Debugging of GDB debugger

Goal

Compile debug version of GDB (*-g* option) and use system *gdb* to debug compiled one. In case system *gdb* isn't present, either compile release version (without *-g* option) of *gdb* or simly install it from software center. Debug version should run *test* program whose source code is:

```
#include <stdio.h>

int main()
{
        int a = 5;
        int *pa = &a;
        printf("'a' is at address '%p'.\n'a' has value '%d'\n", pa, *pa);

        return 0;
}
```

Compile *test* using:

```
gcc -g test.c -o test
```

Compilation of GDB

Let's suppose you want to build *gdb* in *$HOME/builds* folder.

```
cd "$HOME"
# The following two lines should be uncommented just in case you haven't
# already downloaded gdb repo
git clone 'git://sourceware.org/git/binutils-gdb.git'
cp -r binutils-gdb binutils-gdb-backup
cd binutils-gdb
mkdir build && cd build
../configure --enable-tui --enable-source-highlight --with-python=python3 \
 --prefix="$HOME/builds"
make -j$(nproc) >build.log
echo $?
```

If build succeeds, 0 should be printed as the last line in terminal. If it fails, you should find error in *build.log* file, which can be identified using:

```
grep -C4 -i 'error' build.log
```

The most common error is missing dependencies in system. After you install them, retry build process:

```
cd $HOME
rm -rf binutils-gdb
cp -r binutils-gdb-backup binutils-gdb
cd binutils-gdb
mkdir build && cd build
```

```
../configure --enable-tui --enable-source-highlight --with-python=python4 \
--prefix="$HOME/builds"
make -j$(nproc) >build.log
echo $?
```

Repeat above procedure until you get 0 exit status. After you succeed, in *build/gdb* will be present *gdb* executable. Now, before you install it inside aforementioned *$HOME/builds* directory, check whether you got *debug* or *release* version of *gdb*.

Debug / Release

Difference between *debug* and *release* version of executable is that *debug* version retains useful information for debugger, but *release* executable is optimized by compiler and much of these information is stripped. That being said, if you want to debug your program by some debugger, you should always compile it as *debug* version. To find out if executable is *debug* or *release* version, you may use one of the following commands:

```
file <executable file> | grep 'with .debug_info'
echo $?

objdump -Wi <executable file> | grep '.debug_info' -m 1
echo $?
```

Output is 0 if it's *debug* version, and some other number if it's *release*. In case you compiled *release* version, you should include *-g* option in *Makefile* or *configure* script and repeat build process. For more info, call *./configure –help* or consult official gdb documentation.

Install GDB

After build process completes, *gdb* can be installed:

```
cd "$HOME/binutils-gdb/build" # set current working dir to build folder
make install                  # install compiled gdb to specified --prefix folder
```

Note that we didn't use *sudo make install* because we don't need admin privilleges to acces subdir of *$HOME* folder. Another two reasons to use *–prefix* option are to prevent conflict with system *gdb* in folders like */usr/bin*, */usr/local/bin*, */bin*... and conflicts with other users on the same computer.

Running GDB by another GDB instance

Let */usr/bin/gdb* be path to system *gdb* - we will call it *gdb-release*. Let $*HOME/builds/gdb/bin/gdb* be path to just built *gdb* - we will call it *gdb-debug*. On my system, version of *gdb-release* is 9.2 and version of *gdb-debug* is 13.0.50. Let's open *gdb-debug* using *gdb-release*:

```
/usr/bin/gdb $HOME/builds/gdb/bin/gdb
```

By using *show version* we can find out whether we are in *gdb-release* or *gdb-debug* debugger:

```
(gdb) show version
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.1) 9.2
Copyright (C) 2020 Free Software Foundation...
```

which means we are in *gdb-release*. Put breakpoint on function *value_as_address(struct value *value)* which can be found in source code of *gdb-debug*.

```
(gdb) b value_as_address
Breakpoint 1 at 0x5646c0: file ../../gdb/value.c, line 2757
```

Make sure we are still in *gdb-release* debugger:

```
(gdb) show version
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.1) 9.2 ...
```

Load symbols from *test* executable:

```
(gdb) run ~/gdb-test/test # Load symbols from 'test' executable
Starting program: /home/syrmia/bin/gdb/bin/gdb ~/gdb-test/test
[Thread debugging using libthread_db enabled]
[Detaching after vfork from child process 6043]
[New Thread 0x7ffff49fd700 (LWP 6044)] ...
GNU gdb (GDB) 13.0.50.20220815-git
```

Note that *gdb 13.0.50* i.e. *gdb-debug* took control of the terminal. That's because *run* command 'starts debugged program'. Here, since *gdb-release* opened *gdb-debug*, *gdb-debug* is *debugged program*. When we say *run [args]*, *gdb-debug* is being run. Let's find out where are breakpoints placed:

```
(gdb) info b
No breakpoints or watchpoints.
```

We don't see any breakpoints because *gdb-release* is aware of breakpoints set on *gdb-debug*(on function *value_as_address*) and *gdb-debug* is aware of breakpoints set on *test* (no breakpoints yet). Let us see *test* source code:

```
(gdb) list
1   #include <stdio.h>
2
3       int main()
4       {
5               int a = 5;
6               int *pa = &a;
7               printf("'a' is at address '%p'.\n'a' has value '%d'\n", pa, *pa);
8
9               return 0;
10       }
```

and let's put one breakpoint on the very start of our main function and run the *test*:

```
(gdb) b 3
Breakpoint 1 at 0x1175: file test.c, line 4
(gdb) run
Starting program: /home/syrmia/gdb-test/test
[Detaching after vfork from child process 15624]
[Detaching after fork from child process 15625]
...
Thread 1 "gdb" hit Breakpoint 1, value_as_address (val=0x55555622e3b0) at
../../gdb/value.c:2757
(gdb) show version
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.1) 9.2 ...
```

This means we are now in *gdb-release* on line *value_as_address* which is set on *gdb-debug* executable. Before *test* arrived to its breakpoint, *value_as_address* has been called. Let's inspect stack trace of *gdb-debug* process:

```
(gdb) bt
#0  value_as_address (val=0x55555622e3b0) at ../../gdb/value.c:2757
#1  0x00005555559c7cc6 in svr4_handle_solib_event () at ../../gdb/solib-svr4.c:1838
#2  0x00005555559cd7f0 in handle_solib_event () at ../../gdb/solib.c:1338
#3  0x00005555556f4865 in bpstat_stop_status (aspace=<optimized out>,
    bp_addr=bp_addr@entry=140737353955253, thread=thread@entry=0x555556174570, ws=...,
    stop_chain=stop_chain@entry=0x0) at ../../gdb/breakpoint.c:5558
#4  0x000055555587faac in handle_signal_stop (ecs=0x7fffffffdd30)
    at ../../gdb/regcache.h:344
#5  0x000055555588209c in handle_inferior_event (ecs=<optimized out>)
    at ../../gdb/infrun.c:5869
#6  0x00005555558831fb in fetch_inferior_event () at ../../gdb/infrun.c:4233
```

```
#7  0x0000555555bc19c6 in gdb_wait_for_event (block=block@entry=0) at
    ../../gdbsupport/event-loop.cc:670
#8  0x0000555555bc1c86 in gdb_wait_for_event (block=0) at
    ../../gdbsupport/event-loop.cc:569
#9  gdb_do_one_event () at ../../gdbsupport/event-loop.cc:210
#10 0x00005555558c8b55 in start_event_loop () at ../../gdb/main.c:411
#11 captured_command_loop () at ../../gdb/main.c:471
#12 0x00005555558ca725 in captured_main (data=<optimized out>) at
    ../../gdb/main.c:1329
#13 gdb_main (args=<optimized out>) at ../../gdb/main.c:1344
#14 0x000055555565ccd0 in main (argc=<optimized out>, argv=<optimized out>)
    at ../../gdb/gdb.c:32
```

What we knew until now is that content of pointer is address. But, do we need to perform some operations (e.g. shift) on these bytes in order to get usable address for specific architecture? Answer is **yes**, as can be read in 'Pointers Are Not Always Adresses' chapter of this document. That is purpose of *value_as_address* function. There is also a concept of *tagged pointer* which except address can store additional data. Continuing program execution:

```
(gdb) continue
Continuing.

Breakpoint 1, main () at test.c:4
4       {
```

Only now *test's* breakpoint is being hit. If we continue one more time, *test* program will regularly finish it's execution. Also, we see that *gdb-debug* instance stays active:

```
(gdb) c
Continuing.
'a' is at address '0x7fffffffdf3c'.
'a' has value '5'
[Inferior 1 (process 16797) exited normally]
(gdb) show version
GNU gdb (GDB) 13.0.50.20220815-git
```

We found out that even if we don't access any pointer explicitly in *test* code, *value_as_address* in being called. The same function is invoked when we want to *print* some variable from interactive debugger, as the following example describes:

```
# Run gdb-debug using gdb-release
(bash) /usr/bin/gdb $HOME/builds/gdb/bin/gdb

# Make sure we are in gdb-release
(gdb) show version
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.1) 9.2

# Put breakpoint in 'gdb-debug' exe on 'value_as_address' function
(gdb) b value_as_address
Breakpoint 1 at 0x5646c0: file ../../gdb/value.c, line 2757

# Make sure we are still in gdb-release
(gdb) show version
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.1) 9.2 ...

# Load symbols from 'test' executable in 'gdb-debug'
(gdb) run ~/gdb-test/test
Starting program: /home/syrmia/bin/gdb/bin/gdb ~/gdb-vezba/test
[Thread debugging using libthread_db enabled]
[Detaching after vfork from child process 6043]
[New Thread 0x7ffff49fd700 (LWP 6044)] ...
```

```
GNU gdb (GDB) 13.0.50.20220815-git

# Display code that is about to be run
(gdb) list
1       #include <stdio.h>
2
3       int main()
4       {
5               int a = 5;
6               int *pa = &a;
7               printf("'a' is at address '%p'.\n'a' has value '%d'\n", pa, *pa);
8
9               return 0;
10       }

# Put 3 breakpoints in 'test' executable
(gdb) b 3
Breakpoint 1 at 0x1175: file test.c, line 4.
(gdb) b 7
Breakpoint 2 at 0x1193: file test.c, line 7.
(gdb) b 9
Breakpoint 3 at 0x11b1: file test.c, line 9.

# Make sure breakpoints are set
(gdb) info b
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x0000000000001175 in main at test.c:4
2       breakpoint     keep y   0x0000000000001193 in main at test.c:7
3       breakpoint     keep y   0x00000000000011b1 in main at test.c:9

# Start 'test'
(gdb) run
Starting program: /home/syrmia/gdb-test/test
[Detaching after vfork from child process 17597]
[Detaching after fork from child process 17598]
[Detaching after fork from child process 17599]

Thread 1 "gdb" hit Breakpoint 1, value_as_address (val=0x555556225a00)
at ../../gdb/value.c:2757
2757        {

# Finish value_as_address function and continue execution until next breakpoint
(gdb) c
Continuing.

Breakpoint 1, main () at test.c:4
4       {

# Continue to next breakpoint
(gdb) c
Continuing.

Breakpoint 2, main () at test.c:7
7               printf("'a' is at address '%p'.\n'a' has value '%d'\n", pa, *pa);

# Recall 'test' source code to see if 'pa' pointer is initialized
(gdb) list -5
1       #include <stdio.h>
```

```
2
3       int main()
4       {
5               int a = 5;
6               int *pa = &a;
7               printf("'a' is at address '%p'.\n'a' has value '%d'\n", pa, *pa);
8
9               return 0;
10       }
```

```
# Print pointer 'pa'. Note that 'value_as_address' will be hit 3 times
(gdb) print pa

Thread 1 "gdb" hit Breakpoint 1, value_as_address (val=0x555556225a00)
at ../../gdb/value.c:2757
2757        {
(gdb) c
Continuing.

Thread 1 "gdb" hit Breakpoint 1, value_as_address (val=0x555556225a00)
at ../../gdb/value.c:2757
2757        {
(gdb) c
Continuing.

Thread 1 "gdb" hit Breakpoint 1, value_as_address (val=0x5555561f6c10)
at ../../gdb/value.c:2757
2757        {
(gdb) c
Continuing.

# Now 'pa' prints its content which is address of variable a
$1 = (int *) 0x7fffffffdf3c

# Continue 'test' execution
(gdb) c
Continuing.
'a' is at address '0x7fffffffdf3c'.
'a' has value '5'

Breakpoint 3, main () at test.c:9
9               return 0;

# Finish 'test'
(gdb) c
Continuing.
[Inferior 1 (process 17597) exited normally]
```

