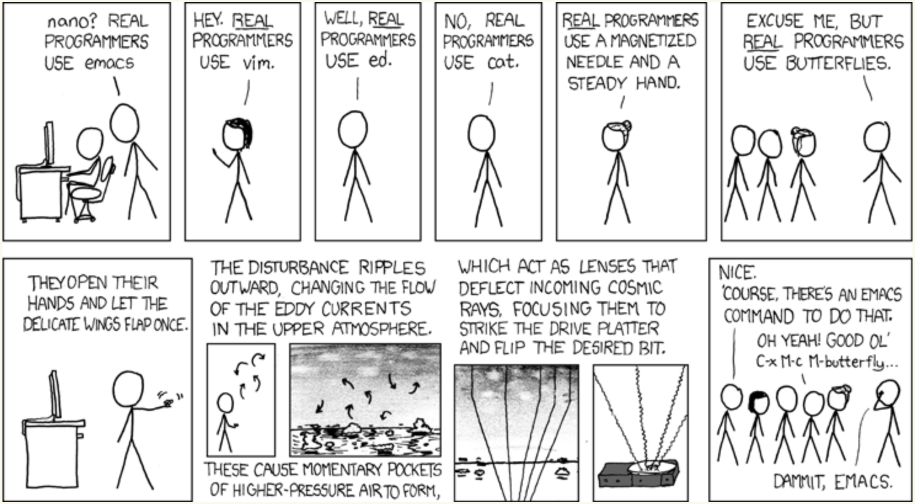
# Linux Kernel Programming
# **Software Engineering Tools and Techniques for the Linux Kernel**

Pierre Olivier

Systems Software Research Group @ Virginia Tech

January 19, 2017


nano? REAL PROGRAMMERS USE emacs
HEY. REAL PROGRAMMERS USE vim.
WELL, REAL PROGRAMMERS USE ed.
NO, REAL PROGRAMMERS USE cat.
REAL PROGRAMMERS USE A MAGNETIZED NEEDLE AND A STEADY HAND.
EXCUSE ME, BUT REAL PROGRAMMERS USE BUTTERFLIES.
THEY OPEN THEIR HANDS AND LET THE DELICATE WINGS FLAP ONCE.
THE DISTURBANCE RIPPLES OUTWARD, CHANGING THE FLOW OF THE EDDY CURRENTS IN THE UPPER ATMOSPHERE.
THESE CAUSE MOMENTARY POCKETS OF HIGHER-PRESSURE AIR TO FORM,
WHICH ACT AS LENSES THAT DEFLECT INCOMING COSMIC RAYS, FOCUSING THEM TO STRIKE THE DRIVE PLATTER AND FLIP THE DESIRED BIT.
NICE. 'COURSE, THERE'S AN EMACS COMMAND TO DO THAT.
OH YEAH! GOOD OL' C-x M-c M-butterfly...
DAMMIT, EMACS.

Source: `https://xkcd.com/378/`

## Outline

## Outline

## Kernel Sources & Compilation

### Getting the kernel sources

- Download the sources here: **`https://www.kernel.org`**
  - Click the large button to get the latest version
  - Want a specific version ?
    - `https://www.kernel.org/pub/` $\rightarrow$ `linux` $\rightarrow$ `kernel` $\rightarrow$ `v`*X*`.`*Y* $\rightarrow$ `linux.`*X*`.`*Y*`.`*Z*`.tar.{gz|xz}`
- Extracting the sources:

```
tar xf linux.X.Y.Z.tar.gz # (Same thing for .tar.xz)
cd linux.X.Y.Z
ls
arch/    crypto/          include/  kernel/      net/             security/
block/   Documentation/   init/     lib/         README           sound/
certs/   drivers/         ipc/      MAINTAINERS  REPORTING-BUGS   tools/
COPYING  firmware/        Kbuild    Makefile     samples/         usr/
CREDITS  fs/              Kconfig   mm/          scripts/         virt/
```

## Kernel Sources & Compilation

### Kernel source directory tree

- Interesting folders [2]:
  - **arch/**: architecture specific code, contains one folder per supported architecture:

```
ls arch/
alpha/  blackfin/  hexagon/  metag/       openrisc/  sh/          x86/
arc/    c6x/       ia64/     microblaze/  parisc/    sparc/       xtensa/
arm/    cris/      Kconfig   mips/        powerpc/   tile/
arm64/  frv/       m32r/     mn10300/     s390/      um/
avr32/  h8300/     m68k/     nios2/       score/     unicore32/
```

    - Example of architecture specific code: boot process, context switch, page table management, etc.
  - **include/**: kernel header files
  - **init/**: kernel initialization code
    - Note that most of the boot process is architecture specific and actually contained in **arch/**
  - **mm/**: memory management
  - **drivers/**: device drivers

## Kernel Sources & Compilation

### Kernel source directory tree (2)

- Interesting folders (continued):
  - **`ipc/`**: Inter-Process Communication
  - **`fs/`**: filesystems
  - **`kernel/`**: generic core kernel
  - **`net/`**: networking
  - **`block/`**: block layer
  - **`lib/`**: helper libraries
  - **`scripts/`**: scripts used during the kernel configuration and compilation process
  - **`Documentation/`**: kernel **documentation** (text files)
  - **`samples/`**: example of usage of some kernel functions/mechanisms
  - **`tools/`**: a set of user space programs, scripts, for various usage (debugging, tracing, performance evaluation, etc.)

