Controlling Execution

/run/media/lijunjie/资料/Learning Video/writing-running-fixing-code/03_testing-and-debugging/03_debugging-tools/05_controlling-execution_instructions.html

The *next* and *step* commands give you the basic ability to advance the state of the program, however, there are also more advanced commands for controlling the execution. If we are debugging a large, complex program, we may not want to step through every line one-by-one to reach the point in the program where we want to gather information.

One of the most useful ways to control the execution of our program is to set a *breakpoint* on a particular line. A breakpoint instructs *gdb* to stop execution whenever the program reaches that particular line. You can set a breakpoint with the *break* command, followed by either a line number, or a function name (meaning to set the breakpoint at the start of that function). In emacs, you can also press C-x space to set a breakpoint at the point. It is also possible to set a breakpoint at a particular memory address, although that is a more advanced feature. When we set a breakpoint, *gdb* will assign it a number, which we can use to identify it to other breakpoint-related commands.

Once we have a breakpoint set, we can *run* the program (or *continue* , if it is already started), and it will execute until the breakpoint is encountered (or some other condition which causes execution to stop). When the breakpoint is encountered, *gdb* will return control to us at a (*gdb)* prompt, allowing us to give it other commands—we might inspect the state of the program, set more breakpoints, and continue.

By default, breakpoints are *unconditional breakpoints — gdb* will stop the program and give you control anytime it reaches the appropriate line. Sometimes, however, we may want to stop under a particular condition. For example, we may have a **for** loop which executes 1,000,000 times, and we need information from the iteration where *i* is *250,000* . With an unconditional breakpoint, the program would stop, and we would need to continue many times before we got the information we wanted. We can instead, use a *conditional breakpoint* —once where we give *gdb* a C expression to evaluate to determine if it should give us control, or let the program continue to run.

We can put a condition on a breakpoint when we create it with the break command by writing if after the location, followed by the conditional expression. We can also add a condition later (or change an existing condition) with the cond command. For example, if we want to make a breakpoint on line size for the condition i==25000, we could tell gdb:

```
(gdb) break 7 if i==250000
```

Alternatively, if the breakpoint already existed, for example, as breakpoint 1, we could write

```
cond 1 i==250000
```

If we write a *cond* command with no expression, then it makes a breakpoint unconditional. We can also *enable* or *disable* breakpoints (by their numeric id). A disabled breakpoint still exists (and can be re-enabled later), but has no effect—it will not cause the program to stop. We can also *delete* a breakpoint by its numeric id. You can use the *info breakpoints* command (which can be abbreviated *i b*) to see the status of current breakpoints.

Two other useful commands to control the execution of the program are *until* , which causes a loop to execute until it finishes (*gdb* stops at the first line after the loop), and *finish* (which can be abbreviated fin), which finishes the current function— *i.e.* , causes execution until the current function returns.