Flipping Bits: Memory Errors in the Machine

Taylor R Campbell
riastradh@NetBSD.org

EuroBSDCon 2024
Dublin, Ireland
September 21, 2024

Flipping Bits: Memory Errors in the Máchine

https://www.NetBSD.org/gallery/presentations/riastradh/eurobsdcon2024/memerr.pdf

Flipping Bits: Memory Errors in the Machine

```
$ git status
On branch trunk
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    "e\370ternal/gpl3/gdb/dist/gdb/testsuite/gdb.linespec/
cpls.cc"

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        external/gpl3/gdb/dist/gdb/testsuite/gdb.linespec/cpls.cc
```

Flipping Bits: Memory Errors in the Machine

```
$ git status
On branch trunk
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    "e\370ternal/gpl3/gdb/dist/gdb/testsuite/gdb.linespec/
cpls.cc"

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        external/gpl3/gdb/dist/gdb/testsuite/gdb.linespec/cpls.cc
```

Memory error caught in the act

```
e\370ternal/gpl3/gdb/dist/gdb/testsuite/gdb.linespec/cpls.cc
external/gpl3/gdb/dist/gdb/testsuite/gdb.linespec/cpls.cc
```

Memory error caught in the act

```
e\370ternal/gpl3/gdb/dist/gdb/testsuite/gdb.linespec/cpls.cc
e\170ternal/gpl3/gdb/dist/gdb/testsuite/gdb.linespec/cpls.cc
```

Memory error not caught in the act

NetBSD problem reports at `https://gnats.NetBSD.org` that I filed before I realized it was bad RAM:

kern/57009 zfs crash in sa_handle_destroy ← zfs_zinactive ← zfs_netbsd_reclaim

Memory errors not caught in the act

NetBSD problem reports at `https://gnats.NetBSD.org` that I filed before I realized it was bad RAM:

kern/57009 zfs crash in sa_handle_destroy ← zfs_zinactive ← zfs_netbsd_reclaim

kern/57020 kernel diagnostic assertion `!RB_SENTINEL_P(tree->rbt_root)` failed: file . . . /sys/arch/x86/x86/pmap.c, line 2261

Memory errors not caught in the act

NetBSD problem reports at `https://gnats.NetBSD.org` that I filed before I realized it was bad RAM:

kern/57009 zfs crash in sa_handle_destroy ← zfs_zinactive ← zfs_netbsd_reclaim

kern/57020 kernel diagnostic assertion `!RB_SENTINEL_P(tree->rbt_root)` failed: file .../sys/arch/x86/x86/pmap.c, line 2261

kern/57024 panic: solaris assert: `arc_decompress(buf) == 0` (0x5 == 0x0), file: .../external/cddl/osnet/dist/uts/common/fs/zfs/arc.c, line: 4962

Memory errors not caught in the act

NetBSD problem reports at `https://gnats.NetBSD.org` that I filed before I realized it was bad RAM:

kern/57009 zfs crash in sa_handle_destroy ← zfs_zinactive ← zfs_netbsd_reclaim

kern/57020 kernel diagnostic assertion `!RB_SENTINEL_P(tree->rbt_root)` failed: file .../sys/arch/x86/x86/pmap.c, line 2261

kern/57024 panic: solaris assert: `arc_decompress(buf) == 0 (0x5 == 0x0)`, file: .../external/cddl/osnet/dist/uts/common/fs/zfs/arc.c, line: 4962

kern/57061 null pointer dereference in zfs dnode_buf_evict_async → dnode_destroy

Coda

- Repeated ZFS scrub turned up no problems[1]
- Ran BIOS diagnostics for multiple days straight
- Narrowed it down to one of two 32 GB DIMMs
- Submitted RMA to RAM manufacturer citing BIOS tests
- Received replacements for both DIMMs in a week or two
- So far so good

[1] ZFS can handle *storage* corruption but not *memory* corruption—see Yupu Zhang, Abhishek Rajimwale, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau, 'End-to-end Data Integrity for File Systems: A ZFS Case Study', USENIX FAST 2010. `https://www.usenix.org/conference/fast-10/end-end-data-integrity-file-systems-zfs-case-study`

Coda

- Repeated ZFS scrub turned up no problems[1]
- Ran BIOS diagnostics for multiple days straight
- Narrowed it down to one of two 32 GB DIMMs
- Submitted RMA to RAM manufacturer citing BIOS tests
- Received replacements for both DIMMs in a week or two
- So far so good
- . . . as far as I know

