

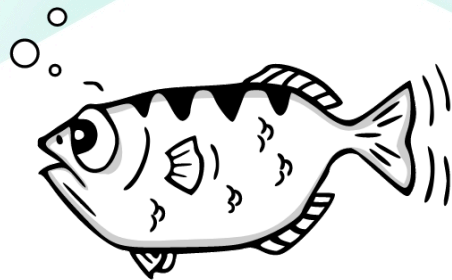
Как я написала своё отладочное расширение для VS Code, и почему оно работает лучше всех остальных

Гусарова Анастасия



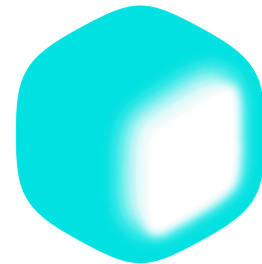
О чём поговорим

2



GDB: The GNU Project Debugger

- Как работает отладка
- Почему существующие решения нам не подошли
- Наш инструментарий отладки
- Подведём итоги

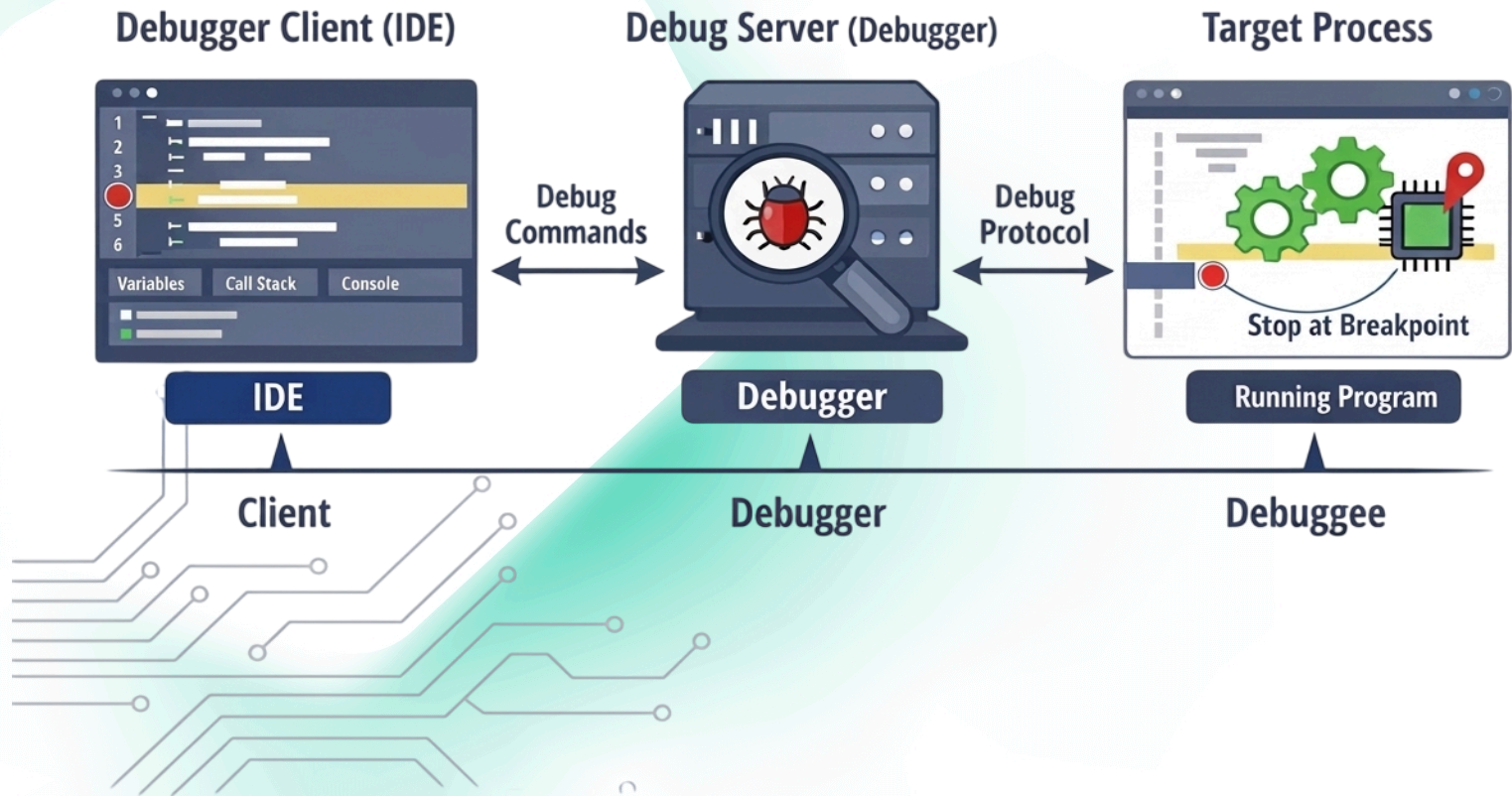


kasperskyOS

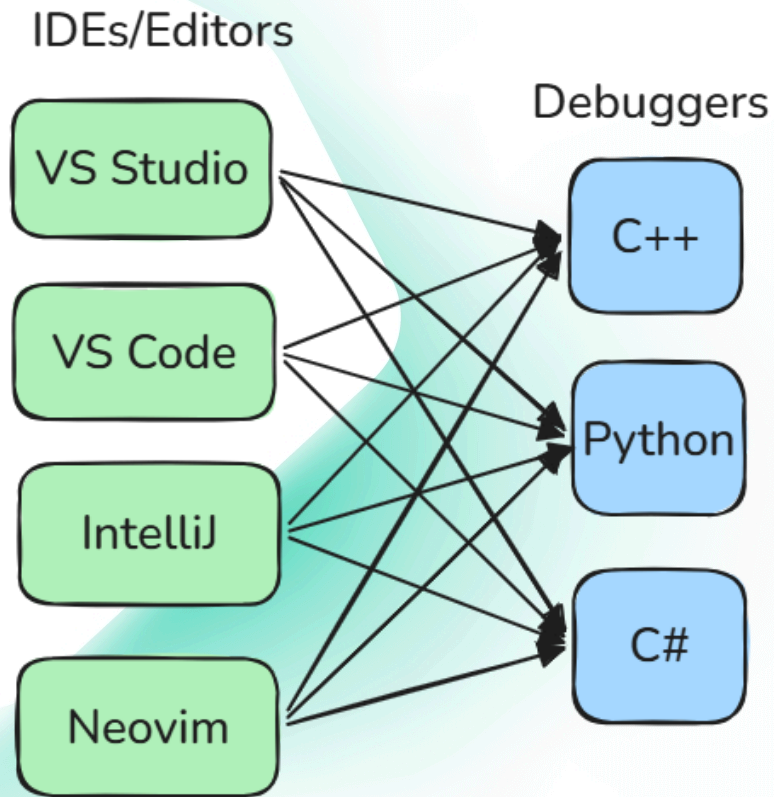
Be Immune



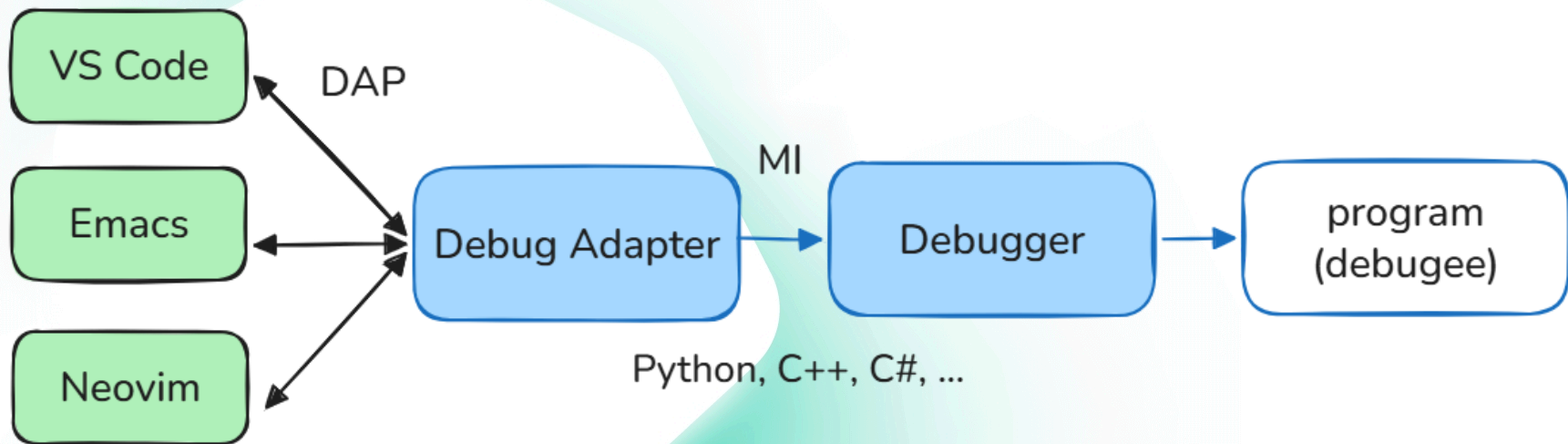
Тулинг отладки



Дикий мир инструментов отладки



Современное решение



DAP

- DAP - абстрактный протокол связи между IDE и отладчиком
- MI - машинно-ориентированный текстовый протокол, используемый для управления отладчиками

Пример DAP-запроса

```
→ Adapter message: {
  "command": "setBreakpoints",
  "arguments": {
    "source": {
      "name": "server.cpp",
      "path": "/home/anastasiia/MyProjects/test_gdb/client-server/server.cpp"
    },
    "lines": [
      53
    ],
    "breakpoints": [
      {
        "line": 53
      }
    ],
    "sourceModified": false
  },
  "type": "request",
  "seq": 26
}
```