## Kernel Sources & Compilation

Compiling and installing the kernel - **Configuration file**

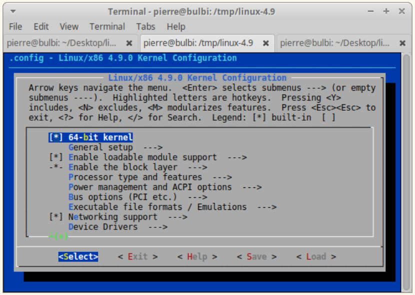
- **Configuration file** defining compilation options ($\sim$ 3500 for x86)
  - Should be present at the root of the source directory and named `.config`
  - Generate the default configuration file for one architecture: `make <arch>_defconfig`
    - Check the default configuration files in `arch/<arch>/configs`
  - Or get the configuration file for an existing kernel running on your target platform
    - Check the kernel version with `uname -a`, then look in `/boot` or `/proc/config.gz`
    - Version of the configuration file older than the target kernel ?
      - `make oldconfig`
      - You will be prompted for each new options
      - Default choice for each new option: `yes "" | make oldconfig`

## Kernel Sources & Compilation

### Compiling and installing the kernel - **Configuration file** (2)

- Edit options: `make menuconfig`
  - Need *libncurses*:

```
sudo apt-get install libncurses5-dev # Debian/Ubuntu
sudo yum install ncurses-devel # Fedora/CentOS/RedHat
```



Terminal - pierre@bulbi: /tmp/linux-4.9
File Edit View Terminal Tabs Help
pierre@bulbi: ~/Desktop/li...
pierre@bulbi: /tmp/linux-4.9
pierre@bulbi: ~/Desktop/li...

```
.config - Linux/x86 4.9.0 Kernel Configuration
                Linux/x86 4.9.0 Kernel Configuration
 Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty
 submenus ----).  Highlighted letters are hotkeys.  Pressing <Y>
 includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to
 exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ]

     [*] 64-bit kernel
         General setup  --->
     [*] Enable loadable module support  --->
     -*- Enable the block layer  --->
         Processor type and features  --->
         Power management and ACPI options  --->
         Bus options (PCI etc.)  --->
         Executable file formats / Emulations  --->
     [*] Networking support  --->
         Device Drivers  --->
     ┴(+)

       <Select>    < Exit >    < Help >    < Save >    < Load >
```

- Search: type `/`
- Help: type `?`
- `[*]` $\rightarrow$ selected,
  `[M]` $\rightarrow$ module

## Kernel Sources & Compilation

### Compiling and installing the kernel - **Compilation and installation**

1. The `.config` file should be ready
2. Compile the kernel: `make bzImage` (x86)
   - The uncompressed kernel binary is `vmlinux`
   - The compressed one (x86) is `arch/x86/boot/bzImage`
3. Compile the modules: `make modules`
4. Installation:

```
sudo make modules_install
sudo make install
```

   + update bootloader configuration (Ubuntu: `sudo update-grub`)

- **Use the parallel build feature of make!**

```
make <target> -j<number of cores>
```

  - Xeon E5-2695:
    - `make bzImage` (equivalent to `make -j1 bzImage`): 12m50s
    - `make bzImage -j2`: 6m48s
    - `make bzImage -j24`: 44s

## Kernel Sources & Compilation

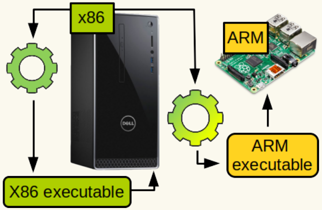
Compiling and installing the kernel - **Compilation and installation: summary**

```
# cd into Linux sources directory:
cd linux-4.9

# Generate default configuration file:
make x86_64_defconfig

# Or use an existing one (potentially older version):
cp -f /boot/config-4.8.0-32-generic .config && yes "" | make oldconfig

# compile the kernel and modules:
make -j4 bzImage
make -j4 modules

# install modules and kernel:
sudo make modules_install
sudo make install

# Update bootloader configuration:
sudo update-grub

# Reboot into new kernel:
sudo reboot
```

## Kernel Sources & Compilation

### Compiling and installing the kernel - **Cross-compiling**

- Compiling on one arch (host) and producing a binary for another (target)
- Embedded systems development
- Cross compiler toolchain: for example `arm-linux-{gcc, ld, ar}`, etc.