Function value_as_address(struct value *value)

The following examples examine *gdb's* stack trace when breakpoint is set on function *value_as_address*. After each example is some conclusion.

Example 1

```
(gdb) run
Starting program: /home/syrmia/gdb-test/test
```

```
[Detaching after vfork from child process 21164]
[Detaching after fork from child process 21165]
[Detaching after fork from child process 21166]

Thread 1 "gdb" hit Breakpoint 1, value_as_address (val=0x555556228300)
at ../../gdb/value.c:2757
2757        {

# Inspect $val address
(gdb) x 0x555556228300
0x555556228300:        0x00000000

# Proceed until 'value_as_address' returns
(gdb) finish
Run till exit from #0  value_as_address (val=0x555556228300) at
../../gdb/value.c:2757
0x00005555559c7cc6 in svr4_handle_solib_event () at ../../gdb/solib-svr4.c:1838
1838            debug_base = value_as_address (val);
Value returned is $1 = 140737354129760

# Inspect return value from 'value_as_address' call
(gdb) x $1
0x7ffff7ffe160 <_r_debug>:        0x00000001   # So return value
# is address of '_r_debug' function

# Print backtrace
(gdb) bt
#0  0x00005555559c7cc6 in svr4_handle_solib_event () at
    ../../gdb/solib-svr4.c:1838
#1  0x00005555559cd7f0 in handle_solib_event () at ../../gdb/solib.c:1338
#2  0x00005555556f4865 in bpstat_stop_status (aspace=<optimized out>,
    bp_addr=bp_addr@entry=140737353955253,
    thread=thread@entry=0x5555561745e0, ws=..., stop_chain=stop_chain@entry=0x0)
    at ../../gdb/breakpoint.c:5558
#3  0x000055555587faac in handle_signal_stop (ecs=0x7fffffffdd30) at
    ../../gdb/regcache.h:344
#4  0x000055555588209c in handle_inferior_event (ecs=<optimized out>)
    at ../../gdb/infrun.c:5869
#5  0x00005555558831fb in fetch_inferior_event () at ../../gdb/infrun.c:4233
#6  0x0000555555bc19c6 in gdb_wait_for_event (block=block@entry=0) at
    ../../gdbsupport/event-loop.cc:670
#7  0x0000555555bc1c86 in gdb_wait_for_event (block=0) at
    ../../gdbsupport/event-loop.cc:569
#8  gdb_do_one_event () at ../../gdbsupport/event-loop.cc:210
#9  0x00005555558c8b55 in start_event_loop () at ../../gdb/main.c:411
#10 captured_command_loop () at ../../gdb/main.c:471
#11 0x00005555558ca725 in captured_main (data=<optimized out>) at
    ../../gdb/main.c:1329
#12 gdb_main (args=<optimized out>) at ../../gdb/main.c:1344
#13 0x000055555565ccd0 in main (argc=<optimized out>, argv=<optimized out>) at
    ../../gdb/gdb.c:32

# Continue execution until next breakpoint
(gdb) c
Continuing.
Address of main is 0x555555555169

Breakpoint 1, main () at test.c:8
```

```
8               printf("'a' is at address '%p'.\n'a' has value '%d'\n", pa, *pa);

# 'a' and 'pa' are initialized. Let's check content of 'a' variable
# Note that 'value_as_address' will be hit 3 times with the same
# $val parameter 2 times and getting different return value each time
(gdb) print a
Thread 1 "gdb" hit Breakpoint 1, value_as_address (val=0x55555622d630) at
../../gdb/value.c:2757
2757        {
(gdb) x 0x55555622d630
0x55555622d630:         0x00000002
(gdb) fin
Run till exit from #0  value_as_address (val=0x55555622d630) at
../../gdb/value.c:2757
0x0000555555826320 in address_from_register (regnum=<optimized out>,
frame=frame@entry=0x55555600eb50)
at ../../gdb/findvar.c:979
979           result = value_as_address (value);
Value returned is $2 = 140737488346928
(gdb) x $2
0x7fffffffdf30:         0x00000000
(gdb) c
Continuing.

Thread 1 "gdb" hit Breakpoint 1, value_as_address (val=0x55555622d630)
at ../../gdb/value.c:2757
2757        {
(gdb) x 0x55555622d630
0x55555622d630:         0x00000002
(gdb) fin
Run till exit from #0  value_as_address (val=0x55555622d630) at
../../gdb/value.c:2757
0x0000555555826320 in address_from_register (regnum=<optimized out>,
frame=frame@entry=0x5555564221b0)
at ../../gdb/findvar.c:979
979           result = value_as_address (value);
Value returned is $3 = 140737488346944
(gdb) x $3
0x7fffffffdf40:         0x55fa72a8
(gdb) c
Continuing.

Thread 1 "gdb" hit Breakpoint 1, value_as_address (val=0x555556231570) at
../../gdb/value.c:2757
2757        {
(gdb) x 0x555556231570
0x555556231570:         0x00000000
(gdb) fin
Run till exit from #0  value_as_address (val=0x555556231570) at
../../gdb/value.c:2757
dwarf_expr_context::fetch_result (this=this@entry=0x7fffffffd860,
type=<optimized out>, type@entry=0x5555561ee8c0,
subobj_type=subobj_type@entry=0x5555561ee8c0,
subobj_offset=subobj_offset@entry=0, as_lval=<optimized out>)
at ../../gdb/dwarf2/expr.c:1004
1004                    retval = value_at_lazy (subobj_type,
Value returned is $4 = 140737488346908
(gdb) x $4
```

```
0x7fffffffdf1c:        0x8d900db2
(gdb) c
Continuing.

# Finally, value of 'a' is being shown
$1 = 5

# Recall 'test' source code
(gdb) list
3       int main()
4       {
5               printf("Address of main is %p\n", main);
6               int a = 5;
7               int *pa = &a;
8               printf("'a' is at address '%p'.\n'a' has value '%d'\n", pa, *pa);
9
10               return 0;
11       }

# Print 'pa' pointer
(gdb) print pa
Thread 1 "gdb" hit Breakpoint 1, value_as_address (val=0x55555622d630) at
../../gdb/value.c:2757
2757        {
(gdb) fin
Run till exit from #0  value_as_address (val=0x55555622d630) at
../../gdb/value.c:2757
dwarf_expr_context::fetch_result (this=this@entry=0x7fffffffd860,
    type=<optimized out>, type@entry=0x5555561f0880,
    subobj_type=subobj_type@entry=0x5555561f0880,
    subobj_offset=subobj_offset@entry=0,
    as_lval=<optimized out>)
    at ../../gdb/dwarf2/expr.c:1004
1004                    retval = value_at_lazy (subobj_type,
Value returned is $5 = 140737488346912
(gdb) x $5
0x7fffffffdf20:        0x55c0c290
(gdb) c
Continuing.

# 'pa' pointer
$2 = (int *) 0x7fffffffdf1c

# Print 'pa' one more time
(gdb) print pa
Thread 1 "gdb" hit Breakpoint 1, value_as_address (val=0x5555562141a0)
at ../../gdb/value.c:2757
2757        {
(gdb) fin
Run till exit from #0  value_as_address (val=0x5555562141a0) at
../../gdb/value.c:2757
dwarf_expr_context::fetch_result (this=this@entry=0x7fffffffd860,
type=<optimized out>, type@entry=0x5555561f0880,
subobj_type=subobj_type@entry=0x5555561f0880,
subobj_offset=subobj_offset@entry=0, as_lval=<optimized out>)
at ../../gdb/dwarf2/expr.c:1004
1004                    retval = value_at_lazy (subobj_type,
Value returned is $6 = 140737488346912
```

```
(gdb) x $6
0x7fffffffdf20:        0x55c0c290
(gdb) c
Continuing.
$3 = (int *) 0x7fffffffdf1c

# Inspect address on which 'pa' points to
(gdb) x $3
Thread 1 "gdb" hit Breakpoint 1, value_as_address (val=0x555556215260)
at ../../gdb/value.c:2757
2757        {
(gdb) c
Continuing.

# 5, which is exactly value of 'a' while 'pa' points to it
0x7fffffffdf1c:        0x00000005
(gdb) c
Continuing.
'a' is at address '0x7fffffffdf1c'.
'a' has value '5'
[Inferior 1 (process 21164) exited normally]
```

We saw that:

- *value_as_address* breakpoint was hit before entering *main* function of *test* program.
- The same breakpoint is hit whenever we *print $some_variable*

