Defect

Detect

# Linux Debugging$^4$

## Accelerated

Dmitry Vostokov

Software Diagnostics Services

Published by OpenTask, Republic of Ireland

OpenTask books and magazines are available through booksellers and distributors worldwide. For further information or comments, send requests to press@opentask.com.

A CIP catalog record for this book is available from the British Library.

ISBN-l3: 978-1912636-71-6 (Paperback)

Revision 1.00 (July 2024)

# Contents

# Exercise UD1 (WinDbg)

**Goal:** Learn how code generation parameters can influence process execution behavior.

**Elementary Diagnostics Patterns:** Crash.

**Memory Analysis Patterns:** Exception Stack Trace; NULL Pointer (Code); Constant Subtrace.

**Debugging Implementation Patterns:** Break-in; Scope; Variable Value; Type Structure; Code Breakpoint.

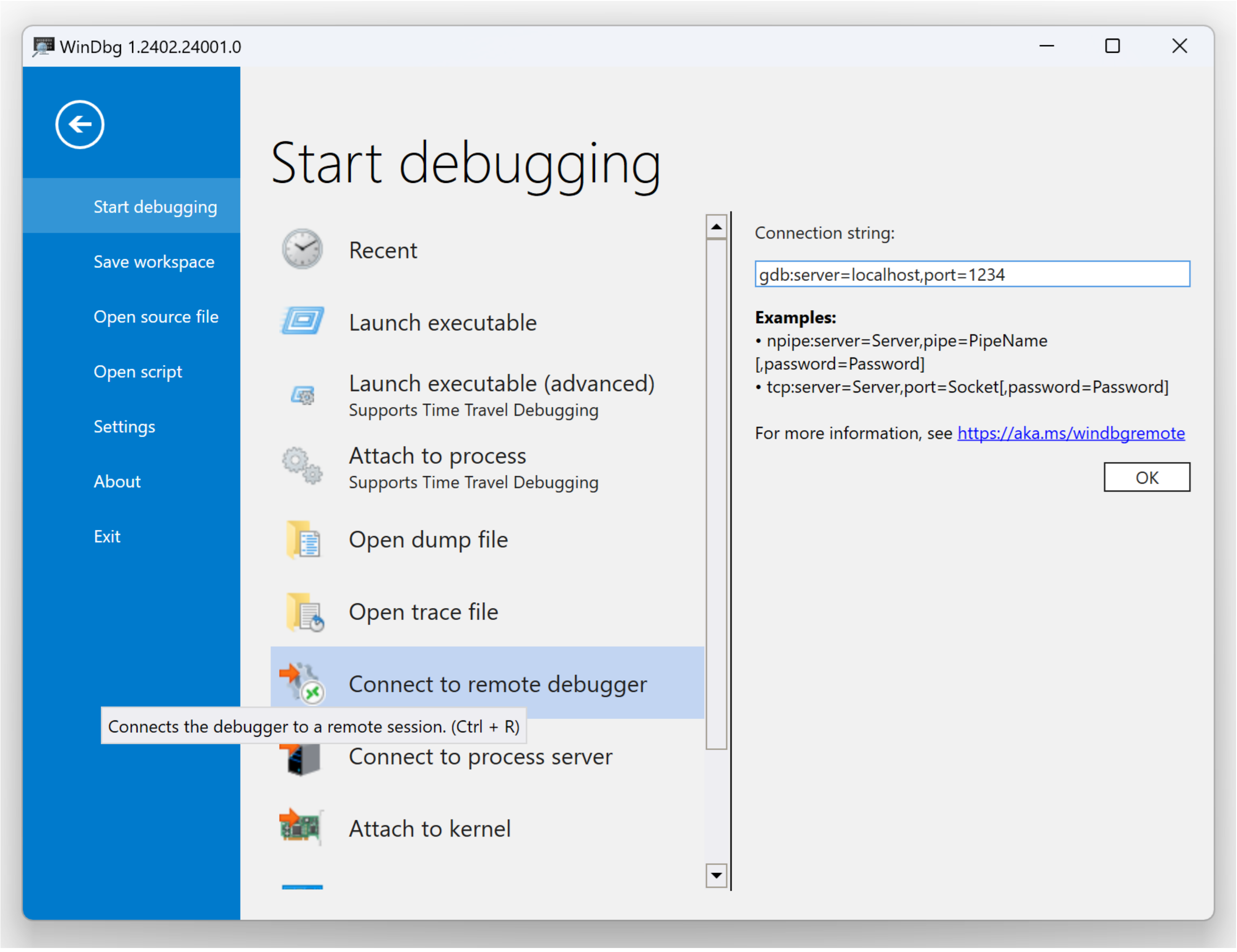
1. The source code and the *Makefile* to build executables and libraries can be found in the *ud1* directory:

```
$ git clone https://bitbucket.org/softwarediagnostics/ald4
```

2. Launch the *ud1a* executable under the *gdbserver*:

```
/mnt/c/ALD4/ud1$ LD_LIBRARY_PATH=. gdbserver localhost:1234 ud1a
Process /mnt/c/ALD4/ud1/ud1a created; pid = 3652
Listening on port 1234
```

3. Connect WinDbg to the remote debugger:



Connection string:

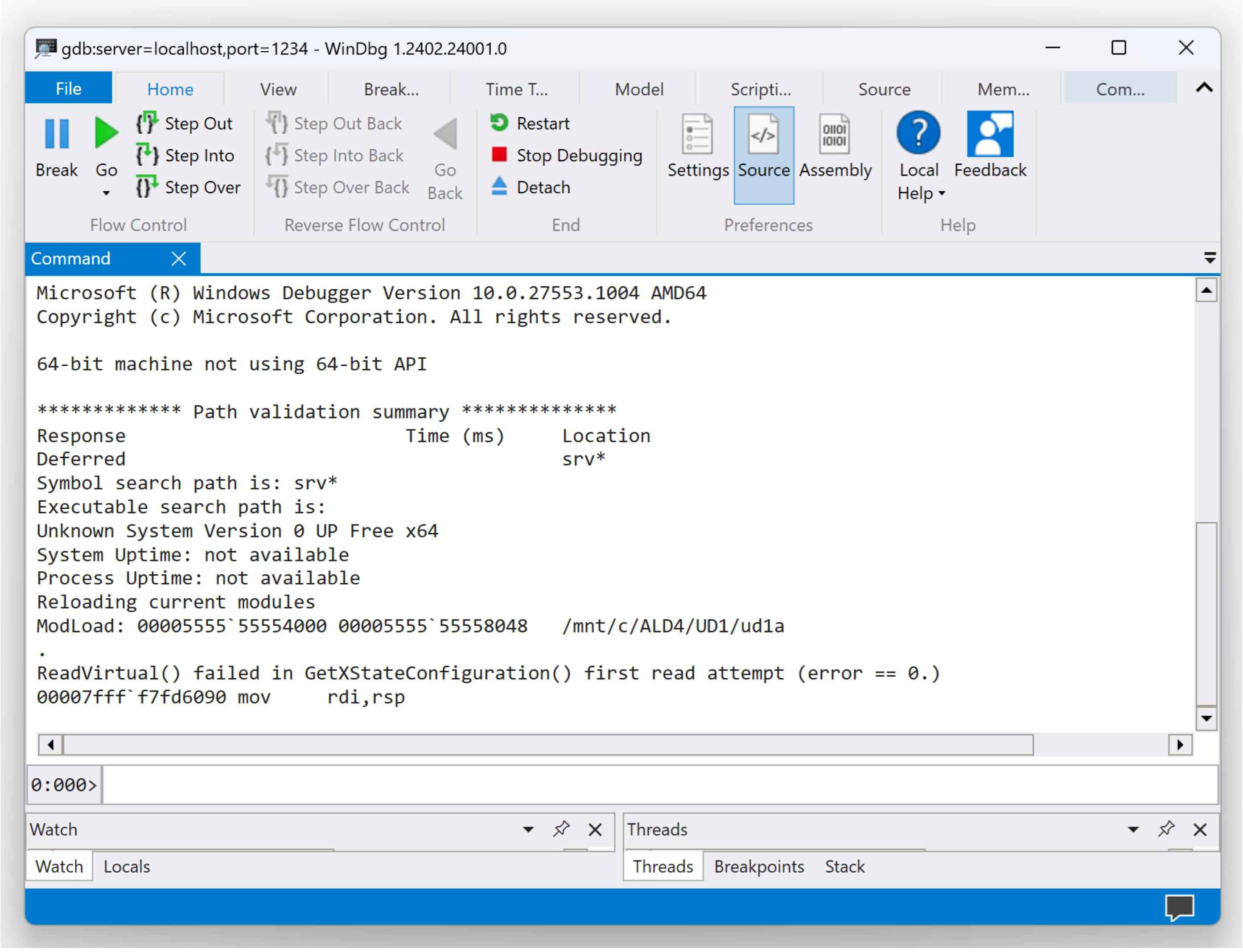
gdb:server=localhost,port=1234

**Examples:**

- npipe:server=Server,pipe=PipeName [,password=Password]
- tcp:server=Server,port=Socket[,password=Password]

For more information, see https://aka.ms/windbgremote

4. We get ready for a debugging session:



```
Microsoft (R) Windows Debugger Version 10.0.27553.1004 AMD64
Copyright (c) Microsoft Corporation. All rights reserved.

64-bit machine not using 64-bit API

************* Path validation summary **************
Response                         Time (ms)     Location
Deferred                                       srv*
Symbol search path is: srv*
Executable search path is:
Unknown System Version 0 UP Free x64
System Uptime: not available
Process Uptime: not available
Reloading current modules
ModLoad: 00005555`55554000 00005555`55558048   /mnt/c/ALD4/UD1/ud1a
.
ReadVirtual() failed in GetXStateConfiguration() first read attempt (error == 0.)
00007fff`f7fd6090 mov     rdi,rsp
```

```
0:000>
```

