ПРОЕКТ по языку Си + PostgreSQL на тему:

"Оптимизация производительности БД PostgreSQL путём автоматического распараллеливания задач"

Оглавление:

Сартаков Алексей
OTUS-2023-C01-SARTAKOV-AP · he/him

Проект выполнил:
Сартаков Алексей Петрович
Тел: *+7-910-797-8034*
Город: *(Нижний Новгород)*
Телеграмм: *https://t.me/Sartakov_Aleksey*

Дата выполнения: *декабре 2023г.*
Ученик курса: *«Программист Си»*
Платформа: *ОТУС (Otus.ru)*

Проверяющий проекта (код ревьюер):
Андрей Кравчук
https://t.me/awkravchuk

Вводная часть.

Работа по анализу биржевых индикаторов требует очень много вычислений.

Имея в распоряжении средний ПК (6 ядер 16 Гб оперативки, SSD диск) хочется получить от него как можно более высокую отдачу. Имея код, который плохо поддаётся распараллеливанию, который при каждом запуске функций проходит структурный анализ кода и сборку функции на её запуск (PostgreSQL) и который нагружает ПК всего на 7%, тут волей неволей начнёшь искать пути по оптимизации решений.

И главная задача данного проекта ускорить вычисление однотипных расчетов, выполняемых в БД PostgreSQL, написанных на процедурном языке PL/pqSQL.

Первое что приходило в голову - написание параллельных функций, которые работают в нескольких потоках . - да работает, но очень сложно все это запускать и поддерживать. Правка функций так же требовала нетривиального подхода. Возможно тогда не был знаком с PostgreSQL (работал на Cache InterSystems), но более 5 параллельных потоков запускать уже было проблемно, регулировать их нагрузку было проблемно, модифицировать код было проблемно. Причем от слова – ОЧЕНЬ НЕУДОБНО.

Потом был переход на PostgreSQL 9.5 , затем 10 и сразу на 11, Затем с выходом 13 версии почти сразу перешёл на неё. Да, увеличилась скорость, удобство написания кода, но сложности с разветвлением потоков - никуда не делись. Так же оставались вопросы с пересборкой функций перед каждым их выполнением. Для обычных задач это не так критично, но для мелких и частых - потери на компиляцию могут составляют до 40% (примеры ниже). Что явно требовало иного подхода.

Python отпал после года работы на нём - написание кода неудобно, скорость работы приложений на Python обычно дольше чем на PL\pgSQL (редко когда Питон был быстрее), а если нужна скорость - то нужна работа с кучей библиотек в Python. Вроде изучаешь один язык а по факту кучу разрозненных приложений. В общем, и по скорости работы (в 80 % случаев) да и по удобству Python явно проигрывал встроенному языку PL\pgSQL в PostgreSQL.

Из всех возможных вариантов оставалось или ручное разделение потоков на фоновые задачи или что то кардинальное - выбрал кардинальное. **Так остановился на языке Си.** Как позже выяснилось - разделение потоков в Python нетривиальная задача, которая к тому же не гарантирует адекватный результат (отдача на затраченное время и стабильность работы).

Честно говоря, пытался изучить язык Си самостоятельно, и как доходил до указателей (*char[], ...) всегда стопорился. И так продолжалось года 2 (Си стал изучать

на год позже Python, примерно с конца 2020). После чего решил заплатить за курсы и все таки выучить Си.

В данном проекте №_1 все задачи и их выполнение будут происходить в нескольких режимах, чтобы хотя бы в грубой форме понять - «стоила ли овчинка выделки». Так ли будет хорош переход от PL/pqSQL к коду на чистом СИ и насколько приближусь к поставленной цели по оптимизации скорости работы приложения и удобству написания кода в режиме многопоточности при переходе на язык СИ.

P.S в ходе изучения Си, уровень поставленных задач расширился до: графического интерфейса, работа с русских текстом в файлах и терминале в Debian 11, сбор статистики из ОС Debian 11 в приложения на Си и в БД PostgreSQL. К данному проекту **есть Проект №_2 по сбору данных о нагрузке ПК**, который напрямую влияет на текущий проект! (см примечания).

Задачи проекта.

Список задач, которые были поставлены на момент начала написания проекта.

ЗАДАЧА №1. Основная цель – ускорить выборку данных из БД PostgreSQL 15 и генерация ТаймФрэймов (свечи с указанным временным периодом, к примеру 3 минуты или 6 минту). При работе в БД эти вычисления занимают чуть более 28 минут.

Они не критичны, т.к их расчеты можно запустить на 15 минут раньше или позже.

Главная цель Задачи №1 сократить время предстартовых вычислений. На основе этого кода создать код больше ориентирована на «Экстренный старт программы».

«Экстренный старт программы». – это когда торговый терминал по каким то причинам давал сбой (пропажа света, сбой на ММВБ, перезагрузка ПК). Такие оказии стабильно случаются 1 раз в полгода и для выхода программы в штатный режим требуется около 20 минут на каждый час торгового дня. Это очень проблемная область работы торгового робота.

ЗАДАЧА №2. В виду того, что все расчеты происходят на ноутбуке, так же должен быть введён контроль предельной нагрузки сервера. Температура процессора должна кратковременно составлять не более 70-92°C и нагрузка CPU не более 50-70%. Это было реализовано с помощью проекта №_2. Смотри Примечание_1.

ЗАДАЧА №3. Вычисления множества индикаторов. Все данные хранятся в той же БД. Нужно сделать предстартовые вычисления индикаторов, а так же делать вычисления тех же индикаторов в режиме реального времени. Здесь. куча мелких однотипных вычислений по 240 биржевым инструментам по 80 параметрам в непрерывном режиме с 10:00 до 23:50.

> Основная вычислительная мощность сервера и затрачиваемое время уходит именно на расчет всех индикаторов по биржевым торговым инструментам. Одна задача решается примерно 0,01 секунды, **но 80*0,01*240 =~ 192 секунд**, а ПК при этом загружен всего на 7%, то есть этот поток теоретически можно раздробить на 6 частей и примерно снизить скорость работы одного цикла до 35-40 секунд (коэффициент распараллеливания кода примерно 50%), плюс сам код написанный на СИ должен работать быстрее, чем тот же код, написанный на процедурном языке PL/pgSQL.

Задача № 3 – снизить время вычислений с 40 секунд до 7 секунд в идеале.

***ВНИМАНИЕ**! Данный проект заточен на работу в ОС Debian 11 с русской кодировкой/локалью системы (ru_RU.UTF-8). В коде, названиях файлов будут использоваться русские буквы. Все внешние текстовые файлы создаются как UTF-8.*

Проект делался на физической машине (не виртуальной). То есть это полноценный материнский (основной) ПК без применения каких либо виртуальных устройств/ОС.

Нюансы проекта.

Главная задача проекта – понять, будут ли улучшения и насколько они будут значимыми пр работе кода в БД PostgreSQL 15 и при работе программы на Си.

А главная задача программы – анализ торговых индикаторов на ММВБ и как/где это делается – не важно.

Главным и очень сильным нюансом является НЕВОЗМОЖНОСТЬ сравнить эти две технологии в лоб.

Дело в том, что при работе в БД и при работе в Си один и тот же подход даёт очень сильные отличия в результатах.

Так, в Си, в основном используются циклы, а вот в БД при работе с циклами (в лоб) очень сильно страдает производительность – раз так в 10-15. Поэтому в БД и используются:

- СОЕДИНЕНИЯ (JOIN),
- КУРСОРЫ (FETCH),
- CTE выборки (with t0 as (...) select ... From t0).
- Оконные функции (OVER)
- Группировка (GROUP BY)

Я хорошо помню первые проблемы выборок больших объемов данных в форме цикла «FOR» внутри БД , он занимал около 10 секунд на 1 акцию.Это значение просто было недопустимым по времени работы. Первая доработка кода для ускорения быстродействия была лет так 5 назад и длилась около 2-3 недель, прежде чем я понял в чем подвох. После модификации, время работы кода сократилось ДО 0,6 секунд, то есть почти в 17 раз.

В результате этого – кода на Си и код работы с БД сильно отличаются (на 100%). Я допускаю, что код на стороне БД можно оптимизировать ещё, но существенных ускорения в работе это не даст. Важен сам порядок цифр/времени работы обоих кодов.

Без Базы Данных код на Си работать не будет, т.к все исторические данные для работы, программа на Си берёт из БД. Копия БД лежит на GitHub в отдельной папке (см ниже инструкцию ниже по установке и настройке БД PostgreSQL 15).

Выложенный код на Си является достаточным для понимания отличий по времени работы между БД PostgreSQL, которая считается одной из самой быстрых/универсальной БД и между программой, написанной на чистом языке Си.

Выложенный в репозитории код на Си составляет около 20% от всего рабочего кода.

Выложенный код БД составляет около 10% от всего работающего кода БД. Это просто к тому, какой объем работы вам потребуется сделать, если захотите создать полностью рабочую программу самостоятельно.

Проект уникален тем, что сравнения, которое здесь производится, доселе никто не делал (в открытом доступе такого точно нет).

Настройки, для развертывания на сервере/ПК.

Порядок компиляции/сборки программы на Си изложена в папке «**код_программы_на_си**» в файле «**запуск_сборки.sh**». По сути, нужно просто скопировать скачанный каталог с программой на Си и запустить «*.sh» файл. Но Базу Данных придётся устанавливать отдельно и там так же всего одна строчка для установки БД (описано ниже). Архив БД см Примечание_2.

НЮАНСЫ при работе с БД PostgreSQL. Дело в том, что не настроенный Postgres иногда выдаёт результат по скорости работы в разы хуже, чем оптимизированный. Отличия достигают до 3-х раз по скорости работы. А вот на программе написанной в Си это никак не сказывается.

Для запуска и настройки самой программы достаточно запустить её без ключей. Появится справка по работе с программой и настройкам ключей.

