Московский авиационный институт
(Национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Курсовой проект
по курсу «Операционные системы»

Индексатор файлов с встроенным поиском и фильтрами

Студент: Саженов Константин Станиславович

Группа: 8О-208

Преподаватель: Е. С. Миронов

Дата:

Оценка:

Москва, 2020

1 Описание

Цель работы: применение практических навыков, полученных в течение прохождения курса. Проведение исследования в выбранной предметной области.
Задание: осуществить индексирование файлов и их расширений в базу данных, реализовать поиск по ней, а также подсчет статистики.

2 Задумка

Очень часто при поиске каких-либо файлов и/или программ, требуется пройти по всей файловой системе компьютера, что не всегда является простой и быстрой операцией. Для решения данной проблемы я решил реализовать свой собственный индексатор файлов, который будет в процессе индексирования собирать как можно больше информации о файле и сохранять её в какую-либо базу данных.
Я выбрал базу данных sqlite из-за своей простоты в использовании, поскольку она не требует дополнительной настройки СУБД и отлично подходит для таких standalone приложений, как мое.

3 Описание программы

3.1 Зависимости и основные моменты алгоритма

В данном проекте использовалась база данных sqlite версии 0.8.6, библиотека SOCI версии 4.0 для кроссплатформенной интеграции с базой данных sqlite и библиотека boost версии 1.75.0, а именно модуль program_options для обработки аргументов командной строки.
Также для сборки проекта понадобится cmake версии 3.17 или выше.

После запуска программа производит обработку аргументов, переданных ей и на их основе либо выдает какое-либо оповещение пользователю о некорректных данных или продолжает свою работу в зависимости от выбранного пользователем сценария.

3.2 Сборка

Для сборки необходимо перейти в директорию с проектом и запустить «cmake .». Необходимо, чтобы SOCI нужной версии был в той же директории, что и cmake(либо в любом указанном пути поиска библиотек cmake).

3.3 Алгоритм индексации

После проверки пути на существование программа запускает команду системный вызов ftw(от англ. «file tree walk» - «прогулка по дереву файлов») с тем, какую директорию следует обходить, функцией-обработчиком, которая должна будет вызываться каждый раз, как ftw найдет файл, который еще не обработала и третьим аргументом ftw принимает максимальное количество разрешенных файловых дескрипторов. Последний аргумент играет немаловажную роль в скорости программы,

поскольку при открытии файловых дескрипторов, ftw не будет закрывать их, чтобы не тратить лишнее время; и только по достижении максимального указанного количества будет закрывать их, чтобы открывать новые.
В функции-обработчике происходит, как ни странно, обработка переданного ей файла. Если функция получила директорию, то она просто завершается успешно. Если же ftw передала функции файл, то происходит индексация заданного файла в другой функции. В других случаях происходит логгирование полученного флага, который указывает на тип файла.
В функции, которая индексирует файл, происходит конвертация даты из формата языка C(timespec) в формат языка C++(time_point).

После всего вышеперечисленного начинается транзакция в sqlite, в которой происходит добавление или обновление полученных данных. Затем происходит подтверждение изменений и выход из функции.

После индексации всех известных файлов, следует удалить все файлы, которых не оказалось в файловой системе.

3.4 Алгоритм поиска

Алгоритм поиска происходит несколько проще.
После определения типа поиска, происходит select-запрос к базе данных, который возвращает все известные данные, которые удовлетворили запросу. На их основе происходит вывод данных на экран.

3.5 Подсчет статистики

Это самая простая часть программы, хоть самый сложный SQL-запрос находится именно здесь.
В данном случае происходит подсчет статистики прямо в SQL-запросе и просто происходит вывод полученных сведений на экран.

