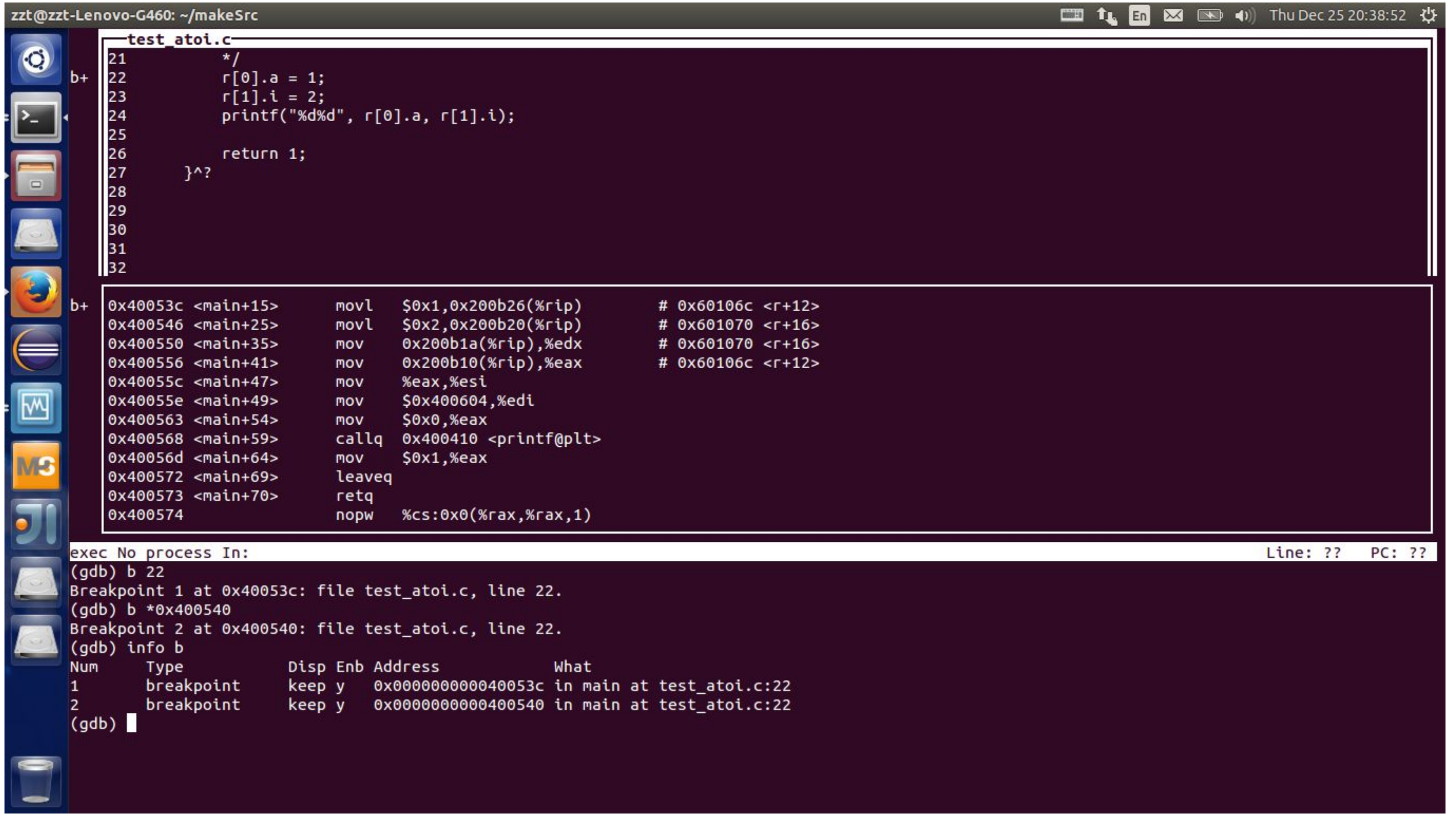


```
zzt@zzt-Lenovo-G460: ~/makeSrc                                   Thu Dec 25 20:38:52
 ┌─test_atoi.c──────────────────────────────────────────────────
 │21            */
b+│22            r[0].a = 1;
 │23            r[1].i = 2;
 │24            printf("%d%d", r[0].a, r[1].i);
 │25
 │26            return 1;
 │27        }^?
 │28
 │29
 │30
 │31
 │32

b+ 0x40053c <main+15>      movl   $0x1,0x200b26(%rip)        # 0x60106c <r+12>
   0x400546 <main+25>      movl   $0x2,0x200b20(%rip)        # 0x601070 <r+16>
   0x400550 <main+35>      mov    0x200b1a(%rip),%edx        # 0x601070 <r+16>
   0x400556 <main+41>      mov    0x200b10(%rip),%eax        # 0x60106c <r+12>
   0x40055c <main+47>      mov    %eax,%esi
   0x40055e <main+49>      mov    $0x400604,%edi
   0x400563 <main+54>      mov    $0x0,%eax
   0x400568 <main+59>      callq  0x400410 <printf@plt>
   0x40056d <main+64>      mov    $0x1,%eax
   0x400572 <main+69>      leaveq
   0x400573 <main+70>      retq
   0x400574                nopw   %cs:0x0(%rax,%rax,1)

exec No process In:                                               Line: ??   PC: ??
(gdb) b 22
Breakpoint 1 at 0x40053c: file test_atoi.c, line 22.
(gdb) b *0x400540
Breakpoint 2 at 0x400540: file test_atoi.c, line 22.
(gdb) info b
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x000000000040053c in main at test_atoi.c:22
2       breakpoint     keep y   0x0000000000400540 in main at test_atoi.c:22
(gdb) 
```