Рисунок №1

В реальности содержание окна подсказки может быть другим.

```
Программа НЕ имеет нужного количества параметров на входе. Сейчас указан(о) 1 параметр(ов).

    *** Инструкция по использованию ключей для корректной работы программы ***
==========================================================================
 Данная программа каждые 60 секунд обновляет данные о нагрузке операционной системы в БД PostgreSQL.
 Программа сохраняет информацию по следующим параметрам:
 НАГРУЗКА CPU. Записывается максимальная, минимальная и средне арифметическая нагрузка системы.
 ОПЕРАТИВНАЯ ПАМЯТЬ. Записывается максимальная, минимальная и средне арифметическое использование памяти.
 ИСПОЛЬЗОВАНИЕ SWOP диска. Указывает количество записанных и считанных на/с SSD диска блоков памяти.
 ТЕМПЕРАТУРА CPU. Записывается максимальная, минимальная и средне арифметическая температура процессора.
 Среднее арифмитическое считается как сумма всех замеров КАЖДЫЕ 2 или N секунды и делённое на количество этих замеров
.
 Каждые 5 (пять) минут в БД создаётся новая запись и последующие 5 минут обновляется именно эта запись.

 Программу можно использовать для авто запуска после перезагрузки операционной системы (через DEMON операц.сист.).
 Для разделении передаваемых в программу ключей нужно использовать пробел или табуляцию.
 Если в каком то имени есть пробел, то такое имя нужно взять в двойные кавычки с каждой стороны, например:
         -schema "схема тест"

 Список ключей и их значение ЖИРНЫЙ - обязательный параметр:
 -db             Название БД в PostgreSQL.
 -schema         Имя схемы в БД в PostgreSQL, в которой находится таблица для внесения записей.
 -table          Имя таблицы в БД в PostgreSQL, в которую вносятся данные о нагрузке ПК/сервера. По умолчанию = 'нагр
узка_системы'
 -host           Имя хоста. Можно указывать как localhost так и его IP адрес в сети (IP4). ПО УМОЛЧАНИЮ = localhost
 -port           Имя порта (socket). По умолчанию PostgreSQL использует порт 5432, но это может быть и 5433 и 5434, .
.. ПО УМОЛЧАНИЮ = 5432
 -user           Имя пользователя (название роли) в БД PostgreSQL от имени которого будут вноситься записи в таблицу
базы данных.
 -psw            Пароль пользователя для доступа к Базе Данных (таблице) в PostgreSQL.
 -delay          Задержка, интервал в целых секундах между считыванием нагрузки операционной системы.
                 Варианты: 1,2,3,4,5,6,10,12,15 секунд. По умолчанию, замеры каждые 2 секунды.
 -log_path       Полный и Абсолютный путь к лог файлу, куда будут писаться ошибки и результат работы программы,  т.к.
 она подсоединяется к БД, то важно контролировать ошибки.
                 Путь по умолчанию = '/var/log/z_nagruzka_pc.log'  но с правами root.
```

После компиляции, программу можно запустить в виде строк кода:

```
cd папка_в_которй_лежит_программа_на_СИ
./ускорение_бд_postgresql -db test_si_2023 -host localhost -num_threads 1 \
-port 5432(или другой) -user postgres -psw 123 -log_path \
/home/postgres/z_mmvb_temp/z_ускорение_бд.log -test 2
```

После «\» не должно быть никаких символов, в том числе и пробелов. **-num_threads 1** это количество потоков (от 1 до 12 максимум)

- В настройке БД PostgreSQL должен иметься/существовать пользователь от имени которого мы соединяемся из программы с БД PostgreSQL.
- Для этого пользователя в БД лучше создать отдельные права на доступ именно к указанной таблице (там потребуется подцепить так же несколько еще дополнительных таблиц). Настройки пользователя в БД PostgreSQL выходят за рамки проекта (там очень все непросто). Можете создать в Посгрес пользователя с правами суперпользователя, хотя в рабочей БД это и не рекомендуется (но у нас тестовая).
- Для работы с БД должен быть открыт доступ из вне (если соединяетесь по сети а не на локальном ПК).

После настройки БД в двух файлах ниже, нужно перезапустить БД PostgreSQL (через «sudo systemctl ...» или через перезагрузку ПК), чтобы оптимизация скорости PostgreSQL вступили в силу.

Настройки PostgreSQL файла /etc/postgresql/15/имя_*/postgresql.conf

```
listen_addresses = '*'          # what IP address(es) to listen on;
max_connections = 200

shared_buffers = 2000MB       # рекомендуется 60-70% от всей оперативки, если работает только одна БД (всего одна копия БД и больше ничего). Если две копии БД (13 и 15 версия или две 15 версии), то оперативку нужно разделить на все БД. Если еще и почтовая служба и еще куча программ, то на БД 40-50% от всей RAM памяти.
temp_buffers = 128MB
work_mem = 200MB
maintenance_work_mem = 256MB

max_worker_processes = 12
max_parallel_workers_per_gather = 4
max_parallel_maintenance_workers = 4
max_parallel_workers = 8

track_activity_query_size = 2048
```

Настройки PostgreSQL файла /etc/postgresql/15/имя_*/pg_hba.conf

```
# TYPE  DATABASE        USER            ADDRESS                 METHOD
# "local" is for Unix domain socket connections only
local   ваша_бд         пользователь_в_бд                       peer
local   all             all                                     scram-sha-256

# IPv4 local connections:
host    all             all             127.0.0.1/32            scram-sha-256
host    all             all             192.168.2.0/24          scram-sha-256
```

Желательно, то же имя, что указано в БД имелось в вашей системе Debian/Ubuntu. Тогда вы сможете запускать скрипты для работы с БД без ввода пароля (эта тема выходит за рамки текущего проекта).

Для разворачивания ПОЛНОЦЕННОЙ копии БД **test_si_2023** текущего проекта на вашем сервере. (сама БД PostgreSQL уже должна быть установлена на вашем ПК) нужно скачать архив с БД, разархивировать его и запустить следующий код по переносу содержимого БД.

1. Сперва создаём саму БД на вашем сервере: подключаемся к серверу на котором будем создавать тестовую БД, к примеру **psql –p 5432**
2. **create database test_si_2023;**
3. **alter role postgres password '123';** если у вас еще не активирована учетная запись в БД (только создали БД).
4. Выходим из psql через \q
5. Распаковываем БД из архива **(архив формата *.7z)**
6. В окне терминала запускаем код по переносу БД:
   psql -h localhost -p 5432 -d test_si_2023 -U postgres < /home/postgres/адрес_к_бд/имя_бд.sql (путь к архиву БД)
7. Подключаемся к БД PostgreSQL через программу DBeaver 23.2.*
8. Начинаем работу через DBeaver или окно терминала.

Практически ко всем таблицам и функциям внутри БД имеются примечания (смотрим БД через DBeaver 23.2.*) .

В БД имеется одна роль = postgres и пароль от неё 123.

Сперва должна быть создана БД на вашем ПК, сперпользователь, пароль суперпользователя и только потом нужно делать импорт тестовой БД в вашу систему!

ИТОГИ. Сравнение результатов Си и PostgreSQL.

Всего ПК имеет 6 реальных ядер и 12 с режимом Hyper threading technology.

Хочется обратить особое внимание на то, что PostgreSQL быстро работает только при выделении 2-4 Гб оперативной памяти.

При этом, для работы самой Си программы потребовалось около 20 Мб на всё!

Всего для последующего анализа подходит 165 инструментов
размер памяти на 1 элемент таймфрэйм_m1: 10160
для всей структуры таймфрэйм_шаблон: 1679344 (всего 11 ТаймФрэймов)

ИТОГО = 11*(10’600 +1’679’244) = 18’588’284 байт

Запуск программы на Си из Debian выглядит примерно как:

```
cd путь_к_папке_с_программой
./ускорение_бд_postgresql -db test_si_2023 -host localhost -num_threads 1 -port 5434 -user postgres -psw 123 -log_path /home/postgres/z_mmvb_temp/z_ускорение_бд.log -test 2
```

Запуск программы в БД PostgreSQL 13 и выше из Debian выгдлядит как

1) Скачать и установить к себе локальную версию БД из GitHub.com (Смотри Примечание_2)
2) Зайти в БД PostgreSQL через терминал как psql -d test_si_2023 -h localhost -p 5434 -U postgres
3) Ввести свой пароль пользователя (для postgres)
4) Ввести код call торги_реальн_время.p_0_предзапуск(); (точка с запятой в конце ОБЯЗАТЕЛЬНА)

Запуск в PostgreSQL.

При работе в PostgreSQL 15 , программа выдаёт результаты по скорости работы как на скриншоте ниже.

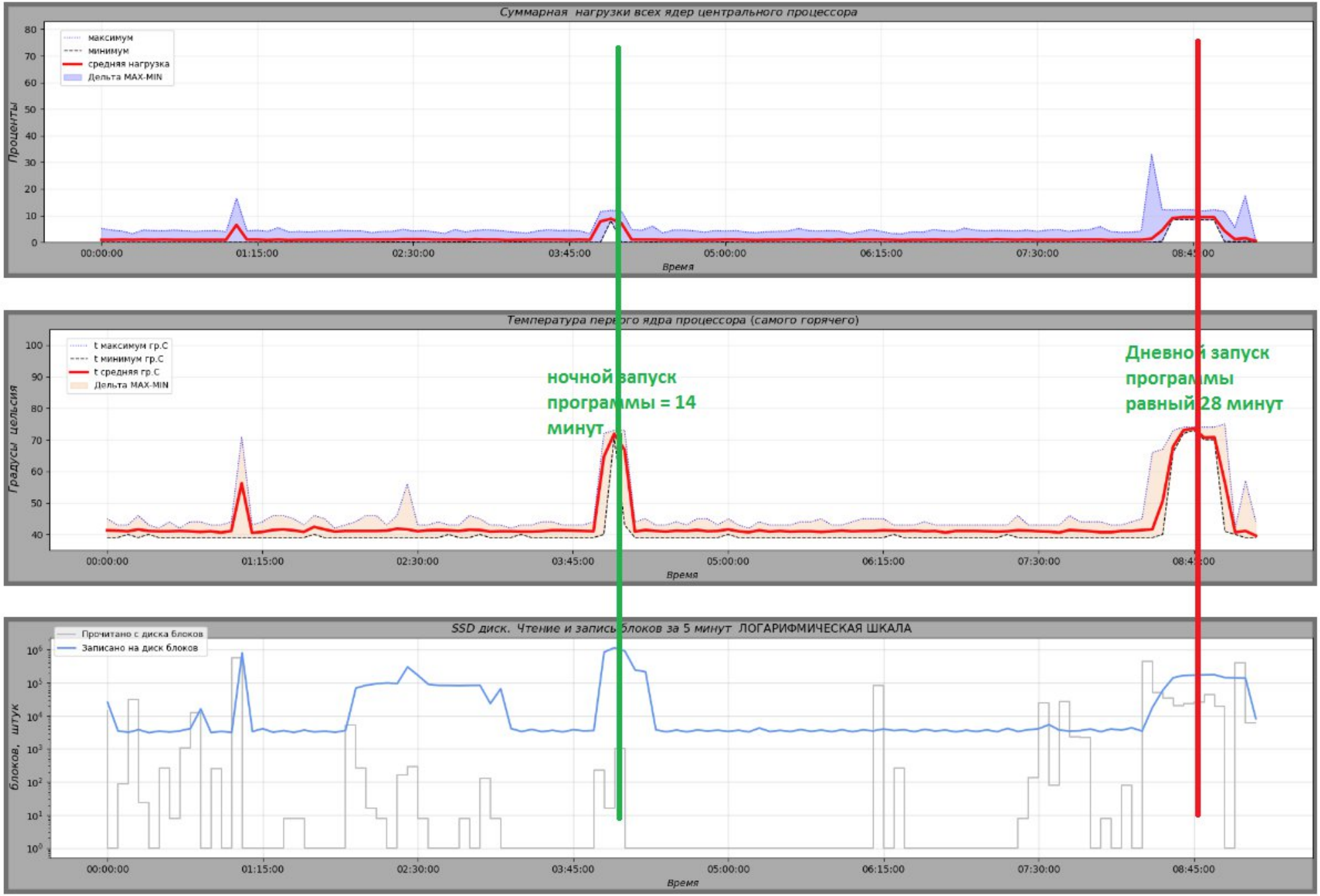
Как можно заметить, среднее время работы около 14 минут. При запуске этого же блока расчетов в дневное/рабочее время, скорость расчетов снижается и увеличивается примерно до 15 минут 30 сек (скриншоты в разделе «Возникшие проблемы и способы их решений»).

