pstore block oops/panic logger

Introduction

pstore block (pstore/blk) is an oops/panic logger that writes its logs to a block device and non-block device before the system crashes. You can get these log files by mounting pstore filesystem like:

```
mount -t pstore pstore /sys/fs/pstore
```

pstore block concepts

pstore/blk provides efficient configuration method for pstore/blk, which divides all configurations into two parts, configurations for user and configurations for driver.

Configurations for user determine how pstore/blk works, such as pmsg_size, kmsg_size and so on. All of them support both Kconfig and module parameters, but module parameters have priority over Kconfig.

Configurations for driver are all about block device and non-block device, such as total_size of block device and read/write operations.

Configurations for user

All of these configurations support both Kconfig and module parameters, but module parameters have priority over Kconfig.

Here is an example for module parameters:

```
pstore_blk.blkdev=/dev/mmcblk0p7 pstore_blk.kmsg_size=64 best_effort=y
```

The detail of each configurations may be of interest to you.

blkdev

The block device to use. Most of the time, it is a partition of block device. It's required for pstore/blk. It is also used for MTD device.

When pstore/blk is built as a module, "blkdev" accepts the following variants:

1. /dev/<disk_name> represents the device number of disk
2. /dev/<disk_name><decimal> represents the device number of partition - device number of disk plus the partition number
3. /dev/<disk_name>p<decimal> - same as the above; this form is used when disk name of partitioned disk ends with a digit.

When pstore/blk is built into the kernel, "blkdev" accepts the following variants:

1. <hex_major><hex_minor> device number in hexadecimal representation, with no leading 0x, for example b302.
2. PARTUUID=00112233-4455-6677-8899-AABBCCDDEEFF represents the unique id of a partition if the partition table provides it. The UUID may be either an EFI/GPT UUID, or refer to an MSDOS partition using the format SSSSSSSS-PP, where SSSSSSSS is a zero-filled hex representation of the 32-bit "NT disk signature", and PP is a zero-filled hex representation of the 1-based partition number.
3. PARTUUID=<UUID>/PARTNROFF=<int> to select a partition in relation to a partition with a known unique id.
4. <major>:<minor> major and minor number of the device separated by a colon.

It accepts the following variants for MTD device:

1. <device name> MTD device name. "pstore" is recommended.
2. <device number> MTD device number.

kmsg_size

The chunk size in KB for oops/panic front-end. It **MUST** be a multiple of 4. It's optional if you do not care oops/panic log.

There are multiple chunks for oops/panic front-end depending on the remaining space except other pstore front-ends.

pstore/blk will log to oops/panic chunks one by one, and always overwrite the oldest chunk if there is no more free chunk.

pmsg_size

The chunk size in KB for pmsg front-end. It **MUST** be a multiple of 4. It's optional if you do not care pmsg log.

Unlike oops/panic front-end, there is only one chunk for pmsg front-end.

Pmsg is a user space accessible pstore object. Writes to */dev/pmsg0* are appended to the chunk. On reboot the contents are available in */sys/fs/pstore/pmsg-pstore-blk-0*.

console_size

The chunk size in KB for console front-end. It **MUST** be a multiple of 4. It's optional if you do not care console log.

Similar to pmsg front-end, there is only one chunk for console front-end.

All log of console will be appended to the chunk. On reboot the contents are available in */sys/fs/pstore/console-pstore-blk-0*.

ftrace_size

The chunk size in KB for ftrace front-end. It **MUST** be a multiple of 4. It's optional if you do not care console log.

Similar to oops front-end, there are multiple chunks for ftrace front-end depending on the count of cpu processors. Each chunk size is equal to ftrace_size / processors_count.

All log of ftrace will be appended to the chunk. On reboot the contents are combined and available in */sys/fs/pstore/ftrace-pstore-blk-0*.