GDB

Danila Kutenin, Google

Telegram: @Danlark

History

- First version in 1986 by Richard Stallman
- Was only for C
- Many problems regarding C++
  - Templates
  - Starting from 2016 more or less stable. Linux kernel, lack of support, no unified design

GDB/LLDB

- Debuggers are platform/architecture specific
- Linux x86-64 and DWARF for ELFs
  - DWARF is a debug data format
  - Protocol is ptrace `$ man 2 ptrace`
    - long ptrace(enum __ptrace_request request, pid_t pid, void *addr, void *data);

GDB/LLDB

- long ptrace(enum __ptrace_request request, pid_t pid, void *addr, void *data);
- PTRACE_{GETREGS,GETFPREGS,SETREGS, DETACH,ATTACH,THREAD_AREA,KILL,CONT,POKEDATA,PEEKDATA}

GDB/LLDB DWARF comp unit

```c
int main() {
   long a = 3;
   long b = 2;
   long c = a + b;
   a = 4;
}
```

```
$ llvm-dwarfdump ./bin
```

GDB/LLDB DWARF comp unit

DW_TAG_compile_unit/DW_TAG_cu

DW_AT_producer	clang version 10.0.0 (tags/RELEASE_10 final)
DW_AT_language	DW_LANG_C_plus_plus
DW_AT_name	/home/danlark/gdb/examples/var.cc
DW_AT_stmt_list	0x00000000
DW_AT_comp_dir	/home/danlark/gdb/examples/build
DW_AT_low_pc	0x00400670
DW_AT_high_pc	0x0040069c

GDB/LLDB DWARF, dwarfdump

```
.debug_line: line number info for a single cu

Source lines (from CU-DIE at .debug_info offset 0x0000000b):

        NS new statement, BB new basic block, ET end of text sequence

        PE prologue end, EB epilogue begin

        IS=val ISA number, DI=val discriminator value

<pc>       [lno,col] NS BB ET PE EB IS= DI= uri: "filepath"
```

GDB/LLDB DWARF

- 0x00400670 [1, 0] NS uri: "/home/danlark/gdb/examples/var.cc"
- 0x00400676 [2,10] NS PE
- 0x0040067e [3,10] NS
- 0x00400686 [4,14] NS
- 0x0040068a [4,16]
- 0x0040068e [4,10]
- 0x00400692 [5, 7] NS
- 0x0040069a [6, 1] NS
- 0x0040069c [6, 1] NS ET

Guaranteed to work
but not obligatory

Must have
for DWARF

GDB/LLDB

```
$ g++/clang++ -O0 -g src.cpp -o src

$ gdb/lldb src

$ gdb/lldb --args src args...?

$ gdb/lldb --pid <pid>
```

GDB/LLDB

```
$ run <args…> (if gdb was execed without)

$ start <args…> (goes to main)

$ kill
```

GDB/LLDB common operations

- Breakpoints
- Stepping
- Expression evaluation
- Backtrace view
- Surrounding info view
- Core dumps

GDB/LLDB breakpoints

```
$ break file.cc:10

$ b main

$ b *0x00400670
```

GDB/LLDB DWARF

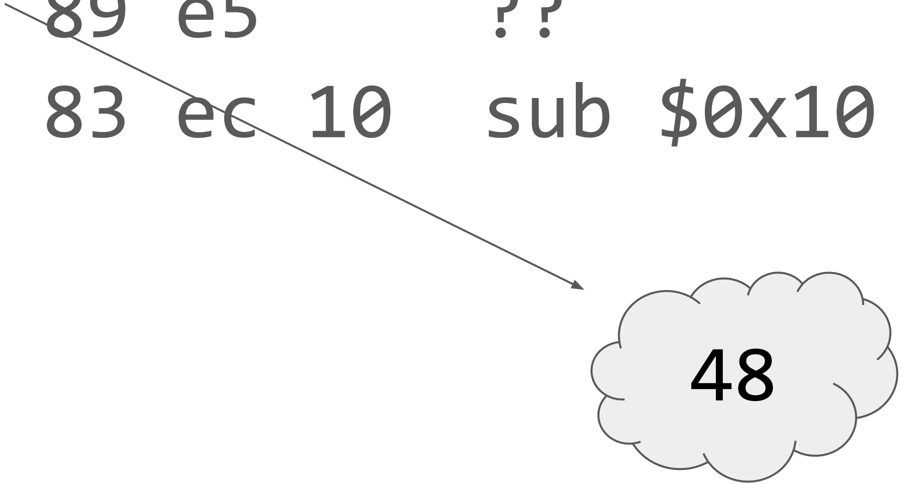
- 0x00400670 [1, 0] NS uri: "/home/danlark/gdb/examples/var.cc"
- 0x00400676 [2,10] NS PE
- 0x0040067e [3,10] NS
- 0x00400686 [4,14] NS   ← **4th line of var.cc**
- 0x0040068a [4,16]
- 0x0040068e [4,10]
- 0x00400692 [5, 7] NS
- 0x0040069a [6, 1] NS
- 0x0040069c [6, 1] NS ET

GDB/LLDB breakpoints

```
55              push %rbp
48 89 e5        mov %rsp, %rsp  ←
48 83 ec 10     sub $0x10 %rsp
```

GDB/LLDB breakpoints

```
55           push %rbp
?? 89 e5     ??
48 83 ec 10  sub $0x10 %rsp
```



GDB/LLDB breakpoints

```
55               push %rbp
cc 89 e5         int3
48 83 ec 10      sub $0x10 %rsp
```

GDB/LLDB breakpoints

```
dotraplinkage void notrace do_int3(struct pt_regs *regs, long errc)

{

…

do_trap(X86_TRAP_BP, SIGTRAP, "int3", regs, errc, NULL);

…

}
```

Get signal, return to debugger, change the instruction, get back to normal execution

GDB/LLDB DWARF comp unit

$ b main

DW_TAG_subprogram (mainly functions)

DW_AT_frame_base	DW_OP_reg6 (frame pointer register)
DW_AT_name	main
DW_AT_low_pc	0x00400670 ← **break here (almost)**
DW_AT_high_pc	0x0040069c
…	

Problems with inlining, subprograms also save inline func info.
Sometimes can be lost. gdb-add-index and **-Wl,--gdb-index** speed up this.

