Link 4.A Symbols

Young W. Lim

2019-03-12 Tue

Outline

Based on

"Self-service Linux: Mastering the Art of Problem Determination",
Mark Wilding
"Computer Architecture: A Programmer's Perspective",
Bryant & O'Hallaron

Compling 32-bit program on 64-bit gcc

- `gcc -v`
- `gcc -m32 t.c`
- `sudo apt-get install gcc-multilib`
- `sudo apt-get install g++-multilib`
- `gcc-multilib`
- `g++-multilib`
- `gcc -m32`
- `objdump -m i386`

TOC: Symbols

Types of Symbols

- Global Symbols *defined* by module m
  - referenced by other modules
  - *nonstatic* c functions
  - *nonstatic* global variables
- Global Symbols *referenced* by module m
  - defined by other module
  - *external* c functions
  - *external* global variables
- Local Symbols *defined* by module m
  - referenced by module m exclusively
  - *static* c functions
  - *static* global variables

Types of Symbols Summary

	Global Variables	Functions
static	local linker Symbols	local linker symbols
non-static	global linker symbols	global linker symbols
external	global linker symbols	global linker symbols

Global Linker Symbols	Local Linker Symbols
non-static global variables	static global variables
non-static (global) functions	static (global) functions
external global variables	static local variables
external functions	

Local Linker Symbols

- visible only in module m
- other module cannot reference
- not the same as local program variables
- symbols that correspond to local / global static variables

Local Variables and Local Linker Symbols

- symbols that correspond to static local variables
    - local linker symbols in the symbol table with a unique name
    - allocated space in .data or .bss
    - are not managed on the stack
- symbols that correspond to non-static local variables
    - not listed in the symbol table .symtab
    - are managed on the stack at the run time
    - are not handled by the linker

.data and .bss sections in ELF

- non-static local variables
  - maintained on the stack
  - at the run time
  - no space in .data or .bss
- .data
  - initialized global variables
  - no local variables
  - actual space in the object file
- .bss
  - uninitialized gloabal variables
  - no local variables
  - no actual space in the object file
  - just a place holder
  - space efficiency

Static Local Variable Examples

```
int f() {
  static int x = 0;
  return x;
}

int g() {
  static int x = 1;
  return x;
}
```

- compiler allocates two integer symbols in .bss
- exports a pair of unique local linker symbols
- for example, x.1 and x.2

Static Functions

- hide variable and function declarations inside modules
- like private declarations in Java and C++
- C *source files* play the role of *modules*
- private to a module
    - any *global variable* with static attribute
    - any *function* with static attribute
- public to other module
    - any *global variable* without static attribute
    - any *function* without static attribute
- use static attribute wherever possible

TOC: Symbol Tables

ELF Symbol Table Entry

```
typedef struct {
  int name;         // string table offset
  int value;        // section offset, or VM address
  int size;         // object size in bytes
  char type:4,      // data, func, section, or src file name (4 bits)
       binding:4;   // local or global (4 bits)
  char reserved;    // unused
  char section;     // section header index, ABS, UNDEF, or COMMON
} Elf_Symbol;
```

Symbol Tables

- built by assemblers
- using symbols in the assembly language .s file
- the compiler exports symbols

name	byte offset to a string table
value	the symbol's address
size	the size of the object in bytes
type	data or function
binding	local or global
section	info about individual sections

Name and Value

- name : string
- value : address

string table	points to the null terminated string name of the symbol
value	the symbol's address relocatable - an offset from the beginning of the section executable - absolute run time address

Section field

- each symbol is associated with a certain section
- section field
- an index into the section header table
- three pseudo-sections

Section field - pseudo sections

ABS	for symbols that should not be relocated
UNDEF	for undefined symbols referenced in an object file defined in another object file
COMMON	for uninitialized data objects that are not yet allocated value : the alignment requirement size : the minimum size

Displaying section table using `readelf -s`

- `readelf -s swap.o`

```
Num:    Value  Size Type    Bind   Vis      Ndx Name
 16: 00000000     4 OBJECT  GLOBAL DEFAULT    6 p0       ; 6 : .data.rel section
 17: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND buf      ; other .o file
 18: 00000004     4 OBJECT  GLOBAL DEFAULT  COM p1       ; uninitialized
 19: 00000000    78 FUNC    GLOBAL DEFAULT    2 swap     ; 2 : .text section
```

- `readelf -S swap.o`

```
[Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
[ 2] .text             PROGBITS        00000000 00003c 00004e 00  AX  0   0  1
[ 6] .data.rel         PROGBITS        00000000 00008c 000004 00  WA  0   0  4
```