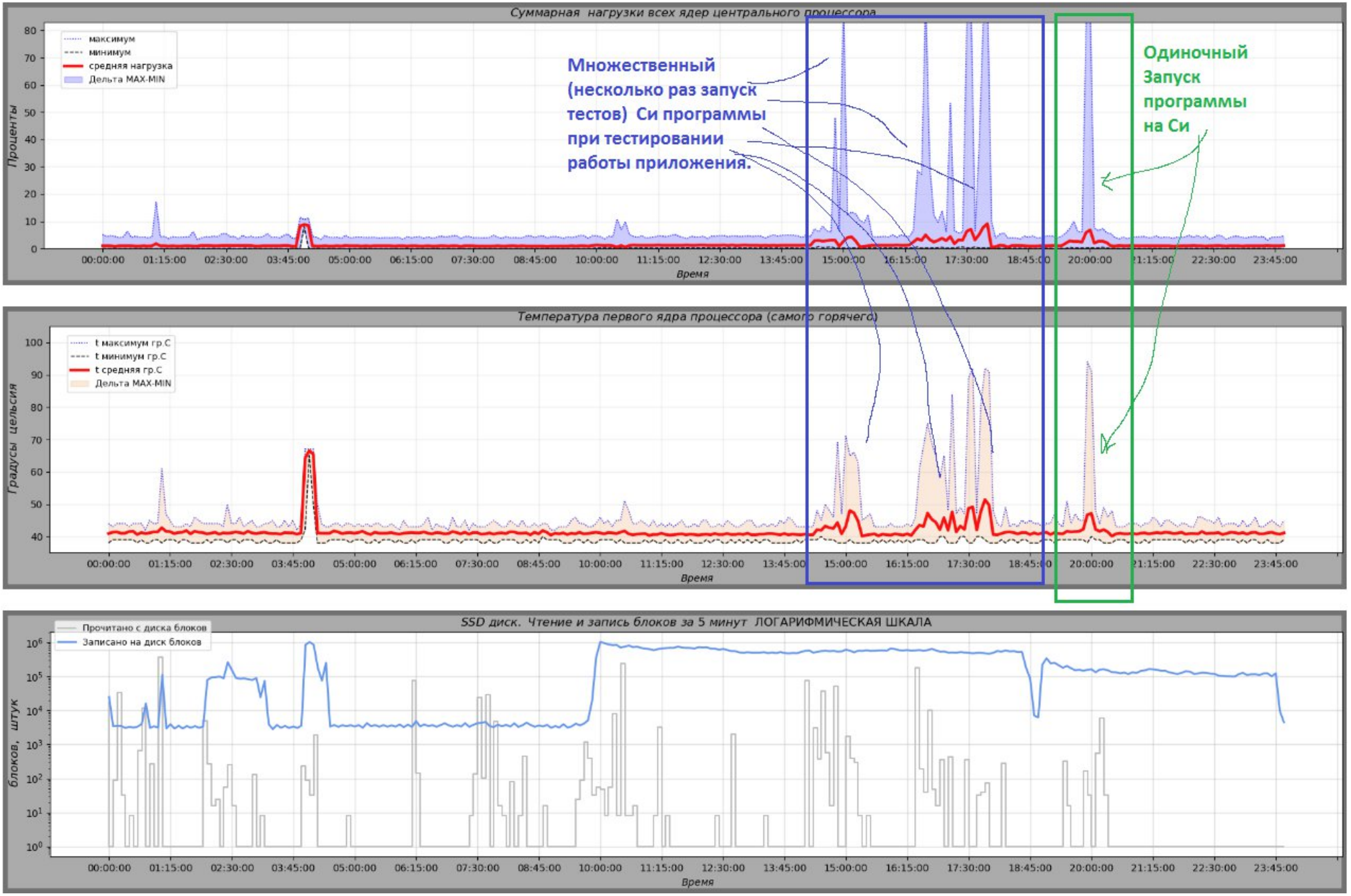
Поэтому, возьмём среднее время работы расчетов в БД PostgreSQL равным 15 минутам.

Запрос ошибок в БД. Доступ ограничен.

Отчёты об ошибках::

№	ИМЯ СХЕМЫ + ФУНКЦИИ	РЕЗУЛЬТАТ	ОПИСАНИЕ ОШИБКИ	ДАТА ЗАВЕРШЕНИЯ	ЗАВЕРШИЛИ В ЧЧ:ММ:СС	ТАЙМИНГ ЧЧ:ММ:СС.MS
1	ежедневный предзапуск по индикаторам()	ok		2024-01-17	04:14:01	00:13:25.14
2	ночь_ежедневка_общий_итог утро среды: последний d1 = 2024-01-16 последний m1 = 2024-01-16 23:49:00 последний h1 = 2024-01-16 23:00:00 последний week_1 = 2024-01-17 последний month_1 = 2024-01-17 Δ газпром (нов-стар): -95.5 в млн ₽	ok		2024-01-17	03:05:03	01:04:21.09
3	ежедневный предзапуск по индикаторам()	ok		2024-01-16	04:14:21	00:13:45.82
4	ночь_ежедневка_общий_итог утро вторника: последний d1 = 2024-01-15 последний m1 = 2024-01-15 23:49:00 последний h1 = 2024-01-15 23:00:00 последний week_1 = 2024-01-16 последний month_1 = 2024-01-16 Δ газпром (нов-стар): -47.6 в млн ₽	ok		2024-01-16	03:05:23	01:04:40.98
5	ежедневный предзапуск по индикаторам()	ok		2024-01-15	04:14:53	00:14:17.24
6	ночь_ежедневка_общий_итог утро понедель-ника: последний d1 = 2024-01-12 последний m1 = 2024-01-12 23:49:00 последний h1 = 2024-01-12 23:00:00 последний week_1 = 2024-01-13 последний month_1 = 2024-01-13 Δ газпром (нов-стар): 0.0 в млн ₽	ok		2024-01-15	02:58:36	00:57:53.62


Суммарная нагрузки всех ядер центрального процессора
максимум
минимум
средняя нагрузка
Дельта MAX-MIN
Проценты
Время
Температура первого ядра процессора (самого горячего)
t максимум гр.С
t минимум гр.С
t средняя гр.С
Дельта MAX-MIN
Градусы цельсия
ночной запуск программы = 14 минут
Дневной запуск программы равный 28 минут
SSD диск. Чтение и запись блоков за 5 минут ЛОГАРИФМИЧЕСКАЯ ШКАЛА
Прочитано с диска блоков
Записано на диск блоков
блоков, штук

