Internet of Tiny Linux (IoTL) status and progress
Internet of Tiny Linux (IoTL)

Discussion about various methods put forward to reduce the size of Linux kernel and user space binaries to make them suitable for small IoT applications.
Very pervasive:

- Cable/ADSL Modems
- Smart Phones
- Smart Watches
- Internet Connected Refrigerators
- WI-FI-Enabled Washing Machines
- Smart TV Sets
- Wi-Fi enabled Light Bulbs
- Connected Cars
- Alarm Systems monitored via Internet
- etc.
"Internet of Things" (IoT)

Problem: **Cost**

Software Development is
- Hard
- Expensive
- Slow

Legacy Software Maintenance is
- Hard
- Expensive
- Uninteresting
"Internet of Things" (IoT)

Another Problem: *Security*

- All solutions _will_ eventually be broken
  - Think NSA…
  - Then professional thieves…
  - Then script kiddies…
- Security Response is a _must_... even for gadgets!*

quote, Bruce Schneier:

“The Internet of Things Is Wildly Insecure-And Often Unpatchable”
"Internet of Things" (IoT)

Solutions:
- Avoid custom base software
- Leverage the Open Source community
- Gather critical mass around common infrastructure
- Share the cost of non-differentiating development
"Internet of Things" (IoT)

**Linux** is a logical choice

- Large community of developers
- Best looked-after network stack
- Extensive storage options
- Already widely used in embedded setups
- Etc.
“Internet of Things” (IoT)

The Linux kernel is a logical choice... BUT

- it is featureful -> Bloat
- its default tuning is for high-end systems
- the emphasis is on scaling up more than scaling down
- its flexible configuration system leads to
  - Kconfig hell
  - suboptimal build
- is the largest component in most Linux-based embedded systems

Linux Kernel Size Reduction is part of the solution*
"Internet of Things" (IoT)

Reducing the Linux Kernel Size

What can be done?

Existing resources:

"Internet of Things" (IoT)

Reducing the Linux Kernel Size

Already Done (few examples):

- Compile out block device support.
- Compile out NTP support.
- Compile out support for capabilities (only root is granted permission).
- Compile out support for non-root users and groups.
- Compile out printk() and related strings.

Needed: more similar config options.
"Internet of Things" (IoT)

Reducing the Linux Kernel Size

Work In Progress: A poor man's LTO

LTO is cool... BUT

Table 1: Full Kernel Build Timing

<table>
<thead>
<tr>
<th>Build Type</th>
<th>Wall Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard Build</td>
<td>4m53s</td>
</tr>
<tr>
<td>LTO Build</td>
<td>10m23s</td>
</tr>
</tbody>
</table>

Table 2: Kernel Rebuild Timing After a Single Change

<table>
<thead>
<tr>
<th>Build Type</th>
<th>Wall Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard Build</td>
<td>0m12s</td>
</tr>
<tr>
<td>LTO Build</td>
<td>6m27s</td>
</tr>
</tbody>
</table>

NOTE: Build Details: Linux v4.2 ARM multi_v7_defconfig gcc v5.1.0 Intel Core2 Q6600 CPU at 2.40GHz
"Internet of Things" (IoT)

Reducing the Linux Kernel Size
Alternative to LTO: `ld -gc-sections`

<table>
<thead>
<tr>
<th>Build Type</th>
<th>Size (bytes)</th>
<th>Reference %</th>
</tr>
</thead>
<tbody>
<tr>
<td>allnoconfig</td>
<td>860508</td>
<td>100%</td>
</tr>
<tr>
<td>allnoconfig + CONFIG_NO_SYSCALLS</td>
<td>815804</td>
<td>94.8%</td>
</tr>
<tr>
<td>allnoconfig + CONFIG_NO_SYSCALLS + CONFIG_GC_SECTIONS</td>
<td>555798</td>
<td>64.6%</td>
</tr>
<tr>
<td>allnoconfig + CONFIG_NO_SYSCALLS + CONFIG_LTO</td>
<td>488264</td>
<td>56.7%</td>
</tr>
</tbody>
</table>

The `-gc-sections` result is somewhat bigger but so much faster to build.
"Internet of Things" (IoT)