4 Примеры использования

Т.к. вся документация к командам встроена в программу, я не буду комментировать примеры. Чтобы понять, что и как необходимо запускать, следует запустить программу с ключом «--help»:

```
┌─sakost@sakost-pc ~/university/2 course/os/os-course-work/cmake-build-debug ‹master*›
└─$ ./os_course_work --help
Boost version: 1_75
Usage:
  os_course_work [options] command index-path ...

Allowed options:
  --help                        produce this help message
  -I [ --index-path ] arg (={empty list})
                                paths to index
  -d [ --index-database-file ] arg (=index.sqlite)
```

```
                                output database file to save indexes
 --command arg (=index)         command to execute(available: index,
                                search, stat)

┌─sakost@sakost-pc ~/university/2 course/os/os-course-work/cmake-build-debug ‹master*›
└─$ ./os_course_work -d ../home.sqlite --command stat
Total files count: 15908
Total extensions count: 144
Total files with extension ".py" are 4596
Total files with extension ".pyc" are 4198
Total files without extensions: 1605
Total files with extension ".pyi" are 1140
Total files with extension ".sip" are 748
Total files with extension ".qml" are 501
Total files with extension ".png" are 244
Total files with extension ".txt" are 224
Total files with extension ".so" are 223
Total files with extension ".qm" are 185
Biggest file with size 39941438B is /home/sakost/.cpan/Metadata
Mean file size is 44391.88B
┌─sakost@sakost-pc ~/university/2 course/os/os-course-work/cmake-build-debug ‹master*›
└─$ ./os_course_work -d ../cmake_build.sqlite --command index -I .
┌─sakost@sakost-pc ~/university/2 course/os/os-course-work/cmake-build-debug ‹master*›
└─$ ./os_course_work -d ../cmake_build.sqlite --command stat
Total files count: 38
Total extensions count: 16
Total files with extension ".cmake" are 8
Total files with extension ".txt" are 5
Total files with extension ".make" are 4
Total files without extensions: 3
Total files with extension ".log" are 3
Total files with extension ".sqlite" are 3
Total files with extension ".bin" are 2
Total files with extension ".out" are 2
Total files with extension ".cbp" are 1
Total files with extension ".cpp" are 1
Biggest file with size 2773832B is /home/sakost/university/2
course/os/os-course-work/cmake-build-debug/CMakeFiles/os_course_work.dir/main.cpp.o
Mean file size is 149711.82B
┌─sakost@sakost-pc ~/university/2 course/os/os-course-work/cmake-build-debug ‹master*›
└─$ ./os_course_work -d ../cmake_build2.sqlite --command index -I ..
zsh: correct '../cmake_build2.sqlite' to '../cmake_build.sqlite' [nyae]? n
┌─sakost@sakost-pc ~/university/2 course/os/os-course-work/cmake-build-debug ‹master*›
└─$ ./os_course_work -d ../cmake_build2.sqlite --command stat
Total files count: 842
Total extensions count: 51
Total files with extension ".cmake" are 122
Total files with extension ".cpp" are 113
Total files with extension ".o" are 107
Total files with extension ".h" are 76
Total files with extension ".make" are 62
Total files without extensions: 55
Total files with extension ".txt" are 50
Total files with extension ".md" are 44
Total files with extension ".sh" are 36
```

```
Total files with extension ".marks" are 17
Biggest file with size 3289736B is /home/sakost/university/2
course/os/os-course-work/build_soci/tests/mysql/CMakeFiles/soci_mysql_test_static.dir/test-mysql.cpp.o
Mean file size is 51286.65B
 ┌─sakost@sakost-pc ~/university/2 course/os/os-course-work/cmake-build-debug ‹master*›
 └─$ ./os_course_work -d ../cmake_build2.sqlite --command search
Invalid argument was passed
Boost version: 1_75
Usage:
  os_course_work [options] command index-path ...

Allowed options:
  --help                        produce this help message
  -I [ --index-path ] arg (={empty list})
                                paths to index
  -d [ --index-database-file ] arg (=index.sqlite)
                                output database file to save indexes
  --command arg (=index)              command to execute(available: index,
                                search, stat)

Search options:
  -b [ --by ] arg (=extension)        searching by some measure(allowed:
                                extension, directory)
  --target arg                        search statement

 ┌─sakost@sakost-pc ~/university/2 course/os/os-course-work/cmake-build-debug ‹master*›
 └─$ ./os_course_work -d ../cmake_build2.sqlite --command search --target .md
1 ↵
Invalid argument was passed
Boost version: 1_75
Usage:
  os_course_work [options] command index-path ...

Allowed options:
  --help                        produce this help message
  -I [ --index-path ] arg (={empty list})
                                paths to index
  -d [ --index-database-file ] arg (=index.sqlite)
                                output database file to save indexes
  --command arg (=index)              command to execute(available: index,
                                search, stat)

 ┌─sakost@sakost-pc ~/university/2 course/os/os-course-work/cmake-build-debug ‹master*›
 └─$ ./os_course_work -d ../cmake_build2.sqlite --command search --target=.md
Found file: /home/sakost/university/2 course/os/os-course-work/soci/www/doc/README.md
Found file: /home/sakost/university/2 course/os/os-course-work/soci/include/private/README.md Found file:
/home/sakost/university/2 course/os/os-course-work/soci/tests/README.md
Found file: /home/sakost/university/2 course/os/os-course-work/soci/RELEASING.md
Found file: /home/sakost/university/2 course/os/os-course-work/soci/README.md Found file:
/home/sakost/university/2 course/os/os-course-work/soci/docs/interfaces.md Found file:
/home/sakost/university/2 course/os/os-course-work/soci/docs/backends/postgresql.md Found file:
/home/sakost/university/2 course/os/os-course-work/soci/docs/backends/odbc.md Found file:
/home/sakost/university/2 course/os/os-course-work/soci/docs/backends/db2.md Found file:
/home/sakost/university/2 course/os/os-course-work/soci/docs/backends/firebird.md Found file:
/home/sakost/university/2 course/os/os-course-work/soci/docs/backends/oracle.md Found file:
```

