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:
студент групи КМ-51
Лук’яненко А. М.

Керівник:
Громова В. В.

Київ — 2018

ЗМІСТ

ВСТУП

Аплікації, написані для операційної системи (ОС) Microsoft Windows, складаються з одного або декількох процесів. Згідно з найпростішим визначенням, під процесом (process) розуміють виконувану програму. Кожний процес забезпечує аплікацію потрібними для виконання ресурсами.

Процес має віртуальний адресовий простір, виконуваний код, відкриті дескриптори системних об'єктів (open handles to system objects), безпековий контекст (security context), унікальний ідентифікатор, змінні середовища, клас пріоритету, мінімальний та максимальний розмір робочої множини (working set), а також щонайменше один потік.

Потік (thread) — це базова одиниця в рамках процесу, якій ОС виділяє процесорний час. Потік може виконувати будь-яку частину коду процесу, у тому числі частини коду, які в даний момент виконує інший потік. Усі потоки процесу мають спільний віртуальний адресовий простір та системні ресурси. Окрім того, кожний процес підтримує опрацьовувачі виключних ситуацій (exception handlers), планувальний пріоритет (scheduling priority), локальну пам'ять (local storage), унікальний ідентифікатор, а також низку структур, які ОС використовує для збереження контексту потоку під час його простоювання. Контекст потоку (thread context) включає значення регістрів процесора, системний стек (kernel stack), блок середовища потоку, а також користувацький стек (user stack) в адресовому просторі процесу, якому належить потік. Потоки також можуть мати власний безпековий контекст.

У даній роботі розглядатимемо засоби роботи з багатопотоковими аплікаціями в ОС Windows версії XP та вище.

1 ПОСТАНОВКА ЗАДАЧІ

У даній лабораторній роботі потрібно ознайомитися з концепцією багатозадачевості в ОС Windows, навчитися створювати за допомогою засобів Windows API багатопотокові аплікації та розбиратися з проблемами, що виникають під час спільного множинного доступу до ресурсів.

У рамках виконання лабораторної роботи потрібно:

1. ознайомитися з теоретичними відомостями;
2. написати будь-якою мовою програмування програму, яка повинна:
   - за потреби створювати додаткові потоки, окрім головного;
   - регулювати доступ потоків до деякого ресурсу (файлу, графічного об’єкта, статичної змінної і т.п.);
   - зокрема, у програмі повинно бути передбачено можливість як синхронного, так і асинхронного доступу потоків до ресурсу (на вимогу користувача);
   - регулювати пріоритети потоків на вимогу користувача;
   - за потреби зупиняти та відновлювати потоки;
   - відлагодити програму в ОС Windows версії XP та вище;
   - підготувати звіт із лабораторної роботи відповідно до вимог.

2 ТЕОРЕТИЧНІ ВІДОМОСТІ

2.1 Режим багатозадачевості

Багатозадачеві операційні системи розділяють наявний процесорний час між процесами чи потоками, які його потребують. ОС Windows розроблено для витискуючої багатозадачевості: вона виділяє квант часу (time slice) кожному потоку, який вона виконує. Щойно квант часу поточного потоку вичерпано, ОС зупиняє його виконання, дозволяючи таким чином виконуватися іншим потокам. Під час перемикання з одного потоку на інший, ОС зберігає контекст витисненого потоку та відновлює збережений раніше контекст наступного потоку в черзі.

Величина кванту часу залежить від ОС та від процесора. Проте, оскільки зазвичай квант часу доволі малий (приблизно 20 мс), створюється враження, що одночасно виконуються багато потоків. Проте, багатопотоковість варто використовувати обережно, оскільки за великої кількості потоків продуктивність ОС може знизитися.

2.2 Планування потоків

Управління багатозадачевістю здійснює планувальник ОС (system scheduler). Його завдання полягає у визначенні потоку, який повинен отримати черговий квант процесорного часу. При цьому планувальник ураховує планувальні пріоритети (scheduling priorities) потоків. Кожному потоку призначають планувальний пріоритет, який може набувати значень від 0 (найнижчий пріоритет) до 31 (найвищий пріоритет).

