

ELFy, obiekty i skrypty linkera

Paweł Kraszewski <root@linuxpedia.pl>

Jak zrobić najmniejszą aplikację*?

```
#include <stdio.h>
```

```
int main( int argc, char **argv )  
{  
    printf( "Hello world!\n" );  
    return 0;  
}
```

* Bez używania** asemblera

** Prawie

Jak zrobić najmniejszą aplikację?

```
> gcc -o main1 --static main1.c
```

```
> ls -al main1
```

```
-rwxr-xr-x 1 LwB LwB 915704 main1
```

Dafuq?


```
> readelf -e main1
```

```
ELF Header:
```

```
Magic:    7f 45 4c 46 02 01 01 03 00 00 00 00 00 00 00 00
Class:                                ELF64
Data:                                2's complement, little endian
Version:                              1 (current)
OS/ABI:                                UNIX - GNU
ABI Version:                           0
Type:                                EXEC (Executable file)
Machine:                               Advanced Micro Devices X86-64
Version:                               0x1
Entry point address:                   0x4009d0
Start of program headers:               64 (bytes into file)
Start of section headers:               913528 (bytes into file)
Flags:                                  0x0
Size of this header:                    64 (bytes)
Size of program headers:                 56 (bytes)
Number of program headers:                6
Size of section headers:                 64 (bytes)
Number of section headers:                34
Section header string table index: 33
```

```
...
```

Section Headers:

[Nr]	Name	Type	Address	Off	Size	ES	Flg	Lk	Inf	Al
[0]		NULL	0000000000000000	000000	000000	00		0	0	0
[1]	.note.ABI-tag	NOTE	00000000000400190	000190	000020	00	A	0	0	4
[2]	.note.gnu.build-id	NOTE	000000000004001b0	0001b0	000024	00	A	0	0	4
readelf: Warning: [3]: Link field (0) should index a symtab section.										
[3]	.rela.plt	RELA	000000000004001d8	0001d8	000108	18	AI	0	25	8
[4]	.init	PROGBITS	000000000004002e0	0002e0	000017	00	AX	0	0	4
[5]	.plt	PROGBITS	00000000000400300	000300	0000b0	00	AX	0	0	16
[6]	.text	PROGBITS	000000000004003b0	0003b0	09f967	00	AX	0	0	16
[7]	__libc_freeres_fn	PROGBITS	0000000000049fd20	09fd20	002529	00	AX	0	0	16
[8]	__libc_thread_freeres_fn	PROGBITS	000000000004a2250	0a2250	0000e1	00	AX	0	0	16
[9]	.fini	PROGBITS	000000000004a2334	0a2334	000009	00	AX	0	0	4
[10]	.rodata	PROGBITS	000000000004a2340	0a2340	01c9a4	00	A	0	0	32
[11]	__libc_subfreeres	PROGBITS	000000000004bece8	0bece8	000050	00	A	0	0	8
[12]	__libc_IO_vtables	PROGBITS	000000000004bed40	0bed40	0006a8	00	A	0	0	32
[13]	__libc_atexit	PROGBITS	000000000004bf3e8	0bf3e8	000008	00	A	0	0	8
[14]	.stapsdt.base	PROGBITS	000000000004bf3f0	0bf3f0	000001	00	A	0	0	1
[15]	__libc_thread_subfreeres	PROGBITS	000000000004bf3f8	0bf3f8	000008	00	A	0	0	8
[16]	.eh_frame	PROGBITS	000000000004bf400	0bf400	00aee4	00	A	0	0	8
[17]	.gcc_except_table	PROGBITS	000000000004ca2e4	0ca2e4	0000c1	00	A	0	0	1
[18]	.tdata	PROGBITS	000000000006caeb8	0caeb8	000020	00	WAT	0	0	8
[19]	.tbss	NOBITS	000000000006caed8	0caed8	000030	00	WAT	0	0	8
[20]	.init_array	INIT_ARRAY	000000000006caed8	0caed8	000010	08	WA	0	0	8
[21]	.fini_array	FINI_ARRAY	000000000006caee8	0caee8	000010	08	WA	0	0	8
[22]	.jcr	PROGBITS	000000000006caef8	0caef8	000008	00	WA	0	0	8
[23]	.data.rel.ro	PROGBITS	000000000006caf00	0caf00	0000e4	00	WA	0	0	32
[24]	.got	PROGBITS	000000000006cafe8	0cafe8	000008	08	WA	0	0	8
[25]	.got.plt	PROGBITS	000000000006cb000	0cb000	000070	08	WA	0	0	8
[26]	.data	PROGBITS	000000000006cb080	0cb080	001ad0	00	WA	0	0	32
[27]	.bss	NOBITS	000000000006ccb60	0ccb50	001898	00	WA	0	0	32
[28]	__libc_freeres_ptrs	NOBITS	000000000006ce3f8	0ccb50	000030	00	WA	0	0	8
[29]	.comment	PROGBITS	0000000000000000	0ccb50	00002d	01	MS	0	0	1
[30]	.note.stapsdt	NOTE	0000000000000000	0ccb80	000f04	00		0	0	4
[31]	.symtab	SYMTAB	0000000000000000	0cda88	00ae30	18		32	712	8
[32]	.strtab	STRTAB	0000000000000000	0d88b8	006645	00		0	0	1
[33]	.shstrtab	STRTAB	0000000000000000	0deefd	00017b	00		0	0	1

