Graham Dungan
April 12, 2025
UIN: 332001764

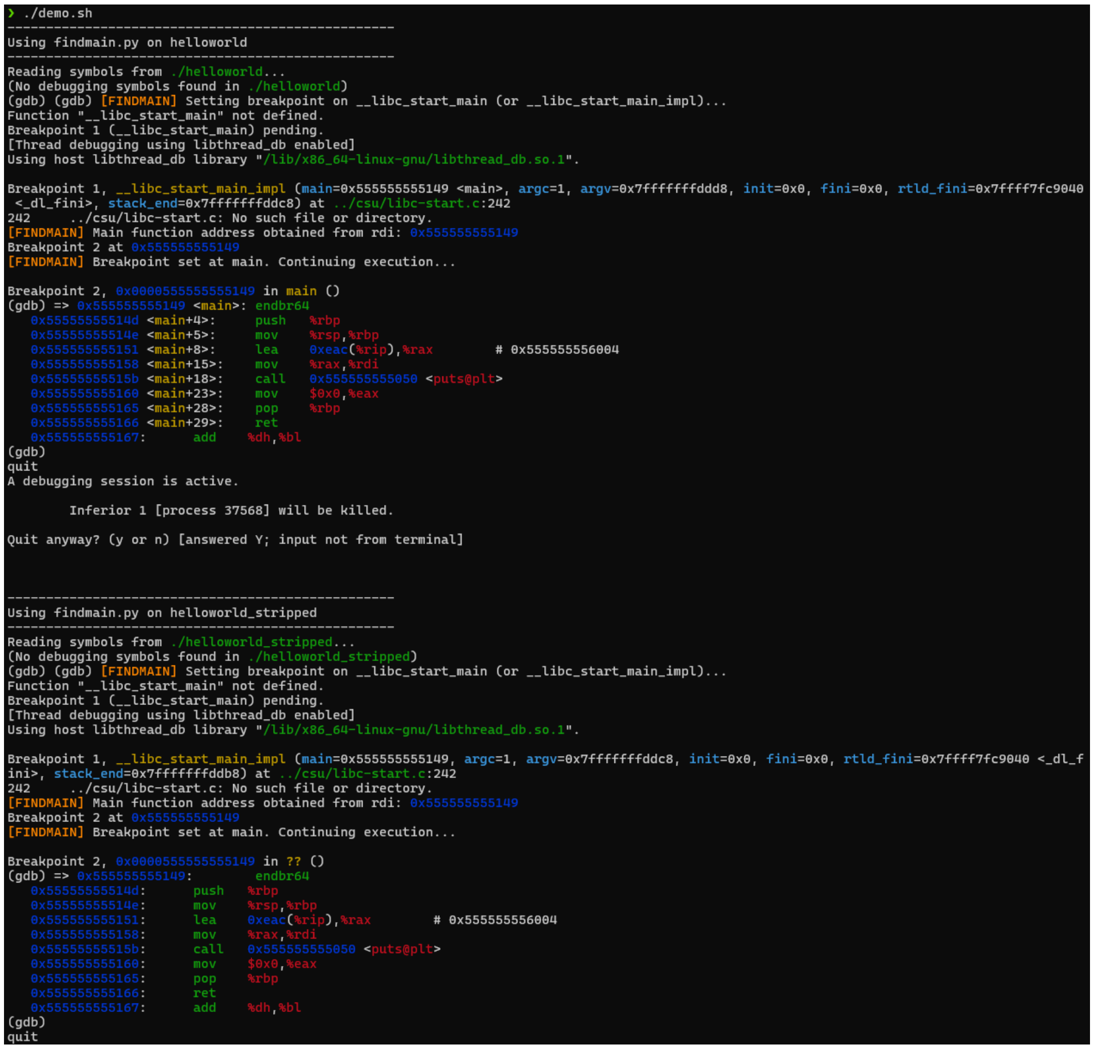
CSCE 413: Software Security
Class 34: GDB

