Computers
Compilation Environment & System Libraries
(Extra Slides)

Grau en Ciència i Enginyeria de Dades

Xavier Martorell, Xavier Verdú

Facultat d'Informàtica de Barcelona (FIB)

Universitat Politècnica de Catalunya (UPC)

2020-2021 Q2

Creative Commons License

Table of Contents

Example Source Code

- Prog.c
  - Main function
  - Global variable
  - Use a variable and a function declared and defined in another module
- Myheader.c
  - Variable and function used by other codes, such as “prog.c”
- All codes and file resources discussed in these slides can be downloaded from this link
- The aim of this doc is to clarify the management and use of Symbol Table and Relocation Records

Example Source Code

- “myvar” is declared and defined in “myheader.h” and used in “prog.c”

prog.c

```c
#include <unistd.h>
#include <stdio.h>
#include "myheader.h"

extern int myvar;

int global = 4;

void mifunc(int op){
        global = op;
}

int main (int argc, char **argv){
        char buf[512];
        int len, result;

        mifunc(10);
        myvar = 20;
        result=suma(global, 30);
        myvar +=result;

        len = sprintf(buf, "Hello World %d, %d\n", result, myvar);
        write(1, buf, len);
        return 0;
}
```

myheader.h

```c
int suma(int op1, int op2);
```

myheader.c

```c
int myvar = 10;

int suma(int op1, int op2){
        int tmp;

        tmp = op1 + op2;
        return tmp;
}
```

Example Source Code

- “myvar” is declared and defined in “myheader.h” and used in “prog.c”

prog.c

```c
#include <unistd.h>
#include <stdio.h>
#include "myheader.h"

extern int myvar;

int global = 4;

void mifunc(int op){
        global = op;
}

int main (int argc, char **argv){
        char buf[512];
        int len, result;

        mifunc(10);
        myvar = 20;
        result=suma(global, 30);
        myvar +=result;

        len = sprintf(buf, "Hello World %d, %d\n", result, myvar);
        write(1, buf, len);
        return 0;
}
```

myheader.h

```c
int suma(int op1, int op2);
```

myheader.c

```c
int myvar = 10;

int suma(int op1, int op2){
        int tmp;

        tmp = op1 + op2;
        return tmp;
}
```

Symbol Table

- A table of memory location and variable or function that can be called from other object files
  - Also it includes symbols used in the object file

<address> <size> Symbol

Symbol Table

- A table of memory location and variable or function that can be called from other object files
  - Also it includes symbols used in the object file

<address> <size> Symbol

```
prog.o:     file format elf64-x86-64

SYMBOL TABLE:
0000000000000000 l    df *ABS*  0000000000000000 prog.c
0000000000000000 l    d  .text  0000000000000000 .text
0000000000000000 l    d  .data  0000000000000000 .data
0000000000000000 l    d  .bss   0000000000000000 .bss
0000000000000000 l    d  .rodata        0000000000000000 .rodata
0000000000000000 l    d  .note.GNU-stack        0000000000000000 .note.GNU-stack
0000000000000000 l    d  .eh_frame      0000000000000000 .eh_frame
0000000000000000 l    d  .comment       0000000000000000 .comment
0000000000000000 g     O .data  0000000000000004 global
0000000000000000 g     F .text  0000000000000013 mifunc
0000000000000013 g     F .text  00000000000000cc main
0000000000000000         *UND*  0000000000000000 myvar
0000000000000000         *UND*  0000000000000000 _GLOBAL_OFFSET_TABLE_
0000000000000000         *UND*  0000000000000000 suma
0000000000000000         *UND*  0000000000000000 sprintf
0000000000000000         *UND*  0000000000000000 write
0000000000000000         *UND*  0000000000000000 __stack_chk_fail
```

```
myheader.o:     file format elf64-x86-64

SYMBOL TABLE:
0000000000000000 l    df *ABS*  0000000000000000 myheader.c
0000000000000000 l    d  .text  0000000000000000 .text
0000000000000000 l    d  .data  0000000000000000 .data
0000000000000000 l    d  .bss   0000000000000000 .bss
0000000000000000 l    d  .note.GNU-stack        0000000000000000 .note.GNU-stack
0000000000000000 l    d  .eh_frame      0000000000000000 .eh_frame
0000000000000000 l    d  .comment       0000000000000000 .comment
0000000000000000 g     O .data  0000000000000004 myvar
0000000000000000 g     F .text  000000000000001a suma
```

