Why is Identical COMDAT Folding called Identical COMDAT Folding?

devblogs.microsoft.com/oldnewthing/20161024-00

October 24, 2016

Raymond Chen

We saw a while ago that the linker will recognize that two functions consist of the same code sequence and will use the same bytes to represent both functions, a process known as *identical COMDAT folding*. But why is it called identical COMDAT folding?

COMDAT is short for "common data", a feature of the FORTRAN programming language.

For those of you who need some brushing up on FORTRAN: Here's a crash course in common data.

In FORTRAN 77, if you want to share variables between functions and subroutines, you put them in a so-called "common data block", usually shortened to just "common block". For example, here are two FORTRAN subroutines that share a variable called `LAST`:

```
C     THE SETLAST SUBROUTINE TAKES ITS
C     PARAMETER AND SAVES IT IN THE
C     COMMON VARIABLE "LAST"

      SUBROUTINE SETLAST(I)

C     DECLARE THE DATA TYPE OF THE PARAMETER "I"
C     AS INTEGER. THIS IS TECHNICALLY NOT NECESSARY,
C     BECAUSE VARIABLES WHOSE NAMES BEGIN WITH THE
C     LETTERS I THROUGH N DEFAULT TO INTEGER.

      INTEGER I

C     DECLARE A VARIABLE CALLED LAST AND
C     PUT IT IN A COMMON BLOCK CALLED /LASTV/

      INTEGER LAST
      COMMON /LASTV/ LAST

C     OKAY, HERE WE GO. SAVE THE VALUE.
      LAST=I
      END

C     THE GETLAST SUBROUTINE RETURNS THE
C     VALUE SET BY THE MOST RECENT CALL TO
C     THE SETLAST SUBROUTINE.

      INTEGER FUNCTION GETLAST()

C     DECLARE A VARIABLE CALLED LAST AND
C     PUT IT IN A COMMON BLOCK CALLED /LASTV/

      INTEGER LAST
      COMMON /LASTV/ LAST

C     RETURN THE VALUE IN "LAST". THIS VALUE
C     WAS PUT THERE BY THE SETLAST SUBROUTINE.

      GETLAST = LAST
      END
```

(Modern FORTRAN supports lowercase, but I grew up in the days before lowercase was invented. Writing FORTRAN code in lowercase just looks wrong to me.)

Both `SETLAST` and `GETLAST` declare a variable called `LAST` and put it in a common block named `LASTV`. The compiler matches up all common blocks with the same name and aliases them together, so that they all refer to the same block of memory.

You can put multiple variables into a common block by separating them with commas.

Note that it is conventional to give the variables in a common block the same name each time they occur. But there's no requirement that they do. You can give the variables different names each time you declare the common block:

```
SUBROUTINE SETLAST(I)
INTEGER I
INTEGER FRED
COMMON /LASTV/ FRED
FRED=I
END

INTEGER FUNCTION GETLAST()
INTEGER BARNEY
COMMON /LASTV/ BARNEY
GETLAST = BARNEY
END
```

This block of code is functionally equivalent to the previous one. Here, the `SETLAST` subroutine calls the sole variable in the block " `FRED` ", whereas the `GETLAST` function calls it " `BARNEY` ". This is perfectly legal, albeit strange.

You aren't even required to match up the data types, as long as the total size of the common block stays the same. For example, you might say

```
INTEGER*2 A
INTEGER*2 B
COMMON /FOURBYTES/ A, B
```

in one function, declaring two two-byte integers in a common block called `FOURBYTES` , and then in a different function, declare it like this:

```
INTEGER*4 I
COMMON /FOURBYTES/ I
```

The two common blocks are four bytes long, so this is perfectly legal. Of course, the results depend on the endianness of the processor.

Okay, so anyway, FORTRAN had these weird things called "common blocks" which are used to get multiple functions to share a chunk of memory. I'm guessing that these things are what the COMDAT object file segments were originally intended for. The rule that normally applies to COMDAT sections is that if the linker sees more than one COMDAT section with the same name, it will keep one of them and throw away the rest. This is why it's important that all common blocks have the same size: You don't know which one the linker is going to use!

The C++ language introduced places where the compiler may end up emitting the same code multiple times, for example, vtables and non-inline versions of inline functions. The compiler can use these old FORTRAN COMDAT segments to hold those things, and rely on the linker to keep only one copy. (Note that the linker doesn't validate that the duplicates are identical. Yet another reason why the C++ language requires that inline functions be identically-defined in all translation units.)

And finally we get to identical COMDAT folding.

The idea is to put not just inline functions and vtables in COMDAT segments. Let's just put *everything* into COMDAT segments. And then let's tell the linker, "Hey, if you see two COMDAT code segments that are byte-for-byte identical, then go ahead and treat them as if they were the same thing."

That's how we got to the name "identical COMDAT folding". We are taking COMDATs, looking for those which are identical, and collapsing (folding) them together.

Bonus chatter: I pulled a fast one in this article. Next week, I'll come back and unwind it a little.

Raymond Chen

Follow