Demonstration Quickstart

A demonstration script `demo.sh` has been included. This demo script runs both `helloworld` and `helloworld_stripped` in GDB. Once in GDB, it runs the `findmain.py` script to reveal that `main` is found within both scripts.
To run this demo,

1. Enter the `Class34` directory.
   `cd Class34`
2. *(Optional)* If the script does not have execution permission, add such permissions.
   `chmod +x ./demo.sh`
3. Run the `demo.sh` script.
   `./demo.sh`

What follows is a screenshot of the output of `demo.sh`.



```
❯ ./demo.sh
--------------------------------------------------
Using findmain.py on helloworld
--------------------------------------------------
Reading symbols from ./helloworld...
(No debugging symbols found in ./helloworld)
(gdb) (gdb) [FINDMAIN] Setting breakpoint on __libc_start_main (or __libc_start_main_impl)...
Function "__libc_start_main" not defined.
Breakpoint 1 (__libc_start_main) pending.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, __libc_start_main_impl (main=0x555555555149 <main>, argc=1, argv=0x7fffffffddd8, init=0x0, fini=0x0, rtld_fini=0x7ffff7fc9040
 <_dl_fini>, stack_end=0x7fffffffddc8) at ../csu/libc-start.c:242
242     ../csu/libc-start.c: No such file or directory.
[FINDMAIN] Main function address obtained from rdi: 0x555555555149
Breakpoint 2 at 0x555555555149
[FINDMAIN] Breakpoint set at main. Continuing execution...

Breakpoint 2, 0x0000555555555149 in main ()
(gdb) => 0x555555555149 <main>: endbr64
   0x55555555514d <main+4>:     push   %rbp
   0x55555555514e <main+5>:     mov    %rsp,%rbp
   0x555555555151 <main+8>:     lea    0xeac(%rip),%rax        # 0x555555556004
   0x555555555158 <main+15>:    mov    %rax,%rdi
   0x55555555515b <main+18>:    call   0x555555555050 <puts@plt>
   0x555555555160 <main+23>:    mov    $0x0,%eax
   0x555555555165 <main+28>:    pop    %rbp
   0x555555555166 <main+29>:    ret
   0x555555555167:      add    %dh,%bl
(gdb)
quit
A debugging session is active.

        Inferior 1 [process 37568] will be killed.

Quit anyway? (y or n) [answered Y; input not from terminal]



--------------------------------------------------
Using findmain.py on helloworld_stripped
--------------------------------------------------
Reading symbols from ./helloworld_stripped...
(No debugging symbols found in ./helloworld_stripped)
(gdb) (gdb) [FINDMAIN] Setting breakpoint on __libc_start_main (or __libc_start_main_impl)...
Function "__libc_start_main" not defined.
Breakpoint 1 (__libc_start_main) pending.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, __libc_start_main_impl (main=0x555555555149, argc=1, argv=0x7fffffffddc8, init=0x0, fini=0x0, rtld_fini=0x7ffff7fc9040 <_dl_f
ini>, stack_end=0x7fffffffddb8) at ../csu/libc-start.c:242
242     ../csu/libc-start.c: No such file or directory.
[FINDMAIN] Main function address obtained from rdi: 0x555555555149
Breakpoint 2 at 0x555555555149
[FINDMAIN] Breakpoint set at main. Continuing execution...

Breakpoint 2, 0x0000555555555149 in ?? ()
(gdb) => 0x555555555149:        endbr64
   0x55555555514d:      push   %rbp
   0x55555555514e:      mov    %rsp,%rbp
   0x555555555151:      lea    0xeac(%rip),%rax        # 0x555555556004
   0x555555555158:      mov    %rax,%rdi
   0x55555555515b:      call   0x555555555050 <puts@plt>
   0x555555555160:      mov    $0x0,%eax
   0x555555555165:      pop    %rbp
   0x555555555166:      ret
   0x555555555167:      add    %dh,%bl
(gdb)
quit
```