[1] ZFS can handle *storage* corruption but not *memory* corruption—see Yupu Zhang, Abhishek Rajimwale, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau, 'End-to-end Data Integrity for File Systems: A ZFS Case Study', USENIX FAST 2010. `https://www.usenix.org/conference/fast-10/end-end-data-integrity-file-systems-zfs-case-study`

Acronym soup of memory errors

- ECC: Error-correcting codes
- SECDED: Single error correction, double error detection
- EDAC: Error detection and correction
- IID: Independent and identically distributed

Error detection example: parity bit

- Data bits $d_1 d_2 \ldots d_n$ have parity bit $p := d_1 \oplus d_2 \oplus \cdots \oplus d_n$ appended
- Flipping bit d_k to $d'_k = d_k \oplus 1$ gives

$$p' := d_1 \oplus d_2 \oplus \cdots \oplus (d_k \oplus 1) \oplus \cdots \oplus d_n = p \oplus 1$$

- Data word is corrupt if $p' \neq p$

Error correction exampìe: Hamming (7,4) SECDED code

- Four-bit data words $d_1 d_2 d_3 d_4$ encoded as seven-bit code words with three parity bits $p_1 p_2 p_3$
- $p_1 := d_1 \oplus d_2 \oplus d_4$
- $p_2 := d_1 \oplus d_3 \oplus d_4$
- $p_3 := d_2 \oplus d_3 \oplus d_4$
- If p_2 is right but p_1 and p_3 are wrong, bit d_2 was probably flipped—correctable
- If p_2 and p_3 are right but p_1 is wrong, at least two bits must have been flipped, but we don't know which—detected but not correctable

Many other examples in practice for 64-bit RAM words or larger units: Hamming codes, BCH codes, Chipkill, . . . (No Galois theory in this talk.)

EDAC threat model: IID bit flips

- EDAC is *not* security against intelligent adversary
- Assumption: EDAC adversary flips each bit independently with equal probability of flipping any bit—IID
  - Fancier assumptions: one of four chips may fail altogether—chipkill
- Non-assumption: Cryptography adversary carefully chooses which bits to flip, requires secret keys and message authentication codes to detect forgery

EDAC threat model: IID bit flips

- EDAC is *not* security against intelligent adversary
- Assumption: EDAC adversary flips each bit independently with equal probability of flipping any bit—IID
  - Fancier assumptions: one of four chips may fail altogether—chipkill
- Non-assumption: Cryptography adversary carefully chooses which bits to flip, requires secret keys and message authentication codes to detect forgery
- (. . . but there is modern cryptography based on *secret* error-correcting codes, like McEliece)

What causes memory errors?

- Cosmic rays
- α-particles
- Electromagnetic pulses
- Overheating
- Faulty electrical connections

Where errors can happen

- Hard disks and other persistent storage
- DRAM module
- Memory interconnect
- PCI interconnect
- CPU caches
- CPU registers

Error severity

- **Corrected**—No data lost
- **Uncorrectable recoverable**—Data lost, but scope of loss is known, e.g. limited to a known word or cache line or page
  - If page is unused, no problem
  - If page is used by userland process, can kill process without other adverse consequences
  - If page is used by VM guest, can terminate that VM guest but not others
- **Uncorrectable fatal**—Data lost and corrupt data may have spread arbitrarily far before detection
  - Corrupt data got copied into cache lines or registers before detection
  - Reliable recovery impossible

Error persistence

- **Soft error**—at location in memory independent of other errors, e.g. cosmic ray flipped a bit
- **Hard error**—at location of flaky memory, will probably continue to flip bits in the same place

Error reporting

- **Synchronous**—delivered by nonmaskable interrupt when CPU loads corrupted memory
- **Asynchronous**—delivered by low-priority interrupt or polling when background memory scrubber runs

Practical visibility of EFAC

How do *you* know when you got a memory error?