From now on, we only show the output from the command window unless we need another view.

```
Microsoft (R) Windows Debugger Version 10.0.27553.1004 AMD64
Copyright (c) Microsoft Corporation. All rights reserved.

64-bit machine not using 64-bit API

************* Path validation summary **************
Response                         Time (ms)     Location
Deferred                                       srv*
Symbol search path is: srv*
Executable search path is:
Unknown System Version 0 UP Free x64
System Uptime: not available
Process Uptime: not available
Reloading current modules
ModLoad: 00005555`55554000 00005555`55558048   /mnt/c/ALD4/ud1/ud1a
.
ReadVirtual() failed in GetXStateConfiguration() first read attempt (error == 0.)
00007fff`f7fd6090 mov     rdi,rsp
```

5. Open a log file (useful when the output doesn't fit into the buffer and we need to search for something):

```
0:000> .logopen C:\ALD4\ud1a.log
Opened log file 'C:\ALD4\ud1a.log'
```

6. The **lm** command lists loaded modules and their addresses (it also shows whether symbols files are loaded):

```
0:000> lm
start             end                 module name
00005555`55554000 00005555`55558048   ud1a       (deferred)
```

7. We continue process execution using the **g** command until we get a segmentation fault:

```
0:000> g
ModLoad: 00007fff`f7fd3000 00007fff`f7fd3000   linux-vdso.so.1
ModLoad: 00007fff`f7fc8000 00007fff`f7fcc048   ./libwindows.so
ModLoad: 00007fff`f7dfd000 00007fff`f7fbc800   /lib/x86_64-linux-gnu/libc.so.6
ModLoad: 00007fff`f7fd5000 00007fff`f7ffe190   /lib64/ld-linux-x86-64.so.2
(e44.e44): Signal SIGSEGV (Segmentation fault) code SEGV_MAPERR (Address not mapped to object)
at 0x5555 originating from PID 70fb
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
Unable to load image ./libwindows.so, Win32 error 0n2
*** WARNING: Unable to verify timestamp for ./libwindows.so
00000000`00005555 ???

0:000> lm
start             end                 module name
00005555`55554000 00005555`55558048   ud1a      T (service symbols: DWARF Private Symbols)
C:\Users\dmitr\AppData\Local\Temp\srcD37D.tmp
00007fff`f7dfd000 00007fff`f7fbc800   libc_so   T (service symbols: ELF Export Symbols)
C:\Users\dmitr\AppData\Local\Temp\srcD880.tmp
00007fff`f7fc8000 00007fff`f7fcc048   libwindows T (service symbols: DWARF Private Symbols)
C:\Users\dmitr\AppData\Local\Temp\srcD12B.tmp
00007fff`f7fd3000 00007fff`f7fd3000   linux_vdso_so    (deferred)
00007fff`f7fd5000 00007fff`f7ffe190   ld_linux_x86_64_so    (deferred)
```

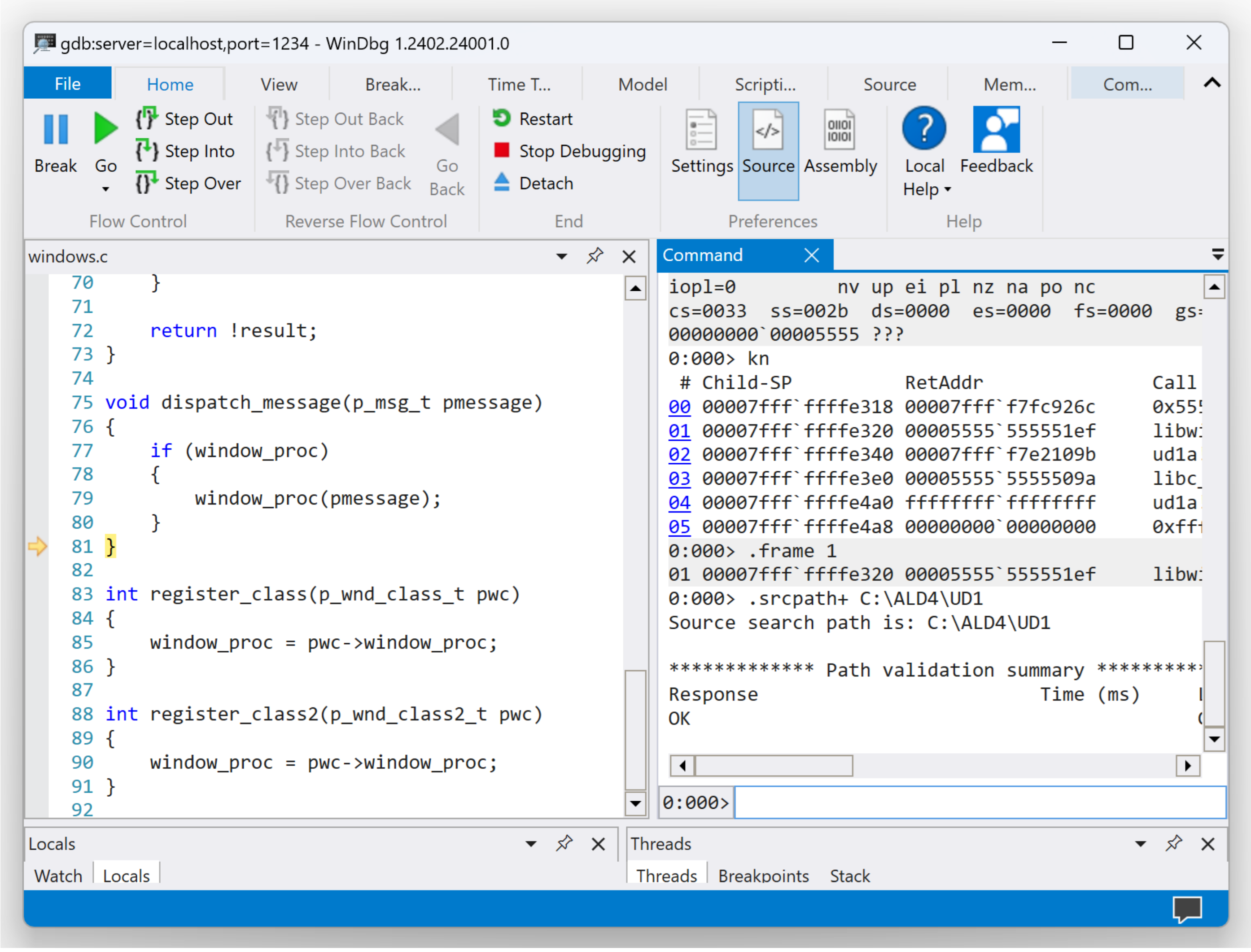
There, we see that the crash happens in the **libwindows** module with the following CPU state:

```
0:000> k
 # Child-SP          RetAddr               Call Site
00 00007fff`ffffe318 00007fff`f7fc926c     0x5555
01 00007fff`ffffe320 00005555`555551ef     libwindows!dispatch_message+0x28
[/mnt/c/ALD4/ud1/windows.c @ 81]
02 00007fff`ffffe340 00007fff`f7e2109b     ud1a!main+0x88 [/mnt/c/ALD4/ud1/ud1.c @ 36]
03 00007fff`ffffe3e0 00005555`5555509a     libc_so!_libc_start_main+0xeb
04 00007fff`ffffe4a0 ffffffff`ffffffff     ud1a!start+0x2a
05 00007fff`ffffe4a8 00000000`00000000     0xffffffff`ffffffff

0:000> r
rax=0000000000005555 rbx=0000000000000000 rcx=00007ffff7ec3594
rdx=00007fffffffe3a0 rsi=00007fffffffe2a0 rdi=00007fffffffe3a0
rip=0000000000005555 rsp=00007fffffffe318 rbp=00007fffffffe330
 r8=00007ffff7fb8d80  r9=00007ffff7fb8d80 r10=fffffffffffff429
r11=00007ffff7fc9244 r12=0000555555555070 r13=00007fffffffe4b0
r14=0000000000000000 r15=0000000000000000
iopl=0         nv up ei pl nz na po nc
cs=0033  ss=002b  ds=0000  es=0000  fs=0000  gs=0000             efl=00010206
00000000`00005555 ???
```

8. We switch to the *libwindows* thread stack frame #1 and set the source code location:

```
0:000> kn
 # Child-SP          RetAddr               Call Site
00 00007fff`ffffe318 00007fff`f7fc926c     0x5555
01 00007fff`ffffe320 00005555`555551ef     libwindows!dispatch_message+0x28
[/mnt/c/ALD4/ud1/windows.c @ 81]
02 00007fff`ffffe340 00007fff`f7e2109b     ud1a!main+0x88 [/mnt/c/ALD4/ud1/ud1.c @ 36]
03 00007fff`ffffe3e0 00005555`5555509a     libc_so!_libc_start_main+0xeb
04 00007fff`ffffe4a0 ffffffff`ffffffff     ud1a!start+0x2a
05 00007fff`ffffe4a8 00000000`00000000     0xffffffff`ffffffff

0:000> .frame 1
01 00007fff`ffffe320 00005555`555551ef     libwindows!dispatch_message+0x28
[/mnt/c/ALD4/ud1/windows.c @ 81]

0:000> .srcpath+ C:\ALD4\ud1
Source search path is: SRV*;C:\ALD4\ud1

************* Path validation summary **************
Response                         Time (ms)     Location
Deferred                                       SRV*
OK                                             C:\ALD4\ud1
```

