Alexandru Totolici
alex.totolici@gmail.com

EXECVUS

ExecVus - Execution Views

Domain, Task, Dataset

When it comes to software debugging, developers have a variety of tools available to them: `gdb`, `otool`, `DTrace`, etc., but most of these tools are nearly impossible to use for kernel debugging. Partly motivated by these shortcomings, the UBC NSS[1] Lab came up with Tralfamadore. One of the many ways in which this tool can help developers is by showing them what parts of their code have been executed, and with what values.

A typical debugging workflow would be:

1. *Data collection.* A developer collects detailed execution information, by running the targeted OS inside a modified Qemu virtual machine; the Tralfamadore-specific modifications made to Qemu allow for very detailed execution traces to be collected.

2. *Data presentation.* The developer can analyse the collected information (the 'trace') in a number of ways; in this case, we're overlaying the execution paths on top of the Linux kernel source code. Annotations show the developer what path execution took through their function(s).

3. *Data navigation.* The developer is interested in inspecting where code is called from and how[2]. They are also interested in following the execution flow up the chain (callers to the code under review) or down (functions called by the code).

Expertise

I have worked with the NSS as a Research Assistant for the past 3 months. During this time I have been briefly exposed to the existing user interface and identified a few shortcomings and potential ways in which to improve and extend it. Discussions with lab colleagues have also proven fruitful in providing me with ideas.

Proposed InfoVis solution

My solution aims to improve the experience of using Tralfamadore in a number of ways:

1. *Code collapse*
In a fashion similar to how IDEs allow users to hide code that they are not working on, ExecVus will hide portions of code that have not been executed in the current trace.

---

[1] Networks, Systems, and Security

[2] 'How' refers to parameter values and data flow, and this is partly future work; ExecVus will be extended to cover these as they're made available.

2. *Code reorder*

The GNU C compiler does a number of optimizations on code, and one of the results of that process is that the order in which code is written is not necessarily the order in which code is executed. When the code is collapsed, the developer also has the option of seeing it in the order in which it was executed, and thus we can avoid the loops that sometimes appear in the application. The options will be **`execution order`** and **`code order`**. When in **`execution order`**, the code has to be collapsed, since we are not going to determine how the non-executed portions of code have been reordered by the compiler.

3. *Improve selection of the execution path*

When numerous paths overlap, limitations in screen visual estate prevent us from displaying all of them simultaneously yet distinctly. The proposed solution will provide a subview that allows the user to easily differentiate between the various paths and select the one that they're interested in.

4. *Hierarchy filtering (on annotations)*

For developers that know exactly where they need to look there will be an option to turn off displaying those folders in the Linux source tree that have no annotations.

5. *Hierarchy overview / quick-jump (***`node peek`***)*

Also, hovering over an element (folder) that has annotations will bring up an inline tree browser that will display the subfolders that also have annotations and allow the developer to quickly jump to where they want to go.

Scenario:

The developer wants to understand why a flag is incorrectly set, so he:

1. Collects a trace with the typical execution that would set said flags.
2. Overlays the trace on the Linux kernel source tree.
3. Opens up ExecVus.
4. Hovers over the **`drivers`** folder to bring up the 'node peek' and navigates to the **`tralf`** folder, where his code resides.
5. Collapses the code.
6. Switches to '**`execution order`**' and scans down to where the flag should have been set.
7. Since the flag is not set, he expands the code, which also switches back to **`code order.`**
8. Looks at 'Call Sources' to figure out where his code is being called from.
9. Navigates upwards the call chain.

