sign up log in tour help

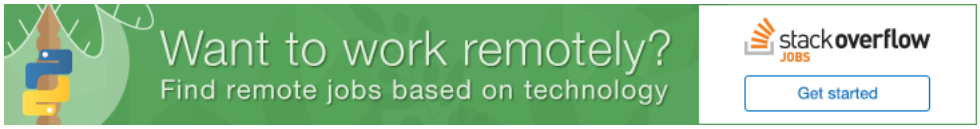
x Dismiss

Join the Stack Overflow Community

Stack Overflow is a community of 6.4 million programmers, just like you, helping each other. Join them; it only takes a minute:

Sign up

What does “COM” means in the Ndx column of the .symtab section?



add2.c:

```
int counter=0;
int a=0;
int b;
int c;
int add(int a, int b) {
    return a+b;
}
```

compilation: gcc -c add2.c -o add2.o

reading the symbol table: readelf --symbols add2.o

```
Symbol table '.symtab' contains 12 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 00000000     0 FILE    LOCAL  DEFAULT  ABS add2.c
     2: 00000000     0 SECTION LOCAL  DEFAULT    1
     3: 00000000     0 SECTION LOCAL  DEFAULT    2
     4: 00000000     0 SECTION LOCAL  DEFAULT    3
     5: 00000000     0 SECTION LOCAL  DEFAULT    5
     6: 00000000     0 SECTION LOCAL  DEFAULT    4
     7: 00000000     4 OBJECT  GLOBAL DEFAULT    3 counter
     8: 00000004     4 OBJECT  GLOBAL DEFAULT    3 a
     9: 00000004     4 OBJECT  GLOBAL DEFAULT  COM b
    10: 00000004     4 OBJECT  GLOBAL DEFAULT  COM c
    11: 00000000    14 FUNC    GLOBAL DEFAULT    1 add
```

What does "COM" means in the Ndx column ? I understand that "counter" and "a" are defined in the section #3 (ie, .bss) and that "add" is defined in the section #1 (ie, .text), but i was expecting "b" and "c" to be defined in the .bss section too, and so get a "3" in the Ndx column.

Thank you

c gcc elf

asked Nov 9 '10 at 19:18

user368507
469 4 20

3 Answers

gcc treats uninitialised globals which are not explicitly declared `extern` as "common" symbols (hence "COM").

Multiple definitions of the same common symbol (across multiple object files) are merged together by the linker when creating the final executable, so that they all refer to the same storage. One of the object files may initialise it to a particular value (in which case it will end up in the data section); if no object files initialise it, is will end up in the BSS; if more than one object initialises it, you'll get a linker error.

In summary, if you have, say, two definitions of `int a`:

- `int a;` in one object and `int a;` in another object is OK: both refer to the same `a`, initialised to 0
- `int a;` in one object and `int a = 42;` in another object is OK: both refer to the same `a`, initialised to 42
- `int a = 23;` in one object and `int a= 42;` in another object will give a link error.

Do note that the use of multiple definitions of the same symbol across two objects is not technically allowed by standard C; but it is supported by many compilers, including gcc, as an extension. (It's listed under "Common extensions" - no pun intended - in the C99 spec.)

edited Nov 9 '10 at 19:53

answered Nov 9 '10 at 19:45

Matthew Slattery
29k 2 64 85

Ok ! I understand that we cannot now where a "COMMON" symbol will end up by looking to one relocatable object file. This is decided by the linker, at linking time, in function of what other relocatable object files do with this symbol. Is it right ? – user368507 Nov 9 '10 at 20:56

Yes, that's right. – Matthew Slattery Nov 10 '10 at 0:03

From this PDF, table 7-11:

> SHN_COMMON
> Symbols defined relative to this section are common symbols, such as FORTRAN COMMON or unallocated C external variables. These symbols are sometimes referred to as tentative.

Also, see this page.

edited Aug 1 '12 at 16:03

Bill the Lizard
226k 139 451 739

answered Nov 9 '10 at 19:37

user200783
4,596 7 37 71

They're uninitialized global variables that are allocated by the linker. Sometimes referred to as communal variables.

Edit: Hrmm, Paul Baker beat me to it, with links no less. use his answer :)

answered Nov 9 '10 at 19:43

JimR
8,583 2 10 20