**Note:** We see a source code window immediately to the left of the command window:



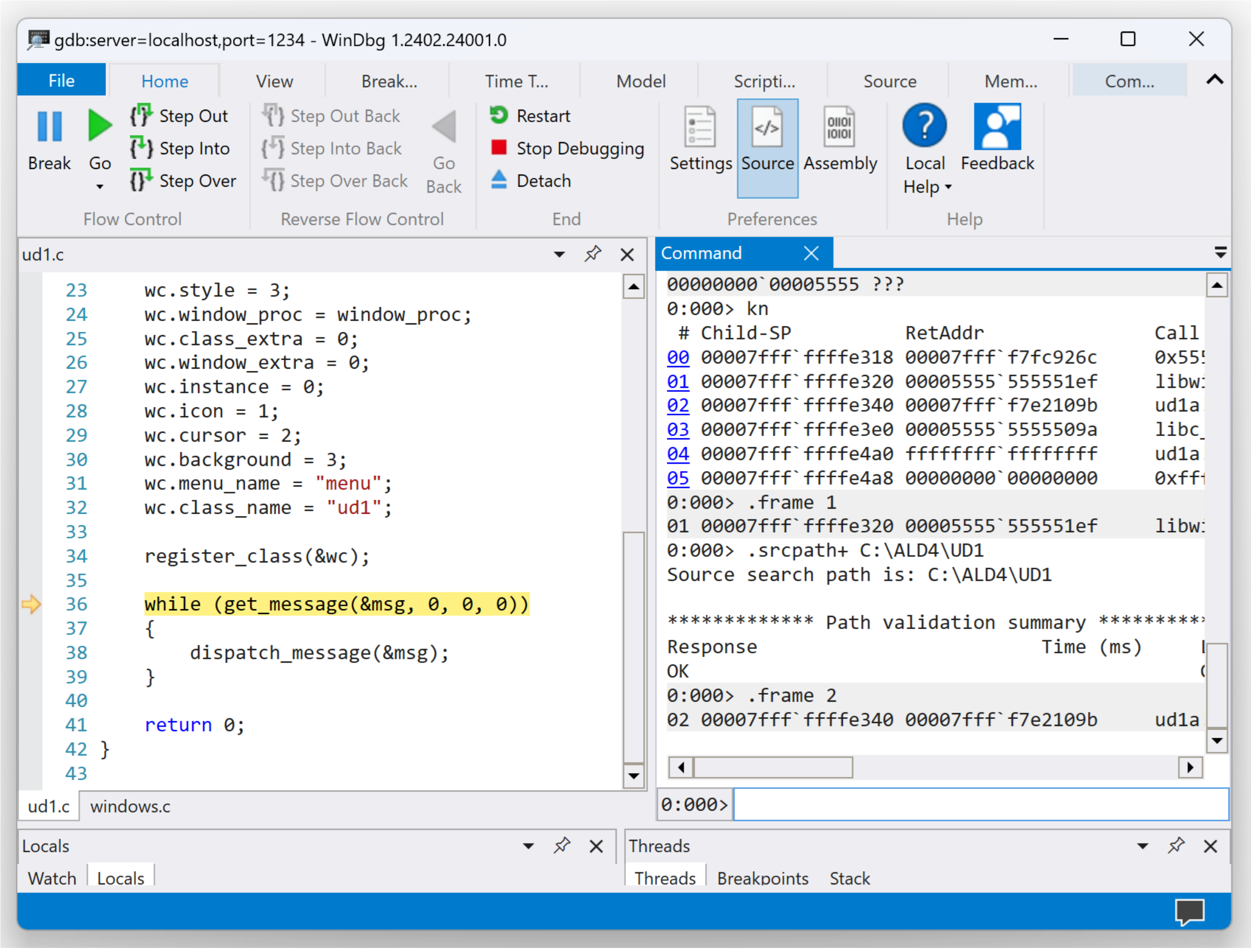
```
70     }
71
72     return !result;
73 }
74
75 void dispatch_message(p_msg_t pmessage)
76 {
77     if (window_proc)
78     {
79         window_proc(pmessage);
80     }
81 }
82
83 int register_class(p_wnd_class_t pwc)
84 {
85     window_proc = pwc->window_proc;
86 }
87
88 int register_class2(p_wnd_class2_t pwc)
89 {
90     window_proc = pwc->window_proc;
91 }
92
```

```
iopl=0         nv up ei pl nz na po nc
cs=0033  ss=002b  ds=0000  es=0000  fs=0000  gs:
00000000`00005555 ???
0:000> kn
 # Child-SP          RetAddr               Call
00 00007fff`ffffe318 00007fff`f7fc926c     0x555
01 00007fff`ffffe320 00005555`555551ef     libw:
02 00007fff`ffffe340 00007fff`f7e2109b     ud1a
03 00007fff`ffffe3e0 00005555`5555509a     libc_
04 00007fff`ffffe4a0 ffffffff`ffffffff     ud1a
05 00007fff`ffffe4a8 00000000`00000000     0xff
0:000> .frame 1
01 00007fff`ffffe320 00005555`555551ef     libw:
0:000> .srcpath+ C:\ALD4\UD1
Source search path is: C:\ALD4\UD1

************* Path validation summary *********
Response                         Time (ms)     
OK
```

```
0:000>
```

9. We see that the *window_proc* pointer is invalid, so we need to investigate when it is set in the *register_class* function below. First, we set the next frame where the *dispatch_message* was called:

```
0:000> dp libwindows!window_proc L1
00007fff`f7fcc040  00000000`00005555

0:000> kn
 # Child-SP          RetAddr               Call Site
00 00007fff`ffffe318 00007fff`f7fc926c     0x5555
01 00007fff`ffffe320 00005555`555551ef     libwindows!dispatch_message+0x28
[/mnt/c/ALD4/ud1/windows.c @ 81]
02 00007fff`ffffe340 00007fff`f7e2109b     ud1a!main+0x88 [/mnt/c/ALD4/ud1/ud1.c @ 36]
03 00007fff`ffffe3e0 00005555`5555509a     libc_so!_libc_start_main+0xeb
04 00007fff`ffffe4a0 ffffffff`ffffffff     ud1a!start+0x2a
05 00007fff`ffffe4a8 00000000`00000000     0xffffffff`ffffffff

0:000> .frame 2
02 00007fff`ffffe340 00007fff`f7e2109b     ud1a!main+0x88 [/mnt/c/ALD4/ud1/ud1.c @ 36]
```

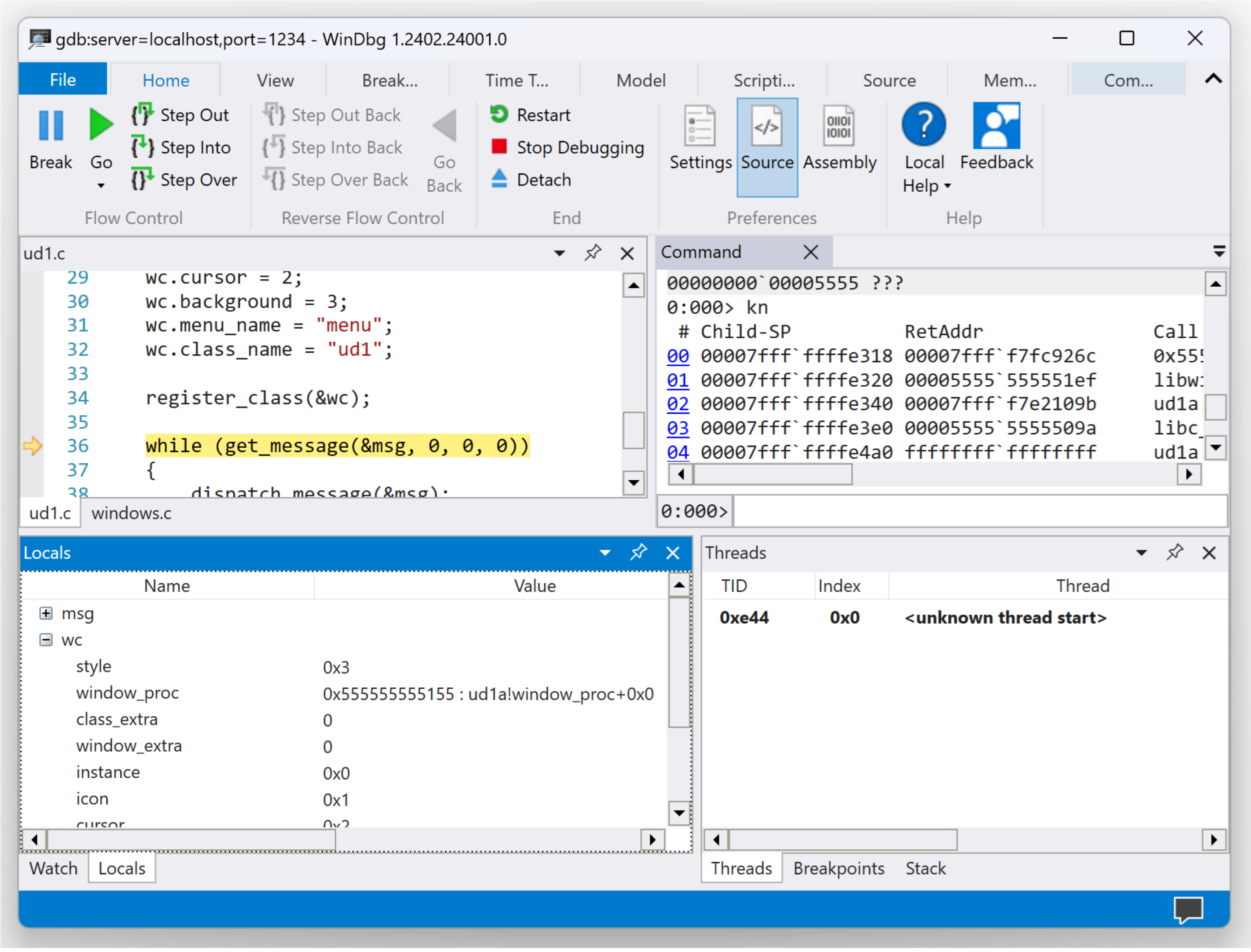


```
    wc.style = 3;
    wc.window_proc = window_proc;
    wc.class_extra = 0;
    wc.window_extra = 0;
    wc.instance = 0;
    wc.icon = 1;
    wc.cursor = 2;
    wc.background = 3;
    wc.menu_name = "menu";
    wc.class_name = "ud1";

    register_class(&wc);

    while (get_message(&msg, 0, 0, 0))
    {
        dispatch_message(&msg);
    }

    return 0;
}