- `swap.c`

```
extern int buf[];
int *p0 = &buf[0];
int *p1;
void swap()
```

```
{ int tmp;
  p1 = &buf[1];
  tmp = *p0;  *p0 = *p1;  *p1 = tmp;
}
```

Displaying section table using objdump -t

- objdump -t swap.o

```
00000000 g     O .data.rel      00000004 p0
00000000         *UND*          00000000 buf
00000004       O *COM*          00000004 p1
00000000 g     F .text          0000004e swap
```

- readelf -s swap.o

```
Num:    Value  Size Type    Bind   Vis      Ndx Name
 16: 00000000     4 OBJECT  GLOBAL DEFAULT    6 p0      ; 6 : .data.rel section
 17: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND buf     ; other .o file
 18: 00000004     4 OBJECT  GLOBAL DEFAULT  COM p1      ; uninitialized
 19: 00000000    78 FUNC    GLOBAL DEFAULT    2 swap    ; 2 : .text section
```

- swap.c

```
extern int buf[];                    { int tmp;
int *p0 = &buf[0];                     p1 = &buf[1];
int *p1;                               tmp = *p0;  *p0 = *p1;  *p1 = tmp;
void swap()                          }
```

Section table fields using `objdump -t`

1. the number is the symbol's value (its address)
2. the set of characters and spaces
   the flag bits that are set on the symbol.
3. the section with which the symbol is associated
   or ABS if the section is absolute (not connected),
   or UND if the section is referenced but not defined
4. a number for common symbols is the alignment
   a number for other symbol is the size.
5. the symbol's name

```
00000000 l     d  .bss   00000000 .bss
00000000 g        .text  00000000 fred
```

Section table flag bits using `objdump -t`

1. local (`l`), global (`g`), unique global (`u`), neither global nor local (a space) or both global and local (`!`)
2. weak (`w`) or strong (a space)
3. constructor (`C`) or an ordinary symbol (a space)
4. a warning (`W`) or a normal symbol (a space).
5. an indirect reference to another symbol (`I`), a function to be evaluated during reloc processing (`i`) or a normal symbol (a space).
6. a debugging symbol (`d`) or a dynamic symbol (`D`) or a normal symbol (a space)
7. the name of a function (`F`) or a file (`f`) or an object (`O`) or just a normal symbol (a space).

```
00000000 l     d  .bss   00000000 .bss
00000000 g        .text  00000000 fred
```

TOC: Symbol Table Examples

Example Programs

```
// main.c --------------------

void swap();

int buf[2]  = {1, 2};

int main()
{
  swap();

  return 0;
}
```

```
// swap.c --------------------

extern int buf[];

int *p0 = &buf[0];
int *p1;

void swap()
{
  int tmp;

  p1 = &buf[1];

  tmp = *p0;
  *p0 = *p1;
  *p1 = tmp;

}
```

objdump -d main.o

```
main.o:     formato del fichero elf32-i386


Desensamblado de la sección .text.startup:

00000000 <main>:
   0:   8d 4c 24 04             lea    0x4(%esp),%ecx
   4:   83 e4 f0                and    $0xfffffff0,%esp
   7:   ff 71 fc                pushl  -0x4(%ecx)
   a:   55                      push   %ebp
   b:   89 e5                   mov    %esp,%ebp
   d:   51                      push   %ecx
   e:   83 ec 04                sub    $0x4,%esp
  11:   e8 fc ff ff ff          call   12 <main+0x12>
  16:   83 c4 04                add    $0x4,%esp
  19:   31 c0                   xor    %eax,%eax
  1b:   59                      pop    %ecx
  1c:   5d                      pop    %ebp
  1d:   8d 61 fc                lea    -0x4(%ecx),%esp
  20:   c3                      ret
```

objdump -d swap.o

```
swap.o:     formato del fichero elf32-i386


Desensamblado de la sección .text:

00000000 <swap>:
   0:   a1 00 00 00 00          mov    0x0,%eax
   5:   8b 0d 04 00 00 00       mov    0x4,%ecx
   b:   c7 05 00 00 00 00 04    movl   $0x4,0x0
  12:   00 00 00
  15:   8b 10                   mov    (%eax),%edx
  17:   89 08                   mov    %ecx,(%eax)
  19:   89 15 04 00 00 00       mov    %edx,0x4
  1f:   c3                      ret
```

readelf output fields

Num	the symbol number
Value	the address of the symbol
Size	the size of the symbol
Type	the symbol type (Func/Object/File/Section/Notype)
Bind	Global/Local/Weak
Vis	default/protected/hidden/internal
Ndx	the section number where the symbol is in / ABS
Name	the symbol nale

readelf symbol type

Func	Function
Object	data / variable
File	source file name
Section	memory section
Notype	untyped absolute symbol or undefined