Спецификация DAP

Пример MI-запроса

7

```
-break-insert ./server.cpp:53
```



Спецификация MI

Пример MI-ответа

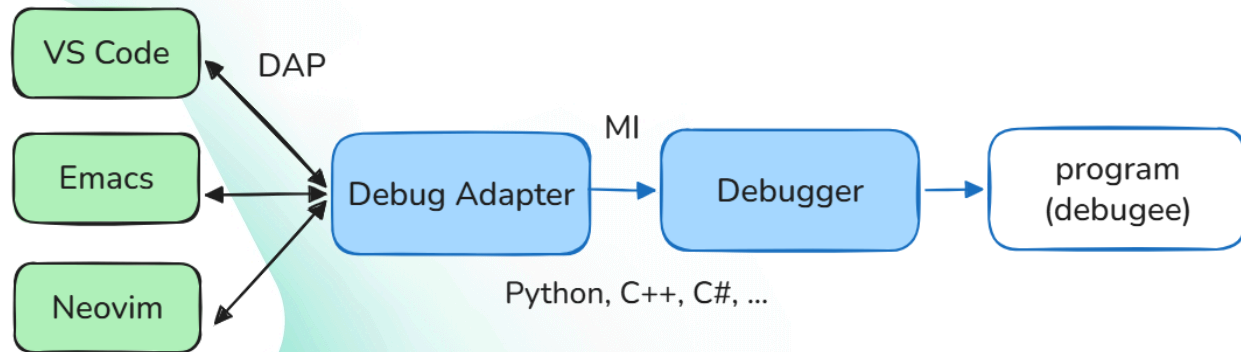
```
✓ ^done,bkpt={
  . . . number="3",
  . . . type="breakpoint",
  . . . disp="keep",
  . . . enabled="y",
  . . . addr="0x0000555555556876",
  . . . func="main()",
  . . . file="server.cpp",
  . . . fullname="/home/anastasiia/MyProjects/test_gdb/client-server/server.cpp",
  . . . line="53",
  . . . thread-groups=["i1"],
  . . . times="0",
  . . . original-location="/home/anastasiia/MyProjects/test_gdb/client-server/server.cpp:53"
}
```

Как работает отладка?



Ставим breakpoint

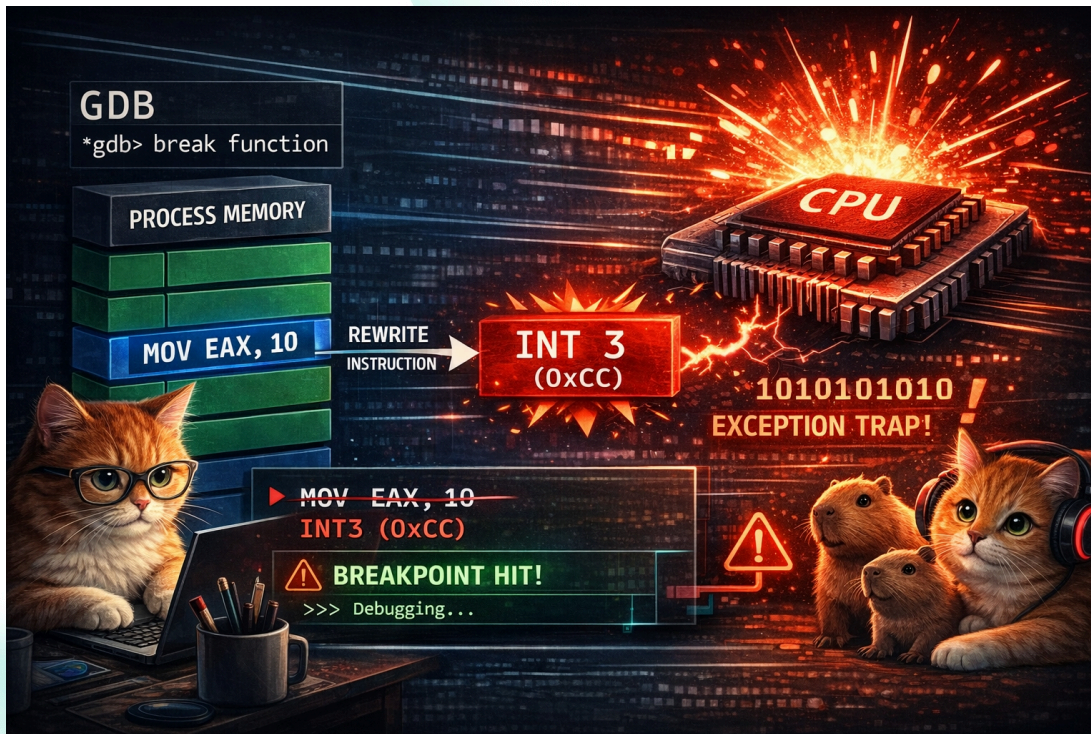
```
55  
● 56      std::c  
57
```



- IDE отправляет запрос в debug adapter (через DAP)
- debug adapter отправляет запрос в GDB (через MI)
- GDB ставит breakpoint в процессе

Как выглядит breakpoint

- GDB переписывает инструкцию и вставляет ловушку



Как выглядит breakpoint x86

- 0xCC - однобайтовый opcode инструкции INT3

Что это даёт?

- можно заменить ровно 1 байт
- не нужно знать длину инструкции
- безопасно в большинстве случаев
- легко восстановить назад

```
[оригинал]  55 48 89 E5 ...  
↓  
[breakpoint] CC 48 89 E5 ...
```

Как это использует GDB

Как выглядит breakpoint на разных архитектурах

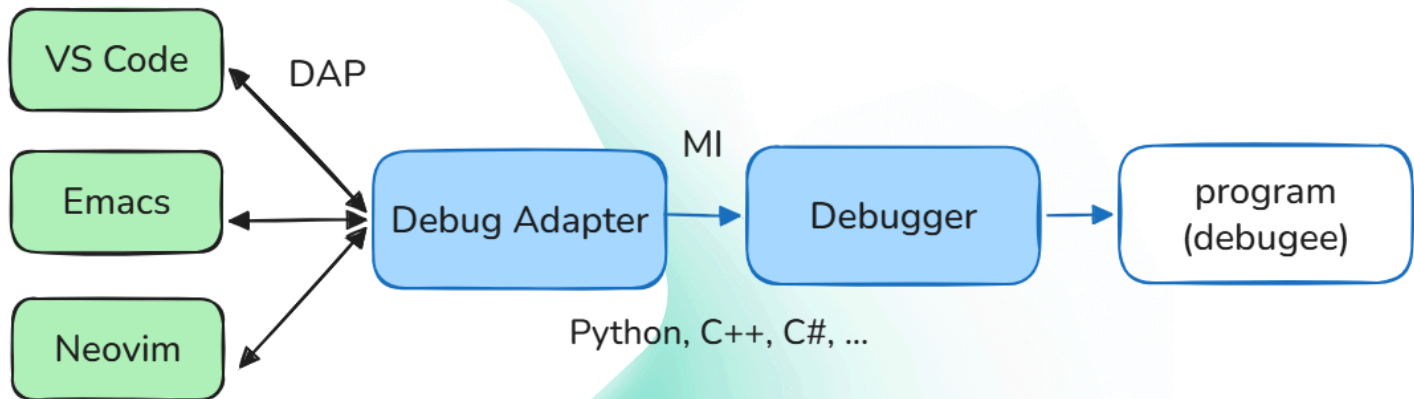
x86:

- INT3 (0xCC) — 1 байт

ARM:

- используется специальная инструкция (BKPT / BRK)
- инструкции фиксированного размера
- исходная инструкция перезаписывается целиком

GDB поставил breakpoint



Обратный путь:

- GDB сообщает в debug adapter
- adapter сообщает IDE
- IDE обновляет UI

Попадаем в breakpoint

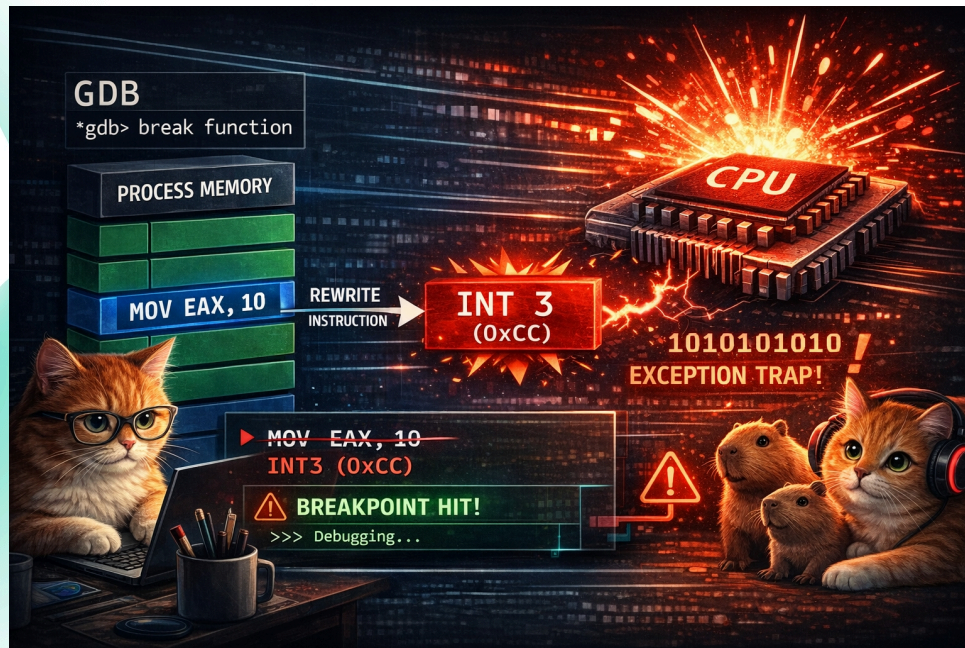


- Проходим путь IDE -> GDB
- GDB продолжает выполнение и выполняет блокирующий вызов `waitpid`

`waitpid` – системный вызов Linux, позволяет процессу ждать изменения состояния другого процесса или потока

Попадаем в breakpoint

- процесс дошел до ловушки
- процессор генерирует исключение (Breakpoint Exception)