Practical visibility of EDAC

Preferably not like this:

```
$ git status
On branch trunk
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    "e\370ternal/gpl3/gdb/dist/gdb/testsuite/gdb.linespec/
cpls.cc"

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        external/gpl3/gdb/dist/gdb/testsuite/gdb.linespec/cpls.cc
```

Practical visibility of EDAC

Preferably more like this:

```
[  4939.045145] apei0: error source 1 reported hardware error:
severity=corrected nentries=1 status=0x12<CE,GEDE_COUNT=0x1>
[  4939.055146] apei0: error source 1 entry 0: SectionType={0xa5bc1114,
0x6f64,0x4ede,0xb8b8,{0x3e,0x83,0xed,0x7c,0x83,0xb1}} (memory error)
[  4939.075146] apei0: error source 1 entry 0: ErrorSeverity=2 (corrected)
[  4939.075146] apei0: error source 1 entry 0: Revision=0x201
[  4939.085146] apei0: error source 1 entry 0: Flags=0x1<PRIMARY>
[  4939.085146] apei0: error source 1 entry 0: FruText=CorrectedErr
[  4939.095147] apei0: error source 1 entry 0: MemoryErrorType=8
(PARITY_ERROR)
```

Practical visibility of EDAC

- DDR memory controller is hardware device with registers
- Documented only under super-secret vendor NDA

Real-world prevalence of memory errors

Vendors insist uncorrectable error probability with scrubber is so negligible, why even bother checking?

Real-world p2evalence of memory errors

[W]e observe DRAM error rates that are orders of magnitude higher than previously reported, with 25,000–70,000 errors per billion device hours per Mb and more than 8% of DIMMs affected by errors per year. We provide strong evidence that memory errors are dominated by hard errors, rather than soft errors, which previous work suspects to be the dominant error mode.[2]

[2] Bianca Schroeder, Eduardo Pinheiro, and Wolf-Dietrich Weber, 'DRAM errors in the wild: a large-scale field study', Communications of the ACM 54(2), 2011, pp. 100–107, `https://dl.acm.org/doi/10.1145/1897816.1897844`.

Support for EDAC

- Not all hardware with 'ECC RAM' does anything with it!
- Intel Xeon server-class CPUs support ECC RAM, but not desktop/mobile-class CPUs
- ∼All AMD CPUs can support ECC RAM
- . . . but some motherboards that physically accept ECC RAM just don't do anything with it!

Must confirm RAM, motherboard, CPU, *and* firmware support EDAC!

Testing EDAC

How do you know what will happen when you get a memory error?

Testing EDAC

- Send a cosmic ray at your RAM

Testing EDAC

- Send a cosmic ray at your RAM
  - Problem: Suns are hard to steer and aim

Testing EDAC

- Send a cosmic ray at your RAM
  - Problem: Suns are hard to steer and aim
- Hold an α-emitter up to your RAM

Testing EDAC

- Send a cosmic ray at your RAM
  - Problem: Suns are hard to steer and aim
- Hold an α-emitter up to your RAM
  - Problem: Polonium-210 is difficult to procure after 2006

Testing EDAC

- Send a cosmic ray at your RAM
  - Problem: Suns are hard to steer and aim
- Hold an α-emitter up to your RAM
  - Problem: Polonium-210 is difficult to procure after 2006
- Electromagnetic pulse (EMP) gun

Testing EDAC

- Send a cosmic ray at your RAM
  - Problem: Suns are hard to steer and aim
- Hold an α-emitter up to your RAM
  - Problem: Polonium-210 is difficult to procure after 2006
- Electromagnetic pulse (EMP) gun
  - Problem: $4,125 and doesn't ship before EuroBSDCon

Testing EDAC

- Send a cosmic ray at your RAM
  - Problem: Suns are hard to steer and aim
- Hold an α-emitter up to your RAM
  - Problem: Polonium-210 is difficult to procure after 2006
- Electromagnetic pulse (EMP) gun
  - Problem: $4,125 and doesn't ship before EuroBSDCon
- Rowhammer

Testing EDAC

- Send a cosmic ray at your RAM
  - Problem: Suns are hard to steer and aim
- Hold an α-emitter up to your RAM
  - Problem: Polonium-210 is difficult to procure after 2006
- Electromagnetic pulse (EMP) gun
  - Problem: $4,125 and doesn't ship before EuroBSDCon
- Rowhammer
  - Problem: Requires doing science on your RAM to apply a rowhammer attack

Testing EDAC

- Send a cosmic ray at your RAM
  - Problem: Suns are hard to steer and aim
- Hold an α-emitter up to your RAM
  - Problem: Polonium-210 is difficult to procure after 2006
- Electromagnetic pulse (EMP) gun
  - Problem: $4,125 and doesn't ship before EuroBSDCon
- Rowhammer
  - Problem: Requires doing science on your RAM to apply a rowhammer attack
- Error injection