Persistent function tracing might be useful for debugging software or hardware related hangs. Here is an example of usage:

```
# mount -t pstore pstore /sys/fs/pstore
# mount -t debugfs debugfs /sys/kernel/debug/
# echo 1 > /sys/kernel/debug/pstore/record_ftrace
# reboot -f
[...]
# mount -t pstore pstore /sys/fs/pstore
# tail /sys/fs/pstore/ftrace-pstore-blk-0
CPU:0 ts:5914676 c0063828  c0063b94  call_cpuidle <- cpu_startup_entry+0x1b8/0x1e0
CPU:0 ts:5914678 c039ecdc  c006385c  cpuidle_enter_state <- call_cpuidle+0x44/0x48
CPU:0 ts:5914680 c039e9a0  c039ecf0  cpuidle_enter_freeze <- cpuidle_enter_state+0x304/0x314
CPU:0 ts:5914681 c0063870  c039ea30  sched_idle_set_state <- cpuidle_enter_state+0x44/0x314
CPU:1 ts:5916720 c0160f59  c015ee04  kernfs_unmap_bin_file <- __kernfs_remove+0x140/0x204
CPU:1 ts:5916721 c05ca625  c015ee0c  __mutex_lock_slowpath <- __kernfs_remove+0x148/0x204
CPU:1 ts:5916723 c05c813d  c05ca630  yield_to <- __mutex_lock_slowpath+0x314/0x358
CPU:1 ts:5916724 c05ca2d1  c05ca638  __ww_mutex_lock <- __mutex_lock_slowpath+0x31c/0x358
```

max_reason

Limiting which kinds of kmsg dumps are stored can be controlled via the `max_reason` value, as defined in include/linux/kmsg_dump.h's `enum kmsg_dump_reason`. For example, to store both Oopses and Panics, `max_reason` should be set to 2 (KMSG_DUMP_OOPS), to store only Panics `max_reason` should be set to 1 (KMSG_DUMP_PANIC). Setting this to 0 (KMSG_DUMP_UNDEF), means the reason filtering will be controlled by the `printk.always_kmsg_dump` boot param: if unset, it'll be KMSG_DUMP_OOPS, otherwise KMSG_DUMP_MAX.

Configurations for driver

A device driver uses `register_pstore_device` with `struct pstore_device_info` to register to pstore/blk.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\[linux-master][Documentation][admin-guide]pstore-blk.rst, line 160)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: fs/pstore/blk.c
   :export:
```

Compression and header

Block device is large enough for uncompressed oops data. Actually we do not recommend data compression because pstore/blk will insert some information into the first line of oops/panic data. For example:

```
Panic: Total 16 times
```

It means that it's OOPS|Panic for the 16th time since the first booting. Sometimes the number of occurrences of oops|panic since the first booting is important to judge whether the system is stable.

The following line is inserted by pstore filesystem. For example:

```
Oops#2 Part1
```

It means that it's OOPS for the 2nd time on the last boot.

Reading the data

The dump data can be read from the pstore filesystem. The format for these files is `dmesg-pstore-blk-[N]` for oops/panic front-end, `pmsg-pstore-blk-0` for pmsg front-end and so on. The timestamp of the dump file records the trigger time. To delete a stored record from block device, simply unlink the respective pstore file.

Attentions in panic read/write APIs

If on panic, the kernel is not going to run for much longer, the tasks will not be scheduled and most kernel resources will be out of service. It looks like a single-threaded program running on a single-core computer.

The following points require special attention for panic read/write APIs:

1. Can **NOT** allocate any memory. If you need memory, just allocate while the block driver is initializing rather than waiting until the panic.
2. Must be polled, **NOT** interrupt driven. No task schedule any more. The block driver should delay to ensure the write succeeds, but NOT sleep.
3. Can **NOT** take any lock. There is no other task, nor any shared resource; you are safe to break all locks.
4. Just use CPU to transfer. Do not use DMA to transfer unless you are sure that DMA will not keep lock.
5. Control registers directly. Please control registers directly rather than use Linux kernel resources. Do I/O map while initializing rather than wait until a panic occurs.
6. Reset your block device and controller if necessary. If you are not sure of the state of your block device and controller when a panic occurs, you are safe to stop and reset them.

pstore/blk supports psblk_blkdev_info(), which is defined in *linux/pstore_blk.h*, to get information of using block device, such as the device number, sector count and start sector of the whole disk.

pstore block internals

For developer reference, here are all the important structures and APIs:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\[linux-master][Documentation][admin-guide]pstore-blk.rst, line 227)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: fs/pstore/zone.c
   :internal:
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\[linux-master][Documentation][admin-guide]pstore-blk.rst, line 230)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/linux/pstore_zone.h
   :internal:
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\[linux-master][Documentation][admin-guide]pstore-blk.rst, line 233)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/linux/pstore_blk.h
   :internal:
```