C++ Russia 2026

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

Гусарова Анастасия

KasperskyOS

О чём поговорим

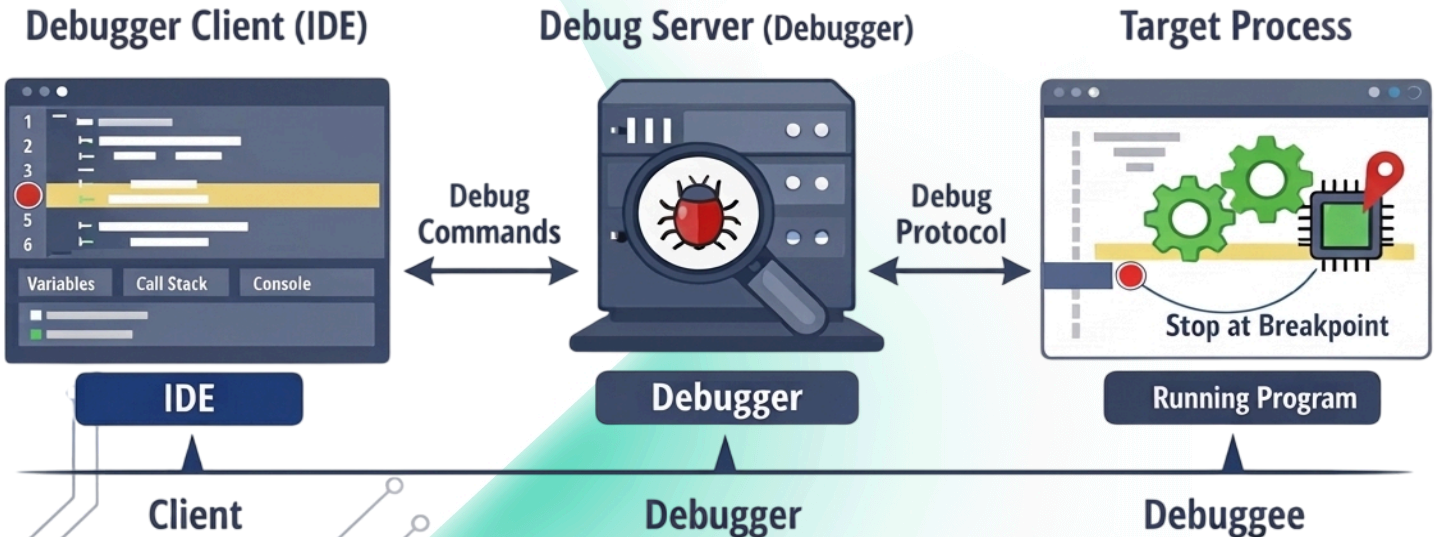
GDB: The GNU Project Debugger

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

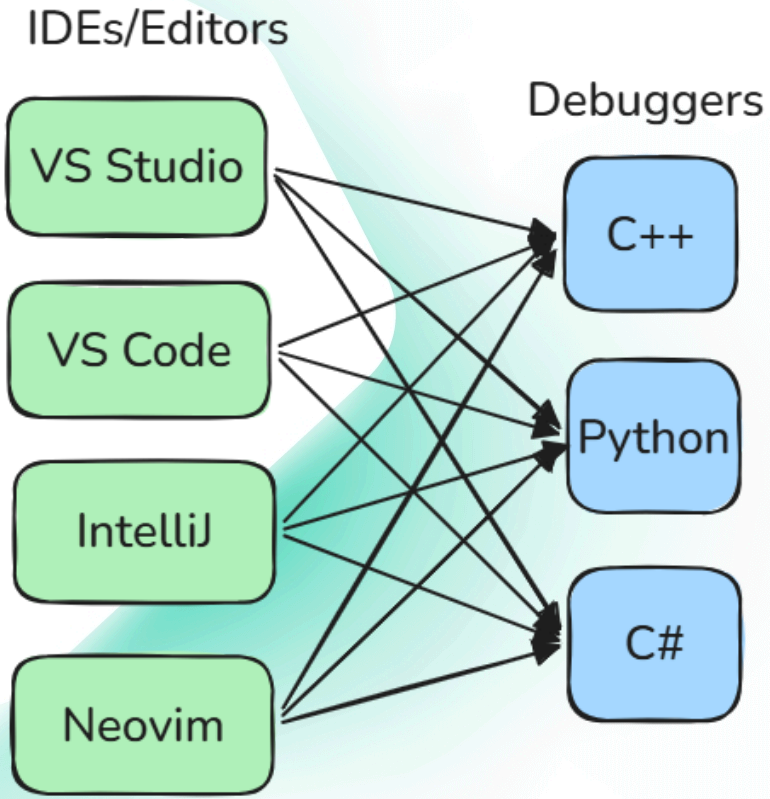
kaspersky OS

Be Immune

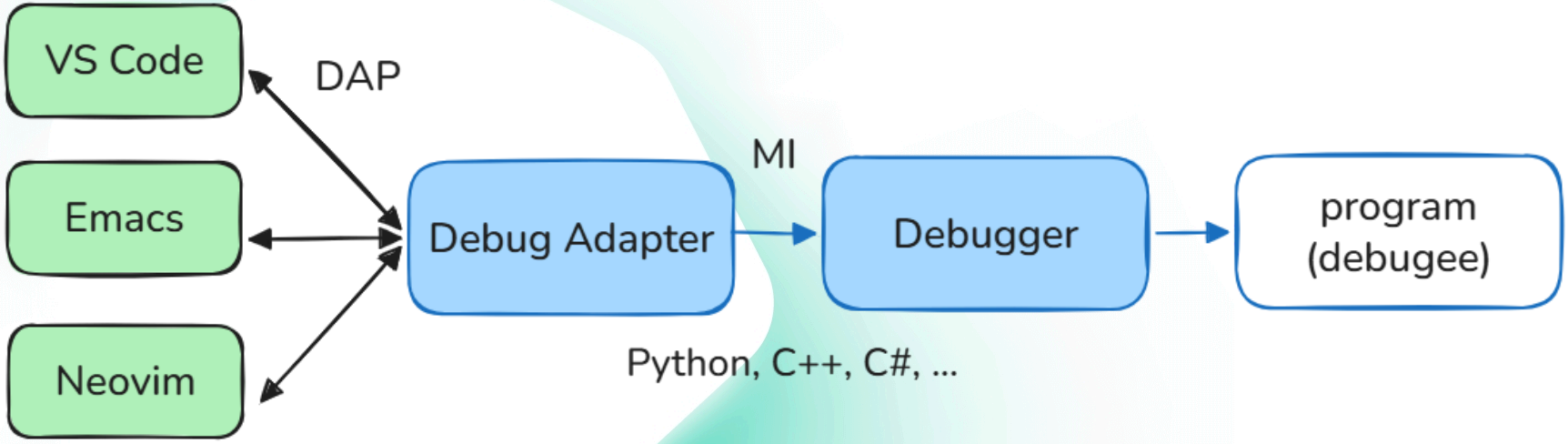
Тулинг отладки



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


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

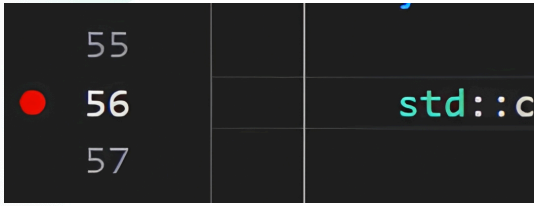
Спецификация MI

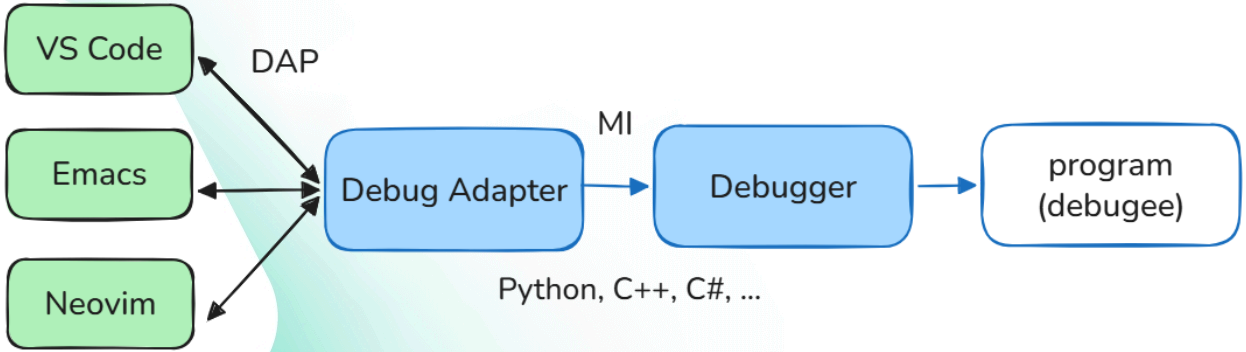
Пример MI-ответа


```
^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

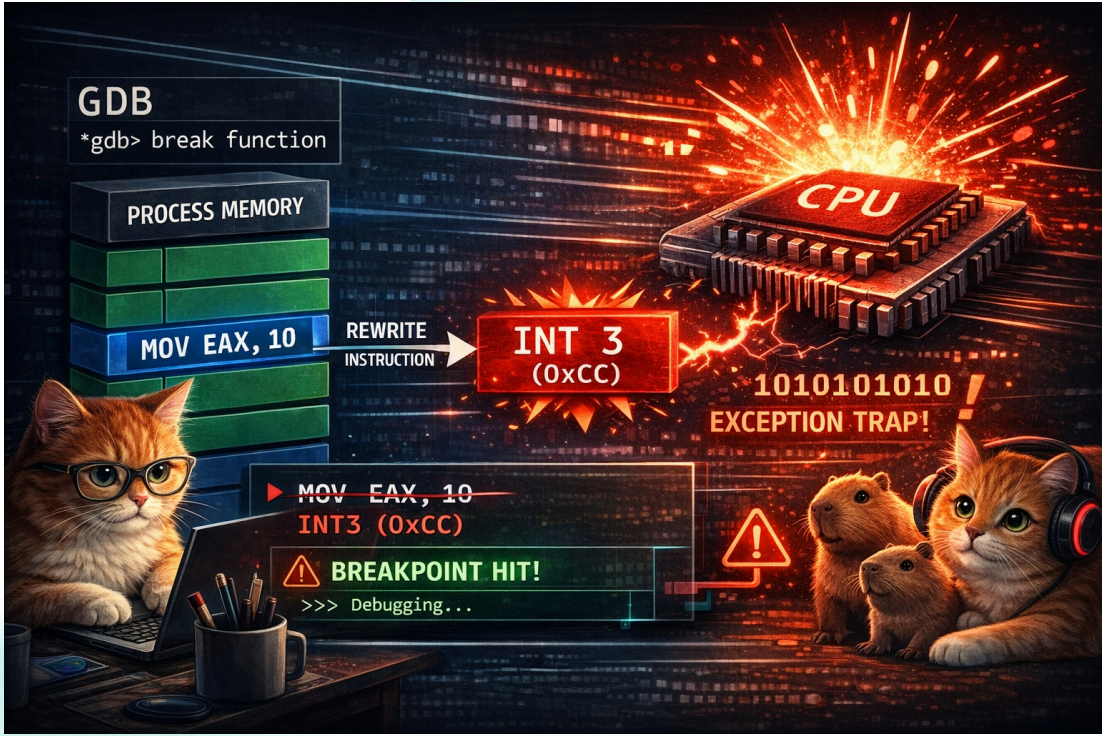




- IDE отправляет запрос в debug adapter (через DAP)
- debug adapter отправляет запрос в GDB (через MI)
- GDB ставит breakpoint в процессе

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

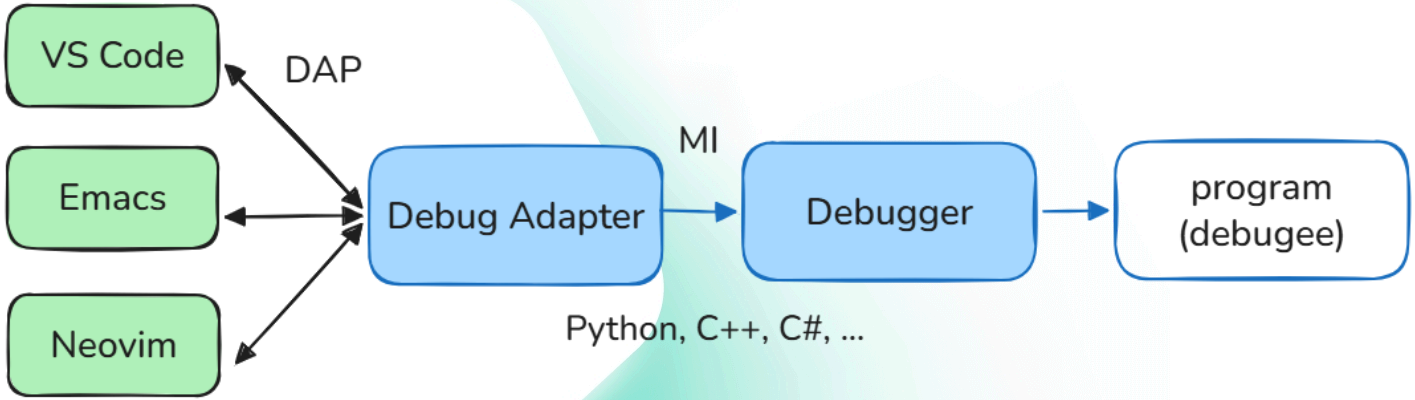
x86:

- 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

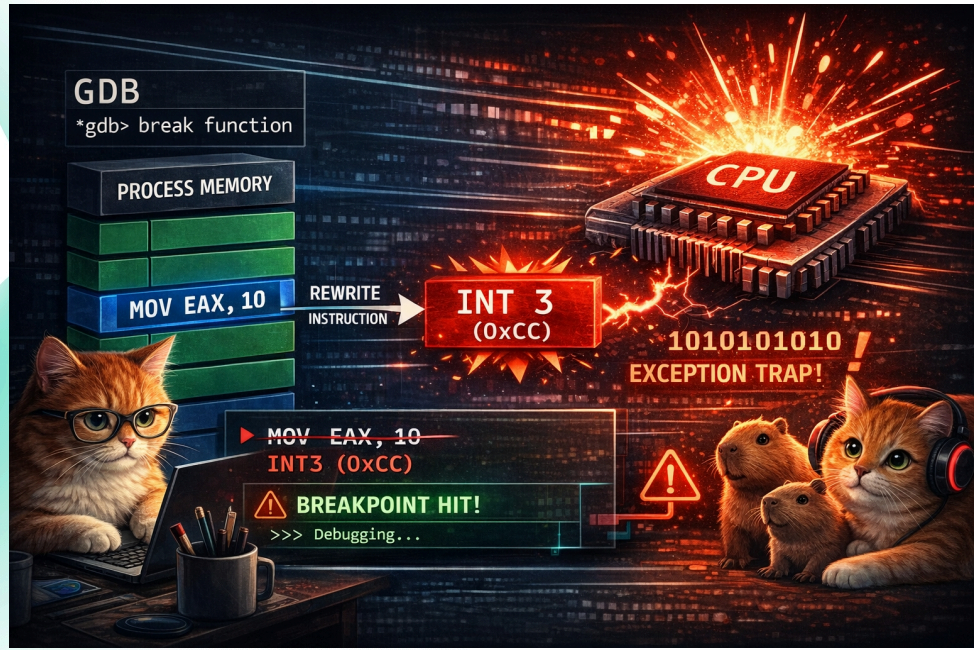
Попадаем в breakpoint

- 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)

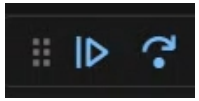
SIGTRAP – сигналы, связанные с отладкой

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

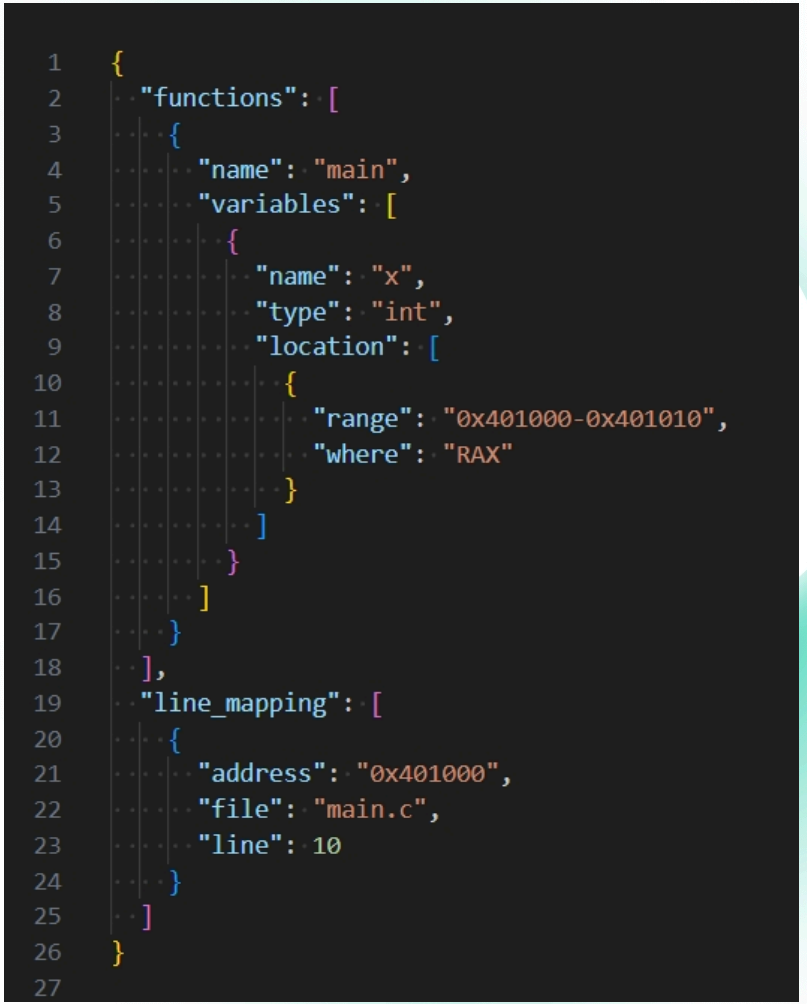
DWARF