Illustrations / Mockups

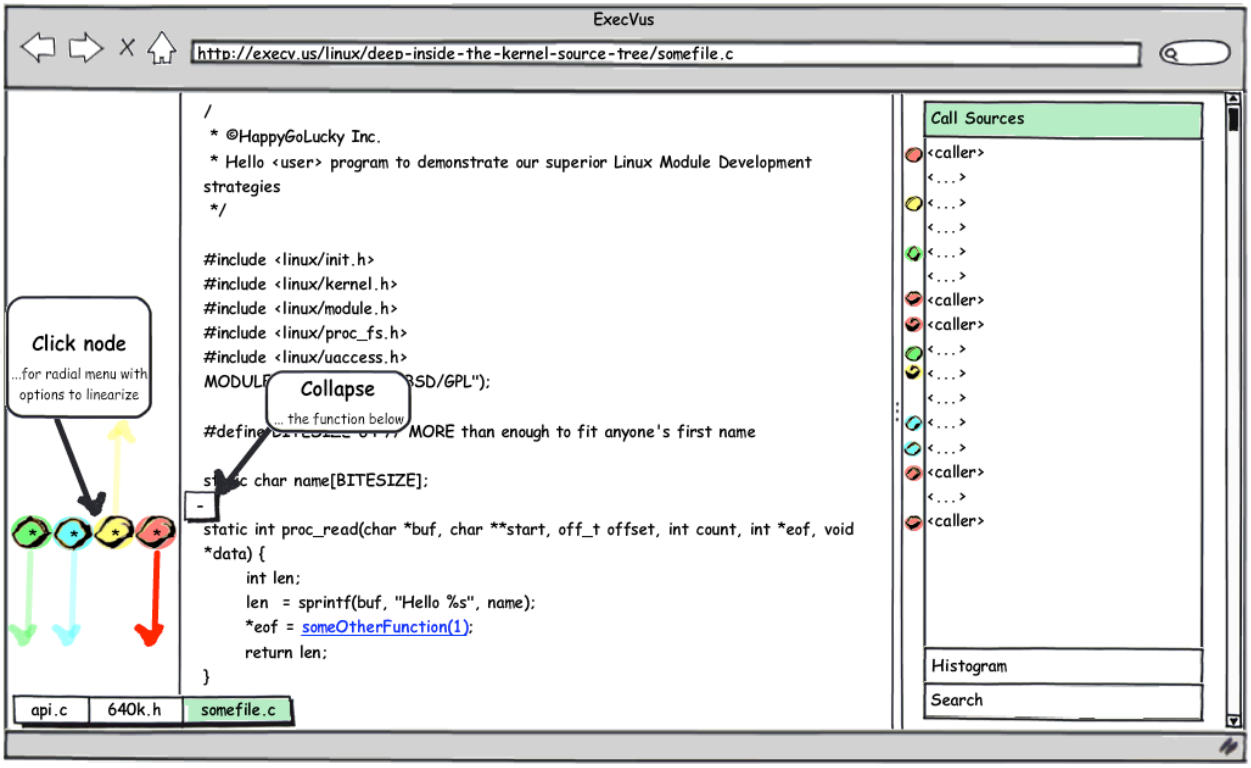

Figure 1: The Code View (click for a larger, online version)

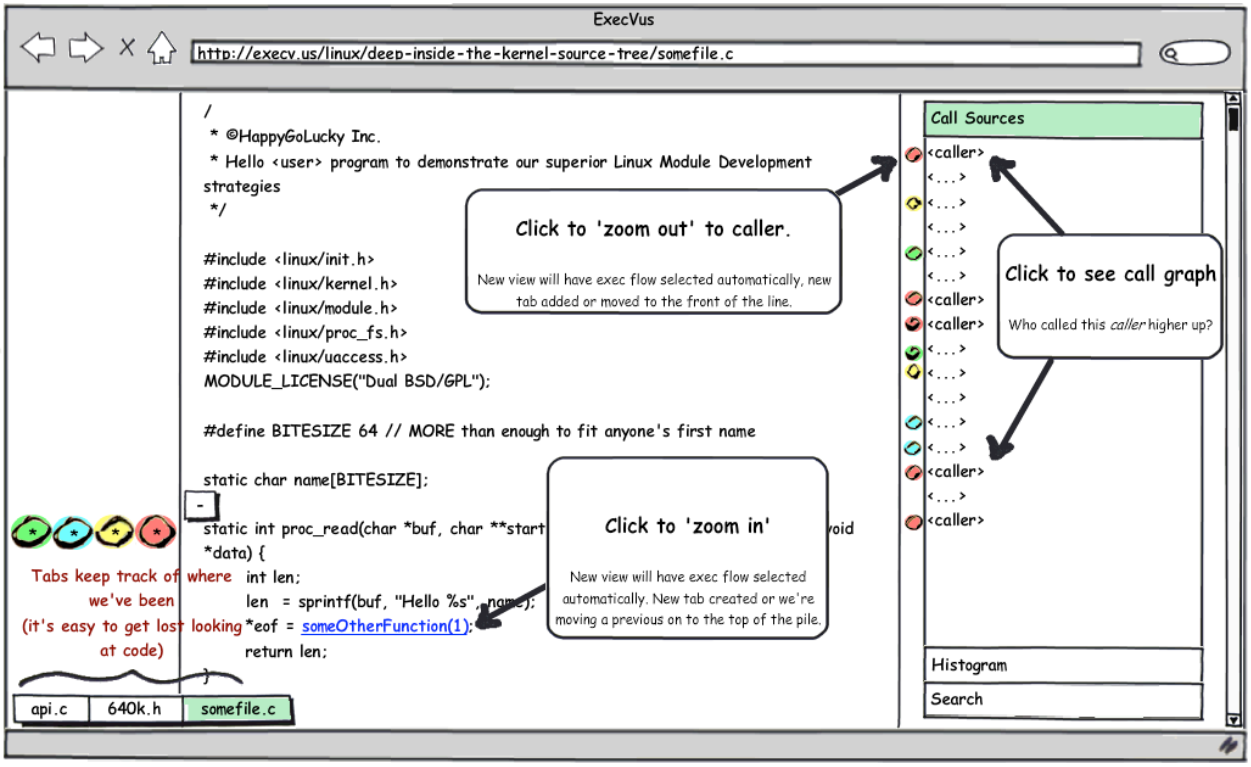

Figure 2: Zoom Controls (click for a larger, online version)

Alexandru Totolici
69229045

Implementation Approach

The existing viewer works as a mercurial plugin, and so will the proposed solution. However, the proposed solution will be written in JavaScript/jQuery, and existing parts will also be rewritten. The current viewer doesn't work outside Gecko browsers, and ideally support for all modern WebKit and Gecko browsers will be provided upon completion.

Milestones

The milestones will reflect completion of the major components, which are:

1. Code collapse / reorder — 13 November
2. Path selector — 20 November
3. Hierarchy graphing/navigation — 27 November
4. Zooming — 11 December

Previous Work

Tools and papers that serve as inspiration:

- Current Tralfamadore viewer
- IDE code collapse
- Path Projection
- Ball & Eick, *Software Visualization in the Large*
- Jakobsen & Hornbæk, *Evaluating a Fisheye View of Source Code*
- Hornbæk & Frøkjær, *Reading Patterns and Usability in Visualization of Electronic Documents*