Попадаем в breakpoint

- Ядро ОС преобразует исключение в сигнал SIGTRAP и останавливает поток отлаживаемого процесса
- GDB с помощью waitpid узнал, что поток остановлен
- GDB спрашивает у ядра причину остановки (si_code)

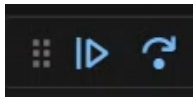
SIGTRAP – сигналы, связанные с отладкой

```
| what | si_code |
|-----+-----|
| software breakpoints (int3) | SI_KERNEL |
| single-steps | TRAP_TRACE |
| single-stepping a syscall | TRAP_BRKPT |
| user sent SIGTRAP | 0 |
| exec SIGTRAP (when no PTRACE_EVENT_EXEC) | 0 |
| hardware breakpoints/watchpoints | TRAP_HWBKPT |
```

- Проходим путь GDB -> IDE

```
55 | |
56 | std::cout <<
57 | |
```

Продолжаем выполнение



- Проходим путь IDE -> GDB
- GDB временно убирает ловушку и восстанавливает исходную инструкцию
- Откатывает указатель инструкций назад

RIP - Instruction Pointer Register (x86)

```
RIP = RIP - 1
```

PC - Program Counter (ARM)

```
PC = PC - sizeof(instruction)
```

Продолжаем выполнение

- Инструкция выполняется
- GDB возвращает ловушку и возобновляет выполнение
- Проходим путь GDB -> IDE

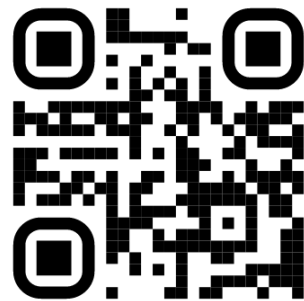
DWARF



Формат отладочной информации,
которая добавляется компилятором

Содержит:

- переменные
- функции
- типы
- соответствие кода и памяти



DWARF

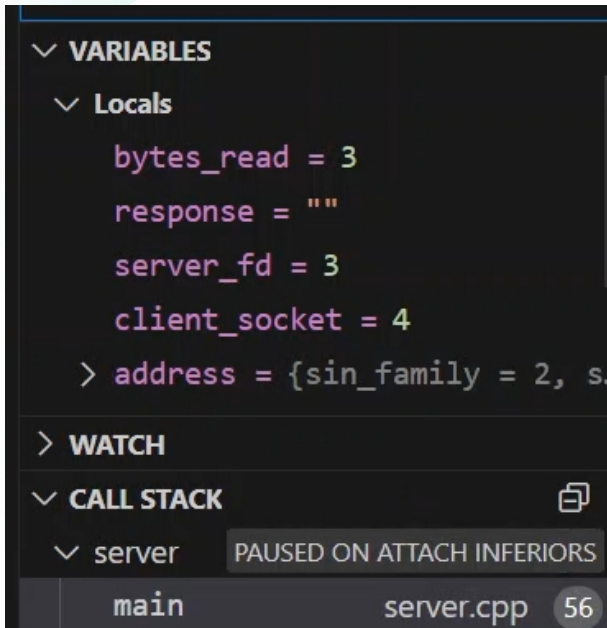
DWARF

DWARF - набор секций

- .debug_info - **что** есть в программе
- .debug_line - **где** это в исходниках
- .debug_loc - **где** это лежит во время выполнения

```
1  {
2  .. "functions": [
3  .. .. {
4  .. .. .. "name": "main",
5  .. .. .. "variables": [
6  .. .. .. .. {
7  .. .. .. .. .. "name": "x",
8  .. .. .. .. .. "type": "int",
9  .. .. .. .. .. "location": [
10 .. .. .. .. .. .. {
11 .. .. .. .. .. .. .. "range": "0x401000-0x401010",
12 .. .. .. .. .. .. .. "where": "RAX"
13 .. .. .. .. .. .. }
14 .. .. .. .. .. ]
15 .. .. .. .. }
16 .. .. .. ]
17 .. .. }
18 .. ],
19 .. "line_mapping": [
20 .. .. {
21 .. .. .. "address": "0x401000",
22 .. .. .. "file": "main.c",
23 .. .. .. "line": 10
24 .. .. }
25 .. ]
26 }
27
```

Переменные

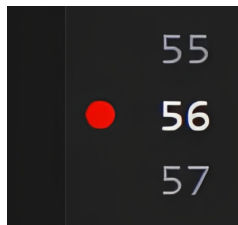


The screenshot shows a debugger's interface with a dark theme. The 'VARIABLES' panel is expanded to show 'Locals'. The variables listed are: bytes_read = 3, response = "", server_fd = 3, client_socket = 4, and address = {sin_family = 2, s...}. Below this is the 'WATCH' panel, which is currently empty. The 'CALL STACK' panel shows the current function as 'main' in 'server.cpp' at line 56. The status bar at the bottom indicates 'PAUSED ON ATTACH INFERIORS'.

```
✓ VARIABLES
  ✓ Locals
    bytes_read = 3
    response = ""
    server_fd = 3
    client_socket = 4
    > address = {sin_family = 2, s...
  > WATCH
  ✓ CALL STACK [copy icon]
    ✓ server PAUSED ON ATTACH INFERIORS
      main server.cpp 56
```

- Проходим путь IDE -> GDB
- GDB использует DWARF (получает .debug_info, .debug_loc и .debug_line)
- GDB читает соответствующие байты из памяти/регистра
- GDB интерпретирует данные согласно типу, получая значение переменной
- Проходим путь GDB -> IDE

Мы разобрали



Как ставятся breakpoints



Умеем выяснить
причину остановки

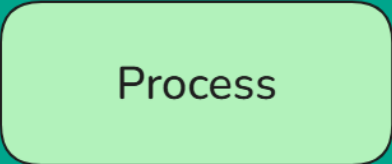


Знаем, что такое
DWARF

Интерпретация процессов в GDB

Процесс в системе

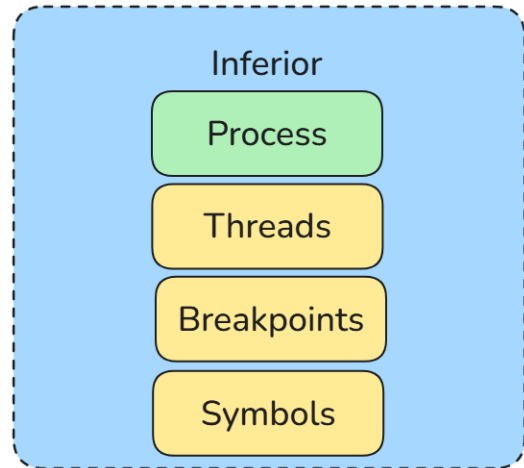
- Просто PID
- Выполняется сам по себе



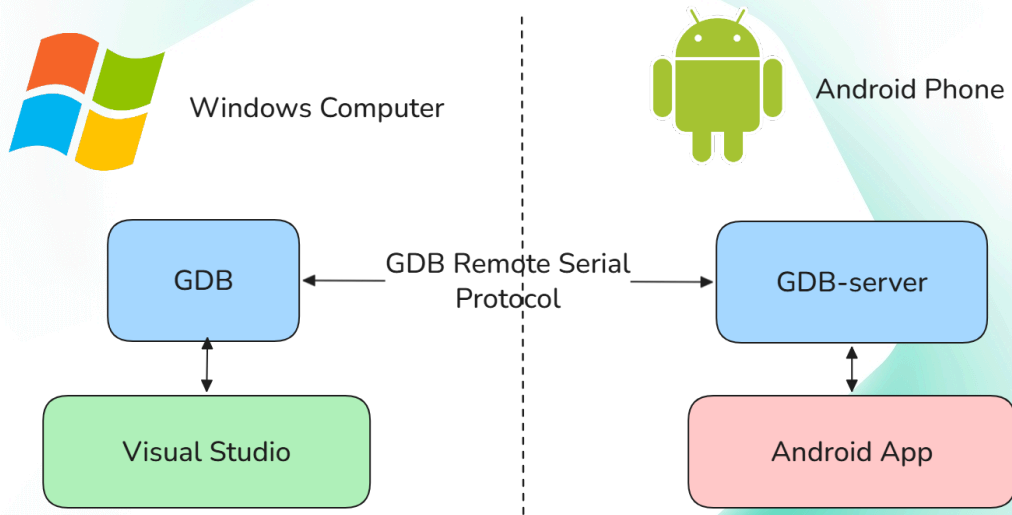
Inferiors

Inferior в GDB

- Представляет процесс
- Содержит состояние, которым управляет GDB (потoki, точки останова, символа)
- Может существовать до запуска процесса



Удалённая отладка

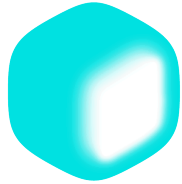


- GDB-сервер - компонент на целевой системе, обеспечивает взаимодействие между GDB и отлаживаемой программой.
- Remote Serial Protocol - протокол обмена данными между GDB и GDB-сервером.



Remote debugging

Kernel GDB-stub



kasperskyOS



gdbserver (Linux)

- Программа-посредник
- Сколько раз запущен gdbserver – столько и соединений
- Подключение к gdbserver / сразу к нужному процессу
- Останавливается только отлаживаемый процесс



Kernel GDB-stub (Kaspersky OS)

- Ядерный gdbstub
- Одно соединение
- Подключение к случайному процессу
- Останавливалась вся система (до недавнего времени)



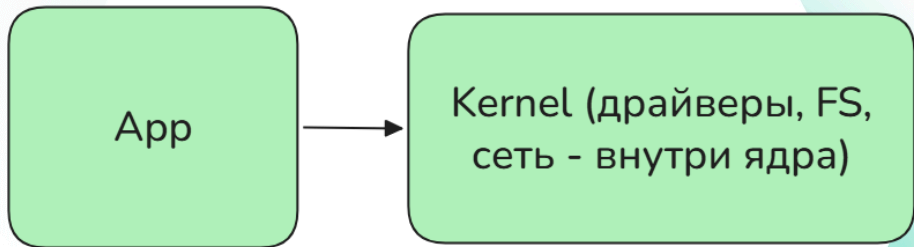
А почему вообще тормозит? Аутопсия стандартного расширения от Microsoft и почему оно тормозит с KasperskyOS

A screenshot of the Microsoft Store page for the C/C++ extension. The page has a dark background. On the left is a circular icon with a purple-to-white gradient and the text 'C/C++'. To the right of the icon, the text 'C/C++' is displayed in a large font. Below this, it says 'Microsoft' with a blue checkmark and the URL 'microsoft.com'. Further right, it shows '96,550,979' downloads and a star rating of '4.5 (598)'. Below the main text, there is a description: 'C/C++ IntelliSense, debugging, and code browsing.' At the bottom, there are three controls: 'Disable' with a dropdown arrow, 'Uninstall' with a dropdown arrow, and 'Auto Update' with a checked checkbox and a gear icon for settings. A blue double-left arrow button is overlaid on the bottom right of the icon area.