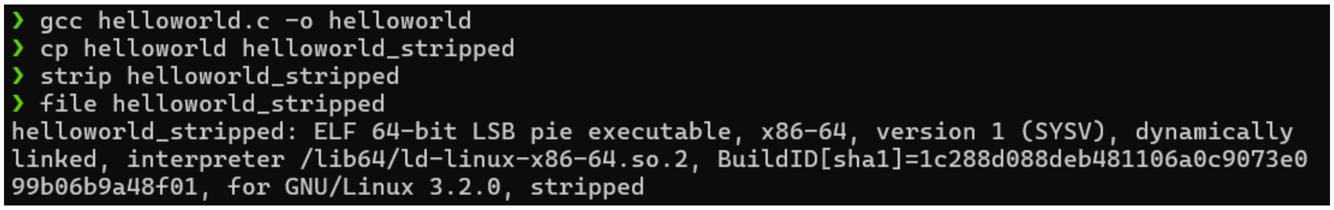
Creating a Script

For this assignment, I created a simple script called `helloworld.c`.

```c
#include <stdio.h>

int main() {
    printf("Hello World!\n");
    return 0;
}
```

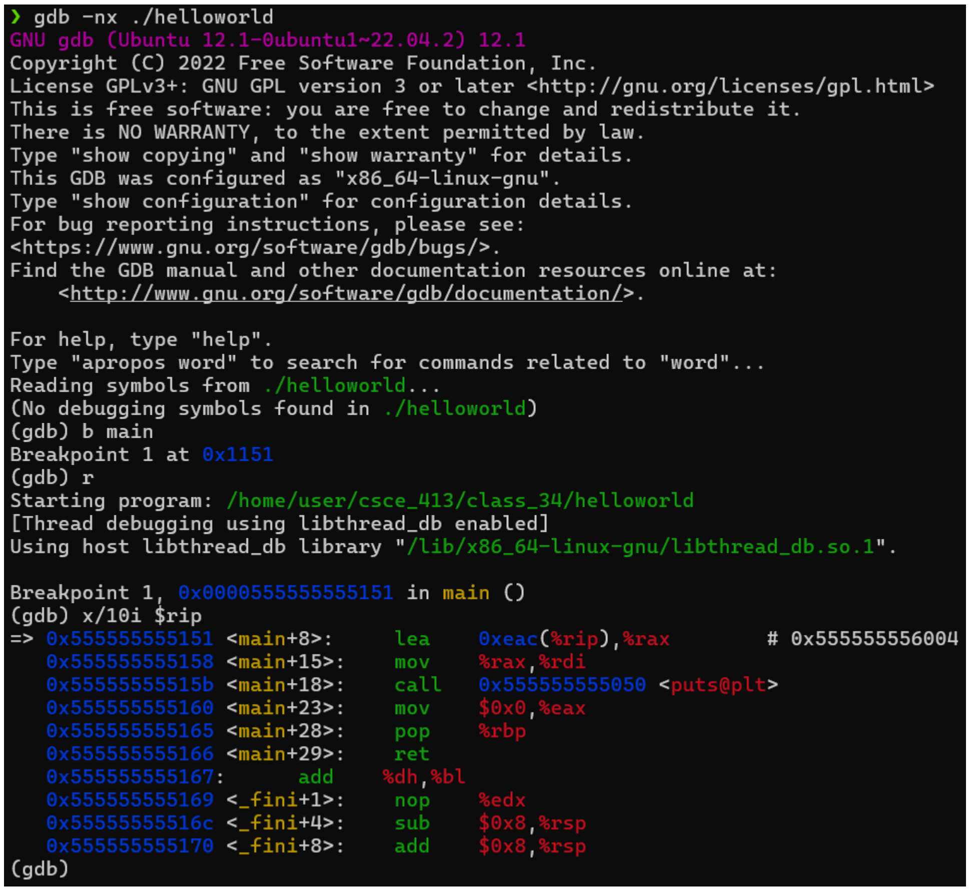
This script simply prints "Hello World!" and terminates. We will compile this script to the binary `helloworld`. We will then create a copy of the script and strip its symbols, called `helloworld_stripped`.



```
❯ gcc helloworld.c -o helloworld
❯ cp helloworld helloworld_stripped
❯ strip helloworld_stripped
❯ file helloworld_stripped
helloworld_stripped: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically
linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=1c288d088deb481106a0c9073e0
99b06b9a48f01, for GNU/Linux 3.2.0, stripped
```