```

```
00000000`00005555 ???
0:000> kn
 # Child-SP          RetAddr               Call
00 00007fff`ffffe318 00007fff`f7fc926c     0x555
01 00007fff`ffffe320 00005555`555551ef     libw
02 00007fff`ffffe340 00007fff`f7e2109b     ud1a
03 00007fff`ffffe3e0 00005555`5555509a     libc_
04 00007fff`ffffe4a0 ffffffff`ffffffff     ud1a
05 00007fff`ffffe4a8 00000000`00000000     0xff
0:000> .frame 1
01 00007fff`ffffe320 00005555`555551ef     libw
0:000> .srcpath+ C:\ALD4\UD1
Source search path is: C:\ALD4\UD1

************ Path validation summary *********
Response                         Time (ms)     I
OK                                             C
0:000> .frame 2
02 00007fff`ffffe340 00007fff`f7e2109b     ud1a
```

```
0:000>
```

10. We can now expand local structures in the *Locals* window (for example, *wc*):



```
    wc.cursor = 2;
    wc.background = 3;
    wc.menu_name = "menu";
    wc.class_name = "ud1";

    register_class(&wc);

    while (get_message(&msg, 0, 0, 0))
    {
        dispatch_message(&msg);
```

```
00000000`00005555 ???
0:000> kn
 # Child-SP          RetAddr               Call
00 00007fff`ffffe318 00007fff`f7fc926c     0x55
01 00007fff`ffffe320 00005555`555551ef     libw
02 00007fff`ffffe340 00007fff`f7e2109b     ud1a
03 00007fff`ffffe3e0 00005555`5555509a     libc_
04 00007fff`ffffe4a0 ffffffff`ffffffff     ud1a
0:000>
```

| Name | Value |
|---|---|
| ⊞ msg | |
| ⊟ wc | |
| style | 0x3 |
| window_proc | 0x555555555155 : ud1a!window_proc+0x0 |
| class_extra | 0 |
| window_extra | 0 |
| instance | 0x0 |
| icon | 0x1 |
| cursor | 0x2 |

| TID | Index | Thread |
|---|---|---|
| **0xe44** | **0x0** | **<unknown thread start>** |

We can also dump this variable using type information:

```
0:000> dt wc
Local var @ 0x7fffffffe350 Type wnd_class_t
   +0x000 style            : 3
   +0x004 window_proc      : 0x00005555`55555155     void  ud1a!window_proc+0
   +0x00c class_extra      : 0n0
   +0x010 window_extra     : 0n0
   +0x014 instance         : 0
   +0x01c icon             : 1
   +0x024 cursor           : 2
   +0x02c background       : 3
   +0x034 menu_name        : 0x00005555`55556004  "menu"
   +0x03c class_name       : 0x00005555`55556009  "ud1"
```

11. We need to make sure that *libwindows* is loaded before we put a breakpoint on the *register_class* function. To do that, we determine the *main* function address to set the breakpoint there first once we restart the debugged process. Then, on break-in we set our *register_class* breakpoint since the library is already loaded.

```
0:000> ln main
Browse module
Set bu breakpoint

 [/mnt/c/ALD4/ud1/ud1.c @ 19] (00005555`55555167)   ud1a!main
Exact matches:
    ud1a!main (int, char **)
```

12. Now, we finish the process (the **g** command) and see WinDbg disconnected. Then we start the **gdbserver** again and reattach WinDbg to the remote debugger.

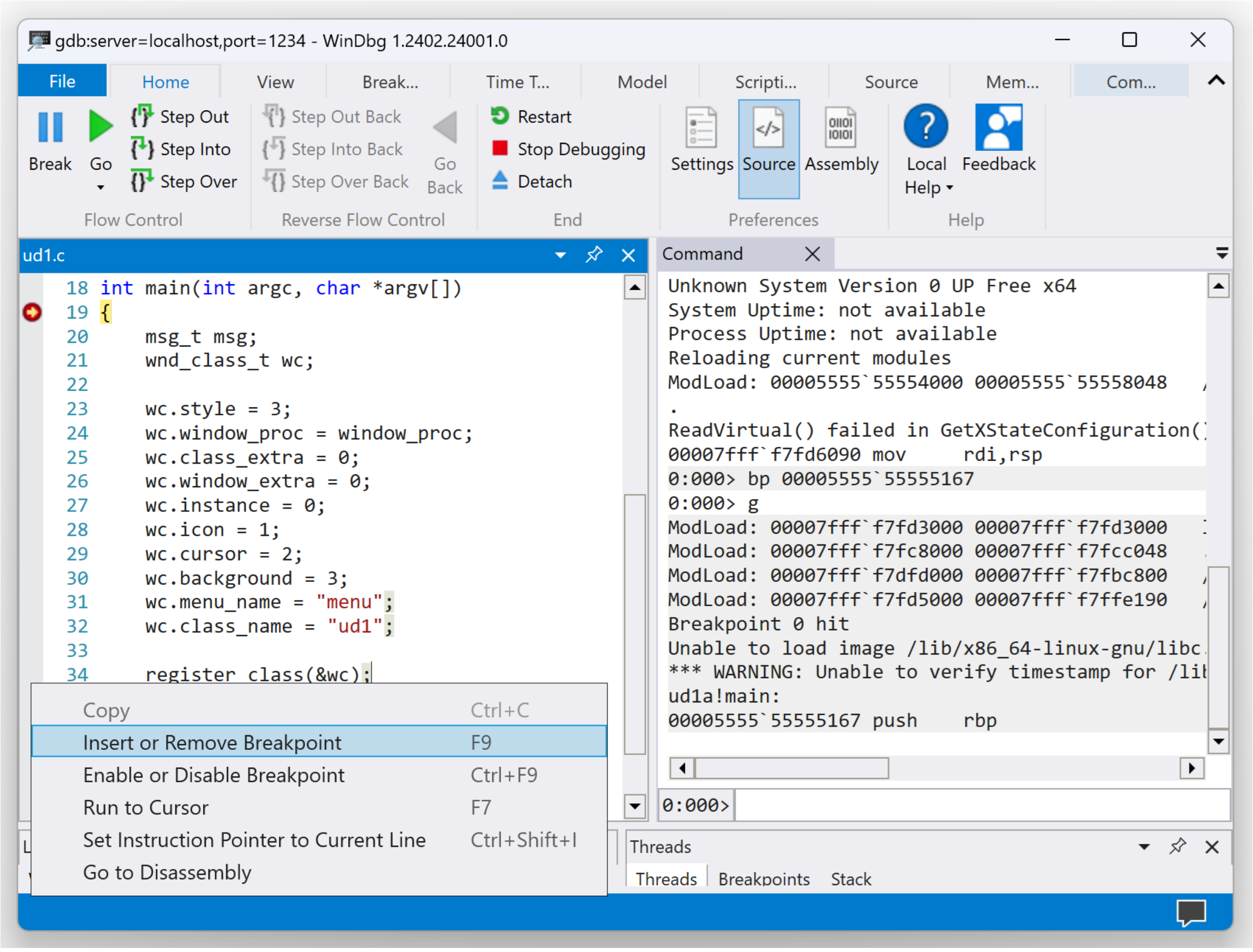
```
/mnt/c/ALD4/ud1$ LD_LIBRARY_PATH=. gdbserver localhost:1234 ud1a
Process /mnt/c/ALD4/ud1/ud1a created; pid = 34
Listening on port 1234
```

```
Microsoft (R) Windows Debugger Version 10.0.27553.1004 AMD64
Copyright (c) Microsoft Corporation. All rights reserved.

64-bit machine not using 64-bit API

************* Path validation summary **************
Response                         Time (ms)     Location
Deferred                                       srv*
Symbol search path is: srv*
Executable search path is:
Unknown System Version 0 UP Free x64
System Uptime: not available
Process Uptime: not available
Reloading current modules
ModLoad: 00005555`55554000 00005555`55558048   /mnt/c/ALD4/ud1/ud1a
.
ReadVirtual() failed in GetXStateConfiguration() first read attempt (error == 0.)
00007fff`f7fd6090 mov     rdi,rsp
```