ОС розподіляє кванти часу згідно з циклічним алгоритмом (round robin), застосовуючи його послідовно до кожної групи потоків з однаковим пріоритетом:

- спочатку ОС намагається виділити квант часу згідно з циклічним алгоритмом потокам із найвищим пріоритетом;
- якщо жоден із потоків із найвищим пріоритетом не готовий до виконання, ОС намагається виділити квант часу згідно з циклічним алгоритмом потокам із наступним за величиною пріоритетом, і так далі;
- якщо потік із вищим пріоритетом стає готовим до виконання у той час, як виконується потік із нижчим пріоритетом, ОС припиняє виконання останнього (на даючи йому завершити свій квант часу) та призначає повний квант часу потоку з вищим пріоритетом.

Пріоритет кожного потоку залежить від класу пріоритету (priority class) процесу, якому він належить, та рівня пріоритету (priority level) потоку в рамках відповідного класу пріоритету. Клас та рівень пріоритету об’єднують для визначення базового пріоритету (base priority) потоку.

2.3 Синхронізація під час виконання потоків

Для того, щоб упередити появу гонок (race conditions) та взаємоблокування (deadlocks) під час виконання декількох потоків, потрібно синхронізувати доступ потоків до спільних ресурсів. Синхронізацію також потрібно здійснювати, щоб забезпечити виконання взаємопов’язаного коду в правильній послідовності.

Наприклад, у деякій аплікації один потік обчислює координати певної фігури в деякій глобальній структурі даних, а інший потік використовує ці координати для малювання фігури. Якщо перший потік устиг змінити одну з координат, але не встиг змінити іншу (оскільки його квант часу завершився раніше), то другий потік може використати для малювання некоректні координати.

Windows API надає можливість використовувати низку об’єктів, дескриптори яких можна використовувати для синхронізації під час виконання декількох потоків:

- вхідні буфери консолі (console input buffers);

- події (events);
- м’ютекси (mutexes);
- процеси (processes);
- семафори (semaphores);
- потоки (threads);
- таймери (timers).

2.4 Призупинення виконання потоку

Будь-який потік може призупиняти та відновлювати виконання іншого потоку. Як було зазначено раніше, якщо потік призупинено, ОС не виділяє йому процесорний час. Для призупинення дії потоку використовують функцію SuspendThread.

Єдиним аргументом цієї функції є дескриптор потоку, який потрібно призупинити. У випадку помилки функція повертає значення –1. У випадку успішного завершення, функція збільшує на 1 лічильник призупинень відповідного потоку та повертає значення цього лічильника до збільшення. Для відновлення дії потоку використовують функцію ResumeThread.

Створення потоку в заблокованому стані з наступним відновленням його виконання з іншого потоку може бути доцільним для:

- ініціалізації стану потоку перед початком його виконання;
- одноразової синхронізації.

Варто зазначити, що використання функції SuspendThread не дозволяє досягти одноразової синхронізації, оскільки під час виконання аплікації неможливо визначити, у якій точці коду буде здійснено призупинення потоку. Натомість, застосування підходу з призупиненням виконання потоку в момент його створення гарантує, що потік буде призупинено в початковій точці коду. Потік може тимчасово призупиняти власне виконання на деякий інтервал часу за допомогою функцій 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 C++ має назву “Sort”. Вона створена за допомогою Windows API та допоміжної бібліотеки QT. Вигляд стартового вікна програми на рисунку 3.1.

