Checking binary compatibility for modern programming language

Áron Baráth, Gábor Alex Ispánovics, Porkoláb Zoltán

Modern programming languages prefer to support rapid development improved with flexible and expressive features. It seems languages take more effort in development rather than long-time support of the code. This trend can be seen in the fact that many languages handles binary compatibility poorly. In many cases only a very little change can cause serious runtime problems, like miscalculations and crashes.

In C [3] and C++ [4] (and in many other languages) binary compatibility starts when linking multiple objects into an executable or dynamic library. Using any build system, we can give situations when the built system will not recognize the dependencies correctly, and the compiler outputs a wrong binary – it is more likely when system-wide headers are also involved. Furthermore, the same issue can arise when linking against static libraries. These problems can be avoided with a local database of detailed information about the types and functions. The problem is getting more uncontrollable when a client uses dynamic libraries. While in C only the name of the functions and objects (also known as global variables) are used in static- and dynamic linking, in C++ *mangled names* are used to link functions. The mangled names will provide a little more validation when loading a library, but it is not nearly sufficient.

In the other hand, we have to calculate with the expected incompatibility: obviously, an interface modification will break the user programs, while modifications are not related to the interface itself are the most problematic. Many languages cannot handle correctly the second case due to optimization reasons or due to information loss.

Note that, the problem of the binary compatibility is not limited to languages like C and C++, but it is a real issue in managed languages as well – for example in Java language. Researches aimed to identify the possible weakpoints [1], others try to provide a solution for that [2].

In this paper we present an experimental method to detect binary incompatibilities in C++ program. Moreover we introduce our experimental programming language, Welltype [5], which is aimed to present a solution to detect unwanted binary incompatibilities at link-time. Our approach will not load incompatible programs into the same runtime context.

References

[1] Dietrich, J., Jezek, K., Brada, P.: *Broken promises: An empirical study into evolution problems in java programs caused by library upgrades*. Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on. IEEE, (2014)

[2] Savga, I., Rudolf M., Goetz, S.: *Comeback!: a refactoring-based tool for binary-compatible framework upgrade*. Companion of the 30th international conference on Software engineering. ACM, (2008)

[3] Kernighan, B. W., and Ritche, D. M.: *The C programming language*. Vol. 2. Englewood Cliffs: prentice-Hall (1988)

[4] Stroustrup, B. *The C++ Programming Language, 4th Edition*. Addison-Wesley (2013)

[5] Welltype web page. `http://baratharon.web.elte.hu/welltype/`