13. We put a breakpoint on the main function address we determined previously and resume execution until it is hit:

```
0:000> bp 00005555`55555167

0:000> g
ModLoad: 00007fff`f7fd3000 00007fff`f7fd3000   linux-vdso.so.1
ModLoad: 00007fff`f7fc8000 00007fff`f7fcc048   ./libwindows.so
ModLoad: 00007fff`f7dfd000 00007fff`f7fbc800   /lib/x86_64-linux-gnu/libc.so.6
ModLoad: 00007fff`f7fd5000 00007fff`f7ffe190   /lib64/ld-linux-x86-64.so.2
Breakpoint 0 hit
Unable to load image /lib/x86_64-linux-gnu/libc.so.6, Win32 error 0n2
*** WARNING: Unable to verify timestamp for /lib/x86_64-linux-gnu/libc.so.6
ud1a!main:
00005555`55555167 push    rbp
```

14. We now put a breakpoint on the call to the *register_class* function (F9) and resume execution:



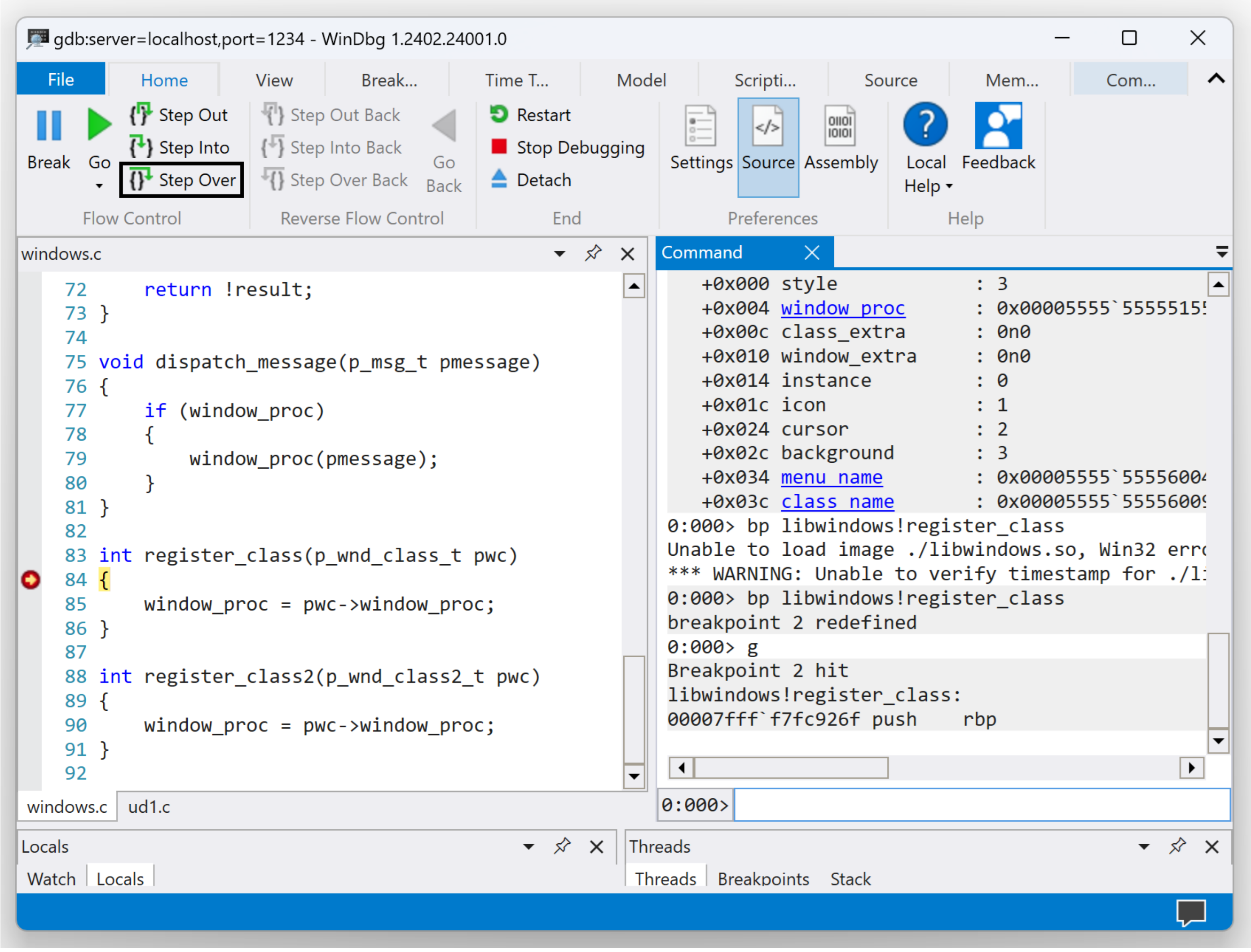
```
int main(int argc, char *argv[])
{
    msg_t msg;
    wnd_class_t wc;

    wc.style = 3;
    wc.window_proc = window_proc;
    wc.class_extra = 0;
    wc.window_extra = 0;
    wc.instance = 0;
    wc.icon = 1;
    wc.cursor = 2;
    wc.background = 3;
    wc.menu_name = "menu";
    wc.class_name = "ud1";

    register_class(&wc);
```

```
Unknown System Version 0 UP Free x64
System Uptime: not available
Process Uptime: not available
Reloading current modules
ModLoad: 00005555`55554000 00005555`55558048
.
ReadVirtual() failed in GetXStateConfiguration(
00007fff`f7fd6090 mov     rdi,rsp
0:000> bp 00005555`55555167
0:000> g
ModLoad: 00007fff`f7fd3000 00007fff`f7fd3000
ModLoad: 00007fff`f7fc8000 00007fff`f7fcc048
ModLoad: 00007fff`f7fdf000 00007fff`f7fbc800
ModLoad: 00007fff`f7fd5000 00007fff`f7ffe190
Breakpoint 0 hit
Unable to load image /lib/x86_64-linux-gnu/libc
*** WARNING: Unable to verify timestamp for /lit
ud1a!main:
00005555`55555167 push    rbp
0:000>
```

```
0:000> g
Breakpoint 1 hit
ud1a!main+0x6e:
00005555`555551d5 lea     rax,[rbp-80h]

0:000> dt ud1a!wc
Local var @ 0x7fffffffe350 Type wnd_class_t
   +0x000 style            : 3
   +0x004 window_proc      : 0x00005555`55555155     void  ud1a!window_proc+0
   +0x00c class_extra      : 0n0
   +0x010 window_extra     : 0n0
   +0x014 instance         : 0
   +0x01c icon             : 1
   +0x024 cursor           : 2
   +0x02c background       : 3
   +0x034 menu_name        : 0x00005555`55556004  "menu"
   +0x03c class_name       : 0x00005555`55556009  "ud1"
```

15. Then, we put a breakpoint inside the *register_class* function and resume execution:

```
0:000> bp libwindows!register_class
Unable to load image ./libwindows.so, Win32 error 0n2
*** WARNING: Unable to verify timestamp for ./libwindows.so

0:000> bp libwindows!register_class
breakpoint 2 redefined

0:000> g
Breakpoint 2 hit
libwindows!register_class:
00007fff`f7fc926f push    rbp
```



```
    return !result;
}

void dispatch_message(p_msg_t pmessage)
{
    if (window_proc)
    {
        window_proc(pmessage);
    }
}

int register_class(p_wnd_class_t pwc)
{
    window_proc = pwc->window_proc;
}

int register_class2(p_wnd_class2_t pwc)
{
    window_proc = pwc->window_proc;
}
```

```
   +0x000 style            : 3
   +0x004 window_proc      : 0x00005555`5555515
   +0x00c class_extra      : 0n0
   +0x010 window_extra     : 0n0
   +0x014 instance         : 0
   +0x01c icon             : 1
   +0x024 cursor           : 2
   +0x02c background       : 3
   +0x034 menu_name        : 0x00005555`5555600
   +0x03c class_name       : 0x00005555`5555600
0:000> bp libwindows!register_class
Unable to load image ./libwindows.so, Win32 err
*** WARNING: Unable to verify timestamp for ./l
0:000> bp libwindows!register_class
breakpoint 2 redefined
0:000> g
Breakpoint 2 hit
libwindows!register_class:
00007fff`f7fc926f push    rbp
```

16. Do one step over to have the parameter initialized and inspect it:

```
0:000> p
libwindows!register_class+0x8:
00007fff`f7fc9277 mov     rax,qword ptr [rbp-8] ss:00007fff`ffffe328=00007fffffffe350
```

```
0:000> dt pwc
Local var @ 0x7fffffffe328 Type p_wnd_class_t
0x00007fff`ffffe350
   +0x000 style            : 3
   +0x008 window_proc      : 0x00000000`00005555     void  +5555
   +0x010 class_extra      : 0n0
   +0x014 window_extra     : 0n0
   +0x018 instance         : 0x00000001`00000000
   +0x020 icon             : 0x00000002`00000000
   +0x028 cursor           : 0x00000003`00000000
   +0x030 background       : 0x55556004`00000000
   +0x038 menu_name        : 0x55556009`00005555  "--- memory read error at address
0x55556009`00005555 ---"
   +0x040 class_name       : 0x00000000`00005555  "--- memory read error at address
0x00000000`00005555 ---"

0:000> dt libwindows!p_wnd_class_t
Ptr64    +0x000 style            : Uint4B
   +0x008 window_proc      : Ptr64     void
   +0x010 class_extra      : Int4B
   +0x014 window_extra     : Int4B
   +0x018 instance         : Uint8B
   +0x020 icon             : Uint8B
   +0x028 cursor           : Uint8B
   +0x030 background       : Uint8B
   +0x038 menu_name        : Ptr64 Char
   +0x040 class_name       : Ptr64 Char
```

17. But if we look at the *ud1a* structure variant, we see its members have different offsets:

```
0:000> dt ud1a!wnd_class_t
   +0x000 style            : Uint4B
   +0x004 window_proc      : Ptr64     void
   +0x00c class_extra      : Int4B
   +0x010 window_extra     : Int4B
   +0x014 instance         : Uint8B
   +0x01c icon             : Uint8B
   +0x024 cursor           : Uint8B
   +0x02c background       : Uint8B
   +0x034 menu_name        : Ptr64 Char
   +0x03c class_name       : Ptr64 Char
```

18. These discrepancies explain the crash. Looking at the *Makefile*, we can see that *ud1a* was compiled with the *-fpack-struct* setting. The *ud1b* executable was compiled without it and runs fine. Also, the problem was coincidentally fixed without changing alignment by using a different, bigger *wnd_class2_t* structure in the *ud1c* executable that adds another 32-bit field that makes both alignments identical.

