НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ  
імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра прикладної математики

Звіт

до лабораторної роботи №4

з дисципліни «Операційні системи»

на тему

«Процеси та потоки в ОС Windows»

|  |  |
| --- | --- |
| Виконав: | Керівник: |
| студент групи КМ-51 | Громова В. В. |
| Лук’яненко А. М. |  |

Київ — 2018

ЗМІСТ

[ВСТУП 3](#_Toc514629282)

[1 ПОСТАНОВКА ЗАДАЧІ 4](#_Toc514629283)

[2 ТЕОРЕТИЧНІ ВІДОМОСТІ 5](#_Toc514629284)

[2.1 Режим багатозадачевостi 5](#_Toc514629285)

[2.2 Планування потокiв 5](#_Toc514629286)

[2.3 Синхронiзацiя пiд час виконання потокiв 6](#_Toc514629287)

[2.4 Призупинення виконання потоку 7](#_Toc514629288)

[3 ОПИС РОЗРОБЛЕНОЇ ПРОГРАМИ 8](#_Toc514629289)

[4 РЕЗУЛЬТАТИ ВИПРОБУВАННЯ РОЗРОБЛЕНОЇ ПРОГРАМИ 11](#_Toc514629290)

[ВИСНОВКИ 14](#_Toc514629291)

[ПОСИЛАННЯ НА ЛIТЕРАТУРНI ДЖЕРЕЛА 15](#_Toc514629292)

[Додаток А Текст програми 16](#_Toc514629293)

# ВСТУП

Аплiкацiї, написанi для операцiйної системи (ОС) Microsoft Windows, складаються з одного або декiлькох процесiв. Згiдно з найпростiшим визначенням, пiд процесом (process) розумiють виконувану програму. Кожний процес забезпечує аплiкацiю потрiбними для виконання ресурсами.

Процес має вiртуальний адресовий простiр, виконуваний код, вiдкритi дескриптори системних об’єктiв (open handles to system objects), безпековий контекст (security context), унiкальний iдентифiкатор, змiннi середовища, клас прiоритету, мiнiмальний та максимальний розмiр робочої множини (working set), а також щонайменше один потiк.

Потiк (thread) — це базова одиниця в рамках процесу, якiй ОС видiляє процесорний час. Потiк може виконувати будь-яку частину коду процесу, у тому числi частини коду, якi в даний момент виконує iнший потiк. Усi потоки процесу мають спiльний вiртуальний адресовий простiр та системнi ресурси. Окрiм того, кожний процес пiдтримує опрацьовувачi виключних ситуацiй (exception handlers), планувальний прiоритет (scheduling priority), локальну пам’ять (local storage), унiкальний iдентифiкатор, а також низку структур, якi ОС використовує для збереження контексту потоку пiд час його простоювання. Контекст потоку (thread context) включає значення регiстрiв процесора, системний стек (kernel stack), блок середовища потоку, а також користувацький стек (user stack) в адресовому просторi процесу, якому належить потiк. Потоки також можуть мати власний безпековий контекст.

У данiй роботi розглядатимемо засоби роботи з багатопотоковими аплiкацiями в ОС Windows версiї XP та вище.

# 1 ПОСТАНОВКА ЗАДАЧІ

У данiй лабораторнiй роботi потрiбно ознайомитися з концепцiєю багатозадачевостi в ОС Windows, навчитися створювати за допомогою засобiв Windows API багатопотоковi аплiкацiї та розбиратися з проблемами, що виникають пiд час спiльного множинного доступу до ресурсiв.

У рамках виконання лабораторної роботи потрiбно:

1. ознайомитися з теоретичними вiдомостями;
2. написати будь-якою мовою програмування програму, яка повинна:

* за потреби створювати додатковi потоки, окрiм головного;
* регулювати доступ потокiв до деякого ресурсу (файлу, графiчного об’єкта, статичної змiнної i т.п.);
* зокрема, у програмi повинно бути передбачено можливiсть як синхронного, так i асинхронного доступу потокiв до ресурсу (на вимогу користувача);
* регулювати прiоритети потокiв на вимогу користувача;
* за потреби зупиняти та вiдновлювати потоки;
* вiдлагодити програму в ОС Windows версiї XP та вище;
* пiдготувати звiт iз лабораторної роботи вiдповiдно до вимог.