Reducing the Linux Kernel Size

The `ld -gc-sections` approach: more intrusive than meets the eye

--- a/arch/arm/include/asm/assembler.h
+++ b/arch/arm/include/asm/assembler.h
@@ -88,6 +88,17 @@
 #endif
 /*
 * Special .pushsection wrapper with explicit dependency to prevent
 * garbage collection of the specified section. This is needed when no
 * explicit symbol references are made to this section.
 */
+ .macro .pushlinkedsection name:vararg
+ .reloc . - 1, R_ARM_NONE, 9909f
+ .pushsection 'name
+9909:
+ .endm
+
+*/

Let's not forget changes to linker scripts, modpost, etc.
"Internet of Things" (IoT)

Reducing the Linux Kernel Size (continued)

The `ld -gc-sections` approach: more intrusive than meets the eye

```c
/* Enable and disable interrupts
 */
#if __LINUX_ARM_ARCH__ >= 6
@@ -239,7 +250,7 @@
9999:
- .pushsection __ex_table,"a";
+ .pushlinkedsection __ex_table,"a";
 .align 3;
 .long 9999b,9001f;
 .popsection
```

Let's not forget changes to linker scripts, modpost, etc.
"Internet of Things" (IoT)

Reducing the Linux Kernel Size

Submitted Upstream: Trim unused exported kernel symbols

- EXPORT_SYMBOL(foo_bar) forces foo_bar() into the kernel even if there is no users.
- Let's export only those symbols needed by the set of configured modules.
- Allows LTO and -gc-sections to get rid of related code.
"Internet of Things" (IoT)

Reducing the Linux Kernel Size

List of symbols required by modules

```bash
$ nm linux/crypto/sha512_generic.ko
00000000 T cleanup_module
    U crypto_register_shashes
00000080 T crypto_sha512_finup
000000c0 T crypto_sha512_update
    U crypto_unregister_shashes
00000000 T init_module
    U memcpy
00000e8 r __module_depends
00000000 t sha384_base_init
00000000 d sha512_algs
00000000 t sha512_algs
00000090 t sha512_base_init
000000d0 t sha512_final
...
"Internet of Things" (IoT)

Reducing the Linux Kernel Size
List of symbols required by modules

```
linux/include/generated/autoksymshh

----
/*
 * Automatically generated file; DO NOT EDIT.
 */

#define __KSYM_crypto_register_shashes 1
#define __KSYM_crypto_unregister_shashes 1
#define __KSYM_memcpy 1
----
```
"Internet of Things" (IoT)

Reducing the Linux Kernel Size

Trim unused exported kernel symbols == some preprocessor magic

Excerpt from mainline submission

--- a/include/linux/export.h
+++ b/include/linux/export.h
@@ -65,6 +65,24 @@

- __attribute__((section("___ksymtab" sec "*" #sym), unused))
+ (unsigned long)&sym, __kstrtab_##sym

+ #ifdef CONFIG_TRIM_UNUSED_KSYMS
+ #include <linux/kconfig.h>
+ #include <generated/autoksyms.h>
+ #define __EXPORT_SYMBOL(sym, sec)
+     __cond_export_sym(sym, sec, config_enabled(__KSYM_##sym))
+ #define __cond_export_sym(sym, sec, conf)
+     ___cond_export_sym(sym, sec, conf)
"Internet of Things" (IoT)

Reducing the Linux Kernel Size

Trim unused exported kernel symbols == some preprocessor magic (Continued)

Excerpt from mainline submission

```c
#define ___cond_export_sym(sym, sec, enabled)            \'
  ___cond_export_sym_##enabled(sym, sec)
#define __cond_export_sym_1(sym, sec) ___EXPORT_SYMBOL(sym, sec)
#define __cond_export_sym_0(sym, sec) /* nothing */
  +
  +
#define __cond_export_sym_0(sym, sec) /* nothing */
  +
#define EXPORT_SYMBOL(sym)
  +
  +
#else
#define __EXPORT_SYMBOL ___EXPORT_SYMBOL
  +
#endif

#define EXPORT_SYMBOL(sym)
  +
  +
#define __EXPORT_SYMBOL(sym, ")"
  +
  +
#define EXPORT_SYMBOL(sym, "")
  +
```

---
"Internet of Things" (IoT)

Reducing the Linux Kernel Size

Trim unused exported kernel symbols == more preprocessor magic

Excerpt from `<linux/kconfig.h>` for the `config_enabled()` definition