- When cross-compiling Linux, the following environment variables are accessed by `make` and must be set:
  - `ARCH`: target architecture
  - `CROSS_COMPILE`: cross-compiler toolchain *prefix*
    - For example with `arm-linux-gcc`, the prefix is `arm-linux-`

## Kernel Sources & Compilation

Compiling and installing the kernel - **Cross-compiling (2)**

- Two solution for setting the environment variables:
  1. With each `make` invocation

```
ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- make defconfig
ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- make menuconfig
# etc.
```

     - Not a very good idea as when one forgot to set these variables, they default to the native environment leading to inconsistent build

  2. Exported in the shell

```
export ARCH=arm64
export CROSS_COMPILE=aarch64-linux-gnu-
make defconfig
make menuconfig
# etc.
```

- Installing a cross-compiled kernel, or a kernel on an ”exotic” platform:
  - Very platform/distribution-dependent

## Outline

## Exploring the code

### Compiling and installing the kernel - **Tools list**

1. Linux Cross Reference
2. Cscope
3. Graphical IDEs
4. Text-based IDEs

## Exploring the code
### Linux Cross Reference

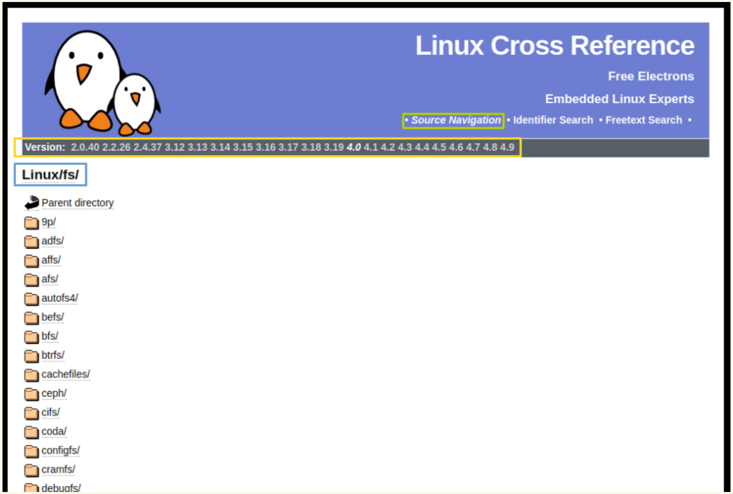
- Code indexing tool [1] with a web interface
  - Don't install it! One instance is running here:

    `http://lxr.free-electrons.com/`

- Allows to:
  - Browse the code of different Linux versions
  - Search for identifiers (functions, variables, etc.)
  - Quickly lookup a function declaration/definition

# Exploring the code

## Linux Cross Reference (2)



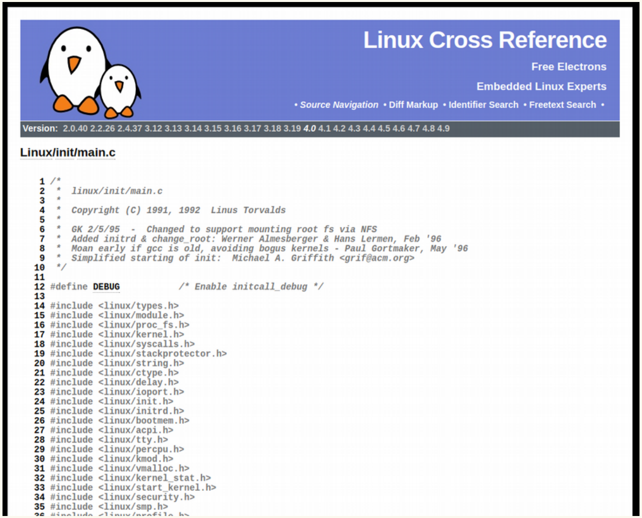
Version: 2.0.40 2.2.26 2.4.37 3.12 3.13 3.14 3.15 3.16 3.17 3.18 3.19 4.0 4.1 4.2 4.3 4.4 4.5 4.6 4.7 4.8 4.9

Linux/fs/

Parent directory
9p/
adfs/
affs/
afs/
autofs4/
befs/
bfs/
btrfs/
cachefiles/
ceph/
cifs/
coda/
configfs/
cramfs/
debugfs/

