LKM Preresolver
A Lightweight Prelinker for Linux Kernel Modules

Embedded Linux Conference – Europe 2010
Carmelo Amoroso
Software Design Manager
carmelo.amoroso@st.com

Rosario Contarino
contarino.rosario@gmail.com

STMicroelectronics
www.stlinux.com
Agenda

- Fast boot & modules
  - pros & cons
- Symbols resolution
  - State of art
- Preresolver
  - Overview
  - Basics
  - Implementation
  - Pros & Cons
  - Examples & Benchmarks
- Next steps
Speed-up boot time

- Kernel bootup time is a hot topic
  - for embedded systems
  - for laptop and netbooks
  - and for desktop system as well

- Modular kernels is becoming an valuable option
  - There are pros and cons
  - Trade-off between boot-time and runtime optimizations
Using modules

- Modules allow to boot with a *thin* kernel
  - Thinner kernels are faster to boot
  - Fit better on small boot flash device

- Defer initialization of some device drivers
  - Move such drivers to be “*modules*”
  - Beneficial for many drivers which are not needed to boot the system
    - Ethernet (PHY reset) can take some time
    - USB initialization is very slow
    - SATA HDDs can take a lot of time

- *Recall, all driver initialization will take some time.*
Using modules (cont’d)

- Moving to a module has impacts anyway
  - It will require some time for module loading for
    - memory allocation, copying from user space
    - symbols resolution
  - Trade-off between boot-time and run-time!

- In summary
  - For best **boot-time**, modularize many device drivers
  - For best **run-time**, use no (few) modules

- We will focus on optimizing modules loading
  - *In particular the symbol resolution process*
Symbols resolution: state of art

- Symbol resolution process is an avoidable step
  - It is time consuming, never optimised
- Until last year almost nobody took care of it !!!

- At ELC-E 2009 a new strategy was proposed
  - Using hash table for speeding-up symbol resolution
  - Based on SysV hash table used in C runtime library
  - With further optimisations to pre-compute hash values
  - Implemented in STLinux kernel (git.stlinux.com)
  - Unfortunately not up-streamed up to now !!!
Symbols resolution: state of art

- Another solution was developed @ ST again
  - Based on GNU hash table
  - No changes required into the Kernel Symbol Table
  - Added optional Bloom Filtering
  - Coming soon in STLinux kernel
  - *Hopefully it will be up-streamed soon*

- Another solution was proposed by A. Jenkins
  - Based on a binary search in the kernel symbol tables
  - Proposed on the LKML in Oct / Nov 2009
Preresolver: overview

- LKM Preresolver is another solution to speed-up module loading
  - It is a software tool working at kernel build time
  - It is based on the standard concept of prelinking
  - The goal is to perform the **resolution** of the undefined symbols at build time
  - But it does not perform all symbol **relocations**
  - It is fully compatible with any module loader implementation

- So it is a sort of **lightweight** prelinker
Preresolver: basics

- vmlinux is a statically linked binary
- Kernel symbols addresses are absolute
  - They are accessible from the ELF by inspecting the kernel symbol tables
- Kbuild guarantees that there are no duplicated exported symbols
Preresolver: strategy

- Lookup undefined symbols into the kernel symbol tables
- Update the LKM by fixing up the symbol table with the proper values
  - Preresolved symbols are marked as SHN_ABS
  - Symbols not resolved into the vmlinux are kept unchanged (SHN_UND)
- Mark the module as preresolved
  - By adding an empty ELF section (.preresolved)
Preresolver: pros & cons

**Benefits**
- Less undefined symbols to be resolved at load time
- In some cases, 100% of symbol resolution can be optimized away!
  - When all undefined symbols are exported by the `vmlinux`
- Module loader can be instructed to perform symbol lookup in the loaded modules only
  - No symbols are expected to be resolved in the kernel symbol tables

**Drawbacks**
- Preresolved modules cannot be used with different kernels
Preresolver: flow

- vmlinux
- `<module>.ko`
- `<module>.o`
- modpost
- `<module>.mod.o`
- `<module>.mod.c`
- gcc

Flowchart:
- mmap
- fixup
- preresolved
- ld
Preresolver: examples (1)

Examples using a kernel 2.6.32.16 (82 modules)

```
LD   vmlinux
... [SNIP] ...
Building modules, stage 2.
MODPOST 82 modules
...[SNIP] ...
CC    drivers/ata/libata.mod.o
LD [M] drivers/ata/libata.ko
PRERESV drivers/ata/libata.ko (104/104 *fully* preresolved)
CC    drivers/ata/pata_platform.mod.o
LD [M] drivers/ata/pata_platform.ko
PRERESV drivers/ata/pata_platform.ko (11/26 preresolved)
CC    drivers/ata/sata_stm.mod.o
LD [M] drivers/ata/sata_stm.ko
PRERESV drivers/ata/sata_stm.ko (20/41 preresolved)
CC    drivers/hid/usbhid/usbhid.mod.o
LD [M] drivers/hid/usbhid/usbhid.ko
PRERESV drivers/hid/usbhid/usbhid.ko (59/74 preresolved)
CC    drivers/i2c/i2c-core.mod.o
LD [M] drivers/i2c/i2c-core.ko
PRERESV drivers/i2c/i2c-core.ko (47/47 *fully* preresolved)
```
Inspecting the `usbcore.ko` .symtab

```bash
readelf -s usbcore.ko | grep UND | grep printk
```

```
725:00000000 0 NOTYPE GLOBAL DEFAULT UND printk
```