Finding Main Manually

For reference, we will use GDB to manually find the address of `main`. We can do this by simply placing a breakpoint at main and running the program.



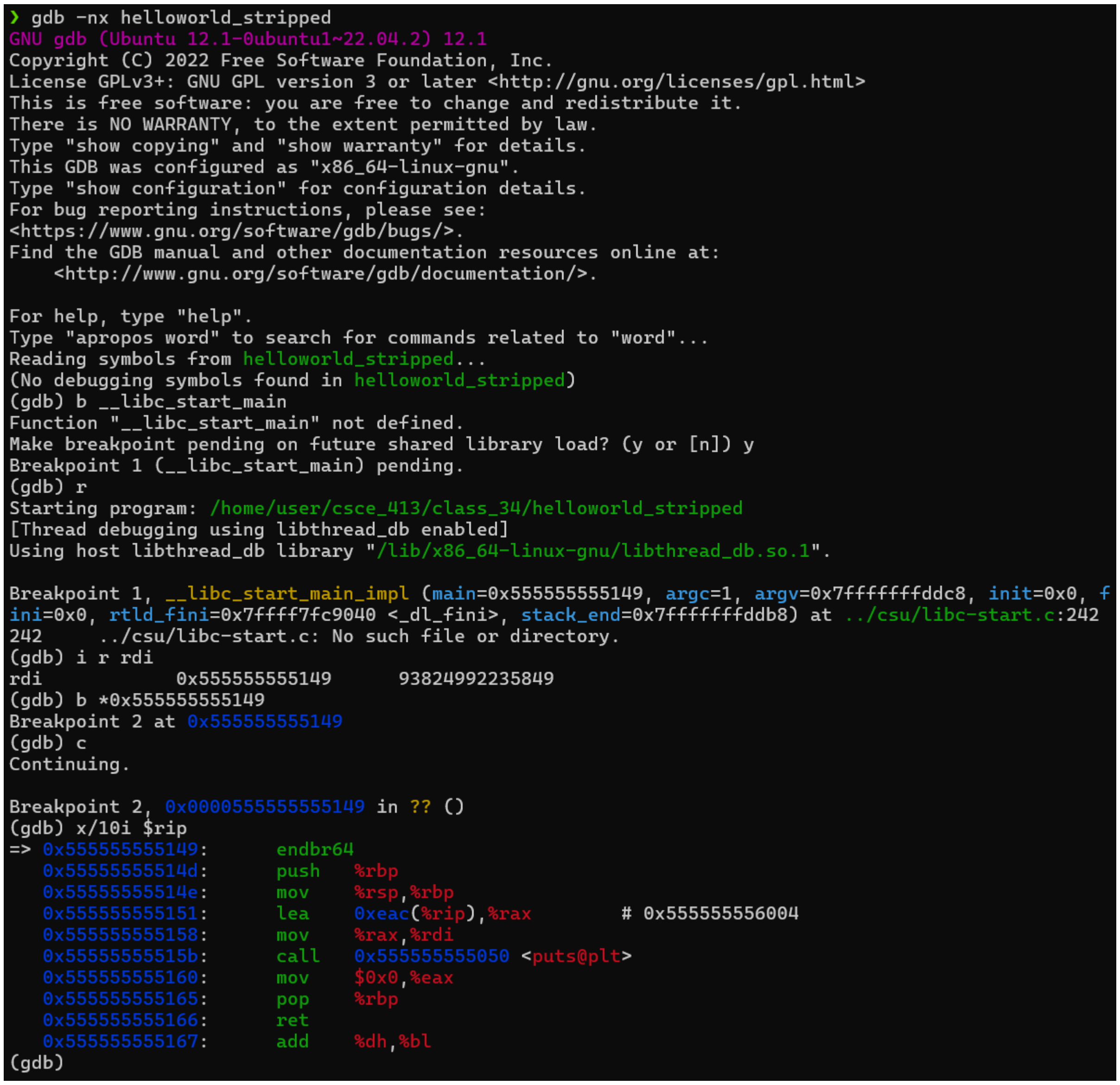
```
❯ gdb -nx ./helloworld
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04.2) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./helloworld...
(No debugging symbols found in ./helloworld)
(gdb) b main
Breakpoint 1 at 0x1151
(gdb) r
Starting program: /home/user/csce_413/class_34/helloworld
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, 0x0000555555555151 in main ()
(gdb) x/10i $rip
=> 0x555555555151 <main+8>:     lea    0xeac(%rip),%rax        # 0x555555556004
   0x555555555158 <main+15>:    mov    %rax,%rdi
   0x55555555515b <main+18>:    call   0x555555555050 <puts@plt>
   0x555555555160 <main+23>:    mov    $0x0,%eax
   0x555555555165 <main+28>:    pop    %rbp
   0x555555555166 <main+29>:    ret
   0x555555555167:      add    %dh,%bl
   0x555555555169 <_fini+1>:    nop    %edx
   0x55555555516c <_fini+4>:    sub    $0x8,%rsp
   0x555555555170 <_fini+8>:    add    $0x8,%rsp
(gdb)
```

