redhat.

GDB Debugger's New Features

Jan Kratochvíl
GDB software developer, Red Hat
2017-01-25

GDB Debugger's New Features

Agenda

- Fully-featured Consoles
- Reusing g++ for C++ parsing
- gdbserver Even for Local Processes
- Container Debugging
- DWARF-5

Fully-featured Consoles

Fully-featured Consoles

old: Fedora 25 - Eclipse CDT 9.1

Debugger console was not really supported.

- Accessed by Debug → gdb
- It shows no "**(gdb)**" prompt
- There is no up/down arrow history
- Missing readline editing, tab-completion
- Implemented by MI command: **-interpreter-exec console "print 1"**

Fully-featured Consoles

new: Fedora 26 - Eclipse CDT 9.2

Debugger console is now a normal UI functionality.

- Accessed by Window → Show View → Debugger Console
- It behaves as a normal text GDB interface (CLI)
- Implemented by CLI command: **new-ui mi /dev/pts/6**
- https://wiki.eclipse.org/CDT/FullGDBConsole

Reusing g++ for C++ parsing

Reusing g++ for C++ parsing

(gdb) compile print/code ...

Internal GDB parser + evaluation:
(gdb) print 1+2
$1 = 3

External GCC compiler:
(gdb) compile print 1+2
$2 = 3

Reusing g++ for C++ parsing

(gdb) compile print/code ...

```
#include <map>
#include <vector>
struct Cache {
  const int i;
  std::vector<int> index;
  Cache(int i_):i(i_) { while (index.size()<10000) index.push_back(rand()); }
  bool operator < (const Cache &other) const { return i<other.i; }
};
int main() {
  std::map<Cache,int> cache;
  for (int i=0;i<10;++i)
    cache[Cache(i)]=i;
  return 0;
}
```

Reusing g++ for C++ parsing

(gdb) compile print/code ...

(gdb) p cache
$1 = std::map with 10 elements = {[{i = 0, index = std::vector of length 10000, capacity 16384 = {1804289383, 846930886, 1681692777, 1714636915, 1957747793, 424238335, 719885386, 1649760492, 596516649, 1189641421, 1025202362, 1350490027, 783368690, 1102520059, 2044897763, 1967513926, 1365180540, 1540383426, 304089172, 1303455736, 35005211, 521595368, 294702567, 1726956429, 336465782, 861021530, 278722862, 233665123, 2145174067, 468703135, 1101513929, 1801979802, 1315634022, 635723058, 1369133069, 1125898167, 1059961393, 2089018456, 628175011, 1656478042, 1131176229, 1653377373, 859484421, 1914544919, 608413784, 756898537, 1734575198, 1973594324, 149798315, 2038664370, 1129566413, 184803526, 412776091, 1424268980, 1911759956, 749241873, 137806862, 42999170, 982906996, 135497281, 511702305, 2084420925, 1937477084, 1827336327, 572660336, 1159126505, 805750846, 1632621729, 1100661313, 1433925857, 1141616124,

Reusing g++ for C++ parsing

(gdb) compile print/code ...

(gdb) set print elements 8
(gdb) p cache
$1 = std::map with 10 elements = {[{i = 0, index = std::vector of length 10000, capacity 16384 = {1804289383, 846930886, 1681692777, 1714636915, 1957747793, 424238335, 719885386, 1649760492...}}] = 0, [{i = 1, index = std::vector of length 10000, capacity 16384 = {89057537, 1840048410, 427773756, 762677667, 742585312, 1447032062, 1904054136, 1665967229...}}] = 1, [{i = 2, index = std::vector of length 10000, capacity 16384 = {851227066, 1907169184, 1672626973, 15224425, 1327707705, 1030165428, 454990854, 1575562599...}}] = 2, [{i = 3, index = std::vector of length 10000, capacity 16384 = {218746804, 231106831, 1109549898, 2097647419, 2119090367, 1322638762, 947642132, 1611526430...}}] = 3...}
(gdb) _

Reusing g++ for C++ parsing

(gdb) compile print/code ...

- Python pretty printer

```
(gdb) python
>class CachePrinter:
>       def __init__ (self, val):
>               self.val = val
>       def to_string (self):
>               return str (self.val['i'])
>def pretty_printer_lookup (val):
>       if str(val.type.unqualified()) == "Cache":
>               return CachePrinter(val)
>       return None
>gdb.pretty_printers.append(pretty_printer_lookup)
>end
(gdb) print cache
$2 = std::map with 10 elements = {[0] = 0, [1] = 10, [2] = 20, [3] = 30, [4] = 40, [5]
= 50, [6] = 60, [7] = 70, [8] = 80, [9] = 90}
```

Reusing g++ for C++ parsing

(gdb) compile print/code ...

(gdb) p cache[Cache(0)]
A syntax error in expression, near `(0)]'.
(gdb) p cache.begin()
Cannot evaluate function -- may be inlined

... what else?

- JIT-like C++ command

Reusing g++ for C++ parsing

(gdb) compile print/code ...

