Frankenstein’s Disk Drive

Chuck Tuffli
chuck@freebsd.org

BSDCan Conference
University of Ottawa, Ontario
May 17-18, 2019
Anyone seen an NVMe card enter read only mode and freak out with unexplained write activity that paralyzes the rest of the system?
The Disclaimer

- Discussion here is NOT A CRITIQUE
- Bugs are normal
- All handled the abuse well
Testing Drivers
static void nvme_qpair_complete_tracker(struct nvme_qpair *qpair, struct nvme_tracker *tr,
    struct nvme_completion *cpl, error_print_t print_on_error)
{
    struct nvme_request  *req;
    boolean_t            retry, error;

    req = tr->req;
    error = nvme_completion_is_error(cpl);

    if (error) {
        ??????
    }

    ...
}
The Usual

- Add “error modes” to driver
  - It can (accidentally) go boom
  - Modes are static
  - Kernel == pain

- “Mocking”
  - Can control stimulus
  - Environment is different
The Alternative

- Virtual hardware
  - bhyve emulated device
- Customized firmware
  - User defined plug-ins
bhyve NVMe™ Emulation

Kernel space

User space

“VM run”

“VM exit”

guest

NVMe driver

doorbell write

bhyve

pci_nvme.c

VM run

VM exit
NVMe 101

Queue starts here.
NVMe Queues

- Producer-consumer queue
- Host memory + head / tail registers
  - Produce to tail
  - Consume from head
  - Infer work to do if (head != tail)
- Allocate in pairs for bi-directional messaging
  - Submission Queue : host to device
  - Completion Queue : device to host
Queue Creation

<table>
<thead>
<tr>
<th>Completion Queue (CQ)</th>
<th>Submission Queue (SQ)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>“Create CQ”</strong></td>
<td><strong>“Create SQ”</strong></td>
</tr>
<tr>
<td></td>
<td><strong>DW[0]</strong></td>
</tr>
<tr>
<td><strong>Host Memory Address</strong></td>
<td><strong>DW[0]</strong></td>
</tr>
<tr>
<td><strong>Size</strong></td>
<td><strong>DW[6-7]</strong></td>
</tr>
<tr>
<td><strong>ID</strong></td>
<td><strong>DW[6-7]</strong></td>
</tr>
<tr>
<td><strong>Associated Interrupt</strong></td>
<td><strong>DW[10]</strong></td>
</tr>
<tr>
<td><strong>DW[10]</strong></td>
<td><strong>Size</strong></td>
</tr>
<tr>
<td><strong>DW[11]</strong></td>
<td><strong>ID</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Associated CQ</strong></td>
</tr>
<tr>
<td></td>
<td><strong>DW[11]</strong></td>
</tr>
</tbody>
</table>
Queue Creation

Completion Queue (CQ)

- "Create CQ"
- DW[0]
- Host Memory Address
- DW[6-7]
- Size | ID
- DW[10]
- Associated Interrupt
- DW[11]

Submission Queue (SQ)

- "Create SQ"
- DW[0]
- Host Memory Address
- DW[6-7]
- Size | ID
- DW[10]
- Associated CQ
- DW[11]

Register 0x10xx
Head Pointer
Queue Creation

Completion Queue (CQ)

```
<table>
<thead>
<tr>
<th>“Create CQ”</th>
<th>DW[0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host Memory Address</td>
<td>DW[6-7]</td>
</tr>
<tr>
<td>Size</td>
<td>ID</td>
</tr>
<tr>
<td>Associated Interrupt</td>
<td>DW[11]</td>
</tr>
</tbody>
</table>
```

Submission Queue (SQ)

```
<table>
<thead>
<tr>
<th>“Create SQ”</th>
<th>DW[0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host Memory Address</td>
<td>DW[6-7]</td>
</tr>
<tr>
<td>Size</td>
<td>ID</td>
</tr>
<tr>
<td>Associated CQ</td>
<td>DW[11]</td>
</tr>
</tbody>
</table>
```

Register

- Head Pointer: 0x10xx
- Tail Pointer: 0x10xx
Command / Completion

Host

SQ

cQ

Drive

tail  head
head  tail
Command / Completion

Host

SQ

tail
head

head
tail

cQ

Drive
Command / Completion

Host

SQ

tail

head

head

tail

Drive

cQ
Command / Completion

Host

sq

head

tail

Drive

cq

head

tail
NVMe “Pipeline”

- Classic RISC pipeline
  - Instruction fetch
  - Instruction decode
  - Execute
  - Memory access
  - Writeback

- NVMe controller processing
  - SQ entry fetch
  - Operation code decode
  - Execute operation
  - Writeback CQ entry

- Plug-in access at each “stage”
The Plug-in

- Approach similar to DTrace’s SDT provider
- Plug-in provided to bhyve via shared library
- Tap name tuple