Формат отладочной информации, которая добавляется компилятором
Содержит:

- переменные
- функции
- типы
- соответствие кода и памяти

DWARF

DWARF

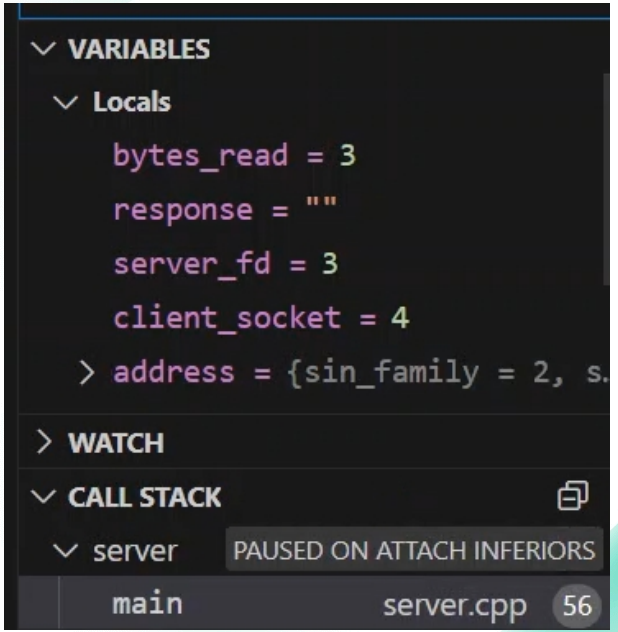


```
{
  "functions": [
    {
      "name": "main",
      "variables": [
        {
          "name": "x",
          "type": "int",
          "location": [
            {
              "range": "0x401000-0x401010",
              "where": "RAX"
            }
          ]
        }
      ]
    }
  ],
  "line_mapping": [
    {
      "address": "0x401000",
      "file": "main.c",
      "line": 10
    }
  ]
}
```

DWARF - набор секций

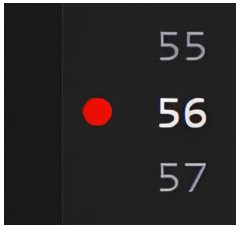
- .debug_info - **что** есть в программе
- .debug_line - **где** это в исходниках
- .debug_loc - **где** это лежит во время выполнения

Переменные



- 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

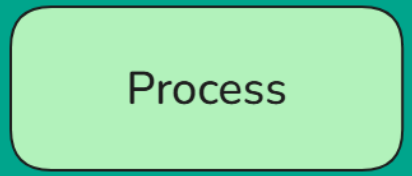
Умеем выяснить причину остановки

Знаем, что такое DWARF

Интерпретация процессов в GDB

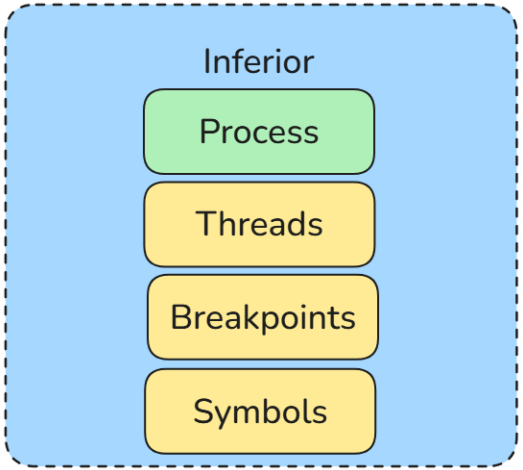
Процесс в системе

- Просто PID
- Выполняется сам по себе



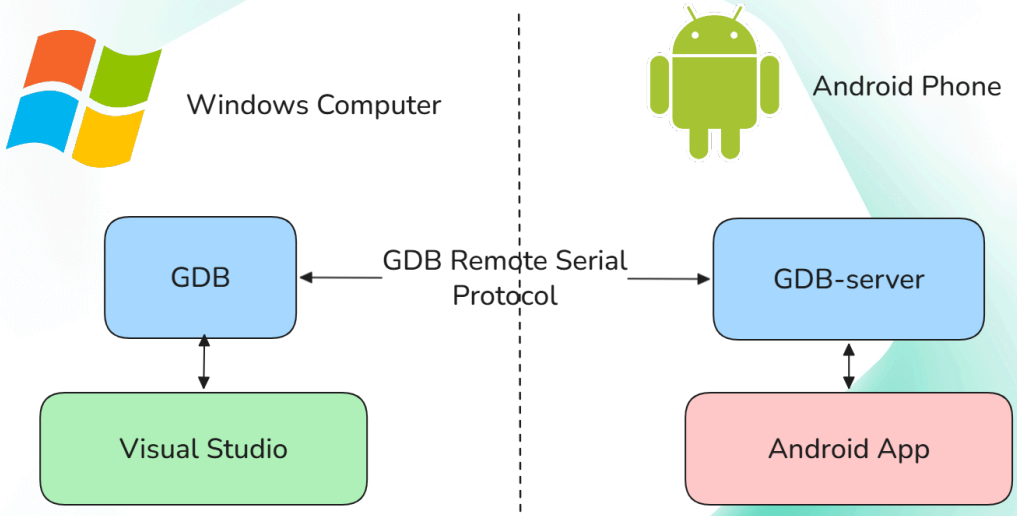
Inferior в GDB

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



Inferiors

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

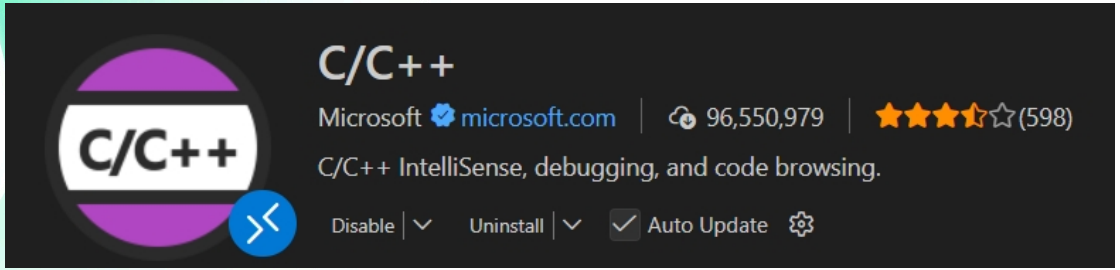
gdbserver (Linux)