Testing EDAC

- Send a cosmic ray at your RAM
  - Problem: Suns are hard to steer and aim
- Hold an α-emitter up to your RAM
  - Problem: Polonium-210 is difficult to procure after 2006
- Electromagnetic pulse (EMP) gun
  - Problem: $4,125 and doesn't ship before EuroBSDCon
- Rowhammer
  - Problem: Requires doing science on your RAM to apply a rowhammer attack
- Error injection
  - Problem: Not all hardware supports it

Testing EDAC

- Send a cosmic ray at your RAM
  - Problem: Suns are hard to steer and aim
- Hold an α-emitter up to your RAM
  - Problem: Polonium-210 is difficult to procure after 2006
- Electromagnetic pulse (EMP) gun
  - Problem: $4,125 and doesn't ship before EuroBSDCon
- Rowhammer
  - Problem: Requires doing science on your RAM to apply a rowhammer attack
- Error injection
  - Problem: Not all hardware supports it
  - . . . but if your hardware does, this is the easiest option

Testing EDAC with error injection

Error injection in principle:

- Write to hardware register in memory controller
- Error report comes flying out as if real error
- Engineer confirms it works, moves on to other task

Testing EDAC with error inkection

Error injection in practice:

- Wait, how can I control which memory location gets corrupted—make sure it's in an unused test page?

Testing EDAC with error inkection

Error injection in practice:

- Wait, how can I control which memory location gets corrupted—make sure it's in an unused test page?
  - Vendor: You can't. Wherever is the next memory transaction in your highly parallel multicore system with DMA engines doing I/O!

Testing EDAC with error inkection

Error injection in practice:

- Wait, how can I control which memory location gets corrupted—make sure it's in an unused test page?
  - Vendor: You can't. Wherever is the next memory transaction in your highly parallel multicore system with DMA engines doing I/O!
- Wait, injecting a *correctable* error actually corrupted memory?

Testing EDAC with error inkection

Error injection in practice:

- Wait, how can I control which memory location gets corrupted—make sure it's in an unused test page?
  - Vendor: You can't. Wherever is the next memory transaction in your highly parallel multicore system with DMA engines doing I/O!
- Wait, injecting a *correctable* error actually corrupted memory?
  - panic: diagnostic assertion: "critical data structure hopelessly destroyed" failed at kernel.c line 4

Testing EDAC with error inkection

Error injection in practice:

- Wait, how can I control which memory location gets corrupted—make sure it's in an unused test page?
  - Vendor: You can't. Wherever is the next memory transaction in your highly parallel multicore system with DMA engines doing I/O!
- Wait, injecting a *correctable* error actually corrupted memory?
  - panic: diagnostic assertion: "critical data structure hopelessly destroyed" failed at kernel.c line 4
  - Workaround: Inject error in parity bits, not data bits, if you can specify error pattern or syndrome

Testing EDAC with error inkection

Error injection in practice:

- Wait, how can I control which memory location gets corrupted—make sure it's in an unused test page?
  - Vendor: You can't. Wherever is the next memory transaction in your highly parallel multicore system with DMA engines doing I/O!
- Wait, injecting a *correctable* error actually corrupted memory?
  - panic: diagnostic assertion: "critical data structure hopelessly destroyed" failed at kernel.c line 4
  - Workaround: Inject error in parity bits, not data bits, if you can specify error pattern or syndrome
- Wait, why isn't the machine responding at serial console? . . . or ILO? . . . or remote reset? . . . or. . .

Testing EDAC with error inkection

Error injection in practice:

- Wait, how can I control which memory location gets corrupted—make sure it's in an unused test page?
  - Vendor: You can't. Wherever is the next memory transaction in your highly parallel multicore system with DMA engines doing I/O!
- Wait, injecting a *correctable* error actually corrupted memory?
  - panic: diagnostic assertion: "critical data structure hopelessly destroyed" failed at kernel.c line 4
  - Workaround: Inject error in parity bits, not data bits, if you can specify error pattern or syndrome
- Wait, why isn't the machine responding at serial console? . . . or ILO? . . . or remote reset? . . . or. . .
  - Engineer treks across town to the data center to unplug it and plug it back in again

APEI: ACPI Platform Error Interface

APEI:[3] Standard interface in ACPI abstracting EDAC device registers—WARNING: ETLA overload

- `BERT` Boot Error Record Table
- `HEST` Hardware Error Source Table
- `EINJ` Error INJection
- `ERST` Error Record Serialization Table

