ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНТЕЛЕКТУАЛЬНИХ ТЕХНОЛОГІЙ І ЗВ'ЯЗКУ

Звіт
з дисципліни Технологіх Розподілених Систем та Паралельних Обчислень

Лабораторна робота №1

Виконав: студент 3 курсу, групи ІПЗ-4.04
спеціальності
121 Інженерія програмного забезпечення

_____Бухта М.М.
Перевірив_____Развалінов В.Ю.

Одеса 2024

ЗАВДАННЯ

Опис завдання:

1. Створіть клас, метод main() якого виводить на консоль слова речення «We have the java learning course!» із затримкою 1 секунда.
2. Створіть два потоки, один з яких виводить на консоль символ ‘-‘, а інший – символ ‘|’. Запустіть потоки в основній програмі так, щоб вони по черзі виводили свої символи в рядок. Виведіть на консоль 10 таких рядків.
3. Створіть клас Counter з методами increment() та decrement(), які збільшують та зменшують значення лічильника відповідно. Створіть два потоки, один з яких збільшує 1000 разів значення лічильника, а інший – зменшує 1000 разів значення лічильника. Використовуючи методи sleep() та/або join() добийтесь правильної роботи лічильника при одночасній роботі з ним двох потоків.
4. Реалізуйте стрічковий алгоритм або алгоритм Фокса множення матриць. Результат множення записуйте в об’єкт класу Result. Виконайте експерименти з різною кількістю потоків та різною розмірністю матриць, які перемножуються, реєструючи час виконання. Побудуйте графіки відповідних залежностей.

Код програми:

main.cpp

```
#include "Core/Starter.hpp"


int main(int argc, char *argv[]) {
   auto app = Core::Starter::create(argc, argv);


   return app->main();
}
```

Core/Starter.hpp

```
#ifndef __LAB1_SOURCEFILES_CORE_STARTER_HPP__
```

```
#define __LAB1_SOURCEFILES_CORE_STARTER_HPP__


#include <QObject>
#include <QSharedPointer>
#include <QAtomicPointer>
#include <QCoreApplication>


namespace Core {


class Starter : public QObject {
    Q_OBJECT


public:
    static QSharedPointer<Starter> create(int argc, char *argv[], QObject* parent =
nullptr);



    int main();


protected:
    virtual int executeApp();


protected:
    Starter() = delete;
    Starter(int argc, char *argv[], QObject* parent = nullptr);
    Q_DISABLE_COPY(Starter);
```

```
    void init_connections();

protected:
    QCoreApplication _qt_app;

signals:
    void finish();

private:
    int _argc;
    char ** _argv;
};

} // namespace Core

#endif // __LAB1_SOURCEFILES_CORE_STARTER_HPP__
```

Core/Starter.cpp

```
#include "Starter.hpp"
#include "Launcher/LaboratoryStarter.hpp"

namespace Core {

QSharedPointer<Starter> Starter::create(int argc, char *argv[], QObject* parent) {
    static QSharedPointer<Starter> instance(new Launcher::Starter(argc, argv, parent));
```

```
    return instance;
}


int Starter::main() {
    int ret = executeApp();


    return ret;
}


int Starter::executeApp() {
    return _qt_app.exec();
}


Starter::Starter(int argc, char *argv[], QObject* parent)
: QObject{parent}
, _argc{argc}
, _argv{argv}
, _qt_app{argc, argv} {
    init_connections();
}


void Starter::init_connections() {
    connect(this, &Starter::finish, &_qt_app, &QCoreApplication::quit);
}


} // namespace Core
```

Core/BaseInfoDisplayer.hpp

```
#ifndef __LAB1_SOURCEFILES_CORE_BASEINFODISPLAYER_HPP__
#define __LAB1_SOURCEFILES_CORE_BASEINFODISPLAYER_HPP__

#include <QRunnable>
#include <QString>

namespace Core {

class BaseInfoDisplayer : public QRunnable {
public:
    BaseInfoDisplayer(void);

    void run() override;

private:
    QString _msg;
};

} // Core

#endif // __LAB1_SOURCEFILES_CORE_BASEINFODISPLAYER_HPP__
```