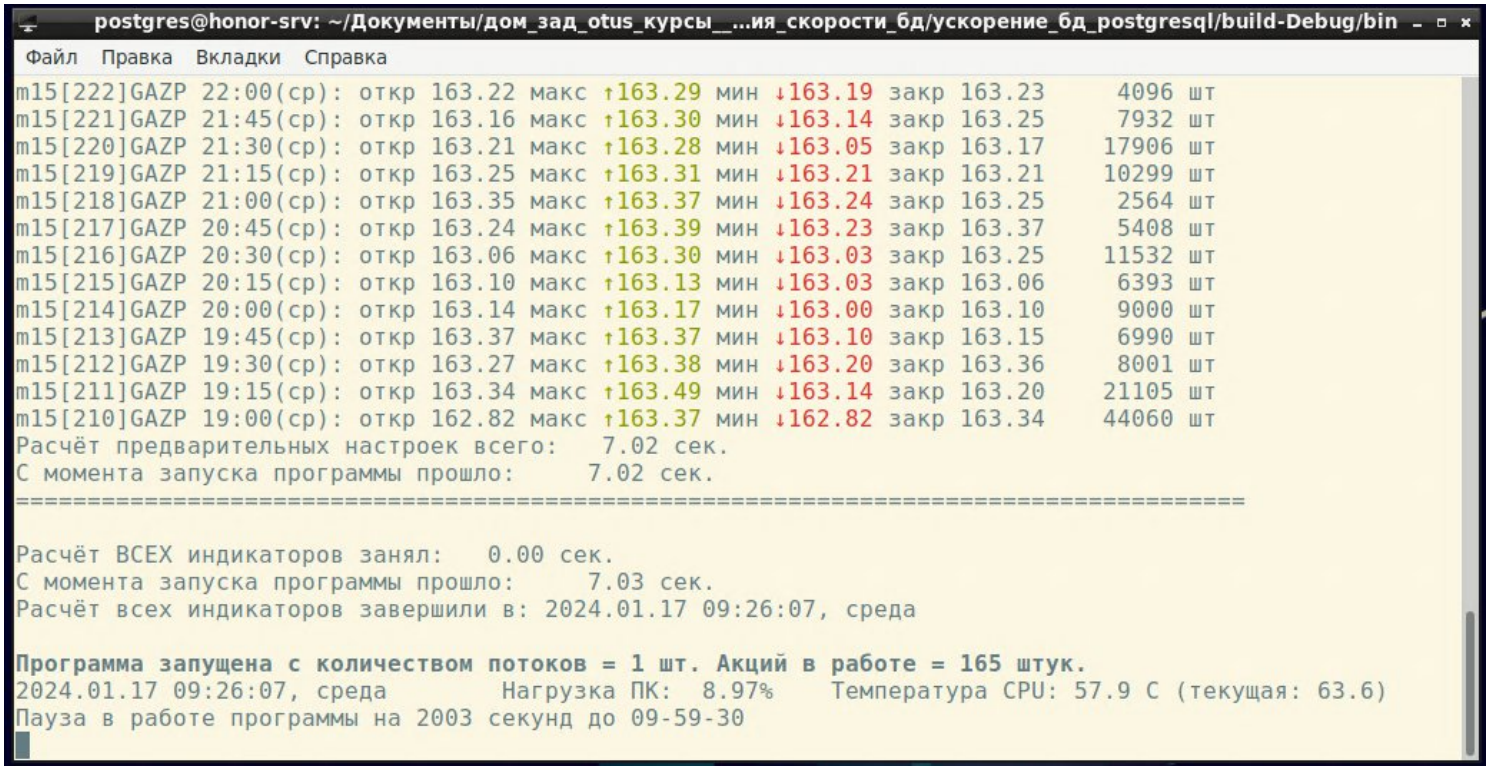

Суммарная нагрузки всех ядер центрального процессора
максимум
минимум
средняя нагрузка
Дельта MAX-MIN
Проценты
Время
Множественный (несколько раз запуск тестов) Си программы при тестировании работы приложения.
Одиночный Запуск программы на Си
Температура первого ядра процессора (самого горячего)
t максимум гр.С
t минимум гр.С
t средняя гр.С
Дельта MAX-MIN
Градусы цельсия
Время
SSD диск. Чтение и запись блоков за 5 минут ЛОГАРИФМИЧЕСКАЯ ШКАЛА
Прочитано с диска блоков
Записано на диск блоков
блоков, штук
Время

Запуск в программе на Си.

Здесь картина радует глаз. В одно поточном режиме программа на Си делает выборки из БД Postgres и компоновки таймфрэймов за 7 секунд ВСЕГО. Вместо 15 минут. **Разница составляет 128 раз!**

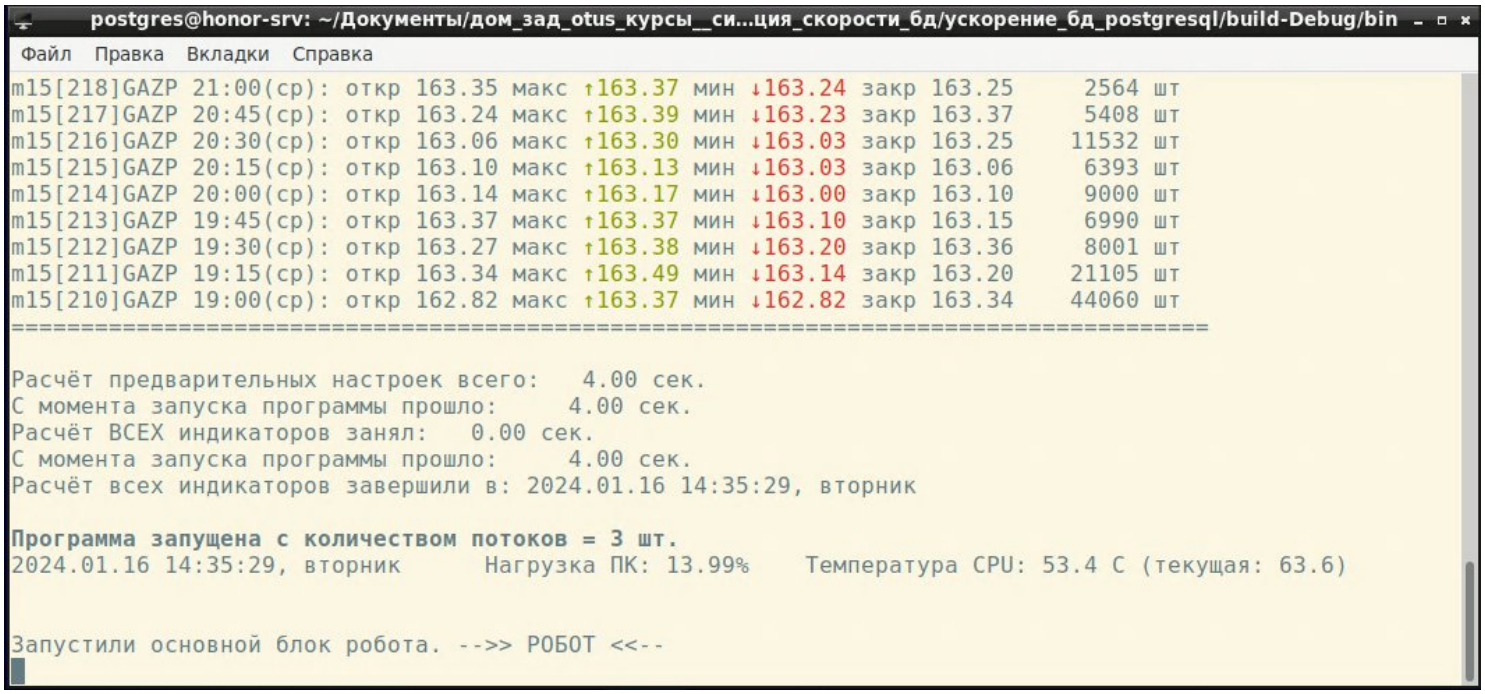
Вы можете сами все проверить – при выводе – можно выводить значения в терминал запущенной программы на Си.

Расчет нагрузки в программе на Си в 1 поток.



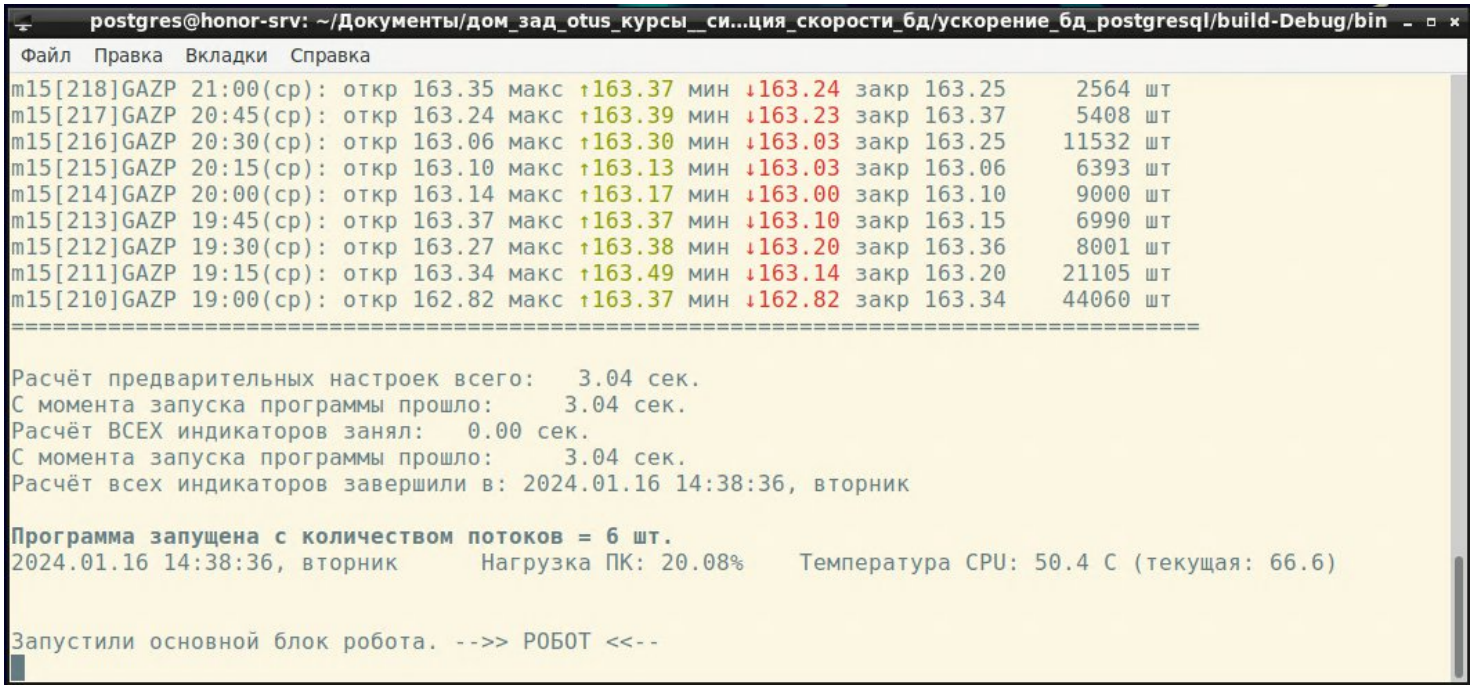
Теперь запуск в 3 потока (половина от физического количества ядер ПК).

Расчет нагрузки в программе на Си в 3 потока.