Key to Flags:

W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
L (link order), O (extra OS processing required), G (group), T (TLS),
C (compressed), x (unknown), o (OS specific), E (exclude),
l (large), p (processor specific)

Program Headers:

Type	Offset	VirtAddr	PhysAddr	FileSiz	MemSiz	Flg	Align
LOAD	0x000000	0x0000000000040000	0x0000000000040000	0x0ca3a5	0x0ca3a5	R E	0x200000
LOAD	0x0caeb8	0x000000000006caeb8	0x000000000006caeb8	0x001c98	0x003570	RW	0x200000
NOTE	0x000190	0x00000000000400190	0x00000000000400190	0x000044	0x000044	R	0x4
TLS	0x0caeb8	0x000000000006caeb8	0x000000000006caeb8	0x000020	0x000050	R	0x8
GNU_STACK	0x000000	0x0000000000000000	0x0000000000000000	0x000000	0x000000	RW	0x10
GNU_RELRO	0x0caeb8	0x000000000006caeb8	0x000000000006caeb8	0x000148	0x000148	R	0x1

Section to Segment mapping:

Segment Sections ...

00 .note.ABI-tag .note.gnu.build-id .rela.plt .init .plt .text __libc_freeres_fn
__libc_thread_freeres_fn .fini .rodata __libc_subfreeres __libc_IO_vtables __libc_atexit
.stapsdt.base __libc_thread_subfreeres .eh_frame .gcc_except_table
01 .tdata .init_array .fini_array .jcr .data.rel.ro .got .got.plt .data .bss
__libc_freeres_ptrs
02 .note.ABI-tag .note.gnu.build-id
03 .tdata .tbss
04
05 .tdata .init_array .fini_array .jcr .data.rel.ro .got

I co teras?

Sprawdzimy plik obiektowy

```
> gcc -c main1.c
> objdump -s main1.o
```

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

Contents of section .text:

```
0000 554889e5 4883ec10 897dfc48 8975f048  UH..H....}.H.u.H
0010 8d3d0000 0000e800 000000b8 00000000  .=.....
0020 c9c3                                     ..
```

Contents of section .rodata:

```
0000 48656c6c 6f20776f 726c6421 00          Hello world!.
```

Contents of section .comment:

```
0000 00474343 3a202855 62756e74 7520362e  .GCC: (Ubuntu 6.
0010 332e302d 31327562 756e7475 32292036  3.0-12ubuntu2) 6
0020 2e332e30 20323031 37303430 3600      .3.0 20170406.
```

Contents of section .eh_frame:

```
0000 14000000 00000000 017a5200 01781001  .....zR..x..
0010 1b0c0708 90010000 1c000000 1c000000  .....
0020 00000000 22000000 00410e10 8602430d  ....".....A....C.
0030 065d0c07 08000000  .].....
```