Symbol Table

- A table of memory location and variable or function that can be called from other object files
  - Also it includes symbols used in the object file

<address> <size> Symbol

```
prog.o:     file format elf64-x86-64

SYMBOL TABLE:
0000000000000000 l    df *ABS*  0000000000000000 prog.c
0000000000000000 l    d  .text  0000000000000000 .text
0000000000000000 l    d  .data  0000000000000000 .data
0000000000000000 l    d  .bss   0000000000000000 .bss
0000000000000000 l    d  .note.GNU-stack 0000000000000000 .note.GNU-stack
0000000000000000 l    d  .eh_frame 0000000000000000 .eh_frame
0000000000000000 l    d  .comment 0000000000000000 .comment
0000000000000000 g     O .data  0000000000000004 global
0000000000000000 g     F .text  0000000000000013 mifunc
0000000000000013 g     F .text  00000000000000cc main
0000000000000000         *UND*  0000000000000000 myvar
0000000000000000         *UND*  0000000000000000 _GLOBAL_OFFSET_TABLE_
0000000000000000         *UND*  0000000000000000 suma
0000000000000000         *UND*  0000000000000000 sprintf
0000000000000000         *UND*  0000000000000000 write
0000000000000000         *UND*  0000000000000000 __stack_chk_fail
```

In the Symbol Table of “prog.o”:
“myvar” is undefined

```
myheader.o:     file format elf64-x86-64

SYMBOL TABLE:
0000000000000000 l    df *ABS*  0000000000000000 myheader.c
0000000000000000 l    d  .text  0000000000000000 .text
0000000000000000 l    d  .data  0000000000000000 .data
0000000000000000 l    d  .bss   0000000000000000 .bss
0000000000000000 l    d  .note.GNU-stack 0000000000000000 .note.GNU-stack
0000000000000000 l    d  .eh_frame 0000000000000000 .eh_frame
0000000000000000 l    d  .comment 0000000000000000 .comment
0000000000000000 g     O .data  0000000000000004 myvar
0000000000000000 g     F .text  000000000000001a suma
```

In the Symbol Table of “myheader.o”:
“myvar” can be found in offset “0” in “.data” section and its size is 4 bytes

Relocation Records

- Information about addresses referenced in the object file. The linker adjust them once the final memory allocation is resolved
  - References to symbols that cannot be resolved
  - The data allows calculate what bytes have to be modified when the symbols are resolved