Example 2

Run program one more time with focus on *backtrace*:

```
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.1) 9.2
Reading symbols from /home/syrmia/builds/gdb/bin/gdb...
r(gdb) b value_as_address
Breakpoint 1 at 0x5646c0: file ../../gdb/value.c, line 2757.
(gdb) r gdb-test/test
Starting program: /home/syrmia/builds/gdb/bin/gdb gdb-test/test
GNU gdb (GDB) 13.0.50.20220815-git
Reading symbols from gdb-test/test...
(gdb) list
1       #include <stdio.h>
2
3       int main()
4       {
5               printf("Address of main is %p\n", main);
6               int a = 5;
7               int *pa = &a;
8               printf("'a' is at address '%p'.\n'a' has value '%d'\n", pa, *pa);
9
10               return 0;


# Put breakpoints on 'main' and 'printf()' lines and then run program
(gdb) b main
Breakpoint 1 at 0x1175: file test.c, line 4.
(gdb) b 8
Breakpoint 2 at 0x11ab: file test.c, line 8.
(gdb) r
```

```
Starting program: /home/syrmia/gdb-test/test
[Detaching after vfork from child process 26556]...

Thread 1 "gdb" hit Breakpoint 1, value_as_address (val=0x555556228750) at
../../gdb/value.c:2757
2757        {

# We hit 'value_as_address' breakpoint before entering main function of 'test'
# Display stack trace before reaching main function of 'test' then continue
# execution
(gdb) bt
#0  value_as_address (val=0x555556228750) at ../../gdb/value.c:2757
#1  0x00005555559c7cc6 in svr4_handle_solib_event () at
    ../../gdb/solib-svr4.c:1838
#2  0x00005555559cd7f0 in handle_solib_event () at ../../gdb/solib.c:1338
#3  0x00005555556f4865 in bpstat_stop_status (aspace=<optimized out>,
    bp_addr=bp_addr@entry=140737353955253,
    thread=thread@entry=0x555556174570, ws=..., stop_chain=stop_chain@entry=0x0)
     at ../../gdb/breakpoint.c:5558
#4  0x000055555587faac in handle_signal_stop (ecs=0x7fffffffdd40) at
    ../../gdb/regcache.h:344
#5  0x000055555588209c in handle_inferior_event (ecs=<optimized out>) at
    ../../gdb/infrun.c:5869
#6  0x00005555558831fb in fetch_inferior_event () at ../../gdb/infrun.c:4233
#7  0x0000555555bc19c6 in gdb_wait_for_event (block=block@entry=0) at
    ../../gdbsupport/event-loop.cc:670
#8  0x0000555555bc1c86 in gdb_wait_for_event (block=0) at
    ../../gdbsupport/event-loop.cc:569
#9  gdb_do_one_event () at ../../gdbsupport/event-loop.cc:210
#10 0x00005555558c8b55 in start_event_loop () at ../../gdb/main.c:411
#11 captured_command_loop () at ../../gdb/main.c:471
#12 0x00005555558ca725 in captured_main (data=<optimized out>) at
    ../../gdb/main.c:1329
#13 gdb_main (args=<optimized out>) at ../../gdb/main.c:1344
#14 0x000055555565ccd0 in main (argc=<optimized out>, argv=<optimized out>)
    at ../../gdb/gdb.c:32
(gdb) c
Continuing.

Breakpoint 1, main () at test.c:4
4       {
(gdb) c
Continuing.
Address of main is 0x555555555169

Breakpoint 2, main () at test.c:8
8               printf("'a' is at address '%p'.\n'a' has value '%d'\n", pa, *pa);

# 'a' and 'pa' have been initialized.
(gdb) print a

Thread 1 "gdb" hit Breakpoint 1, value_as_address (val=0x555556226c50) at
../../gdb/value.c:2757
2757        {
(gdb) bt
#0  value_as_address (val=0x555556226c50) at ../../gdb/value.c:2757
#1  0x0000555555826320 in address_from_register (regnum=<optimized out>,
    frame=frame@entry=0x55555600eb30)
```

```
    at ../../gdb/findvar.c:979
#2  0x00005555557a229f in read_addr_from_reg (frame=frame@entry=0x55555600eb30,
    reg=<optimized out>)
    at ../../gdb/dwarf2/expr.c:86
#3  0x00005555557ab35e in dwarf2_frame_cache (this_frame=0x55555600eb30,
    this_cache=<optimized out>)
    at ../../gdb/dwarf2/frame.c:962
#4  0x00005555557ab5d7 in dwarf2_frame_this_id (this_frame=0x55555600eb30,
    this_cache=<optimized out>,
    this_id=0x55555600eb90) at ../../gdb/dwarf2/frame.c:1117
#5  0x000055555582b6e4 in compute_frame_id (fi=0x55555600eb30) at
    ../../gdb/frame.c:583
#6  0x000055555582b898 in get_frame_id (fi=fi@entry=0x55555600eb30) at
    ../../gdb/frame.c:626
#7  0x000055555582c4ff in get_prev_frame_always_1 (this_frame=0x55555600eb30)
    at ../../gdb/frame.c:2212
#8  0x000055555582c8c0 in get_prev_frame_always (this_frame=0x55555600eb30) at
    ../../gdb/frame.c:2312
#9  0x000055555582d04d in get_frame_unwind_stop_reason
    (frame=frame@entry=0x55555600eb30) at ../../gdb/frame.c:2987
#10 0x00005555557a76b5 in dwarf2_frame_cfa (this_frame=0x55555600eb30) at
    ../../gdb/dwarf2/frame.c:1356
#11 0x00005555557a4b20 in dwarf_expr_context::execute_stack_op
    (this=0x7fffffffd870,
    op_ptr=<optimized out>,
    op_end=0x5555561c8c60 " \003") at ../../gdb/dwarf2/expr.c:2110
#12 0x00005555557a5c98 in dwarf_expr_context::eval (this=0x7fffffffd870,
    addr=<optimized out>, len=<optimized out>)
    at ../../gdb/dwarf2/expr.c:1238
#13 0x00005555557a51cd in dwarf_expr_context::execute_stack_op
    (this=0x7fffffffd870, op_ptr=0x5555561c8c71 "\021pa",
    op_end=0x5555561c8c71 "\021pa") at ../../gdb/dwarf2/expr.c:1810
#14 0x00005555557a5c98 in dwarf_expr_context::eval (this=0x7fffffffd870,
    addr=<optimized out>, len=<optimized out>)
    at ../../gdb/dwarf2/expr.c:1238
#15 0x00005555557a5f46 in dwarf_expr_context::evaluate
    (this=this@entry=0x7fffffffd870,
    addr=addr@entry=0x5555561c8c6f "\221\\\021pa", len=len@entry=2,
    as_lval=as_lval@entry=true, per_cu=per_cu@entry=0x5555561b7230,
    frame=frame@entry=0x55555600eb30, addr_info=0x0, type=0x5555561ee850,
    subobj_type=0x5555561ee850, subobj_offset=0) at ../../gdb/dwarf2/expr.c:1077
#16 0x00005555557b8716 in dwarf2_evaluate_loc_desc_full (type=0x5555561ee850,
    frame=0x55555600eb30,
    data=0x5555561c8c6f "\221\\\021pa", size=2, per_cu=0x5555561b7230,
    per_objfile=<optimized out>,
    subobj_type=0x5555561ee850, subobj_byte_offset=0, as_lval=true) at
    ../../gdb/dwarf2/loc.c:1519
#17 0x00005555557b8aaa in dwarf2_evaluate_loc_desc (as_lval=true,
    per_objfile=<optimized out>,
--Type <RET> for more, q to quit, c to continue without paging--
    per_cu=<optimized out>, size=<optimized out>, data=<optimized out>,
    frame=<optimized out>, type=<optimized out>)
    at ../../gdb/dwarf2/loc.c:1563
#18 locexpr_read_variable (symbol=<optimized out>, frame=<optimized out>) at
    ../../gdb/dwarf2/loc.c:3053
#19 0x000055555582593f in language_defn::read_var_value (this=<optimized out>,
    var=0x5555561f0740,
    var_block=0x5555561f08e0, frame=0x55555600eb30) at ../../gdb/symtab.h:1232
```

```
#20 0x0000555555806b47 in evaluate_var_value (noside=EVAL_NORMAL,
    blk=<optimized out>, var=0x5555561f0740)
    at ../../gdb/eval.c:559
#21 0x0000555555805e92 in expression::evaluate (this=0x55555621d6b0,
    expect_type=0x0, noside=EVAL_NORMAL)
    at /usr/include/c++/9/bits/unique_ptr.h:360
#22 0x0000555555927fc0 in process_print_command_args (args=<optimized out>,
    print_opts=0x7fffffffdac0,
    voidprint=<optimized out>) at /usr/include/c++/9/bits/unique_ptr.h:360
#23 0x00005555559287eb in print_command_1 (args=<optimized out>, voidprint=1)
    at ../../gdb/printcmd.c:1320
#24 0x0000555555733f45 in cmd_func (cmd=<optimized out>, args=<optimized out>,
    from_tty=<optimized out>)
    at ../../gdb/cli/cli-decode.c:2516
#25 0x000055555a3c257 in execute_command (p=<optimized out>,
    p@entry=0x555555fae210 "print a", from_tty=1)
    at ../../gdb/top.c:699
#26 0x000055555580c965 in command_handler (command=0x555555fae210 "print a")
    at ../../gdb/event-top.c:598
#27 0x000055555580cd51 in command_line_handler (rl=...) at
    ../../gdb/event-top.c:842
#28 0x000055555580d4ec in gdb_rl_callback_handler (rl=0x55555621d690 "print a")
    at /usr/include/c++/9/bits/unique_ptr.h:153
#29 0x000055555ae1e78 in rl_callback_read_char () at
    ../../../readline/readline/callback.c:290