GDB/LLDB DWARF

- 0x00400670 [1, 0] NS uri: "/home/danlark/gdb/examples/var.cc"
- 0x00400676 [2,10] NS PE
- 0x0040067e [3,10] NS ← **Break here, upper is prologue**
- 0x00400686 [4,14] NS
- 0x0040068a [4,16]
- 0x0040068e [4,10]
- 0x00400692 [5, 7] NS
- 0x0040069a [6, 1] NS
- 0x0040069c [6, 1] NS ET

GDB/LLDB DWARF variable

DW_TAG_variable

DW_AT_location	DW_OP_fbreg -8
DW_AT_name	“a”

Conditional breakpoints:

```
$ b file.cc:12 if varname==100
```

Checks location and varname. Optimizations might elide the location.

Clang tries to save debug info, GCC is way more aggressive

GDB/LLDB breakpoints misc

- `$ info b`
- `$ delete <breakpoint#>`
- `$ rbreak <breakpoint regex>`
- `$ tbreak (temp breakpoint)`
- `$ enable/disable <breakpoint#>`
- `$ condition <breakpoint#> <condition>`

GDB/LLDB breakpoints misc

- `$ commands <breakpoint#>`
    - Provides the instructions what to do after the breakpoint hit
- Example:
    - `$ commands 1`
        - `set var a=10`
        - `bt`
        - `continue`

GDB/LLDB breakpoints misc

```
$ watch <memory location/variable>

$ rwatch <memory location/variable>

$ catch (throw, exec, fork, etc)
```

GDB/LLDB stepping

```
$ step (or ‘s’) (dive into functions)
```

```
$ next (or ‘n’) (don’t dive)
```

```
$ jump [line,*address]
```

Type Enter to **repeat** the command

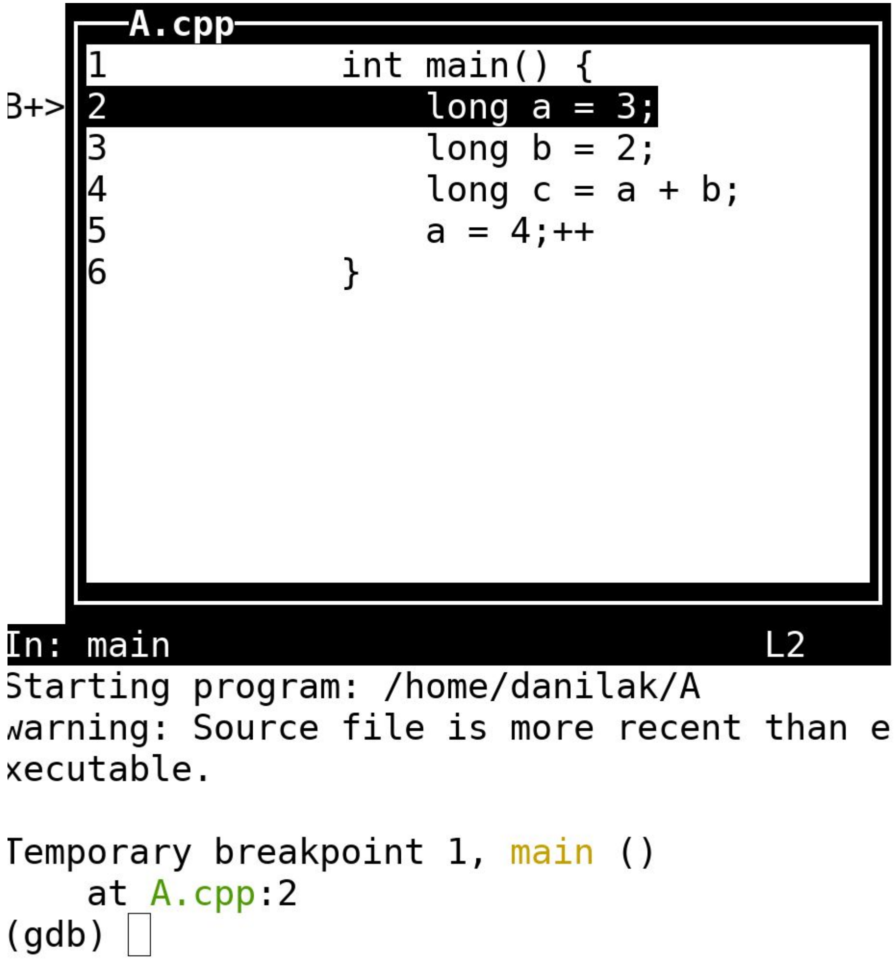
```
$ finish/f (continue until returns)
```

```
$ continue/c
```

GDB/LLDB stepping

Ctrl+X+a (enter UI)

Ctrl+X+a (exit this)