Core/BaseInfoDisplayer.cpp

```
#include "BaseInfoDisplayer.hpp"
```

```
#include <QDebug>

#include <QThread>



#include <iostream>



namespace Core {



BaseInfoDisplayer::BaseInfoDisplayer()

: _msg{"Лабораторна робота №1\n3 дисципліни Технології Розподілених Систем та Паралельних Обчислень\nСтудента групи ІПЗ-3.04\nБухти Микити"}

{



}



void BaseInfoDisplayer::run() {

    for (qint64 i{0}; i < _msg.size(); ++i) {

        std::wcout << static_cast<wchar_t>(_msg[i].unicode()) << std::flush;

        QThread::msleep(50);

    }

    std::wcout << std::endl;

}



} // Core
```

Launcher/LaboratoryStarter.hpp

```
#ifndef __LAB1_LAUNCHER_LABORATORYSTARTER_HPP__
```

```cpp
#define __LAB1_LAUNCHER_LABORATORYSTARTER_HPP__

#include "Core/Starter.hpp"

#include <QThreadPool>

namespace Launcher {

class Starter final : public Core::Starter {
public:
    Starter(int argc, char *argv[], QObject* parent = nullptr);
    Q_DISABLE_COPY(Starter);

private:
    int executeApp() override;

private:
    QThreadPool* _pool;
};

} // Launcher

#endif // __LAB1_LAUNCHER_LABORATORYSTARTER_HPP__
```

Launcher/LaboratoryStarter.cpp

```cpp
#include "LaboratoryStarter.hpp"
```

```cpp
#include "Core/BaseInfoDisplayer.hpp"

#include "Tasks/Task1.hpp"
#include "Tasks/Task2.hpp"
#include "Tasks/Task3.hpp"
#include "Tasks/Task4.hpp"

#include <QCoreApplication>
#include <QDebug>
#include <QTimer>

namespace Launcher {

Starter::Starter(int argc, char *argv[], QObject* parent)
: Core::Starter(argc, argv, parent) {
    _pool = QThreadPool::globalInstance();
    _pool->setMaxThreadCount(1);
}

int Starter::executeApp() {
    _pool->start(new Core::BaseInfoDisplayer());
    _pool->start(new Tasks::Task1);
    _pool->start(new Tasks::Task2);
    _pool->start(new Tasks::Task3);
    _pool->start(new Tasks::Task4);
```

```
    _pool->waitForDone();

    QTimer::singleShot(10, [this]() {
        emit this->finish();
    });

    return Core::Starter::executeApp();
}

} // Launcher
```

Tasks/Task1.hpp

```
#ifndef __LAB1_TASKS_TASK1_HPP__
#define __LAB1_TASKS_TASK1_HPP__

#include <QRunnable>

namespace Tasks {

class Task1 : public QRunnable {
public:
    void run() override;
};

} // Tasks
```

```
#endif // __LAB1_TASKS_TASK1_HPP__
```

Tasks/Task1.cpp

```
#include "Task1.hpp"


#include <QDebug>
#include <QThread>


#include <string>
#include <iostream>


namespace Tasks {


void Task1::run() {
    qDebug() << __PRETTY_FUNCTION__ << "called";
    std::wstringstream str{L"We have the Java learning course!"};
    std::wstring word;


    while(std::getline(str, word, L' ')) {
        std::wcout << word << L' ' << std::flush;
        QThread::msleep(1000);
    }
    std::wcout << std::endl;
}
```

```
} // Tasks
```

Tasks/Task2.hpp

```
#ifndef __LAB2_TASKS_TASK2_HPP__
#define __LAB2_TASKS_TASK2_HPP__

#include <QRunnable>
#include <QObject>
#include <QWaitCondition>
#include <QMutex>

namespace Tasks {

class Task2 : public QRunnable {
private:
    enum class TurnEnumeration : quint8 {
        Dash = 0,
        Pipe = 1
    };

public:
    void run() override;

    virtual ~Task2() = default;

private:
```

```
};


class ConsoleSyncWriter : public QObject {
public:
    ConsoleSyncWriter(const std::wstring& str, const qint32 turn_number);


public:
    static qint32 get_terminal_width();


public slots:
    void write(QAtomicInt& current_turn, qint32 next_turn);
    void write_repeatly(QAtomicInt& current_turn, qint32 next_turn, qint32
repeat_count);


private:
    std::wstring _str;
    const qint32 _turn_number;


    static QWaitCondition _waiter;
    static QMutex _write_mutex;
};


} // namespace Tasks


#endif // __LAB2_TASKS_TASK2_HPP__
```