```c
/*
 * Getting something that works in C and CPP for an arg that may or may
 * not be defined is tricky. Here, if we have `#define CONFIG_BOOGER 1`
 * we match on the placeholder define, insert the "0," for arg1 and generate
 * the triplet (0, 1, 0). Then the last step cherry picks the 2nd arg (a one).
 * When CONFIG_BOOGER is not defined, we generate a (... 1, 0) pair, and when
 * the last step cherry picks the 2nd arg, we get a zero.
 */
#define __ARG_PLACEHOLDER_1 0,
#define config_enabled(cfg) _config_enabled(cfg)
#define _config_enabled(value) __config_enabled(__ARG_PLACEHOLDER_##value)
#define __config_enabled(arg1_or_junk) ___config_enabled(arg1_or_junk 1, 0)
#define ___config_enabled(__ignored, val, ...) val
```
"Internet of Things" (IoT)

Reducing the Linux Kernel Size

Trim unused exported kernel symbols: build refresh dependencies

Avoid rebuilding the whole kernel when content of `<linux/autoksym.h>` changes.

First attempt: augment the fixdep parser

- `EXPORT_SYMBOL()`
- `EXPORT_SYMBOL_GPL()`
- `EXPORT_PER_CPU_SYMBOL()`
- `EXPORT_EARLY_SYMBOL()`
- `ACPI_EXPORT_SYMBOL()`
- `ACPI_EXPORT_SYMBOL_INIT()`

And combinations such as `EXPORT_PER_CPU_SYMBOL_GPL()`, etc.
"Internet of Things" (IoT)

Reducing the Linux Kernel Size

Trim unused exported kernel symbols: build refresh dependencies

Now this:

```c
#define PCI_USER_READ_CONFIG(size, type)               
int pci_user_read_config_##size                                         
   (struct pci_dev *dev, int pos, type *val)                       
{
   [...]
}
EXPORT_SYMBOL_GPL(pci_user_read_config_##size);
```
"Internet of Things" (IoT)

Reducing the Linux Kernel Size

Trim unused exported kernel symbols: build refresh dependencies

Extract symbol name warnings:

```bash
# Filter out exported kernel symbol names advertised as warning pragmas
# by the preprocessor and write them to $(1). We must consider continuation
# lines as well: they start with a blank, or the preceding line ends with
# a ':'. Anything else is passed through as is.
# See also __KSYM_DEP() in include/linux/export.h.
ksym_dep_filter = sed -n
  -e "1 {x; $$!d;}
  -e "!^ / {H; $$!d;}
  -e "x; /$$/ {x; H; $$!d; s/^/ /; x}
  -e ".filter; /^.*KBUILD_AUTOKSYM_DEP: /! {p; b next;}
  -e s/\(\)/\1/; x
  -e "next; $$!d\n  -e "1 q; s/^/ /; x; !^ / b filter"
```
"Internet of Things" (IoT)

Reducing the Linux Kernel Size

Trim unused exported kernel symbols: build refresh dependencies

Dedicated preprocessor pass:

--- a/include/linux/export.h
+++ b/include/linux/export.h
@@ -65,7 +65,18 @@ extern struct module __this_module;
- #ifdef CONFIG_TRIM_UNUSED_KSYMS
+ #if defined(__KSYM_DEPS__)
+ /*
+ * For fine grained build dependencies, we want to tell the build system
+ * about each possible exported symbol even if they're not actually exported.
+ * We use a string pattern that is unlikely to be valid code that the build
+ * system filters out from the preprocessor output (see ksym_dep_filter
+ * in scripts/Kbuild.include).
+ */
"Internet of Things" (IoT)

Reducing the Linux Kernel Size

Trim unused exported kernel symbols: build refresh dependencies (Continued)

Dedicated preprocessor pass:

```c
#define __EXPORT_SYMBOL(sym, sec)    ___KSYM_##sym ___

#ifdef CONFIG_TRIM_UNUSED_KSYMS

#include <linux/kconfig.h>
#include <generated/autoksym.h>
```

```c
```
"Internet of Things" (IoT)

Reducing the Linux Kernel Size

Trim unused exported kernel symbols: some numbers.

Kernel v4.5-rc2 X86 defconfig

