

使用 VScode 调试反汇编代码

by Anxi

本文档介绍如何使用 VScode 调试反汇编代码。

首先是对于一般的 Windows 环境，我们可以直接在工作目录下通过配置.vscode 文件夹下的 launch.json 和 task.json 文件来，通过调试控制台和反汇编视图来完成对于反汇编代码的单步调试。

Step1. 配置 launch.json 和 task.json 文件。

launch.json 如下：

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "Debug",
      "type": "cppdbg",
      "request": "launch",
      "program": "${fileDirname}\\exe_collect\\${fileBasenameNoExtension}.exe",
      "args": [],
      "stopAtEntry": false,
      "cwd": "${fileDirname}",
      "environment": [],
      "externalConsole": false,
      "MIMode": "gdb",
      "miDebuggerPath": "D:\\mingw64\\bin\\gdb.exe",
      "setupCommands": [
        {
          "description": "为 gdb 启用整齐打印",
          "text": "-enable-pretty-printing",
          "ignoreFailures": true
        },
        {
          "description": "将反汇编风格设置为 att",
          "text": "-gdb-set disassembly-flavor att",
          "ignoreFailures": true
        }
      ],
      "preLaunchTask": "C/C++: g++.exe 生成活动文件"
    }
  ]
}
```

task.json 如下：

```
{
  "tasks": [
    {
      "type": "shell"
    }
  ]
}
```

```

    "type": "cppbuild",
    "label": "C/C++: g++.exe 生成活动文件",
    "command": "D:\\mingw64\\bin\\g++.exe",
    "args": [
        "-fdiagnostics-color=always",
        "-g",
        "${file}",
        "-o",
        "${fileDirname}\\exe_collect\\${fileBasenameNoExtension}.exe",
        "-fexec-charset=GBK"
    ],
    "options": {
        "cwd": "${fileDirname}"
    },
    "problemMatcher": [
        "$gcc"
    ],
    "group": {
        "kind": "build",
        "isDefault": true
    },
    "detail": "调试器生成的任务。"
}
],
"version": "2.0.0"
}

```

注意将 gdb.exe 和 g++.exe 的路径改为自己的安装路径（最好使用新版 MinGW）。

Step2. 按下“F5”开始调试。

Step3. 查看反汇编代码。

下面介绍三种查看反汇编代码的方式：

法 1. 确保打开了“调用堆栈”视图



在“调用堆栈”处右键 main(), 即调试断点所在的函数，点击“打开反汇编视图”即可查看反汇编代码。结果如下：



```
test.c 反汇编 launch.json
0x0000000000401540    90          nop
0x000000000040154c    90          nop
0x000000000040154d    90          nop
0x000000000040154e    90          nop
0x000000000040154f    90          nop
0x0000000000401550    55          push %rbp
0x0000000000401551    48 89 e5    mov %rsp,%rbp
0x0000000000401554    48 83 ec 30  sub $0x30,%rsp
0x0000000000401558    e8 23 01 00 00  callq 0x401680 <__main>
0x000000000040155d    c7 45 fc 01 00 00 00  movl $0x1,-0x4(%rbp)
0x0000000000401564    c7 45 f8 02 00 00 00  movl $0x2,-0x8(%rbp)
D 0x000000000040156b  8b 55 f8    mov -0x8(%rbp),%edx
0x000000000040156e    8b 45 fc    mov -0x4(%rbp),%eax
0x0000000000401571    89 c1    mov %eax,%ecx
0x0000000000401573    e8 1f 00 00 00  callq 0x401597 <sum(int, int)>
0x0000000000401578    89 45 f4    mov %eax,-0xc(%rbp)
0x000000000040157b    8b 45 f4    mov -0xc(%rbp),%eax
0x000000000040157e    89 c2    mov %eax,%edx
0x0000000000401580    48 8d 0d 79 2a 00 00  lea 0x2a79(%rip),%rcx      # 0x404000
0x0000000000401587    e8 24 15 00 00  callq 0x402ab0 <printf>
0x000000000040158c    b8 00 00 00 00  mov $0x0,%eax
0x0000000000401591    48 83 c4 30  add $0x30,%rsp
0x0000000000401595    5d        pop %rbp
0x0000000000401596    c3        retq
0x0000000000401597    55        push %rbp
0x0000000000401598    48 89 e5    mov %rsp,%rbp
0x000000000040159b    48 83 ec 10  sub $0x10,%rsp
0x000000000040159f    89 4d 10  mov %ecx,-0x10(%rbp)
0x00000000004015a2    89 55 18  mov %edx,-0x18(%rbp)
0x00000000004015a5    c7 05 61 1a 00 00 02 00 00 00 00  movl $0x2,0x1a61(%rip)      # 0x403010 <version>
0x00000000004015af    8b 55 10  mov 0x10(%rbp),%edx
0x00000000004015b2    8b 45 18  mov 0x18(%rbp),%eax
0x00000000004015b5    01 d0    add %edx,%eax
0x00000000004015b7    89 45 fc    mov %eax,-0x4(%rbp)
0x00000000004015ba    8b 45 fc    mov -0x4(%rbp),%eax
0x00000000004015bd    48 83 c4 10  add $0x10,%rsp
0x00000000004015c1    5d        pop %rbp
0x00000000004015c2    c3        retq
0x00000000004015c3    90        nop
0x00000000004015c4    90        nop
0x00000000004015c5    90        nop
0x00000000004015c6    90        nop
0x00000000004015c7    90        nop
```

法 2. 直接在编辑器中按右键，在弹出的菜单栏中点击“打开反汇编视图”也可正常打开

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