- JIT-like C++ command
- (lldb) expression
  for (const auto &it:cache) printf("%d=%d, ",it.first.i,it.second); putchar('\n');
  0=0, 1=10, 2=20, 3=30, 4=40, 5=50, 6=60, 7=70, 8=80, 9=90,
- (gdb) compile code
  for (const auto &it:cache) printf("%d=%d, ",it.first.i,it.second); putchar('\n');
  gdb command line:1:21: error: specialization of ‘template<class _Tp> class new_allocator’ in different namespace [-fpermissive]
  gdb command line:1:21: error: from definition of ‘template<class _Tp> class new_allocator’ [-fpermissive]
  gdb command line:1:21: error: unhandled TYPE_CODE_ERROR

Reusing g++ for C++ parsing

(gdb) compile print/code ...

```
#include <unistd.h>
#include <stdlib.h>
struct Cache {
  int i;
  int index[10000];
};
int main() {
  struct Cache cache[10];
  for (int i=0;i<10;++i) {
    cache[i].i=i*10;
    for (int j=0;j<10000;++j)
      cache[i].index[j]=rand();
  }
  return 0;
}
```

Reusing g++ for C++ parsing

(gdb) compile print/code ...

- JIT-like C command (C++ for GDB is expected in Fedora 27)

  (gdb) compile code for (int i=0;i<10;++i) printf("%d=%d, ",i,cache[i].i); putchar('\n');
  0=0, 1=10, 2=20, 3=30, 4=40, 5=50, 6=60, 7=70, 8=80, 9=90,

Reusing g++ for C++ parsing

(gdb) compile print/code ...

Future goals

- Using gcc/g++ for any expression evaluation: (gdb) print ...
  - core files - no debuggee execution (GCC GIMPLE/RTL?)
- Fast conditional breakpoints
- “Edit and continue” / “Fix and continue”

gdbserver Even for Local Processes

gdbserver Even for Local Processes

(gdb) target remote |gdbserver - ./hello

local gdb vs. remote gdbserver:

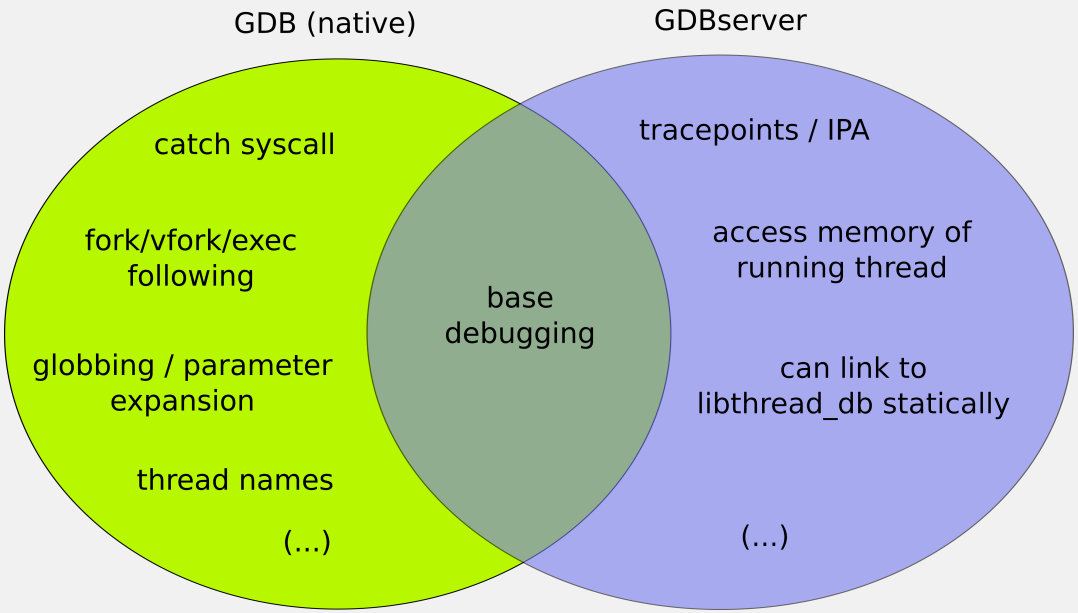
- local run: gdb program
- local attach: gdb -p `pidof program`
- remote run: gdbserver :1234 program

  gdb -ex 'target remote HOST:1234'
- remote attach: gdbserver :1234 `pidof program`
- gdb -ex 'target remote HOST:1234'

