How Not To Build A Binary Translator

Aimilios Tsalapatis, Kenneth R Hancock

Binary translation is a common method used to speed up the execution of software components in systems that cannot run them natively. The process consists of dynamically creating native machine code corresponding to the foreign binary. While the concept is straightforward, there is a slew of ways the construction of the translator can go awry. In this paper, we attempt to enumerate difficulties encountered while writing such an assembler from the 6502 to the x86-64 architecture. Examples include dynamically rewriting the emitted code to connect basic blocks, properly allocating host registers to the guest code, and mixing emulation and binary translation. We discuss the nature of these problems, and offer advice for anyone attempting a similar task.

1 Introduction

1.1 Arguments for Binary Translation

Binary Translation is the process through which software which was meant for a specific hardware platform can be run in another system. Normally, binary programs that cannot be run have to be interpreted instruction by instruction by an emulator in order to reproduce its behavior. However, this approach is very slow, since the amount of native instructions executed for each foreign one can be exceedingly high.

A way to speed up the execution of programs on arbitrary architectures is using binary translation alongside emulation. This technique takes advantage of the fact that most architectures are composed of similar commands, like moves and rudimentary binary operations. Therefore, it is possible to create native executables that perform the same function as the original program. The amount of code needed run is lower, and execution is faster.

Another use for binary translation is in software virtualization. Some programs need to access resources that are not accessible, because they do not exist in the system. Alternatively, the program does not have enough privileges to modify the component, like in the case of userspace applications attempting to change their page table register.

Virtual Machines (VMs) are such a class of programs. The kernel running in the VM is designed to have full access the hardware, which in this case includes sensitive processor registers and emulated devices. A translator has the responsibility of replacing code that attempts to do such accesses with alternative instructions that divert flow to the translator. The program emulates the resource in such a way that the VM can keep executing, and diverts control back to it. Popular emulators that employ binary translation, such as Bochs and QEMU, utilize exactly this kind of scheme in order to emulate devices like disks on demand.

1.2 Binary Translation Internals

Binary translation, much like all forms of compilation, hinges on the concept of basic blocks. Each program can be split into basic blocks, which are sequences of machine instructions that do not contain control flow commands like jumps and branches. Moreover, the block cannot be accessed without

The emulator then goes through the program, creating and then executing a sequence of native instructions corresponding to the current block.