Testing Concurrent Programs

Anam Dodhy

Abstract

Concurrent programming is considered to be one of the most challenging paradigms in the world of programming. And this is mainly due to the fact that a human mind is not trained to think about problems in a concurrent manner and hence, a concurrent program becomes more prone towards human error. Identifying these errors is a painstakingly difficult process and sometimes requires months of debugging. This paper will present three different tools i.e. ConTeGe, CHESS and Eraser and a brief overview of how they work and identify some of the most common errors e.g. deadlocks, livelocks etc. in a given piece of code. These tools have been designed by taking different approaches while making sure that minimum amount of alterations are required in the code under test.

1. Introduction

Concurrent programs are not only hard to implement but they are even harder to debug. It is mainly because in a concurrent programs the concept of a code running in a single thread in a sequential manner is no longer there. Multiple threads are executing various parts of the code at different times and at the same are interacting with another as well via shared memory, variables etc. This type of complexity results in bugs such as Heisenbug [Ref: 18 from CHESS paper] which occur in system which has been running normally for months. These type of bugs are really hard to reproduce and to debug due to the non-deterministic behavior of the threads. This paper will present three different tools all following different approaches. First is ConTeGe which takes widely known thread-safe classes and tests their methods by calling them via multiple threads using a single object. Second is CHESS which takes control over all kinds of nondeterminism a concurrent program can have so that at the end of the day it is able to ensure same result if an execution is executed multiple times. Last is Eraser, which focuses on ensuring that a shared memory variable is always protected by a thread locking mechanism. Next sections will give a brief over of the main idea of all the aforementioned tools and their main components. Later on a comparison would be made among all three tools which would give an idea how they tools differ from one another

2. Some Section

After this great introduction (Section 1), the following provides some additional hints and examples for the layout and style of this paper.

2.1 Citations

Use citations to refer to other papers (**??**) and books (**??**).

2.2 Tables

Table 1 shows how a table looks like.

2.3 Figures

Figure 1 shows a simple figure with a single picture and Figure 2 shows a more complex figure containing subfigures.

English	German
cell phone	Handy
Diet Coke	Coca Cola light

Table 1. Translations.

Figure 1. SOLA logo.

(a) TUDaLogo (b) SOLALogo

Figure 2. Two pictures as part of a single figure through the magic of the subfigure package.

Listing 1. Example usage of the listing package

```
class S {
    int f1 = 42;
    public S(int x) {
            f1 = x;
    }
}
```

2.4 Source code

The listings package provides tools to typeset source code listings. It supports many programming languages and provides a lot of formatting options.

Listing 1 shows an example listing. Code snippets can also be inserted in normal text: `\lstinline|int f1 = 42;|` gives `int f1 = 42;`

2.5 Miscellany

Capitalization. When referring to a named table (such as in the previous section), the word *table* is capitalized. The same is true for figures, chapters and sections.

Bibliography. Use `bibtex` to make your life easier and to produce consistently formatted entries.

Contractions. Avoid contractions. For instance, use rather than "don't."

Style guide. A classic reference book on writing style is Strunk's *The Elements of Style* (**?**).

3. Another Section

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

4. Yet Another Section

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

5. Conclusion

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.