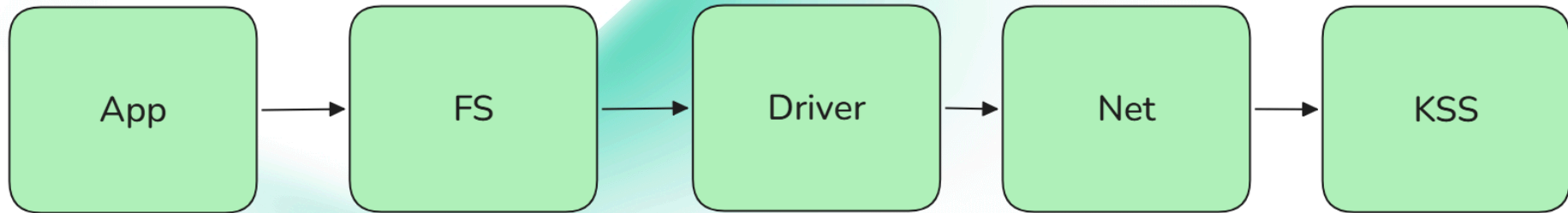
C/C++
Microsoft microsoft.com | 96,550,979 | ★★★★★ (598)
C/C++ IntelliSense, debugging, and code browsing.
Disable | Uninstall | Auto Update

Архитектурные различия Linux и KasperskyOS

Linux



Kaspersky OS



(каждый блок - отдельный процесс)

Попытки продебажить несколько Linux процессов в одной сессии

- -exec add-inferior
- -exec inferior 2
- ps aux | grep ~/client-server/client
- -exec attach 1855572



```
PROBLEMS 1  DEBUG CONSOLE  TERMINAL  OUTPUT  SPELL CHECKER 71  PORTS 1
→ -exec add-inferior
=thread-group-added,id="i2"
[New inferior 2]
Added inferior 2 on connection 1 (native)
→ -exec inferior 2
[Switching to inferior 2 [<null>] (<noexec>)]
→ -exec attach 1855572
Reading symbols from /home/anastasiia/MyProjects/test_gdb/client-server/client...
Attaching to process 1855572
```

Попытки продебажить несколько Linux процессов в одной сессии

```
response = "\022\n\005\000\000\000\000\030..."
server_fd = 3
client_socket = 4
> address = {...}
opt = 1
addrlen = 16
> buffer = [1024]
str = "Hello"
> fruits = std::vector of length 3, c...
> Registers

> WATCH
CALL STACK
> client [1257896] PAUSED ON BREAKPOINT
server [1257078] PAUSED ON EXCEPTION
__GI__libc_read(int fd, void * buf,
main() server.cpp 54:1

PROBLEMS DEBUG CONSOLE TERMINAL OUTPUT SPELL CHECKER 23 PORTS 2
- exec add-inferior
```

```
46 ... perror(s: "accept");
47 ... return 1;
48 ... }
49
50 ... std::cout << "The client is connected.\n";
51
52 ... while (true) {
53 ...     memset(s: buffer, c: 0, n: sizeof(buffer));
54 ...     ssize_t bytes_read = read(fd: client_socket, buf: buffer, nbytes: sizeof(buffer) - 1);
55 ...     if (bytes_read <= 0) {
56 ...         std::cout << "The client has disconnected.\n";
57 ...         break;
58 ...     }
59
60 ...     std::cout << "Received: " << buffer << "\n";
61
62 ...     std::string response = "Server response: ";
63 ...     response += buffer;
```

```
41     ... send(fd: sock, buf: message.c_str(), n: message.length(), flags: 0);
42
43     ... memset(s: buffer, c: 0, n: sizeof(buffer));
44     ... read(fd: sock, buf: buffer, nbytes: sizeof(buffer) - 1);
45
46     ... std::cout << "Response from the server: " << buffer << "\n";
47     ... }
48
49     ... close(fd: sock);
50     ... return 0;
51 }
52
```

Exception has occurred. X
Terminated

WATCH

CALL STACK

- server [578170] **PAUSED**
 - libc.so.6!__GI__libc_read(int fd, void*
 - main() server.cpp 54:1
- client [578899] **PAUSED ON EXCEPTION**
 - main() client.cpp 46:1

Завершение одного из отлаживаемых процессов привело к завершению всей сессии отладки

[Debugger] Allow an option to only exit debugging once all inferiors have exited #8102

New issue



Open



randomdude789 opened on Sep 4, 2021



Type: Feature Request

Currently, once a single inferior exits, the entire debugging process exits as well. The only workaround available is to detach inferiors manually with gdb commands before they end. This is particularly problematic when gdb's detach-on-fork mode is turned on in complex programs with multiple fork calls.

The proposed solution is to add some user-facing option that will allow individual inferiors to exit without ending debugging. Since multiple-inferior debugging seems to work fine otherwise, I'm hoping this is possible.

10

Assignees

No one assigned

Labels

Feature Request

debugger

Type

No type

Projects

No projects

Завершение одного процесса приводит к завершению всей сессии отладки: [Issue #8102](#)



When debugging multiple inferiors with gdb, if one inferior exits the full debug session stops #13893

New issue



Open



lostgoat opened on Sep 2, 2025

...

Assignees

No one assigned

Завершение одного процесса приводит к завершению всей сессии отладки: [Issue #13893](#)



[Visual C++] Allow child process debugging #1211

Open



zjturner opened on Nov 7, 2017

WinDbg has the `.childdb` option which will automatically cause the debugger to attach to child processes when the parent calls `CreateProcess`.

GDB has a similar option, where you can set it to follow forks. [#511](#) is related, and a workaround is offered in the comments to enable setting the follow fork mode to on. But this is not the ideal solution, because it doesn't provide any solution for the Windows Debug Engine, `cppvsdbg`. It would be nice if the `launch.json` configuration supported a simple boolean variable, `debugChildProcess` which you could set to true or false. By default it's `false`, and the user can override it.



63

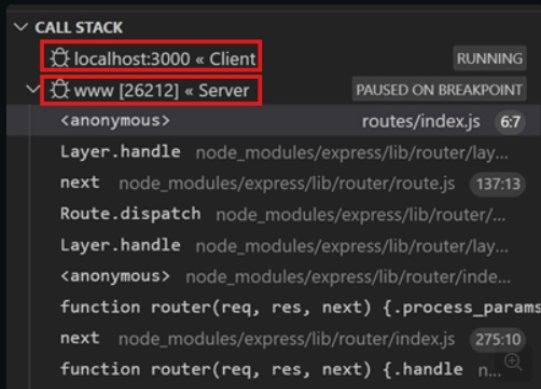
Нет нормальной поддержки отладки дочерних процессов: [Issue #1211](#)



Multi-target debugging

For complex scenarios that involve more than one process (for example, a client and a server), VS Code supports multi-target debugging. After you've started a first debug session, you can launch another debug session. As soon as a second session is up and running, the VS Code UI switches to *multi-target mode*:

- The individual sessions now show up as top-level elements in the **CALL STACK** view.



- The debug toolbar shows the currently active session (and all other sessions are available in a dropdown menu).



[Отладка нескольких процессов в несколько сессий](#)



Отладка нескольких сессий

The screenshot displays the Visual Studio Code interface during a multi-session debugging session. The top toolbar shows 'RUN AND DEBUG' with a 'Client' dropdown menu. The left sidebar contains several panels: 'PROCESSES', 'VARIABLES', and 'WATCH'. The 'VARIABLES' panel shows local variables for the 'Client' session, including 'server_fd = 3', 'client_socket = 32767', 'address = {...}', 'opt = 1', 'addrlen = 16', 'buffer = [1024]', 'str = {...}', and 'fruits = {...}'. The 'WATCH' panel is currently empty. The 'CALL STACK' panel shows two sessions: 'Server' and 'Client', both with a 'PAUSED ON BREAKPOINT' status. The central code editor displays the 'launch.json' file, which defines two configurations: 'Server' and 'Client'. The 'Server' configuration is selected in the right sidebar dropdown menu. The 'Server' configuration is defined as follows:

```
1 {
2   .. "version": "0.2.0",
3   .. "configurations": [
4     .. {
5     ..   "name": "Server",
6     ..   "type": "cppdbg",
7     ..   "request": "launch",
8     ..   "program": "/home/anastasiia/MyProjects/test_gdb/client-server/server",
9     ..   "cwd": ".",
10    .. },
11    .. {
12    ..   "name": "Client",
13    ..   "type": "cppdbg",
14    ..   "request": "launch",
15    ..   "program": "/home/anastasiia/MyProjects/test_gdb/client-server/client",
16    ..   "cwd": ".",
17    .. }
18  .. ]
19 }
20 ..
21
22
23
24
25
26
27
28
```

Отладка нескольких сессий

```
1 {
2   "version": "0.2.0",
3   "configurations": [
4     {
5       "name": "Server",
6       "type": "cppdbg",
7       "request": "launch",
8       "program": "/home/anastasiia/MyProjects/test_gdb/client-server/server",
9       "cwd": ".",
10    },
11    {
12      "name": "Client",
13      "type": "cppdbg",
14      "request": "launch",
15      "program": "/home/anastasiia/MyProjects/test_gdb/client-server/client",
16      "cwd": ".",
17    }
18  ]
19 }
```

- Нужно настраивать launch.json под каждый сценарий
- В KasperskyOS не поддерживается отладка в нескольких сессиях

Very long delay when a breakpoint is hit and many (150+) threads are created by the program #9988

New issue

Open



jusid opened on Oct 12, 2022 · edited by jusid

Edits ...

