VAMPYR: Configurability-Aware Compile-Testing of Source Files

Stefan Hengelein       Daniel Lohmann
stefan.hengelein@fau.de  dl@cs.fau.de

System Software Group
Friedrich-Alexander University Erlangen-Nürnberg (FAU)
https://cados.cs.fau.de

LPC ’14
Before submitting patches kernel developers are expected to follow:

...  

8 Has been carefully reviewed with respect to relevant Kconfig combinations. This is very hard to get right with testing – brainpower pays off here.

...
Before submitting patches kernel developers are expected to follow:

... 

8 Has been carefully reviewed with respect to relevant Kconfig combinations. This is very hard to get right with testing – **brainpower pays off here.**

... 

why is that a problem?
Compile-test BLOCK1 and BLOCK2

```c
#ifdef CONFIG_A
    block1
#else
    block2
#endif
```
Compile-test BLOCK1 and BLOCK2

```c
#define CONFIG_A
    block1
#else
    block2
#endif
```

However, one .CONFIG cannot cover both blocks
Compile-test BLOCK1 and BLOCK2

```c
#ifdef CONFIG_A
    block1
#else
    block2
#endif
```

However, one .CONFIG cannot cover both blocks

Code is often compile-tested with one allyesconfig
Compile-test Block1 and Block2

```c
#ifdef CONFIG_A
  block1
#else
  block2
#endif
```

However, one .CONFIG cannot cover both blocks

Code is often compile-tested with one allyesconfig

Bugs are easily missed!
How hard is the problem?

Configurability:
How hard is the problem?

Configurability:

```c
#ifdef CONFIG_HOTPLUG_CPU
...\n#endif
```

Kbuild

- Makefile
- arch/x86/init.c
- arch/x86/entry32.S
- arch/x86/...
- lib/Makefile
- kernel/sched.c

CPP

```
#ifndef CONFIG_HOTPLUG_CPU
...
#endif
```
How hard is the problem?

Configurability:

Kconfig

Kbuild

CPP

MEMORY MODEL
- FLATMEM
- DISCONTIGMEM
- SPARSEMEM
- NUMA

#ifdef CONFIG_HOTPLUG_CPU
...endif

Makefile
arch/x86/init.c
arch/x86/entry32.S
arch/x86/...
lib/Makefile
kernel/sched.c
...

MEMORY MODELS
- FLATMEM
- DISCONTIGMEM
- SPARSEMEM
- NUMA

depends on

undertaker.cs.fau.de vampyr: Configurability-Aware Compile-Testing of Source Files (LPC '14)
How hard is the problem?

Configurability:

```c
#ifdef CONFIG_HOTPLUG_CPU
...
#endif
```

```
Memorize Model
FLATMEM
DISCONTIGMEM
SPARSEMEM
NUMA
```

Still solvable by brainpower?

How much time does it take to get it right?
Configuration Coverage: The VAMPYR Approach

Idea: Create a maximizing set of configurations

Kconfig

MEMORY MODEL
FLATMEM
DISCONTIGMEM
SPARSEMEM
NUMA
depends on

Kbuild

Makefile
arch/x86/init.c
arch/x86/entry32.S
arch/x86/...
lib/Makefile
kernel/sched.c
...

CPP

#ifdef CONFIG_HOTPLUG_CPU
... modif
#endif

#ifdef CONFIG_HOTPLUG_CPU
...
#endif

calculate configurations for each source file

Config 1
Config 2
Config ...
Config n

apply

sparse
gcc
clang
coccinelle
Idea: Create a **maximizing set of configurations**
Idea: Create a maximizing set of configurations

For each source file, calculate configurations and apply them to:
- gcc
- clang
- coccinelle
- sparse
- undertaker.cs.fau.de

Dependencies:
- MEMORY_MODEL
  - FLATMEM
  - DISCONTIGMEM
- SPARSEMEM
- NUMA

Configurations:
- Config 1
- Config 2
- Config...
- Config n

#ifdef CONFIG_HOTPLUG_CPU
...
#endif

Makefile:
- arch/x86/init.c
- arch/x86/entry32.S
- arch/x86/...
- lib/Makefile
- kernel/sched.c
- ...