#30 0x000055555580bd86 in gdb_rl_callback_read_char_wrapper_noexcept () at
    ../../gdb/event-top.c:188
#31 0x000055555580d3a5 in gdb_rl_callback_read_char_wrapper
    (client_data=<optimized out>) at ../../gdb/event-top.c:204
#32 0x000055555580bc98 in stdin_event_handler (error=<optimized out>,
    client_data=0x555555fafa80)
    at ../../gdb/event-top.c:525
#33 0x0000555555bc19c6 in gdb_wait_for_event (block=block@entry=1) at
    ../../gdbsupport/event-loop.cc:670
#34 0x0000555555bc1c3b in gdb_wait_for_event (block=1) at
    ../../gdbsupport/event-loop.cc:569
#35 gdb_do_one_event () at ../../gdbsupport/event-loop.cc:235
#36 0x00005555558c8b55 in start_event_loop () at ../../gdb/main.c:411
#37 captured_command_loop () at ../../gdb/main.c:471
#38 0x00005555558ca725 in captured_main (data=<optimized out>) at
    ../../gdb/main.c:1329
--Type <RET> for more, q to quit, c to continue without paging--
#39 gdb_main (args=<optimized out>) at ../../gdb/main.c:1344
#40 0x000055555565ccd0 in main (argc=<optimized out>, argv=<optimized out>)
    at ../../gdb/gdb.c:32

(gdb) c
Continuing.

Thread 1 "gdb" hit Breakpoint 1, value_as_address (val=0x555556226c50) at
../../gdb/value.c:2757
2757        {
(gdb) bt
#0  value_as_address (val=0x555556226c50) at ../../gdb/value.c:2757
#1  0x0000555555826320 in address_from_register (regnum=<optimized out>,
    frame=frame@entry=0x55555642c0b0)
    at ../../gdb/findvar.c:979
#2  0x00005555557a229f in read_addr_from_reg (frame=frame@entry=0x55555642c0b0,
```

```
    reg=<optimized out>)
    at ../../gdb/dwarf2/expr.c:86
#3  0x00005555557ab35e in dwarf2_frame_cache (this_frame=0x55555642c0b0,
    this_cache=<optimized out>)
    at ../../gdb/dwarf2/frame.c:962
#4  0x00005555557ab5d7 in dwarf2_frame_this_id (this_frame=0x55555642c0b0,
    this_cache=<optimized out>,
    this_id=0x55555642c110) at ../../gdb/dwarf2/frame.c:1117
#5  0x000055555582b6e4 in compute_frame_id (fi=0x55555642c0b0) at
    ../../gdb/frame.c:583
#6  0x000055555582bcaf in get_prev_frame_maybe_check_cycle
    (this_frame=0x55555600eb30) at ../../gdb/frame.c:2082
#7  0x000055555582c480 in get_prev_frame_always_1 (this_frame=0x55555600eb30)
    at ../../gdb/frame.c:2296
#8  0x000055555582c8c0 in get_prev_frame_always (this_frame=0x55555600eb30) at
    ../../gdb/frame.c:2312
#9  0x000055555582d04d in get_frame_unwind_stop_reason
    (frame=frame@entry=0x55555600eb30) at ../../gdb/frame.c:2987
#10 0x00005555557a76b5 in dwarf2_frame_cfa (this_frame=0x55555600eb30) at
    ../../gdb/dwarf2/frame.c:1356
#11 0x00005555557a4b20 in dwarf_expr_context::execute_stack_op
    (this=0x7fffffffd870, op_ptr=<optimized out>,
    op_end=0x5555561c8c60 " \003") at ../../gdb/dwarf2/expr.c:2110
#12 0x00005555557a5c98 in dwarf_expr_context::eval (this=0x7fffffffd870,
    addr=<optimized out>, len=<optimized out>)
    at ../../gdb/dwarf2/expr.c:1238
#13 0x00005555557a51cd in dwarf_expr_context::execute_stack_op
    (this=0x7fffffffd870, op_ptr=0x5555561c8c71 "\021pa",
    op_end=0x5555561c8c71 "\021pa") at ../../gdb/dwarf2/expr.c:1810
#14 0x00005555557a5c98 in dwarf_expr_context::eval (this=0x7fffffffd870,
    addr=<optimized out>, len=<optimized out>)
    at ../../gdb/dwarf2/expr.c:1238
#15 0x00005555557a5f46 in dwarf_expr_context::evaluate
    (this=this@entry=0x7fffffffd870,
    addr=addr@entry=0x5555561c8c6f "\221\\\021pa", len=len@entry=2,
    as_lval=as_lval@entry=true,
    per_cu=per_cu@entry=0x5555561b7230, frame=frame@entry=0x55555600eb30,
    addr_info=0x0, type=0x5555561ee850,
    subobj_type=0x5555561ee850, subobj_offset=0) at
    ../../gdb/dwarf2/expr.c:1077
#16 0x00005555557b8716 in dwarf2_evaluate_loc_desc_full (type=0x5555561ee850,
    frame=0x55555600eb30,
    data=0x5555561c8c6f "\221\\\021pa", size=2, per_cu=0x5555561b7230,
    per_objfile=<optimized out>,
    subobj_type=0x5555561ee850, subobj_byte_offset=0, as_lval=true) at
    ../../gdb/dwarf2/loc.c:1519
#17 0x00005555557b8aaa in dwarf2_evaluate_loc_desc (as_lval=true,
    per_objfile=<optimized out>,
--Type <RET> for more, q to quit, c to continue without paging--
    per_cu=<optimized out>, size=<optimized out>, data=<optimized out>,
    frame=<optimized out>, type=<optimized out>)
    at ../../gdb/dwarf2/loc.c:1563
#18 locexpr_read_variable (symbol=<optimized out>, frame=<optimized out>)
    at ../../gdb/dwarf2/loc.c:3053
#19 0x000055555582593f in language_defn::read_var_value (this=<optimized out>,
    var=0x5555561f0740,
    var_block=0x5555561f08e0, frame=0x55555600eb30) at ../../gdb/symtab.h:1232
#20 0x00005555558806b47 in evaluate_var_value (noside=EVAL_NORMAL,
```

```
    blk=<optimized out>, var=0x5555561f0740)
    at ../../gdb/eval.c:559
#21 0x0000555555805e92 in expression::evaluate (this=0x55555621d6b0,
    expect_type=0x0, noside=EVAL_NORMAL)
    at /usr/include/c++/9/bits/unique_ptr.h:360
#22 0x0000555555927fc0 in process_print_command_args (args=<optimized out>,
    print_opts=0x7fffffffdac0,
    voidprint=<optimized out>) at /usr/include/c++/9/bits/unique_ptr.h:360
#23 0x00005555559287eb in print_command_1 (args=<optimized out>, voidprint=1)
    at ../../gdb/printcmd.c:1320
#24 0x0000555555733f45 in cmd_func (cmd=<optimized out>, args=<optimized out>,
    from_tty=<optimized out>)
    at ../../gdb/cli/cli-decode.c:2516
#25 0x0000555555a3c257 in execute_command (p=<optimized out>,
    p@entry=0x555555fae210 "print a", from_tty=1)
    at ../../gdb/top.c:699
#26 0x000055555580c965 in command_handler (command=0x555555fae210 "print a")
    at ../../gdb/event-top.c:598
#27 0x000055555580cd51 in command_line_handler (rl=...) at
    ../../gdb/event-top.c:842
#28 0x000055555580d4ec in gdb_rl_callback_handler (rl=0x55555621d690 "print a")
    at /usr/include/c++/9/bits/unique_ptr.h:153
#29 0x0000555555ae1e78 in rl_callback_read_char () at
    ../../../readline/readline/callback.c:290
#30 0x000055555580bd86 in gdb_rl_callback_read_char_wrapper_noexcept () at
    ../../gdb/event-top.c:188
#31 0x000055555580d3a5 in gdb_rl_callback_read_char_wrapper
    (client_data=<optimized out>) at ../../gdb/event-top.c:204
#32 0x000055555580bc98 in stdin_event_handler (error=<optimized out>,
    client_data=0x555555fafa80)
    at ../../gdb/event-top.c:525
#33 0x0000555555bc19c6 in gdb_wait_for_event (block=block@entry=1)
    at ../../gdbsupport/event-loop.cc:670
#34 0x0000555555bc1c3b in gdb_wait_for_event (block=1) at
    ../../gdbsupport/event-loop.cc:569
#35 gdb_do_one_event () at ../../gdbsupport/event-loop.cc:235
#36 0x00005555558c8b55 in start_event_loop () at ../../gdb/main.c:411
#37 captured_command_loop () at ../../gdb/main.c:471
#38 0x00005555558ca725 in captured_main (data=<optimized out>) at
    ../../gdb/main.c:1329
--Type <RET> for more, q to quit, c to continue without paging--
#39 gdb_main (args=<optimized out>) at ../../gdb/main.c:1344
#40 0x000055555565ccd0 in main (argc=<optimized out>, argv=<optimized out>)
    at ../../gdb/gdb.c:32
(gdb) c
Continuing.

Thread 1 "gdb" hit Breakpoint 1, value_as_address (val=0x555556247320)
at ../../gdb/value.c:2757
2757        {
(gdb) bt
#0  value_as_address (val=0x555556247320) at ../../gdb/value.c:2757
#1  0x00005555557a2bd2 in dwarf_expr_context::fetch_result
    (this=this@entry=0x7fffffffd870, type=<optimized out>,
    type@entry=0x5555561ee850, subobj_type=subobj_type@entry=0x5555561ee850,
    subobj_offset=subobj_offset@entry=0,
    as_lval=<optimized out>) at ../../gdb/dwarf2/expr.c:1002
#2  0x00005555557a5f63 in dwarf_expr_context::evaluate
```

```
    (this=this@entry=0x7fffffffd870,
    addr=addr@entry=0x5555561c8c6f "\221\\\021pa", len=len@entry=2,
    as_lval=as_lval@entry=true,
    per_cu=per_cu@entry=0x5555561b7230, frame=frame@entry=0x55555600eb30,
    addr_info=0x0, type=0x5555561ee850,
