# 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**

This work is under a Creative Commons Attribution 4.0 Unported License



The details of this license are publicly available at https://creativecommons.org/licenses/by-nc-nd/4.0

## **Table of Contents**

- Example of source code
- Symbol Table & Relocation Records
  - Object File vs Executable

# 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"

```
#include <unistd.h>
                                                                                                           int suma(int op1, int op2);
                                                                                   myheader.h
                                           prog.c
#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;
                                                                                                             int myvar = 10;
       mifunc(10);
                                                                                                             int suma(int op1, int op2){
       myvar = 20;
                                                                                   myheader.c
                                                                                                                     int tmp;
       result=suma(global, 30);
       myvar +=result;
                                                                                                                     tmp = op1 + op2;
                                                                                                                     return tmp;
       len = sprintf(buf, "Hello World %d, %d\n", result, myvar);
       write(1, buf, len);
       return 0;
                                                                                                                                     5
```

# **Example Source Code**

• "myvar" is declared and defined in "myheader.h" and used in "prog.c"

```
#include <unistd.h>
                                                                                                           int suma(int op1, int op2);
                                                                                   myheader.h
                                           prog.c
#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;
                                                                                                             int myvar = 10;
        mifunc(10);
                                                                                                             int suma(int op1, int op2){
        myvar = 20;
                                                                                   myheader.c
                                                                                                                     int tmp;
        result=suma(global, 30);
       myvar +=result;
                                                                                                                     tmp = op1 + op2;
                                                                                                                      return tmp;
        len = sprintf(buf, "Hello World %d, %d\n", result, myvar);
       write(1, buf, len);
        return 0;
                                                                                                                                     6
```

# 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

00000000000000000

- 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

0000000000000000 stack chk fail

<address> <size> Symbol

```
mvheader.o:
                                                                                                    file format elf64-x86-64
           file format elf64-x86-64
prog.o:
SYMBOL TABLE:
                                                                                 SYMBOL TABLE:
00000000000000000 1
                               000000000000000 prog.c
                                                                                 000000000000000000001
                                                                                                           df *ABS*
                                                                                                                      00000000000000000 myheader.c
000000000000000000 1
                        .text
                               00000000000000000
                                                                                                                      0000000000000000 .text
                                                                                 000000000000000000001
                                                                                                              .text
                        .data
000000000000000000 1
                               000000000000000000
                                                .data
                                                                                 000000000000000000 1
                                                                                                               .data
                                                                                                                      00000000000000000
                                                                                                                                          .data
.bss
                               0000000000000000 .bss
                                                                                                                      0000000000000000 .bss
                                                                                 0000000000000000000001
                                                                                                               .bss
000000000000000000
                        .rodata
                                       0000000000000000 .rodata
                                                                                 0000000000000000 l
                                                                                                                                        0000000000000000 .note.GNU-stack
                                                                                                              .note.GNU-stack
000000000000000000
                        .note.GNU-stack
                                               0000000000000000 .note.GNU-stack
                                       0000000000000000 .eh frame
000000000000000000 1
                         .eh frame
                                                                                 00000000000000000 l
                                                                                                              .eh frame
                                                                                                                               0000000000000000 .eh frame
0000000000000000000001
                                       0000000000000000 .comment
                        .comment
                                                                                 0000000000000000 l
                                                                                                                               0000000000000000 .comment
                                                                                                              .comment
0000000000000000 q
                               0000000000000004 global
                                                                                 00000000000000000 q
                                                                                                                      00000000000000004 myvar
00000000000000000 q
                               0000000000000013 mifunc
                                                                                 00000000000000000 q
                                                                                                                      000000000000001a suma
0000000000000013 q
                       F .text
                               000000000000000cc main
                         *UND*
00000000000000000
                               0000000000000000 myvar
                         *UND*
                               00000000000000000
00000000000000000
                                                GLOBAL OFFSET TABLE
00000000000000000
                         *UND*
                               00000000000000000 suma
00000000000000000
                         *UND*
                               0000000000000000 sprintf
00000000000000000
                         *UND*
                               0000000000000000 write
```

