

How to use Sourcetrail for kernel analysis

ELISA Workshop Lightning Talks
Feb. 3, 2021

Naoto Yamaguchi
Doctor of Informatics, Specialist
Software Fundamental Technology Group
Application Design Department
Connected & Sharing Solutions Division
AISIN AW CO.,LTD.



ENABLING LINUX IN SAFETY APPLICATIONS



What is Sourcetrail

- Free and open-source cross-platform source explorer.
- It support C, C++, Java, Python language.
- It can run **linux**, windows and Mac OS.

Home Blog Documentation GitHub Become a Patron Download

en / de

SOURCETRIL
Free and open-source cross-platform source explorer

Get productive on unfamiliar source code.

► Play Intro Download

NEWS: Sourcetrail 2020.2 now available for download! [Read more](#)

<https://www.sourcetrail.com/>

What we can?

- We can check call graph and sources by GUI tool.

The screenshot displays a GUI tool interface for analyzing code. The left pane shows a call graph for the function 'up'. The graph has a central node 'up' with several incoming edges from nodes: '___addressable_up189 (semaphore.c)', '___up_console_sem', 'seccomp_do_user_notification', and 'seccomp_notify_rcv'. The 'up' node has outgoing edges to '___up', '___raw_spin_lock_irqsave', and '___raw_spin_unlock_irqrestore'. A 'Non-indexed Symbols' box is also visible. The right pane shows the source code for 'semaphore.c' and 'printk.c'. The 'semaphore.c' code shows the definition of the 'up' function, which increments the semaphore count and releases the lock. The 'printk.c' code shows the use of 'up' in the 'printk_safe_enter_irqsave' function.

```
... semaphore.c
171 /**
172  * up - release the semaphore
173  * @sem: the semaphore to release
174  *
175  * Release the semaphore. Unlike mutexes, up() may be called from any
176  * context and even by tasks which have never called down().
177  */
178 void up(struct semaphore *sem)
179 {
180     unsigned long flags;
181
182     raw_spin_lock_irqsave(&sem->lock, flags);
183     if (likely(list_empty(&sem->wait_list)))
184         sem->count++;
185     else
186         __up(sem);
187     raw_spin_unlock_irqrestore(&sem->lock, flags);
188 }
189 EXPORT_SYMBOL(up);
190
191 /* Functions for the contended case */
192
```

```
... __up_console_sem
251     mutex_release(&console_lock_dep_map, 1, ip);
252
253     printk_safe_enter_irqsave(flags);
254     up(&console_sem);
255     printk_safe_exit_irqrestore(flags);
256 }
257 #define up_console_sem() __up_console_sem(_RET_IP_)
258
259 /*
260  * This is used for debugging the mess that is the VT code by
261  * keeping track if we have the console semaphore held. It's
262  * definitely not the perfect debug tool (we don't know if _WE_
```

What we can?

- We can check call graph and sources by GUI tool.

The screenshot displays a GUI tool interface for analyzing code. The top part shows a browser-like window with the address bar containing the symbol name `_raw_spin_lock_irqsave`. Below the browser, a call graph is visible, showing a flow from 'Referencing Symbols' (with a count of 262) to the symbol `_raw_spin_lock_irqsave`, which then points to 'Non-indexed Symbols' (with a count of 2).

On the right side, there is a list of references to the symbol, categorized by file:

- spinlock.c** (3 references):
 - Line 154: `#endif`
 - Line 155: `#endif`
 - Line 156: `#ifndef CONFIG_INLINE_SPIN_LOCK_IRQSAVE`
 - Line 157: `unsigned long __lockfunc _raw_spin_lock_irqsave(raw_spinlock_t *lock)`
 - Line 158: `{`
 - Line 159: `return __raw_spin_lock_irqsave(lock);`
 - Line 160: `}`
 - Line 161: `EXPORT_SYMBOL(_raw_spin_lock_irqsave);`
 - Line 162: `#endif`
 - Line 163: `#endif`
 - Line 164: `#ifndef CONFIG_INLINE_SPIN_LOCK_IRQ`
- async.c** (4 references):
 - Line 84: `async_cookie_t ret = ASYNC_COOKIE_MAX;`
 - Line 85: `unsigned long flags;`
 - Line 86: `spin_lock_irqsave(&async_lock, flags);`
 - Line 87: `spin_lock_irqsave(&async_lock, flags);`
 - Line 88: `spin_lock_irqsave(&async_lock, flags);`
 - Line 89: `if (domain) {`
 - Line 90: `if (!list_empty(&domain->pending))`
 - Line 131: `}`
 - Line 132: `}`
 - Line 133: `/* 2) remove self from the pending queues */`
 - Line 134: `spin_lock_irqsave(&async_lock, flags);`
 - Line 135: `list_del_init(&entry->domain_list);`
 - Line 136: `list_del_init(&entry->global_list);`
 - Line 137: `list_del_init(&entry->global_list);`
 - Line 138: `list_del_init(&entry->global_list);`
 - Line 139: `list_del_init(&entry->global_list);`
 - Line 140: `list_del_init(&entry->global_list);`
 - Line 141: `list_del_init(&entry->global_list);`
 - Line 142: `list_del_init(&entry->global_list);`
 - Line 143: `list_del_init(&entry->global_list);`
 - Line 144: `list_del_init(&entry->global_list);`
 - Line 145: `list_del_init(&entry->global_list);`
 - Line 146: `list_del_init(&entry->global_list);`
 - Line 147: `list_del_init(&entry->global_list);`
 - Line 148: `list_del_init(&entry->global_list);`
 - Line 149: `list_del_init(&entry->global_list);`
 - Line 150: `list_del_init(&entry->global_list);`
 - Line 151: `list_del_init(&entry->global_list);`
 - Line 152: `list_del_init(&entry->global_list);`
 - Line 153: `list_del_init(&entry->global_list);`
 - Line 154: `list_del_init(&entry->global_list);`
 - Line 155: `list_del_init(&entry->global_list);`
 - Line 156: `list_del_init(&entry->global_list);`
 - Line 157: `list_del_init(&entry->global_list);`
 - Line 158: `list_del_init(&entry->global_list);`
 - Line 159: `list_del_init(&entry->global_list);`
 - Line 160: `list_del_init(&entry->global_list);`
 - Line 161: `list_del_init(&entry->global_list);`
 - Line 162: `list_del_init(&entry->global_list);`
 - Line 163: `list_del_init(&entry->global_list);`
 - Line 164: `list_del_init(&entry->global_list);`
 - Line 165: `list_del_init(&entry->global_list);`
 - Line 166: `list_del_init(&entry->global_list);`
 - Line 167: `list_del_init(&entry->global_list);`
 - Line 168: `list_del_init(&entry->global_list);`
 - Line 169: `list_del_init(&entry->global_list);`
 - Line 170: `list_del_init(&entry->global_list);`
 - Line 171: `list_del_init(&entry->global_list);`
 - Line 172: `list_del_init(&entry->global_list);`
 - Line 173: `list_del_init(&entry->global_list);`
 - Line 174: `list_del_init(&entry->global_list);`
 - Line 175: `list_del_init(&entry->global_list);`
 - Line 176: `list_del_init(&entry->global_list);`
 - Line 177: `list_del_init(&entry->global_list);`
 - Line 178: `list_del_init(&entry->global_list);`

How to use

- Build kernel using bear
 - Command: **bear make -j8**
- Issue
 - When I try to analyze to all of kernel, it's too heavy in my PC.
 - I recommend to analyze to partially.
 - Command: **bear make net/ -j8**
- After that
 - We can get “compile_commands.json” file.

How to use

- Down load

- <https://github.com/CoatiSoftware/Sourcetrail/releases>
- wget
https://github.com/CoatiSoftware/Sourcetrail/releases/download/2020.4.35/Sourcetrail_2020_4_35_Linux_64bit.tar.gz

- Install

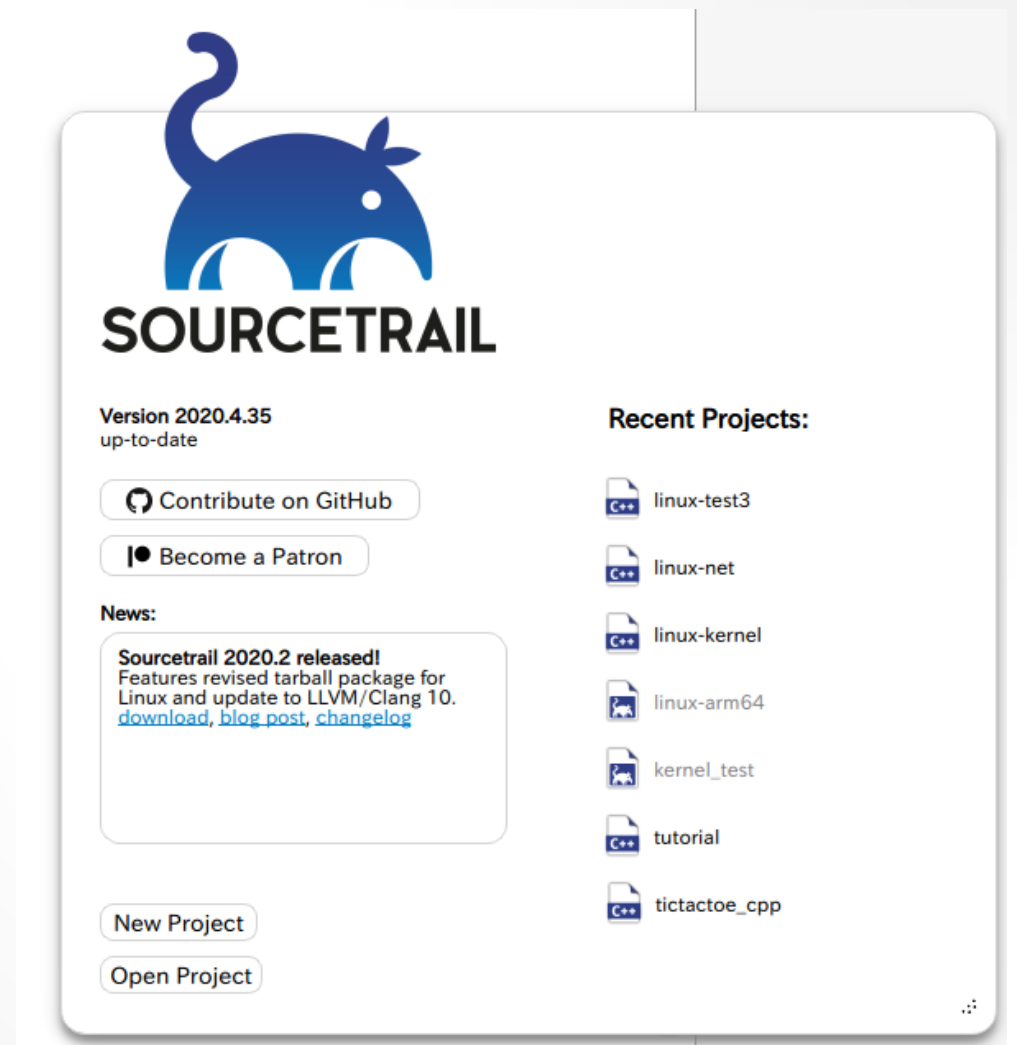
- `tar xvzf Sourcetrail_2020_4_35_Linux_64bit.tar.gz`
- `cd Sourcetrail`
- `sudo install.sh`