# 2 ТЕОРЕТИЧНІ ВІДОМОСТІ

## 2.1 Режим багатозадачевостi

Багатозадачевi операцiйнi системи роздiляють наявний процесорний час мiж процесами чи потоками, якi його потребують. ОС Windows розроблено для витискуючої багатозадачевостi: вона видiляє квант часу (time slice) кожному потоку, який вона виконує. Щойно квант часу поточного потоку вичерпано, ОС зупиняє його виконання, дозволяючи таким чином виконуватися iншим потокам. Пiд час перемикання з одного потоку на iнший, ОС зберiгає контекст витисненого потоку та вiдновлює збережений ранiше контекст наступного потоку в черзi.

Величина кванту часу залежить вiд ОС та вiд процесора. Проте, оскiльки зазвичай квант часу доволi малий (приблизно 20 мс), створюється враження, що одночасно виконуються багато потокiв. Проте, багатопотоковiсть варто використовувати обережно, оскiльки за великої кiлькостi потокiв продуктивнiсть ОС може знизитися.

## 2.2 Планування потокiв

Управлiння багатозадачевiстю здiйснює планувальник ОС (system scheduler). Його завдання полягає у визначеннi потоку, який повинен отримати черговий квант процесорного часу. При цьому планувальник ураховує планувальнi прiоритети (scheduling priorities) потокiв. Кожному потоку призначають планувальний прiоритет, який може набувати значень вiд 0 (найнижчий прiоритет) до 31 (найвищий прiоритет).

ОС розподiляє кванти часу згiдно з циклiчним алгоритмом (round robin), застосовуючи його послiдовно до кожної групи потокiв з однаковим прiоритетом:

* спочатку ОС намагається видiлити квант часу згiдно з циклiчним алгоритмом потокам iз найвищим прiоритетом;
* якщо жоден iз потокiв iз найвищим прiоритетом не готовий до виконання, ОС намагається видiлити квант часу згiдно з циклiчним алгоритмом потокам iз наступним за величиною прiоритетом, i так далi;
* якщо потiк iз вищим прiоритетом стає готовим до виконання у той час, як виконується потiк iз нижчим прiоритетом, ОС припиняє виконання останнього (на даючи йому завершити свiй квант часу) та призначає повний квант часу потоку з вищим прiоритетом.

Прiоритет кожного потоку залежить вiд класу прiоритету (priority class) процесу, якому вiн належить, та рiвня прiоритету (priority level) потоку в рамках вiдповiдного класу прiоритету. Клас та рiвень прiоритету об’єднують для визначення базового прiоритету (base priority) потоку.

## 2.3 Синхронiзацiя пiд час виконання потокiв

Для того, щоб упередити появу гонок (race conditions) та взаємоблокування (deadlocks) пiд час виконання декiлькох потокiв, потрiбно синхронiзувати доступ потокiв до спiльних ресурсiв. Синхронiзацiю також потрiбно здiйснювати, щоб забезпечити виконання взаємопов’язаного коду в правильнiй послiдовностi.

Наприклад, у деякiй аплiкацiї один потiк обчислює координати певної фiгури в деякiй глобальнiй структурi даних, а iнший потiк використовує цi координати для малювання фiгури. Якщо перший потiк устиг змiнити одну з координат, але не встиг змiнити iншу (оскiльки його квант часу завершився ранiше), то другий потiк може використати для малювання некоректнi координати.

Windows API надає можливiсть використовувати низку об’єктiв, дескриптори яких можна використовувати для синхронiзацiї пiд час виконання декiлькох потокiв:

* вхiднi буфери консолi (console input buffers);
* подiї (events);
* м’ютекси (mutexes);
* процеси (processes);
* семафори (semaphores);
* потоки (threads);
* таймери (timers).

## 2.4 Призупинення виконання потоку

Будь-який потiк може призупиняти та вiдновлювати виконання iншого потоку. Як було зазначено ранiше, якщо потiк призупинено, ОС не видiляє йому процесорний час. Для призупинення дiї потоку використовують функцiю SuspendThread.

