APEI Error INJection

EINJ provides a hardware error injection mechanism. It is very useful for debugging and testing APEI and RAS features in general.

You need to check whether your BIOS supports EINJ first. For that, look for early boot messages similar to this one:

```
ACPI: EINJ 0x000000007370A000 000150 (v01 INTEL           00000001 INTL 00000001)
```

which shows that the BIOS is exposing an EINJ table - it is the mechanism through which the injection is done.

Alternatively, look in /sys/firmware/acpi/tables for an "EINJ" file, which is a different representation of the same thing.

It doesn't necessarily mean that EINJ is not supported if those above don't exist: before you give up, go into BIOS setup to see if the BIOS has an option to enable error injection. Look for something called WHEA or similar. Often, you need to enable an ACPI5 support option prior, in order to see the APEI,EINJ,... functionality supported and exposed by the BIOS menu.

To use EINJ, make sure the following are options enabled in your kernel configuration:

```
CONFIG_DEBUG_FS
CONFIG_ACPI_APEI
CONFIG_ACPI_APEI_EINJ
```

The EINJ user interface is in <debugfs mount point>/apei/einj.

The following files belong to it:

- available_error_type

  This file shows which error types are supported:

Error Type Value	Error Description
0x00000001	Processor Correctable
0x00000002	Processor Uncorrectable non-fatal
0x00000004	Processor Uncorrectable fatal
0x00000008	Memory Correctable
0x00000010	Memory Uncorrectable non-fatal
0x00000020	Memory Uncorrectable fatal
0x00000040	PCI Express Correctable
0x00000080	PCI Express Uncorrectable non-fatal
0x00000100	PCI Express Uncorrectable fatal
0x00000200	Platform Correctable
0x00000400	Platform Uncorrectable non-fatal
0x00000800	Platform Uncorrectable fatal

  The format of the file contents are as above, except present are only the available error types.

- error_type

  Set the value of the error type being injected. Possible error types are defined in the file available_error_type above.

- error_inject

  Write any integer to this file to trigger the error injection. Make sure you have specified all necessary error parameters, i.e. this write should be the last step when injecting errors.

- flags

  Present for kernel versions 3.13 and above. Used to specify which of param{1..4} are valid and should be used by the firmware during injection. Value is a bitmask as specified in ACPI5.0 spec for the SET_ERROR_TYPE_WITH_ADDRESS data structure:

  Bit 0
  : Processor APIC field valid (see param3 below).

  Bit 1
  : Memory address and mask valid (param1 and param2).

  Bit 2
  : PCIe (seg,bus,dev,fn) valid (see param4 below).

  If set to zero, legacy behavior is mimicked where the type of injection specifies just one bit set, and param1 is multiplexed.

- param1

  This file is used to set the first error parameter value. Its effect depends on the error type specified in error_type. For example, if error type is memory related type, the param1 should be a valid physical memory address. [Unless "flag" is set - see above]

- param2

  Same use as param1 above. For example, if error type is of memory related type, then param2 should be a physical memory address mask. Linux requires page or narrower granularity, say, 0xfffffffffffff000.

- param3

  Used when the 0x1 bit is set in "flags" to specify the APIC id

- param4 Used when the 0x4 bit is set in "flags" to specify target PCIe device

- notrigger

  The error injection mechanism is a two-step process. First inject the error, then perform some actions to trigger it. Setting "notrigger" to 1 skips the trigger phase, which *may* allow the user to cause the error in some other context by a simple access to the CPU, memory location, or device that is the target of the error injection. Whether this actually works depends on what operations the BIOS actually includes in the trigger phase.

BIOS versions based on the ACPI 4.0 specification have limited options in controlling where the errors are injected. Your BIOS may support an extension (enabled with the param_extension=1 module parameter, or boot command line einj.param_extension=1). This allows the address and mask for memory injections to be specified by the param1 and param2 files in apei/einj.

BIOS versions based on the ACPI 5.0 specification have more control over the target of the injection. For processor-related errors (type 0x1, 0x2 and 0x4), you can set flags to 0x3 (param3 for bit 0, and param1 and param2 for bit 1) so that you have more information added to the error signature being injected. The actual data passed is this:

```
memory_address = param1;
memory_address_range = param2;
apicid = param3;
pcie_sbdf = param4;
```

For memory errors (type 0x8, 0x10 and 0x20) the address is set using param1 with a mask in param2 (0x0 is equivalent to all ones). For PCI express errors (type 0x40, 0x80 and 0x100) the segment, bus, device and function are specified using param1:

```
 31     24 23    16 15    11 10      8  7        0
+-------------------------------------------------+
| segment |   bus  | device | function | reserved |
+-------------------------------------------------+
```

Anyway, you get the idea, if there's doubt just take a look at the code in drivers/acpi/apei/einj.c.

An ACPI 5.0 BIOS may also allow vendor-specific errors to be injected. In this case a file named vendor will contain identifying information from the BIOS that hopefully will allow an application wishing to use the vendor-specific extension to tell that they are running on a BIOS that supports it. All vendor extensions have the 0x80000000 bit set in error_type. A file vendor_flags controls the interpretation of param1 and param2 (1 = PROCESSOR, 2 = MEMORY, 4 = PCI). See your BIOS vendor documentation for details (and expect changes to this API if vendors creativity in using this feature expands beyond our expectations).

An error injection example:

```
# cd /sys/kernel/debug/apei/einj
# cat available_error_type            # See which errors can be injected
0x00000002    Processor Uncorrectable non-fatal
0x00000008    Memory Correctable
0x00000010    Memory Uncorrectable non-fatal
# echo 0x12345000 > param1            # Set memory address for injection
# echo $((-1 << 12)) > param2         # Mask 0xfffffffffffff000 - anywhere in this page
# echo 0x8 > error_type                       # Choose correctable memory error
# echo 1 > error_inject                       # Inject now
```

You should see something like this in dmesg:

```
[22715.830801] EDAC sbridge MC3: HANDLING MCE MEMORY ERROR
[22715.834759] EDAC sbridge MC3: CPU 0: Machine Check Event: 0 Bank 7: 8c00004000010090
[22715.834759] EDAC sbridge MC3: TSC 0
[22715.834759] EDAC sbridge MC3: ADDR 12345000 EDAC sbridge MC3: MISC 144780c86
[22715.834759] EDAC sbridge MC3: PROCESSOR 0:306e7 TIME 1422553404 SOCKET 0 APIC 0
[22716.616173] EDAC MC3: 1 CE memory read error on CPU_SrcID#0_Channel#0_DIMM#0 (channel:0 slot:0 page:0x12345 offset:0x0 grain:32 syndro
```

Special notes for injection into SGX enclaves:

There may be a separate BIOS setup option to enable SGX injection.

The injection process consists of setting some special memory controller trigger that will inject the error on the next write to the target address. But the h/w prevents any software outside of an SGX enclave from accessing enclave pages (even BIOS SMM mode).

The following sequence can be used:

1. Determine physical address of enclave page
2. Use "notrigger=1" mode to inject (this will setup the injection address, but will not actually inject)
3. Enter the enclave
4. Store data to the virtual address matching physical address from step 1
5. Execute CLFLUSH for that virtual address
6. Spin delay for 250ms
7. Read from the virtual address. This will trigger the error

For more information about EINJ, please refer to ACPI specification version 4.0, section 17.5 and ACPI 5.0, section 18.6.