Programming Curriculum Vitae

А. С. Миденков

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME
	2016-08-14	Добавлен Math calculator	

Оглавление

Глава 1

Миденков Алексей Сергеевич

Краткая информация

Дата и место рождения: 18 августа 1978 г., город Кемерово
Образование: ТУСУР (г. Томск), 2006 г., специальность АСОиУ
Телефон: +7 905 901 6065
E-mail: midenok@bk.ru
Skype: midenok

CV:

- HTML
- PDF

Резюме:

- Ведущий разработчик
- Senior Developer

Последнее место работы

Phorm (121 Media). Московский офис ООО ”ФОРМ” (в прошлом ООО ”Передовые интернет решения”) насчитывал порядка 100 человек и состоял из нескольких отделов: Server, Datacapture, UI, DB, QA, SA, HR. Компания специализируется на целевой рекламе. Её основное отличие от конкурентов в том, что она способна собирать целевой профиль и осуществлять поставку рекламы на стороне провайдера, что улучшало качество профиля и соотношение клик : покупка в сравнении с веб-сайтами. Разработкой этой технологии работы с трафиком на стороне провайдера занимался отдел Datacapture.

Обо мне

В компании ”ФОРМ” я работал в отделе Datacapture с 2007-го по 2015 гг., затем в начале 2015-го был переведён в отдел Server, где я проработал несколько месяцев до увольнения. В отделе Datacapture я написал модули обработки запросов для Apache и Nginx, модуль восстановления пользователей по данным браузера, модуль защиты от фишинговых сайтов, новую версию стартующих скриптов, а также другие обновления в рамках соответствующих тикетов. Разработка велась в основном на C++, Perl, XSLT, Bash. Для внутрикластерного взаимодействия использовалась CORBA.

Также, я оптимизировал производительность процессов, исправлял утечки памяти и некорректные операции исполнения, работал с продакшн кластерами. Я хорошо ориентируюсь в большом объёме кода. Разбираюсь в устройстве серверных приложений, владею средствами POSIX, в том числе POSIX Threads. Работал с библиотеками libev и zlib. Писал скрипты для Make, CMake и Automake.

Не лишне будет упомянуть, что мой предпочитаемый дистрибутив – Debian, которым я пользуюсь с 1999-го года. И в котором я работал как с debhelper, так и с CDBS, а также репозиториями reprepro.

Посмею также заметить, что работу я делаю качественно. Тщательно обдумываю свои решения, смотрю на задачу с различных сторон. Если попытаться коротко описать моё жизненное кредо, то я бы представил его двумя афоризмами: «стремиться к идеалу изо всех сил, зная, что он недостижим» и «малыми шагами совершаются большие дела».

Немного деталей

Расскажу поподробнее про модули, которые я написал. Модуль обработки запросов взаимодействует с браузером пользователя. Когда пользователь впервые обращается в домен компании, запрос приходит в этот модуль. Далее производится ряд операций – проверка на возможность установить куки, попытка восстановить куки из локальных хранилищ и наконец присвоение уникального идентификатора. Когда пользователь обращается за рекламой, то модуль может выступать в качестве прокси, если того требует инфраструктура и передать запрос в рекламный сервер.

Модуль восстановления пользователей по данным браузера. Если пользователь удалил свой идентификатор и восстановить его из локального хранилища не получилось, то существовала возможность восстановить его по данным браузера, т.к. все эти данные сохранялись в базе наряду с уникальным идентификатором и IP-адресом. При этом во время поиска по базе использовался алгоритм нечёткого сравнения (расстояние Левенштейна) на случай, если данные браузера немного изменились – например, обновилась версия или были установлены новые шрифты.

И наконец модуль защиты от фишинговых сайтов. В этом модуле производилась проверка оригинального URL пользователя в списке фишинговых URL. Если обнаруживалось соответствие, то пользователю вместо запрашиваемой страницы открывалась страница с предупреждением. Список хранился в shared memory и был доступен всем процессам и потокам демона. При этом он периодически инкрементально обновлялся по HTTP протоколу.

Глава 2

Примеры работ

Evoxy: libev-based HTTP proxy

```
Год: 2016
Инструментарий: C++11, POSIX, libev
```

Исходный код:

- Репозиторий
- Архивы

Текущий статус проекта: в HTTP-proxy реализован минимальный функционал, необходимый для успешной работы: Keep-Alive соединения, обработка Transfer-Encoding: chunked. В будущем предусматривается такая функциональность, как: пайплайнинг запросов (Issue #3), асинхронное разрешение доменных имён (Issue #4), HTTPS.

