Lab 2.2 Running a Hello World Program in C using GCC

Overview

The lab helps familiarize you with writing a simple Hello World program using C, the GCC compiler link, and Pico(a text editor, link). It uses Ubuntu VM created in Lab 2.1.Here is lab objective:

1. Learn to run a program in gcc.
2. Learn to debug a program in gdb.

Steps

1. Open the Terminal in Ubuntu.
2. Create `debug_me.c` and add head file.

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

/* print a given string and a number if a pre-determined format. */
void print_string(int num, char* string)
{
      printf("String '%d' - '%s'\n", num, string);
}

int main(int argc, char* argv[])
{
      int i;

      /* check for command line arguments */
      if (argc < 2) { /* 2 - 1 for program name (argv[0]) and one for a param. */
            printf("Usage: %s [ ...]\n", argv[0]);
    exit(1);
     }

      /* loop over all strings, print them one by one */
      for (argc--,argv++,i=1 ; argc > 0; argc--,argv++,i++) {
            print_string(i, argv[0]);   /* function call */
     }

      printf("Total number of strings: %d\n", i);

      return 0;
}
```

```
xx@xx-virtual-machine: ~
File  Edit  View  Search  Terminal  Help
#include <stdio.h>
#include <stdlib.h>

/* print a given string and a number if a pre-determined format. */
void print_string(int num, char* string)
{
    printf("String '%d' - '%s'\n", num, string);
}

int main(int argc, char* argv[])
{
    int i;

    /* check for command line arguments */
    if (argc < 2) { /* 2 - 1 for program name (argv[0]) and one for a param. */
        printf("Usage: %s [ ...]\n", argv[0]);
        exit(1);
    }

    /* loop over all strings, print them one by one */
    for (argc--,argv++,i=1 ; argc > 0; argc--,argv++,i++) {
        print_string(i, argv[0]);  /* function call */
    }

    printf("Total number of strings: %d\n", i);

    return 0;
}
~
~
~
-- INSERT --                                              24,1          All
```

3. Invoke gdb to debug 'debug_me.c':

```
$ gcc -g debug_me.c -o debug_me
$ gdb debug_me
```

```
xx@xx-virtual-machine:~$ vim debug_me.c
xx@xx-virtual-machine:~$ gcc -g debug_me.c -o debug_me
xx@xx-virtual-machine:~$ gdb debug_me
GNU gdb (Ubuntu 8.1.1-0ubuntu1) 8.1.1
Copyright (C) 2018 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 debug_me...done.
```

4. Run the program inside gdb:

```
(gdb) run "hello, world" "goodbye, world"
```

```
(gdb) run "hello, world" "goodbye, world"
Starting program: /home/xx/debug_me "hello, world" "goodbye, world"
String '1' - 'hello, world'
String '2' - 'goodbye, world'
Total number of strings: 3
[Inferior 1 (process 4785) exited normally]
```

5. Set breakpoints. You can set breakpoints using two methods:

(1)Specifying a specific line of code to stop in:

```
(gdb) break debug_me.c:15
```

(2)Specifying a function name, to break every time it is being called:

```
(gdb) break main
```

```
(gdb) break debug_me.c:15
Breakpoint 1 at 0x5555555546c5: file debug_me.c, line 15.
(gdb) break main
Note: breakpoint 1 also set at pc 0x5555555546c5.
Breakpoint 2 at 0x5555555546c5: file debug_me.c, line 15.
```

6. Step a command at a time. After setting breakpoints, you can run the program step by step. There are also two methods:

```
(gdb) next
```

or

```
(gdb) step
```

You should tell the difference between "next" and "step".