/home/sakost/university/2 course/os/os-course-work/soci/docs/backends/mysql.md Found file: /home/sakost/university/2 course/os/os-course-work/soci/docs/backends/index.md Found file: /home/sakost/university/2 course/os/os-course-work/soci/docs/backends/sqlite3.md Found file: /home/sakost/university/2 course/os/os-course-work/soci/docs/api/client.md Found file: /home/sakost/university/2 course/os/os-course-work/soci/docs/api/backend.md Found file: /home/sakost/university/2 course/os/os-course-work/soci/docs/queries.md Found file: /home/sakost/university/2 course/os/os-course-work/soci/docs/procedures.md Found file: /home/sakost/university/2 course/os/os-course-work/soci/docs/installation.md Found file: /home/sakost/university/2 course/os/os-course-work/soci/docs/lobs.md Found file: /home/sakost/university/2 course/os/os-course-work/soci/docs/logging.md Found file: /home/sakost/university/2 course/os/os-course-work/soci/docs/quickstart.md Found file: /home/sakost/university/2 course/os/os-course-work/soci/docs/errors.md Found file: /home/sakost/university/2 course/os/os-course-work/soci/docs/indicators.md Found file: /home/sakost/university/2 course/os/os-course-work/soci/docs/binding.md Found file: /home/sakost/university/2 course/os/os-course-work/soci/docs/index.md Found file: /home/sakost/university/2 course/os/os-course-work/soci/docs/vagrant.md Found file: /home/sakost/university/2 course/os/os-course-work/soci/docs/structure.md Found file: /home/sakost/university/2 course/os/os-course-work/soci/docs/utilities.md Found file: /home/sakost/university/2 course/os/os-course-work/soci/docs/multithreading.md Found file: /home/sakost/university/2 course/os/os-course-work/soci/docs/types.md Found file: /home/sakost/university/2 course/os/os-course-work/soci/docs/languages/ada/reference.md Found file: /home/sakost/university/2 course/os/os-course-work/soci/docs/languages/ada/index.md Found file: /home/sakost/university/2 course/os/os-course-work/soci/docs/languages/ada/concepts.md Found file: /home/sakost/university/2 course/os/os-course-work/soci/docs/languages/ada/idioms.md Found file: /home/sakost/university/2 course/os/os-course-work/soci/docs/languages/index.md Found file: /home/sakost/university/2 course/os/os-course-work/soci/docs/faq.md Found file: /home/sakost/university/2 course/os/os-course-work/soci/docs/beyond.md Found file: /home/sakost/university/2 course/os/os-course-work/soci/docs/transactions.md Found file: /home/sakost/university/2 course/os/os-course-work/soci/docs/statements.md Found file: /home/sakost/university/2 course/os/os-course-work/soci/docs/boost.md Found file: /home/sakost/university/2 course/os/os-course-work/soci/docs/license.md Found file: /home/sakost/university/2 course/os/os-course-work/soci/docs/connections.md Found file: /home/sakost/university/2 course/os/os-course-work/README.md

```
┌─sakost@sakost-pc ~/university/2 course/os/os-course-work/cmake-build-debug ‹master*›
└─$
```