Программа написана на C++ с использованием элементов 11-го стандарта. Архитектура демона основана на неблокирующихся сокетах и состоит из пула потоков, в каждом из которых запущен цикл событий, обрабатывающий все принятые в данном потоке соединения. Дизайн Evoxy предполагает, что по возможности все ресурсы, необходимые потоку должны быть локальны в этом потоке, а значит свободны от многопоточной конкурентности. На данный момент программа не использует ни одного мьютекса (или какого-либо другого средства синхронизации потоков).

Демон работает на преаллоцированной памяти. На стадии инициализации каждому потоку из кучи выделяются все необходимые пулы памяти. Пулы памяти демона работают с блоками фиксированного размера, т.е. заточены на аллокацию объектов одного определённого типа. Размер пула соответствует установленным в настройках лимитам программы. На данный момент используется всего 3 различных пула: 1 для создания соединения и 2 для кэша доменных имён.

Главный недостаток текущей версии в том, что обработчики соединений Frontend и Backend жёстко связаны друг с другом в соотношении 1:1. Такая схема неэффективна для Keep-Alive соединений, поскольку запрос к каждому новому хосту на стороне Frontend-а обязывает Backend разрывать текущее соединение и устанавливать новое. Будущая версия программы должна иметь пул соединений Backend-ов, которым может пользоваться Frontend (см. Issue #1).

См. также: описание server-demo.

Math calculator

`Год: 2016`
`Инструментарий: C++11`

Исходный код:

- Репозиторий
- Архивы

Калькулятор вычисляет математические выражения с учётом приоритета операций и скобок. В выражениях могут присутствовать унарные математические функции (см. math.h). Вычисление производится двумя стадиями: построение бинарного синтаксического дерева (`Calc::parse()`) и исполнение синтаксического дерева (`Calc::calculate()`). Программа не использует рекурсивные вызовы.

Правила синтаксического дерева:

1. в корне всегда число, исполнение начинается с корня;
2. каждый левый потомок: операция (оператор или функция);
3. каждый правый потомок: число;
4. оператор исполняется над двумя операндами: предок и правый потомок;
5. функция исполняется над одним операндом: предок;
6. если у второго операнда (правый потомок) есть левый потомок (операция), то новая операция имеет приоритет над текущей.

Таким образом, вычисление начинается с корня и продолжается по левой ветви до первого ветвления (левый потомок у операнда). Затем, состояние сохраняется в стек и вычисление начинается в новой ветви (от операнда). Когда ветвь заканчивается, состояние восстанавливается из стека и продолжается вычисление старой ветви. Обход продолжается до тех пор, пока не закончится ветвь при пустом стеке.