Sprawdzimy plik obiektowy

```
> objdump -d -Mintel main1.o
```

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

```
Disassembly of section .text:
```

```
0000000000000000 <main>:
   0: 55                push    rbp
   1: 48 89 e5          mov     rbp, rsp
   4: 48 83 ec 10       sub     rsp, 0x10
   8: 89 7d fc          mov     DWORD PTR [rbp-0x4], edi
  b: 48 89 75 f0       mov     QWORD PTR [rbp-0x10], rsi
  f: 48 8d 3d 00 00 00 00 lea     rdi, [rip+0x0]    # 16 <main+0x16>
16: e8 00 00 00 00    call   1b <main+0x1b>
1b: b8 00 00 00 00    mov     eax, 0x0
20: c9                leave
21: c3                ret
```

Sprawdzimy plik obiektowy

```
> objdump -tr main1.o
```

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

SYMBOL TABLE:

```
0000000000000000 l      df *ABS* 0000000000000000 main.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      F  .text 0000000000000022 main
0000000000000000      *UND* 0000000000000000 _GLOBAL_OFFSET_TABLE_
0000000000000000      *UND* 0000000000000000 puts
```

RELOCATION RECORDS FOR [.text]:

OFFSET	TYPE	VALUE
0000000000000012	R_X86_64_PC32	.rodata-0x0000000000000004
0000000000000017	R_X86_64_PLT32	puts-0x0000000000000004

RELOCATION RECORDS FOR [.eh_frame]:

OFFSET	TYPE	VALUE
0000000000000020	R_X86_64_PC32	.text

Biblioteka standardowa

- „puts” jest częścią biblioteki standardowej
- Biblioteka standardowa jest WIEEELKA
- puts to w skrócie kombinacja write i strlen.
- To może tak zastąpić?

Take #2

```
#include <unistd.h>

const char MSG[] = "Hello world!\n";

int main( int argc, char **argv )
{
    write(1,MSG,sizeof(MSG));
    return 0;
}
```

Take #2

```
> gcc -o main2 --static main2.c
```

```
> ls -al main2
```

```
-rwxr-xr-x 1 LwB LwB 915680 main2
```

Zagładamy

```
> gcc -c main2.c
> objdump -s main2.o
```

```
main2.o:      file format elf64-x86-64
```

```
Contents of section .text:
```

```
0000 554889e5 4883ec10 897dfc48 8975f0ba  UH..H....}.H.u..
0010 0e000000 488d3500 000000bf 01000000  ....H.5.....
0020 e8000000 00b80000 0000c9c3  .........
```

```
Contents of section .rodata:
```

```
0000 48656c6c 6f20776f 726c6421 0a00      Hello world!..
```

```
Contents of section .comment:
```

```
0000 00474343 3a202855 62756e74 7520362e  .GCC: (Ubuntu 6.
0010 332e302d 31327562 756e7475 32292036  3.0-12ubuntu2) 6
0020 2e332e30 20323031 37303430 3600      .3.0 20170406.
```

```
Contents of section .eh_frame:
```

```
0000 14000000 00000000 017a5200 01781001  ....zR..x..
0010 1b0c0708 90010000 1c000000 1c000000  ....
0020 00000000 2c000000 00410e10 8602430d  ....,....A....C.
0030 06670c07 08000000  ....g.....
```


Zagładamy

```
> objdump -d -Mintel main2.o
```

```
main2.o:      file format elf64-x86-64
```

Disassembly of section `.text`:

```
0000000000000000 <main>:
```

```
  0: 55                push    rbp
  1: 48 89 e5          mov     rbp, rsp
  4: 48 83 ec 10       sub     rsp, 0x10
  8: 89 7d fc          mov     DWORD PTR [rbp-0x4], edi
  b: 48 89 75 f0       mov     QWORD PTR [rbp-0x10], rsi
  f: ba 0e 00 00 00   mov     edx, 0xe
14: 48 8d 35 00 00 00 00 lea     rsi, [rip+0x0]      # 1b <main+0x1b>
1b: bf 01 00 00 00   mov     edi, 0x1
20: e8 00 00 00 00   call    25 <main+0x25>
25: b8 00 00 00 00   mov     eax, 0x0
2a: c9              leave   rcx
2b: c3              ret
```

Zagładamy

```
> objdump -tr main2.o
```

```
main2.o:      file format elf64-x86-64
```

SYMBOL TABLE:

```
0000000000000000 l      df *ABS* 0000000000000000 main2.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      0 .rodata 0000000000000000e MSG
0000000000000000 g      F .text 0000000000000002c main
0000000000000000      *UND* 0000000000000000 _GLOBAL_OFFSET_TABLE_
0000000000000000      *UND* 0000000000000000 write
```

RELOCATION RECORDS FOR [.text]:

OFFSET	TYPE	VALUE
0000000000000017	R_X86_64_PC32	MSG-0x0000000000000004
0000000000000021	R_X86_64_PLT32	write-0x0000000000000004

RELOCATION RECORDS FOR [.eh_frame]:

OFFSET	TYPE	VALUE
0000000000000020	R_X86_64_PC32	.text

Biblioteka standardowa #2

Co to jest funkcja `write`?