"nvme:admin:decode"

Module  Component  Stage
Plug-in API

- bhyve expects setup() and teardown()
- Provides way to attach / detach taps

```c
int plugin_tap_attach(const char *name, void *cb);
int plugin_tap_detach(const char *name);

#include "plugin.h"

static DECL_NPLUGIN_ADMIN_DECODE(admin_cmd);

int setup(void)
{
    ...
    plugin_tap_attach("nvme:admin:decode", admin_cmd);
    ...
}
```
plugin_tap_t nvme_plugin_admin_decode[1] = {
    [0] = {
        .name = "nvme:admin:decode",
        .cb = NULL,
        .enable = false
    }
};
PLUGIN_TAP_SET(nvme_plugin_admin_decode);

static void
pci_nvme_handle_admin_cmd(struct pci_nvme_softc* sc, uint64_t value)
{
    ...
    while (sqhead != atomic_load_acq_short(&sq->tail)) {
        cmd = &(sq->qbase)[sqhead];
        compl.status = 0;

        if (nvme_plugin_admin_decode->enable) {
            nvme_plugin_admin_decode_callback_t cb = nvme_plugin_admin_decode->cb;
            cb(NVME_BDF(), cmd, &compl, 0);
        }

        switch (cmd->opc) {
            case NVME_OPC_DELETE_IO_SQ:
                ...
        }
    }
}
Read-only Failure
Sep 11 09:48:49 xxxxxxxxx nvme1: async event occurred (log page id=0x2)
Sep 11 09:48:49 xxxxxxxxx nvme1: async event occurred (log page id=0x2)
Sep 11 09:48:49 xxxxxxxxx nvme1: media placed in read only mode
Sep 11 09:48:49 xxxxxxxxx nvme1: async event occurred (log page id=0x2)
Sep 11 09:48:49 xxxxxxxxx nvme1: media placed in read only mode
Sep 11 09:48:49 xxxxxxxxx nvme1: media placed in read only mode
static DECLPLUGIN_IO_DECODE(io_ro_fail);

int setup(void)
{
    plugin_tap_attach("nvme:io:decode", io_ro_fail);
    return (0);
}

int teardown(void)
{
    return (0);
}

static size_t n_writes = 5000; /* “Fail” after 5,000 Write commands */

static int io_ro_fail(uint32_t bdf, struct nvme_command *cmd, struct nvme_completion *cmp, uint32_t sqid)
{
    if (cmd->opc == NVME_OPC_WRITE) {
        if (n_writes)
            n_writes--;
        else {
            NVME_STATUS_SET(cmp->status, NVME_SCT_GENERIC, NVME_SC_SUCCESS);
            return (1);
        }
    }
    return (0);
}
static DECL_NPLUGIN_IO_DECODE(io_ro_fail);

int setup(void)
{
    plugin Tap attach("nvme:io:decode", io_ro_fail);
    return (0);
}

int teardown(void)
{
    return (0);
}

static size_t n_writes = 5000; /* Fail after 5,000 Write commands */

static int io_ro_fail(uint32_t bdf, struct nvme_command *cmd, struct nvme_completion *cmp, uint32_t sqid)
{
    if (cmd->opc == NVME_OPC_WRITE) {
        if (n_writes) n_writes--;
        else {
            NVME_STATUS_SET(cmp->status,
                            NVME_SCT_COMMAND_SPECIFIC, NVME_SC_ATTEMPTED_WRITE_TO_RO_PAGE);
            return (1);
        }
    }
    return (0);
}
Yay, errors!

```
root@freebsd:~ # nvme0: WRITE sqid:2 cid:126 nsid:1 lba:128120 len:16
nvme0: WRITE TO RO PAGE (01/82) sqid:2 cid:126 cdw0:0
nvme0: WRITE sqid:1 cid:126 nsid:1 lba:528 len:16
nvme0: WRITE TO RO PAGE (01/82) sqid:1 cid:126 cdw0:0
```

```
root@freebsd:~ # zpool status -v
  pool: sparks
  state: DEGRADED
  status: One or more devices has experienced an error resulting in data corruption. Applications may be affected.
  action: Restore the file in question if possible. Otherwise restore the entire pool from backup.
  see: http://illumos.org/msg/ZFS-8000-8A
  scan: none requested
  config:

    NAME        STATE     READ WRITE CKSUM
    sparks      DEGRADED     0     0     0
    ada1        ONLINE       0     0     0
    logs
        nvd0      FAULTED     0     4     0  too many errors

errors: Permanent errors have been detected in the following files:

    sparks/zv1:<0x0>
```
static DECL_NPLUGIN_IO_DECODE(io_ro_fail);