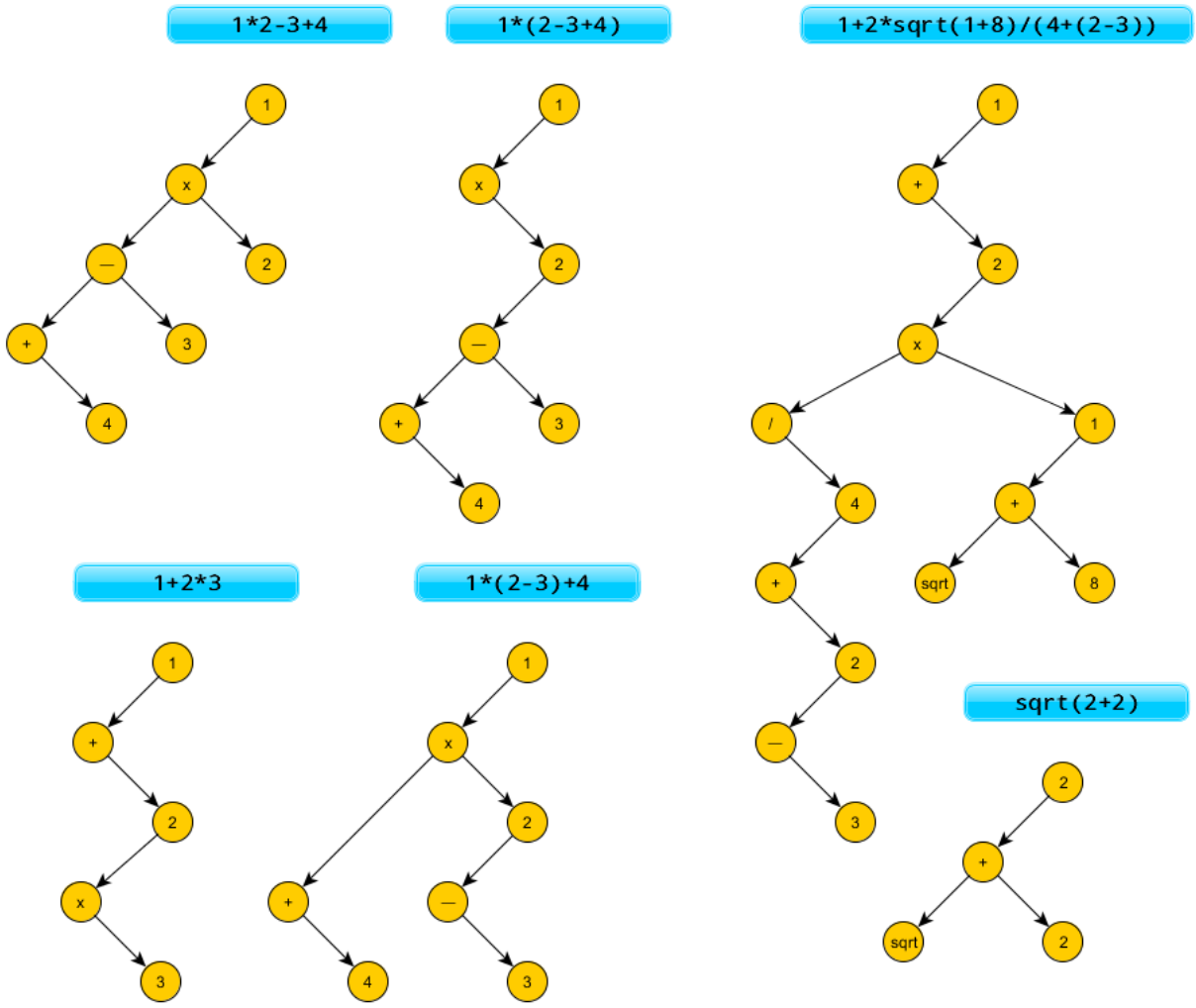

Рис. 2.1: Примеры синтаксических деревьев Math calculator

C++ MySQL wrapper

Год: 2013
Инструментарий: C++, MySQL Connector/C

Исходный код:

- Репозиторий
- Архивы

C++ обёртка вокруг MySQL Connector/C. Основное её преимущество перед стандартным C++ коннектором -- многопоточный пул коннектов. Обёртка имеет дружественный интерфейс: не нужно вручную создавать и разрушать объекты; текст SQL кода инстанцируется выводом в `stringstream`.

Example 2.1 Схема использования MySQL wrapper

```
Conf c;
c.host = ”localhost”;
c.user = ”user”;
c.passwd = ”password”;
c.db = ”database”;

Connection::Pool pool(pool_size);

try {
    pool.connect(c);
    Query q(pool);
    q << ”-- any SQL code (multiple statements allowed)\n”;

    q.execute(); // or q.execute_only() if no data returned

    Row row;
    while (row = q.fetch_row()) {
        // Get data from row.
        //   integers are accessed with (POS_INDEX);
        //   strings are accessed with [POS_INDEX],
        // like here:
        int integer_value = row(0);
        std::string string_value = row[1];
   }
} catch (Database::Error &ex) {
    std::cerr << ex.what();
} catch (Database::Exception &ex) {
    std::cerr << ex.what();
}
```

См. также рабочий пример использования: example.cpp.

Набросок обработки Bid-запросов в AdServer

Год: 2015
Инструментарий: UML class diagram

Диаграмма основана на UML диаграмме классов и представляет собой смесь поведенческой и структурной диаграммы. Диаграмма позволила быстро понять соотношение и поведение объектов в контексте конкретной задачи. Ни одна из существующих поведенческих видов диаграмм UML для этого не подошла.