- Jest to tzw *cienki wrapper* wywołujący kernelowy syscall `sys_write`.

To może pozbyć się biblioteki standardowej i bezpośrednio wywołać `sys_write`? Jak wywołać syscalla w C?

- Za pomocą funkcji biblioteki standardowej `syscall(...)`

Oh, wait...

No dobra... a bez libc?

```
const char MSG[] = "Hello world!\n";
#define __NR_write 1

static int inline my_write(int fd, const void *buf, int size)
{
    int ret;
    asm volatile
    (
        "syscall"
        : "=a" (ret)
        : "0"(__NR_write), "D"(fd), "S"(buf), "d"(size)
        : "cc", "rcx", "r11", "memory"
    );
    return ret;
}

int main( int argc, char **argv )
{
    my_write(1,MSG,sizeof(MSG));
    return 0;
}
```

No dobra... a bez libc?

```
> gcc -o main3 -O2 --static main3.c  
> ls -al main3
```

```
-rwxr-xr-x 1 LwB LwB 915680 main3
```

Whyyyyyyyyyyyyyyy?

<main>:

Zaglądamy

```
> gcc -c -O2 main3.c
> objdump -s main3.o
```

```
main3.o:          file format elf64-x86-64
```

```
Contents of section .text.startup:
```

```
0000 b8010000 00ba0e00 0000488d 35000000 .....H.5 ...
0010 0089c70f 0531c0c3 .....1..
```

```
Contents of section .rodata:
```

```
0000 48656c6c 6f20776f 726c6421 0a00      Hello world!..
```

```
Contents of section .comment:
```

```
0000 00474343 3a202855 62756e74 7520362e  .GCC: (Ubuntu 6.
0010 332e302d 31327562 756e7475 32292036  3.0-12ubuntu2) 6
0020 2e332e30 20323031 37303430 3600      .3.0 20170406.
```

```
Contents of section .eh_frame:
```

```
0000 14000000 00000000 017a5200 01781001 .....zR.. x ..
0010 1b0c0708 90010000 14000000 1c000000 .....
0020 00000000 18000000 00000000 00000000 .....
```


Zagładamy

```
> objdump -dMintel main3.o
```

```
main3.o:          file format elf64-x86-64
```

```
Disassembly of section .text.startup:
```

```
0000000000000000 <main>:
```

0:	b8 01 00 00 00	mov	eax,0x1
5:	ba 0e 00 00 00	mov	edx,0xe
a:	48 8d 35 <u>00 00 00 00</u>	lea	rsi,[rip+0x0]
		# 11	<main+0x11>
11:	89 c7	mov	edi,edx
13:	0f 05	syscall	
15:	31 c0	xor	eax,edx
17:	c3	ret	

Zagładamy

```
> objdump -tr main3.o
```

```
main3.o:      file format elf64-x86-64
```

SYMBOL TABLE:

```
0000000000000000 l      df *ABS* 0000000000000000 main3.c
0000000000000000 l      d  .text 0000000000000000 .text
0000000000000000 l      d  .data 0000000000000000 .data
0000000000000000 l      d  .bss 0000000000000000 .bss
0000000000000000 l      d  .text.startup 0000000000000000 .text.startup
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      F  .text.startup 0000000000000018 main
0000000000000000 g      0  .rodata 000000000000000e MSG
```

RELOCATION RECORDS FOR [.text.startup]:

OFFSET	TYPE	VALUE
000000000000000d	R_X86_64_PC32	MSG-0x0000000000000004

RELOCATION RECORDS FOR [.eh_frame]:

OFFSET	TYPE	VALUE
0000000000000020	R_X86_64_PC32	.text.startup

Ok, ale...

