Converting old watchdog drivers to the watchdog framework

by Wolfram Sang <wsa@kernel.org>

Before the watchdog framework came into the kernel, every driver had to implement the API on its own. Now, as the framework factored out the common components, those drivers can be lightened making it a user of the framework. This document shall guide you for this task. The necessary steps are described as well as things to look out for.

Remove the file_operations struct

Old drivers define their own file_operations for actions like open(), write(), etc... These are now handled by the framework and just call the driver when needed. So, in general, the 'file_operations' struct and assorted functions can go. Only very few driver-specific details have to be moved to other functions. Here is a overview of the functions and probably needed actions:

- open: Everything dealing with resource management (file-open checks, magic close preparations) can simply go. Device specific stuff needs to go to the driver specific start-function. Note that for some drivers, the start-function also serves as the ping-function. If that is the case and you need start/stop to be balanced (clocks!), you are better off refactoring a separate start-function.
- close: Same hints as for open apply.
- write: Can simply go, all defined behaviour is taken care of by the framework, i.e. ping on write and magic char ('V') handling.
- ioctl: While the driver is allowed to have extensions to the IOCTL interface, the most common ones are handled by the framework, supported by some assistance from the driver:

  WDIOC_GETSUPPORT:
  : Returns the mandatory watchdog_info struct from the driver

  WDIOC_GETSTATUS:
  : Needs the status-callback defined, otherwise returns 0

  WDIOC_GETBOOTSTATUS:
  : Needs the bootstatus member properly set. Make sure it is 0 if you don't have further support!

  WDIOC_SETOPTIONS:
  : No preparations needed

  WDIOC_KEEPALIVE:
  : If wanted, options in watchdog_info need to have WDIOF_KEEPALIVEPING set

  WDIOC_SETTIMEOUT:
  : Options in watchdog_info need to have WDIOF_SETTIMEOUT set and a set_timeout-callback has to be defined. The core will also do limit-checking, if min_timeout and max_timeout in the watchdog device are set. All is optional.

  WDIOC_GETTIMEOUT:
  : No preparations needed

  WDIOC_GETTIMELEFT:
  : It needs get_timeleft() callback to be defined. Otherwise it will return EOPNOTSUPP

  Other IOCTLs can be served using the ioctl-callback. Note that this is mainly intended for porting old drivers; new drivers should not invent private IOCTLs. Private IOCTLs are processed first. When the callback returns with -ENOIOCTLCMD, the IOCTLs of the framework will be tried, too. Any other error is directly given to the user.

Example conversion:

```
-static const struct file_operations s3c2410wdt_fops = {
-       .owner          = THIS_MODULE,
-       .llseek         = no_llseek,
-       .write          = s3c2410wdt_write,
-       .unlocked_ioctl = s3c2410wdt_ioctl,
-       .open           = s3c2410wdt_open,
-       .release        = s3c2410wdt_release,
-};
```

Check the functions for device-specific stuff and keep it for later refactoring. The rest can go.

Remove the miscdevice

Since the file_operations are gone now, you can also remove the 'struct miscdevice'. The framework will create it on watchdog_dev_register() called by watchdog_register_device():

```
-static struct miscdevice s3c2410wdt_miscdev = {
```

```
-       .minor          = WATCHDOG_MINOR,
-       .name           = "watchdog",
-       .fops           = &s3c2410wdt_fops,
-};
```

Remove obsolete includes and defines

Because of the simplifications, a few defines are probably unused now. Remove them. Includes can be removed, too. For example:

```
- #include <linux/fs.h>
- #include <linux/miscdevice.h> (if MODULE_ALIAS_MISCDEV is not used)
- #include <linux/uaccess.h> (if no custom IOCTLs are used)
```

Add the watchdog operations

All possible callbacks are defined in 'struct watchdog_ops'. You can find it explained in 'watchdog-kernel-api.txt' in this directory. start() and owner must be set, the rest are optional. You will easily find corresponding functions in the old driver. Note that you will now get a pointer to the watchdog_device as a parameter to these functions, so you probably have to change the function header. Other changes are most likely not needed, because here simply happens the direct hardware access. If you have device-specific code left from the above steps, it should be refactored into these callbacks.

Here is a simple example:

```
+static struct watchdog_ops s3c2410wdt_ops = {
+       .owner = THIS_MODULE,
+       .start = s3c2410wdt_start,
+       .stop = s3c2410wdt_stop,
+       .ping = s3c2410wdt_keepalive,
+       .set_timeout = s3c2410wdt_set_heartbeat,
+};
```

A typical function-header change looks like:

```
-static void s3c2410wdt_keepalive(void)
+static int s3c2410wdt_keepalive(struct watchdog_device *wdd)
 {
...
+
+       return 0;
 }

...

-       s3c2410wdt_keepalive();
+       s3c2410wdt_keepalive(&s3c2410_wdd);
```

Add the watchdog device

Now we need to create a 'struct watchdog_device' and populate it with the necessary information for the framework. The struct is also explained in detail in 'watchdog-kernel-api.txt' in this directory. We pass it the mandatory watchdog_info struct and the newly created watchdog_ops. Often, old drivers have their own record-keeping for things like bootstatus and timeout using static variables. Those have to be converted to use the members in watchdog_device. Note that the timeout values are unsigned int. Some drivers use signed int, so this has to be converted, too.

Here is a simple example for a watchdog device:

```
+static struct watchdog_device s3c2410_wdd = {
+       .info = &s3c2410_wdt_ident,
+       .ops = &s3c2410wdt_ops,
+};
```

Handle the 'nowayout' feature

A few drivers use nowayout statically, i.e. there is no module parameter for it and only CONFIG_WATCHDOG_NOWAYOUT determines if the feature is going to be used. This needs to be converted by initializing the status variable of the watchdog_device like this:

```
.status = WATCHDOG_NOWAYOUT_INIT_STATUS,
```

Most drivers, however, also allow runtime configuration of nowayout, usually by adding a module parameter. The conversion for this would be something like:

```
watchdog_set_nowayout(&s3c2410_wdd, nowayout);
```

The module parameter itself needs to stay, everything else related to nowayout can go, though. This will likely be some code in

open(), close() or write().

Register the watchdog device

Replace misc_register(&miscdev) with watchdog_register_device(&watchdog_dev). Make sure the return value gets checked and the error message, if present, still fits. Also convert the unregister case:

```
-       ret = misc_register(&s3c2410wdt_miscdev);
+       ret = watchdog_register_device(&s3c2410_wdd);

...

-       misc_deregister(&s3c2410wdt_miscdev);
+       watchdog_unregister_device(&s3c2410_wdd);
```

Update the Kconfig-entry

The entry for the driver now needs to select WATCHDOG_CORE:

- select WATCHDOG_CORE

Create a patch and send it to upstream

Make sure you understood Documentation/process/submitting-patches.rst and send your patch to linux-watchdog@vger.kernel.org. We are looking forward to it :)