Єдиним аргументом цiєї функцiї є дескриптор потоку, який потрiбно призупинити. У випадку помилки функцiя повертає значення –1. У випадку успiшного завершення, функцiя збiльшує на 1 лiчильник призупинень вiдповiдного потоку та повертає значення цього лiчильника до збiльшення. Для вiдновлення дiї потоку використовують функцiю ResumeThread.

Створення потоку в заблокованому станi з наступним вiдновленням його виконання з iншого потоку може бути доцiльним для:

* iнiцiалiзацiї стану потоку перед початком його виконання;
* одноразової синхронiзацiї.

Варто зазначити, що використання функцiї SuspendThread не дозволяє досягти одноразової синхронiзацiї, оскiльки пiд час виконання аплiкацiї неможливо визначити, у якiй точцi коду буде здiйснено призупинення потоку. Натомiсть, застосування пiдходу з призупиненням виконання потоку в момент його створення гарантує, що потiк буде призупинено в початковiй точцi коду. Потiк може тимчасово призупиняти власне виконання на деякий iнтервал часу за допомогою функцiй Sleep та SleepEx.

# 3 ОПИС РОЗРОБЛЕНОЇ ПРОГРАМИ

Для того, щоб добре показати роботу потоків, була створена програма для сортування масивів. Кожний потік виконує сортування певного масиву, таким чином можна побачити скільки потоків працює одночасно, з якою швидкістю та пріоритетом.

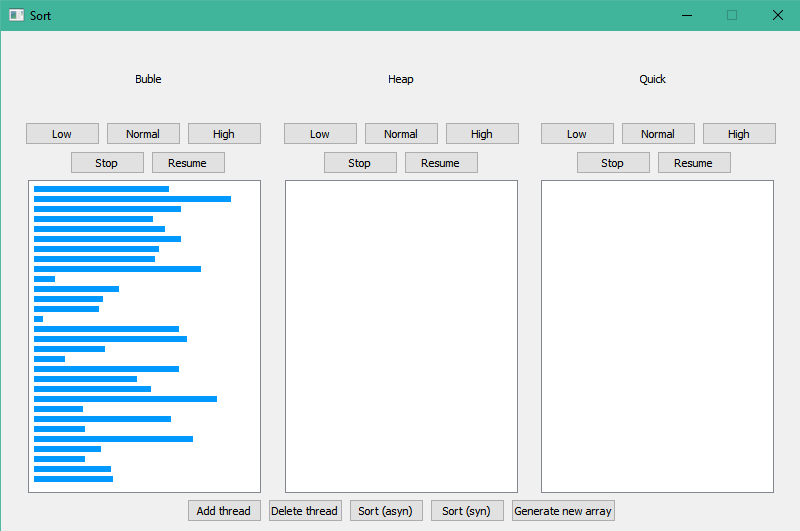
Розроблена програма на мові С++ має назву “Sort”. Вона створена за допомогою Windows API та допоміжної бібліотеки QT. Вигляд стартового вікна програми на рисунку 3.1.

Рисунок 3.1 – Стартове вікно програми

Стартове вікно містить в собі функціонал, що дозволяє створювати та видаляти потоки за допомогою клавіш “Add thread” та “Delete thread”, зупиняти та відновлювати роботу потоків, сортувати масиви, використовуючи синхронний та асинхроний запуск потоків, а також встановлювати проритет для кожного потоку в режимі синхронного сортування.

В програмі використані такі алгоритми сортування масивів:

* Bubble sort – у поданому наборі даних (списку чи масиві) порівнюються два сусідні елементи. Якщо один з елементів, не відповідає критерію сортування (є більшим, або ж, навпаки, меншим за свого сусіда), то ці два елементи міняються місцями. Прохід по списку продовжується доти, доки дані не будуть відсортованими:
* Heap sort – cортування пірамідою використовує бінарне сортувальне дерево. Сортувальне дерево — це таке дерево, у якого виконані умови:

1. Кожен лист має глибину або *d*, або *(d – 1)*, де *d* — максимальна глибина дерева;
2. Значення в будь-якій вершині не менші (інший варіант — не більші) за значення їх нащадків;

* Quick sort – ідея алгоритму полягає в переставлянні елементів масиву таким чином, щоб його можна було розділити на дві частини і кожний елемент з першої частини був не більший за будь-який елемент з другої. Впорядкування кожної з частин відбувається рекурсивно. Алгоритм швидкого сортування може бути реалізований як у масиві, так і в двозв'язному списку.