- Run

- `/opt/sourcetrail/Sourcetrail.sh`

How to use

- When we success to run, we can see this menu.
- Select a “New Project”



How to use

- Input project name and project location.
- Push “Add Source Group”

Sourcetrail

New Project

Overview

General

Source Groups:

Sourcetrail Project Name* linux-test-example

Sourcetrail Project Location* /home/user/test/projects

* required

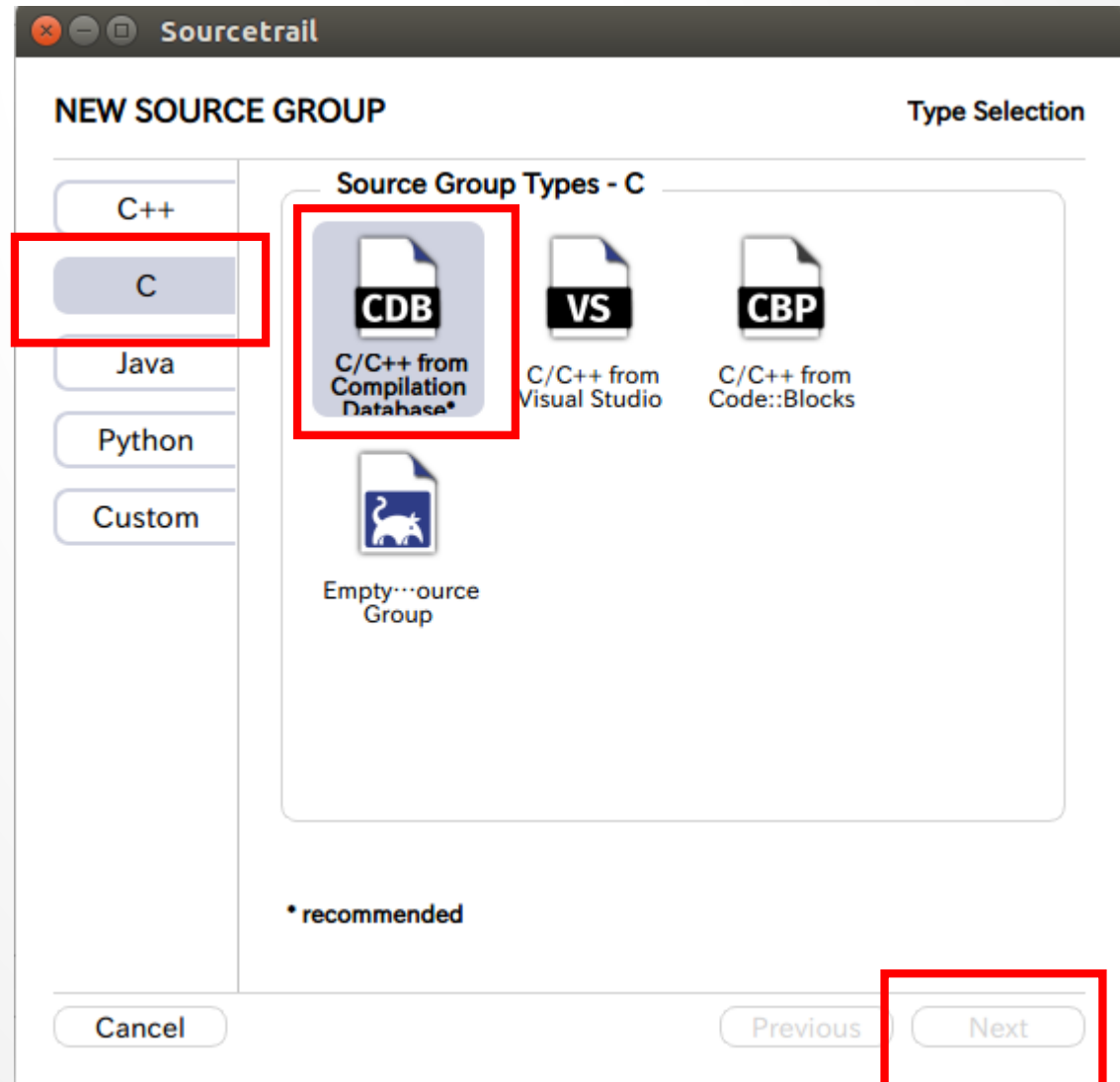
Please add at least one source group to your project. A source group specifies which source files should be analyzed by Sourcetrail and includes all parameters required to analyze those source files. A Sourcetrail project may contain multiple source groups, which may be necessary if you want to analyze source files from different projects that do not share the same parameters.