19. We continue execution (**g**) to have the remote process finished and then close WinDbg.

# Exercise UD1 (GDB)

**Goal:** Learn how code generation parameters can influence process execution behavior.

**Elementary Diagnostics Patterns:** Crash.

**Memory Analysis Patterns:** Exception Stack Trace; NULL Pointer (Code); Constant Subtrace.

**Debugging Implementation Patterns:** Break-in; Scope; Variable Value; Type Structure; Code Breakpoint.

1. The source code and the *Makefile* to build executables and libraries can be found in the *ud1* directory:

```
$ git clone https://bitbucket.org/softwarediagnostics/ald4
```

2. When we launch the *ud1a* executable, it crashes:

```
/mnt/c/ALD4/ud1$ LD_LIBRARY_PATH=. ./ud1a
Segmentation fault
```

3. We run the executable under GDB until it shows a segmentation fault:

```
/mnt/c/ALD4/ud1$ LD_LIBRARY_PATH=. gdb ./ud1a
GNU gdb (Debian 8.2.1-2+b3) 8.2.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./ud1a...done.

(gdb) r
Starting program: /mnt/c/ALD4/ud1/ud1a

Program received signal SIGSEGV, Segmentation fault.
0x0000000000005555 in ?? ()
```

4. The **info proc mappings** and **info sharedlibrary** commands list loaded modules and their addresses and show if symbols are available:

```
(gdb) info proc mappings
process 14581
Mapped address spaces:

          Start Addr           End Addr       Size     Offset objfile
      0x555555554000     0x555555555000     0x1000        0x0 /mnt/c/ALD4/ud1/ud1a
      0x555555555000     0x555555556000     0x1000     0x1000 /mnt/c/ALD4/ud1/ud1a
      0x555555556000     0x555555557000     0x1000     0x2000 /mnt/c/ALD4/ud1/ud1a
      0x555555557000     0x555555558000     0x1000     0x2000 /mnt/c/ALD4/ud1/ud1a
      0x555555558000     0x555555559000     0x1000     0x3000 /mnt/c/ALD4/ud1/ud1a
      0x7ffff7dfa000     0x7ffff7dfd000     0x3000        0x0
      0x7ffff7dfd000     0x7ffff7e1f000    0x22000        0x0 /lib/x86_64-linux-gnu/libc-2.28.so
      0x7ffff7e1f000     0x7ffff7f66000   0x147000    0x22000 /lib/x86_64-linux-gnu/libc-2.28.so
      0x7ffff7f66000     0x7ffff7fb2000    0x4c000   0x169000 /lib/x86_64-linux-gnu/libc-2.28.so
      0x7ffff7fb2000     0x7ffff7fb3000     0x1000   0x1b5000 /lib/x86_64-linux-gnu/libc-2.28.so
      0x7ffff7fb3000     0x7ffff7fb7000     0x4000   0x1b5000 /lib/x86_64-linux-gnu/libc-2.28.so
      0x7ffff7fb7000     0x7ffff7fb9000     0x2000   0x1b9000 /lib/x86_64-linux-gnu/libc-2.28.so
      0x7ffff7fb9000     0x7ffff7fbd000     0x4000        0x0
      0x7ffff7fc8000     0x7ffff7fc9000     0x1000        0x0 /mnt/c/ALD4/ud1/libwindows.so
      0x7ffff7fc9000     0x7ffff7fca000     0x1000     0x1000 /mnt/c/ALD4/ud1/libwindows.so
      0x7ffff7fca000     0x7ffff7fcb000     0x1000     0x2000 /mnt/c/ALD4/ud1/libwindows.so
      0x7ffff7fcb000     0x7ffff7fcc000     0x1000     0x2000 /mnt/c/ALD4/ud1/libwindows.so
      0x7ffff7fcc000     0x7ffff7fcd000     0x1000     0x3000 /mnt/c/ALD4/ud1/libwindows.so
      0x7ffff7fcd000     0x7ffff7fcf000     0x2000        0x0
      0x7ffff7fcf000     0x7ffff7fd3000     0x4000        0x0 [vvar]
      0x7ffff7fd3000     0x7ffff7fd5000     0x2000        0x0 [vdso]
      0x7ffff7fd5000     0x7ffff7fd6000     0x1000        0x0 /lib/x86_64-linux-gnu/ld-2.28.so
      0x7ffff7fd6000     0x7ffff7ff4000    0x1e000     0x1000 /lib/x86_64-linux-gnu/ld-2.28.so
      0x7ffff7ff4000     0x7ffff7ffc000     0x8000    0x1f000 /lib/x86_64-linux-gnu/ld-2.28.so
      0x7ffff7ffc000     0x7ffff7ffd000     0x1000    0x26000 /lib/x86_64-linux-gnu/ld-2.28.so
--Type <RET> for more, q to quit, c to continue without paging--
      0x7ffff7ffd000     0x7ffff7ffe000     0x1000    0x27000 /lib/x86_64-linux-gnu/ld-2.28.so
      0x7ffff7ffe000     0x7ffff7fff000     0x1000        0x0
      0x7ffffffde000     0x7ffffffff000    0x21000        0x0 [stack]
```

```
(gdb) info sharedlibrary
From                To                  Syms Read   Shared Object Library
0x00007ffff7fd6090  0x00007ffff7ff3b50  Yes         /lib64/ld-linux-x86-64.so.2
0x00007ffff7fc9050  0x00007ffff7fc92a3  Yes         ./libwindows.so
0x00007ffff7e1f320  0x00007ffff7f6514b  Yes         /lib/x86_64-linux-gnu/libc.so.6
```

5. We see that the crash happens in the **libwindows** module with the following CPU state:

```
(gdb) bt
#0  0x0000000000005555 in ?? ()
#1  0x00007ffff7fc926c in dispatch_message (pmessage=0x7fffffffe380) at windows.c:76
#2  0x00005555555551ef in main (argc=1, argv=0x7fffffffe498) at ud1.c:35
```

```
(gdb) info r
rax            0x5555              21845
rbx            0x0                 0
rcx            0x7ffff7ec3594      140737352840596
rdx            0x7fffffffe3b0      140737488348080
rsi            0x7fffffffe2b0      140737488347824
rdi            0x7fffffffe3b0      140737488348080
rbp            0x7fffffffe340      0x7fffffffe340
rsp            0x7fffffffe328      0x7fffffffe328
r8             0x7ffff7fb8d80      140737353846144
r9             0x7ffff7fb8d80      140737353846144
r10            0xfffffffffffff429  -3031
r11            0x7ffff7fc9244      140737353912900
r12            0x555555555070      93824992235632
r13            0x7fffffffe4c0      140737488348352
r14            0x0                 0
r15            0x0                 0
rip            0x5555              0x5555
eflags         0x10206             [ PF IF RF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
```

```
fs             0x0                 0
gs             0x0                 0
```

6. We switch to stack frame #1 and check the source code:

```
(gdb) frame 1
#1  0x00007ffff7fc926c in dispatch_message (pmessage=0x7fffffffe3b0) at windows.c:76
76                window_proc(pmessage);

(gdb) list
71
72      void dispatch_message(p_msg_t pmessage)
73      {
74          if (window_proc)
75          {
76              window_proc(pmessage);
77          }
78      }
79
80      int register_class(p_wnd_class_t pwc)

(gdb) p window_proc
$1 = (void (*)(p_msg_t)) 0x5555

(gdb) list 83
78          {
79              window_proc(pmessage);
80          }
81      }
82
83      int register_class(p_wnd_class_t pwc)
84      {
85          window_proc = pwc->window_proc;
86      }
87
```

7. We see that the *window_proc* pointer is invalid, so we need to investigate when it is set in the *register_class* function below. First, we set the next frame where the *dispatch_message* was called:

```
(gdb) bt
#0  0x0000000000005555 in ?? ()
#1  0x00007ffff7fc926c in dispatch_message (pmessage=0x7fffffffe3b0) at windows.c:76
#2  0x00005555555551ef in main (argc=1, argv=0x7fffffffe4c8) at ud1.c:35

(gdb) frame 2
#2  0x00005555555551ef in main (argc=1, argv=0x7fffffffe4c8) at ud1.c:35
35                  dispatch_message(&msg);

(gdb) list
30
31              register_class(&wc);
32
33              while (get_message(&msg, 0, 0, 0))
34              {
35                  dispatch_message(&msg);
36              }
37
38          return 0;
```

```
39      }
```