Tasks/Task2.cpp

```
#include "Task2.hpp"


#include <QDebug>
#include <QThread>


#include <iostream>
#include <string>


#include <sys/ioctl.h>


namespace Tasks {


QWaitCondition ConsoleSyncWriter::_waiter;
QMutex ConsoleSyncWriter::_write_mutex;


void Task2::run() {
    qDebug() << __PRETTY_FUNCTION__ << "called";

    auto console_writer = [](const std::wstring& str)->void {
        std::wcout << str;
    };


    QThread dash_th, pipe_th;
    ConsoleSyncWriter dash_writer{L"-", static_cast<qint32>(TurnEnumeration::Dash)};
```

```
    ConsoleSyncWriter pipe_writer{L"|", static_cast<qint32>(TurnEnumeration::Pipe)};

    qint8 rows_count = 10;

    qint8 colums_count = ConsoleSyncWriter::get_terminal_width();

    QAtomicInt current_turn = static_cast<qint32>(TurnEnumeration::Dash);



    dash_writer.moveToThread(&dash_th);

    pipe_writer.moveToThread(&pipe_th);



    QObject::connect(&dash_th, &QThread::started, [&](){

        dash_writer.write_repeatly(current_turn, static_cast<qint32>(TurnEnumera-
tion::Pipe), rows_count * colums_count / 2);

        dash_th.quit(); // in case if quit isn't used, the main thread is locked on the wait()
method called (str 46);

    });

    QObject::connect(&pipe_th, &QThread::started, [&](){

        pipe_writer.write_repeatly(current_turn, static_cast<qint32>(TurnEnumera-
tion::Dash), rows_count * colums_count / 2);

        pipe_th.quit();

    });



    dash_th.start();

    pipe_th.start();


    // There is a framework bug?

    dash_th.wait();

    pipe_th.wait();

}
```

```
ConsoleSyncWriter::ConsoleSyncWriter(const std::wstring& str, const qint32 turn_number)
: _str{str}
, _turn_number{turn_number} {

}

qint32 ConsoleSyncWriter::get_terminal_width() {
    struct winsize w;
    ioctl(STDOUT_FILENO, TIOCGWINSZ, &w);
    return w.ws_col;
}

void ConsoleSyncWriter::write(QAtomicInt& current_turn, qint32 next_turn) {
    QMutexLocker locker(&_write_mutex);

    while (current_turn.loadAcquire() != _turn_number) {
        _waiter.wait(&_write_mutex);
    }

    std::wcout << _str << std::flush;

    current_turn.storeRelease(next_turn);
    _waiter.wakeOne();
}
```

```
void ConsoleSyncWriter::write_repeatly(QAtomicInt& current_turn, qint32 next_turn,
qint32 repeat_count) {
    for (qint32 i{0}; i < repeat_count; ++i) {
        write(current_turn, next_turn);
    }

    qDebug() << __PRETTY_FUNCTION__ << "finished";
}

} // Tasks
```

Tasks/Task3.hpp

```
#ifndef __LAB3_TASKS_TASK1_HPP__
#define __LAB3_TASKS_TASK1_HPP__

#include <QRunnable>
#include <QAtomicInt>

namespace Tasks {

class Task3 : public QRunnable {
public:
    void run() override;
};

class Counter {
public:
```

```
    Counter();
    void increment();
    void decrement();

    qint32 get_counter() const noexcept;

private:
    QAtomicInt _counter;
};

} // Tasks

#endif // __LAB3_TASKS_TASK1_HPP__
```

Tasks/Task3.cpp

```
#include "Task3.hpp"

#include <QDebug>
#include <QMetaObject>
#include <QtConcurrent/QtConcurrent>
#include <QThread>

#include <thread>

namespace Tasks {
```

```
void Task3::run() {
    qDebug() << __PRETTY_FUNCTION__ << "called";


    QThread increment_th, decrement_th;
    Counter counter;
    qint32 loop_count{1000};


    QObject::connect(&increment_th, &QThread::started, [&counter, loop_count, &incre-
ment_th](){
        for (qint32 i{0}; i < loop_count; ++i) {
            counter.increment();
            increment_th.quit();
        }
    });
    QObject::connect(&decrement_th, &QThread::started, [&counter, loop_count, &decre-
ment_th](){
        for (qint32 i{0}; i < loop_count; ++i) {
            counter.decrement();
            decrement_th.quit();
        }
    });


    increment_th.start();
    decrement_th.start();


    increment_th.wait();
    decrement_th.wait();
```

```
    qInfo() << "Counter value = " << counter.get_counter();
}


Counter::Counter()
: _counter{0} {


}


void Counter::increment() {
    ++_counter;
}


void Counter::decrement() {
    --_counter;
}


qint32 Counter::get_counter() const noexcept {
    return _counter.loadAcquire();
}


} // Tasks
```