int setup(void)
{
    plugin_tap_attach("nvme:io:decode", io_ro_fail);
    srand(time(NULL));
    return (0);
}

int teardown(void)
{
    return (0);
}

static bool is_readonly = false;

static int io_ro_fail(uint32_t bdf, struct nvme_command *cmd, struct nvme_completion *cmp, uint32_t sqid)
{
    if (cmd->opc == NVME_OPC_WRITE) {
        if (is_readonly) {
            NVME_STATUS_SET(cmp->status, NVME_SCT_COMMAND_SPECIFIC, NVME_SC_ATTEMPTED_WRITE_TO_RO_PAGE);
            return (1);
        }
        else
            is_readonly = (random() % 100) > 42;
    }
    return (0);
}
#define MAX_WRITES 5000 /* Fail after 5,000 Write commands */

static int
io_ro_fail(uint32_t bdf, struct nvme_command *cmd, struct nvme_completion *cmp, uint32_t sqid)
{
    int rc = 0;

    if (cmd->opc == NVME_OPC_WRITE) {
        if (n_writes >= MAX_WRITES) {
            if (n_writes > (MAX_WRITES + 10)) n_writes = 0;

            NVME_STATUS_SET(cmp->status,
                NVME_SCT_COMMAND_SPECIFIC,
                NVME_SC_ATTEMPTED_WRITE_TO_RO_PAGE);
            rc = 1;
        }
        n_writes++;
    }

    return (rc);
}
1 + 1 = 
2 + 2 =
Sum Command

```
plugin_tap_attach("nvme:admin:writeback", sum);
...
static int sum(uint32_t bdf, struct nvme_completion *cmp, uint32_t sqid)
{
    struct nvme_command *cmd = find_cmd(sqid, cmp->cid);

    if (cmd == NULL) {
        printf("%s: cache miss for CID=%04x\n", __func__, cmp->cid);
        return (0);
    }

    if (cmd->opc == 0x80) {
        uint32_t sum;

        sum = cmd->cdw10 + cmd->cdw11;
        cmp->cdw0 = sum;

        NVME_STATUS_SET(cmp->status, NVME_SCT_GENERIC, NVME_SC_SUCCESS);
        return (1);
    }

    return (0);
}
```

```
root@freebsd:~ # ./add /dev/nvme0 10 5
10 + 5 = 15
root@freebsd:~ #
```
tbuf = calloc(MAX_E, sizeof(struct trc_ent));

out = fopen("/tmp/ptrace", "a");

plugin_tap_attach("pci:regread:writeback", cfgrd);
plugin_tap_attach("pci:regwrite:decode", cfgwr);
plugin_tap_attach("pci:msix:writeback", msix);
plugin_tap_attach("nvme:regread:writeback", regrd);
plugin_tap_attach("nvme:regwrite:decode", regwr);
plugin_tap_attach("nvme:msixread:writeback", msixrd);
plugin_tap_attach("nvme:msixwrite:decode", msixwr);
plugin_tap_attach("nvme:admin:decode", admin_cmd);
plugin_tap_attach("nvme:admin:writeback", cpl);
plugin_tap_attach("nvme:io:decode", io_cmd);
plugin_tap_attach("nvme:io:writeback", cpl);
OpenBSD Boot

Index  Time          PCI          Event

[ 586] 1556804537:200909267 00:06.0 REG 0014 <- 00460001
[ 587] 1556804537:200915971 00:06.0 REG 001c -> 00000001
[ 588] 1556804537:200925803 00:06.0 REG 1000 <- 00000001
[ 589] 1556804537:200926382 00:06.0 ADM SQ=0 CID=0000 OPC=Identify NSID=0 prp1=bfb24000 prp2=0 CDW10=000000001
[ 590] 1556804537:200928002 00:06.0 CPL SQ=0 CID=0000 P=1 STS=0x0000 CDW0=0xddebebac0
[ 591] 1556804537:200935816 00:06.0 REG 1004 <- 00000001
[ 592] 1556804537:202818938 00:06.0 REG 1000 <- 00000002
[ 593] 1556804537:202819148 00:06.0 ADM SQ=0 CID=0000 OPC=Create IO SQ NSID=0 prp1=bfb25000 prp2=0 CDW10=007f0001
[ 594] 1556804537:202819551 00:06.0 CPL SQ=0 CID=0000 P=1 STS=0x0000 CDW0=0xddebebac0
[ 595] 1556804537:202829033 00:06.0 REG 1004 <- 00000002
[ 596] 1556804537:202836171 00:06.0 REG 1000 <- 00000003
[ 597] 1556804537:202836260 00:06.0 ADM SQ=0 CID=0000 OPC=Create IO SQ NSID=0 prp1=bfbfc000 prp2=0 CDW10=007f0001
[ 598] 1556804537:202836774 00:06.0 CPL SQ=0 CID=0000 P=1 STS=0x0000 CDW0=0xddebebac0
[ 599] 1556804537:202843934 00:06.0 REG 1004 <- 00000003
[ 600] 1556804537:202854092 00:06.0 REG 0010 <- 00000001
[ 601] 1556804537:204672931 00:06.0 REG 1000 <- 00000004
[ 602] 1556804537:204673170 00:06.0 ADM SQ=0 CID=0000 OPC=Identify NSID=0x1 prp1=bfb20000 prp2=0 CDW10=00000000
[ 603] 1556804537:204674262 00:06.0 CPL SQ=0 CID=0000 P=1 STS=0x0000 CDW0=0xddebebac0
[ 604] 1556804537:204683593 00:06.0 REG 1004 <- 00000004
[ 605] 1556804538:517281174 00:06.0 REG 1008 <- 00000001
[ 606] 1556804538:517281875 00:06.0 IO SQ=1 CID=0000 OPC=Read NSID=0x1 LBA=0 prp1=bff7000 prp2=0
[ 607] 1556804538:517388326 00:06.0 CPL SQ=1 CID=0000 P=1 STS=0x0000 CDW0=0
The Wireshark