<address of the modification> <type> <how to compute the address>

Relocation Records

- Information about addresses referenced in the object file. The linker adjust them once the final memory allocation is resolved

```c
#include <unistd.h>
#include <stdio.h>
#include "myheader.h"

extern int myvar;

int global = 4;

void mifunc(int op){
        global = op;
}

int main (int argc, char **argv){
        char buf[512];
        int len, result;

        mifunc(10);
        myvar = 20;
        result=suma(global, 30);
        myvar +=result;

        len = sprintf(buf, "Hello World %d, %d\n", result, myvar);
        write(1, buf, len);
        return 0;
}
```

```
0000000000000013 <main>:
  13:   55                      push   %rbp
  14:   48 89 e5                mov    %rsp,%rbp
  17:   48 81 ec 30 02 00 00    sub    $0x230,%rsp
  1e:   89 bd dc fd ff ff       mov    %edi,-0x224(%rbp)
  24:   48 89 b5 d0 fd ff ff    mov    %rsi,-0x230(%rbp)
  2b:   64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
  32:   00 00
  34:   48 89 45 f8             mov    %rax,-0x8(%rbp)
  38:   31 c0                   xor    %eax,%eax
  3a:   bf 0a 00 00 00          mov    $0xa,%edi
  3f:   e8 00 00 00 00          callq  44 <main+0x31>
  44:   c7 05 00 00 00 00 14    movl   $0x14,0x0(%rip)        # 4e <main+0x3b>
  4b:   00 00 00
  4e:   8b 05 00 00 00 00       mov    0x0(%rip),%eax        # 54 <main+0x41>
  54:   be 1e 00 00 00          mov    $0x1e,%esi
  59:   89 c7                   mov    %eax,%edi
  5b:   e8 00 00 00 00          callq  60 <main+0x4d>
  60:   89 85 e8 fd ff ff       mov    %eax,-0x218(%rbp)
  66:   8b 15 00 00 00 00       mov    0x0(%rip),%edx        # 6c <main+0x59>
  6c:   8b 85 e8 fd ff ff       mov    -0x218(%rbp),%eax
  72:   01 d0                   add    %edx,%eax
```

Relocation Records

- Information about addresses referenced in the object file. The linker adjust them once the final memory allocation is resolved
  - References to symbols that cannot be resolved
  - The data allows calculate what bytes have to be modified when the symbols are resolved

```
prog.o:     file format elf64-x86-64

RELOCATION RECORDS FOR [.text]:
OFFSET           TYPE              VALUE
000000000000000c R_X86_64_PC32     global-0x0000000000000004
0000000000000040 R_X86_64_PC32     mifunc-0x0000000000000004
0000000000000046 R_X86_64_PC32     myvar-0x0000000000000008
0000000000000050 R_X86_64_PC32     global-0x0000000000000004
000000000000005c R_X86_64_PLT32    suma-0x0000000000000004
0000000000000068 R_X86_64_PC32     myvar-0x0000000000000004
0000000000000076 R_X86_64_PC32     myvar-0x0000000000000004
000000000000007c R_X86_64_PC32     myvar-0x0000000000000004
0000000000000090 R_X86_64_PC32     .rodata-0x0000000000000004
000000000000009d R_X86_64_PLT32    sprintf-0x0000000000000004
00000000000000c0 R_X86_64_PLT32    write-0x0000000000000004
00000000000000d9 R_X86_64_PLT32    __stack_chk_fail-0x0000000000000004
```

