# A programmer's view: from source code to bit-stream and finally, the executing result; examples using GNU Utils under Linux x86-64

# A programmer's view: how does my computer work? (by Yingtao WANG)

```c
#include <stdio.h>

int main()
{
    printf("Hello World!\n");

    return 0;
}
```

$gcc -E HelloWorld.c

pre-processing

```
# 1 "HelloWorld.c"
# 1 "<built-in>"
# 1 "<command-line>"
# 31 "<command-line>"
# 1 "/usr/include/stdc-predef.h" 1 3 4
# 32 "<command-line>" 2
# 1 "HelloWorld.c"
# 1 "/usr/include/stdio.h" 1 3 4
# 27 "/usr/include/stdio.h" 3 4
# 1 "/usr/include/x86_64-linux-gnu/bits
```

$gcc -S HelloWorld.c

parsing

Lexical Analyzer (front-end)

Token stream (& symbol-table)
Linux tool: flex / lex (obs.)

Syntax Analyzer

Syntax tree
Linux tool: bison / yacc (obs.)

Semantic Analyzer

Syntax tree with type checking

Intermediate Code Generator (front-end)

Intermediate Code (for abstract machine):
- Machine like but independent
- Be easy to produce
- Be easy to translate into the target machine

Code Optimizer & Generator (back-end)

generating

```
        .file   "HelloWorld.c"
        .text
        .section    .rodata
.LC0:
        .string "Hello World!"
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        pushq   %rbp
```

$as HelloWorld.s && gcc HelloWorld.o -o HelloWorld

assembly

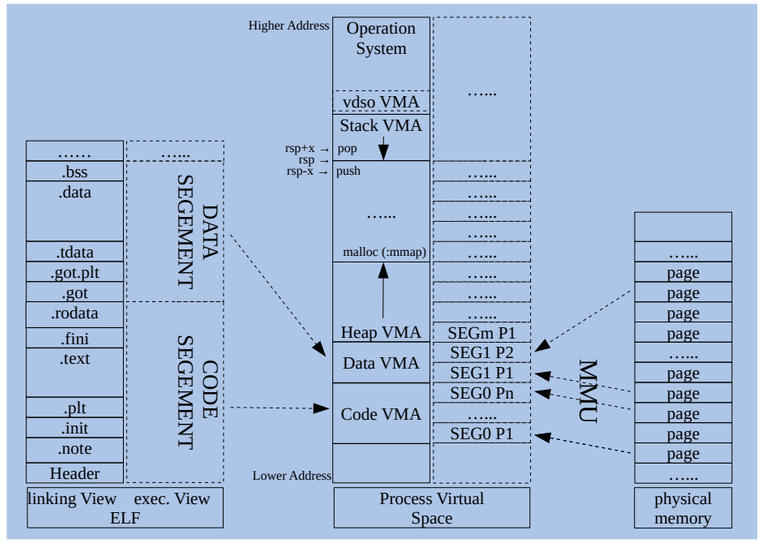
```
Programmer View$objdump -hw  HelloWorld.o

HelloWorld.o:     file format elf64-x86-64

Sections:
Idx Name          Size      VMA               LMA               File off  Algn  Flags
  0 .text         00000017  0000000000000000  0000000000000000  00000040  2**0  CONTENTS
  1 .data         00000000  0000000000000000  0000000000000000  00000057  2**0  CONTENTS
  2 .bss          00000000  0000000000000000  0000000000000000  00000057  2**0  ALLOC
  3 .rodata       00000004  0000000000000000  0000000000000000  00000057  2**0  CONTENTS
  4 .comment      0000002c  0000000000000000  0000000000000000  00000064  2**0  CONTENTS
  5 .note.GNU-stack 00000000  0000000000000000  0000000000000000  00000090  2**0  CONTENTS
  6 .eh_frame     00000038  0000000000000000  0000000000000000  00000090  2**3  CONTENTS
Programmer View$
```

## Linking Process

Two-pass linking (static):
- Address and storage (virtual memory space) allocation
- Symbol (variable / function) binding/resolution and relocation