### dump_nvmeb.pcap

<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Source</th>
<th>Protocol</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>444</td>
<td>2.188943</td>
<td>00:00:05.0</td>
<td>NVMe/b</td>
<td>Mem Reg Write</td>
</tr>
<tr>
<td>445</td>
<td>2.188944</td>
<td>00:00:05.0</td>
<td>NVMe/b</td>
<td>Command</td>
</tr>
<tr>
<td>447</td>
<td>2.188981</td>
<td>00:00:05.0</td>
<td>NVMe/b</td>
<td>Mem Reg Write</td>
</tr>
<tr>
<td>448</td>
<td>2.189015</td>
<td>00:00:05.0</td>
<td>NVMe/b</td>
<td>Command</td>
</tr>
<tr>
<td>449</td>
<td>2.189016</td>
<td>00:00:05.0</td>
<td>NVMe/b</td>
<td>Mem Reg Write</td>
</tr>
<tr>
<td>450</td>
<td>2.189025</td>
<td>00:00:05.0</td>
<td>NVMe/b</td>
<td>Command</td>
</tr>
<tr>
<td>451</td>
<td>2.189043</td>
<td>00:00:05.0</td>
<td>NVMe/b</td>
<td>Mem Reg Write</td>
</tr>
<tr>
<td>452</td>
<td>2.195920</td>
<td>00:00:05.0</td>
<td>NVMe/b</td>
<td>Command</td>
</tr>
<tr>
<td>453</td>
<td>2.195922</td>
<td>00:00:05.0</td>
<td>NVMe/b</td>
<td>Mem Reg Write</td>
</tr>
<tr>
<td>454</td>
<td>2.195931</td>
<td>00:00:05.0</td>
<td>NVMe/b</td>
<td>Command</td>
</tr>
<tr>
<td>455</td>
<td>2.195951</td>
<td>00:00:05.0</td>
<td>NVMe/b</td>
<td>Mem Reg Write</td>
</tr>
<tr>
<td>456</td>
<td>2.195971</td>
<td>00:00:05.0</td>
<td>NVMe/b</td>
<td>Command</td>
</tr>
<tr>
<td>457</td>
<td>2.195971</td>
<td>00:00:05.0</td>
<td>NVMe/b</td>
<td>Mem Reg Write</td>
</tr>
<tr>
<td>458</td>
<td>2.195978</td>
<td>00:00:05.0</td>
<td>NVMe/b</td>
<td>Command</td>
</tr>
<tr>
<td>459</td>
<td>2.195990</td>
<td>00:00:05.0</td>
<td>NVMe/b</td>
<td>Mem Reg Write</td>
</tr>
</tbody>
</table>

**Frame 460:** 32 bytes on wire (256 bits), 32 bytes captured (256 bits)

- NVMe/bhyve Record Type: Mem Reg (1)
- NVMe/bhyve PCI Domain: 0
- NVMe/bhyve PCI Bus: 0
- NVMe/bhyve PCI Device: 5
- NVMe/bhyve PCI Function: 0
- NVMe/bhyve Register Length: 4
- NVMe/bhyve Register Address: 0x00000000000001010 SQ 2 Tail Doorbell
- NVMe/bhyve Register Value: 0x000000000000000021
The Wireshark
The Wireshark
The Wireshark
The Future

• All the bugs!
• Access data too
• Allow asynchronous command / completion
• “Events”
• Safety
• <Your idea goes here>
Questions?

chuck@FreeBSD.org | people.freebsd.org/~chuck/doc/frankendrive

Slides and notes in directory