How did I find the old Windows binaries and compilers for the processor retrospective series?

devblogs.microsoft.com/oldnewthing/20210111-00

January 11, 2021

Raymond Chen

Friend of the blog Malcolm Smith asked me how I find the raw materials for the the various processor retrospective series. "Do you keep copies of the old RISC compilers? Do you use gcc? I don't think the Microsoft compilers were cross compilers, so how do you get the hardware to run these obsolete versions of Windows? And where do you get a debugger?"

Inside Microsoft, we have a server that keeps old copies of Microsoft software. It doesn't have everything, but it has a lot.

I start by downloading the processor documentation from the manufacturer and reading through the entire instruction set. That teaches me about the processor architecture in general. The next step is seeing how Windows uses it.

That part usually starts with digging out the Windows NT installation CD for the relevant architecture and extracting the `NOTEPAD.EXE` program. I choose Notepad because it's relatively small, or at least it *was* relatively small at the time. Furthermore, I have an old copy of the source code, which makes the reverse-compiling easier. The source code I have doesn't always perfectly match the build of Windows that the CD was created from, but it's usually close enough.

I send the binary through ODA, the online disassembler. I start doing reverse-compilation of the resulting assembly. Good footholds are imported functions that are called in only one place, like `DragAcceptFiles`.

I don't reverse-compile the entire binary, just enough to get a feel for the compiler's code generation and how various common coding patterns end up compiled. I make sure to look for functions like `CreateWindowExW` which have lots of parameters, so I can see more details of the calling convention. I look for interlocked operations to see how atomic operations work. And I look for lightweight leaf functions and functions with large local variables to see how those are treated.

It's during this phase that I discover things like the horrible code generation penalty for byte-packed structures.

I look at the old Windows kernel source code to reverse-engineer details like the register preservation rules and the size and placement of the red zone. Sometimes I also have to dig in to find out how atomic operations are emulated.

If I need to get a sense for how things looked in the Windows debugger, I extract the disassembler from the debugger source code and recompile it, and then ask it to disassemble some bytes from the binary. This also lets me see which, if any, pseudo-instructions are supported by the Windows disassembler.

Sometimes I get lucky and I can find a cross-compiler: If the processor was supported by Windows Compact Edition, I can go to the Windows CE SDK and extract the compiler. This lets me fill in gaps that aren't answered by the existing code in Notepad, and it lets me verify my understanding by writing test functions that exercise the rules I've reverse-engineered.

And then after I gather all the information, I start writing the articles. I try to make the number of articles a multiple of five, since that lets it occupy an integral number of weeks.

What processor is coming next? I'm not sure. I think I've run out of all the easily-accessible processors that Windows once supported but no longer does. I may have to start looking at processors that are still supported.

Raymond Chen

Follow