- 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)

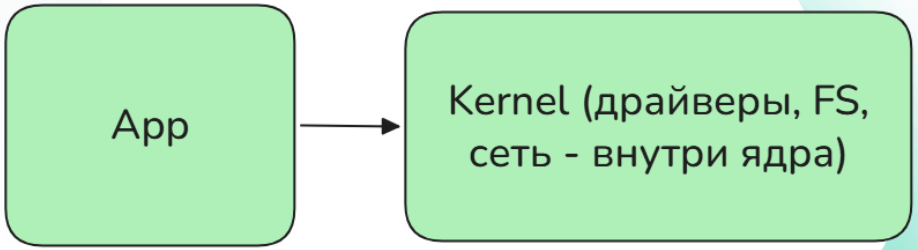
- Ядерный gdbstub
- Одно соединение
- Подключение к случайному процессу
- Останавливалась вся система (до недавнего времени)

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

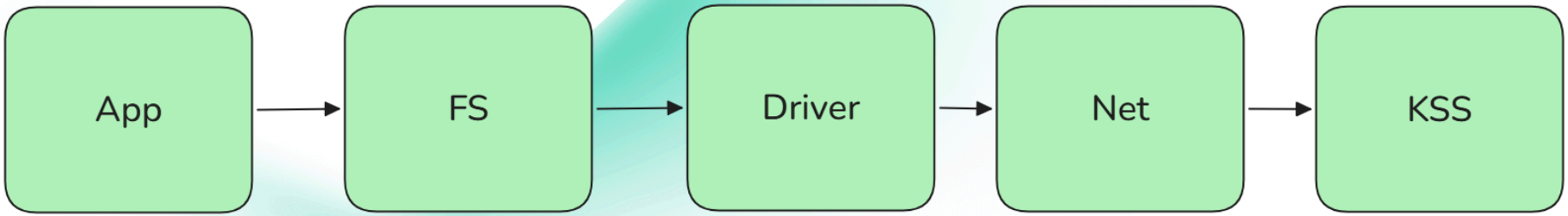


Архитектурные различия Linux и KasperskyOS

Linux



Kaspersky OS

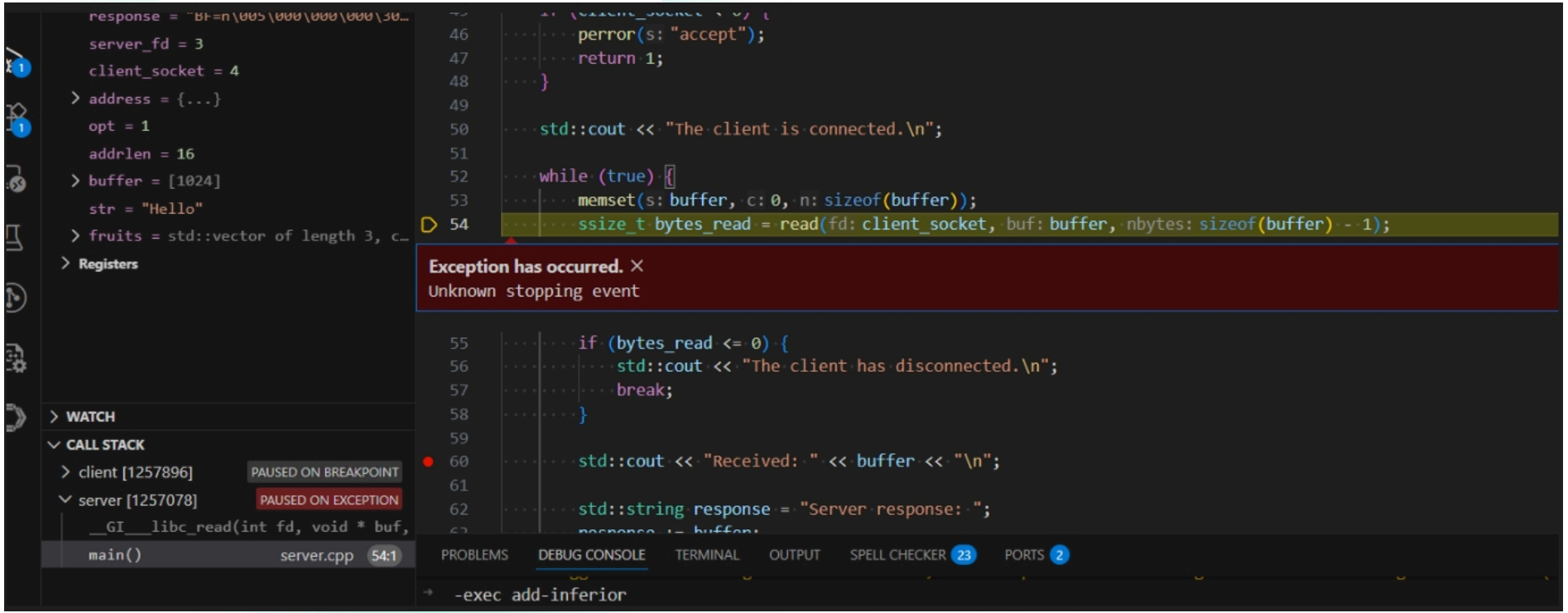


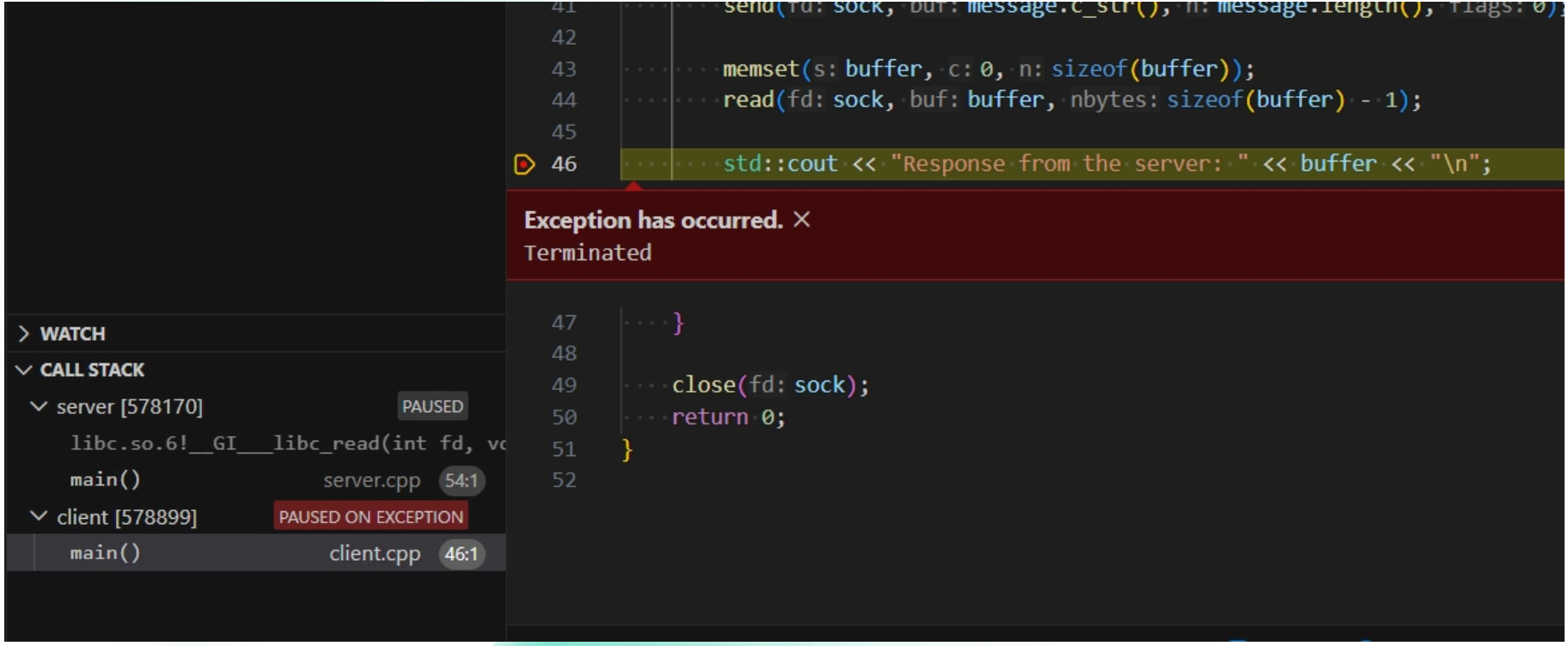
(каждый блок - отдельный процесс)

Попытки продебажить несколько Linux процессов в одной сессии

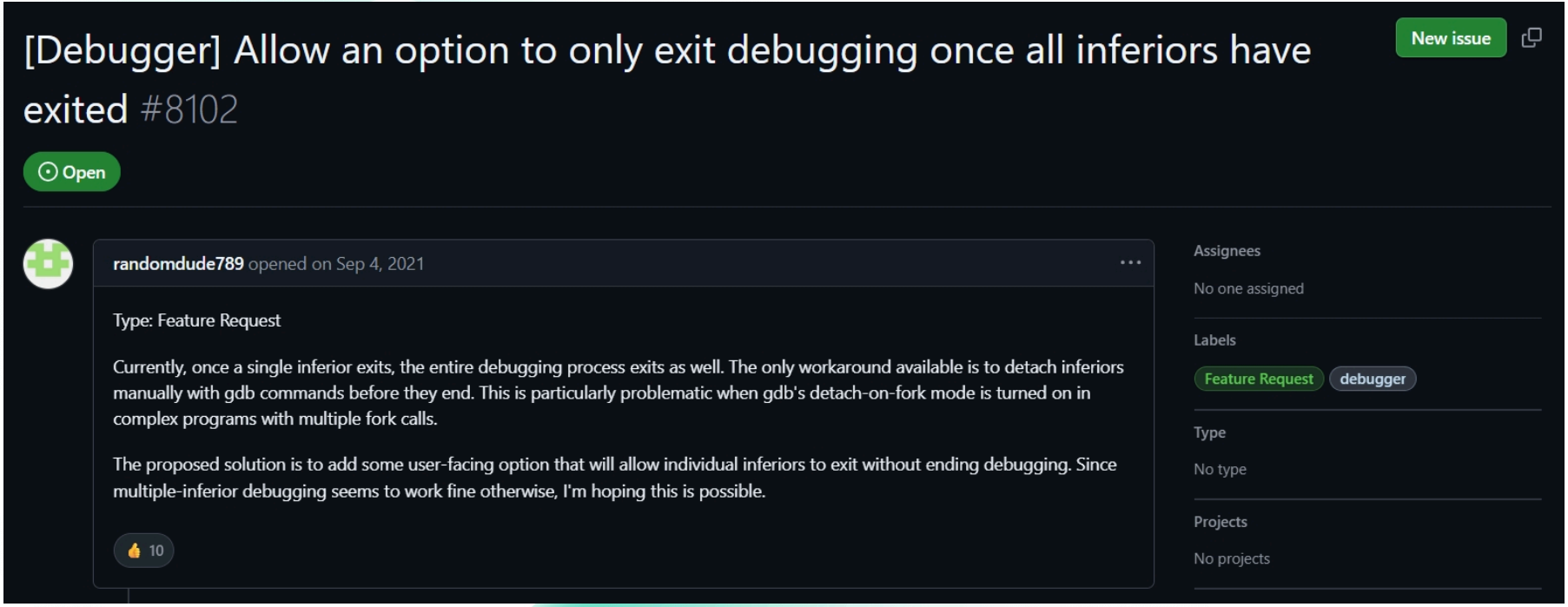
- -exec add-inferior
- -exec inferior 2
- ps aux | grep ~/client-server/client
- -exec attach 1855572

Попытки продебажить несколько Linux процессов в одной сессии

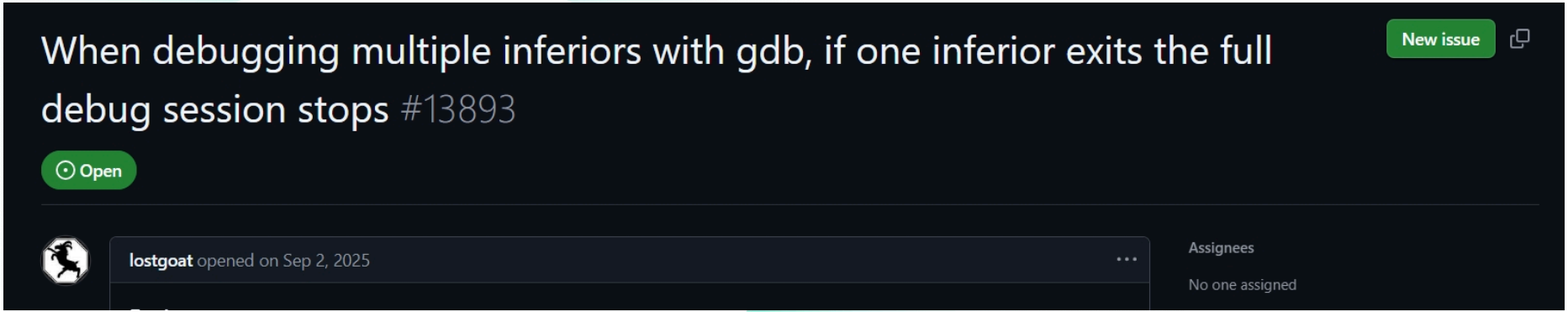




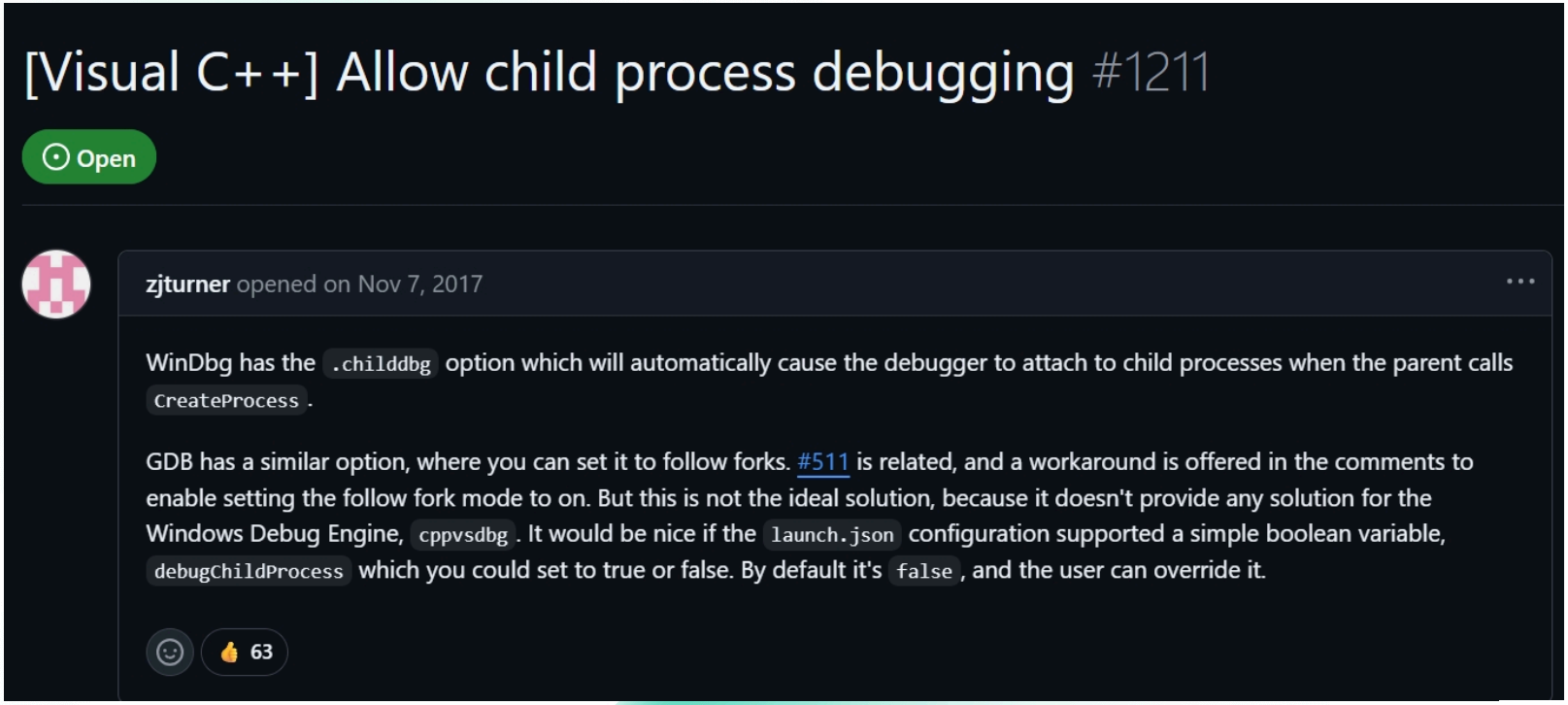
Завершение одного из отлаживаемых процессов привело к завершению всей сессии отладки



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



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

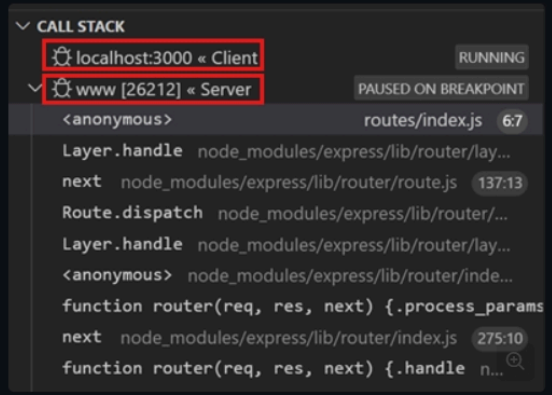


Нет нормальной поддержки отладки дочерних процессов: Issue #1211

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*:

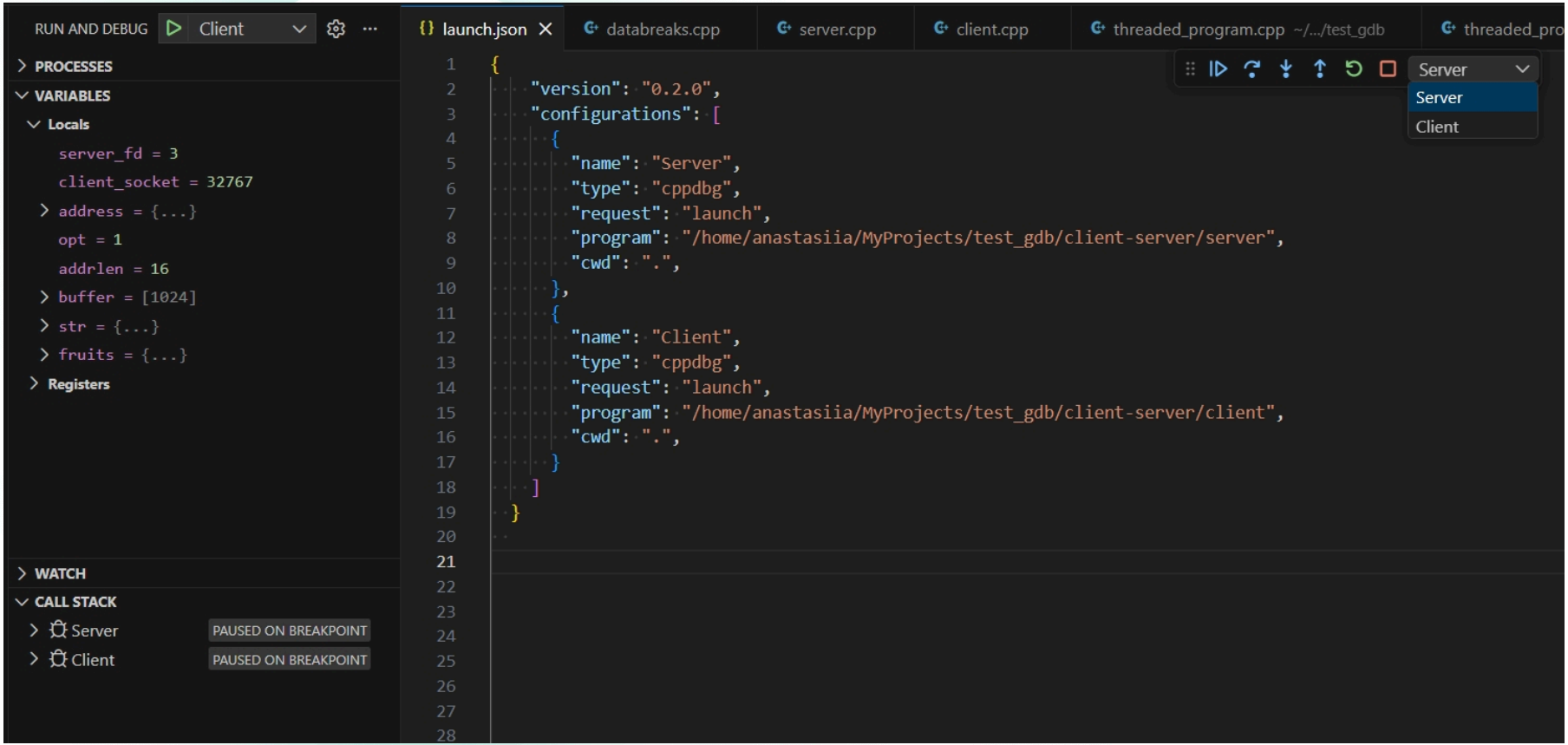
- The individual sessions now show up as top-level elements in the **CALL STACK** view.