Environment

- OS and version: Ubuntu 20.04.5 LTS
- VS Code: 1.72.1

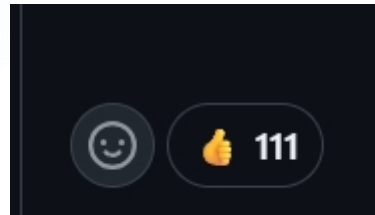
Assignees

No one assigned

Labels

bug debugger performance

Долгое время инициализации при большом количестве потоков: [Issue #9988](#)



Будем делать сами...

Мне не нравится эта экспедиция!
Мне не нравятся эти матросы!
И вообще... что? Да! Нет! Мне
вообще ничего не нравится, сэр!



MIEngine

microsoft/MIEngine



The Visual Studio MI Debug Engine ("MIEngine") provides an open-source Visual Studio Debugger extension that works with MI-enabled debuggers such...

74

Contributors

153

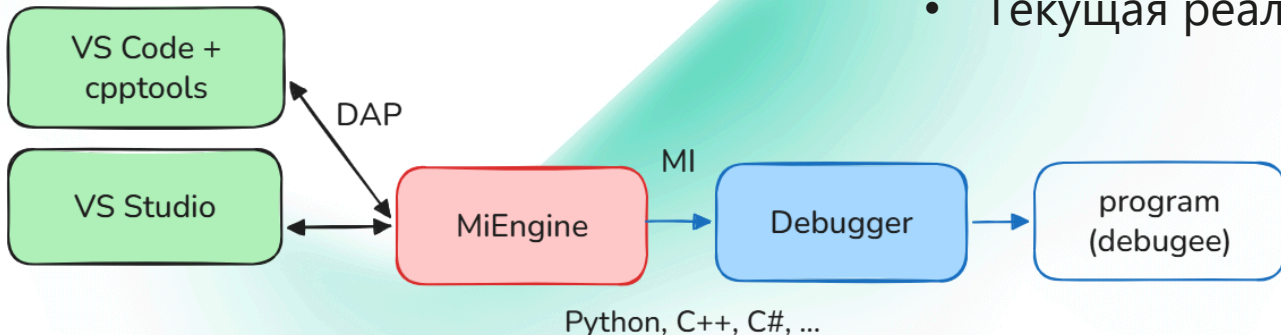
Issues

847

Stars

225

Forks



- Невозможность отлаживаться в одной сессии
- Наши проблемы не в приоритете у мейнтейнеров
- Поддержка форка – трудоёмкая задача
- Текущая реализация нестабильна



Native Debug



Native Debug

WebFreak | 📄 700,244 installs | ★★★★★ (17) | Free

GDB, LLDB & Mago-MI Debugger support for VSCode

[Install](#)



[Trouble Installing?](#) 📄

- Не мейнстримовое
- Нет поддержки отладки нескольких процессов
- Нет поддержки аттача к pid на remote

GDB Debugger - Beyond




GDB Debugger - Beyond

coolchyni |  130,487 installs |  (8) | Free

Debugger with gdb for c,c++,freepascal,fortran and more.

[Install](#)

[Trouble Installing?](#) 

- Не мейнстримовое
- Не реализована большая часть DAP
- Не получилось нормально отладить



CodeLLDB

Vadim Chugunov | 📄 10,143,810 installs | ★★★★★ (96) | Free


Debugger for native code, powered by LLDB. Debug C++, Rust, and other compiled languages.

Install

[Trouble Installing?](#)

arm

Arm CMSIS Debugger

Arm  arm.com | 📄 48,796 installs | ★★★★★ (0) | Free

Run and debug embedded and Edge AI projects on Arm Cortex-M single or multi core devices. Connects via pyOCD to CMSIS-DAP or other GDB servers.

Install

[Trouble Installing?](#)



Cortex-Debug

marus25 | 📄 1,457,908 installs | ★★★★★ (40) | Free

ARM Cortex-M GDB Debugger support for VSCode

Install

[Trouble Installing?](#)

Использование gdb в режиме отладочного сервера для сред разработки (gdb -i=dap)

[projects](#) / [binutils-gdb.git](#) / summary

summary | [shortlog](#) | [log](#) | [commit](#) | [commitdiff](#) | [tree](#)

description gdb and binutils
last change Thu, 26 Mar 2026 13:10:36 +0300 (05:10 -0500)
URL [git://sourceware.org/git/binutils-gdb.git](https://sourceware.org/git/binutils-gdb.git)
[ssh://sourceware.org/git/binutils-gdb.git](https://sourceware.org/git/binutils-gdb.git)
<https://sourceware.org/git/binutils-gdb.git>

shortlog

2 hours ago	Aditya Vidyadhar...	Fix assertion failure while analysing core files in...	master	commit commitdiff tree
2 hours ago	Tom de Vries	[gdb] Add c_ctrl/c_unctrl		commit commitdiff tree
7 hours ago	Alan Modra	aout, ecoff and som free_cached_info		commit commitdiff tree
7 hours ago	Alan Modra	xcoff free_cached_info		commit commitdiff tree
12 hours ago	GDB Administrator	Automatic date update in version.in		commit commitdiff tree
19 hours ago	Tom Tromey	Hold the GIL while clearing gdbpy_reggroup_object_map		commit commitdiff tree
21 hours ago	Jan Vraný	gdb/python: add property ranges to gdb.Block object		commit commitdiff tree
23 hours ago	Tom de Vries	[gdb/record] Fix syscall exit recording for riscv		commit commitdiff tree
23 hours ago	Tom de Vries	[gdb/record] Fix syscall exit recording for arm		commit commitdiff tree
24 hours ago	Jan Vraný	gdb/MAINTAINERS: Update my email address		commit commitdiff tree
26 hours ago	Jens Remus	s390: Skip non-PIC shared library visibility linker...		commit commitdiff tree
29 hours ago	Tom de Vries	[gdb/remote] Use sevenbit_strings == true in escape_buffer		commit commitdiff tree
36 hours ago	GDB Administrator	Automatic date update in version.in		commit commitdiff tree
44 hours ago	Tankut Baris...	gdb: preserve type instance flags in 'x' command		commit commitdiff tree
44 hours ago	Tankut Baris...	gdb: track type instead of gdbarch for 'next_address'		commit commitdiff tree
44 hours ago	Tankut Baris...	gdb: extract a function out of do_examine_next_address		commit commitdiff tree



GDB DAP

[Oleg Tolmatcev](#) | [624 installs](#) | [★★★★★ \(0\)](#) | [Fr](#)

GDB debugging from VSCode

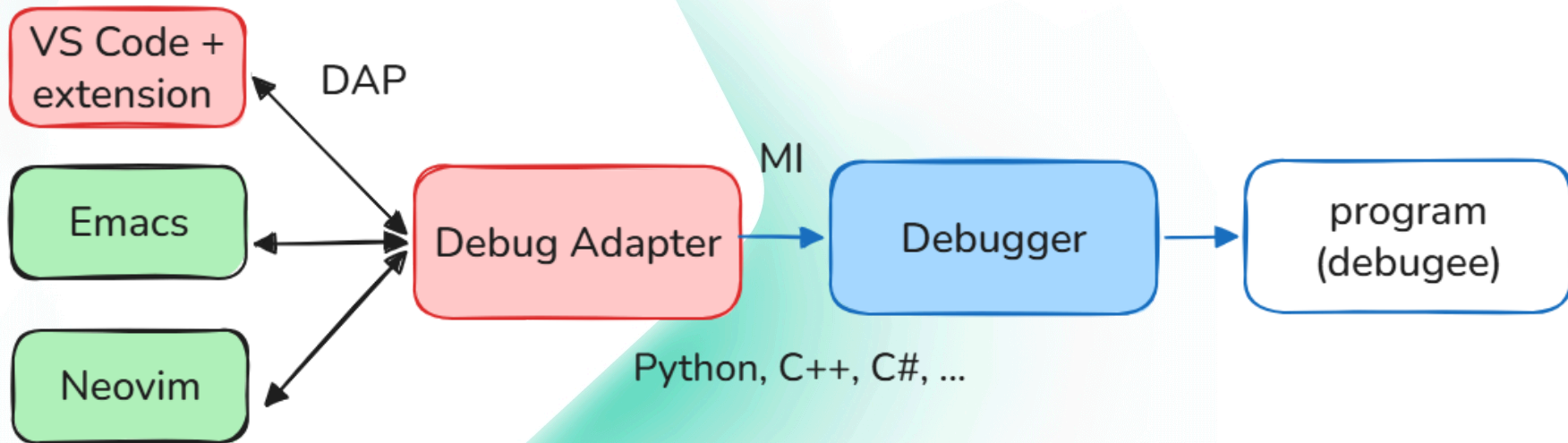
[Install](#)

[Trouble Installing?](#)

