Embedded Kernel Back-Porting

LinuxCon Japan 2012

Simon Horman <simon@horms.net>
Horms Solutions Ltd.

7th June 2012

Contents

Motivation

For some mystical reason a decision has been made to base a project on an old kernel...

Motivation

For some mystical reason a decision has been made to base a project on an old kernel...

...which lacks support for the hardware to be used

Motivation

For some mystical reason a decision has been made to base a project on an old kernel...

...which lacks support for the hardware to be used

...or required features

Adding Features to an Old Kernel

- Add in-house implementation directly to the old kernel
- Backport implementation from mainline kernel

Adding Features to an Old Kernel

- Add in-house implementation directly to the old kernel
  - Perhaps the most obvious option
- Backport implementation from mainline kernel
  - Mainline implementation may neet to be made first

Adding Features to an Old Kernel

- Add in-house implementation directly to the old kernel
  - Perhaps the most obvious option
  - Optimial for lack of code reuse
- Backport implementation from mainline kernel
  - Mainline implementation may neet to be made first
  - More opportunity for code reuse
  - More opportunity to leverage existing, open solution

Strategy

- Backport small isolated components
- Full backport of targeted components;
- Selected backport of dependencies from other subsystems

Backport Small Isolated Components

- Individual drivers or subsystems

Backport Small Isolated Components

- Individual drivers or subsystems
- For a new board
  - Clock
  - SoC
  - Board
  - Serial driver
  - Then, SMP and other drivers

Full Backport of Targeted Components

- Can we assume that most changes made mainline are either bug fixes or enhancements *and* that testing will show up any regressions?

Full Backport of Targeted Components

- Can we assume that most changes made mainline are either bug fixes or enhancements *and* that testing will show up any regressions?
- If so, it would appear to make sense to make a full backport of a component from mainline.

Full Backport of Targeted Components

- Can we assume that most changes made mainline are either bug fixes or enhancements *and* that testing will show up any regressions?
- If so, it would appear to make sense to make a full backport of a component from mainline.
  - Backport all patches
  - Reduces the scope for conflicts

Selective Backport of Dependencies

- Constrains the overall scope of the backport

Mechanism

- Obtain List of Target-Files
- Mine Patches from Git
- Apply Patches
- Conflict Resolution Strategies
- If at First a Backport Fails

Obtain List of Target-Files

- Obtain a list of files that are directly rated to the component to be backported
  - Be aware that files are added, removed and renamed over time
  - Find all file names used between the source and target kernel versions
  - Ask people who know the code for guidance
  - Kconfig and Makefiles are often too hot to be useful

Mine Patches from Git

- Use git to obtain a short-list of patches

```
git log --oneline v3.0..v3.4 -- fileA fileB ...
```

Apply and Manage Patches

- git cherry-pick -xs + git rebase -i, quilt, ...

Apply and Manage Patches

- git cherry-pick -xs + git rebase -i, quilt, ...
- It is useful to:
  - Note the commit id of the patch in mainline
  - Note any conflicts
  - Sign off the patch, presumably you will distribute it to

Sample Changelog

```
commit 37e7a4e1eddd663a2c5fddaabf80598f204fea62
Author: Paul Gortmaker <paul.gortmaker@windriver.com>
Date:   Sun Jul 31 16:17:29 2011 -0400

    arm: Add export.h to ARM specific files as required.

    These files all make use of one of the EXPORT_SYMBOL variants
    or the THIS_MODULE macro.  So they will need <linux/export.h>

    Signed-off-by: Paul Gortmaker <paul.gortmaker@windriver.com>
    (cherry picked from commit dc28094b905a872f8884f1f1c48ca86b3b78583a)

    Conflicts:

            arch/arm/common/it8152.c

    Signed-off-by: Simon Horman <horms@verge.net.au>
```

Conflict Resolution Strategies

- Drop the patch
  - Is the patch in scope or does it just happen to touch a target-file?

Conflict Resolution Strategies

- Drop the patch
  - Is the patch in scope or does it just happen to touch a target-file?
- Trim the patch
  - Is this a tree-wide patch with only a small portion in scope?

Conflict Resolution Strategies

- Drop the patch
  - Is the patch in scope or does it just happen to touch a target-file?
- Trim the patch
  - Is this a tree-wide patch with only a small portion in scope?
- Manually resolve conflict
  - Is this hot file? Kconfig, Makefile, ...

Conflict Resolution Strategies

- Drop the patch
  - Is the patch in scope or does it just happen to touch a target-file?
- Trim the patch
  - Is this a tree-wide patch with only a small portion in scope?
- Manually resolve conflict
  - Is this hot file? Kconfig, Makefile, ...
- Add dependency
  - Is some new infrastructure or helper-function required?

If at First a Backport Fails: Small Steps

- Incremental Backports
  - Are there a large number of patches?
  - Backport to an intermediate kernel version.
    - Backport from 3.4 to 3.0, then 2.6.35;
    - Backport from 3.4 to 3.3, then 3.3, to 3.2;...
  - Can be time consuming

If at First a Backport Fails: Small Steps

- Incremental Backports
  - Are there a large number of patches?
  - Backport to an intermediate kernel version.
    - Backport from 3.4 to 3.0, then 2.6.35;
    - Backport from 3.4 to 3.3, then 3.3, to 3.2;...
  - Can be time consuming
- Bisection
  - Useful in conjunction with incremental backports
  - Can be very time consuming

If at First a Backport Fails: Change Scope

- Increase Scope
  - Does a dependency need to be satisfied by adding backport of another component?

If at First a Backport Fails: Change Scope

- Increase Scope
  - Does a dependency need to be satisfied by adding backport of another component?
- Decrease Scope
  - Are there too many unnecessary changes.
  - Could dependencies be provided by a selective backport instead of a full backport?

If at First a Backport Fails: Seek Advice

- As for guidance from someone familiar with the mainline code

Example

Backport of CMT timer driver from 3.4 to 3.0

- Simple real-world example

Target-Files

- `include/linux/sh_timer.h`
- `include/linux/sh_cmt.h`
- `drivers/clocksource/sh_cmt.c`

Patch Short-List

```
$ git log --oneline --no-merges v3.0..v3.4 -- \
    include/linux/sh_timer.h include/linux/sh_cmt.h \
    drivers/clocksource/sh_cmt.c
615a445 PM / shmobile: Make CMT driver use pm_genpd_dev_always_on()
7deeab5 drivers/clocksource: Add module.h to those who were using it
        implicitly
3f7e5e2 clocksource: sh_cmt: wait for CMCNT on init V2
```

Apply Patches

```
$ git checkout v3.0 -b 3.0/cmt
Switched to a new branch '3.0/cmt'
$ git cp -xs 615a445 7deeab5 3f7e5e2
[3.0/cmt.tmp ab672c8] clocksource: sh_cmt: wait for CMCNT on init V2
 Author: Magnus Damm <damm@opensource.se>
 1 file changed, 32 insertions(+), 2 deletions(-)
[3.0/cmt.tmp 32fae08] drivers/clocksource: Add module.h to those who
 were using it implicitly
 Author: Paul Gortmaker <paul.gortmaker@windriver.com>
 3 files changed, 3 insertions(+)
[3.0/cmt.tmp 68f65b2] PM / shmobile: Make CMT driver use
 pm_genpd_dev_always_on()
 Author: Rafael J. Wysocki <rjw@sisk.pl>
 1 file changed, 4 insertions(+)
```

Preliminary Backport Complete

Now to test and submit to LTSI for others to enjoy

Questions?