# Exploring the code

## Linux Cross Reference (2)

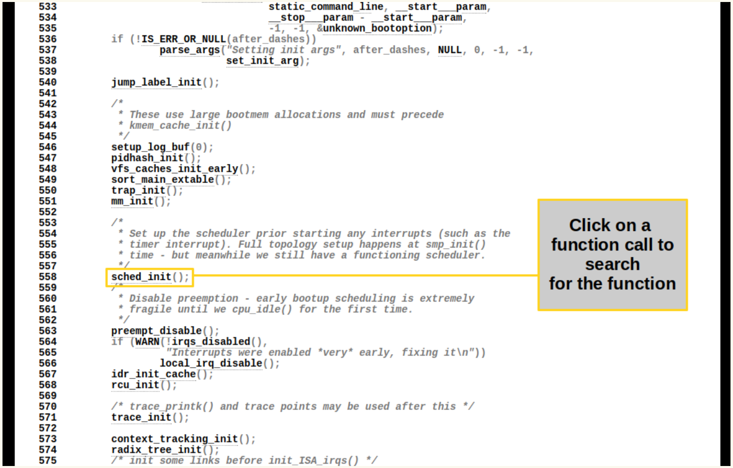

Linux Cross Reference
Free Electrons
Embedded Linux Experts
• Source Navigation • Diff Markup • Identifier Search • Freetext Search •
Version: 2.0.40 2.2.26 2.4.37 3.12 3.13 3.14 3.15 3.16 3.17 3.18 3.19 4.0 4.1 4.2 4.3 4.4 4.5 4.6 4.7 4.8 4.9

Linux/init/main.c

```c
/*
 *  linux/init/main.c
 *
 *  Copyright (C) 1991, 1992  Linus Torvalds
 *
 *  GK 2/5/95  -  Changed to support mounting root fs via NFS
 *  Added initrd & change_root: Werner Almesberger & Hans Lermen, Feb '96
 *  Moan early if gcc is old, avoiding bogus kernels - Paul Gortmaker, May '96
 *  Simplified starting of init:  Michael A. Griffith <grif@acm.org>
 */

#define DEBUG           /* Enable initcall_debug */

#include <linux/types.h>
#include <linux/module.h>
#include <linux/proc_fs.h>
#include <linux/kernel.h>
#include <linux/syscalls.h>
#include <linux/stackprotector.h>
#include <linux/string.h>
#include <linux/ctype.h>
#include <linux/delay.h>
#include <linux/ioport.h>
#include <linux/init.h>
#include <linux/initrd.h>
#include <linux/bootmem.h>
#include <linux/acpi.h>
#include <linux/tty.h>
#include <linux/percpu.h>
#include <linux/kmod.h>
#include <linux/vmalloc.h>
#include <linux/kernel_stat.h>
#include <linux/start_kernel.h>
#include <linux/security.h>
#include <linux/smp.h>
```

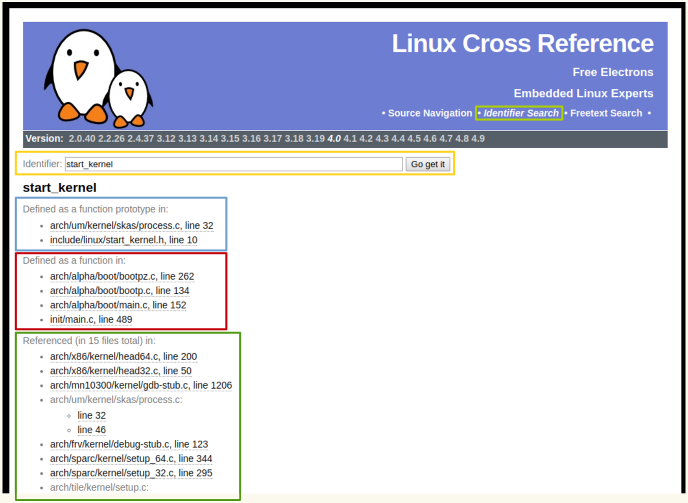
# Exploring the code

## Linux Cross Reference (2)



```
                                static_command_line, __start___param,
                                __stop___param - __start___param,
                                -1, -1, &unknown_bootoption);
        if (!IS_ERR_OR_NULL(after_dashes))
                parse_args("Setting init args", after_dashes, NULL, 0, -1, -1,
                           set_init_arg);

        jump_label_init();

        /*
         * These use large bootmem allocations and must precede
         * kmem_cache_init()
         */
        setup_log_buf(0);
        pidhash_init();
        vfs_caches_init_early();
        sort_main_extable();
        trap_init();
        mm_init();

        /*
         * Set up the scheduler prior starting any interrupts (such as the
         * timer interrupt). Full topology setup happens at smp_init()
         * time - but meanwhile we still have a functioning scheduler.
         */
        sched_init();
        /*
         * Disable preemption - early bootup scheduling is extremely
         * fragile until we cpu_idle() for the first time.
         */
        preempt_disable();
        if (WARN(!irqs_disabled(),
                 "Interrupts were enabled *very* early, fixing it\n"))
                local_irq_disable();
        idr_init_cache();
        rcu_init();

        /* trace_printk() and trace points may be used after this */
        trace_init();

        context_tracking_init();
        radix_tree_init();
        /* init some links before init_ISA_irqs() */
```

## Exploring the code
### Linux Cross Reference (2)


Linux Cross Reference
Free Electrons
Embedded Linux Experts
• Source Navigation • Identifier Search • Freetext Search •
Version: 2.0.40 2.2.26 2.4.37 3.12 3.13 3.14 3.15 3.16 3.17 3.18 3.19 4.0 4.1 4.2 4.3 4.4 4.5 4.6 4.7 4.8 4.9
Identifier: start_kernel
Go get it