# Symbol Table

\*UND\*

\*UND\*

00000000000000000

00000000000000000

00000000000000000

00000000000000000 sprintf

00000000000000000 stack chk fail

00000000000000000 write

- 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>

```
mvheader.o:
                                                                                          file format elf64-x86-64
          file format elf64-x86-64
prog.o:
SYMBOL TABLE:
                                                                          SYMBOL TABLE:
                                                                          0000000000000000000001
                                                                                                 df *ABS*
                                                                                                           00000000000000000 myheader.c
000000000000000000 1
                                                                          0000000000000000 .text
                                                                                                    .text
000000000000000000 1
                                                                          000000000000000000 1
                                                                                                    .data
                                                                                                           00000000000000000
                                                                          00000000000000000000001
                                                                                                    .bss
                                                                                                           0000000000000000 .bss
     In the Symbol Table of "prog.o":
0000
                                                                          00000000000000000 l
                                                                                                    .note.GNU-stack
                                                                                                                            0000000000000000 .note.GNU-stack
                                                               .GNU-stack
00000
      'myvar" is undefined
                                                                          00000000000000000 l
                                                                                                    .eh frame
                                                                                                                   0000000000000000 .eh frame
                                                                          0000000000000000000
                                                                                                                   0000000000000000 .comment
                                                                                                     .comment
                                                                          0000000000000000 g
                                                                                                           00000000000000004 myvar
                                                                                                  0 .data
                                                                          000000000000000000 q
                                                                                                           000000000000001a suma
                                            GLOBAL OFFSET TABLE
                                                                               In the Symbol Table of "myheader.o":
```

Symbol

"myvar" can be found in offset "0" in ".data"

section and its size is 4 bytes

- 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>

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

```
#include <unistd.h>
#include <stdio.h>
#include "myheader.h"
                                                                                        0000000000000013 <main>:
extern int myvar;
                                                                                          13:
                                                                                                55
                                                                                                                        push
                                                                                                                              %rbp
                                                                                                48 89 e5
                                                                                                                              %rsp,%rbp
                                                                                                                        mov
int global = 4;
                                                                                                48 81 ec 30 02 00 00
                                                                                                                               $0x230,%rsp
                                                                                                89 bd dc fd ff ff
                                                                                                                        mov
                                                                                                                               %edi,-0x224(%rbp)
void mifunc(int op){
                                                                                                48 89 b5 d0 fd ff ff
                                                                                                                       mov
                                                                                                                              %rsi,-0x230(%rbp)
         global = op;
                                                                                                                              %fs:0x28,%rax
                                                                                                64 48 8b 04 25 28 00
                                                                                                                        mov
                                                                                                00 00
                                                                                                48 89 45 f8
                                                                                                                              %rax,-0x8(%rbp)
                                                                                          34:
                                                                                                31 c0
                                                                                                                               %eax,%eax
                                                                                                                        xor
int main (int argc, char **argv){
                                                                                                bf 0a 00 00 00
                                                                                                                               $0xa,%edi
                                                                                                                        mov
         char buf[512];
                                                                                                e8 00 00 00 00
                                                                                                                              44 < main + 0 \times 31 >
         int len, result;
                                                                                                c7 05 00 00 00 00 14
                                                                                                                               $0x14,0x0(%rip)
                                                                                                                                                     # 4e <main+0x3b>
                                                                                                00 00 00
                                                                                                8b 05 00 00 00 00
                                                                                                                              0x0(%rip),%eax
                                                                                                                                                    # 54 <main+0x41>
         mifunc(10);
                                                                                                                        mov
                                                                                                be 1e 00 00 00
                                                                                                                               $0x1e,%esi
                                                                                                                        mov
         myvar = 20;
                                                                                          59:
                                                                                                89 c7
                                                                                                                               %eax,%edi
                                                                                                                        mov
         result=suma(global, 30);
                                                                                          5b:
                                                                                                e8 00 00 00 00
                                                                                                                        callq 60 <main+0x4d>
         myvar +=result;
                                                                                                89 85 e8 fd ff ff
                                                                                                                        mov
                                                                                                                               %eax,-0x218(%rbp)
                                                                                                8b 15 00 00 00 00
                                                                                                                               0x0(%rip),%edx
                                                                                                                                                    # 6c <main+0x59>
                                                                                                                        mov
         len = sprintf(buf, "Hello World %d, %d\n", result, myvar);
                                                                                                8b 85 e8 fd ff ff
                                                                                                                               -0x218(%rbp),%eax
                                                                                                                        mov
         write(1, buf, len);
                                                                                                01 d0
                                                                                          72:
                                                                                                                        add
                                                                                                                               %edx,%eax
         return 0;
                                                                                                                                                         11
```