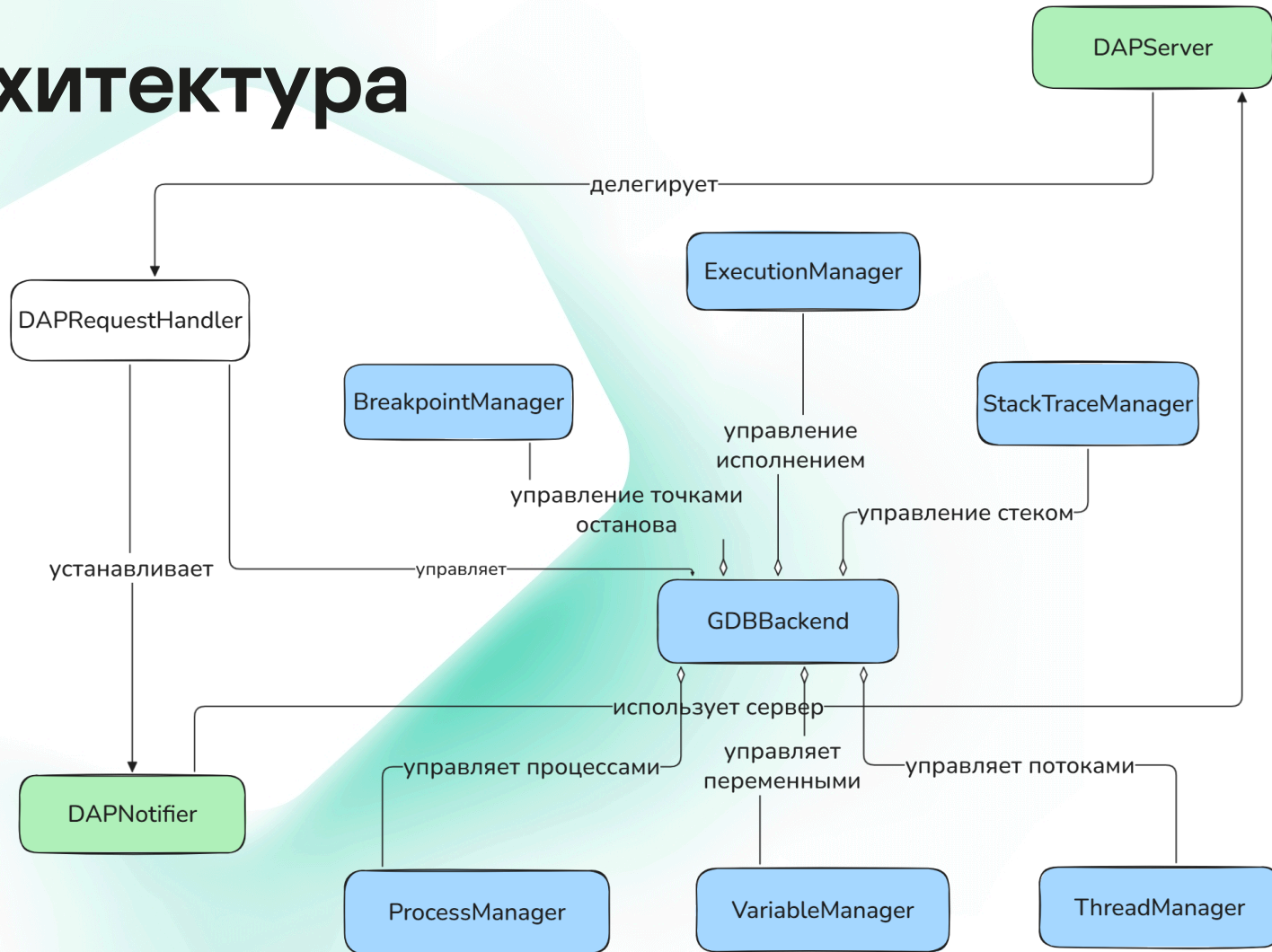
- Проект в стадии развития
- Существующие расширения слабые



Что мы сделали?



Архитектура



Кастомные запросы и сценарии

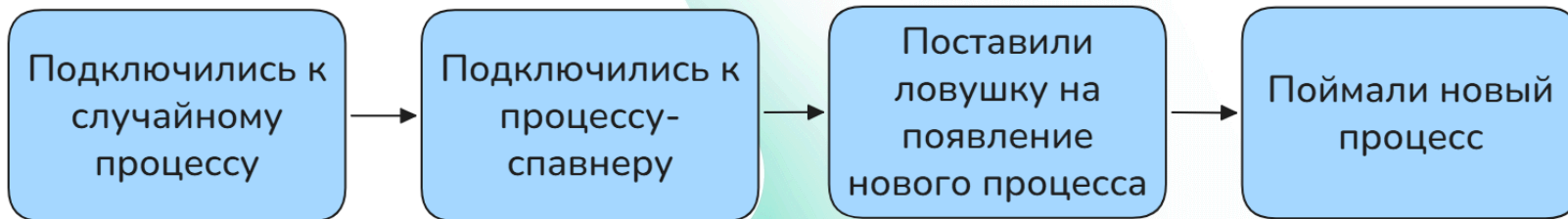
- `handleNewProcess`
- `listProcesses`
- `addInferiors`
- `detachInferiors`
- `selectInferior`
- Запуск приложения через отдельный процесс

Debug Adapter Protocol

The Debug Adapter Protocol (DAP) defines the abstract protocol used between a development tool (e.g. IDE or editor) and a debugger.

☆ Star 1,714

Отладка со старта в Kaspersky OS



С чем столкнулись

pygdbmi 0.11.0.0

```
pip install pygdbmi
```



IoManager implementation #109

Open



Неправильное использование таймаута: [Issue #109](#)

Неочевидности интеграции с VSCode

Почему не обновился UI?

Почему IDE не отправляет
какие-то запросы?

Почему ...?

Почему не обновился UI?

← Invalidated Event

This event signals that some state in the debug adapter has changed and requires that the client needs to re-render the data snapshot previously requested.

Debug adapters do not have to emit this event for runtime changes like stopped or thread events because in that case the client refetches the new state anyway. But the event can be used for example to refresh the UI after rendering formatting has changed in the debug adapter.

Почему IDE не отправляет какие-то запросы?

← Initialized Event

This event indicates that the debug adapter is ready to accept configuration requests (e.g. `setBreakpoints` , `setExceptionBreakpoints`).

- adapters sends `initialized` event (after the `initialize` request has returned)

Смотрим, что получилось

54

Поставим
breakpoint

```
server.cpp 9+ x launch.json
client-server > server.cpp > main
8 int main() {
47
48 while (true) {
49     memset(buffer, 0, sizeof(buffer));
50     ssize_t bytes_read = read(client_socket, buffer, sizeof(buffer) - 1);
51     if (bytes_read <= 0) {
52         std::cout << "The client has disconnected.\n";
53         break;
54     }
55
56     std::cout << "Received: " << buffer << "\n";
57
58     std::string response = "Server response: ";
59     response += buffer;
60
61     send(client_socket, response.c_str(), response.length(), 0);
62 }
63
64 close(fd: client_socket);
65 close(fd: server_fd);
```

PROBLEMS 21 DEBUG CONSOLE TERMINAL OUTPUT SPELL CHECKER PORTS 2

~/MyProjects/test_gdb

>

Подготовим конфигурацию и запустим отладку

55

The image shows the Visual Studio Code interface with the `launch.json` file open in the editor. The configuration is for a C++ server application. The `launch.json` file contains the following JSON configuration:

```
1 {
2   "version": "0.2.0",
3   "configurations": [
4     {
5       "type": "debug_adapter",
6       "request": "launch",
7       "name": "Run and debug",
8       "gdbPath": "",
9       "program": "/home/anastasiia/MyProjects/test_gdb/client-server/server",
10      "verbose": true
11    }
12  ]
13 }
14 }
15 }
```

The interface also shows the `server.cpp` file with 9 lines of code and the `launch.json` file with 15 lines. The terminal at the bottom shows the current directory as `~/MyProjects/test_gdb` and the user as `with anastasiia@gusarova-a`. A blue button labeled "Add Configuration..." is visible in the bottom right corner of the editor area.

Попали в main

client-server > server.cpp > main

```
1 #include <iostream>
2 #include <cstring>
3 #include <unistd.h>
4 #include <netinet/in.h>
5
6 constexpr int PORT = 65432;
7
8 int main() {
9     int server_fd, client_socket;
10    struct sockaddr_in address{};
11    int opt = 1;
12    int addrlen = sizeof(address);
13    char buffer[1024];
14
15    server_fd = socket(AF_INET, SOCK_STREAM, 0);
16    if (server_fd == 0) {
17        perror("socket failed");
18        return 1;
19    }
20
21    setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT, &opt, sizeof(opt));
```

server.cpp 9+ x launch.json

PROCESSES

- anastasiia: /home/anastasiia/...

VARIABLES

- Locals
 - server_fd = -135871592
 - client_socket = 32767
 - address = {sin_family = 1, s...
 - opt = -135825632
 - addrlen = 32767
- WATCH
- CALL STACK
 - main server.cpp 8

PROBLEMS 21 DEBUG CONSOLE TERMINAL OUTPUT SPELL CHECKER PORTS 2 Filter (e.g. text, !exclude, \escape)