**start_kernel**

Defined as a function prototype in:

- arch/um/kernel/skas/process.c, line 32
- include/linux/start_kernel.h, line 10

Defined as a function in:

- arch/alpha/boot/bootpz.c, line 262
- arch/alpha/boot/bootp.c, line 134
- arch/alpha/boot/main.c, line 152
- init/main.c, line 489

Referenced (in 15 files total) in:

- arch/x86/kernel/head64.c, line 200
- arch/x86/kernel/head32.c, line 50
- arch/mn10300/kernel/gdb-stub.c, line 1206
- arch/um/kernel/skas/process.c:
  - line 32
  - line 46
- arch/frv/kernel/debug-stub.c, line 123
- arch/sparc/kernel/setup_64.c, line 344
- arch/sparc/kernel/setup_32.c, line 295
- arch/tile/kernel/setup.c:

## Exploring the code

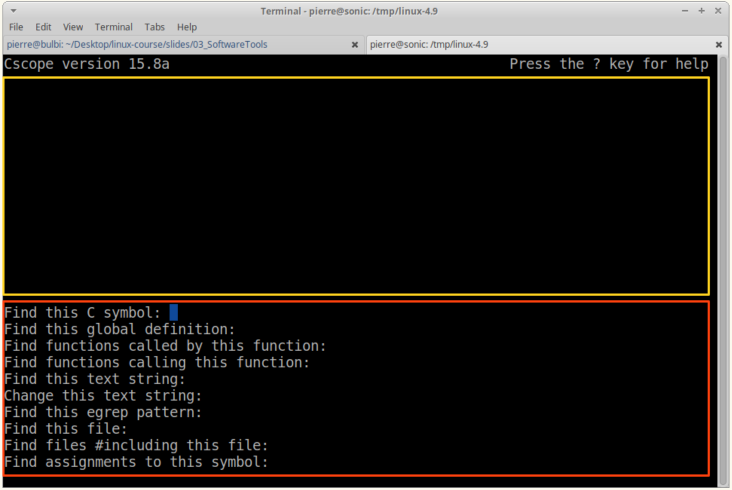
### Cscope

- Command line tool to browse (potentially large) C codebases
- Installation: `sudo {apt-get|yum} install cscope`
- Usage:

```
cd <linux source directory>
make cscope
# or:
ARCH=x86 make cscope
# The regular way is to use cscope -R but this Makefile target is optimized for the
      kernel source code
```

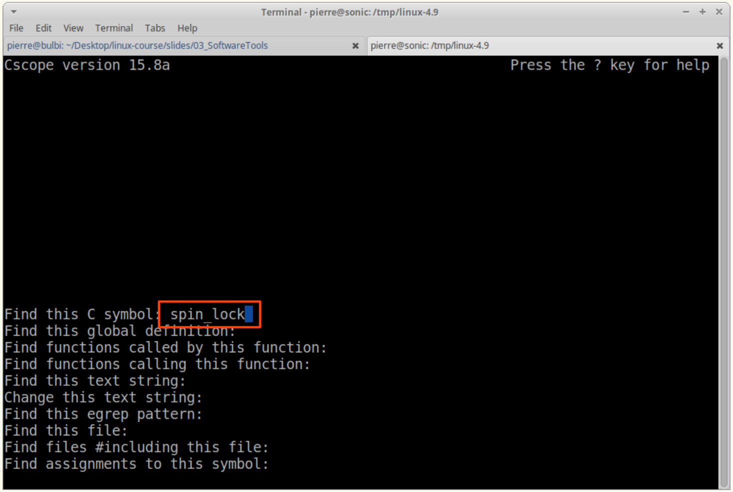
- Building the database takes a few seconds on the first run or each time the code changes
- Search for:
  - C identifier occurrences (variable name, function name, typedef/struct, label)
  - Functions/variables definitions
  - Functions called by/calling function *f*
  - Text string

## Exploring the code
### Cscope (2)


Terminal - pierre@sonic: /tmp/linux-4.9
File Edit View Terminal Tabs Help
pierre@bulbi: ~/Desktop/linux-course/slides/03_SoftwareTools
pierre@sonic: /tmp/linux-4.9

```
Cscope version 15.8a                                  Press the ? key for help

Find this C symbol:
Find this global definition:
Find functions called by this function:
Find functions calling this function:
Find this text string:
Change this text string:
Find this egrep pattern:
Find this file:
Find files #including this file:
Find assignments to this symbol:
```