5 Исходный код

main.cpp:

```
#ifdef _WIN32
#error "Not for windows"
#endif

#include <iostream>
#include <utility>
#include <vector>
#include <string>
#include <filesystem>
#include <chrono>

#include <soci/boost-optional.h>
#include <soci/boost-tuple.h>
#include <boost/log/core.hpp>
#include <boost/log/trivial.hpp>
#include <boost/log/expressions.hpp>
#include <boost/log/utility/setup/file.hpp>
```

```
#include <boost/log/sources/severity_logger.hpp>
#include <boost/log/sources/record_ostream.hpp>

#include <boost/program_options.hpp>

#define SOCI_USE_BOOST 1
#include <soci/soci.h>
#include <soci/sqlite3/soci-sqlite3.h>

#include <ftw.h>
#include <sys/stat.h>
#include <unistd.h>
#include <libgen.h>

namespace po = boost::program_options;
namespace logging = boost::log;
namespace src = boost::log::sources;
namespace sinks = boost::log::sinks;
namespace keywords = boost::log::keywords;

using namespace soci;

namespace fs = std::filesystem;
using std::cin;
using std::cout;
using std::endl;
using std::string;
using std::vector;

const vector<string> create_database_queries = {"CREATE TABLE IF NOT EXISTS `extensions` ("
                                    "    `id` INTEGER PRIMARY KEY AUTOINCREMENT,"
                                    "    `extension` TEXT NOT NULL UNIQUE"
                                    ");",
                                    "CREATE TABLE IF NOT EXISTS `files`("
                                    "    `id` INTEGER PRIMARY KEY AUTOINCREMENT,"
                                    "    `directory` TEXT NOT NULL,"
                                    "    `filename` TEXT NOT NULL,"
                                    "    `file_size` INTEGER NOT NULL DEFAULT 0,"
                                    "    `last_changed_at` TEXT NOT NULL," // this is a datetime
                                    "    `extension_id` INTEGER NOT NULL,"
                                    "    foreign key (`extension_id`) REFERENCES `extensions`(`id`)"
                                    " ON DELETE CASCADE ON UPDATE CASCADE"
                                    ");"};

static const int max_file_descriptors_count = 256;
static const int stat_ext_sql_limit = 10;

struct command{
   explicit command(string  name): m_name(std::move(name)){}
   string m_name;
};

std::ostream& operator<<(std::ostream& out, const command& cmd){
```

```
    return (out << cmd.m_name);
}

void validate(boost::any& v,
              vector<string> const& values,
              command* /* target_type */,
              int){
    using namespace boost::program_options;
    validators::check_first_occurrence(v);

    string const& s = validators::get_single_string(values);

    if(s == "index" || s == "search" || s == "stat"){
        v = boost::any(command(s));
    } else{
        throw validation_error(validation_error::invalid_option_value);
    }
}



void init_logging() {
    logging::core::get()->set_filter(logging::trivial::severity > logging::trivial::info);
}

void create_database(session &sql) {
    for (const auto &query : create_database_queries) {
        BOOST_LOG_TRIVIAL(debug) << "beginning transaction";
        sql << "BEGIN TRANSACTION;";
        BOOST_LOG_TRIVIAL(trace) << "executing query: " << query;
        sql << query;
        BOOST_LOG_TRIVIAL(debug) << "committing transaction";
        sql << "COMMIT";
    }
}

bool path_exists(const string& str){
    struct stat s;
    int err = stat(str.c_str(), &s);
    if(-1 == err) {
        if(ENOENT == errno) {
            return false;
        } else {
            BOOST_LOG_TRIVIAL(fatal) << "fatal on \"path exists\"";
            exit(EXIT_FAILURE);
        }
    }
    return true;
}

void printUsage(const std::string &argv0, const po::positional_options_description& p,
                const po::options_description& desc)
{
    std::ostream &os = std::cout;
```

```
    os << "Boost version: " << BOOST_LIB_VERSION << endl;

    os << "Usage:" << std::endl;

    // print only basename of argv[0]
    boost::filesystem::path path(argv0);
    os << "  " << path.filename().string();

    os << " [options]";

    std::string last;
    int rep = 0;
    for(int i = 0; i < p.max_total_count(); i++)
    {
        const std::string &n = p.name_for_position(i);
        if(n == last)
        {
            if(!rep) os << " ...";
            if(rep++ > 1000) break;
        }
        else
        {
            os << " " << n;
            last = n;
            rep = 0;
        }
    }
    os << std::endl << std::endl;
    os << desc << std::endl;
}

session& get_sql_instance(){
    static session sql;
    BOOST_LOG_TRIVIAL(trace) << "get sql instance";
    return sql;
}

// Returns number of days since civil 1970-01-01.  Negative values indicate
//    days prior to 1970-01-01.
// Preconditions:  y-m-d represents a date in the civil (Gregorian) calendar
//                 m is in [1, 12]
//                 d is in [1, last_day_of_month(y, m)]
//                 y is "approximately" in
//                   [numeric_limits<Int>::min()/366, numeric_limits<Int>::max()/366]
//                 Exact range of validity is:
//                 [civil_from_days(numeric_limits<Int>::min()),
//                  civil_from_days(numeric_limits<Int>::max()-719468)]
template <class Int>
constexpr
Int
days_from_civil(Int y, unsigned m, unsigned d) noexcept
{
    static_assert(std::numeric_limits<unsigned>::digits >= 18,
```

```
            "This algorithm has not been ported to a 16 bit unsigned integer");
    static_assert(std::numeric_limits<Int>::digits >= 20,
            "This algorithm has not been ported to a 16 bit signed integer");
    y -= m <= 2;
    const Int era = (y >= 0 ? y : y-399) / 400;
    const unsigned yoe = static_cast<unsigned>(y - era * 400);      // [0, 399]
    const unsigned doy = (153*(m + (m > 2 ? -3 : 9)) + 2)/5 + d-1;  // [0, 365]
    const unsigned doe = yoe * 365 + yoe/4 - yoe/100 + doy;         // [0, 146096]
    return era * 146097 + static_cast<Int>(doe) - 719468;
}

// Returns year/month/day triple in civil calendar
// Preconditions:  z is number of days since 1970-01-01 and is in the range:
//                   [numeric_limits<Int>::min(), numeric_limits<Int>::max()-719468].
template <class Int>
constexpr
std::tuple<Int, unsigned, unsigned>
civil_from_days(Int z) noexcept
{
    static_assert(std::numeric_limits<unsigned>::digits >= 18,
            "This algorithm has not been ported to a 16 bit unsigned integer");
    static_assert(std::numeric_limits<Int>::digits >= 20,
            "This algorithm has not been ported to a 16 bit signed integer");
    z += 719468;
    const Int era = (z >= 0 ? z : z - 146096) / 146097;
    const auto doe = static_cast<unsigned>(z - era * 146097);          // [0, 146096]
    const unsigned yoe = (doe - doe/1460 + doe/36524 - doe/146096) / 365;  // [0, 399]
    const Int y = static_cast<Int>(yoe) + era * 400;
    const unsigned doy = doe - (365*yoe + yoe/4 - yoe/100);                // [0, 365]
    const unsigned mp = (5*doy + 2)/153;                                   // [0, 11]
    const unsigned d = doy - (153*mp+2)/5 + 1;                             // [1, 31]
    const unsigned m = mp + (mp < 10 ? 3 : -9);                            // [1, 12]
    return std::tuple<Int, unsigned, unsigned>(y + (m <= 2), m, d);
}

template <class Int>
constexpr
unsigned
weekday_from_days(Int z) noexcept
{
    return static_cast<unsigned>(z >= -4 ? (z+4) % 7 : (z+5) % 7 + 6);
}

template <class To, class Rep, class Period>
To
round_down(const std::chrono::duration<Rep, Period>& d)
{
    To t = std::chrono::duration_cast<To>(d);
    if (t > d)
        --t;
    return t;
}

template <class Duration>
```

```
std::tm
make_utc_tm(std::chrono::time_point<std::chrono::system_clock, Duration> tp)
{
    using namespace std;
    using namespace std::chrono;
    typedef duration<int, ratio_multiply<hours::period, ratio<24>>> days;
    // t is time duration since 1970-01-01
    Duration t = tp.time_since_epoch();
    // d is days since 1970-01-01
    days d = round_down<days>(t);
    // t is now time duration since midnight of day d
    t -= d;
    // break d down into year/month/day
    int year;
    unsigned month;
    unsigned day;
    std::tie(year, month, day) = civil_from_days(d.count());
    // start filling in the tm with calendar info
    std::tm tm = {0};
    tm.tm_year = year - 1900;
    tm.tm_mon = month - 1;
    tm.tm_mday = day;
    tm.tm_wday = weekday_from_days(d.count());
    tm.tm_yday = d.count() - days_from_civil(year, 1, 1);
    // Fill in the time
    tm.tm_hour = duration_cast<hours>(t).count();
    t -= hours(tm.tm_hour);
    tm.tm_min = duration_cast<minutes>(t).count();
    t -= minutes(tm.tm_min);
    tm.tm_sec = duration_cast<seconds>(t).count();
    return tm;
}

void index_file(const string& path, const struct stat* sb){
    session &sql = get_sql_instance();
    BOOST_LOG_TRIVIAL(debug) << "indexing file " << path;
    string base_name(basename((char*)path.c_str())), dir_name(dirname((char*)path.c_str()));
    off_t filesize = sb->st_size;
    timespec last_changed = sb->st_mtim;

    std::chrono::time_point<std::chrono::system_clock, std::chrono::nanoseconds> tp;
    {
        using namespace std::chrono;
        auto d = seconds{last_changed.tv_sec} + nanoseconds{last_changed.tv_nsec};
        decltype(tp) temp_tp{duration_cast<system_clock::duration>(d)};
        tp = temp_tp;
    }
    std::tm last_changed_tm = make_utc_tm(tp);

    try {
        BOOST_LOG_TRIVIAL(debug) << "beginning transaction";
        sql << "BEGIN TRANSACTION;";

        BOOST_LOG_TRIVIAL(debug) << "selecting file";
```

```
        rowset<row> rs = (sql.prepare << "SELECT id FROM `files` WHERE filename = :filename AND directory
= :directory;",
                use( base_name, "filename"), use(dir_name, "directory"));
        const auto& iter_rs = rs.begin();
        if(iter_rs == rs.end()){
            BOOST_LOG_TRIVIAL(debug) << "file not found";
            const auto cxx_path = fs::path(path);
            string ext = cxx_path.has_extension() ? cxx_path.extension() : "";

            // find out extension id
            BOOST_LOG_TRIVIAL(debug) << "finding extension id";
            boost::optional<int> extension_id;