Available on *some* server-class machines—check with `acpidump -dt` or similar

[3] `https://uefi.org/specs/ACPI/6.5/18_Platform_Error_Interfaces.html`

APEI: ACPI Platform Error Interface

```
apei0 at acpi0: ACPI Platform Error Interface
apei0: BERT: OemId <   AMI,AMI BERT,00000000> AslId <    ,00000000>
apei0: BERT: 0x14 bytes at 0x7f340c98
apei0: BERT: no boot errors recorded
apei0: EINJ: OemId <   AMI,AMI EINJ,00000000> AslId <    ,00000000>
apei0: EINJ: can inject: 0x28<MEM_CORRECTABLE,MEM_FATAL>
apei0: ERST: OemId < AMIER,AMI ERST,00000000> AslId <    ,00000000>
apei0: ERST: 0 records in error log 8192 bytes @ 0x7f248050 attr=0
apei0: HEST: OemId <   AMI,AMI HEST,00000000> AslId <    ,00000000>
apei0: HEST: 2 hardware error sources
```

APEI BERT: Boot Error Record Table

Provides error reports early at boot, before OS is listening for active notifications

AREI HEST: Hardware Error Source Table

- Lists sources of hardware error reports
- Covers more than just memory errors—also PCI errors, CPU errors, ...
- Software can respond to non-memory hardware errors by, e.g., disabling a single faulty PCI device

APEI HEST: Hardware Error Source Table

```
Type={Generic Hardware Error Source}
SourceId=0
Enabled={YES}
Number of Records to pre-allocate=1
Max Sections per Record=1
Max Raw Data Length=157
Error Status Address=0x000000007f235018:0[64] (Memory)
HW Error Notification={
        Type={NMI}
        Length=28
        Config Write Enable={}
        Poll Interval=0 msec
        Interrupt Vector=2
        Switch To Polling Threshold Value=0
        Switch To Polling Threshold Window=0 msec
        Error Threshold Value=0
        Error Threshold Window=0 msec
}
Error Block Length=157
```

APEI HEST: Hardware Error Source Table

```
Type={Generic Hardware Error Source}
SourceId=1
Enabled={YES}
Number of Records to pre-allocate=1
Max Sections per Record=1
Max Raw Data Length=157
Error Status Address=0x000000007f2350c0:0[64] (Memory)
HW Error Notification={
        Type={POLLED}
        Length=28
        Config Write Enable={POLL_INTERVAL,POLL_THRESHOLD_VALUE,
                POLL_THRESHOLD_WINDOW,ERR_THRESHOLD_VALUE,
                ERR_THRESHOLD_WINDOW}
        Poll Interval=60000 msec
        Interrupt Vector=2
        Switch To Polling Threshold Value=0
        Switch To Polling Threshold Window=0 msec
        Error Threshold Value=0
        Error Threshold Window=0 msec
}
Error Block Length=157
```

APEI EINJ: Erros Injection table

- List of supported error injection actions

APEI EINJ: Erros Injection table

- ~~List of supported error injection actions~~
- Interpreter for programming language of supported error injection actions

APEI EINJ: Erros Injection table

- ~~List of supported error injection actions~~
- ~~Interpreter for programming language of supported error injection actions~~
- Programming language for interpreter for programming language for supported error actions

APEI EINJ: Error Injection table

Actions:

- BEGIN_INJECTION_OPERATION
- SET_ERROR_TYPE
- SET_ERROR_TYPE_WITH_ADDRESS
- EXECUTE_OPERATION
- CHECK_BUSY_STATUS
- GET_COMMAND_STATUS
- GET_TRIGGER_ERROR_ACTION_TABLE
- ...

APEI EINJ: Error Injection table

Actions:

- BEGIN_INJECTION_OPERATION
- SET_ERROR_TYPE
- SET_ERROR_TYPE_WITH_ADDRESS
- EXECUTE_OPERATION
- CHECK_BUSY_STATUS
- GET_COMMAND_STATUS
- GET_TRIGGER_ERROR_ACTION_TABLE . . . meta-action
- . . .

