
        LKST_ARGP((void*)addr), LKST_ARGP(tv),
        LKST_ARGP(tz), LKST_ARG(0));
    jprobe_return();
    return 0;
}
```

List 10 djprobe's recording handler

```c
void djprobe_func(struct djprobe *djp, struct pt_regs *regs)
{
    lkst_evhandlerprim_entry_log(LKST_ETYPE_DJPROBE_INFO,
        LKST_ARGP(kp->addr),
        LKSTARG32(regs->eax,regs->edx),
        LKSTARG32(regs->ecx, ((unsigned long*)regs->esp)[1]),
        LKST_ARGP(((unsigned long*)regs->esp)[0]));
    return ;
}
```
6.2. Processor 1: Pentium 4 (HT):

6.2.1. Machine Specifications

CPU: Pentium4 3.06GHz 512KB cache (with Hyper Threading)
Memory: 1GB
Distribution: Fedora Core 3
Kernel Version: 2.6.12-lkst23 (SMP kernel)

6.2.2. Benchmark Results

Benchmarking results are shown in List 11 and Figure 5. Both the overheads of kprobes and jprobe are over 1 micro-sec. And the overheads of djprobe and LKST are about 0.1 micro-secs.

**List 11 Results of Measurement by using LKST recording handler**

<table>
<thead>
<tr>
<th></th>
<th>linux-2.6.12-lkst23</th>
<th>kprobes</th>
<th>jprobe</th>
<th>djprobe</th>
<th>LKST (syscall entry)</th>
</tr>
</thead>
<tbody>
<tr>
<td>gtodbench(sample1)</td>
<td>261</td>
<td>1428</td>
<td>1969</td>
<td>375</td>
<td>330</td>
</tr>
<tr>
<td>gtodbench(sample2)</td>
<td>261</td>
<td>1429</td>
<td>1972</td>
<td>375</td>
<td>330</td>
</tr>
<tr>
<td>gtodbench(sample3)</td>
<td>261</td>
<td>1433</td>
<td>1971</td>
<td>375</td>
<td>330</td>
</tr>
<tr>
<td>gtodbench(sample4)</td>
<td>261</td>
<td>1430</td>
<td>1970.5</td>
<td>375</td>
<td>330</td>
</tr>
<tr>
<td>average(nsec)</td>
<td>261</td>
<td>1430</td>
<td>1970.5</td>
<td>375</td>
<td>330</td>
</tr>
<tr>
<td>overhead(nsec)</td>
<td>0</td>
<td>1169</td>
<td>1709.5</td>
<td>114</td>
<td>69</td>
</tr>
</tbody>
</table>

**Figure 5 Overheads of probing and recording on Pentium 4 (HT)**
6.3. Processor 1: Pentium 4 (UP)

6.3.1. Machine Specifications

Processor: Pentium 4 3.06GHz 512KB cache HT-off (UP)
Memory Size: 1024MB
OS: Fedora Core 3
Kernel Version: 2.6.12-lkst23 (UP kernel)

6.3.2. Benchmark Results

Benchmarking results are shown in List 12 and Figure 6. Both overheads of kprobes and jprobe are over 1 micro·sec. And the overhead of djprobe is about 0.1 micro·sec.