TOC: main.o's symbol table

Relocatable ELF Symbol Table of main.o

```
Symbol table '.symtab' contains 13 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 00000000     0 FILE    LOCAL  DEFAULT  ABS main.i
     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    4
     6: 00000000     0 SECTION LOCAL  DEFAULT    5
     7: 00000000     0 SECTION LOCAL  DEFAULT    8
     8: 00000000     0 SECTION LOCAL  DEFAULT    9
     9: 00000000     0 SECTION LOCAL  DEFAULT    7
    10: 00000000    33 FUNC    GLOBAL DEFAULT    5 main
    11: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND swap
    12: 00000000     8 OBJECT  GLOBAL DEFAULT    2 buf

No version information found in this file.
```

Section Header Table of main.o

```
Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .text             PROGBITS        00000000 000034 000000 00  AX  0   0  1
  [ 2] .data             PROGBITS        00000000 000034 000008 00  WA  0   0  4
  [ 3] .bss              NOBITS          00000000 00003c 000000 00  WA  0   0  1
  [ 4] .text.unlikely    PROGBITS        00000000 00003c 000000 00  AX  0   0  1
  [ 5] .text.startup     PROGBITS        00000000 000040 000021 00  AX  0   0 16
  [ 6] .rel.text.startup REL             00000000 0001c4 000008 08   I 12   5  4
  [ 7] .comment          PROGBITS        00000000 000061 000035 01  MS  0   0  1
  [ 8] .note.GNU-stack   PROGBITS        00000000 000096 000000 00      0   0  1
  [ 9] .eh_frame         PROGBITS        00000000 000098 000044 00   A  0   0  4
  [10] .rel.eh_frame     REL             00000000 0001cc 000008 08   I 12   9  4
  [11] .shstrtab         STRTAB          00000000 0001d4 000074 00      0   0  1
  [12] .symtab           SYMTAB          00000000 0000dc 0000d0 10     13  10  4
  [13] .strtab           STRTAB          00000000 0001ac 000016 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)
```

Ndx field of main.o

```
Section Headers:
  [Nr] Name
  [ 0]

  [ 1] .text
  [ 2] .data
  [ 3] .bss
  [ 4] .text.unlikely
  [ 5] .text.startup
  [ 6] .rel.text.startup
  [ 7] .comment
  [ 8] .note.GNU-stack
  [ 9] .eh_frame
  [10] .rel.eh_frame
  [11] .shstrtab
  [12] .symtab
  [13] .strtab
```

Local Symbols of main.o

- local symbols
- linker uses internally

```
Symbol table '.symtab' contains 13 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 00000000     0 FILE    LOCAL  DEFAULT  ABS main.i
     2: 00000000     0 SECTION LOCAL  DEFAULT    1  --> .text
     3: 00000000     0 SECTION LOCAL  DEFAULT    2  --> .data
     4: 00000000     0 SECTION LOCAL  DEFAULT    3  --> .bss
     5: 00000000     0 SECTION LOCAL  DEFAULT    4  --> .text.unlikely
     6: 00000000     0 SECTION LOCAL  DEFAULT    5  --> .text.startup
     7: 00000000     0 SECTION LOCAL  DEFAULT    8  --> .note.GNU-stack
     8: 00000000     0 SECTION LOCAL  DEFAULT    9  --> .eh_frame
     9: 00000000     0 SECTION LOCAL  DEFAULT    7  --> .comment
```

Gloabl Symbols of main.o

buf	global	8-byte object	at an offset 0 in .data
main	global	17-byte function	at an offset 0 in .text.startup
swap	global	0-byte external symbol	

```
Symbol table '.symtab' contains 13 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
    10: 00000000    33 FUNC    GLOBAL DEFAULT    5 main  --> .text.startup
    11: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND swap
    12: 00000000     8 OBJECT  GLOBAL DEFAULT    2 buf   --> .data

No version information found in this file.
```