8. We can now check local variables and their structures (for example, *wc*):

```
(gdb) info locals
msg = {hwnd = 0, message = 275, param1 = 1, param2 = 140737353912581, time = 72844112, pt = {x
= 156, y = 327},
  priv = 0}
wc = {style = 3, window_proc = 0x555555555155 <window_proc>, class_extra = 0, window_extra = 0,
instance = 0,
  icon = 1, cursor = 2, background = 3, menu_name = 0x555555556004 "menu", class_name =
0x555555556009 "ud1"}

(gdb) ptype /o wc
type = struct {
/*    0      |     4 */    uint32_t style;
/*    4      |     8 */    void (*window_proc)(p_msg_t);
/*   12      |     4 */    int32_t class_extra;
/*   16      |     4 */    int32_t window_extra;
/*   20      |     8 */    uint64_t instance;
/*   28      |     8 */    uint64_t icon;
/*   36      |     8 */    uint64_t cursor;
/*   44      |     8 */    uint64_t background;
/*   52      |     8 */    char *menu_name;
/*   60      |     8 */    char *class_name;

                           /* total size (bytes):   68 */
                         }
```

9. We put a breakpoint on the *main* function and resume execution until it is hit:

```
(gdb) c
Continuing.

Program terminated with signal SIGSEGV, Segmentation fault.
The program no longer exists.

(gdb) break main
Breakpoint 1 at 0x55555555517f: file ud1.c, line 20.

(gdb) r
Starting program: /mnt/c/ALD4/ud1/ud1a

Breakpoint 1, main (argc=1, argv=0x7fffffffe4c8) at ud1.c:20
20          wc.style = 3;
```

10. We now put a breakpoint on the call to the *register_class* function and resume execution to inspect the passed value of the *wc* structure:

```
(gdb) list 27, 40
27          wc.background = 3;
28          wc.menu_name = "menu";
29          wc.class_name = "ud1";
30
31              register_class(&wc);
32
33              while (get_message(&msg, 0, 0, 0))
34              {
```

```
35                  dispatch_message(&msg);
36              }
37
38          return 0;
39      }

(gdb) break ud1.c:31
Breakpoint 2 at 0x5555555551d5: file ud1.c, line 31.

(gdb) c
Continuing.

Breakpoint 2, main (argc=1, argv=0x7fffffffe4c8) at ud1.c:31
31              register_class(&wc);

(gdb) p wc
$1 = {style = 3, window_proc = 0x555555555155 <window_proc>, class_extra = 0, window_extra = 0,
instance = 0,
  icon = 1, cursor = 2, background = 3, menu_name = 0x555555556004 "menu", class_name =
0x555555556009 "ud1"}

(gdb) ptype /o wc
type = struct {
/*    0      |     4 */    uint32_t style;
/*    4      |     8 */    void (*window_proc)(p_msg_t);
/*   12      |     4 */    int32_t class_extra;
/*   16      |     4 */    int32_t window_extra;
/*   20      |     8 */    uint64_t instance;
/*   28      |     8 */    uint64_t icon;
/*   36      |     8 */    uint64_t cursor;
/*   44      |     8 */    uint64_t background;
/*   52      |     8 */    char *menu_name;
/*   60      |     8 */    char *class_name;

                           /* total size (bytes):   68 */
                         }
```

11. Then, we put a breakpoint inside the *register_class* function, resume execution, and inspect the parameter:

```
(gdb) break register_class
Breakpoint 3 at 0x7ffff7fc9277: file windows.c, line 82.

(gdb) c
Continuing.

Breakpoint 3, register_class (pwc=0x7fffffffe360) at windows.c:82
82          window_proc = pwc->window_proc;

(gdb) p pwc
$2 = (p_wnd_class_t) 0x7fffffffe360

(gdb) ptype /o pwc
type = struct {
/*    0      |     4 */    uint32_t style;
/* XXX  4-byte hole */
/*    8      |     8 */    void (*window_proc)(p_msg_t);
/*   16      |     4 */    int32_t class_extra;
/*   20      |     4 */    int32_t window_extra;
/*   24      |     8 */    uint64_t instance;
/*   32      |     8 */    uint64_t icon;
/*   40      |     8 */    uint64_t cursor;
/*   48      |     8 */    uint64_t background;
/*   56      |     8 */    char *menu_name;
/*   64      |     8 */    char *class_name;

                           /* total size (bytes):   72 */
                         } *

(gdb) p *pwc
$3 = {style = 3, window_proc = 0x5555, class_extra = 0, window_extra = 0, instance =
4294967296, icon = 8589934592,
  cursor = 12884901888, background = 6148926436540416000,
  menu_name = 0x5555600900005555 <error: Cannot access memory at address 0x5555600900005555>,
  class_name = 0x5555 <error: Cannot access memory at address 0x5555>}
```

12. But if we compare the structure inside the function with the structure variant outside (see step #10), we see its members have different offsets:

```
type = struct {
/*    0      |     4 */    uint32_t style;
/*    4      |     8 */    void (*window_proc)(p_msg_t);
/*   12      |     4 */    int32_t class_extra;
/*   16      |     4 */    int32_t window_extra;
/*   20      |     8 */    uint64_t instance;
/*   28      |     8 */    uint64_t icon;
/*   36      |     8 */    uint64_t cursor;
/*   44      |     8 */    uint64_t background;
/*   52      |     8 */    char *menu_name;
/*   60      |     8 */    char *class_name;

                           /* total size (bytes):   68 */
                         }
```

13. These discrepancies explain the crash. Looking at the *Makefile*, we can see that *ud1a* was compiled with the *-fpack-struct* setting. The *ud1b* executable was compiled without it and runs fine. Also, the problem was coincidentally fixed without changing alignment by using a different, bigger *wnd_class2_t* structure in the *ud1c* executable that adds another 32-bit field that makes both alignments identical.

14. We continue the execution and then quit GDB.

```
(gdb) c
Continuing.

Program received signal SIGSEGV, Segmentation fault.
0x0000000000005555 in ?? ()

(gdb) c
Continuing.

Program terminated with signal SIGSEGV, Segmentation fault.
The program no longer exists.

(gdb) q
```

# Exercise UD1 (LLDB)

**Goal:** Learn how code generation parameters can influence process execution behavior.

**Elementary Diagnostics Patterns:** Crash.

**Memory Analysis Patterns:** Exception Stack Trace; NULL Pointer (Code); Constant Subtrace.

**Debugging Implementation Patterns:** Break-in; Scope; Variable Value; Type Structure; Code Breakpoint.

1. The source code and the *Makefile* to build executables and libraries can be found in the *ud1* directory:

```
$ git clone https://bitbucket.org/softwarediagnostics/ald4
```

2. When we launch the *ud1a* executable, it crashes:

```
/mnt/c/ALD4/ud1$ LD_LIBRARY_PATH=. ./ud1a
Segmentation fault
```

3. We run the executable under LLDB until it shows a segmentation fault:

```
/mnt/c/ALD4/ud1$ LD_LIBRARY_PATH=. lldb ./ud1a
(lldb) target create "./ud1a"
Current executable set to './ud1a' (x86_64).

(lldb) r
Process 112 launched: '/mnt/c/ALD4/ud1/ud1a' (x86_64)
Process 112 stopped
* thread #1, name = 'ud1a', stop reason = signal SIGSEGV: invalid address (fault address:
0x5555)
    frame #0: 0x0000000000005555
error: memory read failed for 0x5400
```

4. The **image list** command lists loaded modules and their addresses:

```
(lldb) image list
[  0] 2C8F3A74-A0FC-977F-6362-C129A1E426DA-25DA9B99                    /mnt/c/ALD4/ud1/ud1a
[  1] 6005CB75-A439-321F-212E-2B1164734DFC-13352DF7                    /mnt/c/ALD4/ud1/libwindows.so
[  2] 83743DDD-4258-A7D1-38A2-8C4F2032D17A-D92A15B5                    /lib/x86_64-linux-gnu/ld-2.28.so
      /usr/lib/debug/.build-id/83/743ddd4258a7d138a28c4f2032d17ad92a15b5.debug
[  3] 6F3490CE-A127-50C7-4A4F-D33AC3B6CAAA-2ACE115B 0x00007ffff7fd3000 [vdso] (0x00007ffff7fd3000)
[  4] C7AA9A1E-121F-E239-5F38-40F3F0213146-046D9FE3                    /lib/x86_64-linux-gnu/libc.so.6
      /usr/lib/debug/.build-id/c7/aa9a1e121fe2395f3840f3f0213146046d9fe3.debug
```

5. We see that the crash happens in the **libwindows** module with the following CPU state:

```
(lldb) bt
* thread #1, name = 'ud1a', stop reason = signal SIGSEGV: invalid address (fault address: 0x5555)
  * frame #0: 0x0000000000005555
    frame #1: 0x00007ffff7fc926c libwindows.so`dispatch_message(pmessage=0x00007fffffffe3b0) at windows.c:76
    frame #2: 0x00005555555551ef ud1a`main(argc=1, argv=0x00007fffffffe4c8) at ud1.c:35
    frame #3: 0x00007ffff7e2109b libc.so.6`__libc_start_main(main=(ud1a`main at ud1.c:16), argc=1,
argv=0x00007fffffffe4c8, init=<unavailable>, fini=<unavailable>, rtld_fini=<unavailable>,
stack_end=0x00007fffffffe4b8) at libc-start.c:308
    frame #4: 0x000055555555509a ud1a`_start + 42
```

```
(lldb) register read
General Purpose Registers:
       rax = 0x0000000000005555
       rbx = 0x0000000000000000
       rcx = 0x00007ffff7ec3594  libc.so.6`__GI___nanosleep + 20 at nanosleep.c:28
       rdx = 0x00007fffffffe3b0
       rdi = 0x00007fffffffe3b0
       rsi = 0x00007fffffffe2b0
       rbp = 0x00007fffffffe340
       rsp = 0x00007fffffffe328
        r8 = 0x00007ffff7fb8d80  libc.so.6`initial
        r9 = 0x00007ffff7fb8d80  libc.so.6`initial
       r10 = 0xfffffffffffff429
       r11 = 0x00007ffff7fc9244  libwindows.so`dispatch_message at windows.c:76
       r12 = 0x0000555555555070  ud1a`_start
       r13 = 0x00007fffffffe4c0
       r14 = 0x0000000000000000
       r15 = 0x0000000000000000
       rip = 0x0000000000005555
    rflags = 0x0000000000010206
        cs = 0x0000000000000033
        fs = 0x0000000000000000
        gs = 0x0000000000000000
        ss = 0x000000000000002b
        ds = 0x0000000000000000
        es = 0x0000000000000000
```