    subobj_type=0x5555561ee850, subobj_offset=0) at ../../gdb/dwarf2/expr.c:1078
#3  0x000055555557b8716 in dwarf2_evaluate_loc_desc_full (type=0x5555561ee850,
    frame=0x55555600eb30,
    data=0x5555561c8c6f "\221\\\021pa", size=2, per_cu=0x5555561b7230,
    per_objfile=<optimized out>,
    subobj_type=0x5555561ee850, subobj_byte_offset=0, as_lval=true) at
    ../../gdb/dwarf2/loc.c:1519
#4  0x000055555557b8aaa in dwarf2_evaluate_loc_desc (as_lval=true,
    per_objfile=<optimized out>,
    per_cu=<optimized out>, size=<optimized out>, data=<optimized out>,
    frame=<optimized out>, type=<optimized out>)
    at ../../gdb/dwarf2/loc.c:1563
#5  locexpr_read_variable (symbol=<optimized out>, frame=<optimized out>)
    at ../../gdb/dwarf2/loc.c:3053
#6  0x000055555582593f in language_defn::read_var_value (this=<optimized out>,
    var=0x5555561f0740,
    var_block=0x5555561f08e0, frame=0x55555600eb30) at ../../gdb/symtab.h:1232
#7  0x0000555555806b47 in evaluate_var_value (noside=EVAL_NORMAL,
    blk=<optimized out>, var=0x5555561f0740)
    at ../../gdb/eval.c:559
#8  0x0000555555805e92 in expression::evaluate (this=0x55555621d6b0,
    expect_type=0x0, noside=EVAL_NORMAL)
    at /usr/include/c++/9/bits/unique_ptr.h:360
#9  0x0000555555927fc0 in process_print_command_args (args=<optimized out>,
    print_opts=0x7fffffffdac0,
    voidprint=<optimized out>) at /usr/include/c++/9/bits/unique_ptr.h:360
#10 0x00005555559287eb in print_command_1 (args=<optimized out>, voidprint=1)
    at ../../gdb/printcmd.c:1320
#11 0x0000555555733f45 in cmd_func (cmd=<optimized out>, args=<optimized out>,
    from_tty=<optimized out>)
    at ../../gdb/cli/cli-decode.c:2516
#12 0x0000555555a3c257 in execute_command (p=<optimized out>,
    p@entry=0x555555fae210 "print a", from_tty=1)
    at ../../gdb/top.c:699
#13 0x000055555580c965 in command_handler (command=0x555555fae210 "print a")
    at ../../gdb/event-top.c:598
#14 0x000055555580cd51 in command_line_handler (rl=...) at
    ../../gdb/event-top.c:842
#15 0x000055555580d4ec in gdb_rl_callback_handler (rl=0x55555621d690 "print a")
--Type <RET> for more, q to quit, c to continue without paging--
    at /usr/include/c++/9/bits/unique_ptr.h:153
#16 0x0000555555ae1e78 in rl_callback_read_char () at
    ../../../readline/readline/callback.c:290
#17 0x000055555580bd86 in gdb_rl_callback_read_char_wrapper_noexcept () at
    ../../gdb/event-top.c:188
#18 0x000055555580d3a5 in gdb_rl_callback_read_char_wrapper
    (client_data=<optimized out>) at ../../gdb/event-top.c:204
#19 0x000055555580bc98 in stdin_event_handler (error=<optimized out>,
    client_data=0x555555fafa80)
    at ../../gdb/event-top.c:525
#20 0x0000555555bc19c6 in gdb_wait_for_event (block=block@entry=1) at
    ../../gdbsupport/event-loop.cc:670
#21 0x0000555555bc1c3b in gdb_wait_for_event (block=1) at
```

```
    ../../gdbsupport/event-loop.cc:569
#22 gdb_do_one_event () at ../../gdbsupport/event-loop.cc:235
#23 0x00005555558c8b55 in start_event_loop () at ../../gdb/main.c:411
#24 captured_command_loop () at ../../gdb/main.c:471
#25 0x00005555558ca725 in captured_main (data=<optimized out>) at
    ../../gdb/main.c:1329
#26 gdb_main (args=<optimized out>) at ../../gdb/main.c:1344
#27 0x000055555565ccd0 in main (argc=<optimized out>, argv=<optimized out>)
    at ../../gdb/gdb.c:32
(gdb) c
Continuing.

# Finally, content of 'a'
$1 = 5

# Let's check 'pa'
(gdb) print pa

Thread 1 "gdb" hit Breakpoint 1, value_as_address (val=0x555556227f80) at
../../gdb/value.c:2757
2757        {
(gdb) bt
#0  value_as_address (val=0x555556227f80) at ../../gdb/value.c:2757
#1  0x00005555557a2bd2 in dwarf_expr_context::fetch_result
    (this=this@entry=0x7fffffffd870, type=<optimized out>,
    type@entry=0x5555561f0810, subobj_type=subobj_type@entry=0x5555561f0810,
    subobj_offset=subobj_offset@entry=0,
    as_lval=<optimized out>) at ../../gdb/dwarf2/expr.c:1002
#2  0x00005555557a5f63 in dwarf_expr_context::evaluate
    (this=this@entry=0x7fffffffd870,
    addr=addr@entry=0x5555561c8c7d "\221`", len=len@entry=2,
    as_lval=as_lval@entry=true,
    per_cu=per_cu@entry=0x5555561b7230, frame=frame@entry=0x55555600eb30,
    addr_info=0x0, type=0x5555561f0810,
    subobj_type=0x5555561f0810, subobj_offset=0) at ../../gdb/dwarf2/expr.c:1078
#3  0x00005555557b8716 in dwarf2_evaluate_loc_desc_full (type=0x5555561f0810,
    frame=0x55555600eb30,
    data=0x5555561c8c7d "\221`", size=2, per_cu=0x5555561b7230,
    per_objfile=<optimized out>,
    subobj_type=0x5555561f0810, subobj_byte_offset=0, as_lval=true) at
    ../../gdb/dwarf2/loc.c:1519
#4  0x00005555557b8aaa in dwarf2_evaluate_loc_desc (as_lval=true,
    per_objfile=<optimized out>,
    per_cu=<optimized out>, size=<optimized out>, data=<optimized out>,
    frame=<optimized out>, type=<optimized out>)
    at ../../gdb/dwarf2/loc.c:1563
#5  locexpr_read_variable (symbol=<optimized out>, frame=<optimized out>)
    at ../../gdb/dwarf2/loc.c:3053
#6  0x000055555582593f in language_defn::read_var_value (this=<optimized out>,
    var=0x5555561f07c0,
    var_block=0x5555561f08e0, frame=0x55555600eb30) at ../../gdb/symtab.h:1232
#7  0x0000555555806b47 in evaluate_var_value (noside=EVAL_NORMAL,
    blk=<optimized out>, var=0x5555561f07c0)
    at ../../gdb/eval.c:559
#8  0x0000555555805e92 in expression::evaluate (this=0x55555622c5c0,
    expect_type=0x0, noside=EVAL_NORMAL)
    at /usr/include/c++/9/bits/unique_ptr.h:360
#9  0x0000555555927fc0 in process_print_command_args (args=<optimized out>,
```

```
    print_opts=0x7fffffffdac0,
    voidprint=<optimized out>) at /usr/include/c++/9/bits/unique_ptr.h:360
#10 0x00005555559287eb in print_command_1 (args=<optimized out>, voidprint=1)
    at ../../gdb/printcmd.c:1320
#11 0x0000555555733f45 in cmd_func (cmd=<optimized out>, args=<optimized out>,
    from_tty=<optimized out>)
    at ../../gdb/cli/cli-decode.c:2516
#12 0x0000555555a3c257 in execute_command (p=<optimized out>,
    p@entry=0x555555fae230 "print pa", from_tty=1)
    at ../../gdb/top.c:699
#13 0x000055555580c965 in command_handler (command=0x555555fae230 "print pa")
     at ../../gdb/event-top.c:598
#14 0x000055555580cd51 in command_line_handler (rl=...) at
    ../../gdb/event-top.c:842
#15 0x000055555580d4ec in gdb_rl_callback_handler
    (rl=0x55555621d690 "print pa")
--Type <RET> for more, q to quit, c to continue without paging--
    at /usr/include/c++/9/bits/unique_ptr.h:153
#16 0x0000555555ae1e78 in rl_callback_read_char () at
    ../../../readline/readline/callback.c:290
#17 0x000055555580bd86 in gdb_rl_callback_read_char_wrapper_noexcept () at
    ../../gdb/event-top.c:188
#18 0x000055555580d3a5 in gdb_rl_callback_read_char_wrapper
    (client_data=<optimized out>) at ../../gdb/event-top.c:204
#19 0x000055555580bc98 in stdin_event_handler (error=<optimized out>,
    client_data=0x555555fafa80)
    at ../../gdb/event-top.c:525
#20 0x0000555555bc19c6 in gdb_wait_for_event (block=block@entry=1) at
    ../../gdbsupport/event-loop.cc:670
#21 0x0000555555bc1c3b in gdb_wait_for_event (block=1) at
    ../../gdbsupport/event-loop.cc:569
#22 gdb_do_one_event () at ../../gdbsupport/event-loop.cc:235
#23 0x00005555558c8b55 in start_event_loop () at ../../gdb/main.c:411
#24 captured_command_loop () at ../../gdb/main.c:471
#25 0x00005555558ca725 in captured_main (data=<optimized out>) at
    ../../gdb/main.c:1329
#26 gdb_main (args=<optimized out>) at ../../gdb/main.c:1344
#27 0x000055555565ccd0 in main (argc=<optimized out>, argv=<optimized out>) at
    ../../gdb/gdb.c:32
(gdb)
(gdb) c
Continuing.

# 'pa' content
$2 = (int *) 0x7fffffffdf3c
(gdb) c
Continuing.
'a' is at address '0x7fffffffdf3c'.
'a' has value '5'
[Inferior 1 (process 26556) exited normally]
```