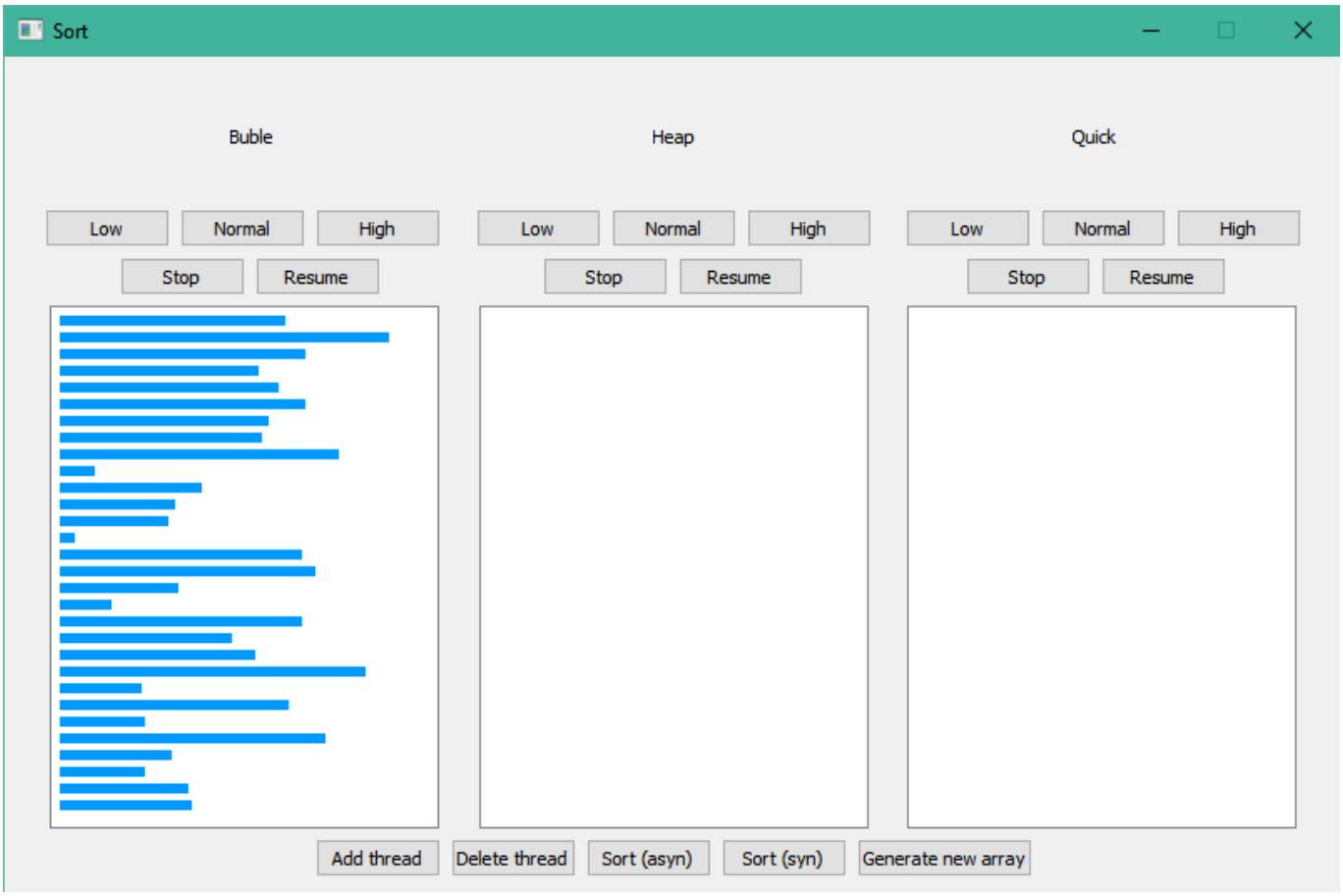

Рисунок 3.1 – Стартове вікно програми

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 – сортування 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).

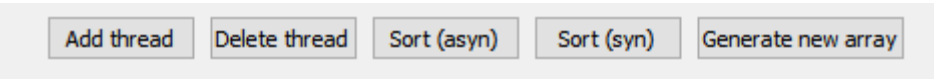

Рисунок 3.2 – Основне меню програми

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).