            BOOST_LOG_TRIVIAL(trace) << "selecting extension from db";
            sql << "SELECT id FROM `extensions` WHERE extension=:ext ;", into(extension_id), use( ext, "ext");
            if(!extension_id.is_initialized()){
                BOOST_LOG_TRIVIAL(trace) << "extension not found. creating new extension";
                sql << "INSERT INTO `extensions`(extension) VALUES (:ext);", use( ext);
                BOOST_LOG_TRIVIAL(debug) << "committing";
                sql << "COMMIT;";
                BOOST_LOG_TRIVIAL(debug) << "beginning transaction";
                sql << "BEGIN TRANSACTION;";
                BOOST_LOG_TRIVIAL(trace) << "selecting extension from db";
                sql << "SELECT id FROM `extensions` WHERE extension=:ext ;", use( ext), into(extension_id);
            }
            assert(extension_id.is_initialized() && "Something went wrong with sql logic");

            // insert file
            BOOST_LOG_TRIVIAL(info) << "inserting file into database";
            sql << "INSERT INTO `files`(directory, filename, file_size, last_changed_at, extension_id) "
                   "VALUES (?, ?, ?, ?, ?);", use(dir_name), use(base_name), use(filesize), use(last_changed_tm),
use(extension_id);
        } else{
            int file_id = iter_rs->get<int>(0); // id
            BOOST_LOG_TRIVIAL(debug) << "found current file with id " << file_id << " in database";
            BOOST_LOG_TRIVIAL(debug) << "updating database info";
            sql << "UPDATE `files` SET "
                   "file_size = :size ,"
                   "last_changed_at = :changed "
                   "WHERE id = :id ;", use(filesize, "size"),
                   use(last_changed_tm, "changed"),
                   use(file_id, "id");
        }

        BOOST_LOG_TRIVIAL(debug) << "committing";
        sql << "COMMIT;";
    } catch (std::exception& err){
        BOOST_LOG_TRIVIAL(info) << "rolling back";
        sql << "ROLLBACK;";
        BOOST_LOG_TRIVIAL(error) << "Error occurred: " << err.what();
    }

}
```

```
//FTW_F - file
//FTW_D - dir
//FTW_DNR - dir without permissions to read
//FTW_NS - stat call failed

int handle_walk(const char* fpath, const struct stat* sb, int typeflag){
    if(typeflag == FTW_D){
        BOOST_LOG_TRIVIAL(trace) << "got directory: " << fpath;
        return 0;
    }
    if(typeflag == FTW_F) {
        BOOST_LOG_TRIVIAL(trace) << "got file: " << fpath;
        string path = fpath;
        index_file(path, sb);
    }
    else{
        BOOST_LOG_TRIVIAL(error) << "got unexpected flag: " << typeflag;
    }
    return 0;
}

void index_files(const po::variables_map& vm){
    const auto &paths = vm["index-path"].as<vector<string>>();
    for (const auto &path : paths) {
        char *rpath = new char[PATH_MAX];
        if(realpath(path.c_str(), rpath) == nullptr){
            BOOST_LOG_TRIVIAL(error) << "invalid path: " << path.c_str();
            delete[] rpath;
            continue;
        }
        string real_path = rpath;
        BOOST_LOG_TRIVIAL(debug) << "indexing path: " << real_path;
        ftw(real_path.c_str(), handle_walk, max_file_descriptors_count);
    }

    // filter deleted files
    session &sql = get_sql_instance();
    try{
        rowset<row> rs = (sql.prepare << "SELECT * FROM `files`;");
        BOOST_LOG_TRIVIAL(debug) << "beginning transaction";
        sql << "BEGIN TRANSACTION;";
        for (const auto &item : rs) {
            const auto &full_path = item.get<string>(1) + "/" + item.get<string>(2);
            if (!path_exists(full_path)) {
                BOOST_LOG_TRIVIAL(info) << "deleting path: " << std::quoted(full_path);
                sql << "DELETE FROM `files` WHERE id=:id ;", use(item.get<int>(0), "id");
            }
        }
        BOOST_LOG_TRIVIAL(debug) << "committing transaction";
        sql << "COMMIT";
    } catch (const std::exception& err){
        BOOST_LOG_TRIVIAL(info) << "rolling back";
        sql << "ROLLBACK;";
        BOOST_LOG_TRIVIAL(error) << "error while walking through all rows: " << err.what();
    }
```

```
}

void search_files(const po::variables_map& vm){
    session& sql = get_sql_instance();

    const auto &search_by = vm["by"].as<string>();
    if(search_by == "extension"){
        string ext = vm["target"].as<string>();
        rowset<row> rs = (sql.prepare << "SELECT f.directory,f.filename FROM files f "
                                         "INNER JOIN extensions e on e.id = f.extension_id WHERE e.extension = :ext ;",
                                         use(ext, "ext"));
        for (const auto &file : rs) {
            cout << "Found file: " << file.get<string>(0) << "/" << file.get<string>(1);
        }
    } else if(search_by == "directory"){
        string dir = vm["target"].as<string>();
        char *rpath = new char[PATH_MAX];
        if(realpath(dir.c_str(), rpath) == nullptr){
            BOOST_LOG_TRIVIAL(fatal) << "invalid dir: " << dir.c_str();
            delete[] rpath;
            return;
        }
        dir = rpath;

        rowset<row> rs = (sql.prepare << "SELECT f.directory,f.filename FROM files f WHERE f.directory = :dir",
use(dir, "dir"));