При написанні програми були використані наступні функції для зв’язку аплікації з ОС та обробкою дій користувача:

* QGridLayout, QWidget, QPushButton – функції для створення клавіш у програмі та їх розміщення у вікні;
* connect(pb\_gen,SIGNAL(clicked()),this,SLOT(gen())) – функція, що дозволяє переключатися між подіями, в залежності від дій користувача чи результатів роботи інших функцій;
* setPriority(Priority) – функція, що встановлювала пріоритет вибраним потокам;
* QBrush, QPen, addPolygon() – функції для візуалізації сортування;
* start(), stop(), run(), resume() – функції для керування роботою потоків.

Основні бібліотеки, що були використані для функціонування процедур:

* <QtCore/QtGlobal>;
* <QThread>;
* <QtWidgets/QMainWindow>;
* <QtWidgets/QGridLayout>.

Головне вікно розбите на 3 частини, в кожній з яких можна дослідити роботу потоків. У верхній частині знаходяться назви кожного алгоритму сортування, одразу під ними можливі пріоритети для синхронного сортування. Також є клавіши “Start” та “Resume” для зупинки та відновлення певного потоку відповідно.

Візуалізація алгоритмів сортування відбувається за допомогою створення діаграм після кожної ітерації для кращого бачення результату.

Користувач може вибрати асинхронну або синхронну роботу потоків, натиснувши відповідну клавішу у меню, що знаходиться у нижній частині вікна (рис.3.2).



Рисунок 3.2 – Основне меню програми

# 4 РЕЗУЛЬТАТИ ВИПРОБУВАННЯ РОЗРОБЛЕНОЇ ПРОГРАМИ

Для того, щоб продемонструвати повний функціонал програми необхідно:

1. додати та видалити потоки;
2. зупинити та відновити роботу потоку;
3. запустити програму синхронному та асинхронному режимах сортування;
4. протестувати синхронний режим з різними комбінаціями пріоритетів;
5. згенерувати новий масив та повторити дії.

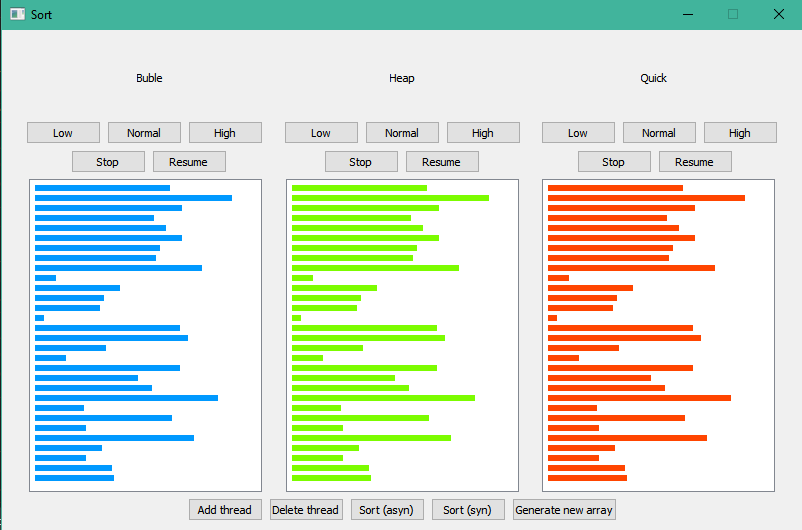
 Спочатку створимо потоки за допомогою клавіши “Add thread”. В даній програмі максимальна кількість потоків дорівнює 3 (рис. 4.1).