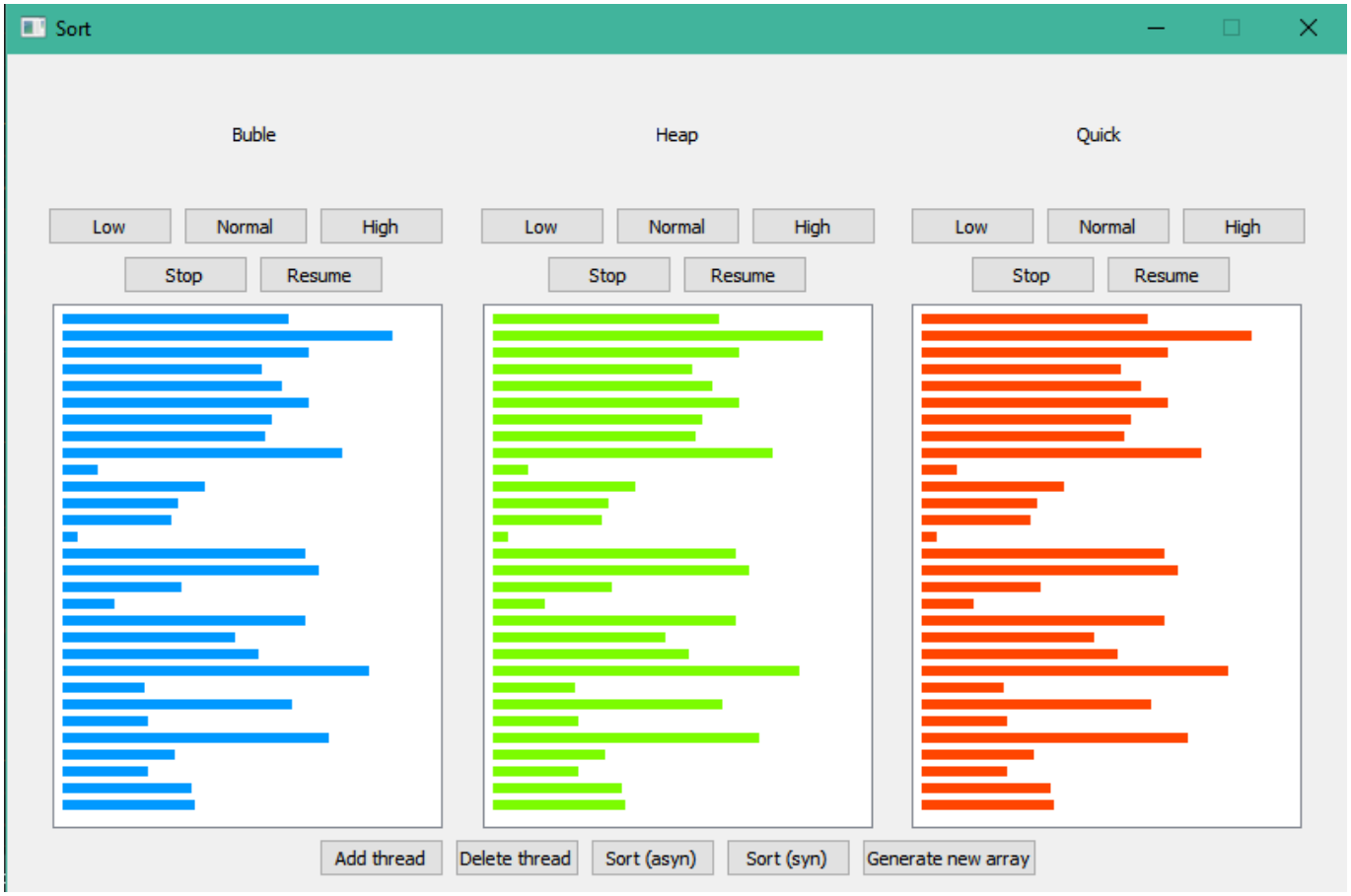

Рисунок 4.1 – Нові потоки у програмі

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.