CPP

#ifdef CONFIG_HOTPLUG_CPU
...
#endif
Evaluation: A Configurability-Aware Tool Driver

Evaluated for Linux/v3.2

Number of found compiler warnings and errors increased significantly

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Increase in detected GCC warnings and errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux/x86</td>
<td>176 → 202 (+15%)</td>
</tr>
<tr>
<td>Linux/mips</td>
<td>158 → 249 (+58%)</td>
</tr>
</tbody>
</table>

Linux/arm: Analysis of Warnings and Errors not found with allyesconfig

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>∑ Less critical GCC messages</td>
<td>223 → 363 (+63%)</td>
</tr>
<tr>
<td>∑ Reported issues by VAMPYR</td>
<td>254 → 454 (+79%)</td>
</tr>
<tr>
<td>∑ Manually validated bugs</td>
<td>31 → 91</td>
</tr>
</tbody>
</table>

Evaluation: A Configurability-Aware Tool Driver

Evaluating for Linux/\nu3.2 \textsuperscript{a}

Number of found compiler warnings and errors increased significantly

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Increase in detected GCC warnings and errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux/x86</td>
<td>176 → 202 (+15%)</td>
</tr>
<tr>
<td>Linux/mips</td>
<td>158 → 249 (+58%)</td>
</tr>
</tbody>
</table>

Linux/arm: Architecture Increase in detected gcc warnings and errors

<table>
<thead>
<tr>
<th>Message Type</th>
<th>Before</th>
<th>After</th>
<th>Increase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Less critical gcc messages</td>
<td>223</td>
<td>363</td>
<td>(+63%)</td>
</tr>
<tr>
<td>Reported issues by vampyr</td>
<td>254</td>
<td>454</td>
<td>(+79%)</td>
</tr>
<tr>
<td>Manually validated bugs</td>
<td>31</td>
<td>91</td>
<td></td>
</tr>
</tbody>
</table>

\textsuperscript{a}https://www4.cs.fau.de/Publications/2014/tartler_14_usenix.pdf

Luckily:

The number of found warnings and errors is lower in Linux/\nu3.17
Example 1 (v3.17 - MIPS)

---

**Found 1 messages with gcc in arch/mips/ath79/mach-db120.c**

... mach-db120.c:132: error: too many arguments to function 'db120_pci_init'
in configs: arch/mips/ath79/mach-db120.c.config1

---

```c
#define CONFIG_PCI
static void __init db120_pci_init(u8 *eeprom) { [...] }
#else
static void __init db120_pci_init(void) {}
#endif

static void __init db120_setup(void) {
    [...]  
    db120_pci_init(art + DB120_PCIE_CALDATA_OFFSET);
}
```

---

Conclusions

intentional broken compilation?

why not #error?
or model configurability better

---

undertaker.cs.fau.de  VAMPYR: Configurability-AwareCompile-Testing of Source Files (LPC '14)
Example 1 (v3.17 - MIPS)

---

```c
# ifdef CONFIG_PCI
static void __init db120_pci_init(u8 *eeprom) {
    [...] 
}
# else
static void __init db120_pci_init(void) {} 
# endif

static void __init db120_setup(void) {
    [...] 
    db120_pci_init(art + DB120_PCIE_CALDATA_OFFSET);
}
```

Conclusions

- intentional broken compilation?
- why not `#error`?
- or model configurability better
#ifdef CONFIG_MMU
static struct gen_pool *atomic_pool;
static bool __in_atomic_pool( [...]) {
    return addr_in_gen_pool(atomic_pool, [...]);
}
#endif

static struct page **__atomic_get_pages(void *addr) {
    phys = gen_pool_virt_to_phys(atomic_pool, [...]);
}

static struct page **__iommu_get_pages([...]) {
    if (__in_atomic_pool(cpu_addr, PAGE_SIZE))
        return __atomic_get_pages(cpu_addr);
}
Example 2 (v3.17 - ARM)

==== Found 8 messages with gcc in arch/arm/mm/dma-mapping.c ====

... dma-mapping.c:1358: error: 'atomic_pool' undeclared (first use in this function)
(in configs: arch/arm/mm/dma-mapping.c.config1)

... dma-mapping.c:1369: error: implicit declaration of function '__in_atomic_pool'
(in configs: arch/arm/mm/dma-mapping.c.config1)...