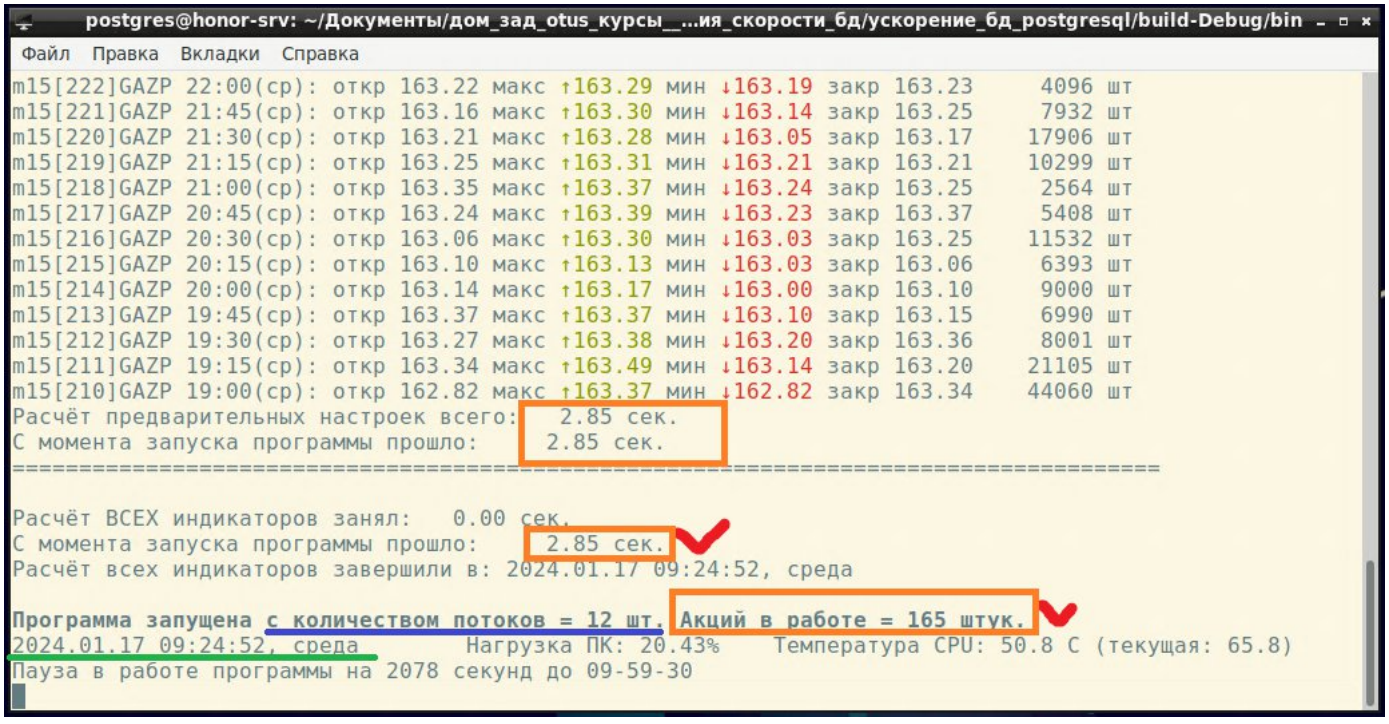
Запуск в 6 потоков – задействуем все физические ядра процессора ПК.

Расчет нагрузки в программе на Си в 6 потоков.



Запуск в 12 потоков – задействуем все физические ядра процессора ПК и все виртуальные ядра процессора.

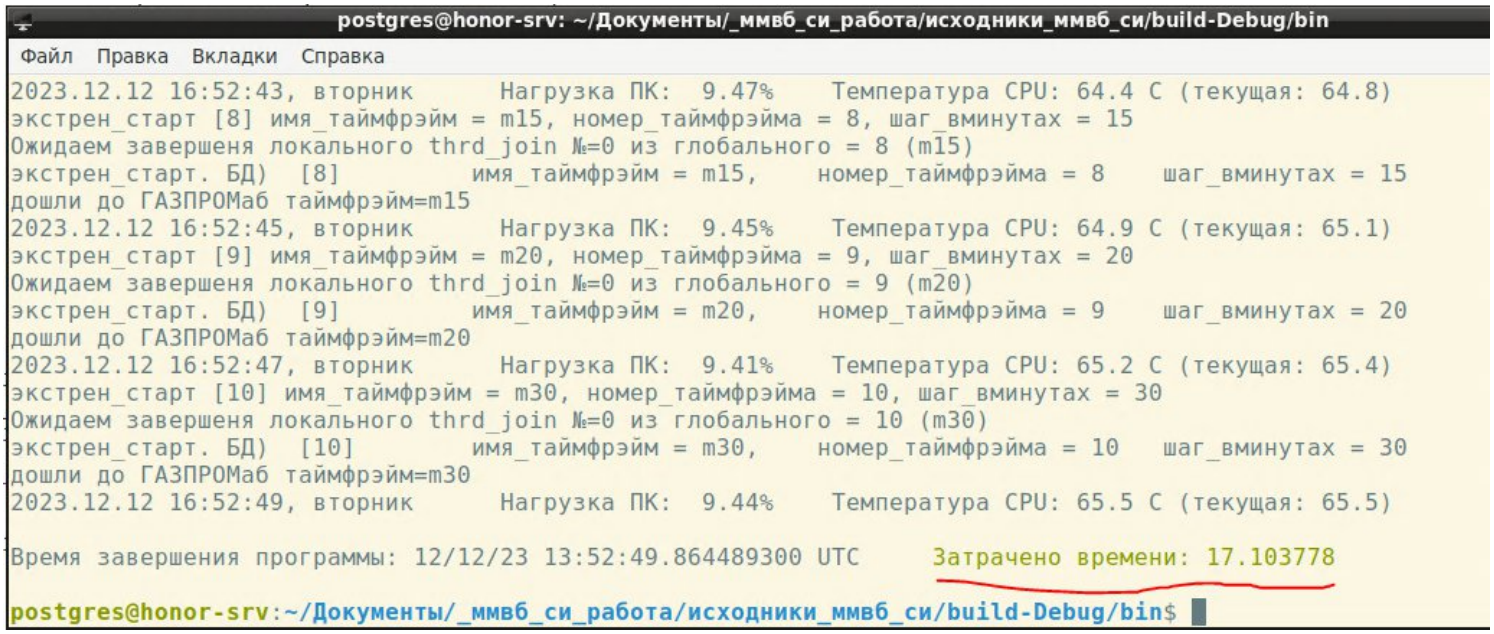
Расчет нагрузки в программе на Си в 12 потоков.



Так же привожу пример времени расчетов в программе Си с учетом выборки из БД и расчета всех индикаторов по всем акциям из фильтра по всем ТаймФрэймам.

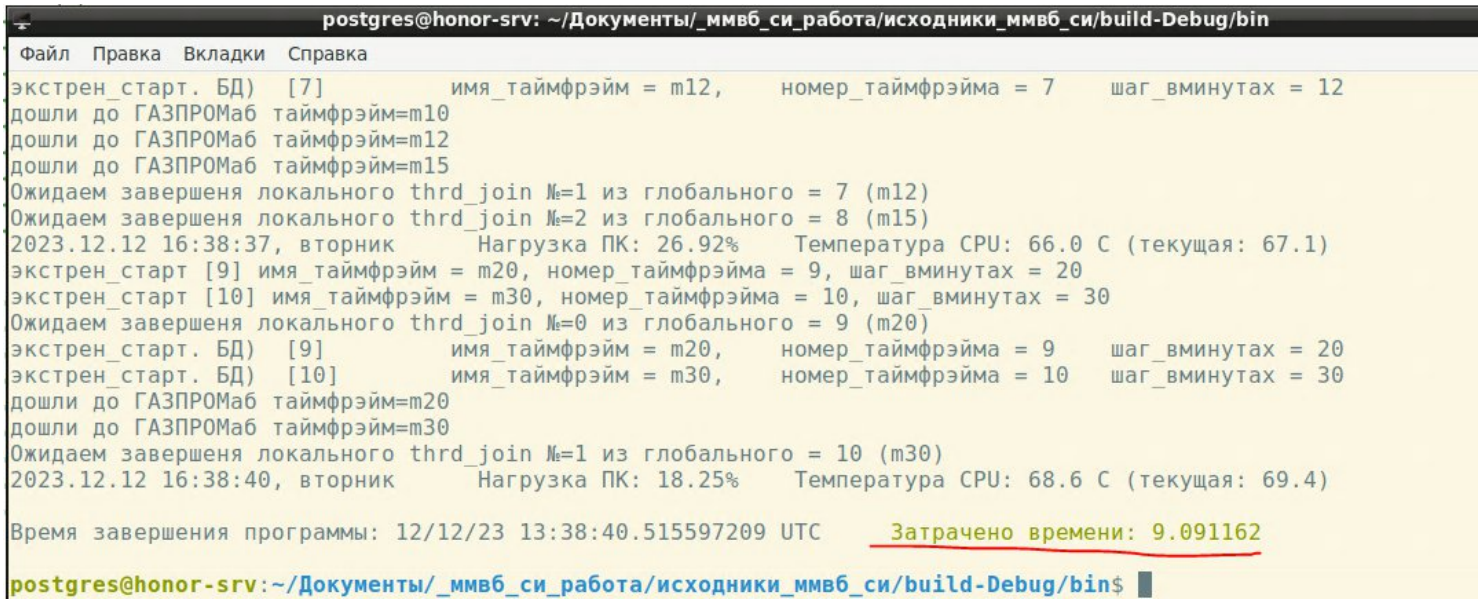
Этот код отсутствует в проекте, но позволяет дать примерное понятие о времени работы программы на Си. ***По аналогии****, это примерно соответствует времени ночных выборок и расчетов индикаторов в PostgreSQL, которые равны 14 минутам ночью и 15 минутам днём.*

Выборка из БД + вычисления всех индикаторов всех акций. 1 поток.



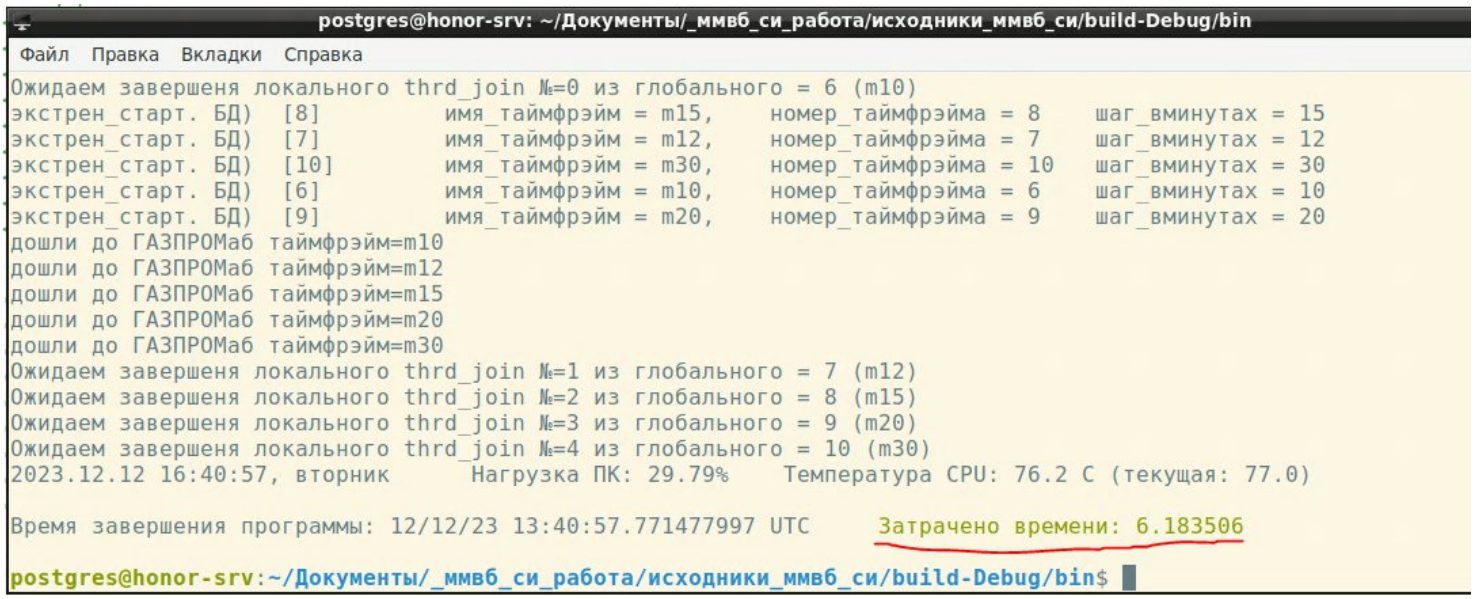
То есть в **однопоточном** режиме программы Си, ВСЕ расчёты занимают примерно 17 секунд. В БД те же самые расчеты занимают от 14 до 15 минут. Итого, чистая разница в производительности составляет для ОДНОПОТОЧНОГО режима **более 50 раз !** (17 секунд и от 14 минут).