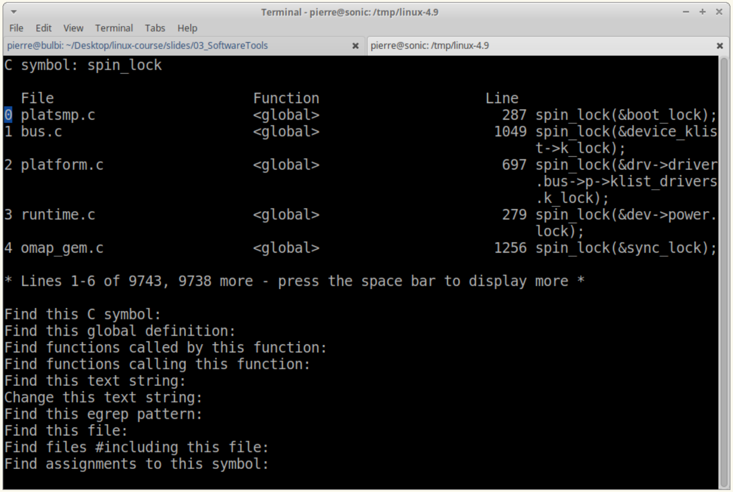
## Exploring the code
Cscope (2)


Terminal - pierre@sonic: /tmp/linux-4.9
File Edit View Terminal Tabs Help
pierre@bulbi: ~/Desktop/linux-course/slides/03_SoftwareTools
pierre@sonic: /tmp/linux-4.9

```
Cscope version 15.8a                                       Press the ? key for help









Find this C symbol: spin_lock
Find this global definition:
Find functions called by this function:
Find functions calling this function:
Find this text string:
Change this text string:
Find this egrep pattern:
Find this file:
Find files #including this file:
Find assignments to this symbol:
```

# Exploring the code

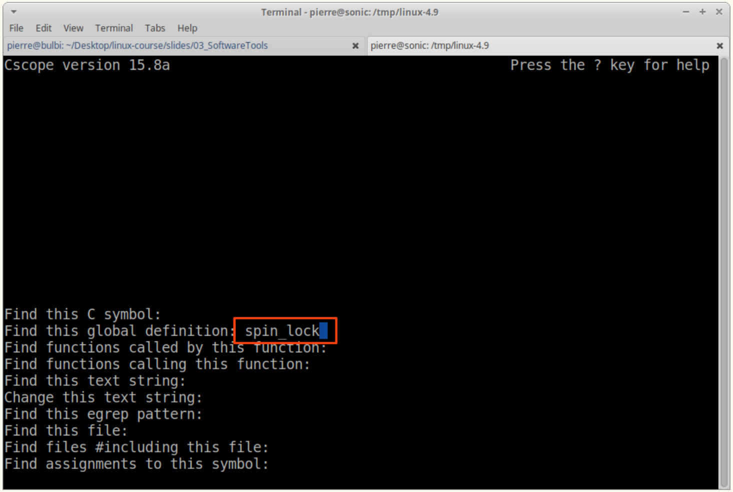
## Cscope (2)


Terminal - pierre@sonic: /tmp/linux-4.9
File Edit View Terminal Tabs Help
pierre@bulbi: ~/Desktop/linux-course/slides/03_SoftwareTools
pierre@sonic: /tmp/linux-4.9

```
C symbol: spin_lock

  File                        Function                    Line
0 platsmp.c                   <global>                     287 spin_lock(&boot_lock);
1 bus.c                       <global>                    1049 spin_lock(&device_klis
                                                               t->k_lock);
2 platform.c                  <global>                     697 spin_lock(&drv->driver
                                                               .bus->p->klist_drivers
                                                               .k_lock);
3 runtime.c                   <global>                     279 spin_lock(&dev->power.
                                                               lock);
4 omap_gem.c                  <global>                    1256 spin_lock(&sync_lock);

* Lines 1-6 of 9743, 9738 more - press the space bar to display more *

Find this C symbol:
Find this global definition:
Find functions called by this function:
Find functions calling this function:
Find this text string:
Change this text string:
Find this egrep pattern:
Find this file:
Find files #including this file:
Find assignments to this symbol:
```

# Exploring the code

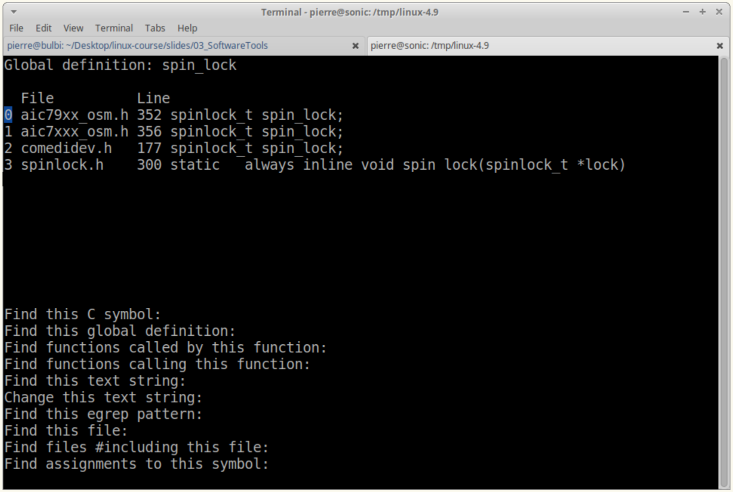
## Cscope (2)