6. We switch to stack frame #1 and check the source code:

```
(lldb) frame select 1
frame #1: 0x00007ffff7fc926c libwindows.so`dispatch_message(pmessage=0x00007fffffffe3b0) at
windows.c:76
   73   {
   74       if (window_proc)
   75       {
-> 76           window_proc(pmessage);
   77       }
   78   }
   79

(lldb) p window_proc
(void (*)(p_msg_t)) $0 = 0x0000000000005555

(lldb) list 80
   80   int register_class(p_wnd_class_t pwc)
   81   {
   82       window_proc = pwc->window_proc;
   83   }
   84
   85   int register_class2(p_wnd_class2_t pwc)
   86   {
   87       window_proc = pwc->window_proc;
   88   }
```

7. We see that the *window_proc* pointer is invalid, so we need to investigate when it is set in the *register_class* function below. First, we set the next frame where the *dispatch_message* was called:

```
(lldb) bt
* thread #1, name = 'ud1a', stop reason = signal SIGSEGV: invalid address (fault address: 0x5555)
    frame #0: 0x0000000000005555
  * frame #1: 0x00007ffff7fc926c libwindows.so`dispatch_message(pmessage=0x00007fffffffe3b0) at windows.c:76
    frame #2: 0x00005555555551ef ud1a`main(argc=1, argv=0x00007fffffffe4c8) at ud1.c:35
    frame #3: 0x00007ffff7e2109b libc.so.6`__libc_start_main(main=(ud1a`main at ud1.c:16), argc=1,
argv=0x00007fffffffe4c8, init=<unavailable>, fini=<unavailable>, rtld_fini=<unavailable>,
stack_end=0x00007fffffffe4b8) at libc-start.c:308
    frame #4: 0x000055555555509a ud1a`_start + 42

(lldb) frame select 2
frame #2: 0x00005555555551ef ud1a`main(argc=1, argv=0x00007fffffffe4c8) at ud1.c:35
   32
   33           while (get_message(&msg, 0, 0, 0))
   34           {
-> 35               dispatch_message(&msg);
   36           }
   37
   38       return 0;

(lldb) list 30
   30
   31           register_class(&wc);
   32
   33           while (get_message(&msg, 0, 0, 0))
   34           {
   35               dispatch_message(&msg);
   36           }
   37
   38       return 0;
   39   }
```

8. We can now check local variables and their structures (for example, *wc*):

```
(lldb) frame variable
(int) argc = 1
(char **) argv = 0x00007fffffffe4c8
(msg_t) msg = {
  hwnd = 0
  message = 275
  param1 = 1
  param2 = 140737353912581
  time = 72844112
  pt = (x = 156, y = 327)
  priv = 0
}
(wnd_class_t) wc = {
  style = 3
  window_proc = 0x0000555555555155 (ud1a`window_proc at ud1.c:11)
  class_extra = 0
  window_extra = 0
  instance = 0
  icon = 1
  cursor = 2
  background = 3
  menu_name = 0x0000555555556004 "menu"
  class_name = 0x0000555555556009 "ud1"
}
(lldb) p &wc
(wnd_class_t *) $1 = 0x00007fffffffe360
```

```
(lldb) memory read 0x00007fffffffe360
0x7fffffffe360: 03 00 00 00 55 51 55 55 55 55 00 00 00 00 00 00  ....UQUUUU......
0x7fffffffe370: 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00  ................
```

9. We put a breakpoint on the *main* function and resume execution until it is hit:

```
(lldb) c
Process 112 resuming
Process 112 exited with status = 11 (0x0000000b)

(lldb) breakpoint set -name main
Breakpoint 1: where = ud1a`main + 24 at ud1.c:20, address = 0x000055555555517f

(lldb) r
Process 176 launched: '/mnt/c/ALD4/ud1/ud1a' (x86_64)
Process 176 stopped
* thread #1, name = 'ud1a', stop reason = breakpoint 1.1
    frame #0: 0x000055555555517f ud1a`main(argc=1, argv=0x00007fffffffe4c8) at ud1.c:20
   20       msg_t msg;
   21       wnd_class_t wc;
   22
-> 23       wc.style = 3;
   24       wc.window_proc = window_proc;
   25       wc.class_extra = 0;
   26       wc.window_extra = 0;
```

10. We now put a breakpoint on the call to the *register_class* function and resume execution to inspect the passed value of the *wc* structure:

```
(lldb) list 27
   27       wc.background = 3;
   28       wc.menu_name = "menu";
   29       wc.class_name = "ud1";
   30
   31           register_class(&wc);
   32
   33           while (get_message(&msg, 0, 0, 0))
   34           {
   35               dispatch_message(&msg);
   36           }

(lldb) breakpoint set -line 31
Breakpoint 2: where = ud1a`main + 110 at ud1.c:31, address = 0x00005555555551d5

(lldb) c
Process 176 resuming
Process 176 stopped
* thread #1, name = 'ud1a', stop reason = breakpoint 2.1
    frame #0: 0x00005555555551d5 ud1a`main(argc=1, argv=0x00007fffffffe4c8) at ud1.c:31
   28       wc.menu_name = "menu";
   29       wc.class_name = "ud1";
   30
-> 31           register_class(&wc);
   32
   33           while (get_message(&msg, 0, 0, 0))
   34           {
```

```
(lldb) p wc
(wnd_class_t) $3 = {
  style = 3
  window_proc = 0x0000555555555155 (ud1a`window_proc at ud1.c:11)
  class_extra = 0
  window_extra = 0
  instance = 0
  icon = 1
  cursor = 2
  background = 3
  menu_name = 0x0000555555556004 "menu"
  class_name = 0x0000555555556009 "ud1"
}

(lldb) memory read &wc
0x7fffffffe360: 03 00 00 00 55 51 55 55 55 55 00 00 00 00 00 00  ....UQUUUU......
0x7fffffffe370: 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00  ................

(lldb) p &wc.window_proc
(void (**)(p_msg_t)) $10 = 0x00007fffffffe364
```

11. Then, we put a breakpoint inside the *register_class* function, resume execution, and inspect the parameter:

```
(lldb) breakpoint set -name register_class
Breakpoint 3: where = libwindows.so`register_class + 8 at windows.c:82, address =
0x00007ffff7fc9277

(lldb) c
Process 176 resuming
Process 176 stopped
* thread #1, name = 'ud1a', stop reason = breakpoint 3.1
    frame #0: 0x00007ffff7fc9277 libwindows.so`register_class(pwc=0x00007fffffffe360) at
windows.c:82
   79
   80   int register_class(p_wnd_class_t pwc)
   81   {
-> 82       window_proc = pwc->window_proc;
   83   }
   84
   85   int register_class2(p_wnd_class2_t pwc)

(lldb) p pwc
(p_wnd_class_t) $5 = 0x00007fffffffe360

(lldb) memory read pwc
0x7fffffffe360: 03 00 00 00 55 51 55 55 55 55 00 00 00 00 00 00  ....UQUUUU......
0x7fffffffe370: 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00  ................

(lldb) p &pwc->window_proc
(void (**)(p_msg_t)) $10 = 0x00007fffffffe368

(lldb) p *pwc
((anonymous struct)) $8 = {
  style = 3
  window_proc = 0x0000000000005555
  class_extra = 0
  window_extra = 0
  instance = 4294967296
  icon = 8589934592
  cursor = 12884901888
  background = 6148926436540416000
  menu_name = 0x5555600900005555 <no value available>
  class_name = 0x0000000000005555 <no value available>
}
```

12. So, if we compare the structure field address inside the function with the structure variant outside (see step #10), we see its members have different offsets:

```
(lldb) p &wc.window_proc
(void (**)(p_msg_t)) $10 = 0x00007fffffffe364
```

13. These discrepancies explain the crash. Looking at the *Makefile*, we can see that *ud1a* was compiled with the *-fpack-struct* setting. The *ud1b* executable was compiled without it and runs fine. Also, the problem was coincidentally fixed without changing alignment by using a different, bigger *wnd_class2_t* structure in the *ud1c* executable that adds another 32-bit field that makes both alignments identical.

14. We continue the execution and then quit LLDB.

```
(lldb) c
Process 176 resuming
Process 176 stopped
* thread #1, name = 'ud1a', stop reason = signal SIGSEGV: invalid address (fault address: 
0x5555)
    frame #0: 0x0000000000005555
error: memory read failed for 0x5400

(lldb) c
Process 176 resuming
Process 176 exited with status = 11 (0x0000000b)

(lldb) q
```