Relocation Records

- Information about addresses referenced in the object file. The linker adjust them once the final memory allocation is resolved
  - References to symbols that cannot be resolved
  - The data allows calculate what bytes have to be modified when the symbols are resolved

```
prog.o:     file format elf64-x86-64

RELOCATION RECORDS FOR [.text]:
OFFSET           TYPE              VALUE
000000000000000c R_X86_64_PC32     global-0x0000000000000004
0000000000000040 R_X86_64_PC32     mifunc-0x0000000000000004
0000000000000046 R_X86_64_PC32     myvar-0x0000000000000008
0000000000000050 R_X86_64_PC32     global-0x0000000000000004
000000000000005c R_X86_64_PLT32    suma-0x0000000000000004
0000000000000068 R_X86_64_PC32     myvar-0x0000000000000004
0000000000000076 R_X86_64_PC32     myvar-0x0000000000000004
000000000000007c R_X86_64_PC32     myvar-0x0000000000000004
0000000000000090 R_X86_64_PC32     .rodata-0x0000000000000004
000000000000009d R_X86_64_PLT32    sprintf-0x0000000000000004
00000000000000c0 R_X86_64_PLT32    write-0x0000000000000004
00000000000000d9 R_X86_64_PLT32    __stack_chk_fail-0x0000000000000004
```

```
0000000000000013 <main>:
  13:   55                      push   %rbp
  14:   48 89 e5                mov    %rsp,%rbp
  17:   48 81 ec 30 02 00 00    sub    $0x230,%rsp
  1e:   89 bd dc fd ff ff       mov    %edi,-0x224(%rbp)
  24:   48 89 b5 d0 fd ff ff    mov    %rsi,-0x230(%rbp)
  2b:   64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
  32:   00 00
  34:   48 89 45 f8             mov    %rax,-0x8(%rbp)
  38:   31 c0                   xor    %eax,%eax
  3a:   bf 0a 00 00 00          mov    $0xa,%edi
  3f:   e8 00 00 00 00          callq  44 <main+0x31>
  44:   c7 05 00 00 00 00 14    movl   $0x14,0x0(%rip)        # 4e <main+0x3b>
  4b:   00 00 00
  4e:   8b 05 00 00 00 00       mov    0x0(%rip),%eax        # 54 <main+0x41>
  54:   be 1e 00 00 00          mov    $0x1e,%esi
  59:   89 c7                   mov    %eax,%edi
  5b:   e8 00 00 00 00          callq  60 <main+0x4d>
  60:   89 85 e8 fd ff ff       mov    %eax,-0x218(%rbp)
  66:   8b 15 00 00 00 00       mov    0x0(%rip),%edx        # 6c <main+0x59>
  6c:   8b 85 e8 fd ff ff       mov    -0x218(%rbp),%eax
  72:   01 d0                   add    %edx,%eax
```

Relocation Records

- Information about addresses referenced in the object file. The linker adjust them once the final memory allocation is resolved
  - References to symbols that cannot be resolved
  - The data allows calculate what bytes have to be modified when the symbols are resolved

```
prog.o:     file format elf64-x86-64

RELOCATION RECORDS FOR [.text]:
OFFSET           TYPE              VALUE
000000000000000c R_X86_64_PC32     global-0x0000000000000004
0000000000000040 R_X86_64_PC32     mifunc-0x0000000000000004
0000000000000046 R_X86_64_PC32     myvar-0x0000000000000008
0000000000000050 R_X86_64_PC32     global-0x0000000000000004
000000000000005c R_X86_64_PLT32    suma-0x0000000000000004
0000000000000068 R_X86_64_PC32     myvar-0x0000000000000004
0000000000000076 R_X86_64_PC32     myvar-0x0000000000000004
000000000000007c R_X86_64_PC32     myvar-0x0000000000000004
0000000000000090 R_X86_64_PC32     .rodata-0x0000000000000004
000000000000009d R_X86_64_PLT32    sprintf-0x0000000000000004
00000000000000c0 R_X86_64_PLT32    write-0x0000000000000004
00000000000000d9 R_X86_64_PLT32    __stack_chk_fail-0x0000000000000004
```

```
0000000000000013 <main>:
  13:   55                      push   %rbp
  14:   48 89 e5                mov    %rsp,%rbp
  17:   48 81 ec 30 02 00 00    sub    $0x230,%rsp
  1e:   89 bd dc fd ff ff       mov    %edi,-0x224(%rbp)
  24:   48 89 b5 d0 fd ff ff    mov    %rsi,-0x230(%rbp)
  2b:   64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
  32:   00 00
  34:   48 89 45 f8             mov    %rax,-0x8(%rbp)
  38:   31 c0                   xor    %eax,%eax
  3a:   bf 0a 00 00 00          mov    $0xa,%edi
  3f:   e8 00 00 00 00          callq  44 <main+0x31>
  44:   c7 05 00 00 00 00 14    movl   $0x14,0x0(%rip)        # 4e <main+0x3b>
  4b:   00 00 00
  4e:   8b 05 00 00 00 00       mov    0x0(%rip),%eax        # 54 <main+0x41>
  54:   be 1e 00 00 00          mov    $0x1e,%esi
  59:   89 c7                   mov    %eax,%edi
  5b:   e8 00 00 00 00          callq  60 <main+0x4d>
  60:   89 85 e8 fd ff ff       mov    %eax,-0x218(%rbp)
  66:   8b 15 00 00 00 00       mov    0x0(%rip),%edx        # 6c <main+0x59>
  6c:   8b 85 e8 fd ff ff       mov    -0x218(%rbp),%eax
  72:   01 d0                   add    %edx,%eax
```