Hint: If your project contains source code for multiple build targets, you can add all of those source files with one single source group as long as they all share the same parameters.

Cancel Add Source Group Create

How to use

- Select “C”
- Select “C/C++ from Compilation Database”
- Push “Next”



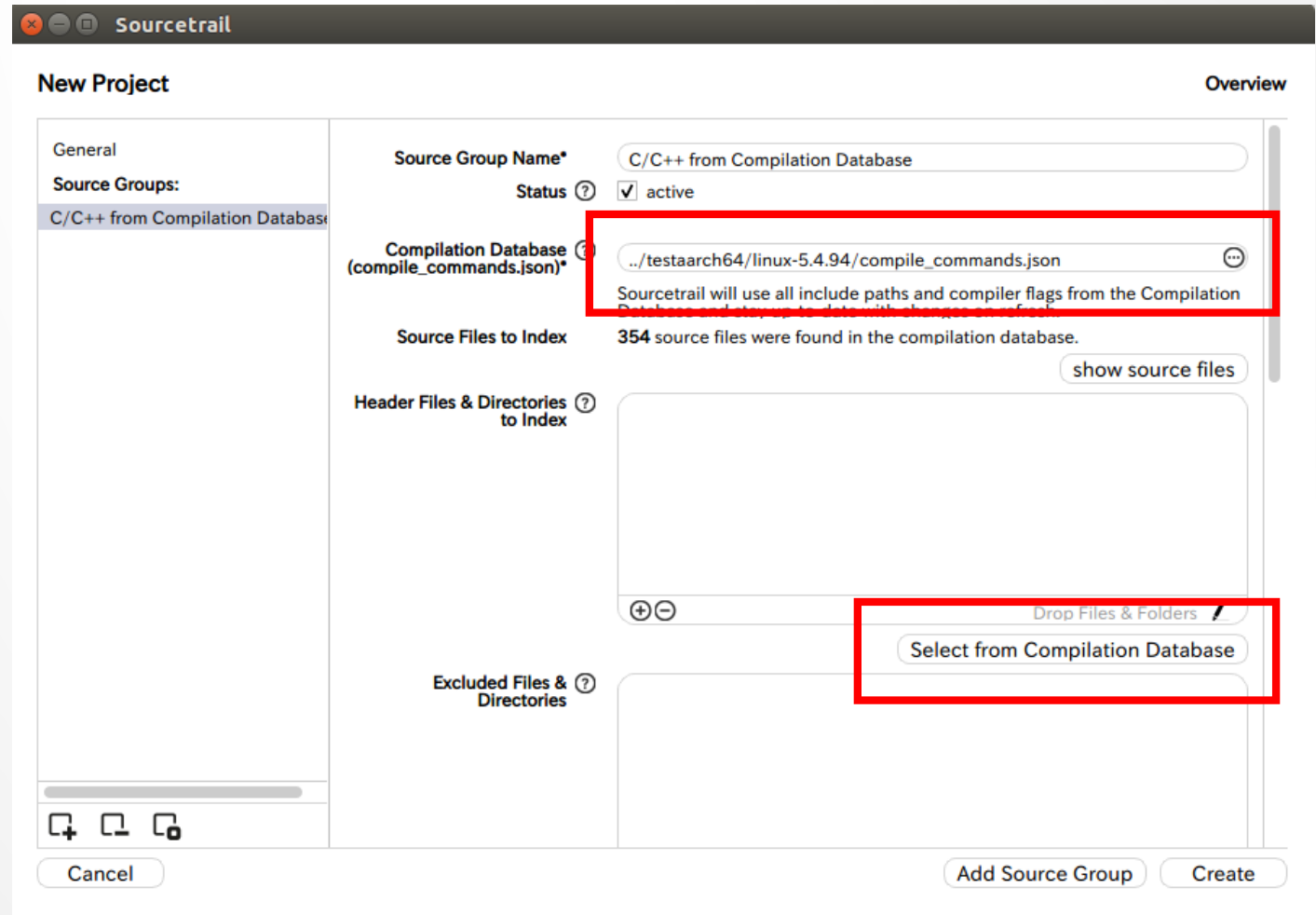
How to use

- Set existing “compile_commands.json” at kernel source.