- 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

```
file format elf64-x86-64
prog.o:
RELOCATION RECORDS FOR [.text]:
OFFSET
                 TYPE
                                    VALUE
0000000000000000 R X86 64 PC32
                                    global-0x00000000000000004
00000000000000000 R X86 64 PC32
                                    mifunc-0x000000000000000004
00000000000000046 R X86 64 PC32
                                    myvar-0x00000000000000008
00000000000000050 R X86 64 PC32
                                    global-0x00000000000000004
0000000000000005c R X86 64 PLT32
                                    suma-0x000000000000000004
00000000000000068 R X86 64 PC32
                                    myvar-0x00000000000000004
00000000000000076 R X86 64 PC32
                                    myvar-0x00000000000000004
0000000000000007c R X86 64 PC32
                                    mvvar-0x000000000000000004
00000000000000000 R X86 64 PC32
                                    .rodata-0x0000000000000004
                                    sprintf-0x0000000000000004
0000000000000000 R X86 64 PLT32
000000000000000 R X86 64 PLT32
                                    write-0x00000000000000004
00000000000000d9 R X86 64 PLT32
                                     stack chk fail-0x00000000000000004
```

- 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

0000000000000013 <main>:

6'1 6 1 1604 00 04			0000000000015 \\\\\\\\\\\\\\\\\\\\\\\\\\					
prog.o:	file format elf64-x86	-64	13:	55	push	%rbp		
			14:	48 89 e5	mov	%rsp,%rbp		
RELOCATION F	RECORDS FOR [.text]:		17:	48 81 ec 30 02 00 00	sub	\$0x230,%rsp		
0FFSET	TYPE	VALUE	le:	89 bd dc fd ff ff	mov	%edi,-0x224(%rbp)		
	0000c R X86 64 PC32	global-0x0000000000000004	24:	48 89 b5 d0 fd ff ff	mov	%rsi,-0x230(%rbp)		
			2b:	64 48 8b 04 25 28 00	mov	%fs:0x28,%rax		
000000000000000000000000000000000000000	00040 R X86 64 PC32	<u>mifunc-0x000000000000004</u>	32:	00 00				
0000000000000	00046 R X86 64 PC32	myvar-0x00000000000000000	34:	48 89 45 f8	mov	%rax,-0x8(%rbp)		
0000000000000	00050 R X86 64 PC32	global-0x0000000000000004	38:	31 c0	xor	%eax,%eax		
			3a:	bf 0a 00 00 00	mov	\$0xa,%edi		
	9005c R_X86_64_PLT32	suma-0x0000000000000004	3f:	e8 00 00 00 00	callq	44 <main+0x31></main+0x31>		
0000000000000	00068 R_X86_64_PC32	myvar-0x0000000000000000	44:	c7 05 00 00 00 00 14	movl	\$0x14,0x0(%rip)	# 4e <main+0x3b></main+0x3b>	
0000000000000000	00076 R X86 64 PC32	myvar-0x000000000000004	4b:	00 00 00				
0000000000000	0007c R X86 64 PC32	myvar-0x0000000000000004	4e:	8b 05 00 00 00 00	mov	0x0(%rip),%eax	# 54 <main+0x41></main+0x41>	
	00090 R X86 64 PC32	.rodata-0x0000000000000004	54:	be 1e 00 00 00	mov	\$0x1e,%esi		
			59:	89 c7	mov	%eax,%edi		
	0009d R_X86_64_PLT32	sprintf-0x0000000000000004	5b:	e8 00 00 00 00	callq	60 <main+0x4d></main+0x4d>		
0000000000000	000c0 R X86 64 PLT32	write-0x000000000000004	60:	89 85 e8 fd ff ff	mov .	%eax,-0x218(%rbp)		
000000000000	000d9 R X86 64 PLT32	stack chk fail-0x000000000000000004	1 66:	8b 15 00 00 00 00	mov	0x0(%rip),%edx	# 6c <main+0x59></main+0x59>	
			6c:	8b 85 e8 fd ff ff	mov	-0x218(%rbp),%eax		
			72:	01 d0	add	%edx,%eax		
			,		5. 5. 6			

