DREU Final Paper

Michelle Tocora

Florida Institute of Technology

[mtoco14@gmail.com](mailto:mtoco14@gmail.com)

August 25, 2015

**"A Study of Synchronization Routine Change in Mozilla"**

**ABSTRACT**

Meticulously studying the version control system repositories of a large open-source software project(Mozilla in my case) to discover various patterns in developing good concurrent software. The data collected and analyzed by the end of this research will be helpful for future researches to use. They will then be able to understand the challenges that developers face when adjusting synchronization details to avoid concurrency bugs while writing multi-threaded software like Mozilla.

**INTRODUCTION**

My part of the research project at the University of Chicago mainly focused on exploring condition-variable synchronization in the open-source software project I was assigned. Condition variables are one of the primitives needed to build concurrent programs like those that are multi-threaded. Its two principal operations are the following as stated in[1]:“The wait( ) call is executed when a thread wishes to put itself to sleep; the signal( ) call is executed when a thread has changed something in the program and thus wants to wake a sleeping thread waiting on this condition.” Each thread in the program runs independently, accesses memory, and operates on program’s behalf. Condition variable synchronization is a main part of the program that keeps it working as expected by coordinating access to memory between threads. Since Mozilla is a highly multi-threaded program, getting to study its revision history was challenging because its size is massive. We are talking of about over 250,000 versions of source code which is way more than a billion lines of code. However, in the several of weeks given, I have planned to create a script that checks out from Mozilla's version control system, Mercurial[4]. Accessing the Mozilla source code will allow me to parse the diff files generated from the repository and understand specific contents of the Mozilla Browser Suite's revisions. Besides scripting, part of this project also demands manual evaluation of the source code. With that done, I will be able to collect statistics by asking myself the following: *Which revisions contain wait/signal changes? How many revisions contain wait/signal changes? How many and which revisions share the same parameter?* The data collected from the questions above will help compare how the software project has matured from time to time as well as benefit upcoming studies in developing tools. For instance, creating automatic condition-variable synchronization would be an influential tool support for adjusting signal/wait operations to assist developers when working on greatly multi-threaded software.

**EXPERIMENTAL DETAILS AND PROCESSES**

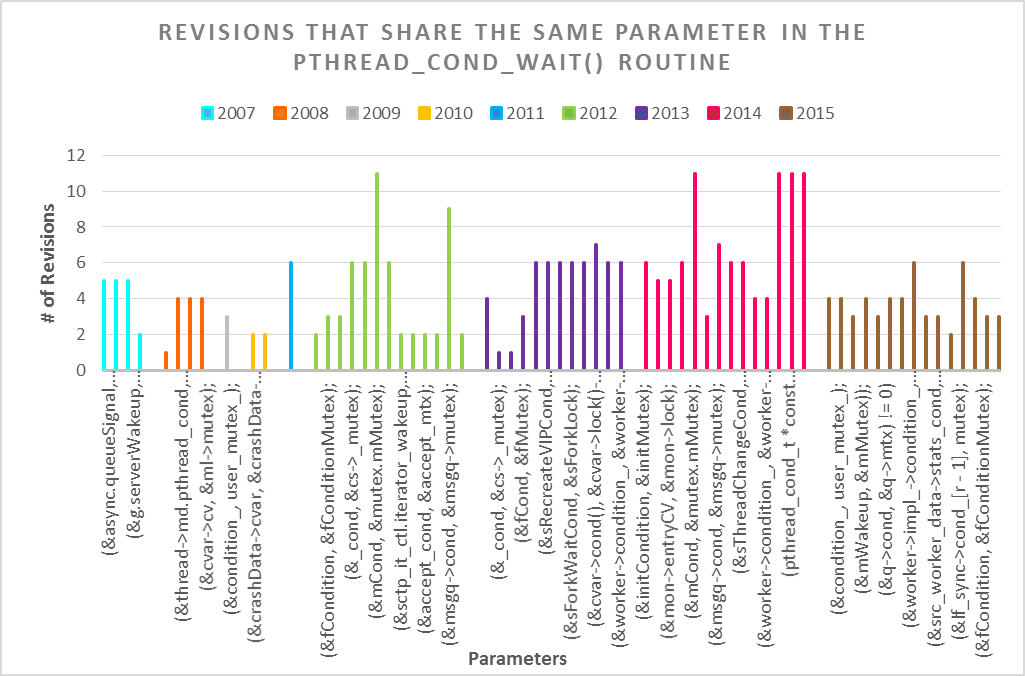
In the beginning, it was important familiarizing myself with concurrent programming and concurrency bugs. Practicing some sample code provided in the online textbook[1] such as moving around the wait/ signal commands allowed me to get a really good background of the different changes I would be seeing in my case study of Mozilla. In addition, I spent a good amount of time in learning basics to Python[2]. This made it easier for me to comprehend the scripts of past researches similar to this project and write my own script too.

The script used to collect my statistics was half created by Rui Gu from Columbia University, USA while the other half was mine’s. It simply parsed the diff files generated from the repository[4] and divided them into the old and new version code as well as added and removed lines. Each revision(257, 268 revisions studied in total) was saved in its own file instead of one big file and named after “diffinfo” followed by its revision number. An example of what revision 14 would be saved as is “diffinfo14.” Doing this allowed me to manually inspect the code within each revision using command line tools. For instance, using ‘grep’ helped speed up in searching the Mozilla libraries' APIs(found the condition variable functions to be from the POSIX library only). The script also permitted me to obtain the log info of each revision with another useful command. Since the time in working on the project was limited, my mentor and I had decided to keep my script as is. Finding structural patterns in order to write some code that categorizes wait/signal function changes would take a while; therefore, leaving me with no results to share in the end.