- Push “Select from Compilation Database”

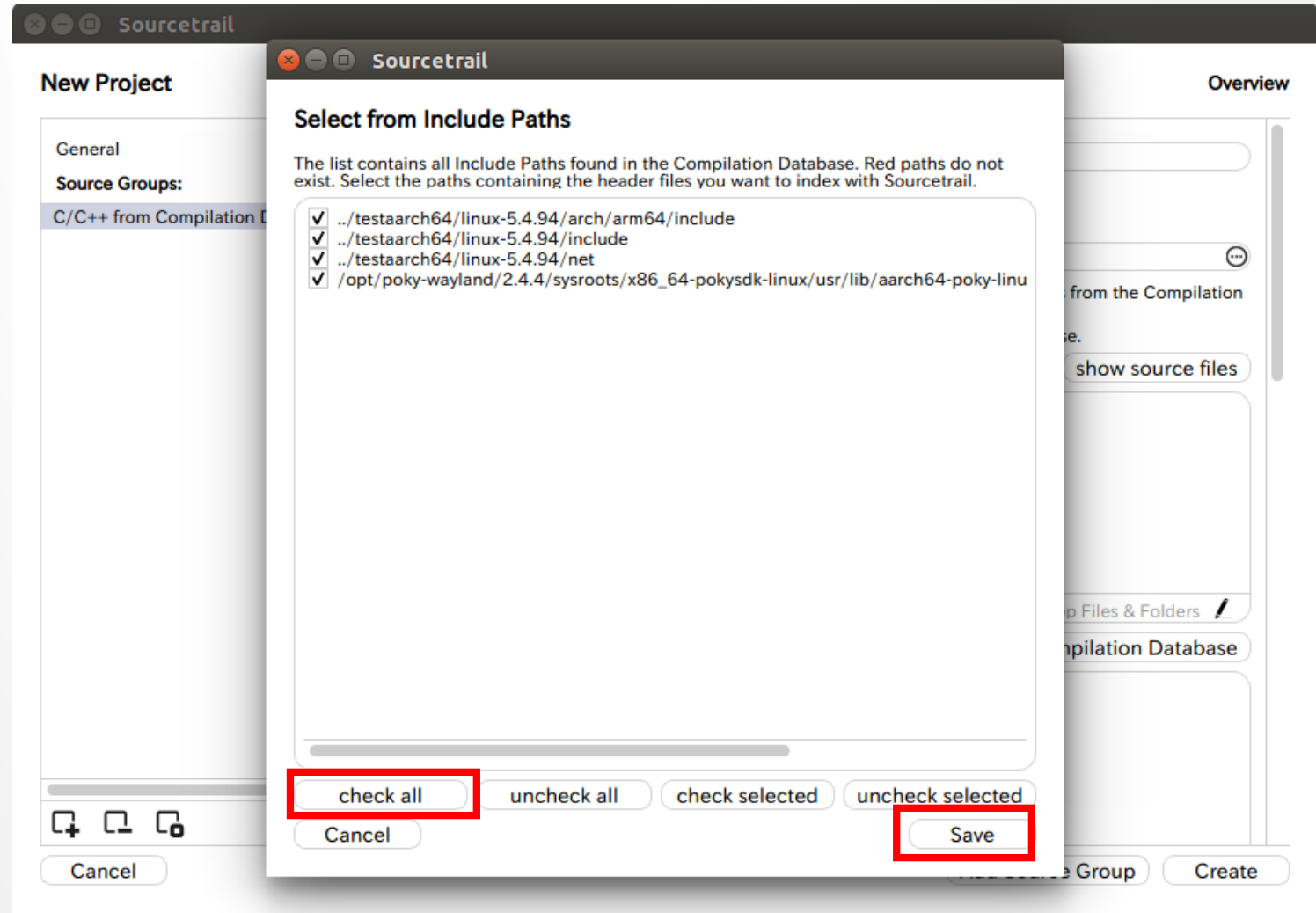
- Push “check all” and “save” in sub menu.

- Push “Create”