Note that:

- Before reaching *value_as_address*, a certain set of event handlers are being called (*fetch_inferior_event*, *handle_inferior_event*, *handle_solib_event*, *svr4_handle_solib_event* etc).
- Command *print $some_var* causes *dwarf_**, *dwarf2_** (and eventually *read_addr_from_reg* and *address_from_register* if *print* is being called for the first time) function calls.
- First *print $some_var* (*print a* in the example) call caused three *value_as_address* breakpoint hits.

There were *read_addr_from_reg* and *address_from_register* stack frames present only in the first two backtraces, but not in the third, when *value* parameter for *value_as_address* function changed from $0x555556226c50$ to $0x555556247320$. Only then, stack frame of *dwarf_expr_context::fetch_result* has been created for the first time. Every latter call of *print $some_var* command called *dwarf_expr_context::fetch_result* function and didn't produce *read_addr_from_reg* and *address_from_register* stack frames and didn't hit *value_as_address* breakpoint three but just one time.

Example 3

In this example, we use *next* command in order to find out which functions are being called after stack frame of *value_as_address* spawns:

```
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.1) 9.2
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from /home/syrmia/builds/gdb/bin/gdb...

(gdb) b value_as_address
Breakpoint 1 at 0x5646c0: file ../../gdb/value.c, line 2757.

(gdb) run gdb-test/test
Starting program: /home/syrmia/builds/gdb/bin/gdb gdb-test/test
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
[Detaching after vfork from child process 8771]
[New Thread 0x7ffff49fc700 (LWP 8772)] ...
GNU gdb (GDB) 13.0.50.20220815-git
Copyright (C) 2022 Free Software Foundation, Inc ...
Reading symbols from gdb-test/test...

(gdb) list
1       #include <stdio.h>
2
3       int main()
4       {
5               printf("Address of main is %p\n", main);
6               int a = 5;
7               int *pa = &a;
8               printf("'a' is at address '%p'.\n'a' has value '%d'\n", pa, *pa);
9
10               return 0;

(gdb) b main
Breakpoint 1 at 0x1175: file test.c, line 4.

(gdb) b 8
Breakpoint 2 at 0x11ab: file test.c, line 8.

(gdb) r
Starting program: /home/syrmia/gdb-test/test
[Detaching after vfork from child process 8782]
[Detaching after fork from child process 8783]
[Detaching after fork from child process 8784]

Thread 1 "gdb" hit Breakpoint 1, value_as_address (val=0x555556228770) at
../../gdb/value.c:2757
2757        {
```

```
(gdb) bt
#0  value_as_address (val=0x5555562287f0) at ../../gdb/value.c:2757
#1  0x00005555559c7cc6 in svr4_handle_solib_event () at
    ../../gdb/solib-svr4.c:1838
#2  0x00005555559cd7f0 in handle_solib_event () at ../../gdb/solib.c:1338
#3  0x00005555556f4865 in bpstat_stop_status (aspace=<optimized out>,
    bp_addr=bp_addr@entry=140737353955253, thread=thread@entry=0x555556174590,
    ws=...,
    stop_chain=stop_chain@entry=0x0) at ../../gdb/breakpoint.c:5558
#4  0x000055555587faac in handle_signal_stop (ecs=0x7fffffffdd20) at
    ../../gdb/regcache.h:344
#5  0x000055555588209c in handle_inferior_event (ecs=<optimized out>) at
    ../../gdb/infrun.c:5869
#6  0x00005555558831fb in fetch_inferior_event () at ../../gdb/infrun.c:4233
#7  0x0000555555bc19c6 in gdb_wait_for_event (block=block@entry=0) at
    ../../gdbsupport/event-loop.cc:670
#8  0x0000555555bc1c86 in gdb_wait_for_event (block=0) at
    ../../gdbsupport/event-loop.cc:569
#9  gdb_do_one_event () at ../../gdbsupport/event-loop.cc:210
#10 0x00005555558c8b55 in start_event_loop () at ../../gdb/main.c:411
#11 captured_command_loop () at ../../gdb/main.c:471
#12 0x00005555558ca725 in captured_main (data=<optimized out>) at
    ../../gdb/main.c:1329
#13 gdb_main (args=<optimized out>) at ../../gdb/main.c:1344
#14 0x000055555565ccd0 in main (argc=<optimized out>, argv=<optimized out>) at
    ../../gdb/gdb.c:32

(gdb) next
2758          struct gdbarch *gdbarch = value_type (val)->arch ();
(gdb)
1027            return this->main_type->code;
(gdb)
2810          val = coerce_array (val);
(gdb)
2849          if (!value_type (val)->is_pointer_or_reference ()
(gdb)
2854          return unpack_long (value_type (val), value_contents (val).data ());
(gdb)
1100          return value->type;
(gdb)
unpack_long (type=0x555556372aa0, valaddr=0x555556231820 "`\341\377\367\377\177")
 at ../../gdb/value.c:2874
2874        {

(gdb) bt
#0  unpack_long (type=0x555556372aa0,
    valaddr=0x555556231820 "`\341\377\367\377\177")
    at ../../gdb/value.c:2874
#1  0x00005555559c7cc6 in svr4_handle_solib_event () at
    ../../gdb/solib-svr4.c:1838
#2  0x00005555559cd7f0 in handle_solib_event () at ../../gdb/solib.c:1338
#3  0x00005555556f4865 in bpstat_stop_status (aspace=<optimized out>,
    bp_addr=bp_addr@entry=140737353955253, thread=thread@entry=0x555556174590, ws=...,
    stop_chain=stop_chain@entry=0x0) at ../../gdb/breakpoint.c:5558
#4  0x000055555587faac in handle_signal_stop (ecs=0x7fffffffdd20) at
    ../../gdb/regcache.h:344
#5  0x000055555588209c in handle_inferior_event (ecs=<optimized out>) at
    ../../gdb/infrun.c:5869
```

```
#6  0x00005555558831fb in fetch_inferior_event () at ../../gdb/infrun.c:4233
#7  0x0000555555bc19c6 in gdb_wait_for_event (block=block@entry=0) at
    ../../gdbsupport/event-loop.cc:670
#8  0x0000555555bc1c86 in gdb_wait_for_event (block=0) at
    ../../gdbsupport/event-loop.cc:569
#9  gdb_do_one_event () at ../../gdbsupport/event-loop.cc:210
#10 0x00005555558c8b55 in start_event_loop () at ../../gdb/main.c:411
#11 captured_command_loop () at ../../gdb/main.c:471
#12 0x00005555558ca725 in captured_main (data=<optimized out>)
    at ../../gdb/main.c:1329
#13 gdb_main (args=<optimized out>) at ../../gdb/main.c:1344
#14 0x0000555555565ccd0 in main (argc=<optimized out>, argv=<optimized out>)
    at ../../gdb/gdb.c:32

(gdb) next
2875          if (is_fixed_point_type (type))
(gdb)
2878          enum bfd_endian byte_order = type_byte_order (type);
(gdb)
2881          int nosign = type->is_unsigned ();
(gdb)
2883          switch (code)
(gdb)
2897                if (type->bit_size_differs_p ())
(gdb)
2912                    if (nosign)
(gdb)
2913                      result = extract_unsigned_integer (valaddr, len, byte_order);
(gdb)
2917                if (code == TYPE_CODE_RANGE)
(gdb)
svr4_handle_solib_event () at ../../gdb/solib-svr4.c:1839
1839            if (debug_base == 0)

(gdb) bt
#0  svr4_handle_solib_event () at ../../gdb/solib-svr4.c:1839
#1  0x00005555559cd7f0 in handle_solib_event () at ../../gdb/solib.c:1338
#2  0x00005555556f4865 in bpstat_stop_status (aspace=<optimized out>,
    bp_addr=bp_addr@entry=140737353955253, thread=thread@entry=0x555556174590,
    ws=...,
    stop_chain=stop_chain@entry=0x0) at ../../gdb/breakpoint.c:5558
#3  0x0000555555587faac in handle_signal_stop (ecs=0x7fffffffdd20) at
    ../../gdb/regcache.h:344
#4  0x0000555555588209c in handle_inferior_event (ecs=<optimized out>) at
    ../../gdb/infrun.c:5869
#5  0x00005555558831fb in fetch_inferior_event () at ../../gdb/infrun.c:4233
#6  0x0000555555bc19c6 in gdb_wait_for_event (block=block@entry=0) at
    ../../gdbsupport/event-loop.cc:670
#7  0x0000555555bc1c86 in gdb_wait_for_event (block=0) at
    ../../gdbsupport/event-loop.cc:569
#8  gdb_do_one_event () at ../../gdbsupport/event-loop.cc:210
#9  0x00005555558c8b55 in start_event_loop () at ../../gdb/main.c:411
#10 captured_command_loop () at ../../gdb/main.c:471
#11 0x00005555558ca725 in captured_main (data=<optimized out>) at
    ../../gdb/main.c:1329
#12 gdb_main (args=<optimized out>) at ../../gdb/main.c:1344
#13 0x0000555555565ccd0 in main (argc=<optimized out>, argv=<optimized out>)
    at ../../gdb/gdb.c:32
```

```
(gdb) next
1843            info->debug_base = 0;
(gdb)
1844            if (locate_base (info) == 0)
(gdb)
1823              = inhibit_section_map_updates (current_program_space);
(gdb)
1874            if (action == UPDATE_OR_RELOAD)
(gdb)
1823              = inhibit_section_map_updates (current_program_space);
(gdb)
1903          if (action == FULL_RELOAD)
(gdb)
1905              if (!solist_update_full (info))
(gdb)
1695          return 1;
(gdb)
handle_solib_event () at ../../gdb/solib.c:1340
1340          current_inferior ()->pspace->clear_solib_cache ();

(gdb) bt
#0  handle_solib_event () at ../../gdb/solib.c:1340
#1  0x00005555556f4865 in bpstat_stop_status (aspace=<optimized out>,
    bp_addr=bp_addr@entry=140737353955253, thread=thread@entry=0x555556174590,
     ws=...,
    stop_chain=stop_chain@entry=0x0) at ../../gdb/breakpoint.c:5558
#2  0x000055555587faac in handle_signal_stop (ecs=0x7fffffffdd20) at
    ../../gdb/regcache.h:344
#3  0x000055555588209c in handle_inferior_event (ecs=<optimized out>) at
    ../../gdb/infrun.c:5869
#4  0x00005555558831fb in fetch_inferior_event () at ../../gdb/infrun.c:4233
#5  0x0000555555bc19c6 in gdb_wait_for_event (block=block@entry=0) at
    ../../gdbsupport/event-loop.cc:670
#6  0x0000555555bc1c86 in gdb_wait_for_event (block=0) at
    ../../gdbsupport/event-loop.cc:569
#7  gdb_do_one_event () at ../../gdbsupport/event-loop.cc:210
#8  0x00005555558c8b55 in start_event_loop () at ../../gdb/main.c:411
#9  captured_command_loop () at ../../gdb/main.c:471
#10 0x00005555558ca725 in captured_main (data=<optimized out>) at
    ../../gdb/main.c:1329
#11 gdb_main (args=<optimized out>) at ../../gdb/main.c:1344
#12 0x000055555565ccd0 in main (argc=<optimized out>, argv=<optimized out>)
    at ../../gdb/gdb.c:32

(gdb) next
1345          target_terminal::ours_for_output ();
(gdb)
1346          solib_add (NULL, 0, auto_solib_add);
(gdb)
1347          target_terminal::inferior ();
(gdb)
target_terminal::inferior () at ../../gdb/target.c:940
940       {
(gdb)
941         struct ui *ui = current_ui;
(gdb)
945         if (ui->prompt_state != PROMPT_BLOCKED)
```

```
(gdb)
952           if (ui != main_ui)
(gdb)
958           struct inferior *inf = current_inferior ();
(gdb)
960           if (inf->terminal_state != target_terminal_state::is_inferior)
(gdb)
962               current_inferior ()->top_target ()->terminal_inferior ();
(gdb)
963               inf->terminal_state = target_terminal_state::is_inferior;
(gdb)
966           m_terminal_state = target_terminal_state::is_inferior;
(gdb)
970           if (check_quit_flag ())
(gdb)
bpstat_stop_status (aspace=<optimized out>,
bp_addr=bp_addr@entry=140737353955253,
thread=thread@entry=0x555556174590, ws=..., stop_chain=stop_chain@entry=0x0)
    at ../../gdb/breakpoint.c:5559
5559                    break;
(gdb)
5569          for (bs = bs_head; bs != NULL; bs = bs->next)
(gdb)
5571              if (!bs->stop)
(gdb)
5574              b = bs->breakpoint_at;
(gdb)
5575              b->check_status (bs);
(gdb)
5576              if (bs->stop)
(gdb)
5608                bs->print_it = print_it_noop;
(gdb)
5569          for (bs = bs_head; bs != NULL; bs = bs->next)
(gdb)
5616          if (! bpstat_causes_stop (bs_head))
(gdb)
5618              if (!bs->stop
(gdb)
5620                  && is_hardware_watchpoint (bs->breakpoint_at))
(gdb)
5617            for (bs = bs_head; bs != NULL; bs = bs->next)
(gdb)
5628          if (need_remove_insert)
(gdb)
5630          else if (removed_any)
(gdb)
5633          return bs_head;
(gdb)
handle_signal_stop (ecs=0x7fffffffdd20) at ../../gdb/infrun.c:6406
6406          if (ecs->event_thread->stop_signal () == GDB_SIGNAL_TRAP
(gdb)
423            return m_suspend.stop_signal;

(gdb) bt
#0  handle_signal_stop (ecs=0x7fffffffdd20) at ../../gdb/gdbthread.h:423
#1  0x000055555588209c in handle_inferior_event (ecs=<optimized out>) at
     ../../gdb/infrun.c:5869
```

```
#2  0x00005555558831fb in fetch_inferior_event () at ../../gdb/infrun.c:4233
#3  0x0000555555bc19c6 in gdb_wait_for_event (block=block@entry=0) at
    ../../gdbsupport/event-loop.cc:670
#4  0x0000555555bc1c86 in gdb_wait_for_event (block=0) at
    ../../gdbsupport/event-loop.cc:569
#5  gdb_do_one_event () at ../../gdbsupport/event-loop.cc:210
#6  0x00005555558c8b55 in start_event_loop () at ../../gdb/main.c:411
#7  captured_command_loop () at ../../gdb/main.c:471
#8  0x00005555558ca725 in captured_main (data=<optimized out>) at
    ../../gdb/main.c:1329
#9  gdb_main (args=<optimized out>) at ../../gdb/main.c:1344
#10 0x000055555565ccd0 in main (argc=<optimized out>, argv=<optimized out>)
     at ../../gdb/gdb.c:32
```

We saw that *value_as_address* calls *unpack_long* which enters *if (code == TYPE_CODE_RANGE)* block and returns. *TYPE_CODE_RANGE* is 'integer within specified bounds' as specified in *gdbtypes.h* header file. After *value_as_address* i.e. *unpack_long* finishes, stack frame shrinks *svr4_handle_solib_event* → *handle_solib_event* etc).