#define CONFIG_MMU
static struct gen_pool *atomic_pool;
static bool __in_atomic_pool(...)
{
  return addr_in_gen_pool(atomic_pool, [...]);
}
#endif

static struct page **__atomic_get_pages(void *addr)
{
  phys = gen_pool_virt_to_phys(atomic_pool, [...]);
}

static struct page **__iommu_get_pages(...)
{
  if (__in_atomic_pool(cpu_addr, PAGE_SIZE))
    return __atomic_get_pages(cpu_addr);
}
Example 2 (v3.17 - ARM)

==== Found 8 messages with gcc in arch/arm/mm/dma-mapping.c ====

... dma-mapping.c:1358: error: 'atomic_pool' undeclared (first use in this function)
(in configs: arch/arm/mm/dma-mapping.c.config1)
... dma-mapping.c:1369: error: implicit declaration of function '__in_atomic_pool'
(in configs: arch/arm/mm/dma-mapping.c.config1) ...

#define CONFIG_MMU
static struct gen_pool *atomic_pool;
static bool __in_atomic_pool([...]) {
    return addr_in_gen_pool(atomic_pool, [...]);
}
#endif

static struct page **__atomic_get_pages(void *addr) {
    phys = gen_pool_virt_to_phys(atomic_pool, [...]);
}
static struct page **__iommu_get_pages([...]) {
    if (__in_atomic_pool(cpu_addr, PAGE_SIZE))
        return __atomic_get_pages(cpu_addr);
}
Example 2 (v3.17 - ARM)

```c
#ifdef CONFIG_MMU
static struct gen_pool *atomic_pool;
static bool __in_atomic_pool([...]) {
    return addr_in_gen_pool(atomic_pool, [...]);
}
#endif

static struct page **__atomic_get_pages(void *addr) {
    phys = gen_pool_virt_to_phys(atomic_pool, [...]);
}
static struct page **__iommu_get_pages([...]) {
    if (__in_atomic_pool(cpu_addr, PAGE_SIZE))
        return __atomic_get_pages(cpu_addr);
}
```

Conclusions

- #ifdef around 'uses' missing or declaration should have been unconditional
#ifdef CONFIG_MIPS_MT_SMP
static int msp_cic_irq_set_affinity(struct irq_data *d, [...]) {
    unsigned long imask = (1 << (irq - MSP_CIC_INTBASE)) ;
    [...]  
    BUG_ON(irq) == MSP_INT_VPE0_TIMER || irq == MSP_INT_VPE1_TIMER);  
}
#endif
Example 3 (v3.17 - MIPS)

```
==== Found 1 messages with gcc in arch/mips/pmcs-msp71xx/msp_irq_cic.c
==== ... msp_irq_cic.c:134: error: 'irq' undeclared (first use in this function)
(in configs: arch/mips/pmcs-msp71xx/msp_irq_cic.c.config0)
=============================================  

#include CONFIG_MIPS_MT_SMP
static int msp_cic_irq_set_affinity(struct irq_data *d, ...
{  
  unsigned long imask = (1 << (irq - MSP_CIC_INTBASE))  
  ;
  [
  BUG_ON(irq) == MSP_INT_VPE0_TIMER || irq == MSP_INT_VPE1_TIMER);
}
#endif
```

Conclusions
- obviously never compiled
- should have been d->irq
Major problem I see is that many architecture maintainers don’t seem to care about MAKE ALLMODCONFIG and/or MAKE ALLYESCONFIG, meaning there is no simple means to at least compile-test all code that _can_ be enabled for a given architecture. And don’t even mention MAKE RANDCONFIG.

Guenter Roeck
“Major problem I see is that many architecture maintainers don’t seem to care about `make allmodconfig` and/or `make allyesconfig`, meaning there is no simple means to at least compile-test all code that can be enabled for a given architecture. And don’t even mention `make randconfig`.