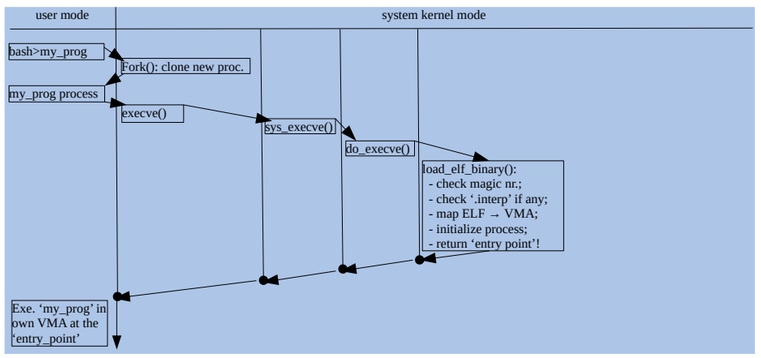
Additional step (dynamic):
- symbol resolution & relocation using Dynamic Linker* ('.interp')
  → loading time relocation

**'Entry Point' of the DL shall be impl. as 'bootstrap code'*




## Loading Process

Three steps loading (process creation):
- Create virtual address space distribution using page directory
- Build mapping ELF ↔ Process Virtual Space using ELF header
- Put the address of 'Entry Point' to the certain register, then start




## Useful GCC & GNU binutils & coreutils

**>objdump -h (sec.-header) | -d (disassemble) | -m intel | -g (debugging) | -w (wide) | -s (check '.interp')**
**>readelf -h (file-header) | -e (headers) | -s (symbols) | -d (dynm.) | -S (sec.) | -l (seg.) | -W (wide) | -a (all)**
**>strip no-option (remove debug sec.) | -s (remove all symbols) | g/S/d (remove debugging symbols)**

**>gcc -static (static-linking) | -ldl (dynamic-loading) | -fno-builtin (no builtin opt.) | -nostdlib | -nostartfiles**
**>ld –verbose (show default ld-script) | -T myscript (customer script)**