Рисунок 4.1 – Нові потоки у програмі

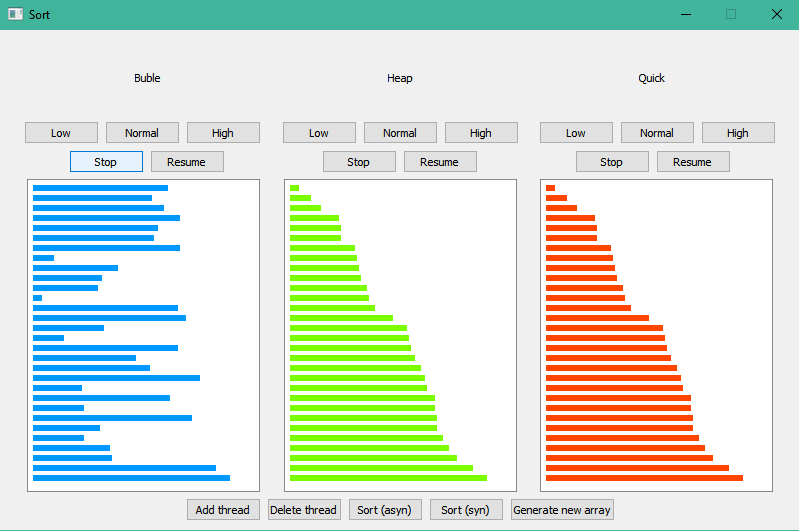
Тепер можна запустити програму в асинхронному режимі – потоки виконуються одночасно та закінчують роботу по завершенню сортування. В цьому режимі також можна протестувати функцію зупинки/відновлення потоків. На рисунку 4.2 показано, що лівий потік був зупинений через втручання користувача, в той час, як інші два вже закінчили свою роботу.

Рисунок 4.2 – Зупинка лівого потоку

Для відновлення роботи потоку достатньо натиснути на клавішу “Resume”. Варто звернути увагу, що при зупиненні потоку користувачем, його робота не відновиться навіть при новому запуску сортування. Тобто, якщо зупинка була здійснена натиском на клавішу “Stop”, то і відновлення буде відбуватися в залежності від дій користувача. Остаточний результат сортування наведений на рисунку 4.3.

Тепер встановимо пріоритет потоків, натиснувши відповідні клавіши під назвами алгоритмів сортування. Далі треба натиснути клавішу “Sort (syn)”. Для прикладу, встановимо такі пріоритети: правий – високий, центровий – нормальний, лівий – низький.