[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

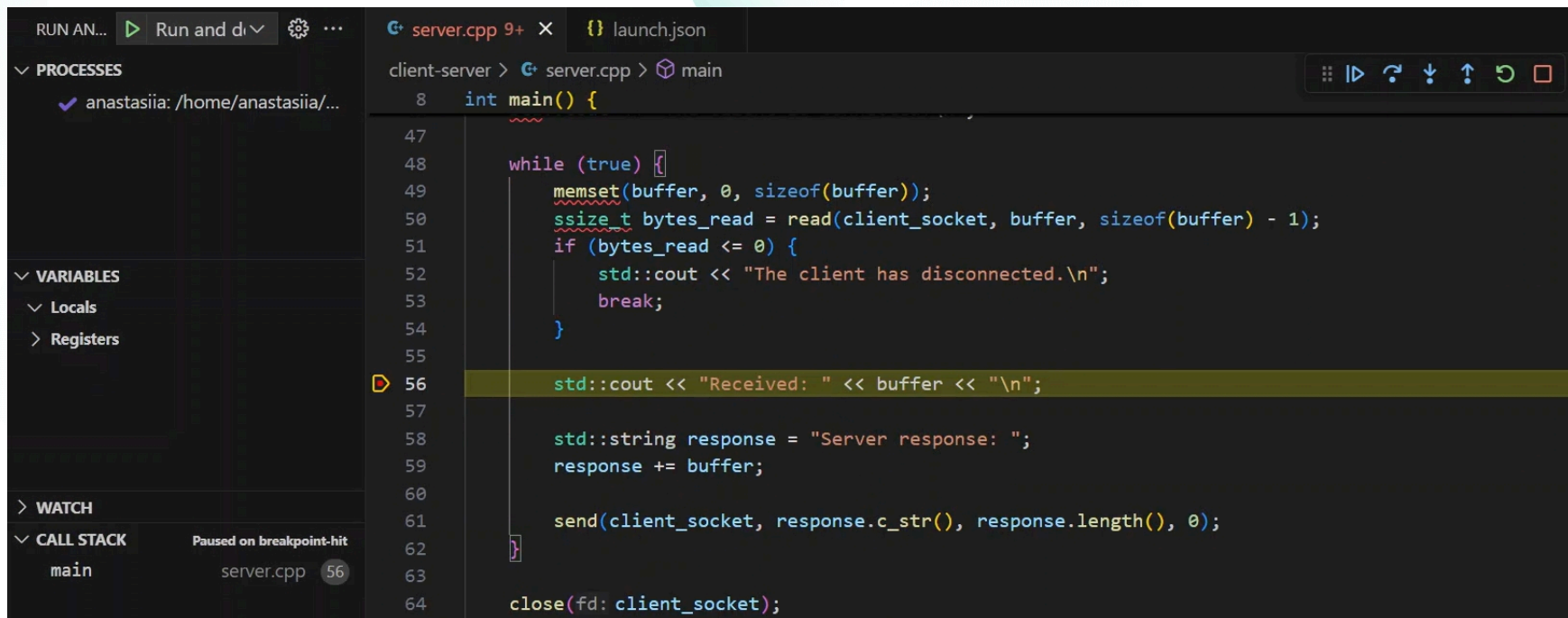
Breakpoint 1, main () at server.cpp:8
8 int main() {

```
PROBLEMS 21  DEBUG CONSOLE  TERMINAL  OUTPUT  SPELL CHECKER  PORTS 2

> ./client-server/client

Enter a message ('exit' to exit): 123
█
```

Запустили клиента,
попали в breakpoint



```
server.cpp 9+ X  {} launch.json

client-server > server.cpp > main
8  int main() {
47
48  while (true) {
49      memset(buffer, 0, sizeof(buffer));
50      ssize_t bytes_read = read(client_socket, buffer, sizeof(buffer) - 1);
51      if (bytes_read <= 0) {
52          std::cout << "The client has disconnected.\n";
53          break;
54      }
55
56  std::cout << "Received: " << buffer << "\n";
57
58  std::string response = "Server response: ";
59  response += buffer;
60
61  send(client_socket, response.c_str(), response.length(), 0);
62
63
64  close(fd: client_socket);
```

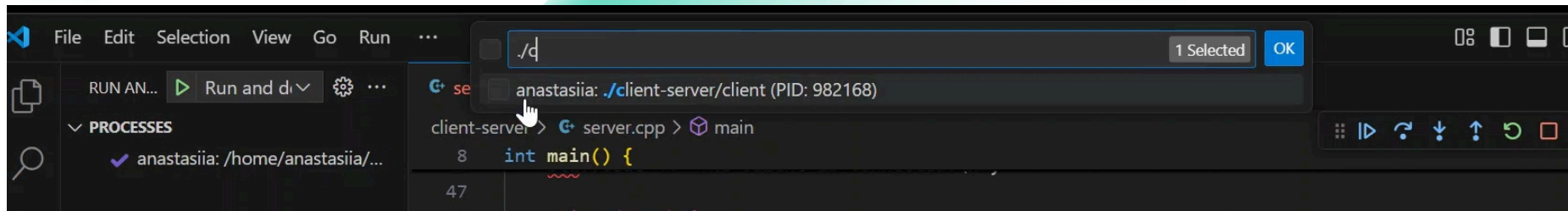
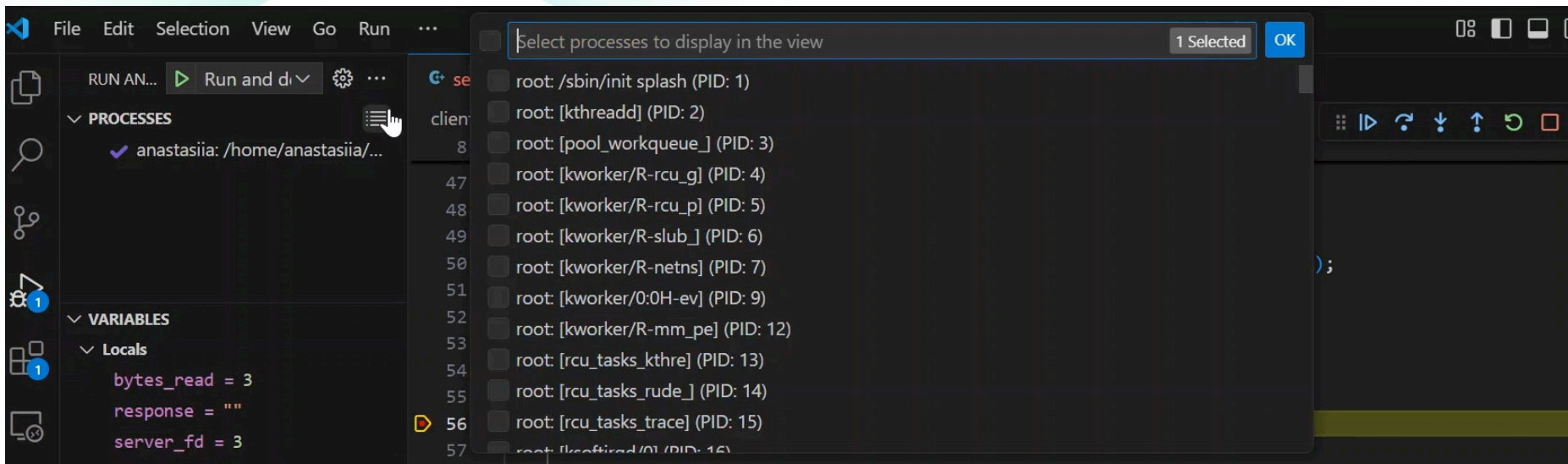
PROCESSES
✓ anastasiia: /home/anastasiia/...

VARIABLES
Locals
Registers

WATCH

CALL STACK
Paused on breakpoint-hit
main server.cpp 56

Добавим в сессию отладки второй процесс



Получилось два процесса

59

The screenshot shows the Visual Studio Code interface during a GDB debug session. The top toolbar includes 'File', 'Edit', 'Selection', 'View', 'Go', 'Run', and a search bar. The left sidebar contains icons for Explorer, Search, Source Control, Run and Debug, and Extensions. The main editor area is split into three panes: 'PROCESSES', 'VARIABLES', and 'CALL STACK'.

PROCESSES: Shows two processes running under the user 'anastasiia':

- anastasiia: /home/anastasiia/...
- anastasiia: ./client-server/clie...

VARIABLES: Shows local variables for the selected process:

- bytes_read = 3
- response = ""
- server_fd = 3
- client_socket = 4
- > address = {sin_family = 2, s...

CALL STACK: Shows the call stack for the 'client' process, which is paused at line 44 of 'client.cpp'. The stack includes:

- server (PAUSED ON ATTACH INFERIORS)
- main (server.cpp:56)
- client (PAUSED)
- __GI__libc_read (./sysdep...)
- main (client.cpp:44) - selected

The main editor displays the code for 'client.cpp' with the following content:

```
client-server > client.cpp > main
8 int main() {
33 while (true) {
36
37     if (message == "exit") {
38         break;
39     }
40
41     send(sock, message.c_str(), message.length(), 0);
42
43     memset(buffer, 0, sizeof(buffer));
44     read(sock, buffer, sizeof(buffer) - 1);
45
46     std::cout << "Response from the server: " << buffer << "\n";
47 }
48
49 close(fd: sock);
50 return 0;
51 }
52
```

The bottom status bar shows 'PROBLEMS 40', 'DEBUG CONSOLE', 'TERMINAL', 'OUTPUT', 'SPELL CHECKER', and 'PORTS 2'. The active terminal shows the path './client-server/client'.

```
39
40
41     send(sock, message.c_str(), message.length(), 0);
42
43     memset(buffer, 0, sizeof(buffer));
44     read(sock, buffer, sizeof(buffer) - 1);
45
46     std::cout << "Response from the server: " << buffer << "\n";
47 }
48
49 close(fd: sock);
50 return 0;
51 }
52
```

Поставим новый
breakpoint (в client)
и попадём в него

The screenshot shows a debugger interface with the following components:

- Process List:** Shows two processes: `anastasiia: /home/anastasiia/...` and `anastasiia: ./client-server/clie...`. The client process is checked.
- Variables Panel:** Shows local variables: `sock = 3`, `serv_addr = {sin_family = 2, ...}`, `buffer = "Server response: 1...`, and `message = "123"`.
- Call Stack:** Shows the call stack with `server` (PAUSED) and `client` (PAUSED ON BREAKPOINT-HIT). The current frame is `main` in `client.cpp` at line 46.
- Code Editor:** Shows the source code of `client.cpp` with a breakpoint (red dot) on line 46: `std::cout << "Response from the server: " << buffer << "\n";`. The code is identical to the one in the top image.
- Debugger Controls:** A toolbar at the top right contains icons for stepping through code. A tooltip for the `Continue (F5)` button is visible.

