Restructure Pstore Ramoops

Zhang Yanmin <yanmin.zhang@intel.com>
Liu Shuo <shuo.a.liu@intel.com>

Agenda

- Embedded devices development problems
- pstore usage
- pstore implementation
- pstore ramoops current defects
- Pstore ramoops restructure and enhance
- Examples
- Notices
- Patchset

Embedded device development problems

- Mobiles/tablets, IVI, Smart terminals, robots, and so on.
- It’s hard to debug board hang issues and various elusive race conditions.
  - After board hangs and resets, all the last logs are lost;
  - UART consumes too much cpu resources if kernel prints out too many logs. It might not output the last logs.
  - Some companies have special hadware tools like lautbach to connect to the devices by JTAG. The tools are very expensive and devices usually have no JTAG at product step.

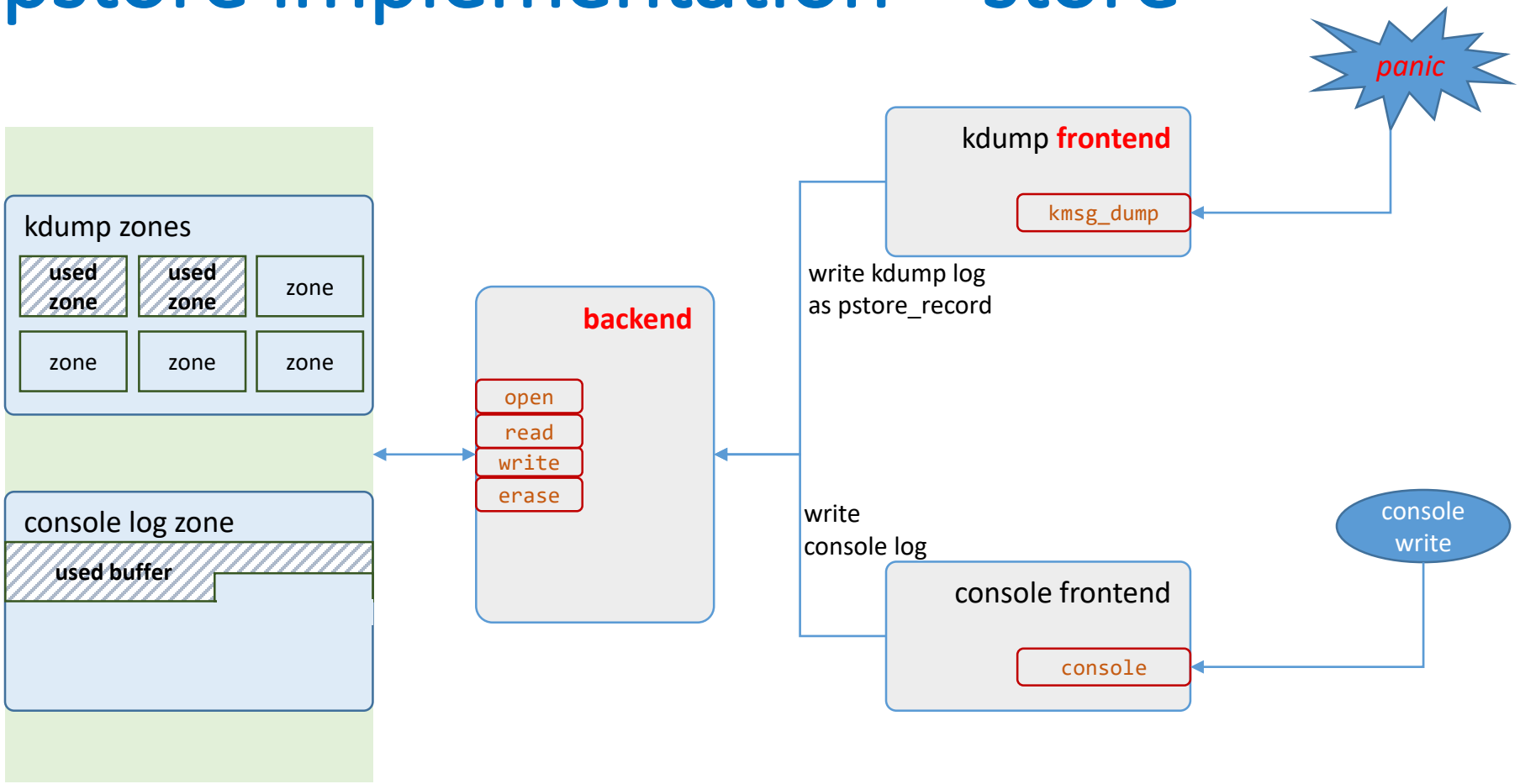
Embedded device development problems (Cont.)