Выборка из БД + вычисления всех индикаторов всех акций. 3 потока.



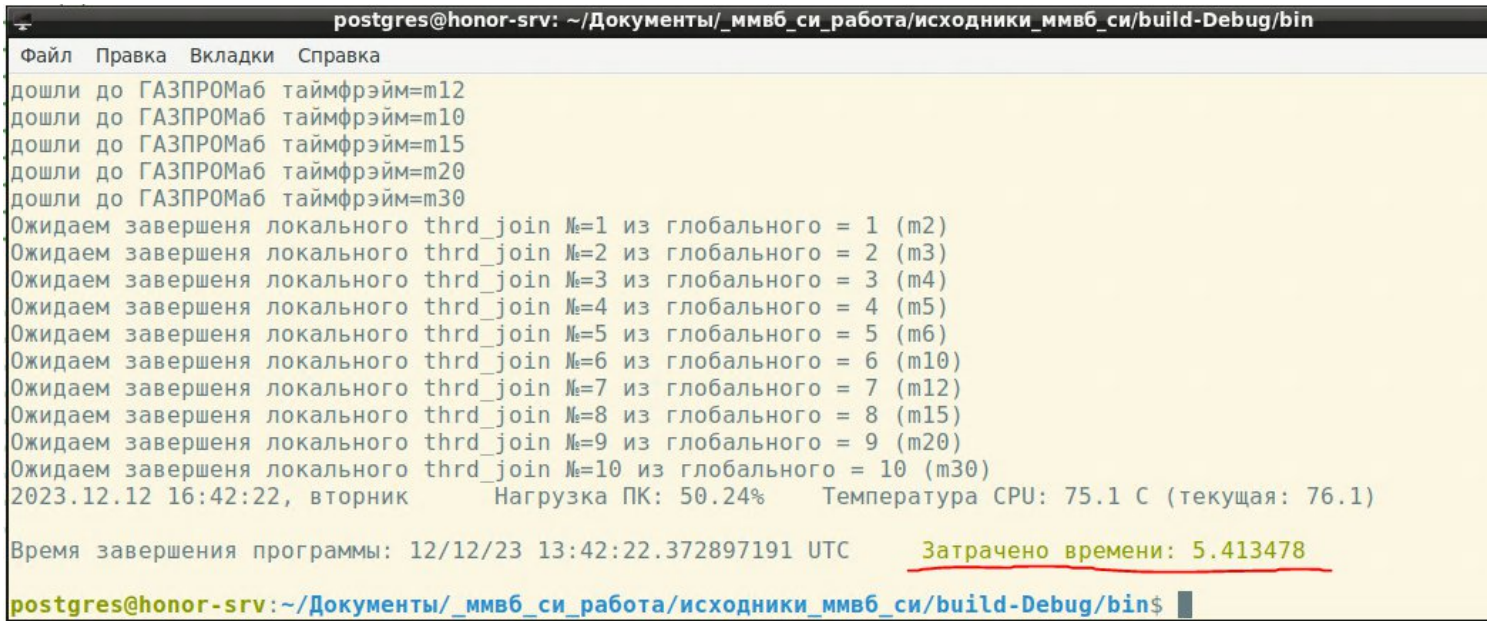
Для 3-х потоков эта разница уже составляет более 93 раз (9 сек и от 14 минут).

Выборка из БД + вычисления всех индикаторов всех акций. 6 потоков.



Для 6-ти потоков эта разница уже составляет более 135 раз (6.2 сек и от 14 минут).

Выборка из БД + вычисления всех индикаторов всех акций. 6 потоков.

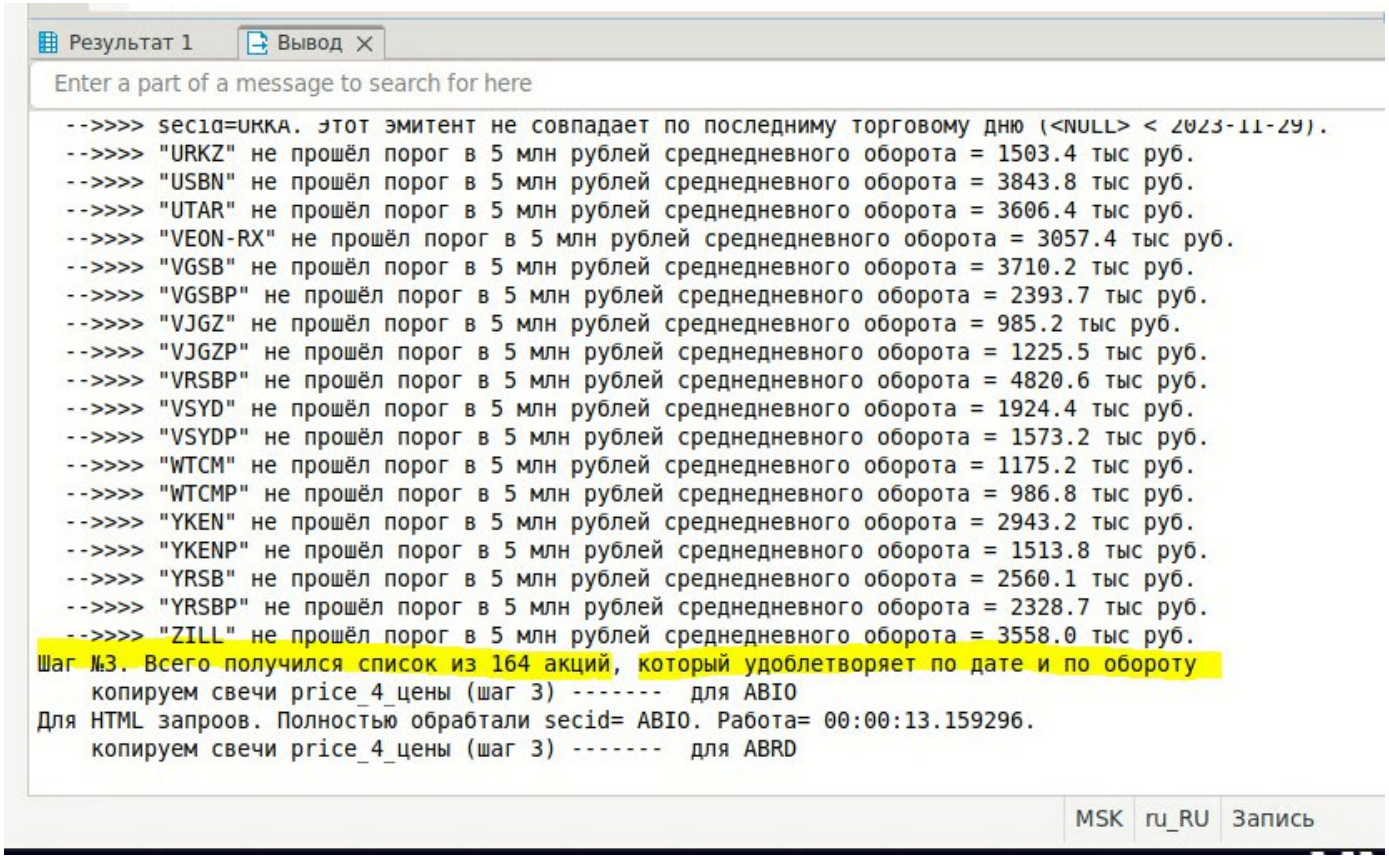


Для 12-ти потоков эта разница уже составляет более 155 раз (5.4 секунды и более 14 минут). 14*60/5.4 = 155

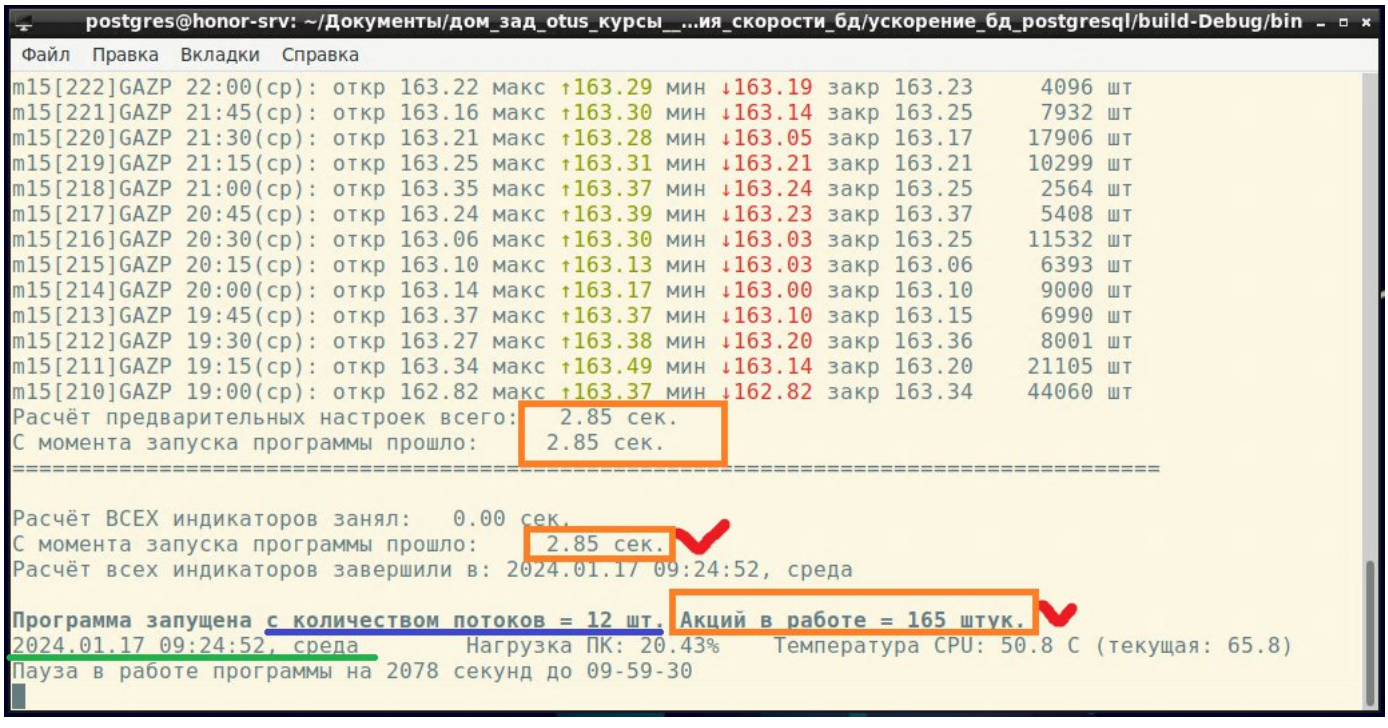
Более того, и что самое важное – код на Си можно еще ускорить. Так распараллеливание работы программы делалось только в одном месте, а расчеты всех индикаторов делались в однопоточном режиме. То есть можно смело еще на 0,5 секунды увеличить скорость работы (были тесты этой задачи в другом месте).