As seen here, we know that the address of `main` is `0x555555555151`. Now, when searching for the address of main in `helloworld_stripped`, we can cross-reference with this address.

We can run `helloworld_stripped` in GDB to begin searching for the address of `main`. Since the binary is dynamically linked and it uses `libc`, we know that it will contain `__libc_start_main`. As per the Linux Standard Base PDA Specification, we know that the first argument of `__libc_start_main` is a pointer to the `main` function. Because of this, we can inspect the `rdi` (argument 1) register once the function is called.
To perform this in GDB, we will place a breakpoint at `__libc_start_main`. Since the binary has not yet been run, we will make the breakpoint pending on a future shared library load. We will then inspect `rdi` and set a new breakpoint at that address, which will be `main`.



```
❯ gdb -nx helloworld_stripped
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04.2) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from helloworld_stripped...
(No debugging symbols found in helloworld_stripped)
(gdb) b __libc_start_main
Function "__libc_start_main" not defined.
Make breakpoint pending on future shared library load? (y or [n]) y
Breakpoint 1 (__libc_start_main) pending.
(gdb) r
Starting program: /home/user/csce_413/class_34/helloworld_stripped
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, __libc_start_main_impl (main=0x555555555149, argc=1, argv=0x7fffffffddc8, init=0x0, f
ini=0x0, rtld_fini=0x7ffff7fc9040 <_dl_fini>, stack_end=0x7fffffffddb8) at ../csu/libc-start.c:242
242     ../csu/libc-start.c: No such file or directory.
(gdb) i r rdi
rdi            0x555555555149      93824992235849
(gdb) b *0x555555555149
Breakpoint 2 at 0x555555555149
(gdb) c
Continuing.

Breakpoint 2, 0x0000555555555149 in ?? ()
(gdb) x/10i $rip
=> 0x555555555149:      endbr64
   0x55555555514d:      push   %rbp
   0x55555555514e:      mov    %rsp,%rbp
   0x555555555151:      lea    0xeac(%rip),%rax        # 0x555555556004
   0x555555555158:      mov    %rax,%rdi
   0x55555555515b:      call   0x555555555050 <puts@plt>
   0x555555555160:      mov    $0x0,%eax
   0x555555555165:      pop    %rbp
   0x555555555166:      ret
   0x555555555167:      add    %dh,%bl
(gdb)
```

As seen here, we have found the beginning of `main`, which is `0x555555555149`. We know this is `main` because we found the same addresses as `helloworld`. We can now automate this process with Python-GDB.

