Function breakpoints, conditional breakpoints, watchpoint, and the continue and finish commands

In this example, we will learn how to set function breakpoints, conditional breakpoints, and use the `continue` command. Then, we will learn how to finish a function call without the need to execute all the code lines in a step-by-step format. The source code is at https://github.com/PacktPublishing/Expert-C-2nd-edition/blob/main/Chapter14/ch14_gdb_2.cpp.

The `dotproduct` function takes two arrays (`x` and `y`) and the size of the arrays (`n`) as parameters. It initializes a pointer `p` to point to the start of array `x` and another pointer `q` also pointing to the start of array `x` (this is a bug and should be `y`). It then calculates the dot product by iterating over the arrays and accumulating the product of corresponding elements.

Again, after building and running `ch14_gdb_2.cpp`, we get the following output:

```
~/wus1/Chapter-13$ g++ -g ch14_gdb_2.cpp -o ch14_gdb_2.out
~/wus1/Chapter-13$ ./ch14_gdb_2.out
dot(x,x) = 55.000000
dot(x,y) = 55.000000
```

Since both `dot(x,x)` and `dot(x,y)` give us the same results, something must be wrong here. Now, let's debug it by learning how to set a breakpoint in the `dot()` function:

- **Function breakpoint**: To set a breakpoint at the beginning of a function, we can use the `b function_name` command. As always, we can use tab completion during input. For instance, let's say we type the following:

```
   (gdb) b dot<Press TAB Key>
```

The following command line will automatically pop up if we do this:

```
   (gdb) b dotproduct(float const*, float const*, int)
```

If it is a member function of a class, its class name should be included, as follows:

```
   (gdb) b MyClass::foo(<Press TAB key>
```

- **Conditional breakpoint**: There are several ways to set a conditional breakpoint:

```
(gdb) b f.cpp:26 if s==0 //set a breakpoint in f.cpp, line
                         //26 if
s==0
(gdb) b f.cpp:20 if ((int)strcmp(y, "hello")) == 0
```

- **List and delete breakpoints**: Once we've set a few breakpoints, we can list or delete them, as follows:

```
     (gdb) i
     (gdb) delete breakpoints 1
     (gdb) delete breakpoints 2-5
```

- **Remove make a breakpoint unconditional**: Since each breakpoint has a number, we can remove a condition from a breakpoint, like so:

```
     (gdb) cond 1 //break point 1 is unconditional now
```

- **Watchpoint**: A watchpoint can stop execution when the value of an expression changes, without having to predict where (in which line) it may happen. There are three kinds of watchpoints:
  - `watch: gdb` will break when a write occurs
  - `rwatch: gdb` will break when a read occurs
  - `awatch: gdb` will break when either a write or a read happens

The following code shows an example of this:

```
   (gdb) watch v //watch the value of variable v
   (gdb) watch *(int*)0x12345678 //watch an int value
pointed
  by an address
   (gdb) watch a*b + c/d // watch an arbitrarily complex
  expression
```

- continue: When we've finished examining the values of variables at a breakpoint, we can use the continue or c command to continue program execution until the debugger encounters a breakpoint, a signal, an error, or normal process termination.
- finish: Once we go inside a function, we may want to execute it continuously until it returns to its caller line. This can be done using the finish command.

Now, let's put these commands together to debug ch14_gdb_2.cpp.The following is the output from our Terminal window. For your convenience, we've separated it into three parts:

```
//gdb output of example ch14_gdb_2.out -- part 1
~/wus1/Chapter-13$ gdb ch14_gdb_2.out                   //cmd 1
...
Reading symbols from ch14_gdb_2.out ... done.

(gdb) b dotproduct(float const*, float const*, int)     //cmd 2
Breakpoint 1 at 0xa5c: file ch14_gdb_2.cpp, line 20.
(gdb) b ch14_gdb_2.cpp:24 if i==1                       //cmd 3
Breakpoint 2 at 0xa84: file ch14_gdb_2.cpp, line 24.
(gdb) i b                                               //cmd 4
Num Type Disp Enb Address What
1 breakpoint keep y 0x0000000000000a5c in dotproduct(float const*,
float
const*, int) at ch14_gdb_2.cpp:20
2 breakpoint keep y 0x0000000000000a84 in dotproduct(float const*,
float
const*, int) at ch14_gdb_2.cpp:24
stop only if i==1
(gdb) cond 2                                            //cmd 5
Breakpoint 2 now unconditional.
(gdb) i b                                               //cmd 6
Num Type Disp Enb Address What
1 breakpoint keep y 0x0000000000000a5c in dotproduct(float const*,
float
const*, int) at ch14_gdb_2.cpp:20
2 breakpoint keep y 0x0000000000000a84 in dotproduct(float const*,
float
const*, int) at ch14_gdb_2.cpp:24
```