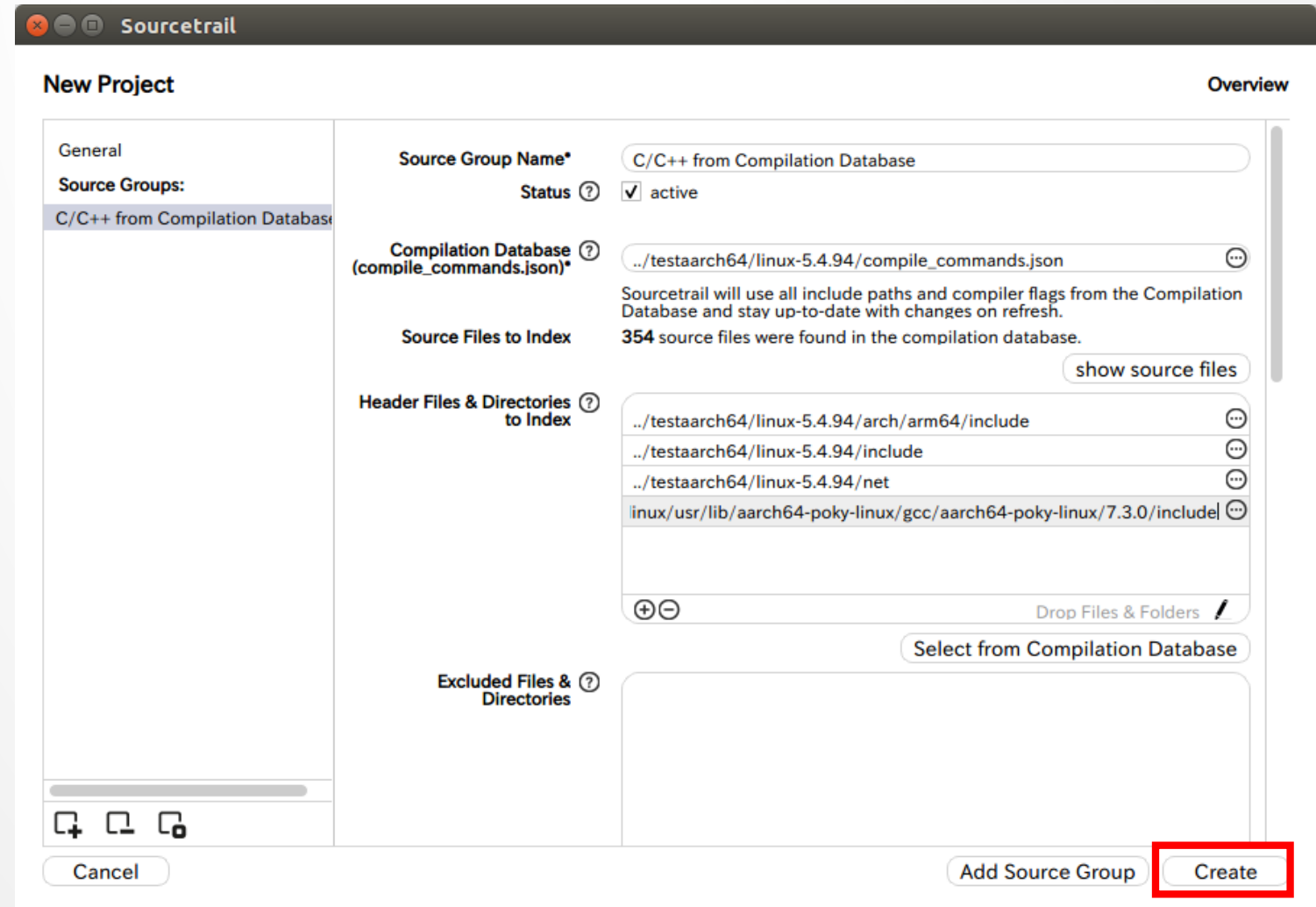
How to use

- Set existing “compile_commands.json” at kernel source.
- Push “Select from Compilation Database”
- Push “check all” and “save” in sub menu.
- Push “Create”



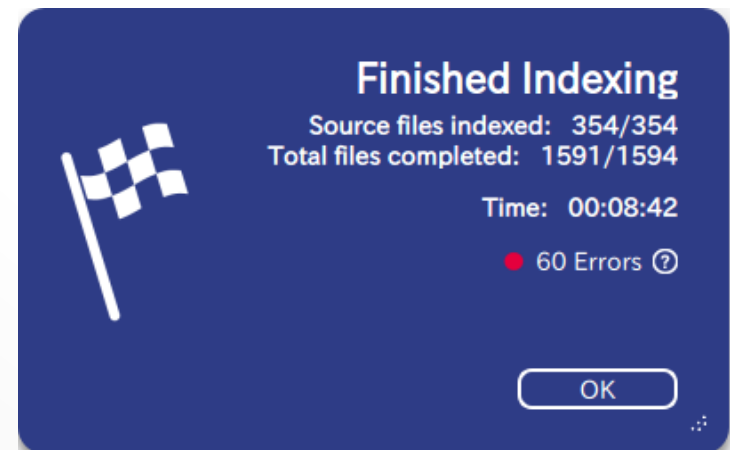
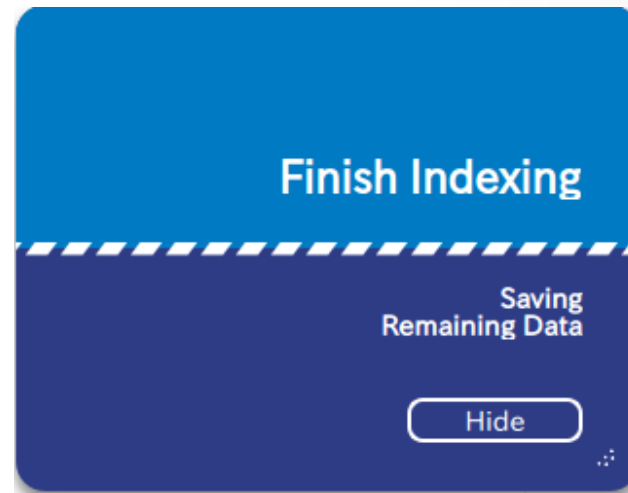
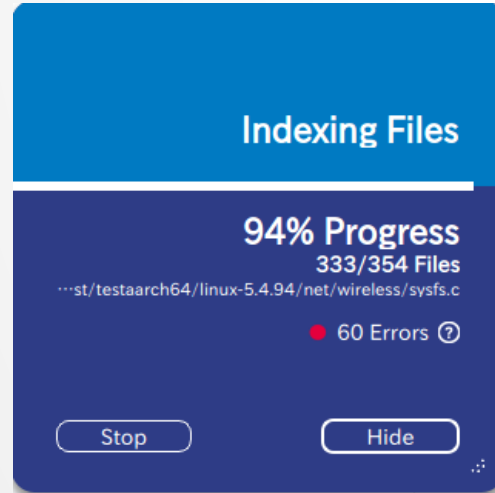
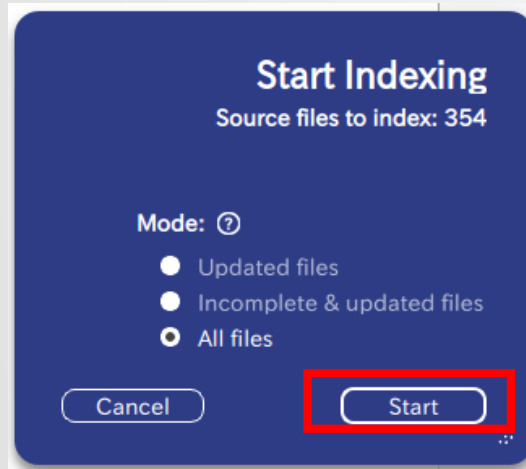
How to use