- 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

00000000000000013 <main>:

617 6 1 7664 00 64			0000000000015 \limati>.					
prog.o:	file format elf64-x86	-64	13:	55	push	%rbp		
			14:	48 89 e5	mov	%rsp,%rbp		
RELOCATION R	RECORDS FOR [.text]:		17:	48 81 ec 30 02 00 00	sub	\$0x230,%rsp		
0FFSET	TYPE	VALUE	1e:	89 bd dc fd ff ff	mov	%edi,-0x224(%rbp)		
	0000c R X86 64 PC32	global-0x0000000000000004	24:	48 89 b5 d0 fd ff ff	mov	%rsi,-0x230(%rbp)		
			2b:	64 48 8b 04 25 28 00	mov	%fs:0x28,%rax		
<u>0000000000000</u>	0040 R X86 64 PC32	<u>mifunc-0x000000000000004</u>	32:	00 00				
0000000000000	00046 R X86 64 PC32	myvar-0x0000000000000000	34:	48 89 45 f8	mov	%rax,-0x8(%rbp)		
0000000000000	00050 R X86 64 PC32	global-0x0000000000000004	38:	31 c0	xor	%eax,%eax		
		5	3a:	bf 0a 00 00 00	mov	\$0xa,%edi		
	0005c R_X86_64_PLT32	suma-0x000000000000004	3f:	e8 00 00 00 00	callq	44 <main+0x31></main+0x31>		
• 00000000000000	00068 R_X86_64_PC32	myvar-0x0000000000000004	44:	c7 05 00 00 00 00 14	movl	\$0x14,0x0(%rip)	# 4e <main+0x3b></main+0x3b>	
• 0000000000000	0076 R X86 64 PC32	myvar-0x000000000000004	4b:	00 00 00		-		
0000000000000	007c R X86 64 PC32	myvar-0x00000000000000004	4e:	8b 05 00 00 00 00	mov	0x0(%rip),%eax	# 54 <main+0x41></main+0x41>	
	00090 R X86 64 PC32	.rodata-0x0000000000000004	54:	be 1e 00 00 00	mov	\$0x1e,%esi		
			59:	89 c7	mov	%eax,%edi		
	0009d R_X86_64_PLT32	sprintf-0x0000000000000004	5b:	e8 00 00 00 00	callq	60 <main+0x4d></main+0x4d>		
0000000000000	000c0 R X86 64 PLT32	write-0x000000000000004	60:	89 85 e8 fd ff ff	mov .	%eax,-0x218(%rbp)		
0000000000000	000d9 R X86 64 PLT32	stack chk fail-0x00000000000000004	66:	8b 15 00 00 00 00	mov	0x0(%rip),%edx	# 6c <main+0x59></main+0x59>	
			6c:	8b 85 e8 fd ff ff	mov	-0x218(%rbp),%eax		
			72:	01 d0	add	%edx,%eax		

- 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>

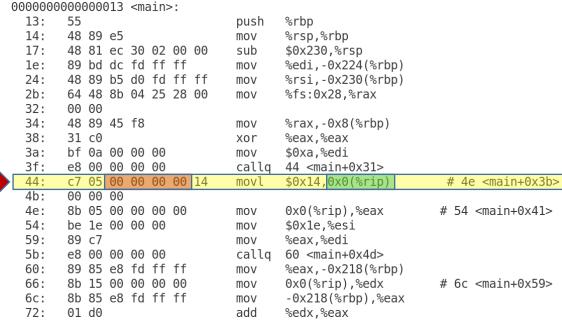
<now to compute the address>

00000000000000046 R\_X86\_64\_PC32

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

myvar-0x0000000000000000

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



- 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>

**Exact address** to be modified (marked in red)

00000000000000046 R X86 64 PC32 It indicates it is a PC-relative offset (%rip; that is, relative instruction pointer). Thus, "myvar" address is relative to the instruction pointer