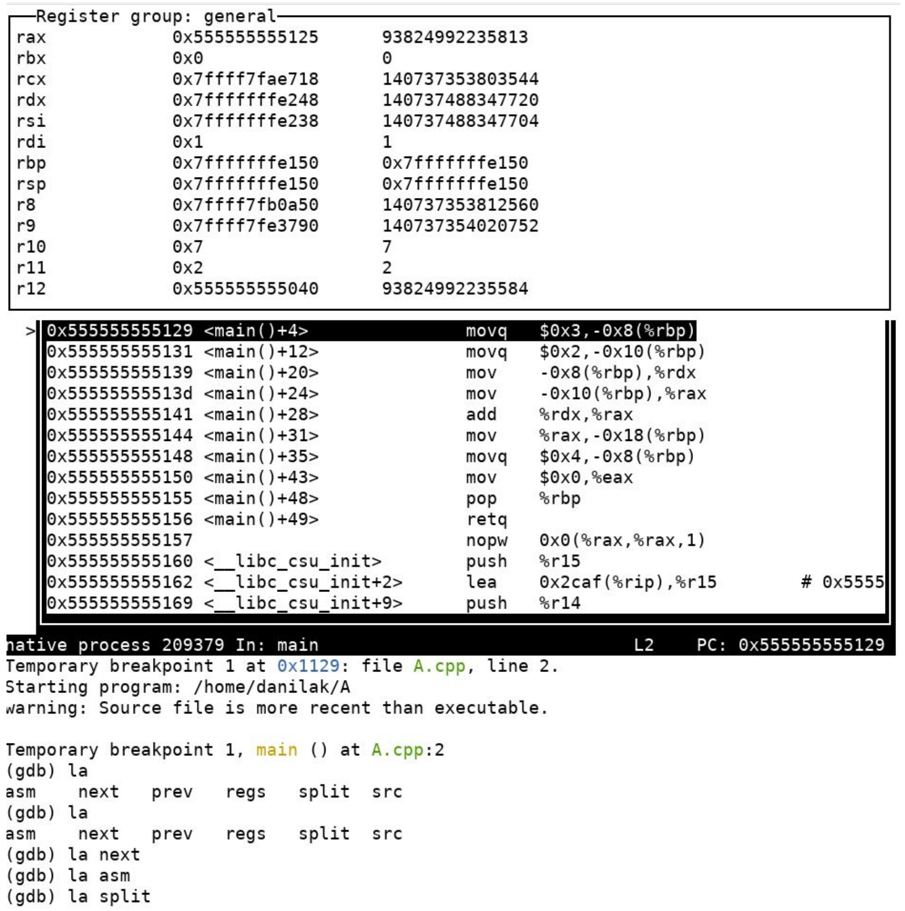


```
   ┌─A.cpp──────────────────────────────────────────┐
   │1           int main() {                         │
B+>│2                   long a = 3;                 │
   │3                   long b = 2;                 │
   │4                   long c = a + b;             │
   │5                   a = 4;++                    │
   │6           }                                   │
   └────────────────────────────────────────────────┘
In: main                                          L2
Starting program: /home/danilak/A
warning: Source file is more recent than e
xecutable.

Temporary breakpoint 1, main ()
    at A.cpp:2
(gdb) 
```

GDB/LLDB stepping

la [asm,regs,split,src]

ASM, registers, multiple at the same time and source code



```
Register group: general
rax            0x555555555125      93824992235813
rbx            0x0                 0
rcx            0x7ffff7fae718      140737353803544
rdx            0x7fffffffe248      140737488347720
rsi            0x7fffffffe238      140737488347704
rdi            0x1                 1
rbp            0x7fffffffe150      0x7fffffffe150
rsp            0x7fffffffe150      0x7fffffffe150
r8             0x7ffff7fb0a50      140737353812560
r9             0x7ffff7fe3790      140737354020752
r10            0x7                 7
r11            0x2                 2
r12            0x555555555040      93824992235584

 > 0x555555555129 <main()+4>            movq   $0x3,-0x8(%rbp)
   0x555555555131 <main()+12>           movq   $0x2,-0x10(%rbp)
   0x555555555139 <main()+20>           mov    -0x8(%rbp),%rdx
   0x55555555513d <main()+24>           mov    -0x10(%rbp),%rax
   0x555555555141 <main()+28>           add    %rdx,%rax
   0x555555555144 <main()+31>           mov    %rax,-0x18(%rbp)
   0x555555555148 <main()+35>           movq   $0x4,-0x8(%rbp)
   0x555555555150 <main()+43>           mov    $0x0,%eax
   0x555555555155 <main()+48>           pop    %rbp
   0x555555555156 <main()+49>           retq
   0x555555555157                       nopw   0x0(%rax,%rax,1)
   0x555555555160 <__libc_csu_init>     push   %r15
   0x555555555162 <__libc_csu_init+2>   lea    0x2caf(%rip),%r15        # 0x5555
   0x555555555169 <__libc_csu_init+9>   push   %r14

native process 209379 In: main                                L2    PC: 0x555555555129
Temporary breakpoint 1 at 0x1129: file A.cpp, line 2.
Starting program: /home/danilak/A
warning: Source file is more recent than executable.

Temporary breakpoint 1, main () at A.cpp:2
(gdb) la
asm    next   prev   regs   split  src
(gdb) la
asm    next   prev   regs   split  src
(gdb) la next
(gdb) la asm
(gdb) la split
```

GDB/LLDB expressions

```
$ p a * 1000

$ set var a=10

$ set $rbp=0x10

$ p ((std::pair<int, int>*)0x123183)->first

$ foo()
```