- Set existing “compile_commands.json” at kernel source.
- Push “Select from Compilation Database”
- Push “check all” and “save” in sub menu.
- Push “Create”



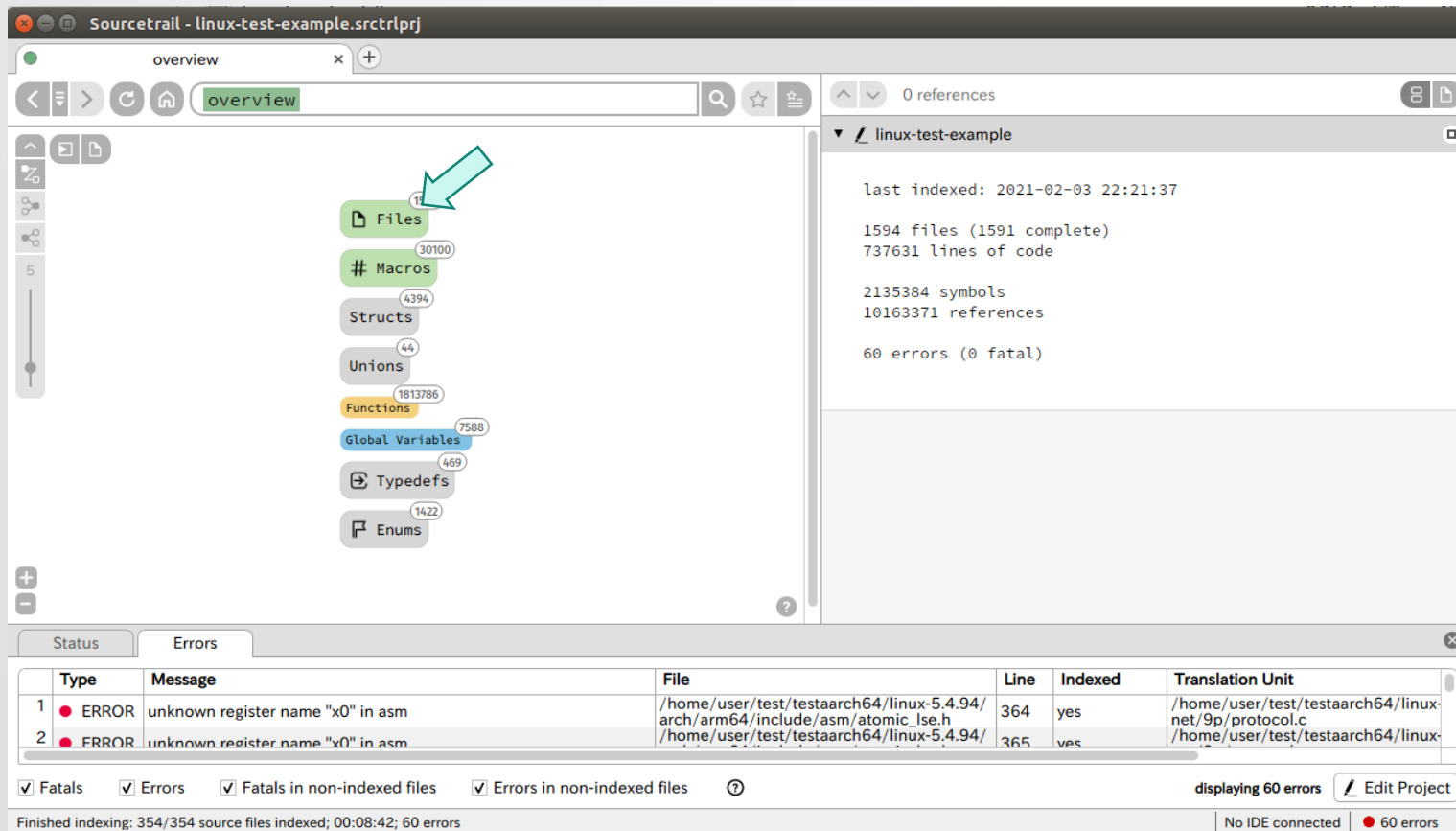
How to use

- Push “Start” and waiting, waiting, waiting....



How to use

- We can analyze to kernel.



The screenshot shows the Sourcetrail interface for a project named 'linux-test-example'. The 'overview' tab is active, displaying a summary of the project's code analysis. A teal arrow points to the 'Files' category in the overview, which has a count of 1594. Other categories include Macros (30100), Structs (4394), Unions (44), Functions (1813786), Global Variables (7588), Typedefs (469), and Enums (1422). The right-hand pane shows the project's statistics: last indexed on 2021-02-03 at 22:21:37, 1594 files (1591 complete) with 737631 lines of code, 2135384 symbols, 10163371 references, and 60 errors (0 fatal). At the bottom, the 'Errors' tab is selected, showing a table of errors.