Using Python-GDB

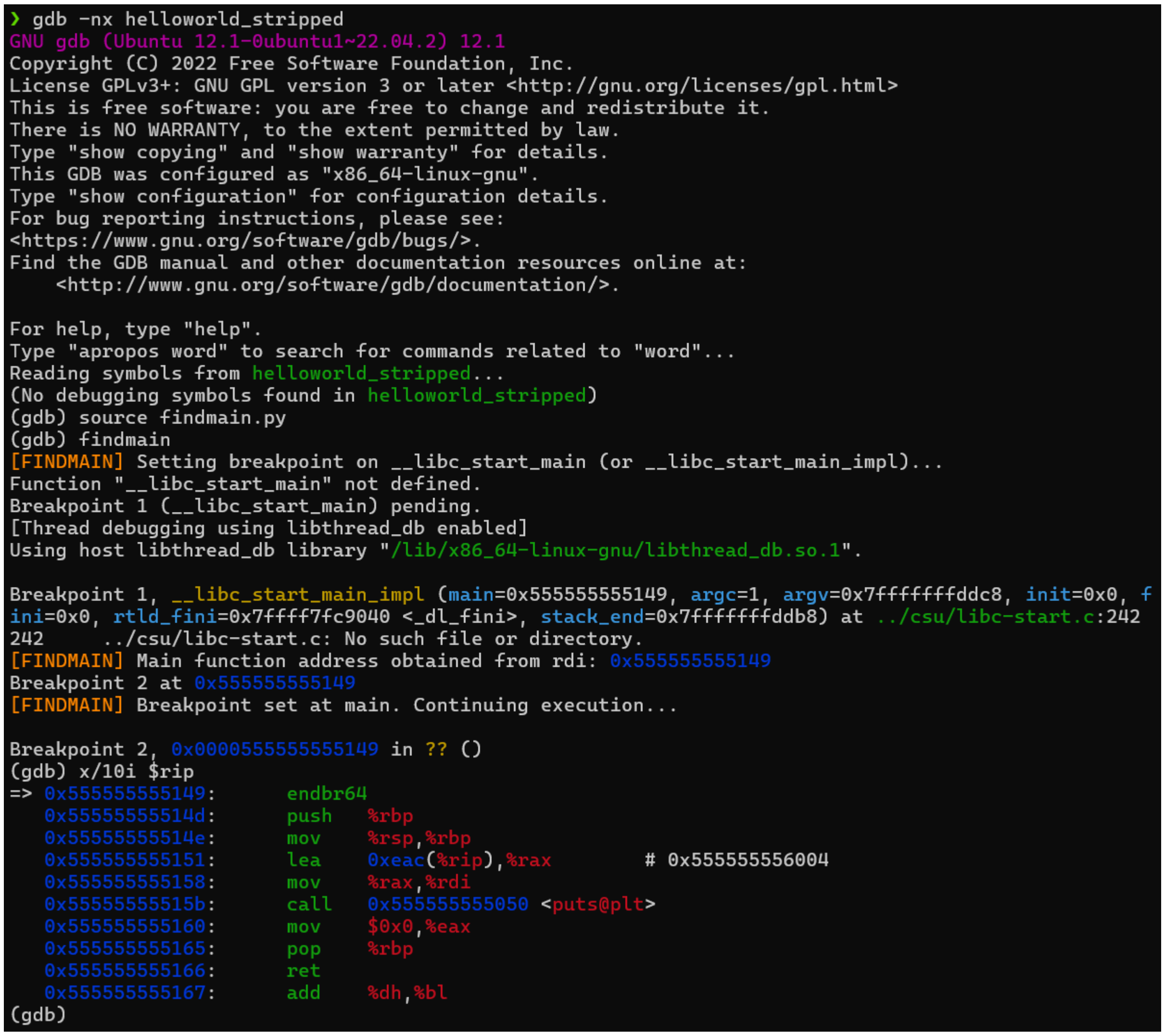
To automate the process of finding `main` in GDB, regardless if the binary is stripped or not, I have created the Python script `findmain.py`,

```
1  import gdb
2
3  class FindMain(gdb.Command):
4      def __init__(self):
5          super(FindMain, self).__init__("findmain", gdb.COMMAND_USER)
6
7      def invoke(self, argument, from_tty):
8          # Set colors
9          orange = "\033[38;5;208m"
10         reset = "\033[0m"
11         blue = "\033[34m"
12
13         # Set a breakpoint at __Libc_start_main
14         gdb.write(f"{orange}[FINDMAIN]{reset} Setting breakpoint on __libc_start_main (or
               __libc_start_main_impl)...\n")
15         bp = gdb.Breakpoint("__libc_start_main")
16
17         # Run the program
18         gdb.execute("run")
19
20         # Create a frame and read the rdi register to find the start of main
21         frame = gdb.selected_frame()
22         main_addr = frame.read_register("rdi")
23
24         # Print that main has been found
25         main_addr_hex = format(int(main_addr), '#x')
26         gdb.write(f"{orange}[FINDMAIN]{reset} Main function address obtained from rdi: {blue}{
               main_addr_hex}{reset}\n")
27
28         # Create a breakpoint at main
29         gdb.execute("break *{}".format(main_addr_hex))
30         gdb.write(f"{orange}[FINDMAIN]{reset} Breakpoint set at main. Continuing execution...\n")
31
32         # Continue to the breakpoint at main
33         gdb.execute("continue")
34
35 FindMain()
```

