UPPSALA
UNIVERSITET

Sprint 3 - Report

by Goblins

March 3rd 2023

1 Introduction

How does Java and C differ when it comes to compilation and binding? Which way of compilation is the best? C, which is run directly on the OS, compiles the source code to assembly code which is then being translated to machine code that can be run on the processor. Java, on the other hand, is not being run directly by the OS. Instead, the source code is compiled into Java bytecode, which is then run on a virtual machine called a JVM (Java Virtual Machine). Why is the compilation process different for the two languages? In this essay, the differences between the two ways of compilation will be discussed and this question will be answered. A detailed description of each way of compilation will be given and the advantages and disadvantages of each way will be discussed in the end.

2 Binding

To understand how C and Java compiles, it is imperative that one understands what binding is and how the two different ways of binding differ. Static binding, which is the first way of binding, takes place during compile-time. This is where entities are bound to their respective types. All entities whose type can be defined at compile time gets bound here[2]. Dynamic binding, which is the second way of binding, occurs during run-time. Entities whose type could not be defined at compile-time gets bound during run-time when more information about them are given[2].

3 Compiling C

The C compiler consists of four different steps. It starts with the pre-processor, and then continues with compilation, assembly and ends with linking.

3.1 Pre-processor

The pre-processor provides the ability to include header-files, use macro expansions, conditional compilation and line control. This stage can also be divided into four phases, starting with trigraph replacement. In this phace, all trigraph sequences, which are are sequences of characters that serve as a placeholders for some other character. These exist because all characters from the C-set are not available on all platforms. Because of this, these characters are replaced by trigraph sequences which are translated to their corresponding character during this phase[1]. The second phase is line splicing, where

physical source lines are processed and spliced into logical lines of code. For example, the code snippet "for(int i = 0; i < 5; i++)" would be divided into multiple different lines where each line does one thing. When the second phase is completed, the third phase begins which is called tokenization. During this phase the result of the previous phase is broken up into pre-processing tokens and white space. Comments in the code are also replaced with white space. The fourth phase handles macro expansion and directive handling[5].

3.2 Compilation

During this stage, the source code is being translated from C-code to assembly code. This assembly code may differ depending on which processor architecture the code is being compiled for. Some compilers translate C-code directly to machine code in order to save time otherwise spent on first translating C-code to assembly code and then translating the assembly code to machine code.

3.3 Assembly

The assembler translates the previously generated assembly code into object code that are put into object files, meaning files with the .o ending. Object code are instructions that can be run on the processor and is in binary format when checking it out. Therefore this code is unreadable to the average human.

3.4 Linking

The linker system takes the object files generated in the previous stage and creates a single executable file, library file or object file depending on what instructions the compiler was given. This includes replacing references in the code to their actual values. When linking, the compiler doesn't know which files to include in order to satisfy all dependencies, because of this the user has to supply the compiler with all the files necessary. The linking-part can be done in two different ways, more on that later[2].

Before this stage finished, the binding phase takes place. In C, due to the fact that C is a statically compiled language, dynamic binding is uncommon. Therefore, almost all the bindings happens here during compile-time. Variables and functions are now bound to their respective values and implementations. When using static binding you spend a bit more time during compile time to bind but doesn't have to spend time during run-time to bind. In order to achieve dynamic binding, one has to go out of their way to write code that

results in dynamic binding. When linking routines and code from outside libraries, there are two different ways of doing this. The C compiler gcc uses dynamic linking as default but can use static linking if the user gives it the necessary arguments[2].

3.4.1 Static linking

Static linking means that references to libraries are replaced with the code that they are referring to. The code is copied over, resulting in possibly many copies of the same code. This is the downside of static linking, more disc space is needed. The advantage of static linking is that the code becomes more portable because the libraries referred to does not have to be present when executing the program. The program also saves time due to the fact that no time have to be spent on accessing libraries during run-time[2].

3.4.2 Dynamic linking

Dynamic linking means that the references to libraries are being kept as they are. When these references are found in the code, the code referred to in the library are accessed and then executed. The advantages of this is that code used often only need to be stored in one location which saves disc space. Bugs that are found in the referred code can be fixed and all parts of the code that use this referred code can take advantage of this without having to re-link the entire program as one would have to do with a statically linked program. On the other hand, when libraries used by dynamic linking are updated and are not backwards compatible, code that uses these libraries will break[2].

4 Compiling Java

To understand how Java-code compiles, one must first have a basic understanding of what a Java Virtual Machine is.

4.1 Java Virtual Machine

A Java Virtual Machine is a virtual machine, as the name states. Often shortened as a JVM. This machine has the ability to run any program that can be translated to Java bytecode. When running a given program, the JVM interprets and translates the successfully translated Java bytecode into machine code. When saying that java code gets interpreted, this is what it means. For the JVM to be able to interpret Java bytecode, it needs a Java Interpreter.

