Инструменты разработчика (часть 2)

2019

Покрытие кода тестами

Оптимизация с помощью профилирования

Способы измерения производительности

Статический анализ кода

Зависимости

Покрытие кода тестами

```
$ meson configure -Db_coverage=true # включить покрытие
$ ninja test                        # запустить тесты
$ ninja coverage-text               # вывод в текстовом виде
$ cat meson-logs/coverage.txt
File       Lines    Exec  Cover   Missing
----------------------------------------
../main.cc     7       7   100%
----------------------------------------
TOTAL          7       7   100%
----------------------------------------
```

LCOV - code coverage report

		Hit	Total	Coverage
Current view:	top level - tmp/cov - main.cc (source / functions)			
Test:	Code coverage	**Lines:**	5	6	83.3 %
Date:	2019-11-07 20:26:49	**Functions:**	1	1	100.0 %
Legend:	Lines: hit not hit | Branches: + taken - not taken # not executed	**Branches:**	1	2	50.0 %

	Branch data	Line data	Source code
1		4	int main() {
2		4	int a = 10, b = 10, d;
3	[- +]	4	if (a > b) {
4		0	d = 30;
5			} else {
6		4	d = 10;
7			}
8		4	return 0;
9			}

Generated by: LCOV version 1.14

```
$ ninja coverage-html                    # в виде веб-страницы
```

```
...
if (a > b && c != 25) {
    ++d;
}
...
// a >  b && c != 25
// a <= b && c != 25
// a >  b && c == 25
// a <= b && c == 25
```

- По строчкам.
- По веткам.
- По входным данным.

```
// кодирование в формате BASE64
void base64_encode(const char* first, size_t n, char* result) {
    size_t rem = n%3;
    size_t m = (rem == 0) ? n : (n-rem);
    for (size_t i=0; i<m; i+=3) {
        *result++ = ...; *result++ = ...;
        *result++ = ...; *result++ = ...;
        ++first;
    }
    if (rem == 1) {
        *result++ = ...; *result++ = ...;
        *result++ = '='; *result++ = '=';
    } else if (rem == 2) {
        *result++ = ...; *result++ = ...;
        *result++ = ...; *result++ = '=';
    }
}
```

How SQLite Is Tested

1. Introduction

The reliability and robustness of SQLite is achieved in part by thorough and careful testing.

As of version 3.29.0 (2019-07-10), the SQLite library consists of approximately 138.9 KSLOC of C code. (KSLOC means thousands of "Source Lines Of Code" or, in other words, lines of code excluding blank lines and comments.) By comparison, the project has 662 times as much test code and test scripts - 91946.2 KSLOC.

1.1. Executive Summary

Three independently developed test harnesses

100% branch test coverage in an as-deployed configuration

- Out-of-memory tests

Оптимизация с помощью профилирования

```
# сборка с профилингом
$ g++ -fprofile-generate main.cc -o main.o
$ g++ -fprofile-generate main.o -o myprog
# запуск тестов
...
# использование информации после профилинга
$ g++ -fprofile-use main.cc -o main.o
$ g++ -fprofile-use main.o -o myprog
```

Тоже самое в Meson build:

```
$ meson configure -Db_pgo=generate
$ meson configure -Db_pgo=use
```

Порядок веток кода

Исходный код на C++:

```
int a = 10, b = 10, d;
if (a > b) {
    d = 30;
} else {
    d = 10;
}
```

Сгенерированный код на ассемблере:

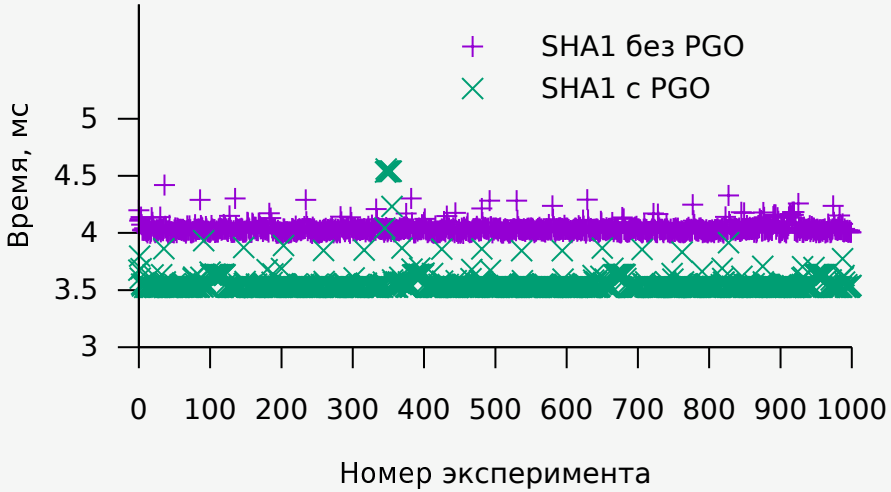
```
    movl    $10, -4(%rbp)
    movl    $10, -8(%rbp)
    movl    -4(%rbp), %eax
    cmpl    -8(%rbp), %eax
    jle .L2
    movl    $30, -12(%rbp)
    jmp .L3
.L2:
    movl    $10, -12(%rbp)
.L3:
    movl    $0, %eax
```

Порядок веток кода

```
void add_new_user(User user) {
    if (!(user.id() >= min_user_id)) {
        throw std::invalid_argument("bad uid"); // редкая ветка
    }
    if (!(user.group_id() >= min_group_id)) {
        throw std::invalid_argument("bad gid"); // редкая ветка
    }
    if (!user.has_valid_name()) {
        throw std::invalid_argument("bad name"); // редкая ветка
    }
    ...
}
```

Порядок веток кода

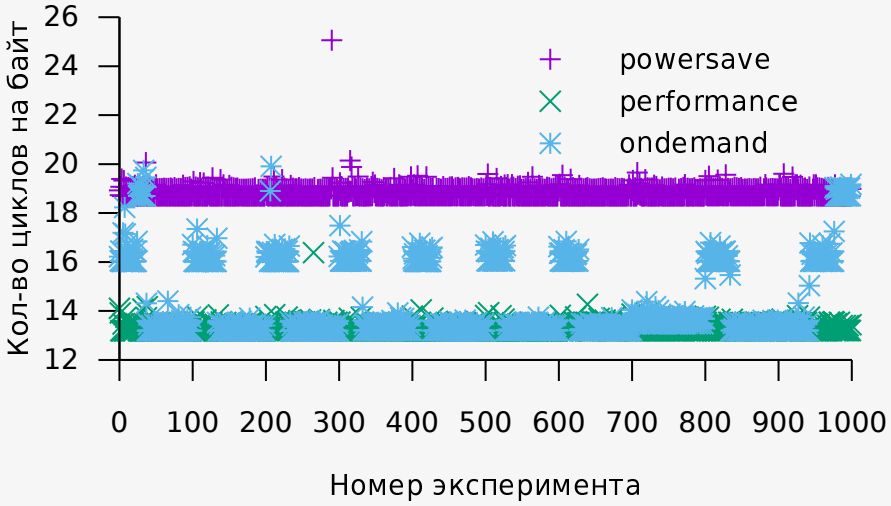
```
#define LIKELY(x) __builtin_expect((x),1)
#define UNLIKELY(x) __builtin_expect((x),0)
void add_new_user(User user) {
    if (UNLIKELY(!(user.id() >= min_user_id))) {
        ...
    }
    if (UNLIKELY(!(user.group_id() >= min_group_id))) {
        ...
    }
    if (UNLIKELY(!user.has_valid_name())) {
        ...
    }
    ...
}
```


SHA1 без PGO
SHA1 с PGO
Время, мс
5
4.5
4
3.5
3
0
100
200
300
400
500
600
700
800
900
1000
Номер эксперимента

Циклы процессора

Для быстрой функции:

```
inline uint64_t cycles() {        // TSC — Time Stamp Counter
    uint32_t high, low;
    asm volatile("lfence\n"       // барьер
                 "rdtsc"          // количество циклов
                 :
                 "=d"(high),      // считать из регистра edx
                 "=a"(low)        // считать из регистра eax
    );
    return ((uint64_t)high << 32) | low;
}
```




```
$ sudo cpupower frequency-set -g ondemand # режим работы процессора
```