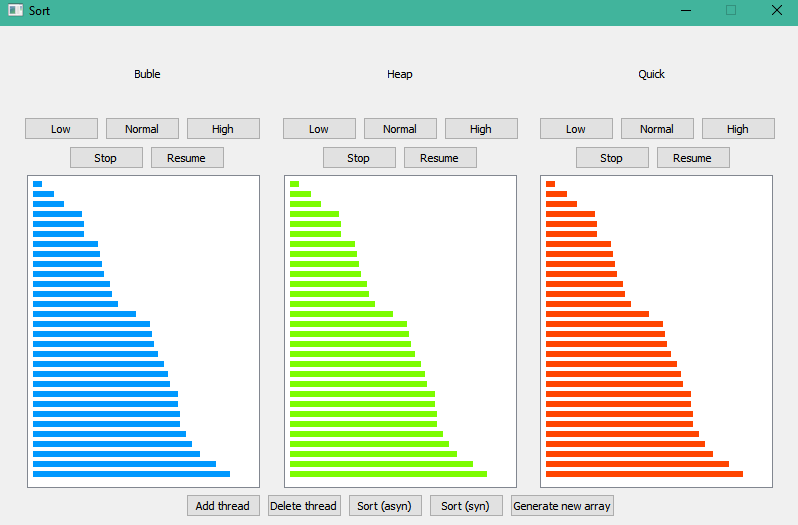


Рисунок 4.3 – Остаточний результат сортування

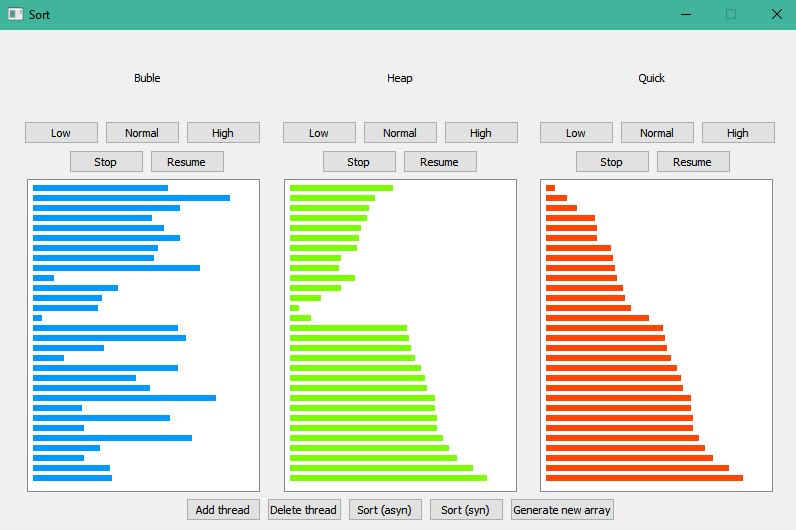
 Зазначимо, що при встановленні однакових пріоритетів, наприклад, лівий та правий – високі пріоритети, потоки будуть виконуватися одночасно, а вже після їх роботи виконається останній потік. Рисунок 4.4 демонструє роботу синхронного режиму, де видно, що правий потік вже завершився, середній почав роботу, а лівий ще у стані очікування, поки середній не закінчить сортування.

Рисунок 4.4 – Робота програми у синхронному режимі

# ВИСНОВКИ

У данiй лабораторнiй роботi була вивчена концепцiя багатозадачевостi в ОС Windows, отримані навички створення за допомогою засобiв Windows API багатопотокові аплiкацiї та роботи з проблемами, що виникають пiд час спiльного множинного доступу до ресурсiв.

Була розроблена програма на мові С++ має назву “Sort”. Вона створена за допомогою Windows API та допоміжної бібліотеки QT для сортування масивів та візуалізації роботи потоків.

При розробці були використані тiльки засоби Windows API та бібліотеки QT. Програма вiдлагоджена в ОС Windows 10 Pro.

# ПОСИЛАННЯ НА ЛIТЕРАТУРНI ДЖЕРЕЛА

1. Microsoft Library [Електронний ресурс]. — Режим доступу: <https://msdn.microsoft.com/en-us/library/>
2. Мешков А. Visual C++ и MFC / А. Мешков, Ю. Тихомиров. — [2-еизд.] — С.-Пб. : BHV, 2001. — 1040 с.
3. Petzold C. Programming Windows / C. Petzold. — [5th ed.] — Microsoft Press, 1998. — 1100 p.
4. Simon R. Windows NT Win32 API SuperBible / R. Simon. — Waite Group Press, 1997. — 1510 p.

# Додаток А Текст програми

#include "mainwindow.h"

void MainWindow::add\_thread(){

if (threads < 3)

{

threads++;

reset();

}

}

void MainWindow::del\_thread(){

if (threads >= 2){

threads--;

}

if (threads == 2)

{gs3->clear();}

if (threads == 1)

{gs2->clear();}

}

void MainWindow::pr\_bh(){pr\_b = 3;}

void MainWindow::pr\_bn(){pr\_b = 2;}

void MainWindow::pr\_bl(){pr\_b = 1;}

void MainWindow::pr\_hh(){pr\_h = 3;}

void MainWindow::pr\_hn(){pr\_h = 2;}

void MainWindow::pr\_hl(){pr\_h = 1;}

void MainWindow::pr\_qh(){pr\_q = 3;}

void MainWindow::pr\_qn(){pr\_q = 2;}

void MainWindow::pr\_ql(){pr\_q = 1;}

//window parameters

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

{

this->setWindowTitle("Sort");

this->setMinimumSize(800,500);

this->setMaximumSize(800,500);

glMain = new QGridLayout;

wCenter = new QWidget;

wCenter->setLayout(glMain);

this->setCentralWidget(wCenter);

bubles = new Bubles();

heap = new Heap();

quick = new Quick();