- No dobrze, nasza aplikacja *nie wywołuje* biblioteki standardowej – to skąd ten rozmiar?
- Nasza aplikacja *jest wywoływana* przez bibliotekę standardową, a konkretnie przez jej część nazywaną *runtime*.
- Jak myślicie, skąd `main` ma argumenty `argc` i `argv`? I ciągle zapomniany argument `env`? (Tak, `main` ma 3 argumenty a nie 2...)
- To pozbyć się `runtime`'a?

No dobra... a bez libc?

```
const char MSG[] = "Hello world!\n";
#define __NR_write 1

static int inline my_write(int fd, const void *buf, int size)
{
    int ret;
    asm volatile
    (
        "syscall"
        : "=a" (ret)
        : "0"(__NR_write), "D"(fd), "S"(buf), "d"(size)
        : "cc", "rcx", "r11", "memory"
    );
    return ret;
}

void _start()
{
    my_write(1,MSG,sizeof(MSG));
}
```

Take #4

```
> gcc -nostdlib --static -O2 -o main4  
main4.c
```

```
> ls -al main4
```

```
-rwxr-xr-x 1 LwB LwB 1392 main4
```

Auć!

```
> ./main4
```

```
Hello world!
```

```
zsh: segmentation fault (core dumped)
```

*One does not simply
return from _start
function*


```
const char MSG[] = "Hello world!\n";
#define __NR_write 1
#define __NR_exit 60

static int inline my_write(int fd, const void *buf, int size)
{
    int ret;
    asm volatile
    (
        "syscall"
        : "=a" (ret)
        : "0"(__NR_write), "D"(fd), "S"(buf), "d"(size)
        : "cc", "rcx", "r11", "memory"
    );
    return ret;
}

static void __attribute__((noreturn)) my_exit(int rc)
{
    asm volatile ( "syscall" :: "a"(__NR_exit), "D"(rc) : );
    __builtin_unreachable();
}

void __attribute__((noreturn)) _start()
{
    my_write(1,MSG,sizeof(MSG));
    my_exit(0);
}
```

Auć!

```
> gcc -nostdlib --static -O2 -o main5  
main5.c
```

```
> ./main5
```

Hello world!

*Program z exit-em jest
mniej awanturujący się*

Aleeee...

```
> ls -al main5
-rwxr-xr-x 1 LwB LwB 1400 kwi 17 15:41 main5
> strip main5
> ls -al main5
-rwxr-xr-x 1 LwB LwB 928 kwi 17 15:48 main5
```

928 bajtów to ciągle dużo, jak na program, który prawie nic nie robi...

Pokaż ELFie co masz w środku...

```
> readelf -SlW main5
```

There are 7 section headers, starting at offset 0×1e0:

Section Headers:

[Nr]	Name	Type	Address	Off	Size	ES	Flg	Lk	Inf	Al
[0]		NULL	0000000000000000	000000	000000	00		0	0	0
[1]	.note.gnu.build-id	NOTE	000000000004000e8	0000e8	000024	00	A	0	0	4
[2]	.text	PROGBITS	00000000000400110	000110	00001f	00	AX	0	0	16
[3]	.rodata	PROGBITS	00000000000400130	000130	00000e	00	A	0	0	8
[4]	.eh_frame	PROGBITS	00000000000400140	000140	000030	00	A	0	0	8
[5]	.comment	PROGBITS	0000000000000000	000170	00002d	01	MS	0	0	1
[6]	.shstrtab	STRTAB	0000000000000000	00019d	00003f	00		0	0	1

Key to Flags:

W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
L (link order), O (extra OS processing required), G (group), T (TLS),
C (compressed), x (unknown), o (OS specific), E (exclude),
l (large), p (processor specific)

Elf file type is EXEC (Executable file)

Entry point 0×400110

There are 3 program headers, starting at offset 64

Program Headers:

Type	Offset	VirtAddr	PhysAddr	FileSiz	MemSiz	Flg	Align
LOAD	0×000000	0×0000000000040000	0×0000000000040000	0×000170	0×000170	R E	0×200000
NOTE	0×0000e8	0×000000000004000e8	0×000000000004000e8	0×000024	0×000024	R	0×4
GNU_STACK	0×000000	0×0000000000000000	0×0000000000000000	0×000000	0×000000	RW	0×10

Section to Segment mapping:

Segment Sections ...