- Embedded devices are powered by battery. Usualy, PMIC supply power to SOC.
- When system hangs, watchdog sends signal to controller=>PMIC to reset the device.
  - PMIC can reset board's most IP components while keeping memory on, so memory content is not lost.
  - Sometimes, power off/on transition can happen quickly, some memory contents can also be kept across the fast reset.
- Linux kernel uses pstore to save logs across the board reset. After reset, kernel picks up the old logs and save them to new memory, and then reuse old memory.

pstore usage

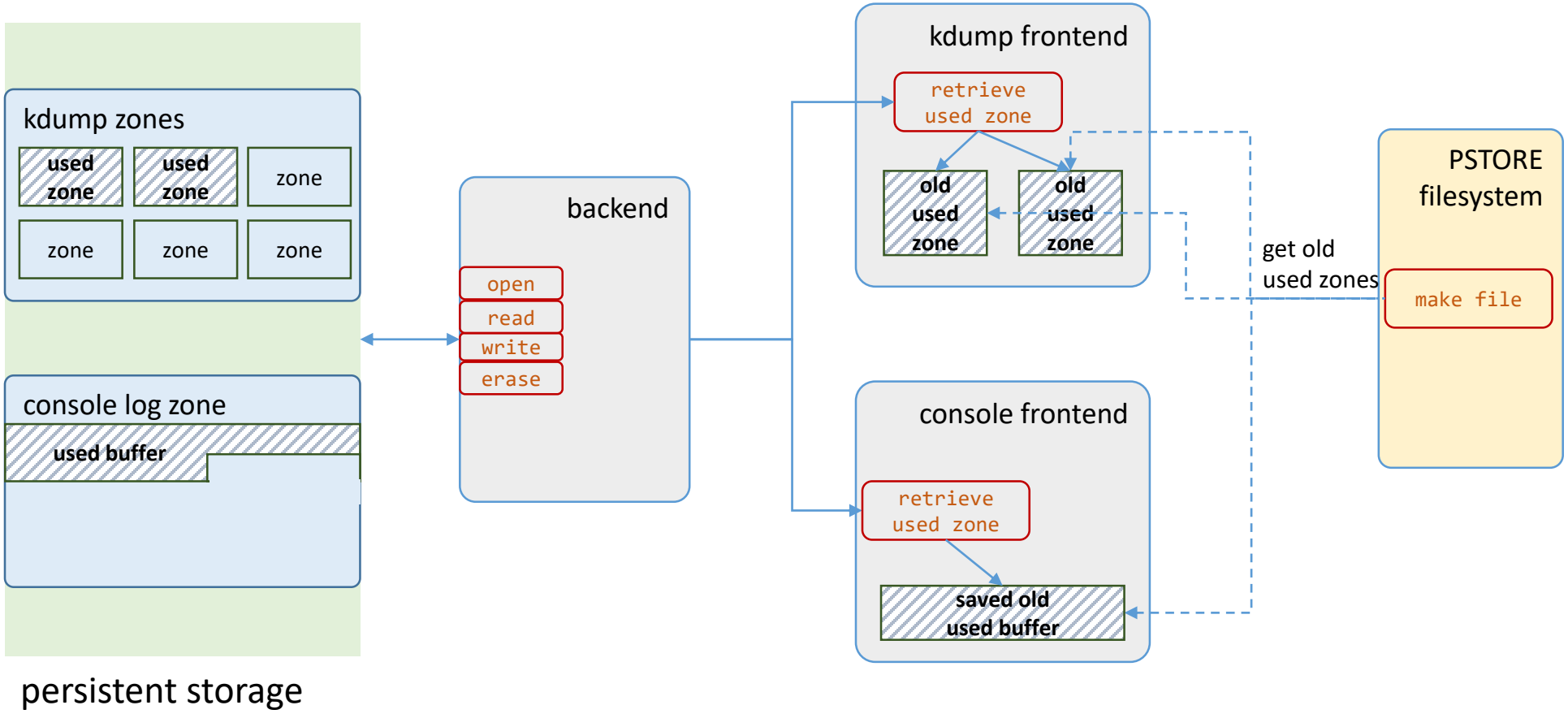
- platform level persistent storage can be used to save critical/crash log before system resets. After reset and system boots again, then be read via ‘pstore’ filesystem.
- We have ramoops now to record the panic/oops log via pstore.