        for (const auto &file : rs) {
            cout << "Found file: " << file.get<string>(0) << "/" << file.get<string>(1);
        }
    }
    else{
        BOOST_LOG_TRIVIAL(info) << "got invalid argument";
        cout << "Invalid argument: " << search_by << endl;
        return;
    }
}

void print_stat(std::ostream& out, const po::variables_map& vm){
    session &sql = get_sql_instance();
    int files_count;
    sql << "SELECT COUNT(*) FROM `files`;", into(files_count);
    out << "Total files count: " << files_count << endl;

    int total_ext_count;
    sql << "SELECT COUNT(*) FROM `extensions`;", into(total_ext_count);
    out << "Total extensions count: " << total_ext_count << endl;

    rowset<row> rs = (sql.prepare << "SELECT e.extension, COUNT(f.extension_id) AS files_count "
                                     "FROM extensions e "
                                     "LEFT JOIN files f ON e.id = f.extension_id GROUP BY e.id "
                                     "ORDER BY files_count DESC "
                                     "LIMIT :limit ;",
                                     use(stat_ext_sql_limit, "limit"));
    for (const auto &extension : rs) {
```

```
        auto ext = extension.get<string>(0);
        int ext_count = std::stoi(extension.get<string>(1));
        if(ext.empty()) {
            out << "Total files without extensions: " << ext_count << endl;
        }
        else{
            out << "Total files with extension " << std::quoted(ext) << " are " << ext_count << endl;
        }
    }
    row biggest_file;
    sql << "SELECT MAX(file_size) AS max_size,directory,filename FROM files;", into(biggest_file);

    out << "Biggest file with size " << biggest_file.get<string>(0) << "B is " << biggest_file.get<string>(1) << "/" <<
biggest_file.get<string>(2) << endl;

    string avg_file_size;
    sql << "SELECT PRINTF(\"%.2f\", AVG(file_size)) AS mean_size FROM files;", into(avg_file_size);
    out << "Mean file size is " << avg_file_size << "B" << endl;
}



int main(int argc, char **argv) {
    init_logging();

    // parsing arguments
    po::options_description desc("Allowed options");
    desc.add_options()
            ("help", "produce this help message")
            ("index-path,I", po::value<vector<string>>()->default_value(vector<string>(), "{empty list}"), "paths to
index")
            ("index-database-file,d", po::value<string>()->default_value("index.sqlite"),
             "output database file to save indexes")
            ("command", po::value<command>()->default_value(command("index")),
             "command to execute(available: index, search, stat)");

    po::positional_options_description p;
    p.add("command", 1);
    p.add("index-path", -1);

    po::options_description search_desc("Search options");
    search_desc.add_options()
            ("by,b", po::value<string>()->default_value("extension"), "searching by some measure"
                                                      "(allowed: extension, directory)")
            ("target", po::value<string>()->required(), "search statement");



    po::variables_map vm;
    try {
        po::store(po::command_line_parser(argc, argv).allow_unregistered().options(desc).positional(p).run(), vm);
        po::notify(vm);
    } catch (...) {
        cout << "Invalid argument was passed" << endl;
```

```
    printUsage(argv[0], p, desc);
    return EXIT_FAILURE;
}

for (const auto &path : vm["index-path"].as<vector<string>>()) {
    if(!path_exists(path)){
        cout << "This path does not exist: " << std::quoted(path) << endl;
        return EXIT_FAILURE;
    }
}

if (vm.count("help")) {
    BOOST_LOG_TRIVIAL(trace) << "got help command";
    printUsage(argv[0], p, desc);

    return EXIT_SUCCESS;
}



session &sql = get_sql_instance();
const auto &database_url = vm["index-database-file"].as<string>();

bool database_exists = path_exists(database_url);

try {
    BOOST_LOG_TRIVIAL(info) << "opening connection to database on url: " << database_url;
    sql.open(sqlite3, database_url);
} catch (const std::exception &err) {
    BOOST_LOG_TRIVIAL(fatal) << "error while connecting to database: " << err.what() << endl;
    return EXIT_FAILURE;
}

if (!database_exists) {
    BOOST_LOG_TRIVIAL(info) << "creating database schema";
    try {
        create_database(sql);
    } catch (const std::exception &err) {
        BOOST_LOG_TRIVIAL(info) << "rolling back";
        sql << "ROLLBACK;";
        BOOST_LOG_TRIVIAL(fatal) << "Error while creating database(tables): " << err.what();
        return EXIT_FAILURE;
    }
}



string cmd = vm["command"].as<command>().m_name;

if (cmd == "index") {
    index_files(vm);
} else if (cmd == "search") {
    try{
        desc.add(search_desc);
        po::store(po::command_line_parser(argc, argv).options(desc).run(), vm);
```

```
            po::notify(vm);
        } catch(...){
            cout << "Invalid argument was passed" << endl;
            printUsage(argv[0], p, desc);

            return EXIT_FAILURE;
        }
        search_files(vm);
    } else if (cmd == "stat") {
        print_stat(cout, vm);
    }
}
```