Example 4

We know that *value_as_address* gets *value* parameter. But, we also know that the same function is invoked even before reaching main. There are certain preparations before *main* function can execute its instructions and by default, entry point for executable is *_start* function. This example proves that *value* parameter isn't address of *_start* function.

Source code of executable is saved to *no-main.c* file:

```
#include <stdio.h>
#include <stdlib.h>

int my_fun()
{
        printf("Hello from my_fun!\n");
        return 0;
}

int _start()
{
        printf("Address of _start is %p\n", _start);
        printf("Address of my_fun is %p\n", my_fun);
        int retcode = my_fun();
        exit(retcode);
}
```

and is compiled with:

```
gcc -g -nostartfiles -o no-main no-main.c
```

Here is debugger output:

```
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.1) 9.2
Reading symbols from /home/syrmia/builds/gdb/bin/gdb...
(gdb) b value_as_address
Breakpoint 1 at 0x5646c0: file ../../gdb/value.c, line 2757.
(gdb) r no-main
Starting program: /home/syrmia/builds/gdb/bin/gdb no-main
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
[Detaching after vfork from child process 13156]
```

```
[New Thread 0x7ffff49fc700 (LWP 13157)] ...
GNU gdb (GDB) 13.0.50.20220815-git
Reading symbols from no-main...
(gdb) list 14
9
10      int _start()
11      {
12              printf("Address of _start is %p\n", _start);
13              printf("Address of my_fun is %p\n", my_fun);
14              int retcode = my_fun();
15              exit(retcode);
16      }
17
(gdb) b _start
Breakpoint 1 at 0x1097: file no-main.c, line 12.
(gdb) r
Starting program: /home/syrmia/gdb-test-no-main/no-main
[Detaching after vfork from child process 13166] ...

Thread 1 "gdb" hit Breakpoint 1, value_as_address (val=0x55555622d1a0) at
../../gdb/value.c:2757
2757        {
(gdb) x 0x55555622d1a0
0x55555622d1a0:        0x00000000
(gdb) c
Continuing.

Breakpoint 1, _start () at no-main.c:12
12              printf("Address of _start is %p\n", _start);
(gdb) c
Continuing.
Address of _start is 0x55555555508b
Address of my_fun is 0x555555555070
Hello from my_fun!
[Inferior 1 (process 13166) exited normally]
```

Since *value_as_address* breakpoint was hit even before *no-main* executable started (*_start* breakpoint), we can conclude that *gdb* performs some preparations before debugged program commences its execution. As we already said, parameter *value* passed to *value_as_address* isn't address of *_start* nor *my_fun* function.

DWARF Functions in GDB

Most common debugging formats are *COFF*, *STABS* and *DWARF*, which will be in our focus. *DWARF* is standardized debugging data format which uses data structure called **DIE** (Debugging Information Entry), noted as *DW_TAG* and organized in a tree structure. Each *DIE* has attributes which begin with *DW_AT* (*DW_AT_name*, *DW_AT_type*, *DW_AT_declaration* etc.). All debug informations are divided into sections, of which most important are:

- **.debug_line** - contains information about line numbers from source code
- **.debug_info** - most important section containing tree structure of debug informations. Refers to data from other sections.
- **.debug_loc** - Lists with locations of variables which can be found in *DW_AT_location* attribute
- **.debug_frame** - Informations about stack frame of called functions

Let us mention some *DIE* nodes:

- **DW_TAG_compile_unit** - node which represents single compilation unit
- **DW_TAG_subprogram** - node which represents subroutine or function
- **DW_TAG_formal_parameter** - function parameter

and some *DIE* attributes:

- **DW_AT_name** - name of variable, function, or parameter
- **DW_AT_type** - type of a variable, function or parameter
- **DW_AT_decl*** - location (file, line, column) where variable or function has been declared
- **DW_AT_low_pc** - address of first instruction of procedure or address to jump to after procedure returns
- **DW_AT_location** - location of variable or parameter. May be single entry or list of locations.

In order to read contents of specific debug section, *llvm-dwarfdump-10* from *clang* module or *objdump* from *gnu binutils* module may be used.

Technical terms

- **DWARF expression** is stream of operations where each operation consists of single opcode and arguments. Number of arguments is implied from opcode. It describes how to compute a value or specify a location.
- **Binary File Descriptor (BFD) Library** is part of *GNU* project and provides compatibility between different object file formats. It's capable to read data from core dump and hence is useful in debugging programs for embedded systems. Since it's used by more *GNU* tools like *gas*, *gld* and *gdb*, it's source code is in *binutils* package.

- **Location Description.** Debugging information must provide consumers a way to find the location of program variables, determine the bounds of dynamic arrays and strings, and possibly to find the base address of a subroutine's stack frame or the return address of a subroutine. Furthermore, to meet the needs of recent computer architectures and optimization techniques, debugging information must be able to describe the location of an object whose location changes over the object's lifetime. Information about the location of program objects is provided by location descriptions. Location description (*DW_AT_location attribute*) can be either in form of:

  1. **single location description**, when object doesn't move during its lifetime or
  2. **location list**, which is used to describe object that has a limited lifetime or changes location as program executes

  Speaking of GDB source code, functions to evaluate *location description* can be found in *loc.c* file and those for evaluation of *DWARF expressions* are in *expr.c*. *dwarf2_evaluate_loc_desc_full* is the key function to compute location description while *dwarf_expr_context::execute_stack_op* is the key function to evaluate *DWARF expression*, like stack trace in the following code shows. Note also that, in order for *dwarf2_evaluate_loc_desc_full* to finish, *DWARF expression* trough *dwarf_expr_context::execute_stack_op* must be evaluated first.

```
(bash) gdb $HOME/builds/gdb/bin/gdb
(gdb) b dwarf2_evaluate_loc_desc_full
(gdb) run test
(gdb) b 8
(gdb) print pa
Thread 1 "gdb" hit Breakpoint 1, dwarf2_evaluate_loc_desc_full
↪  (type=0x5555561f6900, frame=0x55555600eb90, ...
(gdb) step
# repeated 'step' a couple of times...
(gdb) bt
#0  gdbarch_byte_order (gdbarch=gdbarch@entry=0x5555561a7600)
at ../../gdb/gdbarch.c:1458
#1  0x00005555557a3f41 in dwarf_expr_context::execute_stack_op
(this=0x7fffffffd7f0,
op_ptr=0x5555561f85b9 "\221d\237", op_end=0x5555561f85bc "") at
../../gdb/dwarf2/expr.c:1487
#2  0x00005555557a5c98 in dwarf_expr_context::eval (this=0x7fffffffd7f0,
addr=<optimized out>, len=<optimized out>) at ../../gdb/dwarf2/expr.c:1238
#3  0x00005555557a5f46 in dwarf_expr_context::evaluate
(this=0x7fffffffd7f0, addr=<optimized out>, len=<optimized out>,
as_lval=<optimized out>, per_cu=<optimized out>, frame=<optimized out>,
addr_info=0x0, type=0x5555561f6900, subobj_type=0x5555561f6900,
subobj_offset=0) at ../../gdb/dwarf2/expr.c:1077
#4  0x00005555557b8716 in dwarf2_evaluate_loc_desc_full (type=0x5555561f6900,
frame=0x55555600eb90, data=0x5555561f85b9 "\221d\237", size=3,
per_cu=0x5555561bbd80, per_objfile=<optimized out>,
subobj_type=0x5555561f6900, subobj_byte_offset=0, as_lval=true)
at ../../gdb/dwarf2/loc.c:1519
#5  0x00005555557b901b in dwarf2_evaluate_loc_desc (as_lval=true,
per_objfile=<optimized out>, per_cu=<optimized out>, size=<optimized out>,
data=<optimized out>, frame=0x55555600eb90, type=<optimized out>)
at ../../gdb/dwarf2/loc.c:1563
#6  loclist_read_variable (symbol=0x5555561f68b0, frame=0x55555600eb90)
at ../../gdb/dwarf2/loc.c:3890
#7  0x000055555582593f in language_defn::read_var_value
(this=<optimized out>, var=0x5555561f68b0,
var_block=0x5555561f8b30, frame=0x55555600eb90)
at ../../gdb/symtab.h:1232
#8  0x0000555555806b47 in evaluate_var_value (noside=EVAL_NORMAL,
blk=<optimized out>, var=0x5555561f68b0) at ../../gdb/eval.c:559
```

```
#9  0x0000555555805e92 in expression::evaluate (this=0x555556221160,
expect_type=0x0, noside=EVAL_NORMAL)
at /usr/include/c++/9/bits/unique_ptr.h:360
#10 0x0000555555927fc0 in process_print_command_args (args=<optimized out>,
print_opts=0x7fffffffda60, voidprint=<optimized out>) at
/usr/include/c++/9/bits/unique_ptr.h:360
#11 0x00005555559287eb in print_command_1 (args=<optimized out>, voidprint=1)
at ../../gdb/printcmd.c:1320
.
.
.
```

- **CFA (Canonical Frame Address)** is address of caller function's stack frame just before the callee has been called.

  For instance, on Intel x84_64 architecture, we know that there are *rbp* and *rsp* registers pointing to stack base (beginning) and stack top, respectively. CFA there would be content of current *rbp* plus 16 bytes, assuming that stack is full descending. Explanation where from these 16 bytes came from follows:

```
call callee_func
# Address of next instruction after callee_func returns is
# implicitly pushed onto stack (8 bytes) in order for program
# counter register (rip aka pc) to be able to continue execution
# after callee_func returns

# callee_func prologue
enter n, 0

# Previous line is equal to the following three
# push rbp - another 8 bytes onto stack giving 8 + 8 = 16 bytes in total
# mov rbp, rsp
# sub rsp, n - where n means 'n bytes for local variables'

# Consider this and the following two lines to be assembly
# instructions of callee_func and epilogue, when
# callee_func returns to its caller
```

  Previous code snippet may be useful to realize why output of *llvm-dwarfdump-10* has 16 bytes offset when it prints location lists of variables and function parameters.

Common data types

- **struct frame_info** - a structure describing stack frame.
- **struct value** - defines the type of a value.
- **gdb_byte** - unsigned char.
- **struct gdbarch** - representation of architecture on which executable is being run
- **struct symtab** - header (.h, .hpp) or source (.c, .cpp etc.) file
- **struct linetable_entry** - stores (line number, program counter address) pairs. Has additional flags that indicate whether this *pc* data is a good place for placing breakpoint.
- **struct linetable** - array of *struct linetable_entry* values, which is bound to every header and source file i.e. *struct symtab* objects.
- **struct objfile** - object file (usually .o)
- **struct compunit_symtab** - compilation unit

- **struct block** - block of memory containing symbol information
- **struct blockvector** - vector of *blocks*
- **struct general_symbol_info** - structure containing data common to all symbol types
- **struct minimal_symbol_info** - extends *general_symbol_info* and holds information about symbols which aren't neccessary compiled with debug informations. That means, even if a executable hasn't been compiled with debugging information, this struct holds some useful data.