pstore implementation – store



Take kdump and console frontend for example.
There are four frontends avaiable, kdump, console, ftrace and pmsg

pstore implementation – retrieve after reboot

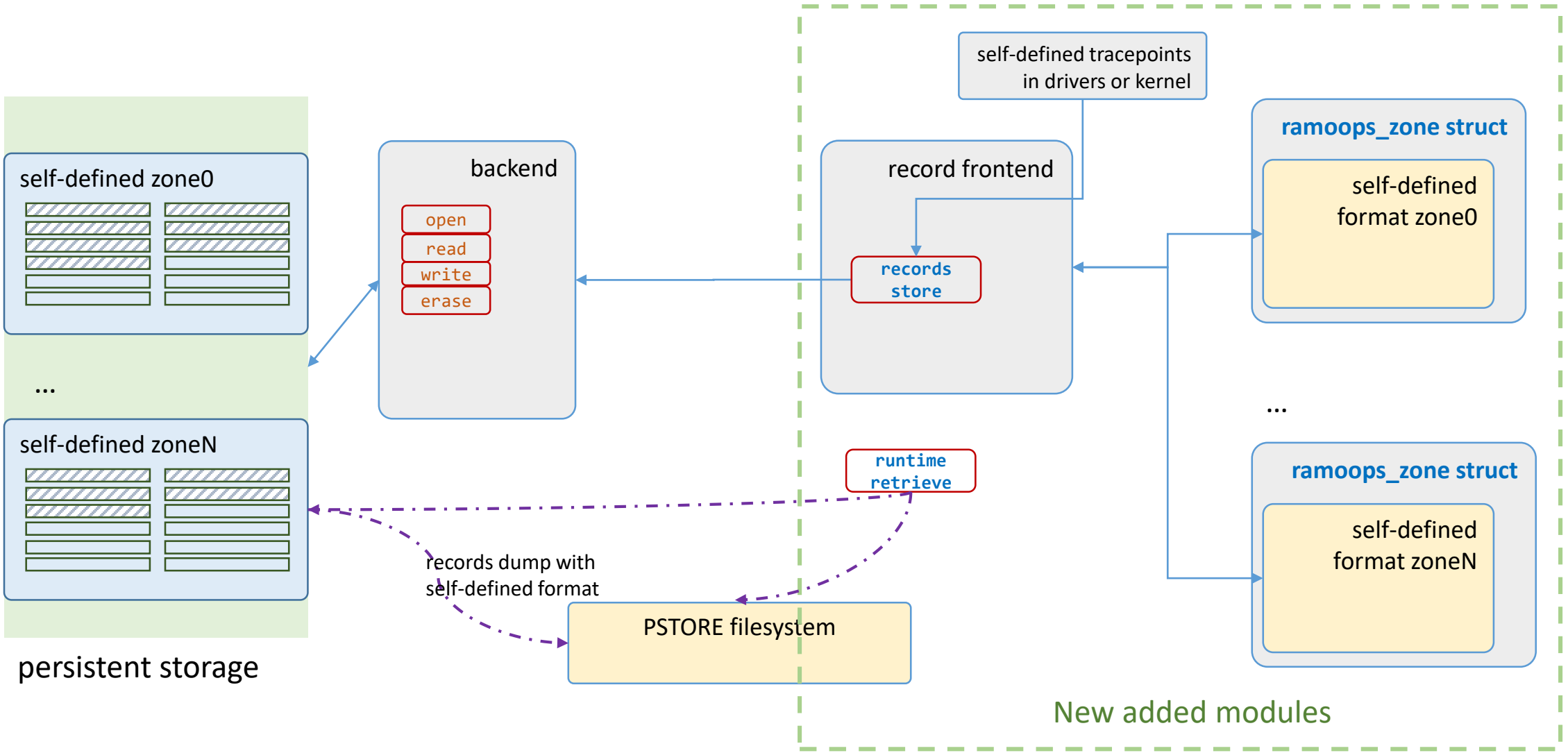


Take kdump and console frontend for example.
There are four frontends avaiable, kdump, console, ftrace and pmsg

pstore ramoops current defects

- It’s very hard to extend it to add new tracers
  - Pstore reserves a big block of memory to save logs. The memory size is calculated at booting with specific **hard-coded** methods.
  - Current ramoops **doesn’t save structured data**, such like records which consists of multiple data entries.
    - Typical record format: timestamp, function callchain, return value, …
  - Current ramoops log saving functions might have some **contentions** at SMP.
  - Current ramoops doesn’t output **current** data by pstore file system.

pstore restructure – add new frontend based on record



New frontend

- Add ramoops_zone
  - +struct ramoops_zone {
  - + **char name[16];**
  - + **unsigned long size;**
  - + struct persistent_ram_zone *prz;
  - + **int item_size;**
  - + **void (*print_record)(struct seq_file *s, void *record);**
  - + void *(*get_new_record)(struct persistent_ram_zone *prz);
  - +};
- Its format is based on record/structure which is self-defined.
- Export several APIs for new persistent ram tracers introduction, easier to make new record tracer
  - static inline void ***ramoops_get_new_record**(struct ramoops_zone *zone)
  - int pstore_print(struct seq_file *m, const char *f, ...)
- Output format is self-defined.