TOC: swap.o's symbol table

Relocatable ELF Symbol Table of swap.o

```
Symbol table '.symtab' contains 13 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 00000000     0 FILE    LOCAL  DEFAULT  ABS swap.i
     2: 00000000     0 SECTION LOCAL  DEFAULT    1
     3: 00000000     0 SECTION LOCAL  DEFAULT    3
     4: 00000000     0 SECTION LOCAL  DEFAULT    5
     5: 00000000     0 SECTION LOCAL  DEFAULT    6
     6: 00000000     0 SECTION LOCAL  DEFAULT    8
     7: 00000000     0 SECTION LOCAL  DEFAULT    9
     8: 00000000     0 SECTION LOCAL  DEFAULT    7
     9: 00000000    32 FUNC    GLOBAL DEFAULT    1 swap
    10: 00000000     4 OBJECT  GLOBAL DEFAULT    3 p0
    11: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND buf
    12: 00000004     4 OBJECT  GLOBAL DEFAULT  COM p1

No version information found in this file.**** A block
```

Section Header Table of swap.o

```
Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .text             PROGBITS        00000000 000040 000020 00  AX  0   0 16
  [ 2] .rel.text         REL             00000000 0001b0 000028 08   I 12   1  4
  [ 3] .data             PROGBITS        00000000 000060 000004 00  WA  0   0  4
  [ 4] .rel.data         REL             00000000 0001d8 000008 08   I 12   3  4
  [ 5] .bss              NOBITS          00000000 000064 000000 00  WA  0   0  1
  [ 6] .text.unlikely    PROGBITS        00000000 000064 000000 00  AX  0   0  1
  [ 7] .comment          PROGBITS        00000000 000064 000035 01  MS  0   0  1
  [ 8] .note.GNU-stack   PROGBITS        00000000 000099 000000 00      0   0  1
  [ 9] .eh_frame         PROGBITS        00000000 00009c 00002c 00   A  0   0  4
  [10] .rel.eh_frame     REL             00000000 0001e0 000008 08   I 12   9  4
  [11] .shstrtab         STRTAB          00000000 0001e8 00006a 00      0   0  1
  [12] .symtab           SYMTAB          00000000 0000c8 0000d0 10     13   9  4
  [13] .strtab           STRTAB          00000000 000198 000017 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)
```

Ndx field of swap.o

```
Section Headers:
  [Nr] Name
  [ 0]
  [ 1] .text
  [ 2] .rel.text
  [ 3] .data
  [ 4] .rel.data
  [ 5] .bss
  [ 6] .text.unlikely
  [ 7] .comment
  [ 8] .note.GNU-stack
  [ 9] .eh_frame
  [10] .rel.eh_frame
  [11] .shstrtab
  [12] .symtab
  [13] .strtab
```

Local Symbols of swap.o

- local symbols
- linker uses internally

```
2: 00000000     0 SECTION LOCAL  DEFAULT    1 --> .text
3: 00000000     0 SECTION LOCAL  DEFAULT    3 --> .data
4: 00000000     0 SECTION LOCAL  DEFAULT    5 --> .bss
5: 00000000     0 SECTION LOCAL  DEFAULT    6 --> .text.unlikely
6: 00000000     0 SECTION LOCAL  DEFAULT    8 --> .note.GNU-stack
7: 00000000     0 SECTION LOCAL  DEFAULT    9 --> .eh_frame
8: 00000000     0 SECTION LOCAL  DEFAULT    7 --> .comment
```

Gloabl Symbols of swap.o

p0	global	4-byte initialized object	at an offset 0 in .data
buf	global	external symbol for init p0	
swap	global	32-byte function	at an offset 0 in .text
p1	global	4-byte uninitialized object	(4-byte alignment, .bss)

```
 9: 00000000    32 FUNC    GLOBAL DEFAULT    1 swap --> .text
10: 00000000     4 OBJECT  GLOBAL DEFAULT    3 p0   --> .data
11: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND buf
12: 00000004     4 OBJECT  GLOBAL DEFAULT  COM p1
```

TOC: Symbol Resolution

Symbol Resolution

The linker resolves symbol reference by associating each reference with exactly one symbol definition from the symbol tables of the input relocatable object files

- symbol resolution
- the linker
- (symbol reference, symbol definition)
- symbol tables
- relocatable object files

Undefined Reference

```
void foo(void);

int main() {
  foo();
  return 0;

}
```

```
/tmp/ccFjbAnB.o: In function ‘main’:
l.c:(.text+0x5): undefined reference to ‘foo’
collect2: error: ld returned 1 exit status
```

Local Symbol Resolution

- a symbol
  - a variable
  - a function
- local symbols
  - defined and referenced in the same module
  - only one definition allowed in a module
  - not all local symbols are linker symbols
- local linker symbols
  - *static* local variables
  - unique name
  - handled by a linker
  - linker symbol table