**List 12 Results of Measurement on Pentium 4**

<table>
<thead>
<tr>
<th></th>
<th>linux-2.6.12</th>
<th>kprobes</th>
<th>jprobe</th>
<th>djprobe</th>
<th>LKST (syscall entry)</th>
</tr>
</thead>
<tbody>
<tr>
<td>gtodbench(sample1)</td>
<td>231</td>
<td>1331</td>
<td>1895</td>
<td>338</td>
<td>296</td>
</tr>
<tr>
<td>gtodbench(sample2)</td>
<td>231</td>
<td>1334</td>
<td>1895</td>
<td>338</td>
<td>296</td>
</tr>
<tr>
<td>gtodbench(sample3)</td>
<td>232</td>
<td>1334</td>
<td>1900</td>
<td>338</td>
<td>296</td>
</tr>
<tr>
<td>gtodbench(sample4)</td>
<td>231</td>
<td>1332</td>
<td>1899</td>
<td>338</td>
<td>296</td>
</tr>
<tr>
<td>average(nsec)</td>
<td>231.25</td>
<td>1332.75</td>
<td>1897.25</td>
<td>338</td>
<td>296</td>
</tr>
<tr>
<td>overhead(nsec)</td>
<td>0</td>
<td>1101.5</td>
<td>1666</td>
<td>106.75</td>
<td>64.75</td>
</tr>
</tbody>
</table>

![overhead of probing and recording]

**Figure 6 Overheads of probes on Pentium4 (UP)**
### 6.4. Processor 2: Athlon64 (Legacy 32bits mode)

#### 6.4.1. Machine Specifications

- Processor: Athlon64 2400+ (32bits) 512KB cache
- Memory Size: 1024MB
- OS: Fedora Core 4
- Kernel Version: 2.6.12-lkst23 (UP kernel)

#### 6.4.2. Benchmark Results

Benchmarking results are shown in List 13 and Figure 7. The overhead of kprobes is over 0.46 micro-secs. And the overhead of jprobe is about 0.65 micro-secs. And the overhead of djprobe and LKST are less than 0.1 micro-secs.

**List 13 Results of Measurement on Athlon64 (Legacy mode)**

<table>
<thead>
<tr>
<th></th>
<th>linux-2.6.12-lkst23</th>
<th>kprobes</th>
<th>jprobe</th>
<th>djprobe</th>
<th>LKST (syscall entry)</th>
</tr>
</thead>
<tbody>
<tr>
<td>gtodbench(sample1)</td>
<td>2257</td>
<td>2717</td>
<td>2910</td>
<td>2328</td>
<td>2313</td>
</tr>
<tr>
<td>gtodbench(sample2)</td>
<td>2254</td>
<td>2716</td>
<td>2906</td>
<td>2326</td>
<td>2317</td>
</tr>
<tr>
<td>gtodbench(sample3)</td>
<td>2255</td>
<td>2717</td>
<td>2903</td>
<td>2327</td>
<td>2316</td>
</tr>
<tr>
<td>gtodbench(sample4)</td>
<td>2256</td>
<td>2716</td>
<td>2902</td>
<td>2329</td>
<td>2319</td>
</tr>
<tr>
<td>average(nsec)</td>
<td>2255.5</td>
<td>2716.5</td>
<td>2905.25</td>
<td>2327.5</td>
<td>2316.25</td>
</tr>
<tr>
<td>overhead(nsec)</td>
<td>0</td>
<td>461</td>
<td>649.75</td>
<td>72</td>
<td>60.75</td>
</tr>
</tbody>
</table>

![Figure 7 Overheads of probes on Athlon64 (Legacy mode)](image-url)
6.5. Processor 3: Pentium M

6.5.1. Machine Specifications
Processor: Pentium M 1600MHz 1MB cache
Memory Size: 758MB
OS: Fedora Core 4
Kernel Version: 2.6.12-lkst23 (UP kernel)

6.5.2. Benchmark Results
Benchmarking results are shown in List 14 and Figure 8. The overhead of kprobes is over 2 micro-secs. And the overhead of jprobe is about 0.78 micro-secs. And the overheads of djprobe and LKST are about -0.1 micro-secs.