- Besides retrieving the last boot tracing log, it also supports **currently runtime tracing log** showing.

Typical record entries: example

- +struct *pstore_gfx_record* {
- +	long param1;
- +	long param2;
- +	long param3;
- +	int real_done;
- +	char info[20];
- +	unsigned long long time;
- +	unsigned int cpu;
- +	**void *address[IRQ_BT_NUM];**
- +};

Collect callchain from stack: x86 cpu, CONFIG_FRAME_POINTER=y

```
+static size_t backtrace_safe(void **array, size_t max_size)
+{
+       unsigned long *bp;
+       unsigned long *caller;
+       unsigned int i;
+       get_bp(bp);
+       caller = (unsigned long *) *(bp+1);
+       for (i = 0; i < max_size; i++)
+               array[i] = 0;
+       for (i = 0; i < max_size; i++) {
+               array[i] = caller;
+               bp = (unsigned long *) *bp;
+               if (!object_is_on_stack(bp) && !object_is_on_irq_stack(bp))
+                       break;
+               caller = (unsigned long *) *(bp+1);
+       }
+       return i + 1;
+}
```

Example 1: A simple tracer of cpufreq

```
+struct norm_zone_test_record {
+ unsigned long val;
+ char str[32];
+};
+
+static void print_record(struct seq_file *s, void *rec)
+{
+ struct norm_zone_test_record *record = rec;
+ pstore_print(s, "%s: %ld\n",
+ record->str, record->val);
+}
+
+DEFINE_PSTORE_RAMZONE(test_zone) = {
+ .size = 4096*1024, /*Bytes*/
+ .name = "test_zone",
+ .item_size = sizeof(struct norm_zone_test_record),
+ .print_record = print_record,
+};
+
+DEFINE_PSTORE_RAMZONE(test_zone1) = {
+ .size = 4096,
+ .name = "test_zone1",
+ .item_size = sizeof(struct norm_zone_test_record),
+ .print_record = print_record,
+};
```