gdbserver Even for Local Processes

(gdb) target remote |gdbserver - ./hello

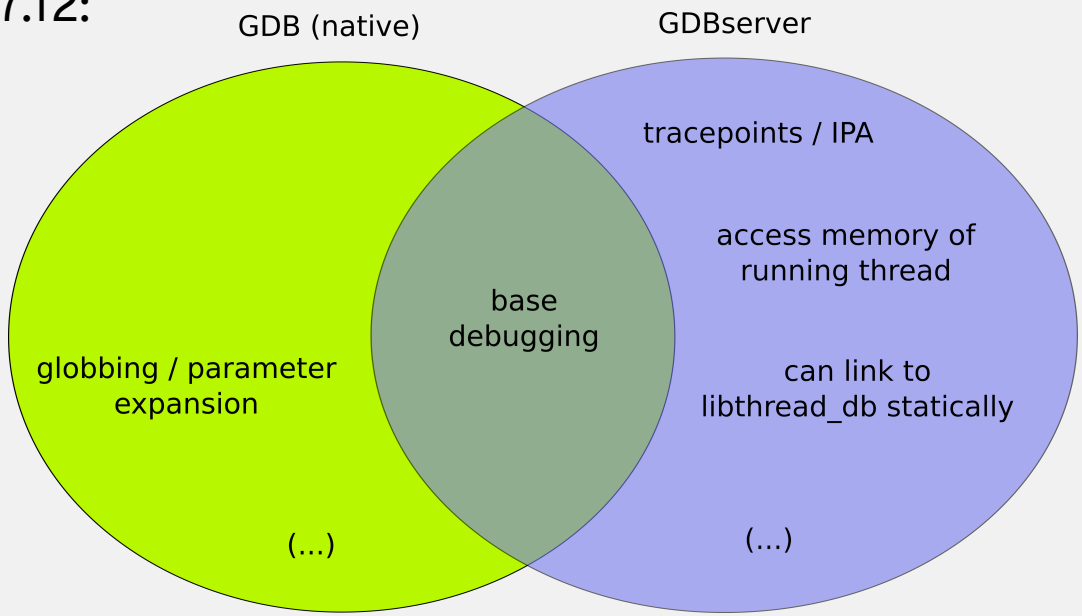
2014:



gdbserver Even for Local Processes

(gdb) target remote |gdbserver - ./hello

2016, gdb-7.12:



gdbserver Even for Local Processes

(gdb) target remote |gdbserver - ./hello

Already merged gdb→gdbserver:

- set follow-fork-mode (child|parent)
- set detach-on-fork on
- catch (fork|vfork|exec)
- catch syscall

gdbserver Even for Local Processes

(gdb) target remote |gdbserver - ./hello

Already merged gdb→gdbserver:

- (gdb) thread name bar

  (gdb) info threads

  * 1 Thread 0x77fc9740 (LWP 932) "bar" main () at foo.c:29

gdbserver Even for Local Processes

(gdb) target remote |gdbserver - ./hello

Still to merge gdb→gdbserver:

- $ gdb /usr/bin/ls
  (gdb) run *
  file1 file2
- gdbserver:
  Process /usr/bin/ls created; pid = 5260
  /usr/bin/ls: cannot access *: No such file or directory

gdbserver Even for Local Processes

(gdb) target remote |gdbserver - ./hello

Still to merge gdb→gdbserver:

- (gdb) set environment FOO=bar
- (gdb) cd /tmp # for inferior

  (gdb) pwd

  Working directory /tmp.

gdbserver Even for Local Processes

(gdb) target remote |gdbserver - ./hello

Still to merge gdb→gdbserver:

- Missing pthread_t:
- GDB native:

  (gdb) info threads

  * 1 Thread 0x7ffff7fcc740 (LWP 19056) "test" main () at test.c:35
- gdbserver:

  (gdb) info threads

  * 1 Thread 19056 "test" main () at test.c:35

Container Debugging

Container Debugging

One can docker run ... but how to run GDB for containerized programs?

Already done:

- Default sysroot is now “target:”
- Automatic “file” command even with gdbserver
- gdb+gdbserver container namespaces for files:
  - gdb -p `pidof program_in_container`

    Reading symbols from target:/lib64/libc.so.6...

Container Debugging

One can docker run … but how to run GDB for containerized programs?

WIP: libthread_db → Infinity:

- failing: gdb -p `pidof multithreaded_program_in_container`
- libthread_db: access internal glibc data independently from GDB
- libthread_db tries to attach threads by their containerized TIDs
- containerized libthread_db load: security issue + arch-compatibility
- https://infinitynotes.org/ - safe bytecode interpreter

DWARF-5

DWARF-5

{gcc,clang} -gdwarf-5 ...

- DWARF-5 vs. DWARF-4 is smaller and even faster (less relocations)
- Various GNU (Fedora) extensions are standardized now
  - .gdb_index → .debug_names - new format
  - DW_TAG_GNU_call_site → DW_TAG_call_site etc. - renames
  - DW_AT_deleted/DW_AT_defaulted, DW_AT_noreturn - new features
- http://dwarfstd.org/ - draft
- Fedora 26 gcc OK, Fedora 26 gdb off-trunk, clang+lldb seem to be OK
- gold is not updated - disabling: -fuse-ld=gold -Wl,--gdb-index

redhat.

THANK YOU

plus.google.com/+RedHat

facebook.com/redhatinc

linkedin.com/company/red-hat

twitter.com/RedHatNews

youtube.com/user/RedHatVideos