GDB Handout

GDB is a debugger that pretty much makes you a magician. You can use the gdb to analyse the program while it is running. Following are the main superpowers you get with gdb:

- Checking out the stack of the program
- Checking out the assembly code of the program while it is running
- Setting up a breakpoint at a particular instruction in code
- Executing program instructions one by one and observing its effects.

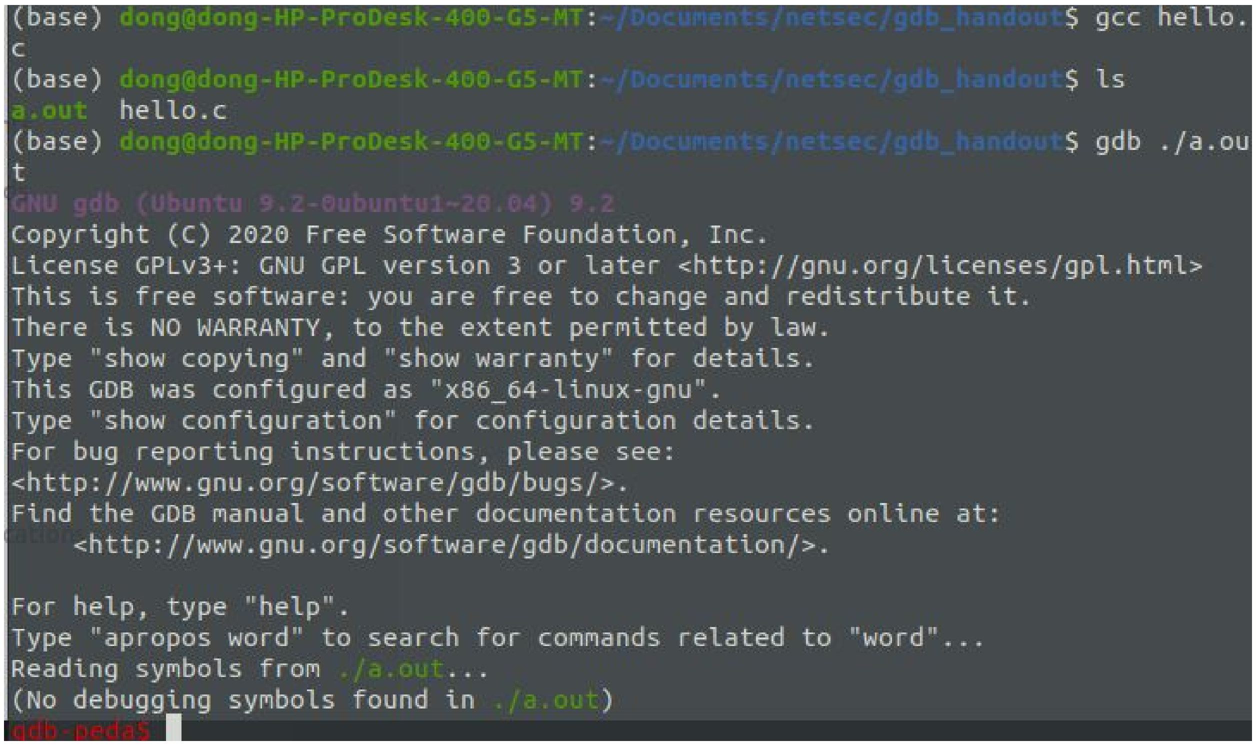
Starting up

To start gdb with a particular program, you can use the following command:

```
$ gdb executable
```

If the executable is named a.out, above command would be:

```
$ gdb ./a.out
```



```
(base) dong@dong-HP-ProDesk-400-G5-MT:~/Documents/netsec/gdb_handout$ gcc hello.
c
(base) dong@dong-HP-ProDesk-400-G5-MT:~/Documents/netsec/gdb_handout$ ls
a.out  hello.c
(base) dong@dong-HP-ProDesk-400-G5-MT:~/Documents/netsec/gdb_handout$ gdb ./a.ou
t
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
Copyright (C) 2020 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:
<http://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 ./a.out...
(No debugging symbols found in ./a.out)
gdb-peda$
```

It will start the gdb as follows.

Then, you can set up a **breakpoint**. A breakpoint is a place where gdb stops the code and allows you to run the code instruction by instruction.

You can set up a break point as follows:

```
gdb$ break function_name
```

E.g to set up a break point on main() function, you will write this.

```
gdb$ break main
```

Now, you can start running the code, using

```
gdb$ run
```

After that you can go to the next instruction using the following instruction:

```
gdb$ ni
```

If you would like to execute more than one instruction in one go, use this:

```
gdb$ n number_of_instruction
```

E.g, this would run next five instructions

```
gdb$ n 5
```

If you would like to step inside a function and run its instruction as well, you can use this instruction:

```
gdb$ si
```

Moreover, if you want to checkout the stack at a particular memory and print out n number of words from that location, you can do that using following:

```
gdb$ x/nw *memory_location
```