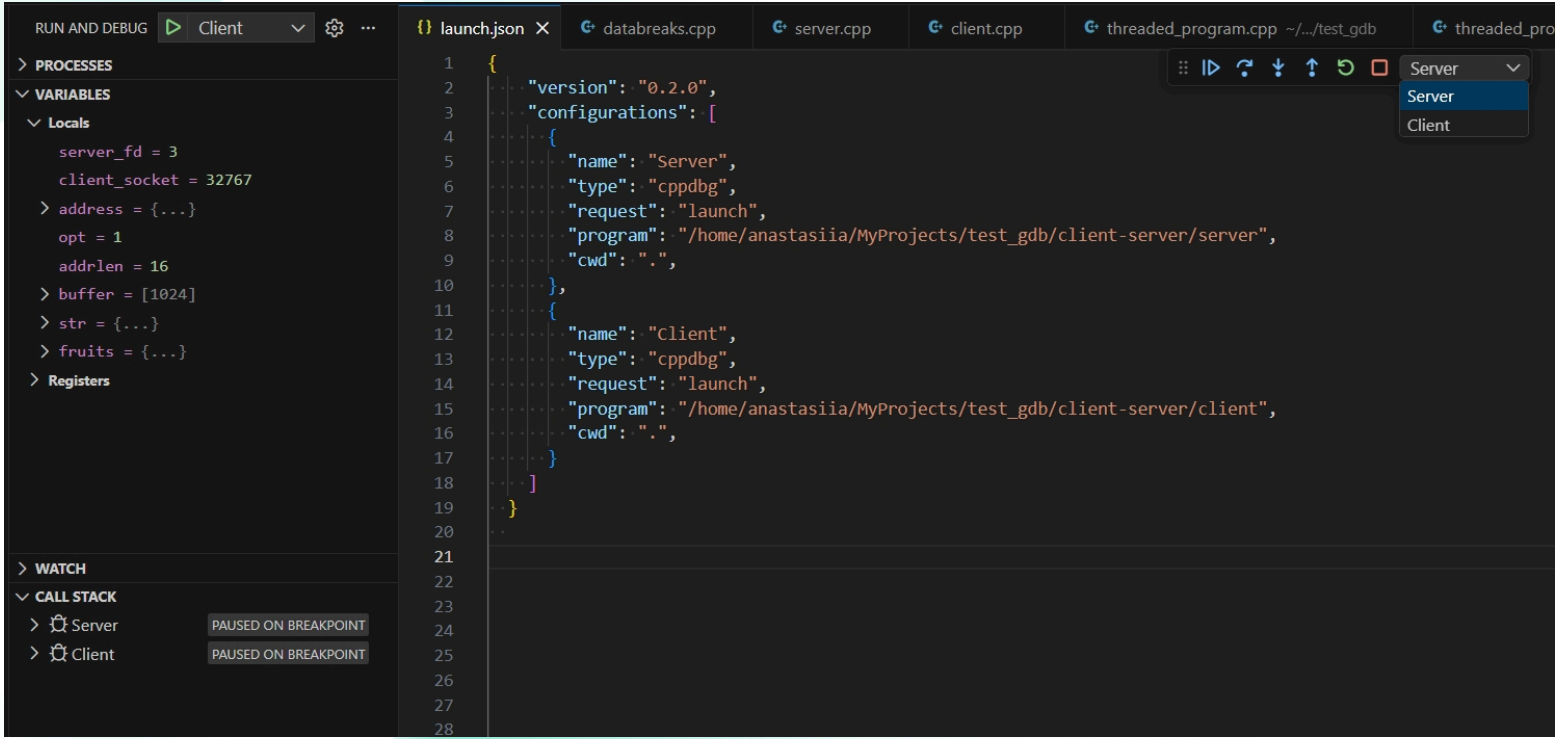
- The debug toolbar shows the currently **active session** (and all other sessions are available in a dropdown menu).

[Отладка нескольких процессов в несколько сессий]

Отладка нескольких сессий

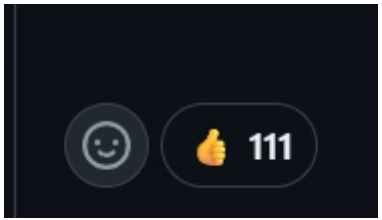


Отладка нескольких сессий



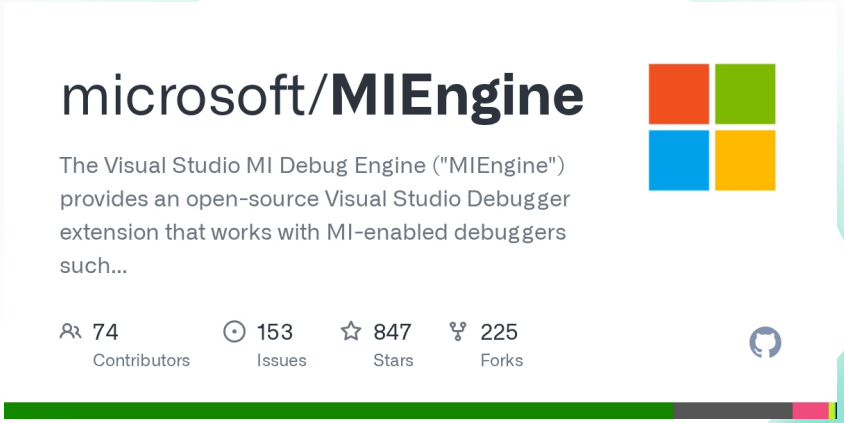
- Нужно настраивать launch.json под каждый сценарий
- В KasperskyOS не поддержана отладка в несколько сессий

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

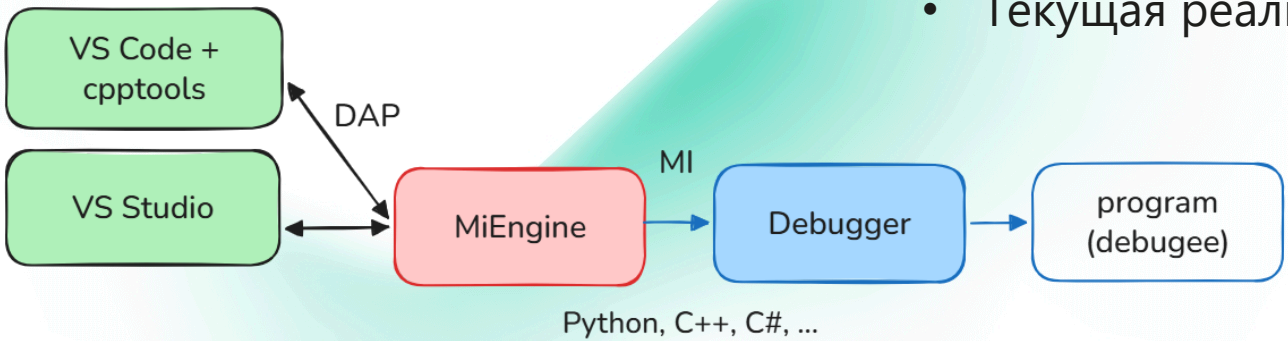


Будем делать сами...