Попадём снова в первый breakpoint (server)

The screenshot shows a C++ IDE with a server program paused at a breakpoint. The call stack on the left indicates the 'server' thread is paused on line 56 of 'server.cpp'. The main code editor shows the following code:

```
51     if (bytes_read <= 0) {
52         std::cout << "The client has disconnected.\n";
53         break;
54     }
55
56     std::cout << "Received: " << buffer << "\n";
57
58     std::string response = "Server response: ";
59     response += buffer;
60
61     send(client_socket, response.c_str(), response.length(), 0);
62 }
63
64 close(fd: client_socket);
65 close(fd: server_fd);
```

The terminal window at the bottom shows the client's interaction with the server:

```
> ./client-server/client
Enter a message ('exit' to exit): 123
Response from the server: Server response: 123
Enter a message ('exit' to exit): 555
```

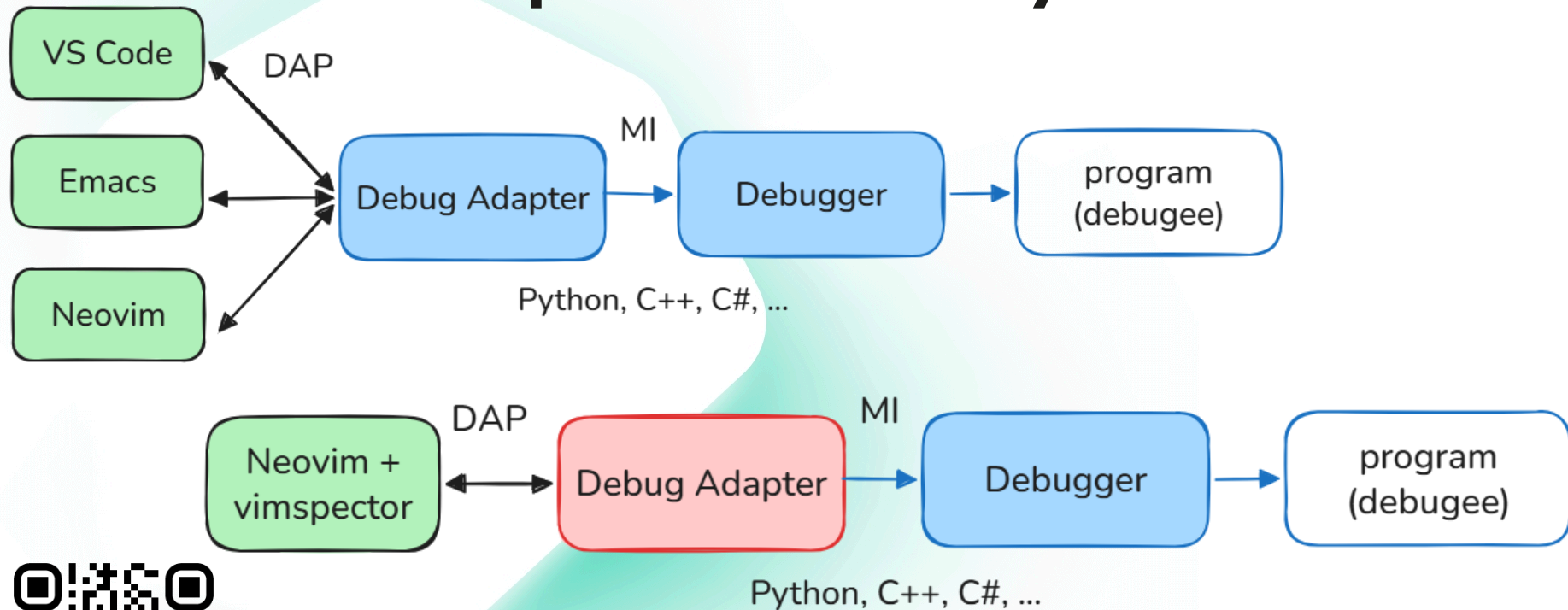
Отладка в KOS

The screenshot displays a debugger interface with the following components:

- PROCESSES:** Client (PID: 16) and Server (PID: 15).
- VARIABLES:** Locals section showing `value = 28913359`, `req`, and `res`.
- WATCH:** Empty.
- CALL STACK:** Shows `ping.Client (16.16)` (PAUSED ON BREAKPOINT-HIT) and `ping.Server (15.15)` (PAUSED). The current frame is `ping /proddir/build/examples/base/ping/arm64_kos/src/c...`.
- SOURCE CODE:** A C++ function `pong` is shown, with a breakpoint at line 37. The code includes comments in Russian and a `printf` statement that is highlighted in yellow.

```
23 {
24     ... struct ping_Connection_Ping_req req;
25     ... struct ping_Connection_Ping_res res;
26
27
28     req.value = value;
29     ... /**
30     ... * Вызов интерфейсного метода ping_Connection_Ping.
31     ... * Серверу будет отправлен запрос для вызова метода Ping интерфейса
32     ... * control.connectionimpl с аргументом value. Вызывающий поток блокируется
33     ... * до момента получения ответа от сервера.
34     ... */
35     ... if (ping_Connection_Ping(&proxy.base, &req, NULL, &res, NULL) == rcOK)
36     ... {
37     ...     fprintf(stderr, "%s Ping(%d), result = %d\n", Tag, value, res.result);
38     ...     value = res.result;
39     ... }
40     ... else
41     ... {
42     ...     fprintf(stderr, "%s Ping(%d), failed\n", Tag, value);
43     ... }
44
45     return value;
46 }
47
48 static uint32_t pong(uint32_t value)
49 {
```

Приятный бонус



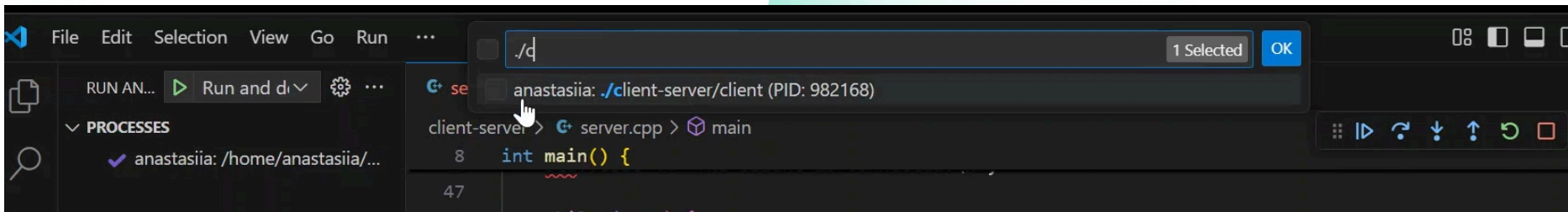
vimspector

Выводы

- Узнали о строении тулинга отладки
- Углубились в то, как вообще работает отладка
- Поговорили об удалённой отладке
- Познакомились с существующим отладочным тулингом
- Узнали про отладку в KasperskyOS

Что получили, реализовав свой Debug Adapter

- Удобный интерфейс для отладки нескольких процессов
- Избавились от существующих проблем
- Открыли возможности для улучшений



Было

The screenshot shows a debugger interface with the following details:

- Exception:** A red box displays "Exception has occurred. X Unknown stopping event".
- CALL STACK:** Shows the client process [1846955] paused on an exception at line 44:1 of client.cpp, with main() as the caller.
- Code:** Lines 41-44 of client.cpp are visible, with line 44 containing `read(fd: sock, bu`.
- WATCH:** The WATCH section is currently empty.

Стало

The screenshot shows a debugger interface with the following details:

- PROCESSES:** The server process [1845606] is shown as "PAUSED ON ATTACH INFERIORS".
- CALL STACK:** Shows the server process paused at line 56 of server.cpp, with main() as the caller.
- Code:** Lines 47-56 of server.cpp are visible, with line 56 containing `while`.
- VARIABLES:** The Locals section shows:
 - `bytes_read = 3`
 - `response = ""`
 - `server_fd = 3`
 - `client_socket = 4`
 - `address = {sin_family = 2, s...`
- WATCH:** The WATCH section is currently empty.

Было

The screenshot shows a debugger interface with a red error message: "Exception has occurred. Terminated". The call stack is expanded to show the client thread [1851151] paused on an exception at line 46 of client.cpp. The server thread [1850958] is also shown, paused at line 54 of server.cpp. The source code on the right shows the client's main function and the server's main function.

```
43 memset(s, 0, sizeof(s));
44 ..... read(fd:
45
46 ..... std::cout
47 ..... }
48
49 ..... close(fd: soc
50 ..... return 0;
51 ..... }
52
```

Exception has occurred. Terminated

WATCH

CALL STACK

- server [1850958] PAUSED
 - libc.so.6!_GI__libc_read(int fd, v
 - main() server.cpp 54:1
- client [1851151] PAUSED ON EXCEPTION
 - main() client.cpp 46:1

Стало

The screenshot shows the debugger's call stack and a terminal window. The call stack shows the client thread [1851151] paused on a signal-received at line 10 of client.cpp. The terminal window shows the message: "Thread 1.1 'client' received signal SIGTERM, Terminated. [Switching to Thread 0x7ffff7e89740 (LWP 1877137)] main () at client.cpp:10 10 int main() { Program terminated with signal SIGTERM, Terminated. The program no longer exists."

CALL STACK Paused on signal-received

- __libc_accept ./sysdeps/unix/sysv/linux/accept.c 26
- main server.cpp 44

Thread 1.1 "client" received signal SIGTERM, Terminated.
[Switching to Thread 0x7ffff7e89740 (LWP 1877137)]
main () at client.cpp:10
10 int main() {
Program terminated with signal SIGTERM, Terminated.
The program no longer exists.

Полезные ссылки

- [KasperskyOS Community Edition](#)
- [Debug Adapter Protocol \(что это\)](#)
- [Debug Adapter Protocol \(спецификация\)](#)
- [GDB/MI интерфейс](#)
- [Серия статей по отладке](#)
- [DWARF](#)
- [Inferiors](#)
- [Удалённая отладка](#)
- [Remote Protocol](#)
- [Gdbserver](#)
- [GDB-сервер KasperskyOS](#)
- [C/C++ for Visual Studio Code](#)
- [MIEngine](#)
- [GDB в режиме DAP](#)
- [Pygdbmi](#)
- [Vimspector](#)
- [Отладка в KasperskyOS](#)

Спасибо!

Гусарова Анастасия

Младший разработчик группы разработки инструментария
KasperskyOS, «Лаборатория Касперского»

Anastasiya.Gusarova@kaspersky.com

KasperskyOS Community Edition



kaspersky

Попробовать отладку