QPushButton \*pb=new QPushButton("Sort (asyn)");

QPushButton \*pb\_syn=new QPushButton("Sort (syn)");

QPushButton \*pb\_add=new QPushButton("Add thread");

QPushButton \*pb\_del=new QPushButton("Delete thread");

QPushButton \*pb\_gen = new QPushButton("Generate new array");

QHBoxLayout \*hb=new QHBoxLayout;

hb->addStretch();

hb->addWidget(pb\_add);

hb->addWidget(pb\_del);

hb->addWidget(pb);

hb->addWidget(pb\_syn);

hb->addWidget(pb\_gen);

hb->addStretch();

glMain->addLayout(hb,6,0);

connect(pb\_add,SIGNAL(clicked()),this,SLOT(add\_thread()));

connect(pb\_del,SIGNAL(clicked()),this,SLOT(del\_thread()));

QHBoxLayout \*hb\_label=new QHBoxLayout;

hb\_label->addStretch(4);

QLabel \*lbl = new QLabel("Buble");

hb\_label->addWidget(lbl);

hb\_label->addStretch(7);

hb\_label->addWidget(new QLabel("Heap"));

hb\_label->addStretch(7);

hb\_label->addWidget(new QLabel("Quick"));

hb\_label->addStretch(4);

glMain->addLayout(hb\_label,0,0);

int w=230,h=310;

gs1=new QGraphicsScene(this);

gs1->setSceneRect(0, 0, w, h);

gv1=new QGraphicsView(gs1);

gv1->setFixedHeight(gs1->height()+3);

gv1->setFixedWidth(gs1->width()+3);

gs2=new QGraphicsScene(this);

gs2->setSceneRect(0, 0, w, h);

gv2=new QGraphicsView(gs2);

gv2->setFixedHeight(gs2->height()+3);

gv2->setFixedWidth(gs2->width()+3);

gs3=new QGraphicsScene(this);

gs3->setSceneRect(0, 0, w, h);

gv3=new QGraphicsView(gs3);

gv3->setFixedHeight(gs3->height()+3);

gv3->setFixedWidth(gs3->width()+3);

QHBoxLayout \*hb\_gv=new QHBoxLayout;

hb\_gv->addStretch(5);

hb\_gv->addWidget(gv1);

hb\_gv->addStretch(5);

hb\_gv->addWidget(gv2);

hb\_gv->addStretch(5);

hb\_gv->addWidget(gv3);

hb\_gv->addStretch(5);

glMain->addLayout(hb\_gv,3,0);

glMain->setRowStretch(2,0);

QHBoxLayout \*hp;

QPushButton \*prior\_bh = new QPushButton("High");

QPushButton \*prior\_hh = new QPushButton("High");

QPushButton \*prior\_qh = new QPushButton("High");

QPushButton \*prior\_bn = new QPushButton("Normal");

QPushButton \*prior\_hn = new QPushButton("Normal");

QPushButton \*prior\_qn = new QPushButton("Normal");

QPushButton \*prior\_bl = new QPushButton("Low");

QPushButton \*prior\_hl = new QPushButton("Low");

QPushButton \*prior\_ql = new QPushButton("Low");

hp=new QHBoxLayout;

hp->addStretch(2);

hp->addWidget(prior\_bl);

hp->addWidget(prior\_bn);

hp->addWidget(prior\_bh);

hp->addStretch(2);

hp->addWidget(prior\_hl);

hp->addWidget(prior\_hn);

hp->addWidget(prior\_hh);

hp->addStretch(2);

hp->addWidget(prior\_ql);

hp->addWidget(prior\_qn);

hp->addWidget(prior\_qh);

hp->addStretch(2);

glMain->addLayout(hp,1,0);

connect(prior\_bh,SIGNAL(clicked()),this,SLOT(pr\_bh()));

connect(prior\_bn,SIGNAL(clicked()),this,SLOT(pr\_bn()));

connect(prior\_bl,SIGNAL(clicked()),this,SLOT(pr\_bl()));

connect(prior\_hh,SIGNAL(clicked()),this,SLOT(pr\_hh()));

connect(prior\_hn,SIGNAL(clicked()),this,SLOT(pr\_hn()));

connect(prior\_hl,SIGNAL(clicked()),this,SLOT(pr\_hl()));

connect(prior\_qh,SIGNAL(clicked()),this,SLOT(pr\_qh()));

connect(prior\_qn,SIGNAL(clicked()),this,SLOT(pr\_qn()));

connect(prior\_ql,SIGNAL(clicked()),this,SLOT(pr\_ql()));