Relocation Records

- Information about addresses referenced in the object file. The linker adjust them once the final memory allocation is resolved
  - References to symbols that cannot be resolved
  - The data allows calculate what bytes have to be modified when the symbols are resolved

RELOCATION RECORD

<address of the modification> <type> <how to compute the address>

```
0000000000000046 R_X86_64_PC32     myvar-0x0000000000000008
```

Exact address to be modified (marked in red)

It indicates it is a PC-relative offset (%rip; that is, relative instruction pointer). Thus, “myvar” address is relative to the instruction pointer

How to calculate the address after being resolved (marked in green)

```
0000000000000013 <main>:
  13:   55                      push   %rbp
  14:   48 89 e5                mov    %rsp,%rbp
  17:   48 81 ec 30 02 00 00    sub    $0x230,%rsp
  1e:   89 bd dc fd ff ff       mov    %edi,-0x224(%rbp)
  24:   48 89 b5 d0 fd ff ff    mov    %rsi,-0x230(%rbp)
  2b:   64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
  32:   00 00
  34:   48 89 45 f8             mov    %rax,-0x8(%rbp)
  38:   31 c0                   xor    %eax,%eax
  3a:   bf 0a 00 00 00          mov    $0xa,%edi
  3f:   e8 00 00 00 00          callq  44 <main+0x31>
  44:   c7 05 00 00 00 00 14    movl   $0x14,0x0(%rip)        # 4e <main+0x3b>
  4b:   00 00 00
  4e:   8b 05 00 00 00 00       mov    0x0(%rip),%eax        # 54 <main+0x41>
  54:   be 1e 00 00 00          mov    $0x1e,%esi
  59:   89 c7                   mov    %eax,%edi
  5b:   e8 00 00 00 00          callq  60 <main+0x4d>
  60:   89 85 e8 fd ff ff       mov    %eax,-0x218(%rbp)
  66:   8b 15 00 00 00 00       mov    0x0(%rip),%edx        # 6c <main+0x59>
  6c:   8b 85 e8 fd ff ff       mov    -0x218(%rbp),%eax
  72:   01 d0                   add    %edx,%eax
```

Relocation Records

- Information about addresses referenced in the object file. The linker adjust them once the final memory allocation is resolved
  - References to symbols that cannot be resolved
  - The data allows calculate what bytes have to be modified when the symbols are resolved

RELOCATION RECORD

<address of the modification> <type> <how to compute the address>

```
0000000000000046 R_X86_64_PC32     myvar-0x0000000000000008
```

Exact address to be modified (marked in red)

It indicates it is a PC-relative offset (%rip; that is, relative instruction pointer). Thus, “myvar” address is relative to the instruction pointer

How to calculate the address after being resolved (marked in green)

“myvar” – 0x08 indicates that:
Once we know the final “myvar” address (@myvar), we have to calculate:
@ToInclude = @myvar-PC-0x08