```
int version = 1;

int sum(int var1, int var2);

int main()
{
    int a, b, result;
    a = 1;
    b = 2;
    result = sum(a, b);
    printf("result=%d\n", result);
}

int sum(int var1, int var2)
{
    int count;
    version = 2;
    count = var1 + var2;
    return (count);
}
```



法3. 在调试控制台中执行GDB命令“disassemble”

```
=thread-group-added,id="i1"
GNU gdb (GDB) 8.1
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-w64-mingw32".
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".
Warning: Debuggee TargetArchitecture not detected, assuming x86_64.
=cmd-param-changed,param="pagination",value="off"
[New Thread 24592.0x60dc]
[New Thread 24592.0x2a4c]

Thread 1 hit Breakpoint 1, main () at D:\code_files\Cpp_Code\test.c:10
10         a = 1;
Loaded 'C:\Windows\SYSTEM32\ntdll.dll'. Symbols loaded.
Loaded 'C:\Windows\System32\kernel32.dll'. Symbols loaded.
Loaded 'C:\Windows\System32\KernelBase.dll'. Symbols loaded.
Loaded 'C:\Windows\System32\msvcrt.dll'. Symbols loaded.

Thread 1 hit Breakpoint 7, main () at D:\code_files\Cpp_Code\test.c:12
12         result = sum(a, b);
Execute debugger commands using "-exec <command>", for example "-exec info registers" will list registers in use (when GDB is the debugger)
```

一般在VScode中运行或调试程序时都会自动跳出一些选项卡，默认是跳转到终端界面，但其实
在终端左边还有一个调试控制台，在该控制台中可以执行一系列的GDB命令。

要是没有自动打开，可以通过“查看->打开视图->调试控制台”唤出。

在给出的输入框中输入“-exec disassemble”命令即可输出当前断点所在函数的反汇编代码。

