Using DTrace API to write my own consumer

Petr Škovroň

Solaris RPE
Sun Microsystems Czech
petr.skovron@sun.com

Jun 27, 2008

Contents

Problem

Problem: Count opens and closes on a given device.

Problem

Problem: Count opens and closes on a given device.

Distinguish userland-originated from kernel-originated calls.

Problem

Problem: Count opens and closes on a given device.

Distinguish userland-originated from kernel-originated calls.

Do not crash

Problem

Problem: Count opens and closes on a given device.

Distinguish userland-originated from kernel-originated calls.

Do not crash

Make the data available to shell

Possible solutions

- Kernel driver overwriting the device's `cb_ops`

Possible solutions

- Kernel driver overwriting the device's `cb_ops`
- Kernel driver for a pseudodevice accessing the underlying physical device

Possible solutions

- Kernel driver overwriting the device's `cb_ops`
- Kernel driver for a pseudodevice accessing the underlying physical device
- DTrace script using dtrace(1)

Possible solutions

- Kernel driver overwriting the device's `cb_ops`
- Kernel driver for a pseudodevice accessing the underlying physical device
- DTrace script using dtrace(1)
- DTrace script using a customized consumer

Stability level

The C API officially private to the implementation and subject to change without notice.

Stability level

The C API officially private to the implementation and subject to change without notice.

"But as a practical matter, I don't think it's going to change much, at least for the foreseeable future—and almost certainly not in a way that breaks existing consumers"

(Bryan Cantrill)

Connect to the DTrace framework

```
int err;
dtrace_hdl_t *dh;

dh = dtrace_open(DTRACE_VERSION, 0, &err);
```

Connect to the DTrace framework

```
int err;
dtrace_hdl_t *dh;

dh = dtrace_open(DTRACE_VERSION, 0, &err);

if (dh == NULL) {
    ERROR("Cannot open dtrace library: %s\n",
        dtrace_errmsg(NULL, err));
}
```

Compile the script

```
FILE *pfile;
dtrace_prog_t *prog;

pfile = fopen(argv[1], "r");
/* Check that fopen succeeded */

prog = dtrace_program_fcompile(dh, pfile,
    DTRACE_C_CPP, 0, NULL);
/* Check that the return value is not NULL */

fclose(pfile);
/* Check that fclose succeeded */
```

Compile the script given as a string

```
char *script = "fbt::spec_open:entry \
    { @foo = count(); }";
dtrace_prog_t *prog;

prog = dtrace_program_fcompile(dh, script,
    DTRACE_PROBESPEC_NAME, DTRACE_C_ZDEFS,
    0, NULL);
/* Check that the return value is not NULL */
```

Set C preprocessor #defines

```
int arch = sysinfo(SI_ARCHITECTURE_64, buf, 128);
char *archsym = (arch < 0) ? "ARCH32" : "ARCH64";

dtrace_setopt(dh, "define", archsym);
/* Check that the return value is not -1 */
```

Go

```
dtrace_proginfo_t info;

dtrace_program_exec(dh, prog, &info);
/* Check that the return value is not -1 */

dtrace_go(dh);
/* Check that the return value is 0 */
```

Pretend interest in the status

```
while (1) {
        (void) dtrace_status(dh);
        sleep(1);
}
```

If you don't do this (for example you forget or your application hangs), your script will be halted.

Reading the data

We want to read values of `ctr` defined as

```
@ctr[category, major, minor] = count();
...
```

Capturing the current state

```
dtrace_aggregate_snap(dh);
/* Check that the return value is 0 */
```

Walk the aggregate

```
typedef struct {
        int maj, min;
        int data[DOPMAX];
} dsm_buf_t;
...

dsm_buf_t buf;
buf.maj = 13; buf.min = 2;

dtrace_aggregate_walk(dh, walk, (void*)&buf);
/* Check that the return value is 0 */
```

Callback function

```
int walk(const dtrace_aggdata_t *data, void *arg)
{
    dtrace_aggdesc_t *aggdesc = data->dtada_desc;
    dtrace_recdesc_t *catrec, ..., *datarec;
    int cat, ..., count;

    catrec = &aggdesc->dtagd_rec[1]; ...
    datarec = &aggdesc->dtagd_rec[4];

    cat = *(int *)(data->dtada_data +
        catrec->dtrd_offset);  ...
    count = *(uint64_t *)(data->dtada_data +
        datarec->dtrd_offset);

    ...
```

Callback function, continued

```
int walk(const dtrace_aggdata_t *data, void *arg)
{
    int cat, maj, min, count;
    dsm_buf_t *pbuf = (dsm_buf_t *)arg;

    ...

    if (maj == pbuf->maj && min == pbuf->min) {
            pbuf->data[cat] = count;
    }

    return (DTRACE_AGGWALK_NEXT);
}
```

Open entry

```
fbt::spec_open:entry {
    self->depth += 1;

    self->dev = (*(struct vnode **)arg0) ->
        v_rdev;
    self->major[self->depth] =
        self->dev >> MINOR_BITS;
    self->minor[self->depth] =
        self->dev & ((1 << MINOR_BITS) - 1);
    self->dev = 0;
    self->layered[self->depth] =
        (arg1 & FKLYR) || (arg2 == OTYP_LYR);

    @ctr[DOPOPENALL,self->major[self->depth],
        self->minor[self->depth]] = count();
}
```

Open return

```
fbt::spec_open:return
   /(arg1==0) && (self->layered[self->depth])/ {
        @ctr[DOPOPENLYR,self->major[self->depth],
           self->minor[self->depth]] = count();
}

fbt::spec_open:return {
        self->major[self->depth] = 0;
        self->minor[self->depth] = 0;
        self->layered[self->depth] = 0;
        self->depth -= 1;
}
```

Thank you!

Summary

Introduction

Compilation and execution the DTrace script

Reading the data

DTrace script