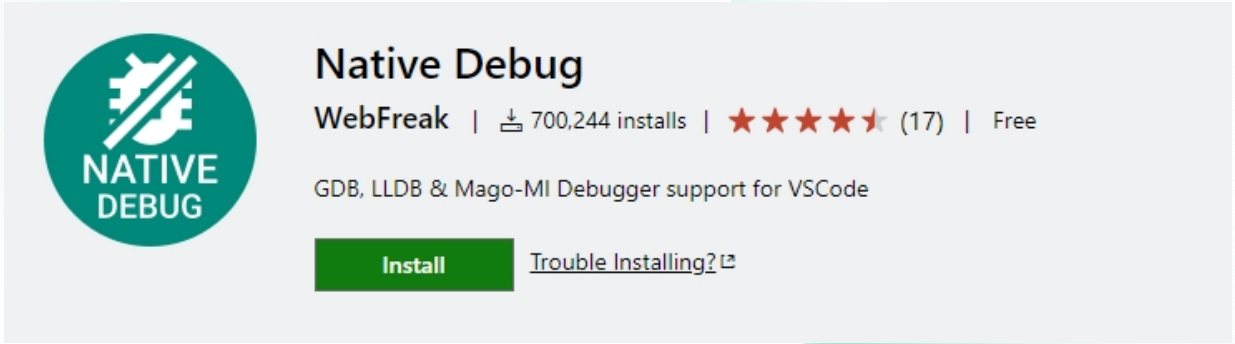
MIEngine



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

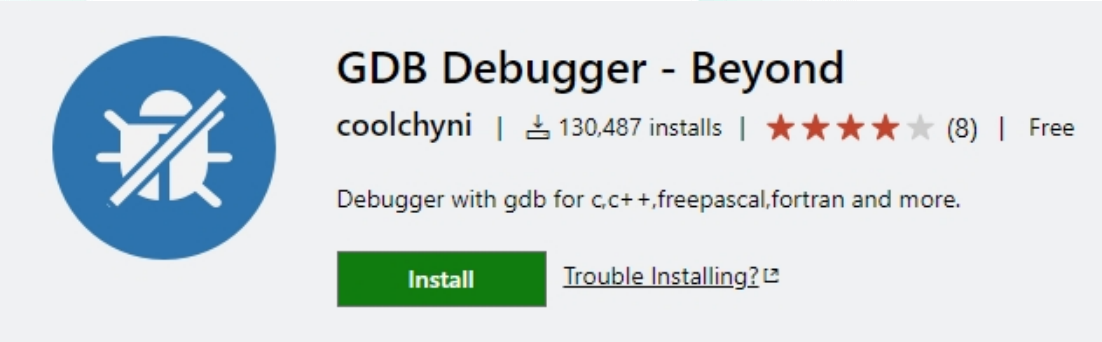


Native Debug



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

GDB Debugger - Beyond



- 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 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

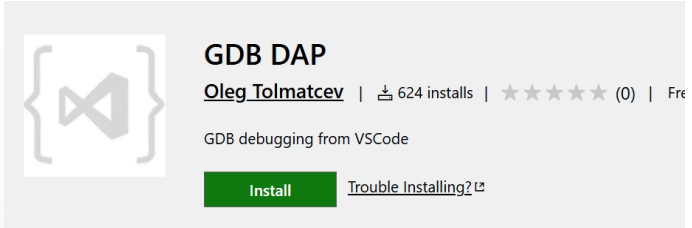
summary | shortlog | log | commit | commitdiff | tree

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
	ssh://sourceware.org/git/binutils-gdb.git
	https://sourceware.org/git/binutils-gdb.git

shortlog

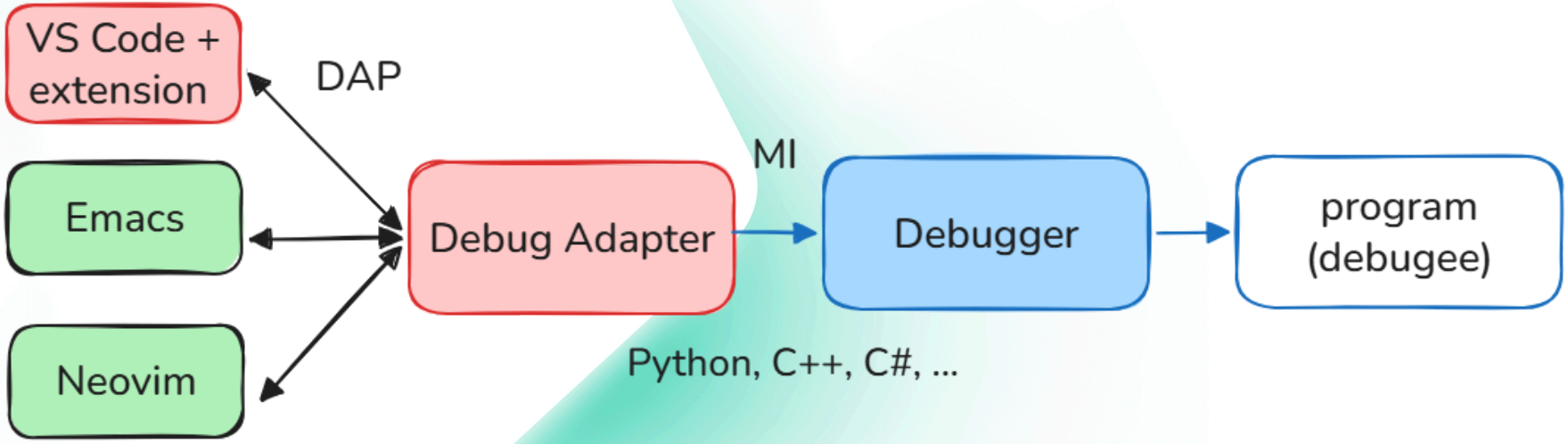
2 hours ago	Aditya Vidyadhar...	**Fix asertion failure while analysing core files in...** master	commit | commitdiff | tree
2 hours ago	Tom de Vries	**[gdb] Add c_ctrl/c_unctrl**	commit | commitdiff | tree
7 hours ago	Alan Modra	**aout, ecoff and som free_cached_info**	commit | commitdiff | tree
7 hours 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 Vrany	**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 Vrany	**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

...

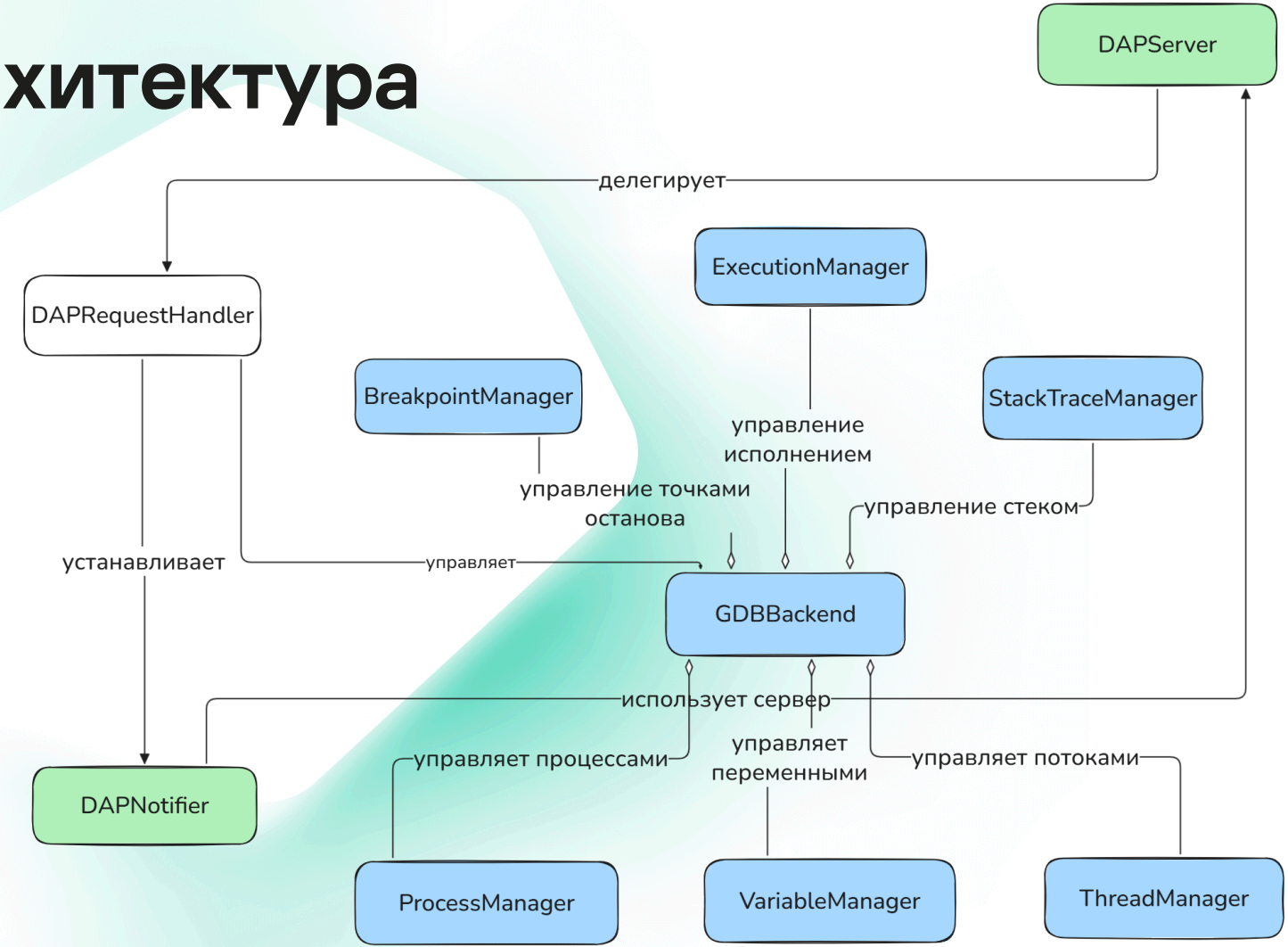


- Проект в стадии развития