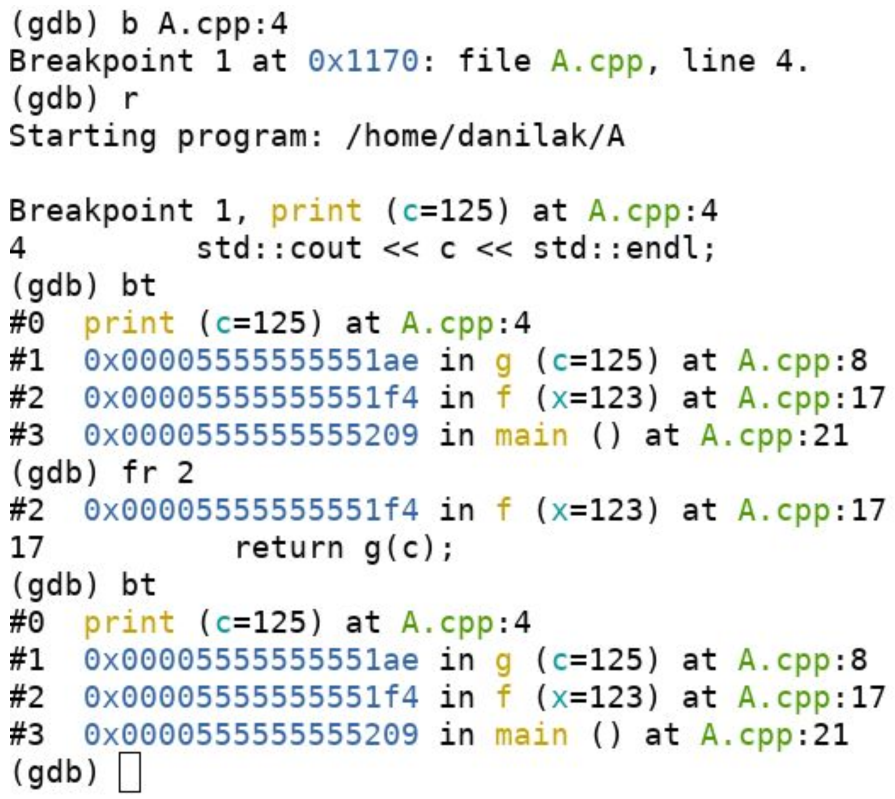
Any C/(some of) C++ expressions

LLDB expressions

- Creates LLVM IR for something complicated, then interprets it :)
- Some problems with inlining and vars might be in different locations. Printing might be slow

GDB/LLDB backtrace

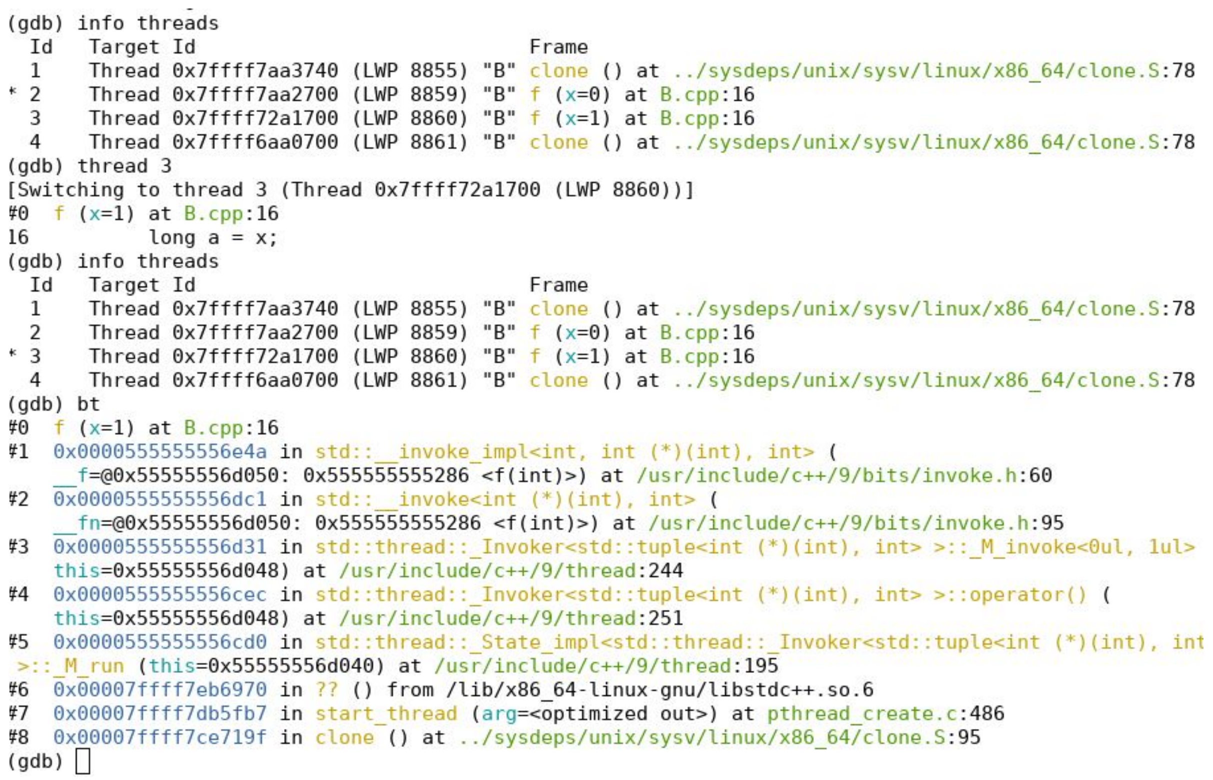
```
$ bt

$ fr <#>

$ up

$ down
```



```
(gdb) b A.cpp:4
Breakpoint 1 at 0x1170: file A.cpp, line 4.
(gdb) r
Starting program: /home/danilak/A

Breakpoint 1, print (c=125) at A.cpp:4
4           std::cout << c << std::endl;
(gdb) bt
#0  print (c=125) at A.cpp:4
#1  0x00005555555551ae in g (c=125) at A.cpp:8
#2  0x00005555555551f4 in f (x=123) at A.cpp:17
#3  0x0000555555555209 in main () at A.cpp:21
(gdb) fr 2
#2  0x00005555555551f4 in f (x=123) at A.cpp:17
17              return g(c);
(gdb) bt
#0  print (c=125) at A.cpp:4
#1  0x00005555555551ae in g (c=125) at A.cpp:8
#2  0x00005555555551f4 in f (x=123) at A.cpp:17
#3  0x0000555555555209 in main () at A.cpp:21
(gdb) 
```