List 14 Results of Measurement on Pentium M

<table>
<thead>
<tr>
<th></th>
<th>linux-2.6.12</th>
<th>kprobes</th>
<th>jprobe</th>
<th>djprobe</th>
<th>LKST (syscall entry)</th>
</tr>
</thead>
<tbody>
<tr>
<td>gtodbench(sample1)</td>
<td>2337</td>
<td>3053</td>
<td>3311</td>
<td>2391</td>
<td>2435</td>
</tr>
<tr>
<td>gtodbench(sample2)</td>
<td>2332</td>
<td>3054</td>
<td>3281</td>
<td>2394</td>
<td>2435</td>
</tr>
<tr>
<td>gtodbench(sample3)</td>
<td>2333</td>
<td>3054</td>
<td>3301</td>
<td>2392</td>
<td>2436</td>
</tr>
<tr>
<td>gtodbench(sample4)</td>
<td>2333</td>
<td>3052</td>
<td>3205</td>
<td>2392</td>
<td>2435</td>
</tr>
<tr>
<td>average(nsec)</td>
<td>2333.75</td>
<td>3053.25</td>
<td>3274.5</td>
<td>2392.25</td>
<td>2435.25</td>
</tr>
<tr>
<td>overhead(nsec)</td>
<td>0</td>
<td>719.5</td>
<td>940.75</td>
<td>58.5</td>
<td>101.5</td>
</tr>
</tbody>
</table>

![overhead of probing and recording](image)

**Figure 8** Overheads of probes on Pentium M
6.6. Summary of Evaluation 3

Same as previous results, the overhead of djprobe was remarkably low. The overheads of kprobes and jprobe were not so heavy on Athlon64 and Pentium M. But the overhead of djprobe was almost same order as LKST's overhead. Those were 6 times or more fast than kprobes and jprobe.

If I would develop a new LKST based on kprobes, it will be 6 times or more heavy than LKST. Whole overhead of LKST with recording the standard kernel events is about 3% (it evaluated by kernel-build time). Thus, by simple arithmetic, the overhead of kprobes-based LKST estimated by the results will be more than 18%. But djprobe-based one will be almost same as LKST, about 3%.

7. Comparing Functionality

I compared these methods in the sight of functionality in List 15.

<table>
<thead>
<tr>
<th>List 15 Probes’ function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installable point</td>
</tr>
<tr>
<td>kprobe: almost anywhere</td>
</tr>
<tr>
<td>jprobe: function entry</td>
</tr>
<tr>
<td>djprobe: function entry</td>
</tr>
<tr>
<td>Probe handler</td>
</tr>
<tr>
<td>kprobe: pre-exec, post-exec, break, fault</td>
</tr>
<tr>
<td>jprobe: pre-exec</td>
</tr>
<tr>
<td>djprobe: pre-exec</td>
</tr>
<tr>
<td>Arguments of probe handler</td>
</tr>
<tr>
<td>kprobe: struct kprobe and struct pt_regs</td>
</tr>
<tr>
<td>jprobe: function arguments</td>
</tr>
<tr>
<td>djprobe: struct djprobe and struct pt_regs (*)</td>
</tr>
</tbody>
</table>

(*) some members(segment registers, eip, and old_eax) can not be accessed.

Installable points of djprobe are limited to the function entry point because djprobe has to replace the instructions at least 5 bytes.

And the 2nd argument of djprobe is a pointer of pt_regs, but it contains only general purpose registers (e*x, e*i, ebp), the flag register and the stack pointer (esp). Currently, segment registers, instruction pointer and old_eax are not saved. Because djprobe handler is jumped from kernel space without fail. It changes no segment.
8. Conclusion

I developed a SMP-safe direct jump probe called djprobe. And I measured the overheads of three probes, kprobe, jprobe and djprobe on three kinds of IA32 processors, Pentium4, Pentium M and Athlon64. In those results, djprobe was faster than other probes on each processor.

And I also measured and compared the overhead of probing and recording by using each probe method and LKST. From those results, it could be estimated that the overhead of djprobe-based LKST will be almost same as current LKST.

From the functionality, djprobe has some limitations, but also it has the major functions which a “flight recorder” system (a.k.a. kernel state tracer) needs.

I think that the djprobe is useful to insert probes into many major (and fixed) function points in the kernel, and kprobes can be used to insert probes into other complementary (and dynamic) function points. I think that we can make the overheads of a “flight recorder” system small by doing that.

Trademarks

Pentium is a trademark of Intel Corporation.

AMD Athlon and combinations thereof are trademarks of Advanced Micro Devices, Inc.

Linux is a trademark of Linus Torvalds.

Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.