Terminal - pierre@sonic: /tmp/linux-4.9
File Edit View Terminal Tabs Help
pierre@bulbi: ~/Desktop/linux-course/slides/03_SoftwareTools
pierre@sonic: /tmp/linux-4.9

```
Cscope version 15.8a                                        Press the ? key for help


Find this C symbol:
Find this global definition: spin_lock
Find functions called by this function:
Find functions calling this function:
Find this text string:
Change this text string:
Find this egrep pattern:
Find this file:
Find files #including this file:
Find assignments to this symbol:
```

# Exploring the code

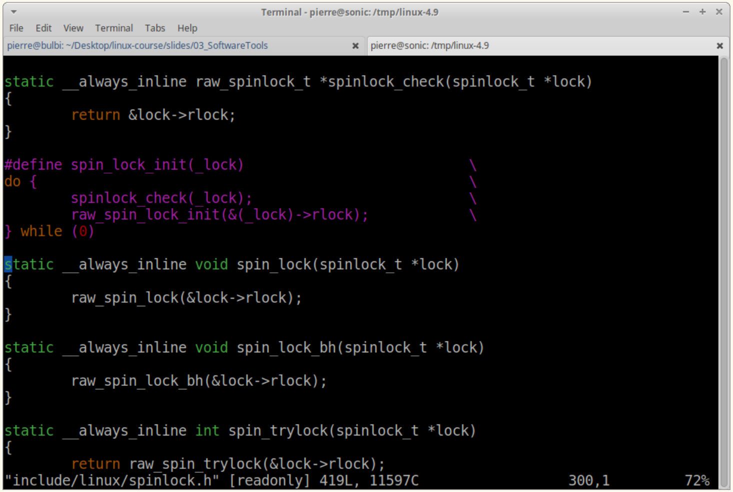
## Cscope (2)


Terminal - pierre@sonic: /tmp/linux-4.9
File Edit View Terminal Tabs Help
pierre@bulbi: ~/Desktop/linux-course/slides/03_SoftwareTools
pierre@sonic: /tmp/linux-4.9

```
Global definition: spin_lock

  File          Line
0 aic79xx_osm.h 352 spinlock_t spin_lock;
1 aic7xxx_osm.h 356 spinlock_t spin_lock;
2 comedidev.h   177 spinlock_t spin_lock;
3 spinlock.h    300 static __always_inline void spin_lock(spinlock_t *lock)









Find this C symbol:
Find this global definition:
Find functions called by this function:
Find functions calling this function:
Find this text string:
Change this text string:
Find this egrep pattern:
Find this file:
Find files #including this file:
Find assignments to this symbol:
```

# Exploring the code

## Cscope (2)


Terminal - pierre@sonic: /tmp/linux-4.9
File Edit View Terminal Tabs Help
pierre@bulbi: ~/Desktop/linux-course/slides/03_SoftwareTools
pierre@sonic: /tmp/linux-4.9

```
static __always_inline raw_spinlock_t *spinlock_check(spinlock_t *lock)
{
        return &lock->rlock;
}

#define spin_lock_init(_lock)                           \
do {                                                    \
        spinlock_check(_lock);                          \
        raw_spin_lock_init(&(_lock)->rlock);            \
} while (0)

static __always_inline void spin_lock(spinlock_t *lock)
{
        raw_spin_lock(&lock->rlock);
}

static __always_inline void spin_lock_bh(spinlock_t *lock)
{
        raw_spin_lock_bh(&lock->rlock);
}

static __always_inline int spin_trylock(spinlock_t *lock)
{
        return raw_spin_trylock(&lock->rlock);
"include/linux/spinlock.h" [readonly] 419L, 11597C             300,1         72%
```

## Exploring the code
### Other code browsing tools

- OpenGrok: `https://opengrok.github.io/OpenGrok/`
- GrokBit: `https://grokbit.com/`
- Plenty of IDEs have code browsing functionalities

## Outline

## Coding
Development Environment

Development Machine:

- Should run Linux as the OS (**Ubuntu**, Debian, Fedora, etc.)
- Development can be done natively or inside a virtual machine
- Machine specs:
  - Uncompressed kernel sources: 762M for Linux 4.9
  - Compiled (Debian default config): 11G
  - 1 CPU and 256MB of ram is sufficient ...
  - ... however more cores (`-j` flag for make) and RAM allows to compile faster
- `ccache` can also speed up the compilation
  - `{apt-get|yum} install ccache`
  - `http://askubuntu.com/questions/470545/how-do-i-set-up-ccache`
- Testing (running a recently modified kernel): **should be done in a virtual machine**

## Coding
### Development Environment (2)

- Editors:
  1. Your regular text editor!
     - *Graphical:* gedit, geany, emacs, kate, etc.
     - *Console-based:* **vim**, nano, emacs, etc.
     - `sudo {apt-get|yum} install <name>`
  2. More complex/complete graphical IDEs:
     - Eclipse, Netbeans, CLion, Visual Studio Code, etc.
     - Not recommended
     - Some of these have interesting code-browsing functions
     - However code indexing for some is disturbed by the large codebase of the kernel