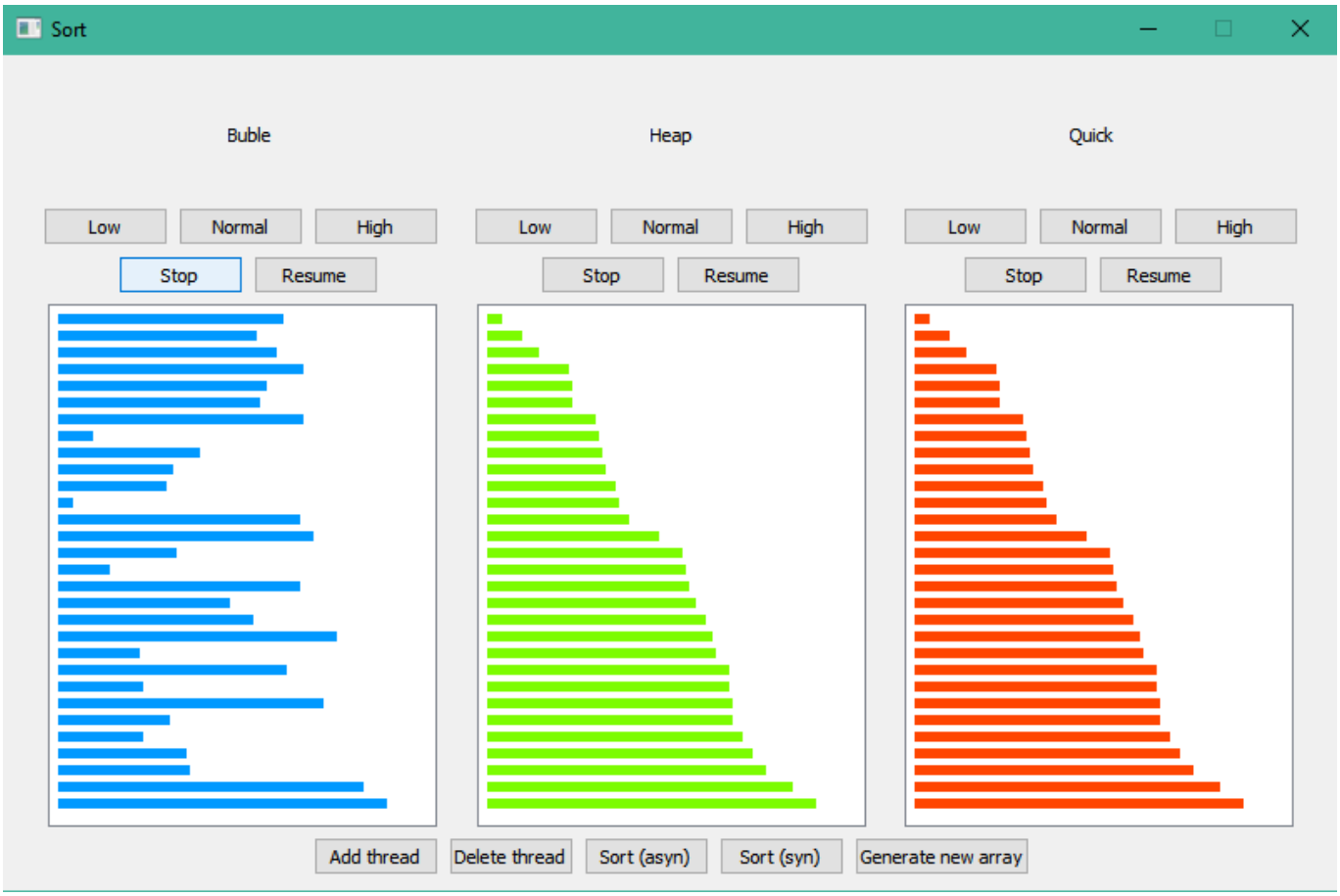

Рисунок 4.2 – Зупинка лівого потоку

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 даній лабораторній роботі була вивчена концепція багатозадачевості в ОС Windows, отримані навички створення за допомогою засобів Windows API багатопотокові аплікації та роботи з проблемами, що виникають під час спільного множинного доступу до ресурсів.

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

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

ПОСИЛАННЯ НА ЛІТЕРАТУРНІ 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
    }
}
```