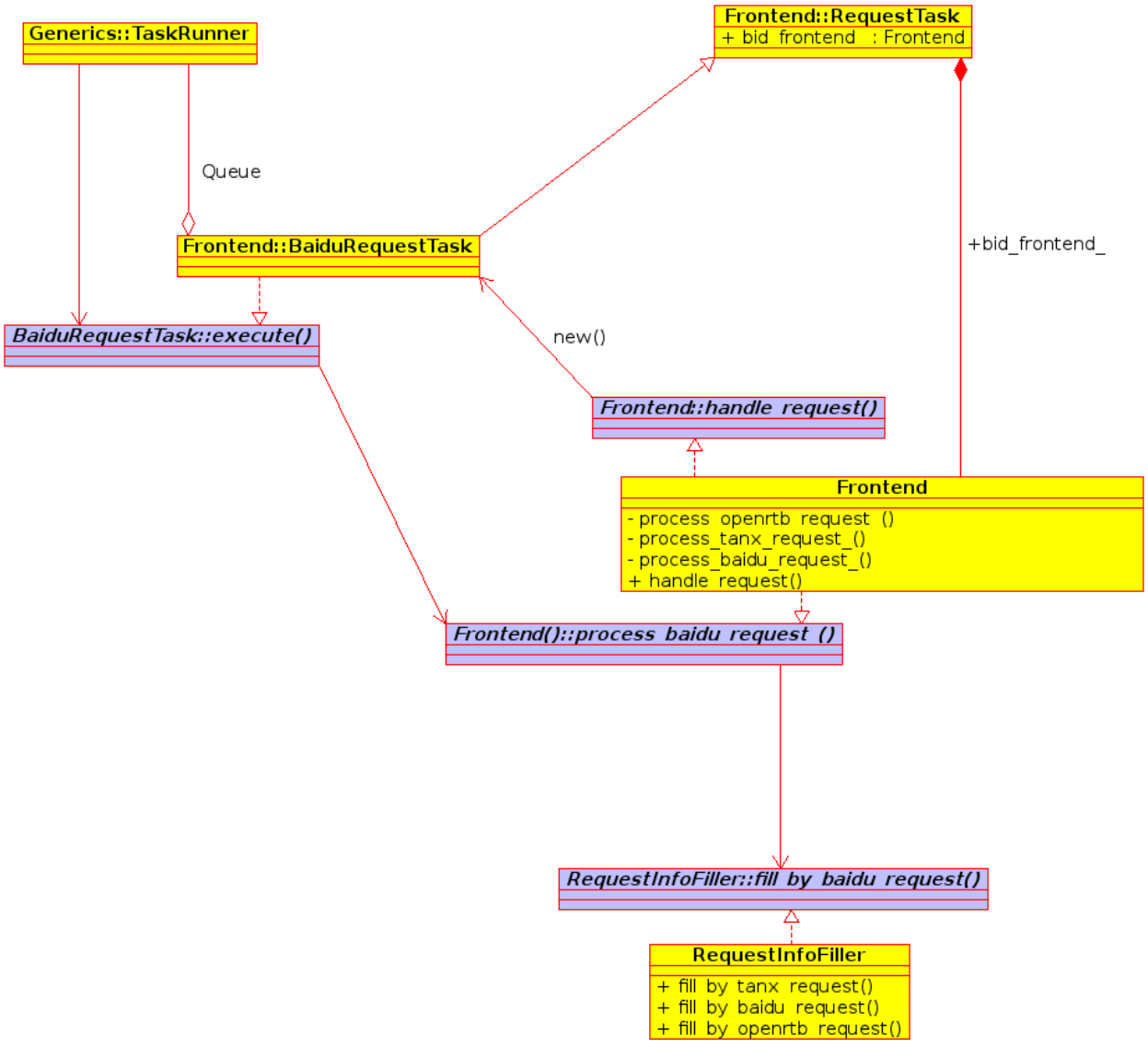

Рис. 2.2: Схема среза по обработке Bid-запросов в AdServer

Поясню устройство диаграммы:

- жёлтым цветом обозначены классы, фиолетовым -- методы;
- ”ассоциацией” обозначены вызовы методов;
- ”реализацией” обозначена принадлежность методов классам;
- ”композицией” показано эксклюзивное владение объектом;
- ”агрегацией” показано временное владение объектом либо хранение указателя на объект.