Integrate **VAMPYR** into your development workflow! replace brainpower by tools!"

Guenter Roeck
Major problem I see is that many architecture maintainers don’t seem to care about `MAKE ALLMODCONFIG` and/or `MAKE ALLYESCONFIG`, meaning there is no simple means to at least compile-test all code that can be enabled for a given architecture. And don’t even mention `MAKE RANDCONFIG`.

Integrate VAMPYR into your development workflow! replace brainpower by tools!

Guenter Roeck
Summary and Conclusions

- **VAMPYR** is a tool to help developers to compile-test their source code.

- The tool is available under **GPLv3**!

- **VAMPYR** has been applied to other configurable system software: busybox, L4/Fiasco

- **Integration into** Undertaker–CheckPatch **is on the way**
Interested?

- Download and try the tool:
  
  https://undertaker.cs.fau.de

- More information and papers on the project’s website:
  
  https://cados.cs.fau.de

- Questions? Contact me directly ...

  stefan.hengelein@fau.de

- ... or write to our mailing list!

  cados-dev@lists.cs.fau.de
VAMPYR reveals issues not covered by allyesconfig:

<table>
<thead>
<tr>
<th>Less critical GCC messages</th>
<th>Compiler Diagnostics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Less critical messages</td>
<td>223 → 363 +140</td>
</tr>
<tr>
<td>Undeclared types/identifiers</td>
<td>4 → 46 +42</td>
</tr>
<tr>
<td>Access to possibly uninitialized data</td>
<td>20 → 22 +2</td>
</tr>
<tr>
<td>Out of bounds array accesses</td>
<td>7 → 13 +4</td>
</tr>
<tr>
<td>Bad pointer casts</td>
<td>0 → 8</td>
</tr>
<tr>
<td>Format string warnings</td>
<td>0 → 1</td>
</tr>
<tr>
<td>Integer overflows</td>
<td>0 → 1</td>
</tr>
<tr>
<td><strong>Σ Bugs found</strong></td>
<td><strong>31 → 91 +60</strong></td>
</tr>
</tbody>
</table>

Σ Reported issues by VAMPYR 254 → 454 +200

Seven patches were submitted and accepted by Linux maintainers
Common approach: use a single configuration (i.e. allyesconfig)

Configuration Coverage is the percentage of code that is covered by all tested configurations

This is insufficient because for allyesconfig:

<table>
<thead>
<tr>
<th>Configuration Coverage on</th>
<th>V3.2</th>
<th>V3.17-RC7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux/x86</td>
<td>78.6%</td>
<td>77%</td>
</tr>
<tr>
<td>Linux/arm</td>
<td>59.7%</td>
<td>69.5%</td>
</tr>
<tr>
<td>Linux/mips</td>
<td>54.6%</td>
<td>52.4%</td>
</tr>
</tbody>
</table>
Evaluation: Setup and Runtime Requirements

- Application of VAMPYR on all 24 Linux Architectures of Linux v3.2
- Used Static Checker: GCC: 4.7
- On average, 1.2 compiler invocations per file
  \[\Rightarrow\] Overhead \(\sim\) 20%

- Runtime on a Standard Intel Quad-Core Workstation:
  - Incremental analysis of an individual file: < 1 minute
  - Generation of 11470 (on Linux/x86) partial configurations in \(\sim\) 4 minutes
  - Analysis of a full architecture: \(\sim\) 2 hours
  - Majority of time is spent with activating KCONFIG configurations
Why not 100 percent Configuration coverage?

- Bugs in `KCONFIG` descriptions in the Linux kernel can cause incorrect expansions of partial configurations.
- Imperfect model extractors can also lower the Configuration Coverage.
Higher Coverage Criteria

- **VAMPYR:** *statement coverage* ($\approx 20\%$ overhead)

- Possibly achievable: *decision coverage*: ($\approx 29\%$ overhead)

- Expensive: *path coverage*: ($??$ overhead)

- Further research: Pairwise testing (cf. related work)