Sparse
a short overview

Nicholas Mc Guire <safety@osadl.org>

March 5, 2019
History
static code testing in Linux

- Formal methods in the Linux kernel start early with the Stanford checker
  - Proprietary? Nop - secret!
  - Later commercialized by Coverity as SWAT
  - Quite a nice number of bugs were found though

- What to do about this situation?
  - Sit down and write your own tool - Linus Torvalds writes sparse (2002)
  - By now it is well established in the kernel and maintained by Luc Van Oostenryck
  - Current version 0.6.0 -
Sparse really is two things

- A generic parse tree library
- A tool handling Linux kernel specifics

For both we simply refer to ”sparse”
What is sparse

- What it's not
  - It is not a compiler or preprocessor
  - It's not a tokenizer nor a context-free parser

- What it is
  - Sparse is a context/type aware semantic source parser
    - what the types are that the grouping implies
  - Mission: create a semantic parse tree (abstract syntax tree (AST) of a C program) suitable for arbitrary further analysis
  - Do one thing do it well: Emit the full parse tree - done.
    - but it does have functionality specifically tailored to check Linux kernel patterns
  - A set of simple clients that are a good starting point for new checkers (see obfuscate.c, test-inspect.c, compile.c)

See the README in the sparse sources for details of phases and flow.
sparse

Goals of sparse

- Enable static checking in mainline kernel
- Mandate its use (i.e. by linux-next building with sparse)
- Continuously update it when new pitfalls emerge (though changrate is slow by now)
- Provide a reliable parser useful to build other tools around (e.g. smatch)
- Use these checkers to catch the common problems that happen during development
sparse
Installation and Usage

$ git clone \ngit://git.kernel.org/pub/scm/devel/sparse/sparse.git

$ make
$ make install

$ cd linux
$ make -j4 C=1
$ make -j4 C=2
Your functions MUST have types - no KR style default int!

It does NOT cover all possible extensions to C - it does cover everything the kernel needs though (GNU and C99)

You get most out of it if you look at the output before/after change - THATs where it will tell you what you messed up most precisely.

If unsure - get on the mailinglist linux-sparsevger.kernel.org

Integrate sparse into your build-env so that you use it continuously - if you wait until you are done - then it will take you out.
sparse
What sparse can detect

- address_space missmatch
- tuple missmatches (i.e. -Wbitwise)
- bad casting (i.e. -Wcast_truncate)
- lock context (i.e. -Wcontext)
- portability warning (i.e. -Wdefault_bitfield_sign)
- big-constants warnings (missing size suffix e.g. LL)

use man sparse for (a lot) more.

Copyright OSADL eG, 2018, CC-BY-SA-4.0
sparse
Example: address_space

```c
__attribute__((address_space(num)))
__user     (1)
__iomem    (2)
__kernel   (default)
```

If code is mixing pointers to different address spaces then you will get a warning. (linux/compile.h)

```c
override with __attribute__((force))
```

If you are converting between address spaces deliberately (i.e. power management does this).
Example: lock context checks

__context__(expression, in_value, out_value)
__acquires(x) __attribute__((context(x,1,0)))
__releases(x) __attribute__((context(x,0,1)))

i.e. __LOCK uses this to check if _spin_lock actually is setting a lock to 1 but never sets it back to 0 (that is acquireing a lock, but not releasing it)
$ cd sparse
$ ./compile compile.c

```
.file  "compile.c"
.text
.type  clean_up_symbols, @function
...
jmp .L2                           # 'break'; go to loop bottom
.L4:                               # end if
.L7:                               # loop top
...
.L13:                               # loop bottom
  movl $0, %eax                    # return
  jmp .L12
.L12:
  addl $164, %esp
ret
.size  main, .-main
.ident  "sparse silly x86 backend (version v0.5.2)"
```

Copyright OSADL eG, 2018, CC-BY-SA-4.0