## Coding

### Vim configuration for kernel coding/code browsing

- Vim can use the tag database of `cscope`, as well as `ctags`
  - `sudo apt-get install cscope exuberant-ctags`
  - `yum install cscope ctags`
- Generate the databases:

```
cd <linux source dir>
make cscope tags -j2
```

- Launch vim:

```
vim init/main.c
```

- Search for function definition/variable declaration:

```
:tag start_kernel
:cs find global start_kernel
```

- Command usage:

```
:help tag
:help cs
```

## Coding
### Vim configuration for kernel coding/code browsing (2)

- Another way to find a function definition/variable declaration:
  - Put the cursor on the symbol name and press ctrl + ]
- To navigate back and forth between file:

```
:bp
:bn
```

- More info:
  http://stackoverflow.com/questions/33676829/vim-configuration-for-linux-kernel-development

## Coding
### Coding style

- Standard coding style for the kernel:
  - ***consistency*** is important to help understanding the code (and grading projects ;))
  - Details: textbook chapter 20 + Linux `Documentation/CodingStyle`
  - **Indentation**: tabs, 8 characters
  - **`switch`**: no need to indent `cases`
  - **spaces**: `if (!x); func_call(a + b);`
  - **braces**: opening: same line, closing: new line
  - **line length**: 80 characters
  - **naming**: no *CamelCase*, use underscores
  - **comments**: C-style (`/* comment */`, no C++ `// comment`)
  - **typedefs**: avoid them
  - **`#ifdef`**: minimize them

## Coding
### Coding style (2)

- **indent**:

```
indent -kr -i8 -ts8 -sob -l80 -ss -bs -psl <file>
```

or look in the kernel sources in `scripts/Lindent` to automatically invoke that command

# Coding

## Coding style (3)

```c
/*
 * a multi-lines comment
 * (no C++ '//' !)
 */

struct foo {
        int member1;
        double member2;
}; /* no typedef ! */

#ifdef CONFIG_COOL_OPTION
int cool_function(void) {
  return 42;
}
#else
int cool_function(void) { }
#endif /* CONFIG_COOL_OPTION */


void my_function(int the_param, char *
     string, int a_long_parameter,
    int another_long_parameter) {
        int x = the_param % 42;

  if (!the_param)
          do_stuff();

  switch (x % 3) {
  case 0:
    do_some_stuff();
    cool_function();
    break;
  case 1:
    /* Fall through */
  default:
    do_other_stuff();
    cool_function();
  }
}
```

- Strict adherence to the kernel code-style is not asked for the projects in this course
  - However, common sense and **consistency** will be evaluated

## Outline

## Version control with Git
### Coding style

- **Git** is a *Version Control* Software (VCS)
- Initially developed by Linus Torvalds
- Extensively used by the Linux community
  - Manage changes made to a codebase, ease the job of one or several programmers working on the same project
    - Code is maintained on a **server**: (not so) centralized codebase
    - Each programmer downloads a **local copy** for modification and **propagate their changes** through atomic actions
    - No deletion, **history maintained**: you can roll-back in case of trouble
    - Git also helps in solving issues that arise with **two programmers working on the same file**
- Git (software) $\neq$ Github (provider)!

## Outline

## Sources of information about Linux

- Books:
  - **Love, R. (2010). *Linux Kernel Development, 3rd Edition*. Addison-Wesley Professional. Pp. xxv, 440.**
  - Bovet, D. P., & Cesati, M. (2005). *Understanding the Linux Kernel, 3rd Edition*. O'Reilly Media. Pp. xvi, 944;
  - Corbet, J., Rubini, A., & Kroah-Hartman, G. (2005). *Linux Device Drivers, 3rd Edition*. O'Reilly Media. Pp xvii, 640;
  - Mauerer, W. (2008). *Professional Linux Kernel Architecture, 1st Edition*. Wrox. Pp. xxx, 1368;
  - Love, R. (2013). *Linux System Programming: Talking Directly to the Kernel and C Library, 2nd Edition*. O'Reilly Media. Pp. xx, 456.

## Sources of information about Linux (2)

- Linux weekly news: `https://lwn.net`
- Kernel mailing lists: `http://vger.kernel.org/vger-lists.html`
- Linux-insides: `https://0xax.gitbooks.io/linux-insides/content/index.html`
  - Not comprehensive, but *relatively* recent information (Linux 3.18)
- Wikis hosted on kernel.org: `https://www.wiki.kernel.org/`
  - Interesting info about filesystems, git, perf, etc.
- Kernel newbies: `https://kernelnewbies.org/`
  - Guides on kernel development

## Bibliography I

[1] Linux cross reference official website.
http://lxr.linux.no/.
Accessed: 2016-12-28.

[2] Tldp - the linux kernel sources.
http://www.tldp.org/LDP/tlk/sources/sources.html.
Accessed: 2016-12-27.