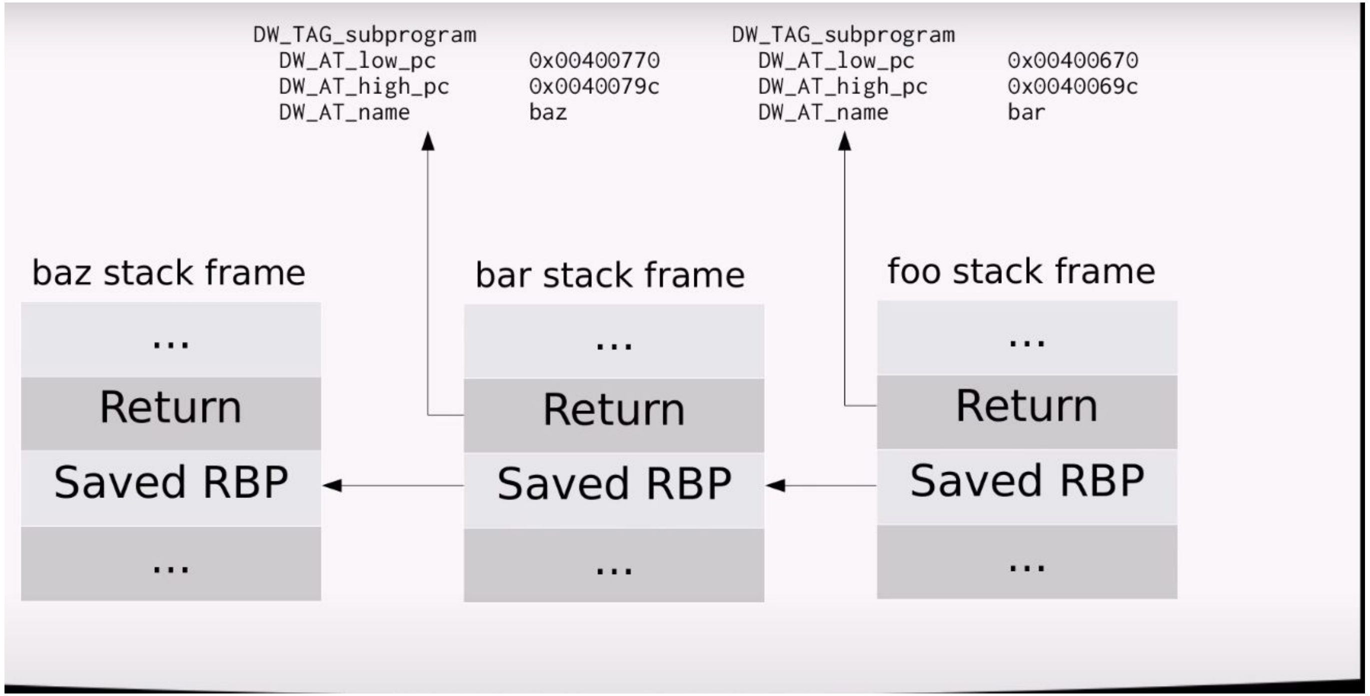
GDB/LLDB threads

```
$ info threads

  $ thread <#>
```



```
(gdb) info threads
  Id   Target Id                                   Frame
  1    Thread 0x7ffff7aa3740 (LWP 8855) "B" clone () at ../sysdeps/unix/sysv/linux/x86_64/clone.S:78
* 2    Thread 0x7ffff7aa2700 (LWP 8859) "B" f (x=0) at B.cpp:16
  3    Thread 0x7ffff72a1700 (LWP 8860) "B" f (x=1) at B.cpp:16
  4    Thread 0x7ffff6aa0700 (LWP 8861) "B" clone () at ../sysdeps/unix/sysv/linux/x86_64/clone.S:78
(gdb) thread 3
[Switching to thread 3 (Thread 0x7ffff72a1700 (LWP 8860))]
#0  f (x=1) at B.cpp:16
16          long a = x;
(gdb) info threads
  Id   Target Id                                   Frame
  1    Thread 0x7ffff7aa3740 (LWP 8855) "B" clone () at ../sysdeps/unix/sysv/linux/x86_64/clone.S:78
  2    Thread 0x7ffff7aa2700 (LWP 8859) "B" f (x=0) at B.cpp:16
* 3    Thread 0x7ffff72a1700 (LWP 8860) "B" f (x=1) at B.cpp:16
  4    Thread 0x7ffff6aa0700 (LWP 8861) "B" clone () at ../sysdeps/unix/sysv/linux/x86_64/clone.S:78
(gdb) bt
#0  f (x=1) at B.cpp:16
#1  0x0000555555556e4a in std::__invoke_impl<int, int (*)(int), int> (
    __f=@0x55555556d050: 0x555555555286 <f(int)>) at /usr/include/c++/9/bits/invoke.h:60
#2  0x0000555555556dc1 in std::__invoke<int (*)(int), int> (
    __fn=@0x55555556d050: 0x555555555286 <f(int)>) at /usr/include/c++/9/bits/invoke.h:95
#3  0x0000555555556d31 in std::thread::_Invoker<std::tuple<int (*)(int), int> >::_M_invoke<0ul, 1ul>
    this=0x55555556d048) at /usr/include/c++/9/thread:244
#4  0x0000555555556cec in std::thread::_Invoker<std::tuple<int (*)(int), int> >::operator() (
    this=0x55555556d048) at /usr/include/c++/9/thread:251
#5  0x0000555555556cd0 in std::thread::_State_impl<std::thread::_Invoker<std::tuple<int (*)(int), int
 >::_M_run (this=0x55555556d040) at /usr/include/c++/9/thread:195
#6  0x00007ffff7eb6970 in ?? () from /lib/x86_64-linux-gnu/libstdc++.so.6
#7  0x00007ffff7db5fb7 in start_thread (arg=<optimized out>) at pthread_create.c:486
#8  0x00007ffff7ce719f in clone () at ../sysdeps/unix/sysv/linux/x86_64/clone.S:95
(gdb) 
```