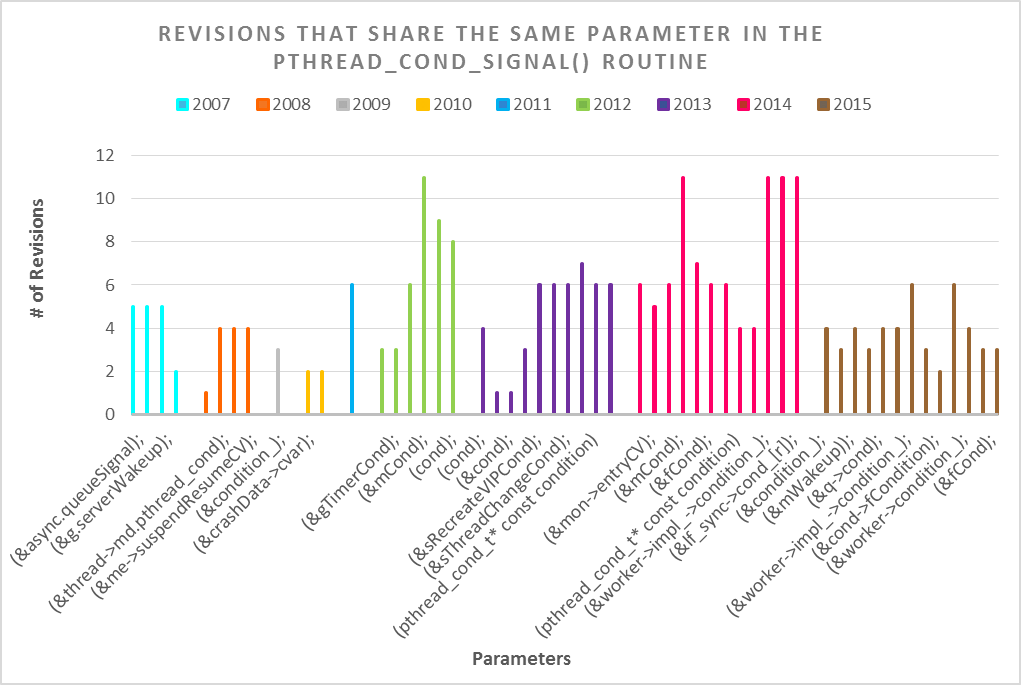
From here and on, everything that will be discussed has been manually evaluated. I came to a conclusion that Mozilla only used “pthread\_cond\_wait( )” and “pthread\_cond\_signal( )” and so those were the keywords I would ‘grep’ in order to figure out what revisions contained signal/ wait operation. After going through the revisions that did contain condition variable synchronization, I was able to count the number of signal and wait changes for every year in its entire development as shown in Figure 1. The last set of statistics were collected with the help of Figure 1. I knew the specific revisions that had wait/signal operations in them, so I split them into two parts as shown in Figures 2 and 3. Moreover, the wait ( ) routine usually contained mutex in one of both its parameters while signal( ) always contained a single parameter. Numerous wait and signal routines from different revisions shared the same parameters, but notice how there are three which do not. The graph does not display the names of parameter well enough and neither the revision numbers due to the lack of space. Nonetheless, that information is saved and visible if requested for future researches to examine.

**RESULTS**

**Figure 1:** Notice how Mozilla has changed greatly over the years because it has the youngest code repository.



**Figure 2:** Some parameters are used highly at certain period of time and others used less at another period of time.Note: not all parameters appear on the graph above because the page size of graph had to be reduced to fit in this page. You may see full graph on my website[3].



**Figure 3:** Note: not all parameters appear on the graph above because the page size of graph had to be reduced to fit in this page. You may see full graph on my website[3].

**CONCLUSIONS**

This paper focused on the condition variable synchronization routine changes throughout the history of versions in code repository of Mozilla. I believe that the several statistics collected in this paper will be convenient for researchers who plan on further studying how to help developers confront concurrency bugs and over-synchronization problems. In all, the entire DREU research experience has exposed me to multi-threaded software projects plus motivated me to become a better programmer.

**ACKNOWLEDGEMENTS**

Coming into this internship, I did not know what to expect until I was connected with my mentor, Professor Lu. She always directed me to get a better understanding of anything I was not clear on. Her PhD student, Haopeng, also taught me a lot related to the project. My research experience would have been uninteresting without them, and thankful to have crossed upon such brilliant individuals.

**REFERENCES**

[1] Operating Systems: Three Easy Pieces. (March, 2015). http://pages.cs.wisc.edu/~remzi/OSTEP

[2] The Python Tutorial. (September 1, 2015). https://docs.python.org/2/tutorial/

[3] DREU 2015. (September 6, 2015). http://www.michelletocora.com/

[4] Mozilla Developer Network. Getting Mozilla source code using Mercurial. https://developer.mozilla.org/en~US/docs/Mozilla/Developer\_guide/Source\_Code/Mercurial.