**>ldd myapp.elf (check dependences between shared libs)**

**>LDDEBUG=files myapp.elf (activates dynamic-linker debuging)**

**>ar -t static.a (display content of a static library for example libc.a) | -x static.a (decompress a static lib)**

```
Programmer View >objdump -d -M intel HelloWorld.out

HelloWorld.out:     file format elf64-x86-64


Disassembly of section .init:

00000000000004e8 <_init>:
 4e8:   48 83 ec 08             sub    rsp,0x8
 4ec:   48 8b 05 f5 0a 20 00    mov    rax,QWORD PTR [rip+0x200af5]
 4f3:   48 85 c0                test   rax,rax
 4f6:   74 02                   je     4fa <_init+0x12>
 4f8:   ff d0                   call   rax
 4fa:   48 83 c4 08             add    rsp,0x8
 4fe:   c3                      ret

Disassembly of section .plt:

0000000000000500 <.plt>:
 500:   ff 35 ba 0a 20 00       push   QWORD PTR [rip+0x200aba]
 506:   ff 25 bc 0a 20 00       jmp    QWORD PTR [rip+0x200abc]
 50c:   0f 1f 40 00             nop    DWORD PTR [rax+0x0]

0000000000000510 <puts@plt>:
 510:   ff 25 ba 0a 20 00       jmp    QWORD PTR [rip+0x200aba]
0000000000000520 <__cxa_finalize@plt>:
 520:   ff 25 d2 0a 20 00       jmp    QWORD PTR [rip+0x200ad2]
 526:   66 90                   xchg   ax,ax

Disassembly of section .text:

0000000000000530 <_start>:
 530:   31 ed                   xor    ebp,ebp
 532:   49 89 d1                mov    r9,rdx
 535:   5e                      pop    rsi
 536:   48 89 e2                mov    rdx,rsp
 539:   48 83 e4 f0             and    rsp,0xfffffffffffffff0
 53d:   50                      push   rax
 53e:   54                      push   rsp
 53f:   4c 8d 05 8a 01 00 00    lea    r8,[rip+0x18a]        # 6d0 <
 546:   48 8d 0d 13 01 00 00    lea    rcx,[rip+0x113]       # 660
 54d:   48 8d 3d e6 00 00 00    lea    rdi,[rip+0xe6]        # 63a <
 554:   ff 15 86 0a 20 00       call   QWORD PTR [rip+0x200a86]
 55a:   f4                      hlt
 55b:   0f 1f 44 00 00          nop    DWORD PTR [rax+rax*1+0x0]

0000000000000560 <deregister_tm_clones>:
 560:   f3 c3                   repz ret
 562:   66 0f 1f 44 00 00       nop    WORD PTR [rax+rax*1+0x0]

0000000000000630 <frame_dummy>:
 630:   55                      push   rbp
 631:   48 89 e5                mov    rbp,rsp
 634:   5d                      pop    rbp
 635:   e9 66 ff ff ff          jmp    5a0 <register_tm_clones>

000000000000063a <main>:
 63a:   55                      push   rbp
 63b:   48 89 e5                mov    rbp,rsp
 63e:   48 8d 3d 9f 00 00 00    lea    rdi,[rip+0x9f]        # 6e4 <
 645:   e8 c6 fe ff ff          call   510 <puts@plt>
 64a:   b8 00 00 00 00          mov    eax,0x0
 64f:   5d                      pop    rbp
 650:   c3                      ret
 651:   66 2e 0f 1f 84 00 00    nop    WORD PTR cs:[rax+rax*1+0x0]
 658:   00 00 00
 65b:   0f 1f 44 00 00          nop    DWORD PTR [rax+rax*1+0x0]

0000000000000660 <__libc_csu_init>:
 660:   41 57                   push   r15
 662:   41 56                   push   r14
 664:   49 89 d7                mov    r15,rdx
 667:   41 55                   push   r13
 669:   41 54                   push   r12
 66b:   4c 8d 25 8e 01 00 00    lea    r12,[rip+0x18e]        # 800 <
 672:   55                      push   rbp
 673:   48 8d 2d 8e 01 00 00    lea    rbp,[rip+0x18e]        # 808 <
 67a:   53                      push   rbx
 67b:   41 89 fd                mov    r13d,edi
 67e:   49 89 f6                mov    r14,rsi
 681:   4c 29 e5                sub    rbp,r12
 684:   48 83 ec 08             sub    rsp,0x8
 688:   48 c1 fd 03             sar    rbp,0x3
 68c:   e8 67 fe ff ff          call   4f8 <_init>
 691:   48 85 ed                test   rbp,rbp
 694:   74 20                   je     6b6 <__libc_csu_init+0x56>
 696:   31 db                   xor    ebx,ebx
 698:   0f 1f 84 00 00 00 00    nop    DWORD PTR [rax+rax*1+0x0]
 69f:   00
 6a0:   4c 89 fa                mov    rdx,r15
 6a3:   4c 89 f6                mov    rsi,r14
 6a6:   44 89 ef                mov    edi,r13d
 6a9:   41 ff 14 dc             call   QWORD PTR [r12+rbx*8]
 6ad:   48 83 c3 01             add    rbx,0x1
 6b1:   48 39 dd                cmp    rbp,rbx
 6b4:   75 ea                   jne    6a0 <__libc_csu_init+0x40>
 6b6:   48 83 c4 08             add    rsp,0x8
 6ba:   5b                      pop    rbx
 6bb:   5d                      pop    rbp
 6bc:   41 5c                   pop    r12
 6be:   41 5d                   pop    r13
 6c0:   41 5e                   pop    r14
 6c2:   41 5f                   pop    r15
 6c4:   c3                      ret

00000000000006c5 <_dl_relocate_static_pie>:
 6c5:   f3 c3                   repz ret

Disassembly of section .fini:

00000000000006d4 <_fini>:
 6d4:   48 83 ec 08             sub    rsp,0x8
 6d8:   48 83 c4 08             add    rsp,0x8
 6dc:   c3                      ret
```

```
Programmer View >objdump -d -M intel /usr/lib/x86_64-linux-gnu/crti.o

/usr/lib/x86_64-linux-gnu/crti.o:     file format elf64-x86-64


Disassembly of section .init:

0000000000000000 <_init>:
   0:   48 83 ec 08             sub    rsp,0x8
   4:   48 8b 05 00 00 00 00    mov    rax,QWORD PTR [rip+0x0]