Because of this an interpreter is built into the JVM. The machine code generated is unique for each processor architecture because of the differences in how processors work[6].

All machine code is not generated at the same time when compiling. The JVM uses something called JIT-compiling. JIT stands for "just in time". A JIT-compiler compiles the code right before it needs to be executed. The advantages of doing this is that the program can run faster because it does not need to wait until all code have been translated to machine code. Depending on the implementation of the JVM, the JVM can mark pieces of code as "hotspots", meaning code that is used often. The JIT will now compile those "hotspots" before running the program, and then compile everything else while running. For this to work the JVM has to keep statistics about what code is used often[7].

4.2 Compilation

When compiling a Java program, what the compiler does is that it translates the Java source code into Java bytecode that the JVM can understand. The Java compiler is smart enough to understand what files the program is dependent on in order to work. Those files are therefore included automatically[3].

4.3 Linking

The final part of compiling a Java program is linking. The linking part starts of with verification. In this stage, the JVM makes sure the class follows the correct language structure. The second stage is called the preparation stage, in which the JVM allocates memory in order to store the class variables. These variables are set to their default values, which depend on the type of the specific variable. The final stage is the resolution stage, and this is where binding takes place. In Java, private, final and static methods and variables are examples that use static binding because their types can be defined at compile-time. An example of dynamic binding is an object that is an instance of a subclass. Because the object is an instance of both the subclass and the superclass, its type can not be defined during compile time. Therefore this object has to be bound dynamically. In short, the linker system takes a class or interface and turns it into a run-time state so that the JVM can execute it[4]. As mentioned earlier in section 3.4, there are two ways to link code from libraries. Java uses dynamic linking and can not use static linking. The reason why Java can not use static linking is because that would break one of the core concepts of Java, that you should write code once and use it

everywhere. This shows that using dynamic linking is good when you want to re-use code[4].

5 Conclusion

The C compiler consists of four steps: the preprocessing stage, the compilation stage, the assembly stage and the linking stage. C is for the most part statically bound and can be either statically linked or dynamically linked. All code in a C-program is translated into machine code which is being run directly on the processor by the OS. Java on the other hand, uses a compiler to translate the Java source code into Java bytecode which is then run by a Java Virtual Machine, shortened JVM. The JVM translates the Java bytecode to machine code just before that piece of code needs to be executed while C-code has to be compiled on the same processor architecture that it is going to run on. Java is both statically- and dynamically bound depending on whether the type can be defined at compile-time or not. Java can only be dynamically linked, while C can be either dynamically linked or statically linked depending on the arguments given to the compiler.

So, to answer the question given in the introduction, which way of compilation is the best? The answer to this question is that neither way of compilation is better than the other. It all depends on what the specific program requires. For example, if the program is going to be run on only one architecture, then C's way of compiling is the most efficient way. But if the program is going to be run on multiple different architectures, then Java's way of compiling may be better as it uses a JVM which is platform-independent.

Referenser

[1] IBM. Trigraph sequences — Ibm. `https://www.ibm.com/docs/de/zos/2.3.0?topic=set-trigraph-sequences`, 2021. [Online; accessed 21-December-2022].

[2] IBM. When to use dynamic linking and static linking — Ibm. `https://www.ibm.com/docs/en/aix/7.2?topic=techniques-when-use-dynamic-linking-static-linking&fbclid=IwAR3y_fqJT4Ef6qlY5I4dXrIJzGV5zNDZGwkk9NZ5p2_GIOjgbOG5UotEQ_Q`, 2022. [Online; accessed 30-November-2022].

[3] Oracle. Compiling - Running a Simple Program — Oracle. https://www.oracle.com/java/technologies/compile.html?fbclid=IwAR1yG4dEQ5YM-gEGERjyLfsku4uA1UOmOh8FZMWs3YM1lk_5HfpwQMbg5B4, 2022. [Online; accessed 30-November-2022].

[4] Oracle. Loading, linking, and Initializing — Oracle. https://docs.oracle.com/javase/specs/jvms/se7/html/jvms-5.html?fbclid=IwAR0tH5ZYx43xM-OywrOBNhQZ5u4y0_QV7ZqyOowOkwTkNCjZsZwVlOIEhGQ#jvms-5.2, 2022. [Online; accessed 30-November-2022].

[5] Wikipedia. C preprocessor — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=C%20preprocessor&oldid=1123475111, 2022. [Online; accessed 30-November-2022].

[6] Wikipedia. Java virtual machine — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Java%20virtual%20machine&oldid=1114756187, 2022. [Online; accessed 30-November-2022].

[7] Wikipedia. Just-in-time compilation — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Just-in-time%20compilation&oldid=1124576287, 2022. [Online; accessed 30-November-2022].