- Существующие расширения слабые

Что мы сделали?



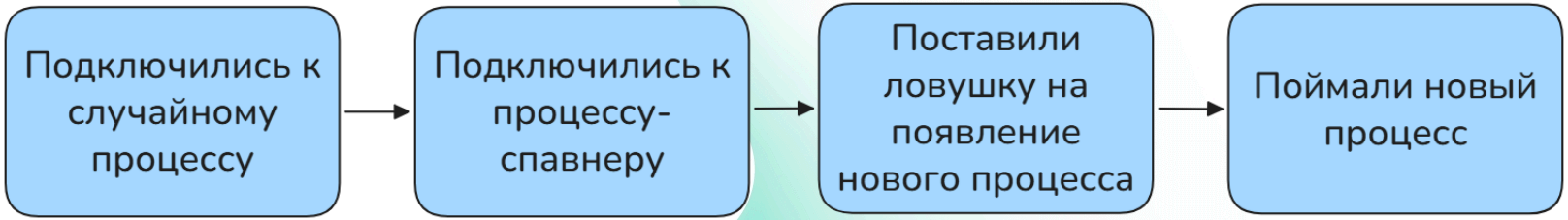
Архитектура



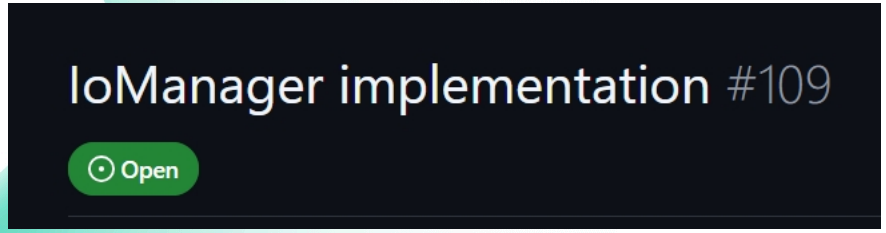
Кастомные запросы и сценарии

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

Отладка со старта в Kaspersky OS



С чем столкнулись



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?

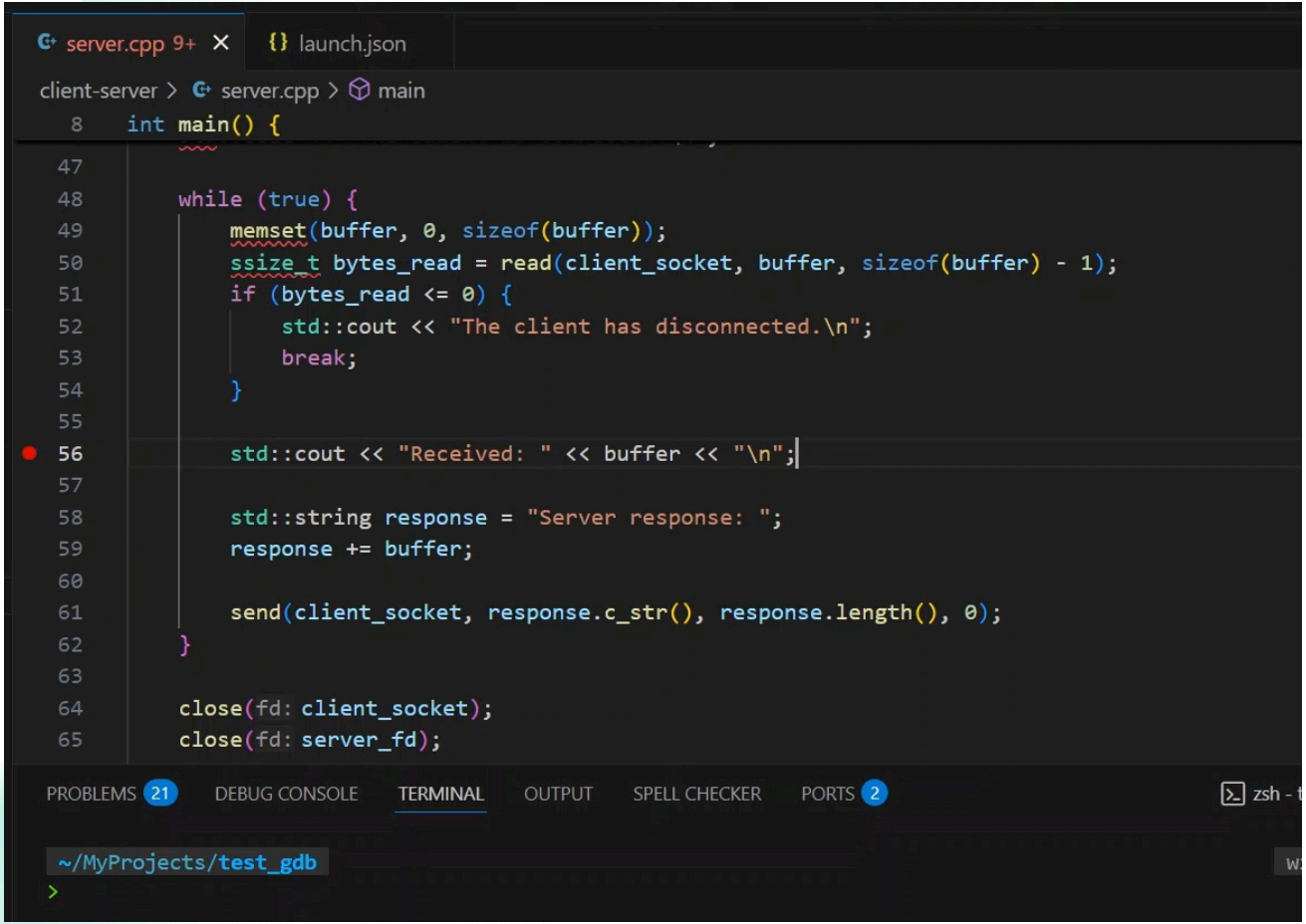
⬅️ Initialized Event

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

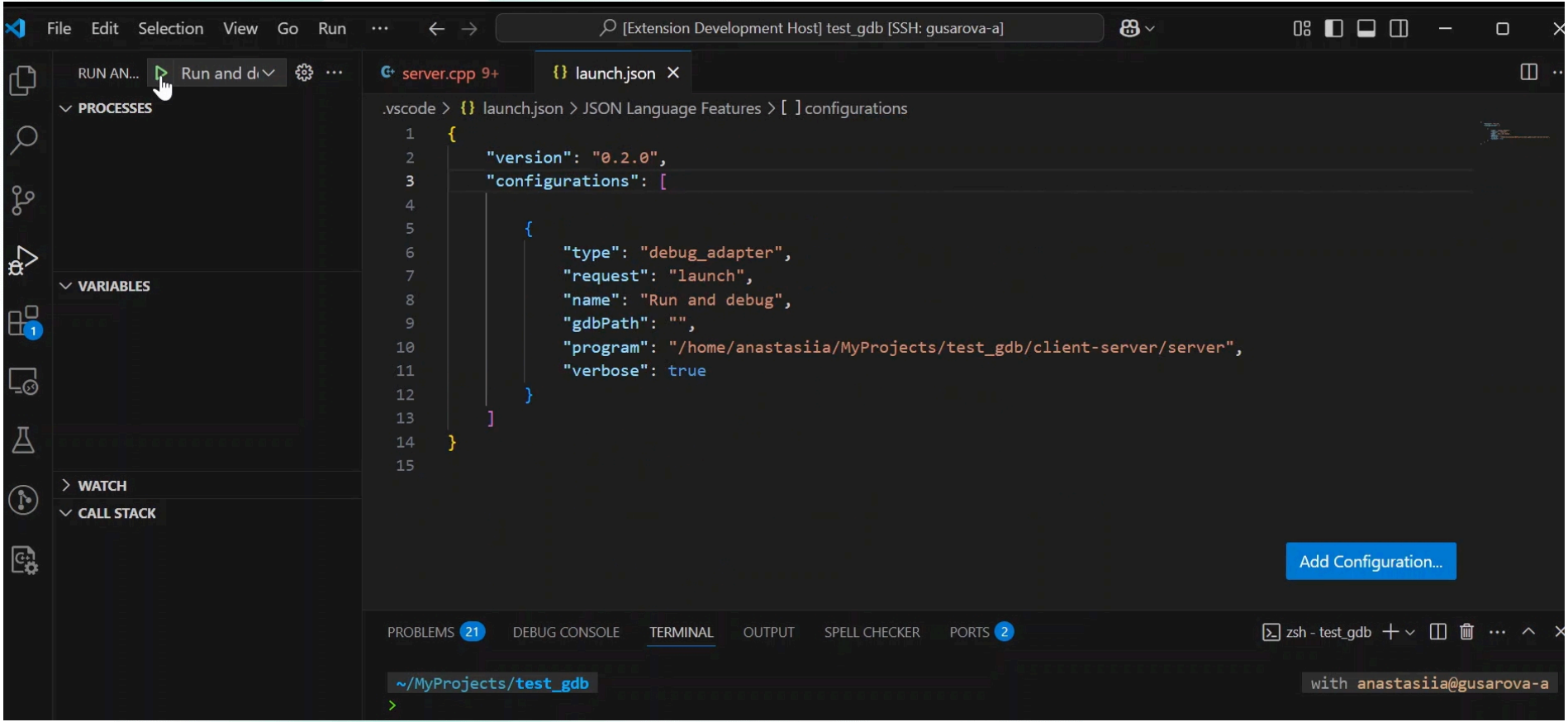
- adapters sends `initialized` event (after the `initialize` request has returned)

Смотрим, что получилось

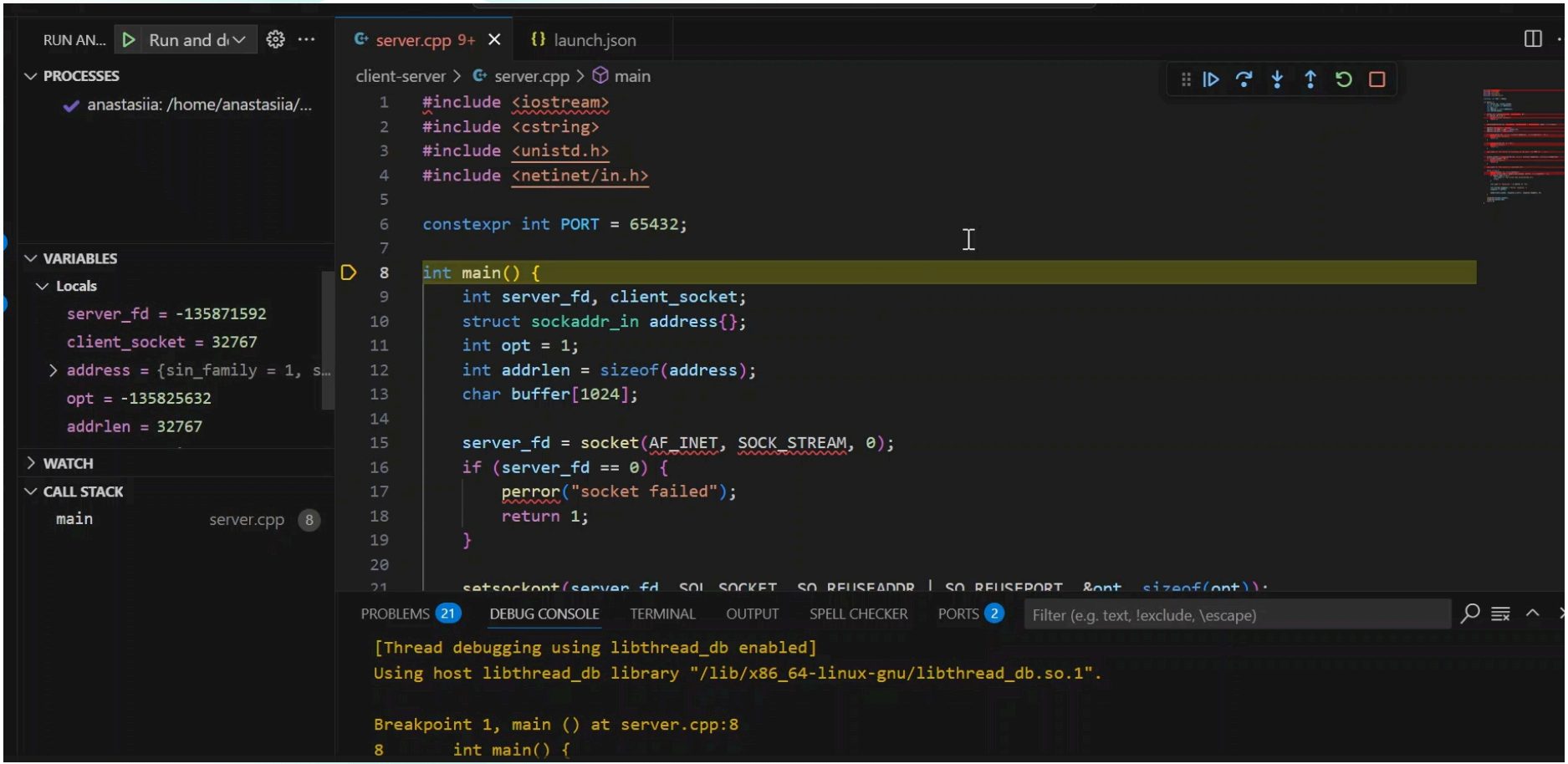
Поставим breakpoint



Подготовим конфигурацию и запустим отладку



Попали в 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

RUN AN... Run and d

PROCESSES

anastasiia: /home/anastasiia/...

VARIABLES

Locals

Registers

WATCH

CALL STACK Paused on breakpoint-hit

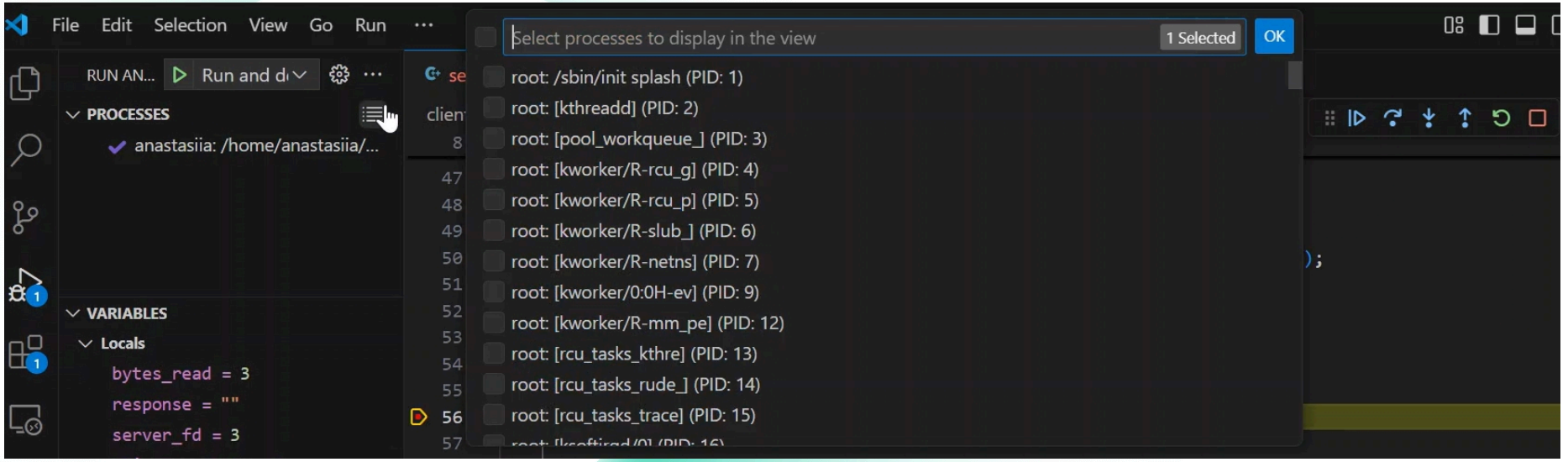
main server.cpp 56

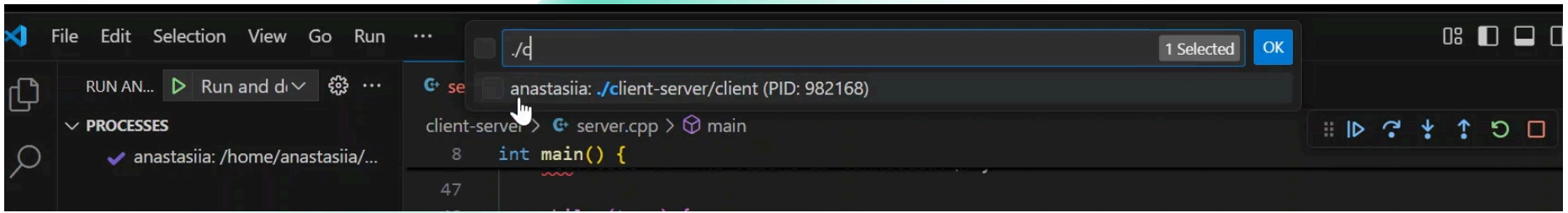
server.cpp 9+ launch.json

client-server > server.cpp > main

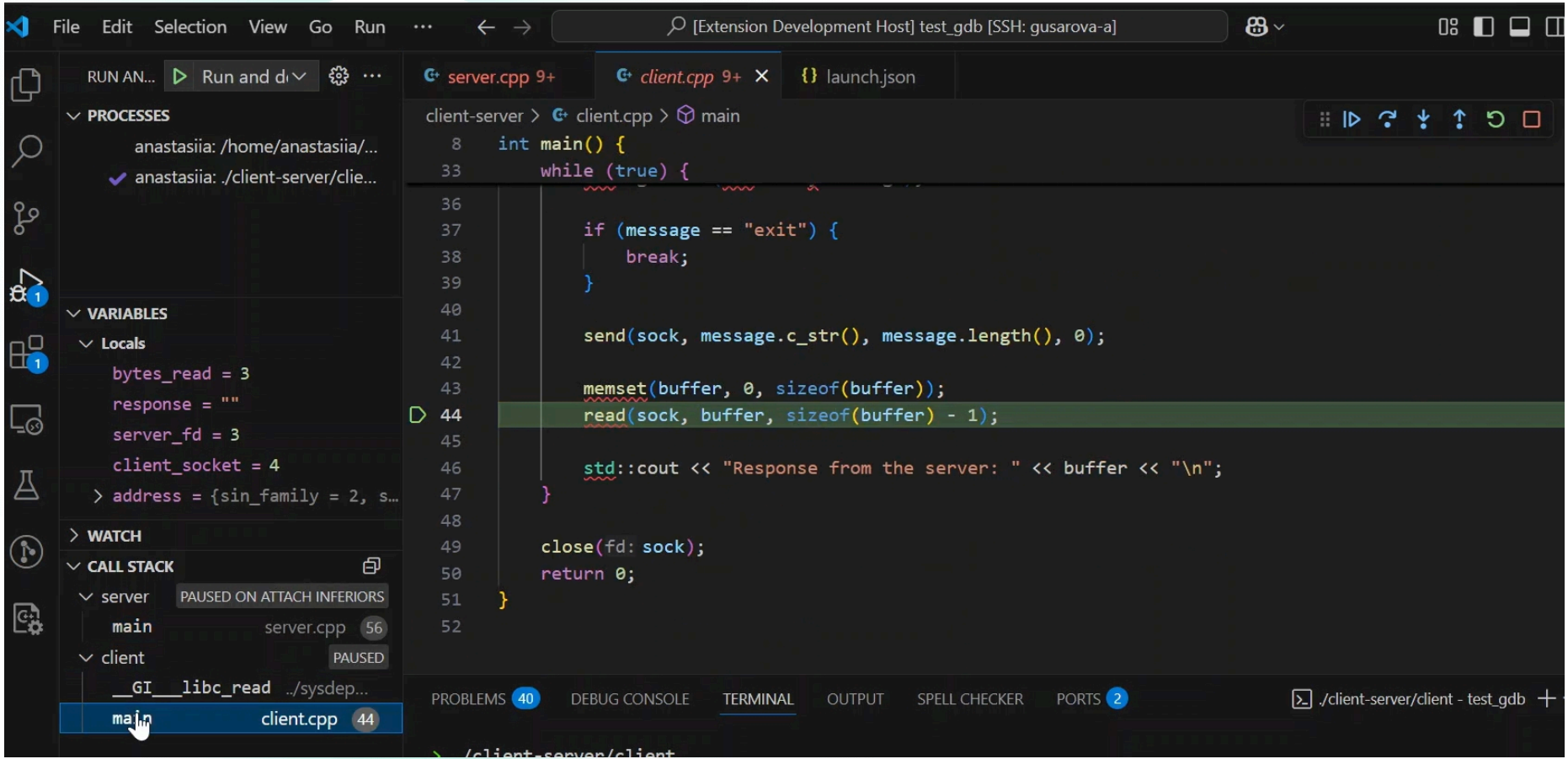
```
8   int main() {
47
48      while (true) {
49          memset(buffer, 0, sizeof(buffer));