```
→ -exec disassemble
Dump of assembler code for function main():
 0x0000000000401550 <+0>: push  %rbp
 0x0000000000401551 <+1>: mov   %rsp,%rbp
 0x0000000000401554 <+4>: sub   $0x30,%rsp
 0x0000000000401558 <+8>: callq 0x401680 <_main>
 0x000000000040155d <+13>: movl  $0x1,-0x4(%rbp)
 0x0000000000401564 <+20>: movl  $0x2,-0x8(%rbp)
=> 0x000000000040156b <+27>: mov   -0x8(%rbp),%edx
 0x000000000040156e <+30>: mov   -0x4(%rbp),%eax
 0x0000000000401571 <+33>: mov   %eax,%ecx
 0x0000000000401573 <+35>: callq 0x401597 <sum(int, int)>
 0x0000000000401578 <+40>: mov   %eax,-0xc(%rbp)
 0x000000000040157b <+43>: mov   -0xc(%rbp),%eax
 0x000000000040157e <+46>: mov   %eax,%edx
 0x0000000000401580 <+48>: lea   0x2a79(%rip),%rcx      # 0x404000
 0x0000000000401587 <+55>: callq 0x402ab0 <printf>
 0x000000000040158c <+60>: mov   $0x0,%eax
 0x0000000000401591 <+65>: add   $0x30,%rsp
 0x0000000000401595 <+69>: pop   %rbp
 0x0000000000401596 <+70>: retq 
End of assembler dump.
```

但是该种方法并不能对反汇编代码进行单步调试，不过在 Unix V6++系统中，不知道由于什么原因，反汇编视图无法显示正常的反汇编代码，可以通过该方法对照查看当前正在执行的反汇编代码。

Step4. 在反汇编视图中通过单步执行方式调试代码，可以实现对于反汇编代码的单步执行任务。

使用该方法也可以在反汇编视图中对反汇编代码进行断点调试，例如：

0x000000000040156b	8b 55 f8	mov -0x8(%rbp),%edx
0x000000000040156e	8b 45 fc	mov -0x4(%rbp),%eax
● 0x0000000000401571	89 c1	mov %eax,%ecx
0x0000000000401573	e8 1f 00 00 00	callq 0x401597 <sum(int, int)>
▷ 0x0000000000401578	89 45 f4	mov %eax,-0xc(%rbp)
0x000000000040157b	8b 45 f4	mov -0xc(%rbp),%eax
0x000000000040157e	89 c2	mov %eax,%edx
0x0000000000401580	48 8d 0d 79 2a 00 00	lea 0x2a79(%rip),%rcx # 0x404000
0x0000000000401587	e8 24 15 00 00	callq 0x402ab0 <printf>
0x000000000040158c	b8 00 00 00 00	mov \$0x0,%eax
0x0000000000401591	48 83 c4 30	add \$0x30,%rsp

在 Unix V6++环境中，通过查看给出的 launch.json 文件不难发现，其中表示要调试的程序的 program 属性对应的路径为 kernel.exe 导致只能调试内核程序。在 VScode 中，不像 eclipse 那么复杂，可以直接修改该路径从而实现对于自己编写的程序的调试。

例如，我想要对 ls 指令进行调试，只需要将 launch.json 文件修改为如下代码：



```
[{"name": "Debug test", "type": "cppdbg", "request": "launch", "cwd": "${workspaceFolder}/src", "miDebuggerServerAddress": "localhost:1234", "miDebuggerPath": "E:/Unix_V6++/MinGW/bin/gdb.exe", "setupCommands": [{"description": "为 gdb 启用整齐打印", "text": "-enable-pretty-printing", "ignoreFailures": true}, {"description": "将反汇编风格设置为 att", "text": "-gdb-set disassembly-flavor att", "ignoreFailures": true}], "program": "${workspaceFolder}/src/program/objs/ls.exe", "args": [], "externalConsole": false, "MIMode": "gdb", "preLaunchTask": "OOS Run" // 前置任务：启动 Bochs，运行 OOS}
```

注意，这里相较于原始的 launch.json 还添加了一些对于 gdb 的控制命令。其次，还需注意将 GDB 路径修改为自己的安装路径。

修改成上述代码之后就可以对 ls 程序进行正常的调试工作。但是这里会出现的一个问题就是，对于一些命令，有可能使用 cd 命令也会导致断点被触发，这里暂时并未想明白原因。不过我们可以不用 cd 进入 bin 文件夹，直接运行相应命令也可以 debug。

使用上面提供的方法进入反汇编视图会发现，视图中无法正常显示反汇编代码，这时我们可以在调试控制台中执行“exec disassemble”命令来查看正确的反汇编代码，然后在反汇编视图中单步执行反汇编代码对照调试控制台的结果进行调试。这里暂未找出解决方法。

以 ls 为例，通过上述方法进行调试的示例如下图：