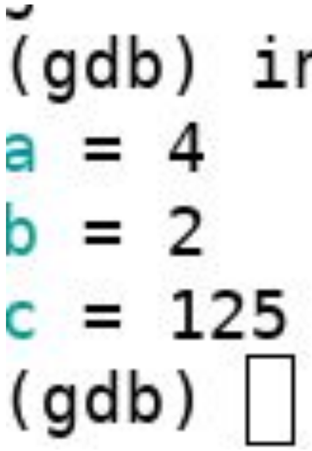
GDB/LLDB backtrace



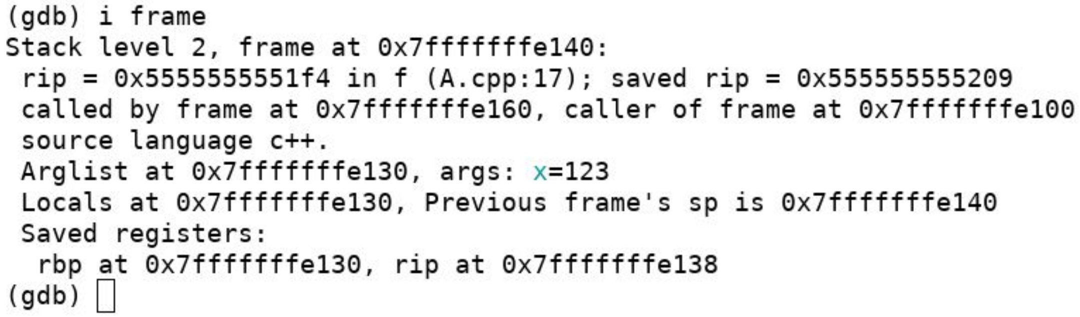
Optimizers might use RBP for various reasons. To disable them, use -fno-omit-frame-pointer or -O0

GDB/LLDB surround info

- `info r`
- `i frame`
- `info locals`
- `info all-reg`
- `display var`
- `d $reg`
- `info b`



```
(gdb) info locals
a = 4
b = 2
c = 125
(gdb) 
```



```
(gdb) i frame
Stack level 2, frame at 0x7fffffffe140:
 rip = 0x5555555551f4 in f (A.cpp:17); saved rip = 0x555555555209
 called by frame at 0x7fffffffe160, caller of frame at 0x7fffffffe100
 source language c++.
 Arglist at 0x7fffffffe130, args: x=123
 Locals at 0x7fffffffe130, Previous frame's sp is 0x7fffffffe140
 Saved registers:
  rbp at 0x7fffffffe130, rip at 0x7fffffffe138
(gdb) 
```

GDB/LLDB cores

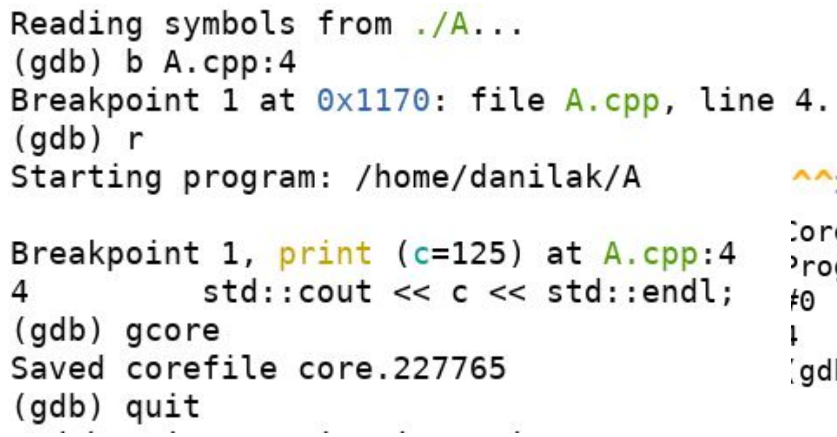
- When the program segfaults, it can a core dump
- Core dump is a memory footprint of the stackframe
- **You can read the memory but cannot run and modify**
- **You must have the exact executable that produced the core**