Building the Executable

SYMBOL TABLE OF THE EXECUTABLE

```
0000000000000000 l    df *ABS*  0000000000000000              myheader.c
0000000000000000 l    df *ABS*  0000000000000000              crtstuff.c
0000000000000a2c l     O .eh_frame      0000000000000000                __FRAME_END__
0000000000000000 l    df *ABS*  0000000000000000
0000000000200db0 l       .init_array    0000000000000000                __init_array_end
0000000000200db8 l     O .dynamic       0000000000000000                _DYNAMIC
0000000000200da8 l       .init_array    0000000000000000                __init_array_start
0000000000000898 l       .eh_frame_hdr  0000000000000000                __GNU_EH_FRAME_HDR
0000000000200fa8 l     O .got   0000000000000000              _GLOBAL_OFFSET_TABLE_
0000000000000870 g     F .text  0000000000000002              __libc_csu_fini
0000000000000000  w      *UND*  0000000000000000              _ITM_deregisterTMCloneTable
0000000000201000  w      .data  0000000000000000              data_start
0000000000000000       F *UND*  0000000000000000              write@@GLIBC_2.2.5
00000000000006fa g     F .text  0000000000000013              mifunc
0000000000201018 g       .data  0000000000000000              _edata
0000000000000874 g     F .fini  0000000000000000              _fini
0000000000201010 g     O .data  0000000000000004              global
0000000000000000       F *UND*  0000000000000000              __stack_chk_fail@@GLIBC_2.4
00000000000007d9 g     F .text  000000000000001a              suma
0000000000201014 g     O .data  0000000000000004              myvar
0000000000000000       F *UND*  0000000000000000              __libc_start_main@@GLIBC_2.2.5
0000000000201000 g       .data  0000000000000000              __data_start
```

"myvar" – 0x08 indicates that:
Once known the final "myvar" address (@myvar), the linker calculates the @ to include:
@ToInclude = @myvar-PC-0x08
@ToInclude = 0x201014 – PC – 0x08

Building the Executable

CODE

```
000000000000070d <main>:
 70d:   55                      push   %rbp
 70e:   48 89 e5                mov    %rsp,%rbp
 711:   48 81 ec 30 02 00 00    sub    $0x230,%rsp
 718:   89 bd dc fd ff ff       mov    %edi,-0x224(%rbp)
 71e:   48 89 b5 d0 fd ff ff    mov    %rsi,-0x230(%rbp)
 725:   64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
 72c:   00 00
 72e:   48 89 45 f8             mov    %rax,-0x8(%rbp)
 732:   31 c0                   xor    %eax,%eax
 734:   bf 0a 00 00 00          mov    $0xa,%edi
 739:   e8 bc ff ff ff          callq  6fa <mifunc>
 73e:   c7 05 cc 08 20 00 14    movl   $0x14,0x2008cc(%rip)        # 201014 <myvar>
 745:   00 00 00
 748:   8b 05 c2 08 20 00       mov    0x2008c2(%rip),%eax        # 201010 <global>
 74e:   be 1e 00 00 00          mov    $0x1e,%esi
 753:   89 c7                   mov    %eax,%edi
 755:   e8 7f 00 00 00          callq  7d9 <suma>
 75a:   89 85 e8 fd ff ff       mov    %eax,-0x218(%rbp)
 760:   8b 15 ae 08 20 00       mov    0x2008ae(%rip),%edx        # 201014 <myvar>
 766:   8b 85 e8 fd ff ff       mov    -0x218(%rbp),%eax
 76c:   01 d0                   add    %edx,%eax
 76e:   89 05 a0 08 20 00       mov    %eax,0x2008a0(%rip)        # 201014 <myvar>
 774:   8b 0d 9a 08 20 00       mov    0x20089a(%rip),%ecx        # 201014 <myvar>
 77a:   8b 95 e8 fd ff ff       mov    -0x218(%rbp),%edx
```

"myvar" – 0x08 indicates that:
Once known the final "myvar" address (@myvar), the linker calculates the @ to include:
@ToInclude = @myvar-PC-0x08
@ToInclude = 0x201014 – PC – 0x08
@ToInclude = 0x201014 – 0x740 – 0x08
@ToInclude = 0x201014 – 0x748
@ToInclude = 0x2008cc

The Executable

- The Symbol Table has most of the symbols solved
  - Except the symbols that belong to Dynamic Libraries if the executable uses them

- The Relocation Records are empty
  - Except the references to elements of Dynamic Libraries
    - objdump –r
      - To list relocation records for Static elements
    - objdump –R
      - To list relocation records for Dynamic elements