Inspecting the .symtab after the Preresolver

```bash
readelf -s usbcore.ko | grep printk$
```

```
727:80192b44 0 NOTYPE GLOBAL DEFAULT ABS printk
```

Inspecting the `vmlinux` symbol table, we have

```bash
readelf -s vmlinux | grep \ printk$
```

```
14176:80192b44 40 FUNC GLOBAL DEFAULT 2 printk
```

- The `printk` is actually resolved with the absolute address
- The symbol type is changed
Preresolver: figures (best cases)

<table>
<thead>
<tr>
<th>Module</th>
<th>Std</th>
<th>Preres</th>
<th>Module</th>
<th>Std</th>
<th>Preres</th>
</tr>
</thead>
<tbody>
<tr>
<td>xfs.ko</td>
<td>242</td>
<td>1</td>
<td>ntfs.ko</td>
<td>95</td>
<td>0</td>
</tr>
<tr>
<td>ext4.ko</td>
<td>276</td>
<td>39</td>
<td>jbd2.ko</td>
<td>87</td>
<td>0</td>
</tr>
<tr>
<td>nfs.ko</td>
<td>243</td>
<td>44</td>
<td>hostap.ko</td>
<td>88</td>
<td>2</td>
</tr>
<tr>
<td>sunrpc.ko</td>
<td>188</td>
<td>0</td>
<td>mmc_core.ko</td>
<td>77</td>
<td>0</td>
</tr>
<tr>
<td>jfs.ko</td>
<td>166</td>
<td>0</td>
<td>lockd.ko</td>
<td>104</td>
<td>34</td>
</tr>
<tr>
<td>usbcore.ko</td>
<td>163</td>
<td>0</td>
<td>snd.ko</td>
<td>67</td>
<td>1</td>
</tr>
<tr>
<td>cifs.ko</td>
<td>152</td>
<td>0</td>
<td>squashfs.ko</td>
<td>61</td>
<td>0</td>
</tr>
<tr>
<td>fat.ko</td>
<td>108</td>
<td>0</td>
<td>pegasus.ko</td>
<td>68</td>
<td>9</td>
</tr>
<tr>
<td>ide-core.ko</td>
<td>108</td>
<td>0</td>
<td>usbhid.ko</td>
<td>74</td>
<td>15</td>
</tr>
<tr>
<td>libata.ko</td>
<td>104</td>
<td>0</td>
<td>usb-storage.ko</td>
<td>75</td>
<td>16</td>
</tr>
<tr>
<td>jffs2.ko</td>
<td>110</td>
<td>6</td>
<td>mmc_block.ko</td>
<td>59</td>
<td>7</td>
</tr>
<tr>
<td>smbfs.ko</td>
<td>103</td>
<td>0</td>
<td>usbnet.ko</td>
<td>61</td>
<td>10</td>
</tr>
<tr>
<td>Module</td>
<td>Std</td>
<td>Preres</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----</td>
<td>--------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sata_stm.ko</td>
<td>41</td>
<td>21</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>net1080.ko</td>
<td>22</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pata_platform.ko</td>
<td>26</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>aead.ko</td>
<td>22</td>
<td>13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cdc_ether</td>
<td>17</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>chainiv.ko</td>
<td>18</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ecb.ko</td>
<td>13</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>arc4.ko</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cdc_subset.ko</td>
<td>6</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>aes_generic.ko</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Preresolver: benchmarks (1)

- **Scenario 1: Embedded Set-Top-Boxes**
  - ST40 cpu (SH-4 based)
  - Running STLinux distro w/ kernel 2.6.32.16
    - GNU hash loader was used
    - 160 modules loaded/unloaded sequentially

<table>
<thead>
<tr>
<th>No. deps</th>
<th>Symbol resolution</th>
<th>Module loading</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-68,53%</td>
<td>-0,89%</td>
</tr>
<tr>
<td>1</td>
<td>-17,25%</td>
<td>-0,47%</td>
</tr>
<tr>
<td>&gt; 1</td>
<td>-15,85%</td>
<td>-0,45%</td>
</tr>
</tbody>
</table>

*Times measured using gettimeofday
Gain is ((new-old)/old)*100*
Preresolver: benchmarks (2)

Cumulative Curve (ST40)
Modules resolution times

- Standard Kernel
- Preresolver
Scenario 2: laptop/desktop systems

- Based on Intel x86_64 cpu
- Running ArchLinux distro w/ kernel 2.6.33.3
  - 84 modules (default), “modprobed”

<table>
<thead>
<tr>
<th>No. deps</th>
<th>Symbol resolution</th>
<th>Module loading</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-98.72%</td>
<td>-50.55%</td>
</tr>
<tr>
<td>1</td>
<td>-69.08%</td>
<td>-48.22%</td>
</tr>
<tr>
<td>&gt; 1</td>
<td>-32.08%</td>
<td>-30.05%</td>
</tr>
</tbody>
</table>

Times measured using gettimeofday
Gain is ((new-old)/old)*100
Preresolver: benchmarks (4)
Next steps

- Trying to upstream !!!

- Further optimisations
  - Strip down kernel by removing kernel symbol tables
    - Works only with preresolved modules
    - Acceptable? In an embedded scenario likely yes
  - Direct binding
    - Use dependencies information in .modinfo section
    - Perform lookup into a subset of modules

- Merge all together (GNU hash, Preresolver, Direct binding) for **fastest module loading**
Thanks for your attention