GDB/LLDB cores

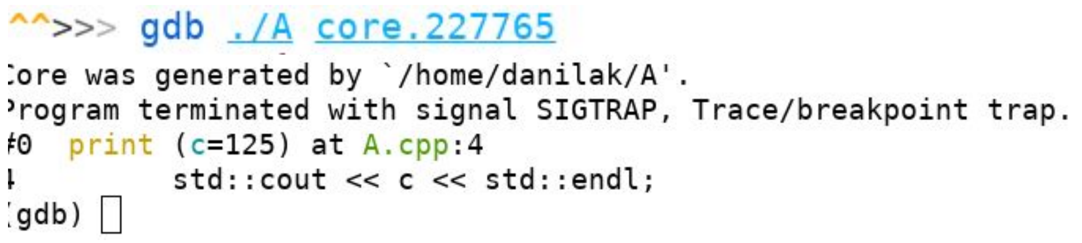
```
$ gdb/lldb src coredump
```

You can generate coredump inside the gdb

```
$ (gdb) gcore filename
```



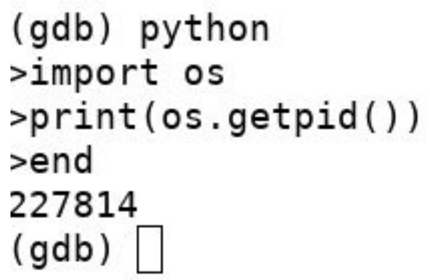
```
Reading symbols from ./A...
(gdb) b A.cpp:4
Breakpoint 1 at 0x1170: file A.cpp, line 4.
(gdb) r
Starting program: /home/danilak/A

Breakpoint 1, print (c=125) at A.cpp:4
4           std::cout << c << std::endl;
(gdb) gcore
Saved corefile core.227765
(gdb) quit
```



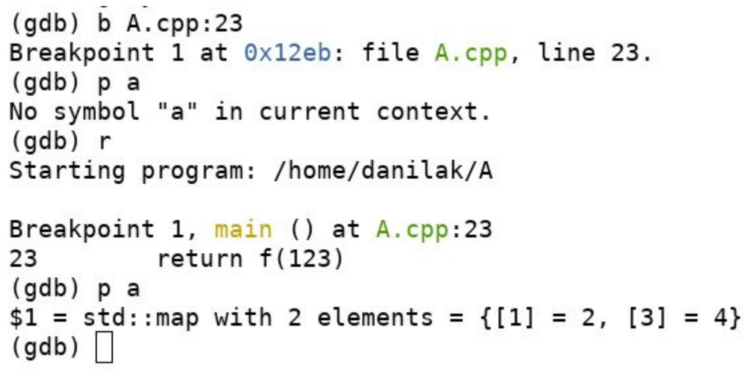
```
^^>>> gdb ./A core.227765
ore was generated by `/home/danilak/A'.
rogram terminated with signal SIGTRAP, Trace/breakpoint trap.
0  print (c=125) at A.cpp:4
          std::cout << c << std::endl;
gdb)
```

GDB/LLDB pretty printers

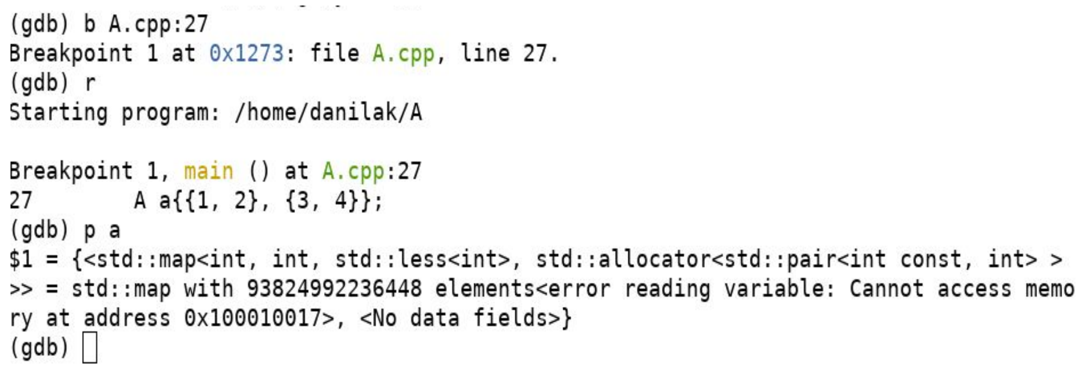
- GDB has Python interpreter inside
  - $ python
- Pretty printers are python modules



```
(gdb) python
>import os
>print(os.getpid())
>end
227814
(gdb)
```



```
(gdb) b A.cpp:23
Breakpoint 1 at 0x12eb: file A.cpp, line 23.
(gdb) p a
No symbol "a" in current context.
(gdb) r
Starting program: /home/danilak/A

Breakpoint 1, main () at A.cpp:23
23          return f(123)
(gdb) p a
$1 = std::map with 2 elements = {[1] = 2, [3] = 4}
(gdb)
```



```
(gdb) b A.cpp:27
Breakpoint 1 at 0x1273: file A.cpp, line 27.
(gdb) r
Starting program: /home/danilak/A

Breakpoint 1, main () at A.cpp:27
27          A a{{1, 2}, {3, 4}};
(gdb) p a
$1 = {<std::map<int, int, std::less<int>, std::allocator<std::pair<int const, int> >
>> = std::map with 93824992236448 elements<error reading variable: Cannot access memo
ry at address 0x100010017>, <No data fields>}
(gdb)
```

GDB/LLDB pipeline

- gdb/lldb src [coredump]
  - breakpoint something, possibly conditional
  - $ run
  - $ bt
  - $ next, step
  - $ print, display
  - iterate

GDB/LLDB homework

- You will be given binaries and core dumps
- Need to extract some flag from it given the conditions
- We tried hard to allow you to try out many things and play around

GDB/LLDB links

- GDB cheat sheet
- GDB maintainer talks
- Write your own debugger
- GDB automation
- How GDB/LLDB works.