Example 1: A simple tracer of cpufreq (Cont.)

```
+static void add_test_record(char *str, unsigned long val)
+{
+ struct norm_zone_test_record *record;
+ record = persistent_ram_new_record(test_zone.prz);
+ if (record) {
+ record->val = val;
+ strcpy(record->str, str);
+ }
+ record = persistent_ram_new_record(test_zone1.prz);
+ if (record) {
+ record->val = val;
+ strcpy(record->str, str);
+ }
+}
+
+static int test_cpufreq_transition(struct notifier_block *nb,
+ unsigned long event, void *data)
+{
+ add_test_record("cpufreq transition", event);
+ return 0;
+}
+
+static struct notifier_block freq_transition = {
+ .notifier_call = test_cpufreq_transition,
+};
+static int __init norm_zone_test_init(void)
+{
+ cpufreq_register_notifier(&freq_transition,
+ CPUFREQ_TRANSITION_NOTIFIER);
+ return 0;
+}
```

Example 2: trace device I/O access operations

- Many hang are caused by device I/O operations.
- Create a new tracer to track all I/O operations started from CPU
- #define build_mmio_read(name, size, type, reg, barrier) \
- static inline type name(const volatile void __iomem *addr) \
- +{ type ret; \
- + void *record; \
- + record = *add_precord(#name, (void *)addr, 0)*; \
- + asm volatile("mov" size " %1,%0":reg (ret) \
- +:"m" (*(volatile type __force *)addr) barrier); \
- + *done_precord(record, (unsigned long)ret);* \
- + return ret; }
- #define build_mmio_write(name, size, type, reg, barrier) \
- static inline void name(type val, volatile void __iomem *addr) \
- +{ void *record; \
- + record = *add_precord(#name, (void *)addr, (unsigned long)val);* \
- + asm volatile("mov" size " %0,%1": :reg (val), \
- +"m" (*(volatile type __force *)addr) barrier); \
- + *done_precord(record, 0);* \
- +}

Example 2 (Cont.)

- 32.906101 C0 Thread-1 1 readl ffffc90000076540 (null) 40000100
- intel_gpio_get addr[ffffffff813da7ee]
- _gpiod_get_raw_value addr[ffffffff813de861]
- gpiod_get_value_cansleep addr[ffffffff813de949]
- value_show addr[ffffffff813e07af]
- dev_attr_show addr[ffffffff815f6ad0]
- sysfs_kf_seq_show addr[ffffffff8125ce06]
- kernfs_seq_show addr[ffffffff8125b763]
- seq_read addr[ffffffff8120519b]
- kernfs_fop_read addr[ffffffff8125bfba]
- __vfs_read addr[ffffffff811e1998]
- vfs_read addr[ffffffff811e1ff6]
- SyS_read addr[ffffffff811e2da9]
- entry_SYSCALL_64_fastpath addr[ffffffff81aa7597]
- 32.913632 C0 kworker/u8:4 1 writel ffffc90001000f04 9100205c (null)
- __gen9_decoupled_mmio_access addr[ffffffff81575b8a]
- gen9_decoupled_read32 addr[ffffffff8157f2c9]
- intel_ring_get_active_head addr[ffffffff8156fdf4]
- i915_hangcheck_elapsed addr[ffffffff815114ef]
- process_one_work addr[ffffffff810b4dea]
- worker_thread addr[ffffffff810b5426]
- kthread addr[ffffffff810ba4dd]
- ret_from_fork addr[ffffffff81aa78ef]

Notice: function symbol address

- Need disable kaslr:
  - Add nokaslr to kernel cmdline
  - CONFIG_RANDOMIZE_BASE=n
- Modules: Load to different address at different booting
  - Keep them compile-in
  - Load them in sequence
- Some BIOS might reconfig memory to different address at every booting. Need disable this feature.

Initial patchset

- https://lkml.org/lkml/2014/5/6/20
- https://lkml.org/lkml/2014/5/6/19
- https://lkml.org/lkml/2014/5/6/23
- https://lkml.org/lkml/2014/5/6/21
- https://lkml.org/lkml/2014/5/6/22