Global Symbol Resolution

- a symbol
  - a variable
  - a function
- global symbols
  - a symbol that is not defined in the current module
  - assumed that it is defined in some other module
  - linker symbols
  - handled by a linker
  - linker symbol table
- if the linker is unable to find the definition in its input modules print error messages and terminates

Multiply Defined Gloabal Symbols

- when a symbol is defined by multiple object files
    - the linker must either flag an error
    - choose one of the definitions and discard the others

- at compile time,
  the compiler exports each global symbol to the assembler
  as either strong or weak
- the assembler encodes this information implcitly
  in the symbol table of the relocatable object file.

Strong and Weak Symbols

- strong symbols
  - functions
  - *initialized* global variabls
- weak symbols
  - *unitialized* global variabls

Strong and Weak Symbol Examples

```
// main.c --------------------

void swap();

int buf[2]  = {1, 2};

int main()
{
  swap();

  return 0;
}
```

strong symbols:
(buf, p0, main, swap)
weak symbols : (p1)

```
// swap.c --------------------

extern int buf[];

int *p0 = &buf[0];
int *p1;

void swap()
{
  int tmp;

  p1 = &buf[1];

  tmp = *p0;
  *p0 = *p1;
  *p1 = tmp;

}
```

Rules for Multiply Defined Symbols

1. Multiple strong symbols are not allowed
2. Given a strong symbol and multiple weak symbols, choose the strong symbols
3. Given multiple weak symbols, choose any of the weak symbols

Symbol Types

	local variables	global variables, functions
non-static	stack, run time	global linker symbol
static	local linker symbol	local linker symbol

	global variables
initialized	strong symbol
un-initialized	weak symbol

TOC: Multiply Defined Global Symbol Resolution Examples

Example (1) : multiple strong symbols

- main : multiple strong symbols

```
/* t1.c */                          /* t2.c */
int main() {                        int main() {
  return 0;                           return 0;
}                                   }
```

```
$ gcc -c t1.c
$ gcc -c t2.c
$ gcc t1.o t2.o
t2.o: In function ‘main’:
t2.c:(.text+0x0): multiple definition of ‘main’
t1.o:t1.c:(.text+0x0): first defined here
collect2: error: ld returned 1 exit status
```

Example (2) : multiple strong symbols

- x : multiple strong symbols

```
/* t3.c */
int x = 111;

int main() {

  return 0;

}
```

```
/* t4.c */
int x = 222;

void f() {


}
```

```
$ gcc -c t3.c
$ gcc -c t4.c
$ gcc t3.o t4.o
t4.o:(.data+0x0): multiple definition of ‘x’
t3.o:(.data+0x0): first defined here
collect2: error: ld returned 1 exit status
```

Example (3) : strong and weak symbols

- x : a strong symbol in t5.c, a weak symbol in t6.c
- the initial value (111) of a strong symbol is changed into 222 in f()

```
/* t5.c */
#include <stdio.h>

int x = 111;
void f();

int main() {
  f();
  printf("x= %d \n", x);
  return 0;
}

$ gcc -c t5.c
$ gcc -c t6.c
$ gcc t5.o t6.o
$ ./a.out
x= 222
```

```
/* t6.c */
int x;

void f() {

  x = 222;

}
```

Example (4) : weak symbols

- x : weak symbols
- the initial value (111) of a strong symbol is changed into 222 in f()

```
/* t7.c */
#include <stdio.h>
int x;
void f();

int main() {
  x = 111;
  f();
  printf("x= %d \n", x);
  return 0;
}
```

```
/* t8.c */
int x;

void f() {

  x = 222;

}
```

```
$ vi t7.c
$ gcc -c t7.c
$ gcc -c t8.c
$ gcc t7.o t8.o
$ ./a.out
x= 222
```

Example (5) : strong and weak symbols of different types

- x : a strong symbol of int and a weak symbol of double
- the value y is contaminated

```
/* t9.c */
#include <stdio.h>

int x = 111;
int y = 222;
void f();

int main() {
  f();
  printf("x= %d y= %d\n", x, y);
  return 0;
}
```

```
/* t10.c */
double x;

void f() {

  x = -0.0;

}
```

```
$ ./a.out
x= 0 y= -2147483648
```

```
$ gcc -c t9.c
$ gcc -c t10.c
$ gcc t9.o t10.o
/usr/bin/ld: Warning: alignment 4 of symbol ‘x’ in t9.o is
smaller than 8 in t10.o
```