Итого – ЗАДАЧА №1 выполнена очень успешно. Реальное время сократилось до 7 секунд при старте программы вместо прежних 15 минут! Это явный прогресс.

Выборка количества акций при работе в PostgreSQL (164 акции):



Выборка количества акция при работе в Си программе (165 акций).



Здесь отличия могут быть из за самих фильтров – в БД он немного другой (более строгий).

Есть еще один блок программы – экстренный старт программы, когда котировки идут из разных мест (исторические котировки и котировки текущего/сегодняшнего) дня и приходится в аварийном порядке делать перерасчеты всех индикаторов с кучей выборок данных.

Этот кусок кода самый долгий по времени выполнения и PostgreSQL выполняет его около 40 минут при перезагрузке QUIK в 13 часов дня (по большей части – это именно проблема самого QUIK с получением и проверкой тиковых данных).

Если анализировать итоговые значения критического участка кода программы **«экстренного старта программы»,** то получаем следующие значения:

Потоков, шт	*Время работы, сек*	*Нагрузка CPU % по завершении работы*	*Температура CPU °C по завершении работы*
1	42,5	31,9	81,1
3	26,4	35,0	90,9
6	23,2	34,2	88,7
12	22,8	36,5	88,7

При том, что при работе в PostgreSQL разбор тех же данных занимал около 40 минут. Поэтому, раньше, сбой работы ПК приводил к невозможности торговать далее, т.к вторичные расчеты длились просто непомерно долго.

При работе в режиме **«экстренного старта программы»** происходит принудительное увеличение паузы между работой циклов программы (то есть программа принудительно начинает тормозить скорость работы), т.к ноутбук выходит за рамки штатного нагрева. Но через 10 секунд все приходит в норму, т.к программа прекращает первичное наполнение, а в штатном режиме нагрузка составляет около 2-х процентов и температура держится стабильно на уровне 42-46 градусов цельсия.

Итого – ЗАДАЧА №2 выполнена успешно.Ноутбук четко реагирует на перегрев и снижает на незначительное время нагрузку на процессор. То есть работать начинает дольше, но этот перегрев длится очень недолго, после чего ПК начинает работать в штатном режиме (без всяких ограничений).

Работа и нагрузка программы в штатном/торговом режиме.

В штатном/рабочем режиме в программе в конце «бесконечного» цикла стоит пауза равная «0,45-время цикла». Обычно цикл выполняется от 0.15 до 0.36 секунд днём и около 0,10-0,17 секунд в вечернюю сессию. То есть 165 акций с кучей ТаймФрэймов с кучей индикаторов (около 320 на 1 акцию) выполняется в бесконечном цикле не дольше 0, 45 секунд и при этом нагрузка на ПК находится на уровне простоя!

Итого – ЗАДАЧА №3 выполнена очень успешно. **Реальное время сократилось до 0,45 секунд при желаемом результате в 5 - 7 секунд.** То есть перевыполнение скорости работы программы в 10 раз. При этом есть еще задел на увеличение скорости работы программы еще на 10-15% путём доработки кода (распараллеливания некоторых блоков работы программы, которые очень хорошо поддаются разделению на потоки).

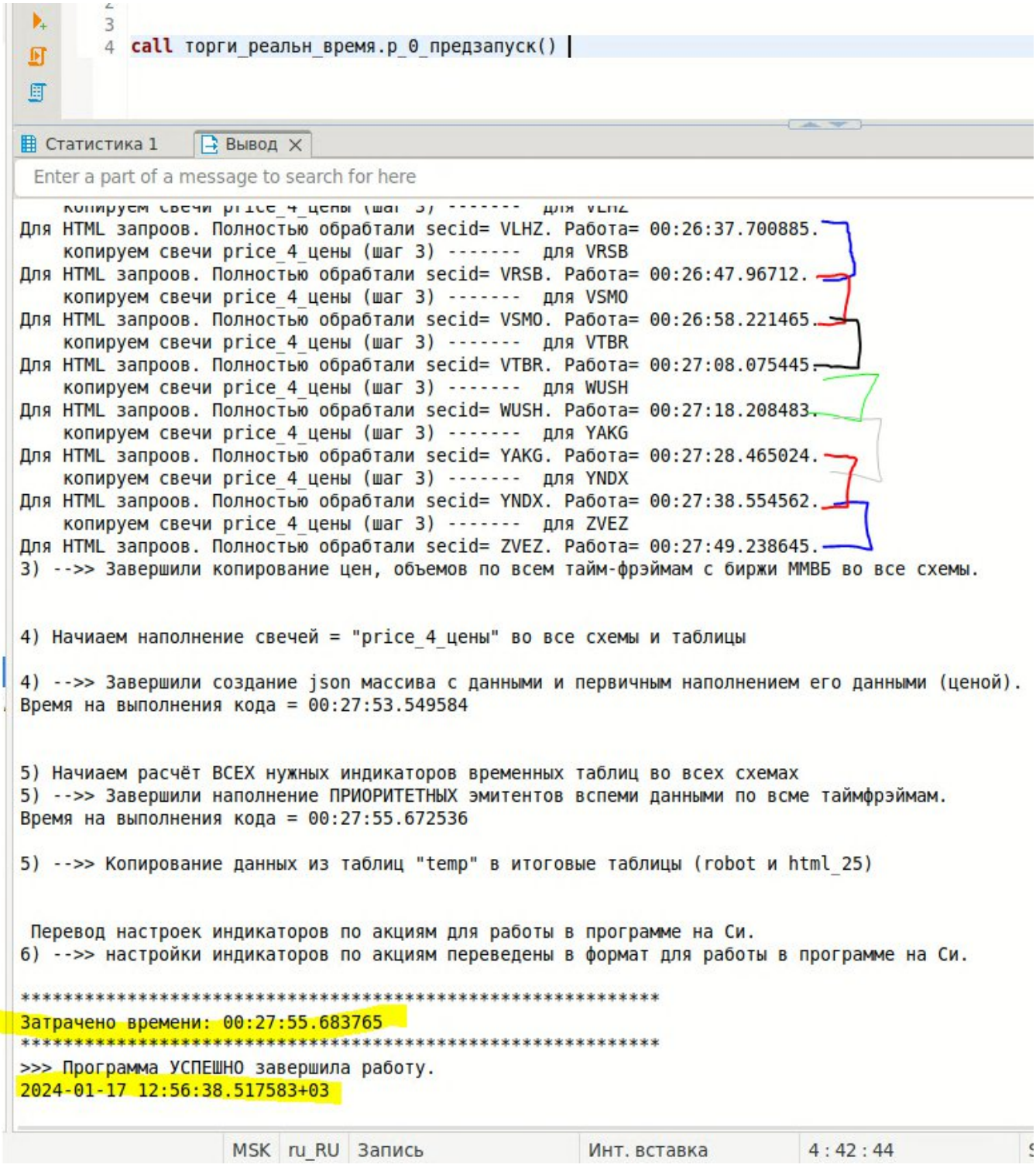
Но пока надобности в это нет, т.к даже цикл равный 0.45 секунд задействован не полностью и остается примерно 0.1-0.3 секунды на простой, что и позволяет использовать ПК практически в очень щадящем режиме.

Возникшие проблемы и способы их решений.