00	.note.gnu.build-id	.text	.rodata	.eh_frame
01	.note.gnu.build-id			
02				

Tak naprawdę potrzebujemy tylko .text i .rodata ...

Skrypty linkera

```
> cat main5.ld
```

```
OUTPUT_FORMAT("elf64-x86-64", "elf64-x86-64", "elf64-x86-64")
OUTPUT_ARCH(i386:x86-64)

ENTRY(_start);

SECTIONS
{
    PROVIDE (__executable_start =
                SEGMENT_START("text-segment", 0x400000));
    . = SEGMENT_START("text-segment", 0x400000) + SIZEOF_HEADERS;
    .text : { *(.text*); *(.rodata*); }

    /DISCARD/ :
    {
        *(.comment*)
        *(.eh_frame*)
        *(.note.gnu.build-id*)
    }
}
```

Skrypty linkera

```
> ld -T main5.ld -s -o main5 main5.o
> ls -al main5
-rwxr-xr-x 1 LwB LwB 432  main5
> ./main5
Hello world!
```


Skrypt linkera

```
> readelf -WSl main5
```

There are 3 section headers, starting at offset 0xf0:

Section Headers:

[Nr]	Name	Type	Address	Off	Size	ES	Flg	Lk	Inf	Al
[0]		NULL	0000000000000000	000000	000000	00		0	0	0
[1]	.text	PROGBITS	000000000004000b0	0000b0	00002e	00	AX	0	0	16
[2]	.shstrtab	STRTAB	0000000000000000	0000de	000011	00		0	0	1

Key to Flags:

W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
L (link order), O (extra OS processing required), G (group), T (TLS),
C (compressed), x (unknown), o (OS specific), E (exclude),
l (large), p (processor specific)

Elf file type is EXEC (Executable file)

Entry point 0x4000b0

There are 2 program headers, starting at offset 64

Program Headers:

Type	Offset	VirtAddr	PhysAddr	FileSiz	MemSiz	Flg	Align
LOAD	0x000000	0x0000000000040000	0x0000000000040000	0x0000de	0x0000de	R E	0x200000
GNU_STACK	0x000000	0x0000000000000000	0x0000000000000000	0x000000	0x000000	RW	0x10

Section to Segment mapping:

Segment Sections ...

00 .text

01

Skrypty linkera

```
> objdump -dMintel -s main5
```

```
main5:      file format elf64-x86-64
```

```
Contents of section .text:
```

```
4000b0 b8010000 00ba0e00 0000488d 350f0000 .....H.5 ...
4000c0 0089c70f 0531fffb 3c000000 0f056690 .....1..<.....f.
4000d0 48656c6c 6f20776f 726c6421 0a00      Hello world!..
```

```
Disassembly of section .text:
```

```
00000000004000b0 <.text>:
```

```
4000b0: b8 01 00 00 00      mov     eax,0x1
4000b5: ba 0e 00 00 00      mov     edx,0xe
4000ba: 48 8d 35 0f 00 00 00 lea     rsi,[rip+0xf]          # 0x4000d0
4000c1: 89 c7              mov     edi,eax
4000c3: 0f 05              syscall
4000c5: 31 ff              xor     edi,edi
4000c7: b8 3c 00 00 00      mov     eax,0x3c
4000cc: 0f 05              syscall
4000ce: 66 90              xchg    ax,ax
4000d0: 48                rex.W
4000d1: 65 6c              gs ins  BYTE PTR es:[rdi],dx
4000d3: 6c                ins     BYTE PTR es:[rdi],dx
4000d4: 6f                outs    dx,DWORD PTR ds:[rsi]
4000d5: 20 77 6f           and     BYTE PTR [rdi+0x6f],dh
4000d8: 72 6c              jb      0x400146
4000da: 64 21 0a           and     DWORD PTR fs:[rdx],ecx
```


To wszystko

Udało się zejść z rozmiarem pliku wykonywalnego z **915704** bajtów do **432** bajtów z zachowaniem pełnej (acz niewielkiej) funkcjonalności.

Co straciliśmy (ale można dopisać własne wersje):

- Biblitekę standardową, a z nią dobroci typu `printf`, `atexit` czy `malloc/free`
- Możliwość pracy na `float/double`
- Przy wykorzystaniu `g++` - dużą część funkcjonalności C++ (wyjątki, konstruktory/destruktory, `new/delete`, STL)
- `pthread`