50          ssize_t bytes_read = read(client_socket, buffer, sizeof(buffer) - 1);
51          if (bytes_read <= 0) {
52              std::cout << "The client has disconnected.\n";
53              break;
54          }
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);
```

Добавим в сессию отладки второй процесс

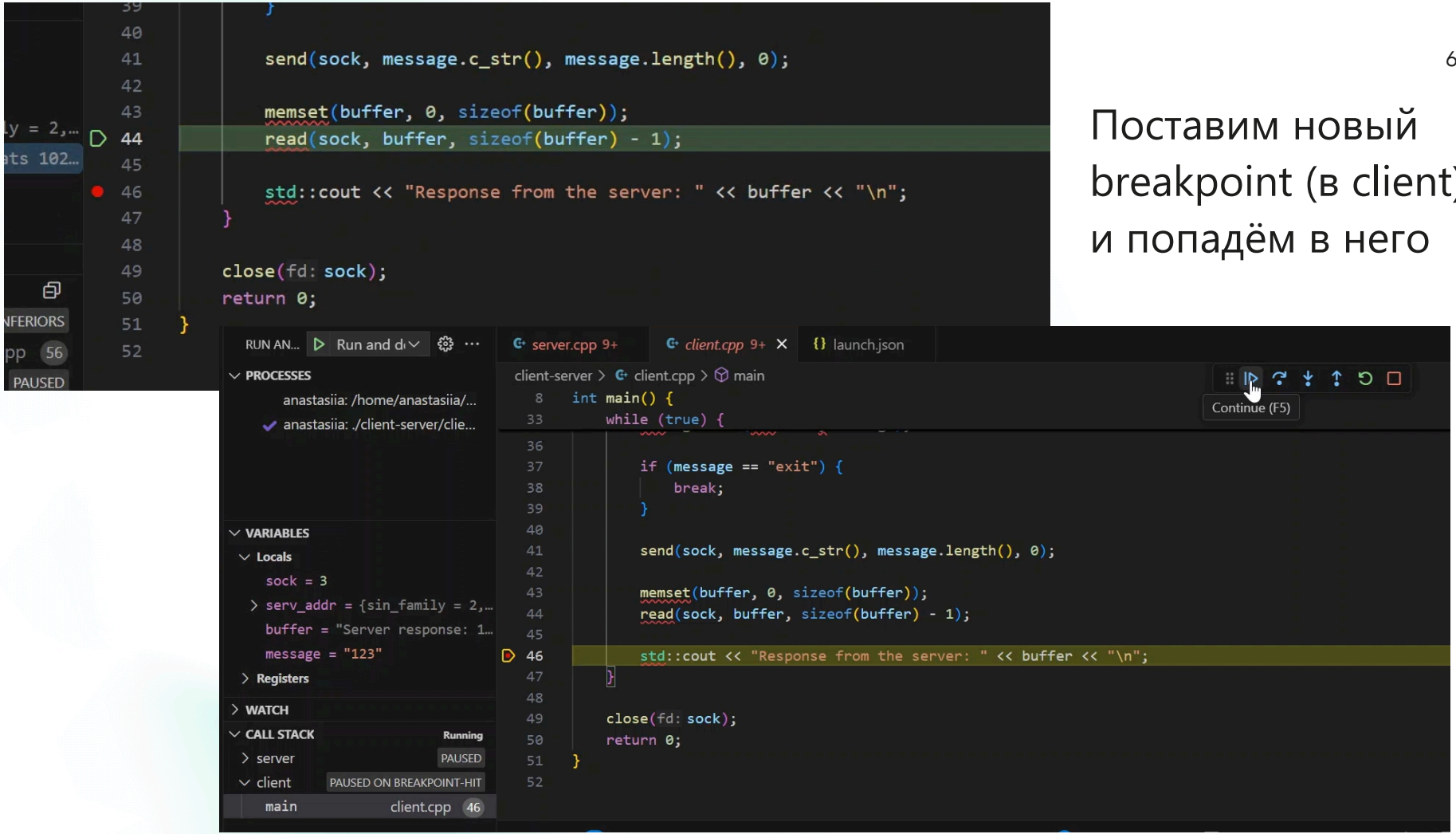




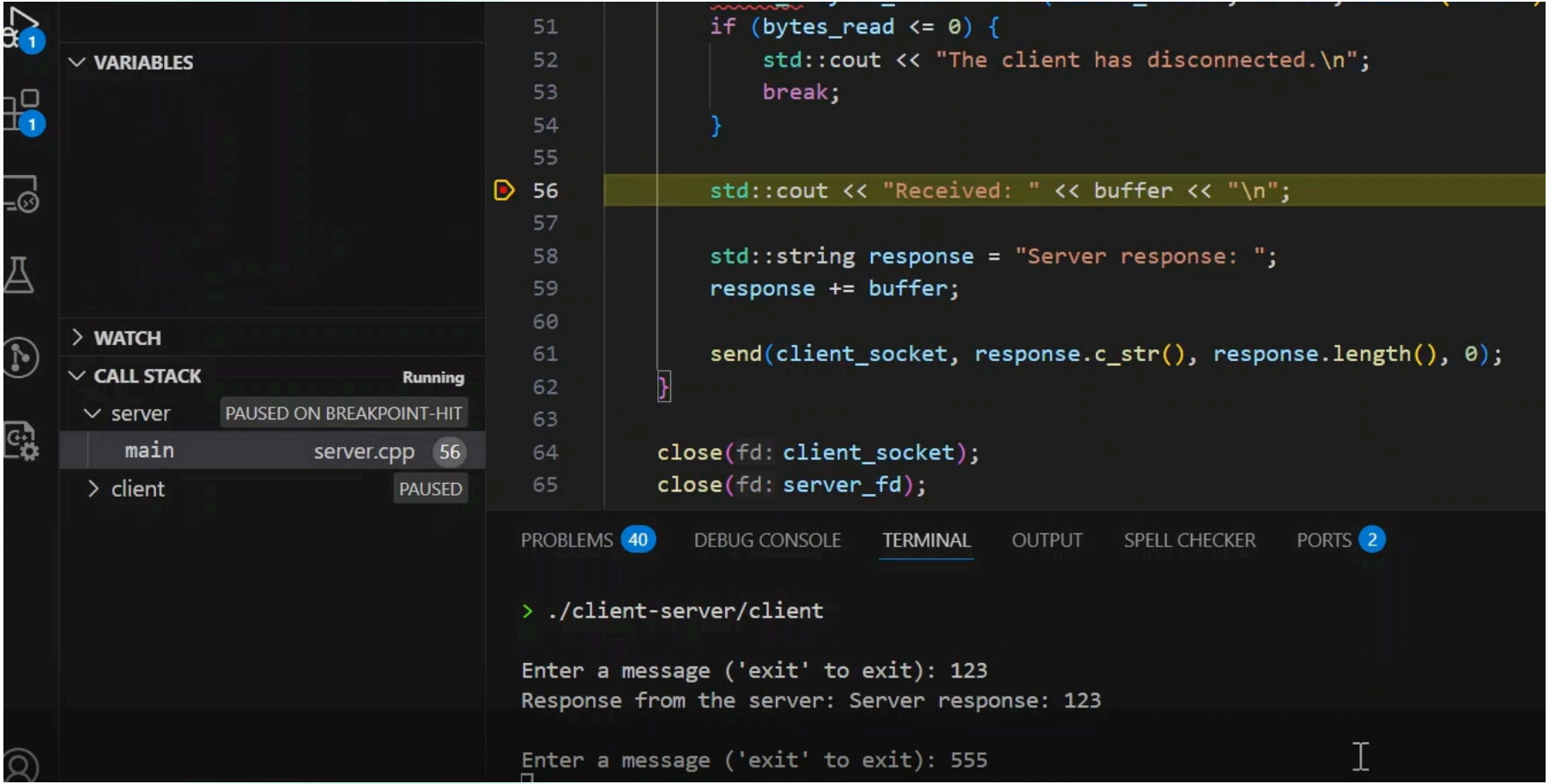
Получилось два процесса



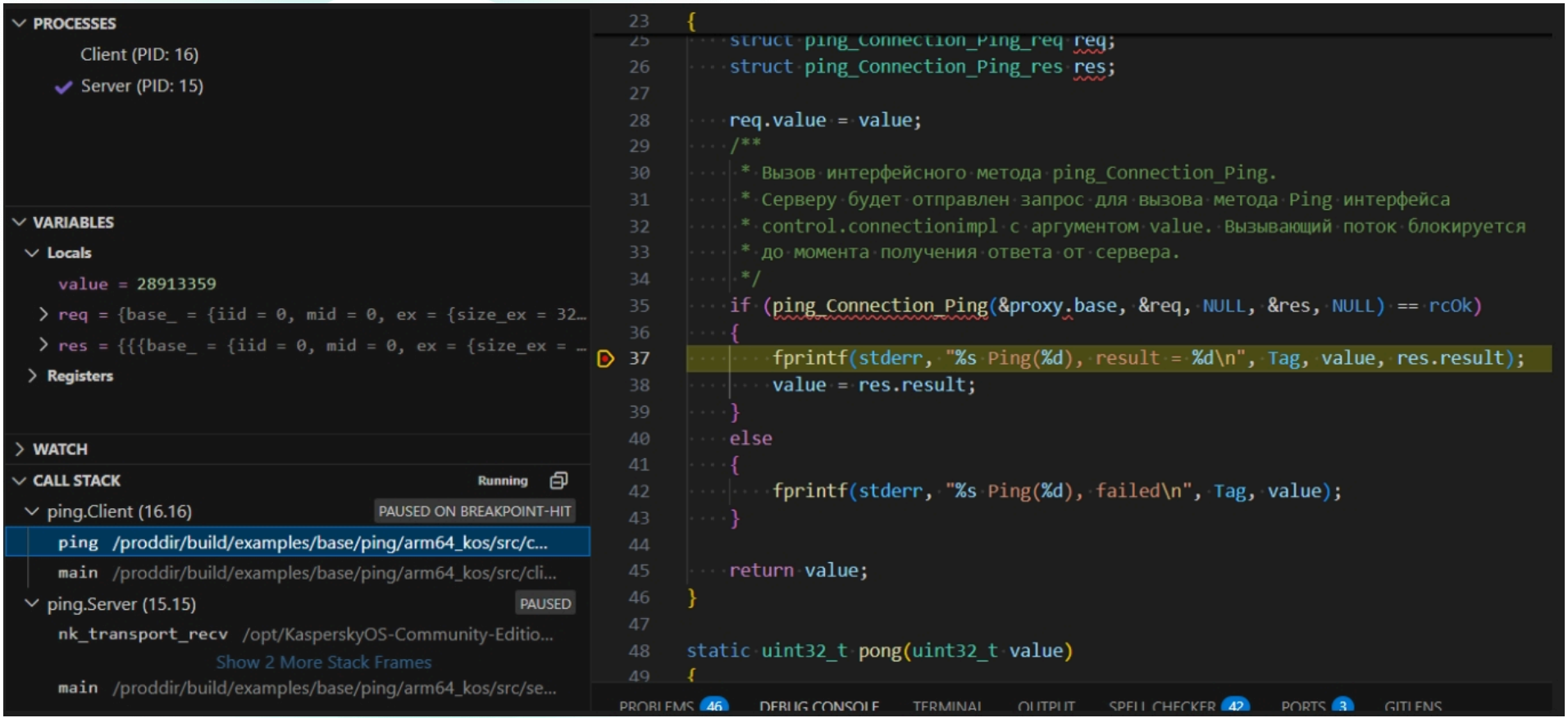
Поставим новый breakpoint (в client) и попадём в него



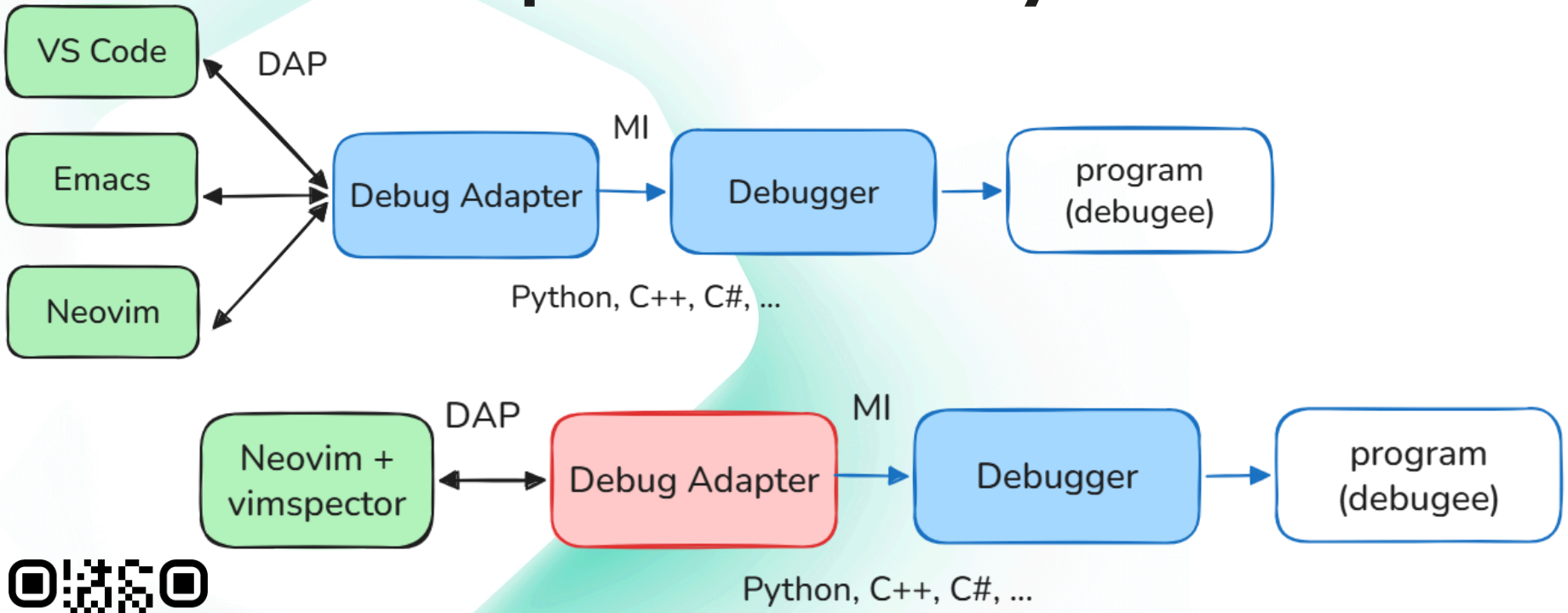
Попадём снова в первый breakpoint (server)



Отладка в KOS

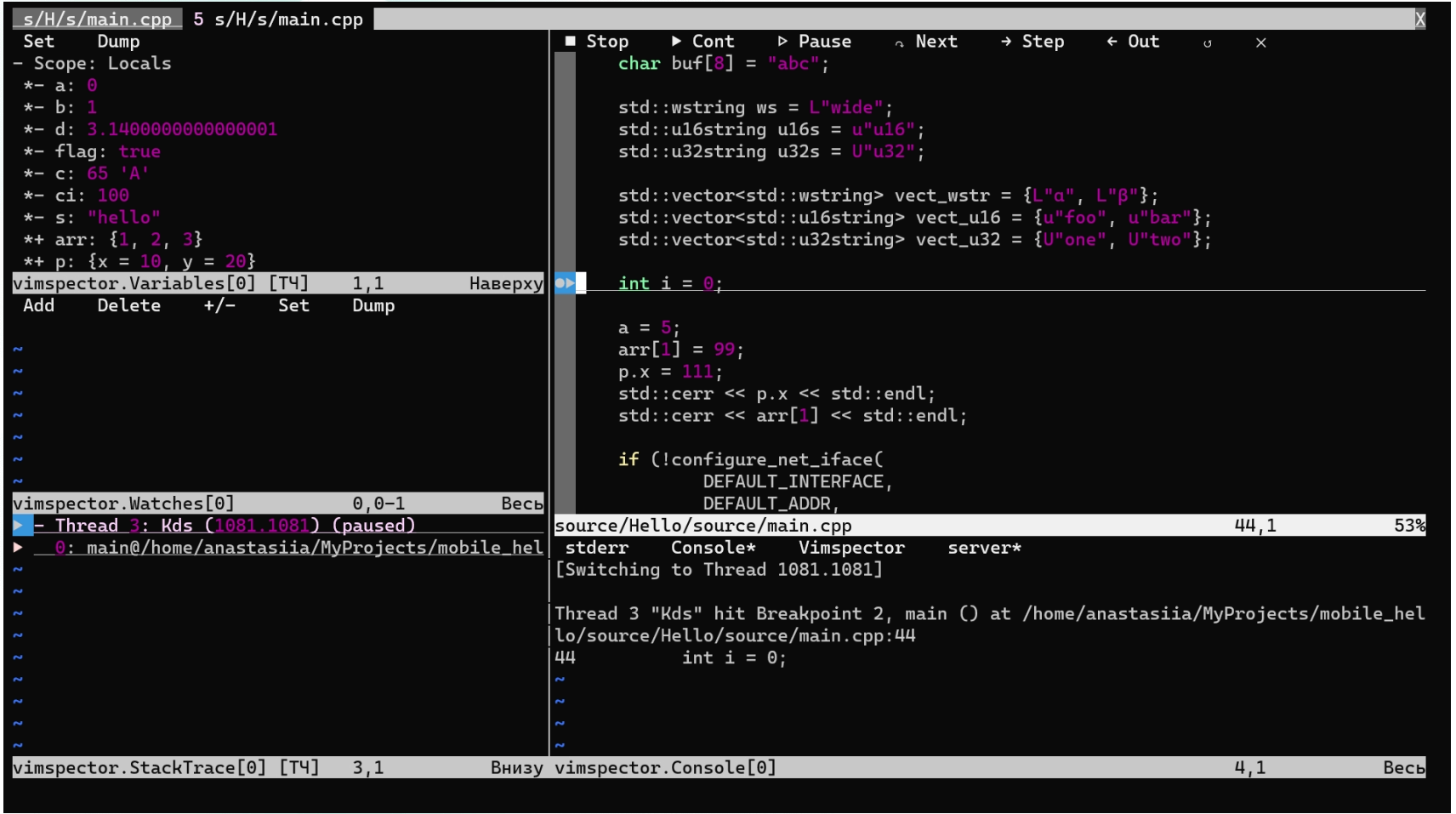


Приятный бонус



vimspector

А попробуем Neovim

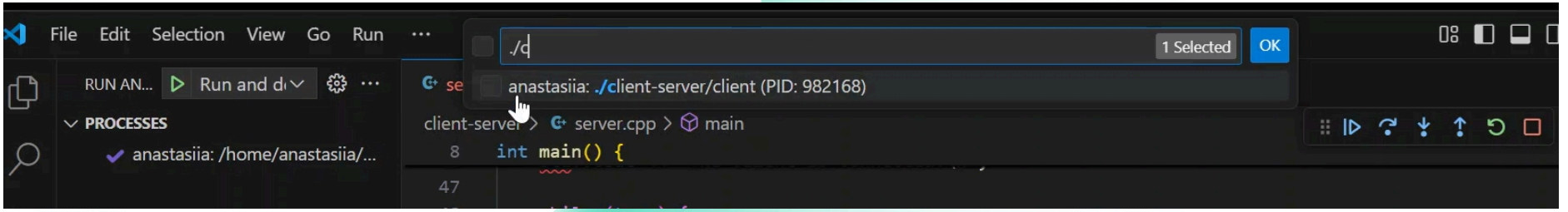


Выводы

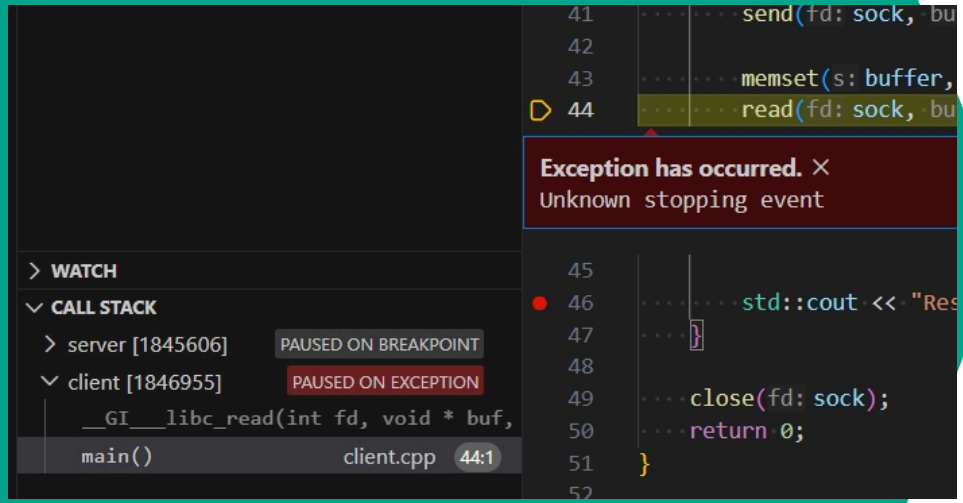
- Узнали о строении тулинга отладки
- Углубились в то, как вообще работает отладка
- Поговорили об удалённой отладке
- Познакомились с существующим отладочным тулингом
- Узнали про отладку в KasperskyOS

Что получили, реализовав свой Debug Adapter

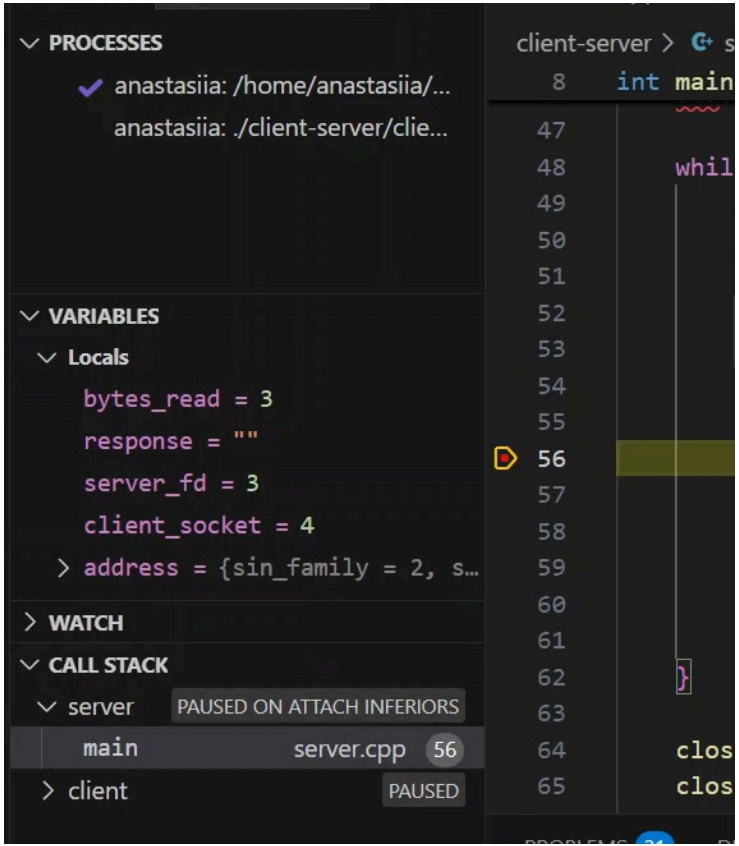
- Удобный интерфейс для отладки нескольких процессов
- Избавились от существующих проблем