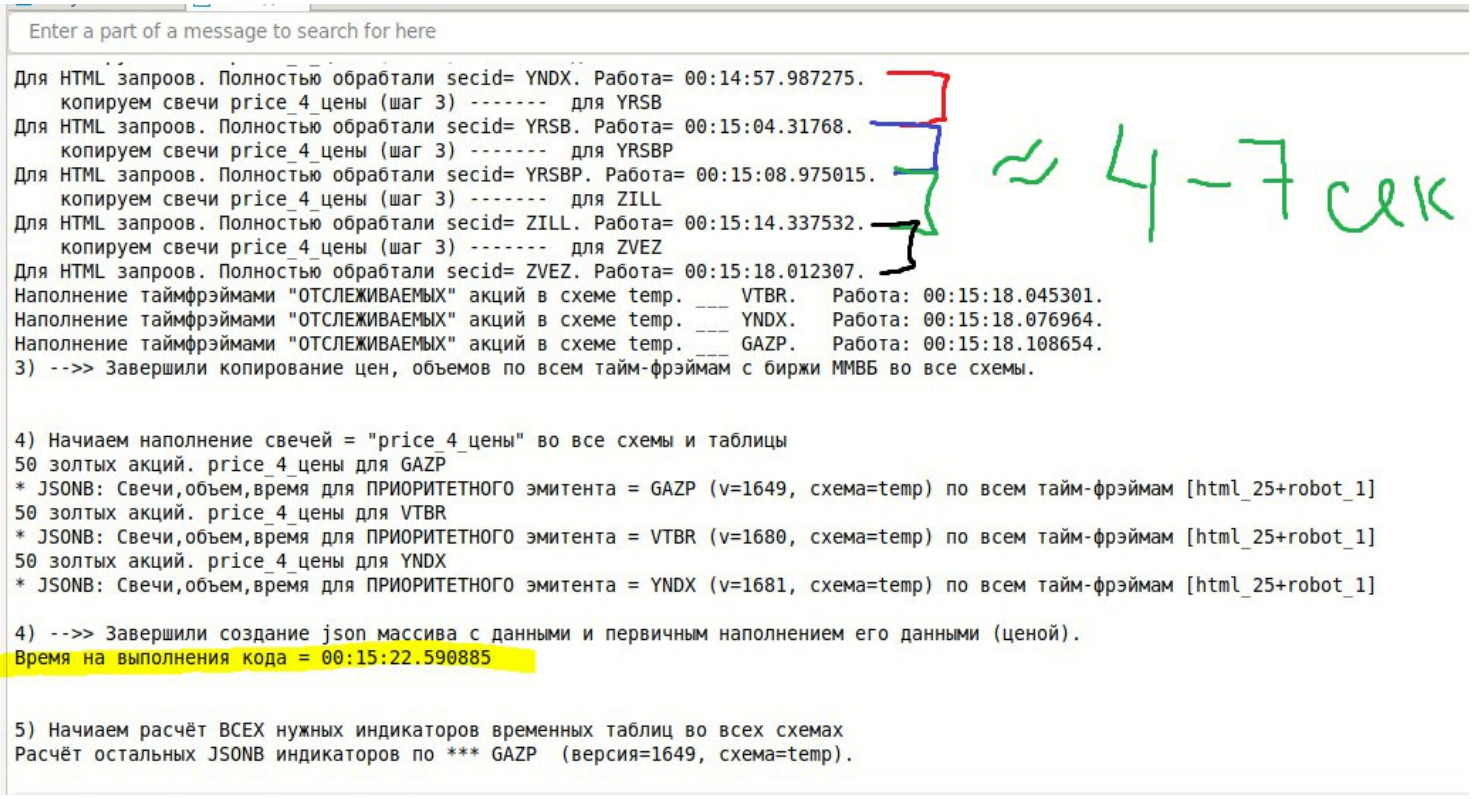
ПРОБЛЕМА №1.

При создании общедоступной версии БД, были удалены многие блоки из БД в расчётах. По идее, они никак не участвовали в расчетах, но по какой то причине сказались на быстродействии работы PostgreSQL. Ниже результаты рабочей и тестовой работы БД.

Скорость работы **тестовой версии БД** (стабильно 10 секунд на 1 акцию):



Скорость работы **рабочей версии** БД:



Во всех сравнениях будет учитываться время работы рабочей версии БД (**15 минут** работы).

ПРОБЛЕМА №2.

Охлаждение ноутбука - я специально искал ноутбук с очень хорошим охлаждением. Honor MateBook 16 в металлическом корпусе с хитрой системой охлаждения. Она реально очень быстро охлаждает центральный процессор (за 3-4 секунды), вот только защитить процессор от быстрого нагрева эта система не способна. Помогает защита от нагрева из первого проекта путём увеличения паузы в работе циклов программы.

ПРОБЛЕМА №3.

Работа с массивом указателей. При малонагруженном режиме – все ОК, но как только начинается высокочастотные запросы – то все начинает сыпаться.

глоб_об_таймфрэймы_описание.список_имен=(char *[]){"m1","m2","m3","m4","m5","m6","m10","m12","m15","m20","m30"}; не работает. Точнее какое то время работает, потом начинает сыпаться.

Вариант когда принудительно каждому элементу массива назначено своё имя исправило эту проблему. Позже я везде использовал только явные массивы с явными строковыми значениями (не указатель, а как массив из букв).

ПРОБЛЕМА №4.

Даты. В БД время (дата+время) идут как странно. Я 5 лет работал и все было нормально.

Но при работе в Си начались танцы с бубном. Нужно переводить во время как эпоха (количество секунд с 1970 года). В БД указатель стоит как TZ+0 то есть время Лондона, но фактически я вижу время Москвы. При переводе в эпоху и назад из эпохи время у меня становится на 3 часа больше. То есть было 23-49, а при возврате в БД время уже 02-49 (плюс 3 часа). При попытке работать в локальным временем – все расчеты увеличились на 7% примерно и все равно шли ошибки при расчетах из за пересечения этого самого времени ПОЛНОЧЬ.В общем – пока просто работаю как и работал, а при внесении времени из программы Си в БД – **от времени «эпоха» отнимаю 3 часа и тога время становится нормальным.** Это явный костыль - и там где у нас отличия всего в 2 часа уже пойдут проблемы (не критичные, т.к. торги начинаются в 10 утра, а будут начинаться в 9 утра и заканчиваться не в 23-50 а в 22-50), но пока ответа не нашел.

По то есть где то идет некорректный перевод времени. БД PostgreSQL божится что она работает со временем как TZ+0, но я вижу Московское время, Си по определению работает только с числами а не с временем, но при возврате того же самого времени из Си в БД время прирастает 3 часами .

ПРОБЛЕМА №5.

При работе с БД – на каждый запрос нужно соединение. Но это соединение есть не что иное как структура, внутрь которой пишутся результаты. Поэтому, в многопоточном режиме идут проблемы, т.к. разные потоки записывают разные данные внутрь одной структуры. Пришлось при многопоточном режиме создавать отдельные соединения.

На каждое новое соединение уходит 0,025 секунды. То есть это расточительно по ресурсам. Кроме того, при закрытии потока, нужно удалить соединение.

Была чехарда с тем, что в однопоточном режиме соединения не создавались а в многопоточном режиме не удалялись – и примерно на 100-ом соединении с БД вылизала ошибка – проблема с подсоединением к БД. Благо это предполагалось – просто пришлось в явно многопоточных блоках указать, что все соединения явно создаются заново и в конце функции удаляются явно.

ПРОБЛЕМА №6.

Самое сложное – наверное создать структуру заголовочных файлов (*.h) внутри проекта (реально тяжело укладывалась в голове структура подключений *.h файлов, особенно проблемно было при работе с глобальными структурами в заголовочных файлах – вечно какие то были нестыковки). Так же была проблема - подсоединить в CodeLite все

внешние библиотеки. Периодически даже встроенные библиотеки типа многопоточности отпадали. Пришлось все что хоть раз отваливалось как библиотека – принудительно добавлять к проекту. Благо, при выводе ошибки – указывался файл к этому самому исполняемому файлу и путь к нему.

Когда знаешь как это делать – то просто, но когда пути неизвестны, то отняло много времени.

ПРОБЛЕМА №7.

Работа с русскими названиями, особенно между кодом на Си и в БД. При переносе русского текста в БД были ошибки. Я решил работать с широкими символами wchar_t. Благо ввод вывод в терминале с ними уже делал. Переделал код (много где) и оказалось, что протокол соединения с БД PostgreSQL работает только с char* (массивом символов).

Это было очень болезненным ударом. То есть в функции для работы с БД прямо указано – что передаются только char* и никак иначе и в описании ошибки так же говорилось – что символы wchar_t неприменимы и нужны только обычные символы.

Благо был вариант «Б», который очень легко позволил обойти это ограничение. Просто массив символов увеличиваем в 2 раза. К примеру нужно переслать в БД текстовое описание для индикатора как «средняя». Раньше был массив char текс[8]. Где 7 букв и 1 ноль символ. **Теперь же указываю char текст[16]**, т.к программа все равно создаёт элементы кратные 8.

Теперь у нас 14 элементов приходится на русские буквы = «средняя» где на 1 русскую букву приходится по 2 символа в массиве и в конце ноль-символ. После этого проблемы с передачей русского текста в БД исчезли и внутри БД Постгрес появлялся русский корректный текст/описание.

Кстати, никаких сложностей при использовании русских переменных, русских названий файлов на жестких дисках, русских имен функций, русских имён структур при работе в Си – не наблюдалось (в CodeLite). При работе из Си программы/кода с БД PostgreSQL при работе с русскими именами схем, таблиц, столбцов – так же проблем не было.

А вот при работе из Си с названием самой БД на русском языке – были проблемы. Пришлось БД в PostgreSQL переименовать в латинские буквы.

Так же в самой ОС Debian при работе с systemctl демон-файлами так же были проблемы при работе с русскими буквами (но только в названии имени файлов).

В общем – особых неудобств с русскими буквами в коде Си при работе в ОС Debian 11 в CodeLite не обнаружены.

Указатели и сноски.

Примечание_1. **Нагрузка CPU.** Для серверов предельной нагрузкой считается значение в 80% после чего деградация скорости работы сервера резко возрастает и при этом рост идет нелинейный. Поэтому, одной из главных задач - держать уровень суммарной нагрузки в районе 50-70%, чтобы задействовать всю мощь ПК. После чего включается искусственное торможение скорости работы. При входе в стандартный режим, все ограничители плавно уменьшаются до базового состояния*. Как показала практика, в данном проекте, метод вполне хорошо себя зарекомендовал.*

Температура CPU. В виду того, что все расчеты происходят на ноутбуке, так же должен быть введён контроль температуры. При этом, при достижении критической температуры должна производиться запись в журнал учёта о текущей температуре процессора и нагрузке системы. Может так случиться, что суммарная нагрузка будет около 50% - 60%, а температура будет превышать критическую.

Ориентировочная температура *отсечения нагрузки около 70-92 градусов цельсия* (с учетом того, что почти все и всегда её занижают на 5-8 градусов). По документации, процессор AMD 5600H имеет потолок нагрева в 105 градусов. Так же цель этой задачи - посмотреть насколько эффективно охлаждение ноутбука и возможно (возможно) ноутбук придётся менять на что то более стойкое к нагрузкам и охлаждениям (в запасе имеется специализированная подставка с принудительным охлаждением ноутбука).

ПОДРОБНЕЕ: «Проект Нагрузка на ПК». Ознакомится и скачать можно из репозитория proekt_nagruzka_PC. *(https://github.com/OTUS-2023-C01-SARTAKOV-AP/proekt_nagruzka_PC).*

Примечание_2. «Проект ММВБ анализ». Ознакомится и скачать можно из репозитория proekt_mmvb_analiz. *(https://github.com/OTUS-2023-C01-SARTAKOV-AP/proekt_mmvb_analiz).*

На github.com выкладываются (>>>**перейти**):

- копия БД PostgreSQL (её урезанная, но полностью рабочая в данном проекте). **Суперпользователь postgres, пароль 123**
- КОД ПРОГРАММЫ НА СИ
- Описание проекта (текущая инструкция).

Система протестирована на тестовой копии БД. Использовалась ОС Debian 11 с русской локалью. Проблем с работой программы нет.