Note difference between *symtab*, *objfile* and *compilation unit*. *struct symtab* corresponds to single file (usually .h or .cpp). *struct compunit_symtab* corresponds to compilation unit while *struct objfile* corresponds to object file, which is product of compiling compilation unit, before linking begins. Interestingly, more object files can be combined into one e.g. by using *ld* linker:

```
ld -relocatable a.o b.o -o c.o
```

and then, on environments that support standard UNIX magic numbers, *c.o* would have value 'OMAGIC' (magic number is number in the very beginning of every file which is used for file associations similarly how Windows uses file extensions) which declares *c.o* as partially linked file. Further usage of newly created *c.o* is equal as with any other object file. Having that said, single object file (*struct objfile*) may have multiple compilation units (*struct compunit_symtab* objects). That's the reason why *struct objfile* has *list of compunits* attribute, but not just single *compunit* attribute. Also, *struct compunit_symtab* contains debugging information in specific format (stabs, dwarf, coff etc.) and symbol tables a.k.a. blockvectors.

Symtab objects which share compilation unit form linked list (*next* attribute) with convention that main.c symtab is the head node. In the same way are connected *struct compunit_symtab* objects which refer to the same *struct objfile*. Object files (*struct objfile* objects) which share the same *BFD* store common data into *struct objfile_per_bfd_storage* object.

Some Functions

```
/* The engine for the expression evaluator.  Using the context in this
   object, evaluate the expression between OP_PTR and OP_END.

   Called by evaluate and dwarf_expr_context::eval functions.
   Has roughly 900 lines of code.    */
void
dwarf_expr_context::execute_stack_op (const gdb_byte *op_ptr,
                                      const gdb_byte *op_end)


/* Evaluate a location description, starting at DATA and with length
   SIZE, to find the current location of variable of TYPE in the
   context of FRAME.  If SUBOBJ_TYPE is non-NULL, return instead the
   location of the subobject of type SUBOBJ_TYPE at byte offset
   SUBOBJ_BYTE_OFFSET within the variable of type TYPE.

   Called by function dwarf2_evaluate_loc_desc   */
static struct value *
dwarf2_evaluate_loc_desc_full (struct type *type, struct frame_info *frame,
                               const gdb_byte *data, size_t size,
                               dwarf2_per_cu_data *per_cu,
                               dwarf2_per_objfile *per_objfile,
                               struct type *subobj_type,
                               LONGEST subobj_byte_offset,
                               bool as_lval)


/* Fetch the result of the expression evaluation in a form of
```

```
   a struct value, where TYPE, SUBOBJ_TYPE and SUBOBJ_OFFSET
   describe the source level representation of that result.
   AS_LVAL defines if the fetched struct value is expected to
   be a value or a location description.  */
value *fetch_result (struct type *type, struct type *subobj_type,
                             LONGEST subobj_offset, bool as_lval);


/* Compute the DWARF CFA (Canonical Frame Address) for a frame.  */
CORE_ADDR dwarf2_frame_cfa (struct frame_info *this_frame);


/* Return a "struct frame_info" corresponding to the frame that called
   THIS_FRAME.  Returns NULL if there is no such frame.

   Unlike get_prev_frame, this function always tries to unwind the
   frame.  */
extern struct frame_info *get_prev_frame_always (struct frame_info *);


/* Return the per-frame unique identifer.  Can be used to relocate a
   frame after a frame cache flush (and other similar operations).  If
   FI is NULL, return the null_frame_id.

   NOTE: kettenis/20040508: These functions return a structure.  On
   platforms where structures are returned in static storage (vax,
   m68k), this may trigger compiler bugs in code like:

   if (frame_id_eq (get_frame_id (l), get_frame_id (r)))

   where the return value from the first get_frame_id (l) gets
   overwritten by the second get_frame_id (r).  Please avoid writing
   code like this.  Use code like:

   struct frame_id id = get_frame_id (l);
   if (frame_id_eq (id, get_frame_id (r)))

   instead, since that avoids the bug.  */
extern struct frame_id get_frame_id (struct frame_info *fi);


/* Compute the frame's uniq ID that can be used to, later, re-find the
   frame.  */
static void
compute_frame_id (struct frame_info *fi)
```

Reading of Symbols

GDB typically reads symbols twice - first is an initial scan which just reads 'partial symbols'; these are partial information for the static/global symbols in a symbol file. Afterwards, when symbol is really needed, say *print $var* called, *lookup_symbol* function from *struct objfile* invokes and symbols are expanded to the entire compilation unit.

When does GDB read debug informations? What information should debugger know if we invoke *list* command, knowing that it prints out line numbers and source file code snippet?

```
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.1) 9.2
Reading symbols from /home/syrmia/builds/gdb/bin/gdb...
```

```
# Put a breakpoint on function which is called to access .debug_line section
b (gdb) b get_debug_line_section
Breakpoint 1 at 0x26d000: file ../../gdb/dwarf2/read.c, line 19760.

# and on a function which returns file and line number for a given address (pc)
(gdb) b find_pc_sect_line
Breakpoint 2 at 0x4b1fa0: file ../../gdb/symtab.c, line 3110.
(gdb) run test
Starting program: /home/syrmia/builds/gdb/bin/gdb test
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
[Detaching after vfork from child process 164341]

# We are now in gdb-debug. Let's see source code of debugged program (test)
GNU gdb (GDB) 13.0.50.20220815-git
Reading symbols from test...
(gdb) list

Thread 1 "gdb" hit Breakpoint 1, get_debug_line_section (cu=0x555556093190)
at ../../gdb/dwarf2/read.c:19760
19760           if (cu->dwo_unit && cu->per_cu->is_debug_types)

# In order to print out line numbers of source file, we need to access
# .debug_line section, which is performed by function get_debug_line_section

(gdb) bt
#0  get_debug_line_section (cu=0x555556093190) at
../../gdb/dwarf2/read.c:19760
#1  0x00005555557c1db9 in dwarf_decode_line_header (sect_off=(unknown: 0),
cu=0x555556093190,
comp_dir=0x55555618251d "/home/syrmia/test") at
../../gdb/dwarf2/read.c:19791
#2  0x00005555557d64c4 in handle_DW_AT_stmt_list (die=0x5555561eda40,
cu=0x555556093190, fnd=...,
lowpc=4224) at ../../gdb/dwarf2/file-and-dir.h:66
#3  0x00005555557eb11e in read_file_scope (cu=0x555556093190,
die=0x5555561eda40)
at ../../gdb/dwarf2/read.c:9626
#4  process_die (die=0x5555561eda40, cu=0x555556093190) at
../../gdb/dwarf2/read.c:8635
#5  0x00005555557eee62 in process_full_comp_unit
(pretend_language=<optimized out>, cu=0x555556093190)
at ../../gdb/dwarf2/read.c:8404
#6  process_queue (per_objfile=0x555556033980) at ../../gdb/dwarf2/read.c:7650
#7  dw2_do_instantiate_symtab (skip_partial=<optimized out>,
per_objfile=0x555556033980,
per_cu=0x5555561bbd20) at ../../gdb/dwarf2/read.c:2063
#8  dw2_instantiate_symtab (per_cu=0x5555561bbd20, per_objfile=0x555556033980,
skip_partial=<optimized out>) at ../../gdb/dwarf2/read.c:2085
#9  0x00005555557ef4fc in dw2_expand_symtabs_matching_one(dwarf2_per_cu_data *,
dwarf2_per_objfile *, gdb::function_view<bool(compunit_symtab*)>)
(per_cu=<optimized out>, per_objfile=<optimized out>,
expansion_notify=..., file_matcher=...) at ../../gdb/dwarf2/read.c:3976
#10 0x00005555557f0733 in cooked_index_functions::expand_symtabs_matching
(objfile*, gdb::function_view<bool (char const*, bool)>,
lookup_name_info const*, gdb::function_view<bool (char const*)>,
gdb::function_view<bool (compunit_symtab*)>,
enum_flags<block_search_flag_values>, domain_enum, search_domain) (
```

```
this=<optimized out>, objfile=<optimized out>, file_matcher=...,
lookup_name=<optimized out>,
symbol_matcher=..., expansion_notify=..., search_flags=...,
domain=VAR_DOMAIN, kind=ALL_DOMAIN)
at ../../gdb/dwarf2/read.c:18739
#11 0x00005555559ed4d8 in objfile::lookup_symbol (this=0x5555561bc030,
kind=<optimized out>, name=<optimized out>, domain=<optimized out>) at
../../gdb/../gdbsupport/function-view.h:298
# .
# .
# .
(gdb) c
Continuing.

Thread 1 "gdb" hit Breakpoint 2, find_pc_sect_line (pc=4224,
section=0x5555561b1548, notcurrent=0)
at ../../gdb/symtab.c:3110
3110        {
(gdb) bt
#0  find_pc_sect_line (pc=4224, section=0x5555561b1548, notcurrent=0)
at ../../gdb/symtab.c:3110
#1  0x0000555555a06fbd in find_function_start_sal_1 (func_addr=4224,
section=0x5555561b1548,
funfirstline=<optimized out>) at ../../gdb/symtab.c:3682
#2  0x0000555555a071bb in find_function_start_sal (sym=sym@entry=0x5555561f6750,
funfirstline=funfirstline@entry=true) at ../../gdb/block.h:112
#3  0x00005555559d3652 in select_source_symtab (s=<optimized out>) at
../../gdb/source.c:329
#4  0x000055555573041d in list_command (arg=<optimized out>,
from_tty=<optimized out>)
at ../../gdb/cli/cli-cmds.c:1185
#5  0x0000555555733f45 in cmd_func (cmd=<optimized out>, args=<optimized out>,
from_tty=<optimized out>)
at ../../gdb/cli/cli-decode.c:2516
#6  0x0000555555a3c257 in execute_command (p=<optimized out>,
p@entry=0x555555fae230 "list", from_tty=1)
at ../../gdb/top.c:699
#7  0x000055555580c965 in command_handler (command=0x555555fae230 "list") at
../../gdb/event-top.c:598
#8  0x000055555580cd51 in command_line_handler (rl=...) at
../../gdb/event-top.c:842
# .
# .
# .
(gdb) c
Continuing.
1       #include <stdio.h>
2
3       int main()
4       {
5               printf("Address of main is %p\n", main);
6               int a = 5;
7               int *pa = &a;
8               printf("'a' is at address '%p'.\n'a' has value '%d'\n", pa, *pa);
9
10               return 0;
(gdb) list 5
1       #include <stdio.h>
```

```
2
3       int main()
4       {
5               printf("Address of main is %p\n", main);
6               int a = 5;
7               int *pa = &a;
8               printf("'a' is at address '%p'.\n'a' has value '%d'\n", pa, *pa);
9
10               return 0;
} ...
```

We saw that both *get_debug_line_section* and *find_find_pc_sect_line* breakpoints, in that order, were hit only one time - on the first execution of *list* because *GDB* caches already read symbols.