```
(gdb) run "hello, world" "goodbye, world"
Starting program: /home/xx/debug_me "hello, world" "goodbye, world"

Breakpoint 1, main (argc=3, argv=0x7fffffffdf48) at debug_me.c:15
15          if (argc < 2) { /* 2 - 1 for program name (argv[0]) and one for a pa
ram. */
(gdb) next
21          for (argc--,argv++,i=1 ; argc > 0; argc--,argv++,i++) {
(gdb) next
22              print_string(i, argv[0]);  /* function call */
(gdb) next
String '1' - 'hello, world'
21          for (argc--,argv++,i=1 ; argc > 0; argc--,argv++,i++) {
(gdb) step
22              print_string(i, argv[0]);  /* function call */
(gdb) step
print_string (num=2, string=0x7fffffffe2e8 "goodbye, world") at debug_me.c:7
7           printf("String '%d' - '%s'\n", num, string);
```

The difference between "next" and "step" is that "step" stops inside a called function, while "next" executes called functions at (nearly) full speed, stopping only at the next line in the current function. Note that if thread debugging is enabled, the next statement may be in a different thread.

7. Print variables. When running the program step by step, you can print the contents of a variable with a command like this:

```
(gdb) print i
```

```
(gdb) run "hello, world" "goodbye, world"
Starting program: /home/xx/debug_me "hello, world" "goodbye, world"

Breakpoint 1, main (argc=3, argv=0x7fffffffdf48) at debug_me.c:15
15          if (argc < 2) { /* 2 - 1 for program name (argv[0]) and one for a p
aram. */
(gdb) step
21          for (argc--,argv++,i=1 ; argc > 0; argc--,argv++,i++) {
(gdb) print i
$1 = 0
(gdb) step
22              print_string(i, argv[0]);  /* function call */
(gdb) print i
$2 = 1
```

```
(gdb) finish
Run till exit from #0  __printf (format=0x5555555547d4 "String '%d' - '%s'\n")
    at printf.c:33
String '1' - 'hello, world'
print_string (num=1, string=0x7fffffffe2db "hello, world") at debug_me.c:8
8       }
Value returned is $3 = 28
(gdb) next
main (argc=2, argv=0x7fffffffdf50) at debug_me.c:21
21          for (argc--,argv++,i=1 ; argc > 0; argc--,argv++,i++) {
(gdb) print i
$4 = 1
(gdb) next
22              print_string(i, argv[0]);  /* function call */
(gdb) print i
$5 = 2
(gdb) next
String '2' - 'goodbye, world'
21          for (argc--,argv++,i=1 ; argc > 0; argc--,argv++,i++) {
(gdb) print i
$6 = 2
(gdb) step
25          printf("Total number of strings: %d\n", i);
(gdb) print i
$7 = 3
```

8. Examine the function call stack.

(1)Run the the program step by step to line 7. To examine the function call stack, type:

```
(gdb) where
```

```
(gdb) run "hello, world" "goodbye, world"
Starting program: /home/xx/debug_me "hello, world" "goodbye, world"

Breakpoint 1, main (argc=3, argv=0x7fffffffdf48) at debug_me.c:15
15          if (argc < 2) { /* 2 - 1 for program name (argv[0]) and one for a p
aram. */
(gdb) step
21          for (argc--,argv++,i=1 ; argc > 0; argc--,argv++,i++) {
(gdb) step
22              print_string(i, argv[0]);  /* function call */
(gdb) step
print_string (num=1, string=0x7fffffffe2db "hello, world") at debug_me.c:7
7           printf("String '%d' - '%s'\n", num, string);
(gdb) where
#0  print_string (num=1, string=0x7fffffffe2db "hello, world") at debug_me.c:7
#1  0x0000555555554716 in main (argc=2, argv=0x7fffffffdf50) at debug_me.c:22
```

(2)You will see currently executing function "print_string" and the function "main" which called it. Then type "frame 0" and "frame 1" to see the difference:

```
(gdb) frame 0
(gdb) print i
...
(gdb) frame 1
(gdb) print i
```

```
(gdb) frame 0
#0  print_string (num=1, string=0x7fffffffe2db "hello, world") at debug_me.c:7
7           printf("String '%d' - '%s'\n", num, string);
(gdb) print i
No symbol "i" in current context.
(gdb) frame 1
#1  0x0000555555554716 in main (argc=2, argv=0x7fffffffdf50) at debug_me.c:22
22              print_string(i, argv[0]);  /* function call */
(gdb) print i
$9 = 1
```

The value of i is different in different frames because i is not defined in `print_string` but in `main` its value is equal to 1.