QHBoxLayout \*hs;

QPushButton \*pb\_stop\_b = new QPushButton("Stop");

QPushButton \*pb\_resume\_b = new QPushButton("Resume");

QPushButton \*pb\_stop\_h = new QPushButton("Stop");

QPushButton \*pb\_resume\_h = new QPushButton("Resume");

QPushButton \*pb\_stop\_q = new QPushButton("Stop");

QPushButton \*pb\_resume\_q = new QPushButton("Resume");

hs=new QHBoxLayout;

hs->addStretch(2);

hs->addWidget(pb\_stop\_b);

hs->addStretch();

hs->addWidget(pb\_resume\_b);

hs->addStretch(3);

hs->addWidget(pb\_stop\_h);

hs->addStretch();

hs->addWidget(pb\_resume\_h);

hs->addStretch(3);

hs->addWidget(pb\_stop\_q);

hs->addStretch();

hs->addWidget(pb\_resume\_q);

hs->addStretch(2);

glMain->addLayout(hs,2,0);

brush = QBrush(QColor(0,153,255));

pen = QPen(QColor(0,153,255),0);

this->gen();

this->reset();

connect(pb\_gen,SIGNAL(clicked()),this,SLOT(gen()));

connect(pb,SIGNAL(clicked()),this,SLOT(start()));

connect(pb\_syn,SIGNAL(clicked()),this,SLOT(start\_syn()));

connect(bubles,SIGNAL(draw(int,QVector<int>)),this,SLOT(draw(int,QVector<int>)));

connect(heap,SIGNAL(draw(int,QVector<int>)),this,SLOT(draw(int,QVector<int>)));

connect(quick,SIGNAL(draw(int,QVector<int>)),this,SLOT(draw(int,QVector<int>)));

connect(pb\_stop\_b,SIGNAL(clicked()),this,SLOT(stop\_b()));

connect(pb\_stop\_h,SIGNAL(clicked()),this,SLOT(stop\_h()));

connect(pb\_stop\_q,SIGNAL(clicked()),this,SLOT(stop\_q()));

connect(pb\_resume\_b,SIGNAL(clicked()),this,SLOT(resume\_b()));

connect(pb\_resume\_h,SIGNAL(clicked()),this,SLOT(resume\_h()));

connect(pb\_resume\_q,SIGNAL(clicked()),this,SLOT(resume\_q()));

}

MainWindow::~MainWindow()

{

}

#include "mainwindow.h"

void MainWindow::draw(int gs\_s, QVector<int> array){

QGraphicsScene \*gs;

switch(gs\_s){ //выбираем сцену для соответствующего алгоритма и даем ей свои цвета

case 1: gs=gs1;

brush = QBrush(QColor(0,153,255));

pen = QPen(QColor(0,153,255),0);

break;

case 2: gs=gs2;

brush = QBrush(QColor(124,252,0));

pen = QPen(QColor(124,252,0),0);

break;

case 3: gs=gs3;

brush = QBrush(QColor(255,69,0));

pen = QPen(QColor(255,69,0),0);

break;

}

gs->clear(); //очищаем предыдущую сортировку

int max=array[0]; //начальное условие для поиска максимального элемента в массиве (чтобы определить масштаб)

for(int i=1;i<array.count();i++) if(array[i]>max)max=array[i]; //поиск максимального элемента

int k=(gs->width()-10)/max; //вычисляем масштаб - ширина сцены минус запас 10px и делим это все на максимальное значение

for(int i=0;i<array.count();i++){

gs->addPolygon(QPolygon(QRect(5,i\*10+5,k\*array[i],5)),pen,brush); //рисуем прямоугольник

}

}