Календарное время

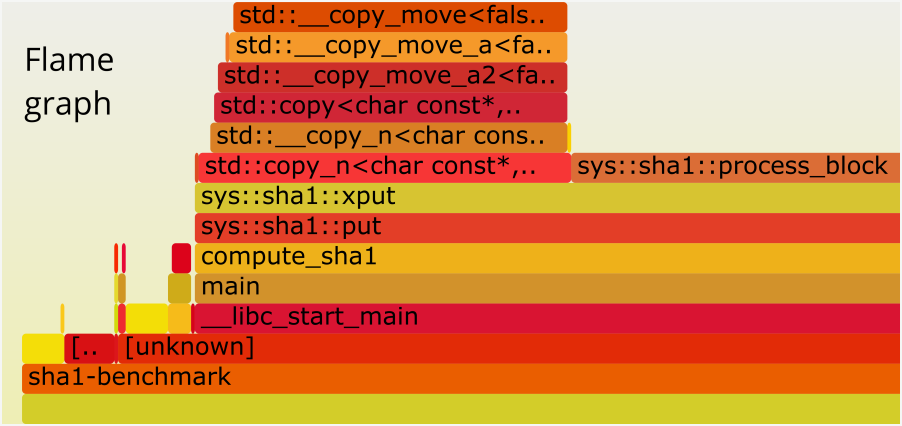
Измерение календарного (реального) времени:

```
using namespace std::chrono;
auto t0 = high_resolution_clock::now();
...
auto t1 = high_resolution_clock::now();
std::cout << duration_cast<milliseconds>(t1-t0).count() << "мс\n";
```

Распределение времени

```
Samples: 620  of event 'cycles:upp', Event count (approx.): 795163312
  Children      Self  Command          Shared Object          Symbol
+   96,57%     0,00%  sha1-benchmark   [unknown]              [.] 0x41fd89415541f689
+   96,57%     0,00%  sha1-benchmark   libc-2.27.so           [.] __libc_start_main
+   96,57%     0,00%  sha1-benchmark   sha1-benchmark         [.] main
+   96,57%     0,00%  sha1-benchmark   sha1-benchmark         [.] compute_sha1
+   96,57%     0,00%  sha1-benchmark   sha1-benchmark         [.] sys::sha1::put
+   96,40%     0,35%  sha1-benchmark   libunistdx.so.0.4.13   [.] sys::sha1::xput
+   74,85%    73,37%  sha1-benchmark   libunistdx.so.0.4.13   [.] sys::sha1::process_block
+   20,86%     0,47%  sha1-benchmark   libunistdx.so.0.4.13   [.] std::copy_n<char const*,
+   20,04%     0,17%  sha1-benchmark   libunistdx.so.0.4.13   [.] std::copy<char const*, ur
+   19,86%     0,00%  sha1-benchmark   libunistdx.so.0.4.13   [.] std::__copy_n<char const*
+   19,69%     0,17%  sha1-benchmark   libunistdx.so.0.4.13   [.] std::__copy_move_a2<false
+   19,51%    19,51%  sha1-benchmark   libunistdx.so.0.4.13   [.] std::__copy_move<false, 1
+   19,51%     0,00%  sha1-benchmark   libunistdx.so.0.4.13   [.] std::__copy_move_a<false,
+    1,13%     0,52%  sha1-benchmark   libunistdx.so.0.4.13   [.] sys::to_host_format<unsig
```

```
$ perf record -F 199 -g ./myprog    # 199 Гц + стек вызовов
$ perf report                       # интерактивная таблица
```




```
$ perf record -F 199 -g ./myprog
$ cp ~/github/FlameGraph/*.pl .   # копируем скрипты
$ perf script | ./stackcollapse-perf.pl | ./flamegraph.pl > g.svg
```