   b:   48 85 c0                test   rax,rax
   e:   74 02                   je     12 <_init+0x12>
  10:   ff d0                   call   rax

Disassembly of section .fini:

0000000000000000 <_fini>:
   0:   48 83 ec 08             sub    rsp,0x8
Programmer View >objdump -d -M intel /usr/lib/x86_64-linux-gnu/crtn.o

/usr/lib/x86_64-linux-gnu/crtn.o:     file format elf64-x86-64


Disassembly of section .init:

0000000000000000 <.init>:
   0:   48 83 c4 08             add    rsp,0x8
   4:   c3                      ret

Disassembly of section .fini:

0000000000000000 <.fini>:
   0:   48 83 c4 08             add    rsp,0x8
   4:   c3                      ret
```

```
Programmer View >objdump -d -M intel /usr/lib/x86_64-linux-gnu/crt1.o

/usr/lib/x86_64-linux-gnu/crt1.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <_start>:
   0:   31 ed                   xor    ebp,ebp
   2:   49 89 d1                mov    r9,rdx
   5:   5e                      pop    rsi
   6:   48 89 e2                mov    rdx,rsp
   9:   48 83 e4 f0             and    rsp,0xfffffffffffffff0
   d:   50                      push   rax
   e:   54                      push   rsp
   f:   4c 8b 05 00 00 00 00    mov    r8,QWORD PTR [rip+0x0]
  16:   48 8b 0d 00 00 00 00    mov    rcx,QWORD PTR [rip+0x0]
  1d:   48 8b 3d 00 00 00 00    mov    rdi,QWORD PTR [rip+0x0]
  24:   ff 15 00 00 00 00       call   QWORD PTR [rip+0x0]
  2a:   f4                      hlt
  2b:   0f 1f 44 00 00          nop    DWORD PTR [rax+rax*1+0x0]
```

```
Programmer View >cat ./TinyHelloWorld.c
char* str = "Hello World!\n";

void print()
{
        asm( "mov rax, 1    \n\t"
             "mov rdi, rax  \n\t"
             "mov rsi, str  \n\t"
             "mov rdx, 13   \n\t"
             "syscall" );
}

void exit()
{
        asm( "mov rax, 60   \n\t"
             "mov rdi, 0    \n\t"
             "syscall" );
}

void nomain()
{
        print();
        exit();
}
Programmer View >gcc TinyHelloWorld.c -masm=intel -e nomain
```

```
gcc parms:
-masm=intel   // intel asm syntax style; default is AT&T
-nostdlib     // don't use std. libs i.e. CRT
-nostartfiles // don't use system 'startup' code crt*.o
-static       // consider static linking
-e nomain     // defined customer's entry-point 'nomain'

entry-point prio. high --> low:
. ld command arg. '-e'
. ENTRY symbol in a ld-script
. default '_start'
. start address of the 'text' section
. address ZERO
```

```
Programmer View >objdump -d -M intel TinyHelloWorld

TinyHelloWorld:     file format elf64-x86-64


Disassembly of section .text:

0000000000400144 <print>:
  400144:       55                      push   rbp
  400145:       48 89 e5                mov    rbp,rsp
  400148:       48 c7 c0 01 00 00 00    mov    rax,0x1
  40014f:       48 89 c7                mov    rdi,rax
  400152:       48 8b 34 25 00 10 60    mov    rsi,QWORD PTR ds:0x601000
  400159:       00
  40015a:       48 c7 c2 0d 00 00 00    mov    rdx,0xd
  400161:       0f 05                   syscall
  400163:       90                      nop
  400164:       5d                      pop    rbp
  400165:       c3                      ret

0000000000400166 <exit>:
  400166:       55                      push   rbp
  400167:       48 89 e5                mov    rbp,rsp
  40016a:       48 c7 c0 3c 00 00 00    mov    rax,0x3c
  400171:       48 c7 c7 00 00 00 00    mov    rdi,0x0
  400178:       0f 05                   syscall
  40017a:       90                      nop
  40017b:       5d                      pop    rbp
  40017c:       c3                      ret

000000000040017d <nomain>:
  40017d:       55                      push   rbp
  40017e:       48 89 e5                mov    rbp,rsp
  400181:       b8 00 00 00 00          mov    eax,0x0
  400186:       e8 b9 ff ff ff          call   400144 <print>
  40018b:       b8 00 00 00 00          mov    eax,0x0
  400190:       e8 d1 ff ff ff          call   400166 <exit>
  400195:       90                      nop
  400196:       5d                      pop    rbp
  400197:       c3                      ret
```

## Execution Process

Begins at the default entry-point '_start'*:
- system initialization i.e. '_init'*
- app. 'main' start
- code of app. and of static/dynamic libs like CRT (C Run-Time library: libc.a / libc.so)
- app. 'main' end
- system deinitialization i.e. '_fini'*

* '_start' is normally defined in OS file 'crt1.o';
* the header of the '_init' & '_fini' are def. in 'crti.o'
* the end of the '_init' & '_fini' are def. in 'crtn.o'

Structure of '_init' & '_fini':
- header
-customer code if any exists // ((__attribute((__section(".init"))))
- end

Using customer 'init/fini' from GCC CL:
-Wl -init my_init -fini my_fini

Modern mechanism (replacement of 'init/fini'):
__attribute__((constructor)) / ((destructor))

For more infos, pls. Refer to link.

## Example without default 'startup'

You don' really need the 'startup' and even CRT for your app. --- pls. refer to the 'TinyHelloWorld' on the left.

Learn more about Binary File Descriptor library (BFD)

Source code & Compilation → linking & loading → execution