APEI EINJ: Error Injection table

Instructions:

- READ_REGISTER
- READ_REGISTER_VALUE (read and compare w/immediate)
- WRITE_REGISTER
- WRITE_REGISTER_VALUE (write immediate to register)
- NOOP

APEÉ EINJ: Error Injection table

Action	Instruction	Register	Value
SET_ERROR_TYPE	WRITE_REGISTER_VALUE	0x1234	0x42
SET_ERROR_TYPE	WRITE_REGISTER	0x1238	—
SET_ERROR_TYPE	READ_REGISTER	0x123c	—
EXECUTE_OPERATION	READ_REGISTER	0x1000	—
SET_ERROR_TYPE	WRITE_REGISTER	0x1240	—
GET_ERROR_STATUS	READ_REGISTER_VALUE	0x1200	0x8
⋮	⋮	⋮	⋮

APEI EINJ: Error Injection table

```
ACTION={Begin Operation}
INSTRUCTION={Write Register Value}
FLAGS={}
RegisterRegion=0x7f236f98:0[8] (Memory)
MASK=0x000000ff

ACTION={Get Trigger Table}
INSTRUCTION={Read Register}
FLAGS={}
RegisterRegion=0x000000007f236f9a:0[64] (Memory)
MASK=0xffffffffffffffff

ACTION={Set Error Type}
INSTRUCTION={Write Register}
FLAGS={}
RegisterRegion=0x7f236fa2:0[32] (Memory)
MASK=0xffffffff
```

APEI EINJ: Error Injec|ion table

To inject an error, software must execute a sequence of actions:

- BEGIN_INJECTION_OPERATION
- SET_ERROR_TYPE(0x8=<Memory Correctable>)
- EXECUTE_OPERATION
- busy-wait until CHECK_BUSY_STATUS returns completion
- check GET_COMMAND_STATUS
- execute the GET_TRIGGER_ERROR_ACTION_TABLE instructions

APEI ERST: Error Recîrd Serialization Table

- Persistent storage for small files

APEI ERST: Error Recîrd Serialization Table

- ~~Persistent storage for small files~~
- Programming language for reading and writing small files

APEI ERST: Error Recîrd Serialization Table

- ~~Persistent storage for small files~~
- ~~Programming language for reading and writing small files~~
- Interpreter for programming language for reading and writing small files

APEI ERST: Error Recîrd Serialization Table

- ~~Persistent storage for small files~~
- ~~Programming language for reading and writing small files~~
- ~~Interpreter for programming language for reading and writing small files~~
- Programming language for interpreter for programming language for reading and writing small files

APEI ERST: Error Record Serialization Table

Actions:

- BEGIN_WRITE_OPERATION
- BEGIN_READ_OPERATION
- BEGIN_CLEAR_OPERATION
- END_OPERATION
- EXECUTE_OPERATION
- SET_RECORD_OFFSET
- CHECK_BUSY_STATUS
- GET_COMMAND_STATUS
- GET_RECORD_COUNT
- ...

APEI ERST: Error Record Serialization Tab,e

Instructions:

- READ_REGISTER, READ_REGISTER_VALUE
- WRITE_REGISTER, WRITE_REGISTER_VALUE
- NOOP
- LOAD_VAR1, LOAD_VAR2
- STORE_VAR1
- ADD, SUBTRACT
- ADD_VALUE, SUBTRACT_VALUE
- STALL, STALL_WHILE_TRUE
- SKIP_NEXT_INSTRUCTION_IF_TRUE
- GOTO
- SET_SRC_ADDRESS_BASE
- SET_DST_ADDRESS_BASE
- MOVE_DATA

APEI ErST: Error Record Serialization Table

```
ACTION={Set Record Offset}
INSTRUCTION={Write Register}
FLAGS={}
RegisterRegion=0x7f24801c:0[32] (Memory)
MASK=0xffffffff

ACTION={Execute Operation}
INSTRUCTION={Write Register Value}
FLAGS={}
RegisterRegion=0xb2:0[8] (IO)
MASK=0x000000ff

ACTION={Check Busy Status}
INSTRUCTION={Read Register Value}
FLAGS={}
RegisterRegion=0x7f248020:0[8] (Memory)
MASK=0x000000ff
```

APEI ERST: Error Record Serialization Table

- Not completely bonkers: executing ERST actions is maybe less risky than running full ACPI interpreter
- Could use it to save dmesg or core dump on crash in delicate context (no memory allocation, no locks, . . .)
- Exposed in Linux as a file system ‘pstore’
- Not yet used by NetBSD—future work!

NetBSD support

NetBSD support for APEI:

- `apei(4)` driver
- Enabled in current on x86/GENERIC, soon on Arm, aimed at 11.0 or maybe even 10.1
- Supports detecting reports from common hardware error source types
- Supports crude sysctl interface to EINJ
- Code is there to interpret ERST action interpreter, but nothing uses it yet

Live demo

(This space left intentionally blank. Hard to show cosmic rays in a slide.)

Questions?

?