Type	Message	File	Line	Indexed	Translation Unit	
1	ERROR	unknown register name "x0" in asm	/home/user/test/testaarch64/linux-5.4.94/arch/arm64/include/asm/atomic_lse.h	364	yes	/home/user/test/testaarch64/linux-net/9p/protocol.c
2	ERROR	unknown register name "x0" in asm	/home/user/test/testaarch64/linux-5.4.94/	365	yes	/home/user/test/testaarch64/linux-

displaying 60 errors Edit Project

Finished indexing: 354/354 source files indexed; 00:08:42; 60 errors | No IDE connected | 60 errors

How to use

- We can analyze to kernel.

The image displays two screenshots of the Sourcetrail IDE interface, showing the analysis of a Linux kernel test example project.

Top Screenshot (Project Overview):

- Project: linux-test-example
- Files: 1594
- Macros: 30100
- Structs: 4394
- Unions: 44
- Functions: 1813786
- Global Variables: 7588
- Typedefs: 469
- Enums: 1422

Bottom Screenshot (Error Details):

last indexed: 2021-02-03 22:21:37

1594 files (1591 complete)
737631 lines of code

2135384 symbols
10163371 references

60 errors (0 fatal)

Type	Message	File	Line	Indexed	Translation Unit	
1	ERROR	unknown register name "x0" in asm	/home/user/test/testaarch64/linux-5.4.94/arch/arm64/include/asm/atomic_lse.h	364	yes	/home/user/test/testaarch64/linux-net/9p/protocol.c
2	ERROR	unknown register name "x0" in asm	/home/user/test/testaarch64/linux-5.4.94/	365	yes	/home/user/test/testaarch64/linux-

displaying 60 errors | Edit Project

Finished indexing: 354/354 source files indexed; 00:08:42; 60 errors

How to use

- We can analyze to kernel.

The screenshot displays a web browser window with the address bar set to 'up'. The main content area shows a graph of kernel symbols. A central node labeled 'up' is connected to several other nodes: '___addressable_up189 (semaphore.c)', '__up_console_sem', 'seccomp_do_user_notification', 'seccomp_notify_rcv', and a 'Non-indexed Symbols' box containing '___up', '_raw_spin_lock_irqsave', and '_raw_spin_unlock_irqrestore'. A blue arrow points to the 'up' node.

On the right side, the browser shows the source code for 'semaphore.c' and 'printk.c'. The 'semaphore.c' code includes the following function:

```
171 /**  
172  * up - release the semaphore  
173  * @sem: the semaphore to release  
174  *  
175  * Release the semaphore. Unlike mutexes, up() may be called from any  
176  * context and even by tasks which have never called down().  
177  */  
178 void up(struct semaphore *sem)  
179 {  
180     unsigned long flags;  
181  
182     raw_spin_lock_irqsave(&sem->lock, flags);  
183     if (likely(list_empty(&sem->wait_list)))  
184         sem->count++;  
185     else  
186         __up(sem);  
187     raw_spin_unlock_irqrestore(&sem->lock, flags);  
188 }  
189 EXPORT_SYMBOL(up);  
190  
191 /* Functions for the contended case */  
192
```

The 'printk.c' code shows the following usage:

```
... __up_console_sem  
251     mutex_release(&console_lock_dep_map, 1, ip);  
252  
253     printk_safe_enter_irqsave(flags);  
254 |     up(&console_sem);  
255     printk_safe_exit_irqrestore(flags);  
256 }  
257 #define up_console_sem() __up_console_sem(_RET_IP_)  
258  
259 /*  
260  * This is used for debugging the mess that is the VT code by  
261  * keeping track if we have the console semaphore held. It's  
262  * definitely not the perfect debug tool (we don't know if _WE_
```

How to use

- We can analyze to kernel.

The screenshot displays a web browser window with the address bar showing `_raw_spin_lock_irqsave`. The main content area shows a diagram with three nodes: `Referencing Symbols` (with a count of 262), `_raw_spin_lock_irqsave` (highlighted in orange), and `Non-indexed Symbols` (with a count of 2). An arrow points from the first node to the second, and another from the second to the third.

On the right side, there is a panel titled "256 references" showing code snippets from `spinlock.c` and `async.c`. The `spinlock.c` snippet shows the definition of `__raw_spin_lock_irqsave` and its export. The `async.c` snippet shows several calls to `spin_lock_irqsave` within the `lowest_in_progress` and `async_run_entry_fn` functions.

Conclusion

- This lightning talk shown “How to use Sourcetrail for kernel analysis”.
- This tool is good solution for source code analysis with GUI.
- On the other hand, this tool is too heavy for all kernel source code analysis... I recommend to analyze to partially such as kernel/, net/, etc...
- This tool can analyze in cross environment. I success to analyze arm64 kernel in x86_64 PC.
 - When you want to try, it only require to create the "compile_commands.json" file by kernel cross building.

Thanks!

When you have a question, please send mail to mailing-lists.