myvar-0x00000000000000008

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

```
000000000000000000001
                       df *ABS*
                                  00000000000000000
                                                                 myheader.c
                                                                 crtstuff.c
000000000000000000001
                       df *ABS*
                                  00000000000000000
                        0 .eh frame
                                                                            FRAME END
00000000000000a2c l
                                          00000000000000000
00000000000000000 l
                       df *ABS*
                                 000000000000000000
00000000000200db0 l
                           .init array
                                          00000000000000000
                                                                            init array end
                        0 .dynamic
                                                                           DYNAMIC
0000000000200db8 l
                                          00000000000000000
0000000000200da8 l
                                          00000000000000000
                                                                            init array start
                          .init array
                           .eh frame hdr
                                                                            GNU EH FRAME HDR
0000000000000898 l
                                          00000000000000000
                                                                  GLOBAL OFFSET TABLE
0000000000200fa8 l
                                  00000000000000000
                        0 .got
                                                                    libc csu fini
0000000000000870 q
                        F .text
                                 00000000000000000
                                                                  ITM deregisterTMCloneTable
                          *UND*
00000000000000000
                                  00000000000000000
                                 00000000000000000
0000000000201000
                           .data
                                                                  data start
                        F *UND*
                                                                 write@@GLIBC 2.2.5
00000000000000000
                                  00000000000000000
                                                                 mifunc
00000000000006fa q
                                  00000000000000013
                        F .text
0000000000201018 q
                           .data
                                  00000000000000000
                                                                  edata
                                                                   fini
0000000000000874 g
                        F .fini
                                 00000000000000000
0000000000201010 q
                          .data
                                  00000000000000004
                                                                  alobal
00000000000000000
                        F *UND*
                                  00000000000000000
                                                                    stack chk fail@@GLIBC 2.4
00000000000007d9 q
                        F .text
                                 0000000000000001a
                                                                  suma
00000000000201014 q
                        0 .data
                                  000000000000000004
                                                                 myvar
000000000000000000
                        F *UND*
                                  00000000000000000
                                                                    libc start main@@GLIBC 2.2.5
0000000000201000 q
                           .data
                                 00000000000000000
                                                                    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>:
        55
 70d:
                                 push
                                        %rbp
        48 89 e5
                                        %rsp,%rbp
 70e:
                                 mov
711:
        48 81 ec 30 02 00 00
                                        $0x230,%rsp
                                 sub
718:
        89 bd dc fd ff ff
                                        %edi,-0x224(%rbp)
                                 mov
        48 89 b5 d0 fd ff ff
 71e:
                                        %rsi,-0x230(%rbp)
                                 mov
725:
        64 48 8b 04 25 28 00
                                        %fs:0x28,%rax
                                 mov
72c:
        00 00
        48 89 45 f8
72e:
                                        %rax,-0x8(%rbp)
                                 mov
        31 c0
732:
                                        %eax,%eax
                                 xor
734:
        bf 0a 00 00 00
                                        $0xa,%edi
                                 mov
        e8 bc ff ff ff
                                        6fa <mifunc>
739:
                                 callq
        c7 05 cc 08 20 00 14
73e:
                                        $0x14,0x2008cc(%rip)
                                                                     # 201014 <myvar>
                                 movl
745:
        00 00 00
748:
        8b 05 c2 08 20 00
                                                                    # 201010 <qlobal>
                                        0x2008c2(%rip),%eax
                                 mov
74e:
        be 1e 00 00 00
                                        $0x1e,%esi
                                 mov
753:
        89 c7
                                        %eax,%edi
                                 mov
755:
        e8 7f 00 00 00
                                 callq
                                        7d9 <suma>
        89 85 e8 fd ff ff
75a:
                                        %eax,-0x218(%rbp)
                                 mov
 760:
        8b 15 ae 08 20 00
                                        0x2008ae(%rip),%edx
                                                                    # 201014 <myvar>
                                 mov
        8b 85 e8 fd ff ff
766:
                                        -0x218(%rbp),%eax
                                 mov
        01 d0
                                 add
                                        %edx,%eax
 76c:
76e:
        89 05 a0 08 20 00
                                        %eax,0x2008a0(%rip)
                                                                    # 201014 <myvar>
                                 mov
774:
        8b 0d 9a 08 20 00
                                        0x20089a(%rip),%ecx
                                                                    # 201014 <myvar>
                                 mov
77a:
        8b 95 e8 fd ff ff
                                        -0x218(%rbp),%edx
                                 mov
```

```
"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