GNU C++ Debugger GDB

May. 24, 2019

Pei-Luan Tai

Outline

1. Introduction
2. basic commands: `run`, `quit`, `break`, `list`, `next/step`, `help`
3. examine tool: `print`, `examine`, `info`, `watch`
4. interactive demo1 and demo2.
5. gdb script

Essential debugging technique:
print out the info and then
check whether it matches your expectation.

But… It could mess up your source code, especially for complex bugs
You might:

- to set some variables to specific values for experimenting.
- to disable/enable certain “if statements” for experimenting.

GDB helped me debug a strange behavior of dose engine; the program was halted at certain steps **randomly** after around 20,000 steps with **previous nuclear physics implementation** and my new implementation for the MCS process.

For simpler bugs (no more than a few hours to solve),
I mostly use print commands.

To use GDB, the first thing is to add `-g` or `-ggdb` option for the g++ compiler.

Then;

```
$ gdb  your_exec
```

You will see the welcome screen as the following:

```
GNU gdb (Ubuntu 7.7.1-0ubuntu5~14.04.3) 7.7.1
Copyright (C) 2014 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 "i686-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 code1...done.

##(gdb)
```

The prompt can be customized in ~/.gdbinit file.

To quit GDB: type in `quit` or `q`, or `ctrl+d`.

Basic commands: run or r

```
(gdb) run arg1 arg2 arg3 ….
(gdb) r   arg1 arg2 arg3 ….
```

If you want to automatically set the arguments every time you run the executable:
Use "`set args`" command.

```
(gdb) set args arg1 arg2 arg3 …
```

If there are breakpoints, the program will be halted at the first encountered breakpoint.

Basic commands: break or b (1)

```
break 10          ( set a breakpoint at line 10 )
break xyz.cpp:10  ( set a breakpoint at xyz.cpp line 10 )

break my_func     ( whenever my_func is invoked, break there )
break xyz.cpp:my_func
```

Suppose we have a function: func(int x, int y)
and we only want to check what happen when x==1.
we can do the following:

```
(gdb) break func if x == 1
```

Basic commands: break or b (2)

Using "`continue`" or "`c`" command allows us to go directly to next break point.

DISABLE OR DELETE BREAKPOINTS

`disable n1 n2 n3` ... (disable breakpoint n1 n2 n3 ...)

`delete n1 n2 n3` ... (delete breakpoint n1 n2 n3 ...)

`info break` can show the break info

Basic commands: list or l

list command can print the source code:

```
list
list 10     <== center at line 10
list 1,50   <== show line 1 to line 50
list function_name
list file:function_name
```

```
5
6           int num;
7
8           do{
9
10              printf( "Enter a positive integer: " );
11
12              std::cin >> num;
13
14          } while ( num < 0 );
```

```
where <== show you where you are at
info line <== show you the current line
```

Basic commands: next/step or n/s

```
int main() {
    int x1 = func1(10);
    int x2 = func2(20);
    int x3 = func3(30);
}
```

"next" commands will not go into a subroutine.
"step" will let us go into func1(), func2(), and func3().

One also can do multiple “next”
`next [N]`,
e.g. `next 3` <== next three subroutines.

examining tool: print or p

```
print my_var
print &my_var  <== memory address
print sizeof( my_var)
```

You use print command to set the value as well:

```
print my_var=10 <== let my variable = 10
```

examining tool: x

“x/Format” can provide low level information.

```
char my_str[50] = “123“;
(gdb) x my_str
0xbfffee08:          0x31
```

It prints out the first byte (1 char = 1 byte), 0x31 = '1'
"0xbfffee08" is the memory address of my_str variable.

```
(gdb) x/6b my_str
0xbfffee08:    0x31    0x32    0x33    0x00    0x00    0x00
```

It prints out first 6 bytes for the my_str.

examining tool: watch

"watch" command sets a watchpoint for an expression.

```
int main( int argc, char** argv ) {

   int x = 0;
   x = func1(10);
   x = func2(20);
   x = func3(30);
}
```

For example, we want to check what the value of x whenever it changes.
then, type in "`watch x`"
it is kind of break, and so we use "continue" command,
when the value of variable x changes, gdb will stop the process,
then print out the old and new value.

examining tool: info

info locals (information for local variables)
info breakpoints
info args
info watch

examining tool: backtrace or bt

"backtrace" command allows us to trace back to understand which are the previous functions to call the current function.

Interactive demo 1: factorial

Interactive demo 2: LinkedList

gdb script example:

in your sh file:

```
gdb --command=gdb_script \
   --args $binPath/src/dose_engine -algorithm 0 \
                  -history_num ${particle_number} \
                  -energy_mean ${incident_energy} \
                   .....
```

in your gdb_script file:

```
break   …MonteCarlo_CPU_MT.cpp:540
run

## lopping
set $ipx=0
while ( $ipx < 395 )
    print line
    continue
    set $ipx=$ipx+1
end
```