Tasks/Task4.hpp

```
#ifndef __LAB4_TASKS_TASK1_HPP__
#define __LAB4_TASKS_TASK1_HPP__
```

```cpp
#include <QRunnable>
#include <QVector>

namespace Tasks {

class Task4 : public QRunnable {
public:
    void run() override;
};

class Result {
public:
    QVector<QVector<int>> _matrix;

    Result(int rows, int cols);

    void set_value(int row, int col, int value);
    int get_value(int row, int col) const;
};

} // Tasks

#endif // __LAB4_TASKS_TASK1_HPP__
```

Tasks/Task4.cpp

```cpp
#include "Task4.hpp"
```

```cpp
#include <QDebug>
#include <QElapsedTimer>
#include <QVector>
#include <QThreadPool>
#include <QMutex>
#include <QMutexLocker>

namespace Tasks {

QMutex mutex;  // Глобальный мьютекс для синхронизации доступа к матрице C

void Task4::run() {
    qDebug() << __PRETTY_FUNCTION__ << "called";

    int size = 1000;
    int num_threads = 4;

    QVector<QVector<int>> A(size, QVector<int>(size, 1));  // Матрица A
    QVector<QVector<int>> B(size, QVector<int>(size, 2));  // Матрица B
    Result C(size, size);

    QElapsedTimer timer;
    timer.start();
```

```
    auto multiply_block = [&A, &B, &C, size](int start_row, int end_row) {
        qDebug() << "Thread started for rows" << start_row << "to" << end_row;
        for (int i = start_row; i < end_row; ++i) {
            for (int j = 0; j < size; ++j) {
                int sum = 0;
                for (int k = 0; k < size; ++k) {
                    sum += A[i][k] * B[k][j];
                }
                // Использование мьютекса для защиты доступа к матрице C
                QMutexLocker locker(&mutex);
                C.set_value(i, j, sum);
            }
        }
        qDebug() << "Thread finished for rows" << start_row << "to" << end_row;
    };


    int rows_per_thread = size / num_threads;
    QThreadPool pool;
    pool.setMaxThreadCount(num_threads);


    for (int t = 0; t < num_threads; ++t) {
        int start_row = t * rows_per_thread;
        int end_row = (t == num_threads - 1) ? size : start_row + rows_per_thread;
        pool.start([&multiply_block, start_row, end_row](){ multiply_block(start_row,
end_row); });
    }


    pool.waitForDone();
```

```
    qint64 elapsed = timer.elapsed();
    qDebug() << "Time elapsed with" << num_threads << "threads:" << elapsed << "ms";
}

Result::Result(int rows, int cols) {
    _matrix.resize(rows);
    for (auto& row : _matrix) {
        row.resize(cols, 0);
    }
}

void Result::set_value(int row, int col, int value) {
    _matrix[row][col] = value;
}

int Result::get_value(int row, int col) const {
    return _matrix[row][col];
}

} // namespace Tasks
```

Результат виконання:

```
mbukhta@mbukhta:~/Documents/University/Grade_4/Multythreading/Lab_1/Lab_1_VSCode
/build$ ./multythreading_lab_1
Лабораторна робота №1
З дисципліни Технології Розподілених Систем та Паралельних Обчислень
Студента групи ІП3-3.04
Бухти Микити
virtual void Tasks::Task1::run() called
We have the Java learning course!
virtual void Tasks::Task2::run() called
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|
void Tasks::ConsoleSyncWriter::write_repeatly(QAtomicInt&, qint32, qint32) finis
hed
void Tasks::ConsoleSyncWriter::write_repeatly(QAtomicInt&, qint32, qint32) finis
hed
virtual void Tasks::Task3::run() called
Counter value =  0
virtual void Tasks::Task4::run() called
Thread started for rows 250 to 500
Thread started for rows 0 to 250
Thread started for rows 500 to 750
Thread started for rows 750 to 1000
Thread finished for rows 500 to 750
Thread finished for rows 250 to 500
Thread finished for rows 750 to 1000
Thread finished for rows 0 to 250
Time elapsed with 4 threads: 30073 ms
```

Рисунок 1 – результат виконання лабоpатнорної роботи №1