Разберём содержимое диаграммы. Класс `Frontend` в вызове `handle_request()` создаёт `BaiduRequestTask`, который складывается в очередь `TaskRunner`. `TaskRunner` вызывает `execute()` у `BaiduRequestTask`, который в свою очередь вызывает `process_baidu_request()` у `Frontend`. Каким образом -- показано в верхнем правом углу: `BaiduRequestTask` унаследован от `RequestTask`, а тот в свою очередь содержит указатель на объект `Frontend`: `bid_frontend_`. Итак, `process_baidu_request()` вызывает `fill_by_baidu_request()` у `RequestInfoFiller`. На этом контекст схемы заканчивается.

Подчеркну, что диаграмма описывает лишь очень небольшой срез, необходимый для решения конкретной задачи. В огромном объёме кода подобные компактные наброски на мой взгляд жизненно важны, т.к. они позволяют быстро вспоминать контекст задачи. Возможно, в диаграмме была допущена неточность: `RequestTask` вряд ли владеет объектом `Frontend`, отношение композиции должно быть заменено отношением агрегации.

Bush: build shell

```
Год: 2007-2015
Инструментарий: Bash
```

Исходный код:

- bush

Скрипт для автоматизации workflow по разработке задач. Запускает сессию Bash с окружением, настроенным под контекст конкретного тикета задачи. В окружение добавляются CLI-команды, которые значительно сокращают рутинные действия по подготовке задачи к работе. Например, при помощи только одной команды ’new’ происходят следующие действия:

- в SVN создаётся бранч задачи, а на диске его рабочая копия;
- при необходимости чекаутятся и компилируются все необходимые библиотеки;
- компилируется сама задача (т.е. основное приложение из рабочей копии);
- инстанцируются файлы конфигурации и запускается приложение задачи.

Каждая задача имеет свою директорию с номером и кратким названием, внутри которой хранятся все файлы задачи: рабочие копии, деревья компиляции, деревья инсталяции, конфигурация, скрипты, заметки и пр. В данный момент скрипт заточен под задачи конкретной компании, но может быть переделан под любой workflow.

System troubleshooting report

Год: 2010
Инструментарий: Perl, Linux system utilities

Исходный код:

- trouble-report

Скрипт собирает системные данные хоста для случаев критических неисправностей: информацию по системным ресурсам, системные логи и логи программы, информацию по коркам, конфигурацию, текущее поведение системы и пр. Скрипт настраивается модификацией `@REGISTRY`:

Example 2.2 Настройка работы trouble-report

```
my @REGISTRY = (
    {
        title => 'Uptime',
        command => 'uptime',
        filename => 'common.txt'
    },
    {
        title => 'Processes by user',
        command => 'ps uax',
        filename => 'ps.txt'
    },
    {
        title => 'Process tree',
        command => 'pstree -p',
        filename => 'pstree.txt'
    },
    {
        title => 'Last log files',
        command => "ls -1rt $0{program_root}/var/log/*|tail|xargs tail",
        filename => 'logs.txt'
    },
    # ... and so on ...
);
```

Скрипт имеет процедуру бинарного поиска для быстрого поиска записи лога по timestamp: `binary_find`.

Browser list parser

Год: 2013
Инструментарий: Perl

Исходный код:

- browserlist-parser.pl

Скрипт подготавливает данные, необходимые для парсинга строчек User-Agent. Он преобразует списки сайта useragentstring.com в упорядоченные списки токенов, по которым происходит идентификация браузера и операционной системы. Основная сложность заключается в том, что строка User-Agent может содержать несколько токенов от разных браузеров. Так, например строка для AOL 9.7:

```
Mozilla/4.0 (compatible; MSIE 8.0; AOL 9.5; AOLBuild 4337.43; Windows NT 6.0;
Trident/4.0; SLCC1; .NET CLR 2.0.50727; Media Center PC 5.0; .NET CLR 3.5.21022;
.NET CLR 3.5.30729; .NET CLR 3.0.30618)
```

содержит токен MSIE, поэтому проверка на AOL должна проходить раньше, чем на MSIE. Скрипт анализирует такие появления токенов от других браузеров и определяет порядок в котором токены должны проверяться, чтобы избежать неверного определения. См. также: описание использования скрипта.

