Executing main() in C/C++ – behind the scene

How to write a C program to print "Hello world" without main() function?
*At first, it seems impractical to execute a program without a **main()** function because the **main()** function is the entry point of any program.*

Let us first understand what happens under the hood while executing a C program in Linux system, how main() is called and how to execute a program without main().

Following setup is considered for the demonstration.

- Ubuntu 16.4 LTS operating system
- GCC 5.4.0 compiler
- objdump utility

From C/C++ programming perspective, the program entry point is main() function. From the perspective of program execution, however, it is not. Prior to the point when the execution flow reaches to the main(), calls to few other functions are made, which setup arguments, prepare environment variables for program execution etc.

The executable file created after compiling a C source code is a Executable and Linkable Format (ELF) file.

Every ELF file have a ELF header where there is a **e_entry** field which contains the program memory address from which the execution of executable will start. This memory address point to the **_start()** function.
After loading the program, loader looks for the **e_entry** field from the ELF file header. Executable and Linkable Format (ELF) is a common standard file format used in UNIX system for executable files, object code, shared libraries, and core dumps.

Let's see this using an example. I'm creating a **example.c** file to demonstrate this.

```
int main()
{
    return(0);
}
```

Now compiling this using following commands

```
gcc -o example example.c
```

Now an **example** executable is created, let us examine this using objdump utility

```
objdump -f example
```

This outputs following critical information of executable on my machine. Have a look at start address below, this is the address pointing to _start() function.

```
example:     file format elf64-x86-64
architecture: i386:x86-64, flags 0x00000112:
EXEC_P, HAS_SYMS, D_PAGED
start address 0x00000000004003e0
```

We can cross check this address by deassembling the executable, the output is long so I'm just pasting the output which shows where this address **0x00000000004003e0** is pointing

```
objdump --disassemble  example
```

Output :

```
00000000004003e0 <_start>:
4003e0:       31 ed                   xor    %ebp,%ebp
4003e2:       49 89 d1                mov    %rdx,%r9
4003e5:       5e                      pop    %rsi
4003e6:       48 89 e2                mov    %rsp,%rdx
4003e9:       48 83 e4 f0             and    $0xfffffffffffffff0
4003ed:       50                      push   %rax
4003ee:       54                      push   %rsp
4003ef:       49 c7 c0 60 05 40 00    mov    $0x400560,%r8
4003f6:       48 c7 c1 f0 04 40 00    mov    $0x4004f0,%rcx
4003fd:       48 c7 c7 d6 04 40 00    mov    $0x4004d6,%rdi
400404:       e8 b7 ff ff ff          callq  4003c0
400409:       f4                      hlt
40040a:       66 0f 1f 44 00 00       nopw   0x0(%rax,%rax,1)
```

The role of _start() function

The _start() function prepare the input arguments for another function **_libc_start_main()** which will be called next. This is prototype of **_libc_start_main()** function. Here we can see the arguments which were prepared by _start() function.

```
int __libc_start_main (
  int (*main) (int, char * *, char * *), /* address of main function*/
  int argc,                           /* number of command line args*/
  char ** ubp_av,                          /* command line arg array*/
  void (*init) (void),                   /* address of init function*/
  void (*fini) (void),                   /* address of fini function*/
  void (*rtld_fini) (void),
                         /* address of dynamic linker fini function */
  void (* stack_end)                     /* end of the stack address*/
);
```

The role of _libs_start_main() function is following –

- Preparing environment variables for program execution
- Calls **_init()** function which performs initialization before the main() function start.
- Register **_fini()** and **_rtld_fini()** functions to perform cleanup after program terminates

After all the prerequisite actions has been completed, _libc_start_main() calls the main() function.

Writing program without main()

Now we know how the call to the main() is made.To make it clear, main() is nothing but a agreed term for startup code. We can have any name for startup code it doesn't necessarily have to be “main”. As _start() function by default calls main(), we have to change it if we want to execute our custom startup code. We can override the _start() function to make it call our custom startup code not main(). Let's have an example, save it as **nomain.c** –

```
#include<stdio.h>
#include<stdlib.h>
void _start()
{
    int x = my_fun(); //calling custom main function
    exit(x);
}

int my_fun() // our custom main function
{
    printf("Hello world!\n");
    return 0;
}
```

Now we have to force compiler to not use it's own implementation of _start().In GCC we can do this using **-nostartfiles**

```
gcc -nostartfiles -o nomain nomain.c
```

Execute the executable nomain

```
./nomain
```