- Открыли возможности для улучшений



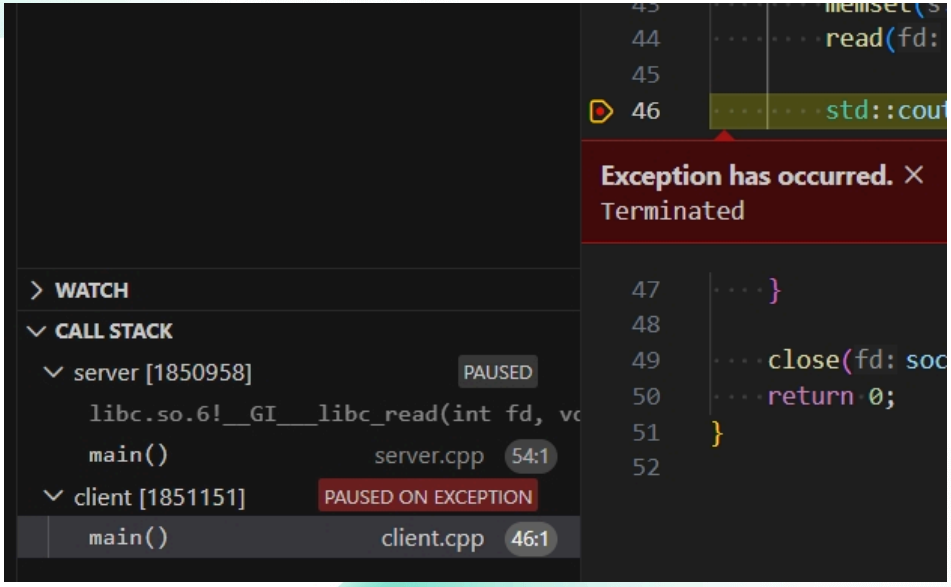
Было


41 send(fd: sock, bu
42
43 memset(s: buffer,
44 read(fd: sock, bu
Exception has occurred. ×
Unknown stopping event
45
46 std::cout << "Res
47 }
48
49 close(fd: sock);
50 return 0;
51 }
52
WATCH
CALL STACK
server [1845606] PAUSED ON BREAKPOINT
client [1846955] PAUSED ON EXCEPTION
__GI___libc_read(int fd, void * buf,
main() client.cpp 44:1

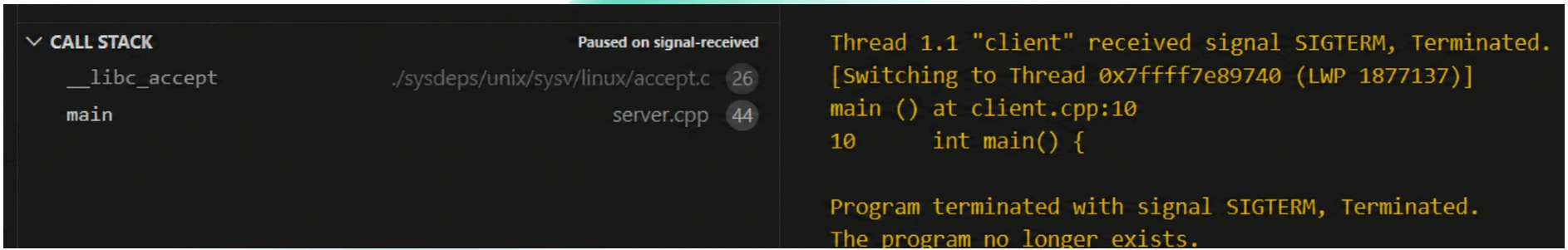
Стало


PROCESSES
anastasiia: /home/anastasiia/...
anastasiia: ./client-server/clie...
VARIABLES
Locals
bytes_read = 3
response = ""
server_fd = 3
client_socket = 4
address = {sin_family = 2, s...
WATCH
CALL STACK
server PAUSED ON ATTACH INFERIORS
main server.cpp 56
client PAUSED
client-server
8 int main
47
48 whil
49
50
51
52
53
54
55
56
57
58
59
60
61
62 }
63
64 clos
65 clos

Было


44 read(fd:
45
46 std::cout
Exception has occurred.
Terminated
WATCH
CALL STACK
server [1850958] PAUSED
libc.so.6!__GI___libc_read(int fd, vo
main() server.cpp 54:1
client [1851151] PAUSED ON EXCEPTION
main() client.cpp 46:1
47 }
48
49 close(fd: soc
50 return 0;
51 }
52

Стало


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

Попробовать отладку

kaspersky