Другое

Symlinked install

Год: 2007-2014
Инструментарий: CMake, Bash, Perl

Исходный код:

- ln-install

Утилита позволяет избежать шага `make install` в цикле ”редактирование-компиляция-запуск”, что значительно облегчает работу с тяжеловесным кодом. Для этого `make install` вместо копирования файлов создаст символические ссылки на дерево компиляции. Утилита работает с деревьями Autoconf, CMake и MakeMaker.

Конвертер ошибок GCC в MSVC

Год: 2016
Инструментарий: Perl

Исходный код:

- gcc2msvc.pl

Обёртка для GCC позволяет использовать этот компилятор в Visual Studio с корректным выводом ошибок в окно Error List. Данный пример демонстрирует работу `Select` в связке с `Open3`.

Реверсинг кода игры Atomic Bomberman

Год: 2013
Инструментарий: Ida Pro 6.1; Syser 1.99

В ходе анализа кода DirectX игры Atomic Bomberman были найдены функции, отвечающие за работу основного меню ($main_menu_loop), запуска новой игры ($new_game_options, $new_game_players, $new_game), работу настроек игры ($options_menu, $load_options), а также основные игровые переменные, такие как $newgame_goal_amount, $goal_type, $newgame_theme. Все символы с '$' были разработаны мной в процессе реверсинга, полный их список (73) можно посмотреть в bm95_discovered.map. Заметки по анализу находятся в bomberman.txt.

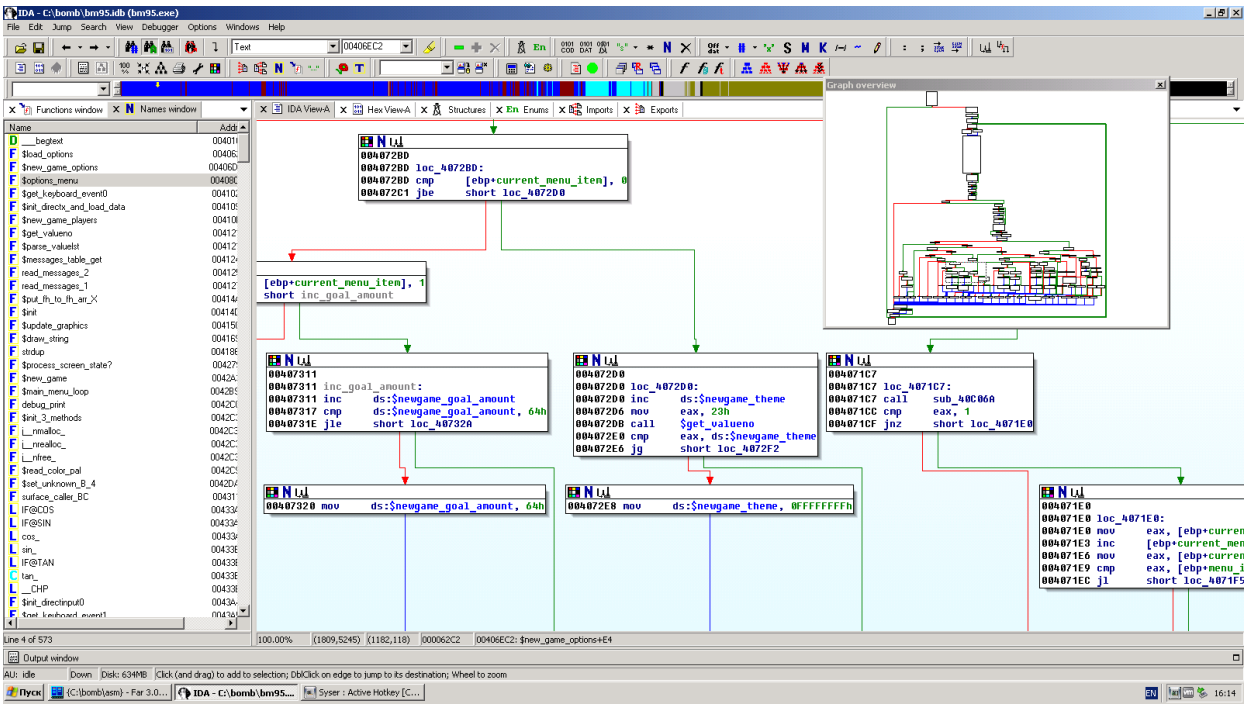

Рис. 2.3: Анализ Atomic Bomberman (скриншот IDA Pro)