This program uses Python's GDB library to perform the same steps as used to previously find `main`. To elaborate;

1. Line 14 sets a breakpoint at `__libc_start_main`.
2. Line 18 runs the program.
3. Line 22 reads the `rdi` register to find the address of main.
4. Line 29 creates a breakpoint at the main address.
5. Line 33 continues execution, stopping at main.

The following screenshot is an example of this script successfully finding the `main` function of `helloworld_stripped`.



```
❯ gdb -nx helloworld_stripped
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04.2) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from helloworld_stripped...
(No debugging symbols found in helloworld_stripped)
(gdb) source findmain.py
(gdb) findmain
[FINDMAIN] Setting breakpoint on __libc_start_main (or __libc_start_main_impl)...
Function "__libc_start_main" not defined.
Breakpoint 1 (__libc_start_main) pending.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, __libc_start_main_impl (main=0x555555555149, argc=1, argv=0x7fffffffddc8, init=0x0, f
ini=0x0, rtld_fini=0x7ffff7fc9040 <_dl_fini>, stack_end=0x7fffffffddb8) at ../csu/libc-start.c:242
242     ../csu/libc-start.c: No such file or directory.
[FINDMAIN] Main function address obtained from rdi: 0x555555555149
Breakpoint 2 at 0x555555555149
[FINDMAIN] Breakpoint set at main. Continuing execution...

Breakpoint 2, 0x0000555555555149 in ?? ()
(gdb) x/10i $rip
=> 0x555555555149:      endbr64
   0x55555555514d:      push   %rbp
   0x55555555514e:      mov    %rsp,%rbp
   0x555555555151:      lea    0xeac(%rip),%rax        # 0x555555556004
   0x555555555158:      mov    %rax,%rdi
   0x55555555515b:      call   0x555555555050 <puts@plt>
   0x555555555160:      mov    $0x0,%eax
   0x555555555165:      pop    %rbp
   0x555555555166:      ret
   0x555555555167:      add    %dh,%bl
(gdb)
```

As seen here, the `findmain.py` script has successfully found and created a breakpoint at the `main` function.