еще графики

Инструмент	Накладные расходы	Портируемость	Автом.	Ед. измерения
`rdtsc`	низкие	нет	нет	циклы
`std::chrono`	средние	да	нет	микросекунды
`perf`	высокие	да	да	проценты

Статический анализ кода

Тестовая программа:

```
int func() {
    std::cout << "111\n";
}
int main() {
    if (func() > 10) {
        std::cout << ">10\n";
    }
}
```

В терминале:

```
$ g++ -Werror=return-type main.cc -o main.o
main.cc: В функции «int func()»:
main.cc:4:1: ошибка: в функции, которая должна возвращать значение,
отсутствует оператор return [-Werror=return-type]
```

Статический анализ кода

Тестовая программа:

```
namespace {
    int func() { return 0; }
}
int main() {}
```

В терминале:

```
$ cppcheck --enable=all main.cc
Checking main.cc ...
[main.cc:2]: (style) The function 'func' is never used.
```

Окружение (зависимости)

- Singularity
- Docker
- Vagrant

Рецепт Singularity

```
Bootstrap: yum
OSVersion: 28
MirrorURL: https://...
Include: dnf

%post
dnf install -y gcc-c++ meson gtest-devel git
git clone https://... .
meson . build
cd build
ninja install
dnf erase -y gcc-c++ meson gtest-devel git
dnf clean all
rm -rf /var/cache/*

%runscript
/path/to/your/app
```

Singularity и Docker

```
$ singularity build my-python docker://python:latest
$ ./my-python --version
```

Singularity и Docker

```
Bootstrap: docker
From: ubuntu:16.04

%post
apt-get -y update
apt-get -y install fortune cowsay lolcat

%environment
export LC_ALL=C
export PATH=/usr/games:$PATH

%runscript
fortune | cowsay | lolcat
```

Singularity и OpenGL

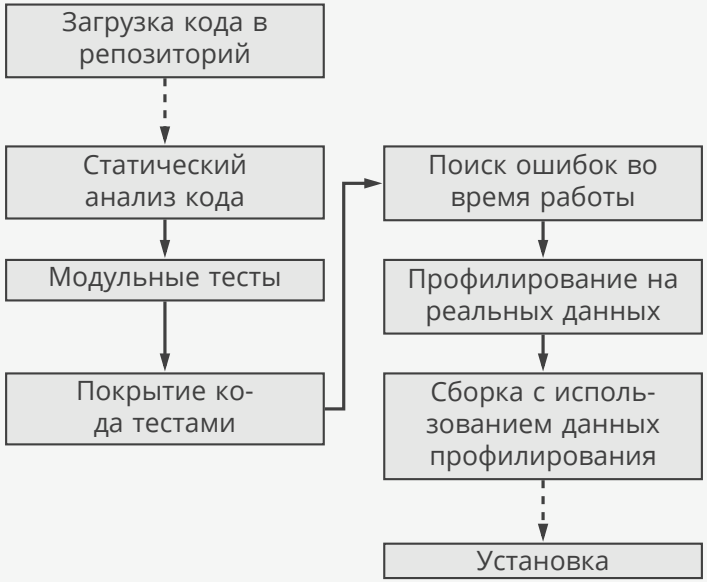
```
Bootstrap: yum
OSVersion: 28
MirrorURL: https://...
Include: dnf

%post
dnf --refresh -y install VirtualGL hostname mesa-dri-drivers

%runscript
vglrun glxspheres64
```

По типу программ:

- Docker: сервисы на несколько узлов и т.п.
- Singularity: пакетная обработка данных, высокопроизводительные вычисления, сборка кода.
- Vagrant: сборка кода, сервисы на несколько узлов, самый переносимый вариант.

По назначению:

- Docker: тестирование, развертка.
- Singularity: сборка, тестирование, развертка.
- Vagrant: сборка, тестирование.


Загрузка кода в репозиторий
Статический анализ кода
Модульные тесты
Покрытие кода тестами
Поиск ошибок во время работы
Профилирование на реальных данных
Сборка с использованием данных профилирования
Установка

Ссылки

- How SQLite is tested?
- Flame Graph.
- Singularity User Guide.