CMakeLists.txt:

```
cmake_minimum_required(VERSION 3.17)
project(os_course_work)

set(CMAKE_CXX_STANDARD 17)

set(SOCI_SOURCE_DIR "${PROJECT_SOURCE_DIR}/soci")
set(CMAKE_MODULE_PATH ${SOCI_SOURCE_DIR}/cmake ${CMAKE_MODULE_PATH})
set(CMAKE_MODULE_PATH ${SOCI_SOURCE_DIR}/cmake/modules ${CMAKE_MODULE_PATH})
find_package(Soci REQUIRED)
list(APPEND LIBS ${SOCI_LIBRARY} ${SOCI_sqlite3_PLUGIN})

find_package(Boost 1.75 COMPONENTS program_options log REQUIRED)
include_directories( ${Boost_INCLUDE_DIR} )
list(APPEND LIBS ${Boost_LIBRARIES})

add_executable(${PROJECT_NAME} main.cpp helpers.h)

target_link_libraries(${PROJECT_NAME} PRIVATE ${LIBS})
```

6 Заключение

Проделав данный проект, я заключил, что даже самые, казалось бы, сложные программы, могут быть реализованы(с минимальным функционалом) в несколько сотен строк кода.

В результате я получил конечную программу, которую можно использовать в качестве небольшой утилиты для некоторых задач. Также я организовал структуру программного кода так, чтобы его можно было легко дополнять и совершенствовать новыми командами и способами поиска.

В данной работе я познакомился со многими видами запросов к БД sqlite, которая оказалась крайне востребованной в мире desktop и standalone приложений. Также я заключил, что данная БД является одной из самых простых, что несомненно сказывается на скорости разработки и развертывании приложении в «боевых условиях».

В том числе я познакомился с такой полезной функцией UNIX, как ftw(от англ. file tree walk), которая позволяет пройтись по всему дереву файлов данной ФС и обработать их индивидуально.

В дополнении ко всему, я научился правильно обрабатывать аргументы командной строки, которые указывает пользователь, с помощью библиотеки program_options из набора библиотек boost, что, я полагаю, является не самой простой задачей из-за многофункциональности данного набора.
Считаю, что проделанная работа хорошо закрепила мои навыки в создании программ под ОС и подарила, в том числе, и новые, которые я получил в процессе написания взаимодействия программы с файловой системой ОС.