In part one, we have the following six commands:

- cmd 1: We start gdb with the parameter of the built executable file, ch14_gdb_2.out. This briefly shows us its version and document and usage information, and then tells us that the reading symbols process has been completed and is waiting for the next command.
- cmd 2: We set a breakpoint function (at dotproduct()).
- cmd 3: A conditional breakpoint is set.
- cmd 4: It lists information about the breakpoints and tells us that we have two of them.
- cmd 5: We set breakpoint 2 as unconditional.
- cmd 6: We list the breakpoint information again. At this point, we can see two breakpoints. These are located at *lines 20 and 24* in the ch14_gdb_2.cp file, respectively.

Next, let's look at the gdb output in part two:

```
//gdb output of example ch14_gdb_2.out -- part 2
(gdb) r                                         //cmd 7
Starting program: /home/nvidia/wus1/Chapter-13/ch14_gdb_2.out
[Thread debugging using libthread_db enabled]
```

```
Using host libthread_db library "/lib/aarch64-linuxgnu/
libthread_db.so.1".
Breakpoint 1, dotproduct (x=0x7fffffed68, y=0x7fffffed68, n=5) at
ch14_gdb_2.cpp:20
20 const float *p = x;
(gdb) p x                                        //cmd 8
$1 = (const float *) 0x7fffffed68
(gdb) c                                          //cmd 9
Continuing.
Breakpoint 2, dotproduct (x=0x7fffffed68, y=0x7fffffed68, n=5) at
ch14_gdb_2.cpp:24
24 s += (*p) * (*q);
(gdb) p i                                        //cmd 10
$2 = 0
(gdb) n                                          //cmd 11
23 for(int i=0; i<n; ++i, ++p, ++q){
(gdb) n                                          //cmd 12
Breakpoint 2, dotproduct (x=0x7fffffed68, y=0x7fffffed68, n=5) at
ch14_gdb_2.cpp:24
24 s += (*p) * (*q);
(gdb) p s                                        //cmd 13
$4 = 1
(gdb) watch s                                    //cmd 14
Hardware watchpoint 3: s
```

Part two has the following cmds:

- `cmd 7`: By giving the `run` command, the program starts running and stops at the first breakpoint in file `ch14_gdb_2.cpp`, *line 20*.
- `cmd 8`: We print the value of `x`, which shows its address.
- `cmd 9`: We continue the program. Once it's been continued, it stops at the second breakpoint in *line 24*.
- `cmd 10`: The value of `i` is printed, which is `0`.
- `cmd 11-12`: We use the `next` command twice. At this point, the `s += (*p) * (*q)` statement is executed.
- `cmd 13`: The value of `s` is printed, which is `1`.
- `cmd 14`: We print the value of `s` .

Finally, part three is as follows:

```
//gdb output of example ch14_gdb_2.out -- part 3
(gdb) n                                          //cmd 15
Hardware watchpoint 3: s
Old value = 1
New value = 5
dotproduct (x=0x7fffffed68, y=0x7fffffed68, n=5) at
ch14_gdb_2.cpp:23
23 for(int i=0; i<n; ++i, ++p, ++q){
(gdb) finish                                     //cmd 16
Run till exit from #0 dotproduct (x=0x7fffffed68, y=0x7fffffed68,
n=5) at
ch14_gdb_2.cpp:23
Breakpoint 2, dotproduct (x=0x7fffffed68, y=0x7fffffed68, n=5) at
ch14_gdb_2.cpp:24
24 s += (*p) * (*q);
(gdb) delete breakpoints 1-3                     //cmd 17
(gdb) c                                          //cmd 18
Continuing.
```

```
dot(x,x) = 55.000000
dot(x,y) = 55.000000
[Inferior 1 (process 31901) exited normally]
[Inferior 1 (process 31901) exited normally]
(gdb) q                                                    //cmd 19
~/wus1/Chapter-13$
```

In this part, we have the following commands:

- `cmd 15`: We use the next command to see what the value of `s` is if the next iteration is executed. It shows that the old value of `s` is 1 (s = 1*1) and that the new value is `5` (s=1*1+2*2). So far, so good!
- `cmd 16`: A `finish` command is used to continue running the program until it exits from the function.
- `cmd 17`: We delete breakpoints 1 to 3.
- `cmd 18`: A `continue` command is used.
- `cmd 19`: We quit `gdb` and go back to the Terminal window.