```
exec.0x00401401
Dump of assembler code for function main:
0x00401401 <+0>: push    %ebp
0x00401402 <+1>: mov     %esp,%ebp
0x00401403 <+2>: sub    $0x68,%esp
0x00401404 <+3>: movl   $0xb,-0xec(%ebp)
0x00401405 <+9>: movl   $0xb,-0xf0(%ebp)
0x00401406 <+19>: movl   $0x4056c5,-0x18(%ebp)
0x00401407 <+39>: movl   $0x4056c8,-0x14(%ebp)
0x00401408 <+49>: movl   $0xb,-0x10(%ebp)
0x00401409 <+59>: movl   $0x6c,-0x10(%ebp)
0x0040140a <+69>: movl   $0x4056c8,-0x10(%ebp)
0x0040140b <+79>: movl   $0x4056c9,-0x10(%ebp)
0x0040140c <+89>: movl   $0xb,-0x10(%ebp)
0x0040140d <+99>: movl   $0x68,-0xfc(%ebp)
0x0040140e <+109>: movl   $0xb,-0x11(%ebp)
0x0040140f <+119>: movl   $0xb,-0x12(%ebp)
0x00401410 <+129>: movl   $0xb,-0x12(%ebp)
0x00401411 <+139>: movl   $0xb,-0x534(%ebp)
0x00401412 <+149>: movl   $0x1,-0x538(%ebp)
0x00401413 <+159>: lea    -0x534(%ebp),%eax
0x00401414 <+165>: mov    %eax,-0x52(%ebp)
0x00401415 <+171>: lea    -0x538(%ebp),%eax
0x00401416 <+177>: mov    %eax,-0x530(%ebp)
-> 0x00401401 <+183>: movl   $0x2,0x1(%esp)
0x00401409 <+191>: lea    -0x18(%ebp),%eax
0x0040140f <+197>: mov    %eax,0x18(%esp)
0x00401411 <+201>: lea    -0x528(%ebp),%eax
0x00401419 <+207>: mov    %eax,0x14(%esp)
0x0040141d <+211>: lea    -0x138(%ebp),%eax
0x00401423 <+217>: mov    %eax,0x10(%esp)
0x00401427 <+221>: mov    -0x52(%ebp),%eax
0x0040142d <+227>: mov    $0xc(%esp)
0x00401431 <+231>: mov    $0x530(%ebp),%eax
0x00401437 <+237>: mov    %eax,0x8(%esp)
0x0040143d <+241>: mov    $0xc(%ebp),%eax
0x0040143e <+244>: mov    %eax,0x4(%esp)
0x00401442 <+248>: mov    $0x8(%ebp),%eax
0x00401445 <+251>: mov    %eax,(%esp)
0x00401448 <+254>: call   0x401000(GetOptAndPath)
0x0040144d <+259>: mov    %eax,-0x128(%ebp)
0x00401451 <+265>: cmpl  $0xb,-0x128(%ebp)
0x00401454 <+272>: je    0x40150f_main1+53>
0x00401460 <+278>: mov    $0x128(%ebp),%eax
0x00401466 <+284>: mov    %eax,-0x53(%ebp)
0x0040146c <+290>: cmpl  $0x3,-0x53(%ebp)
0x00401473 <+297>: je    0x4014d6_main1+36>
0x00401475 <+299>: cmpl  $0x3,-0x53(%ebp)
0x0040147c <+306>: jg    0x401495_main1+33>
0x0040147e <+308>: cmpl  $0x1,-0x53(%ebp)
0x00401485 <+315>: je    0x4014a4_main1+34>
0x00401487 <+317>: cmpl  $0x2,-0x53(%ebp)
0x0040148e <+324>: je    0x4014db_main1+36>
0x00401490 <+326>: jmp   0x401401_main1+183>
0x00401495 <+331>: cmpl  $0x68,-0x53(%ebp)
0x0040149c <+338>: je    0x4014f1_main1+423>
0x0040149e <+340>: cmpl  $0x6c,-0x52(%ebp)
0x004014a5 <+347>: je    0x401500_main1+438>
0x004014a7 <+349>: jmp   0x401401_main1+183>
```

注意，在调试控制台输出的反汇编代码中，“=>”指向的是正在执行的汇编指令。

提示：有时候 VScode 的反汇编视图无法正常显示，这种情况只需要重启 VScode 就能解决！！！