```bash
$ size vmlinux
  text  data   bss  dec     hex  filename
12362563  1856456 1101824 15320843    e9c70b  vmlinux

$ wc -l Module.symvers
  8806  Module.symvers

$ size vmlinux
  text  data   bss  dec     hex  filename
12059848  1856456 1101824 15018128    e52890  vmlinux

$ wc -l Module.symvers
  225  Module.symvers
```

Kernel v4.5-rc2 X86 defconfig + CONFIG_TRIM_UNUSED_KSYMS

Because the x86 defconfig only contains 18 modules, the number of needed exported symbols is only 225 out of a possible 8806 for this configuration. The kernel text size shrunk by about 2.4%.
"Internet of Things" (IoT)

Reducing the Linux Kernel Size

Trim unused exported kernel symbols: more numbers, on ARM this time.

Kernel v4.5-rc2 ARM defconfig

```bash
$ size vmlinux
  text  data  bss  dec  hex  filename
13664222  1554068  351368  15569658  ed92fa  vmlinux
$ wc -l Module.symvers
  10044 Module.symvers
```

Kernel v4.5-rc2 ARM defconfig + CONFIG_TRIM_UNUSED_KSYMS

```bash
$ size vmlinux
  text  data  bss  dec  hex  filename
13255051  1554132  351240  15160423  e75467  vmlinux
$ wc -l Module.symvers
  2703 Module.symvers
```

This time many more modules (279 of them) are part of the build configuration. Still, only 2703 out of 10044 exported symbols are required. And despite a smaller number of omitted exports, the kernel shrank by 3%.

NOTE: defconfig is equivalent to multi_v7_defconfig
"Internet of Things" (IoT)

Reducing the Linux Kernel Size

Trim unused exported kernel symbols: more numbers using LTO build.

Kernel v4.5-rc2 ARM defconfig + LTO

```
$ size vmlinux
  text  data  bss  dec  hex  filename
12813766  1538324  344356  14696446  e03ffe  vmlinux
$ wc -l Module.symvers
  8415 Module.symvers
```

Kernel v4.5-rc2 ARM defconfig + LTO + CONFIG_TRIM_UNUSED_KSYMS

```
$ size vmlinux
  text  data  bss  dec  hex  filename
12197437  1536052  338660  14072149  d6b955  vmlinux
$ wc -l Module.symvers
  1742 Module.symvers
```

This time the kernel shrank by 5%.
"Internet of Things" (IoT)

Reducing the Linux Kernel Size

Trim unused exported kernel symbols: more numbers on a small build. Let's have a look at a configuration that is potentially more representative of an embedded target.

Kernel v4.5-rc2 ARM realview_defconfig + LTO

```
$ size vmlinux
          text     data     bss      dec      hex  filename
 4422942  209640  126880    4759462    489fa6  vmlinux
$ wc -l Module.symvers
  5597 Module.symvers
```

```
$ size vmlinux
          text     data     bss      dec      hex  filename
 3823485  205416  125800    4154701    3f654d  vmlinux
$ wc -l Module.symvers
  52 Module.symvers
```

Here we reduced the kernel text by about 13.6%. Disabling module support altogether does reduce it by 13.9% i.e. only 0.3% difference.

This means the overhead of using modules on embedded targets is greatly reduced.
"Internet of Things" (IoT)

Reducing the Linux Kernel Size *AND* User Space Size

What next?

- Apply same trick to system calls
- Tool to determine list of system calls used by user space

To be effective, this implies:

- Statically linked applications
- Single executable file (busybox style)

For more flexibility:

- Shared library with minimal support (discard unused objects)
- Fine grained object dependencies (more .o files with less code for each)
"Internet of Things" (IoT)

Reducing the Linux Kernel Size *AND* User Space Size

Miscellaneous:

- Code utilization profiling
- Binary size regression tests
- NoMMU Linux on Cortex-A processors
- Mainline support for FDPIC executable format
- Revive XIP support for the kernel and user space
- More kernel feature modularization
"Internet of Things" (IoT)

Questions?