МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Інститут **КНІТ**
Кафедра **ПЗ**

ЗВІТ

До лабораторної роботи № 5
На тему: "*Багатопоточність в операційній сисемі WINDOWS. Створення, керування та синхронізація потоків*"
З дисципліни: "Операційні системи"

Лектор:
старший викладач кафедри ПЗ
Грицай О.Д.

Виконав:
студент групи ПЗ-22
Коваленко Д.М.

Прийняла:
старший викладач кафедри ПЗ
Грицай О.Д.

«_____» _____ 2022 р.
$\sum$ = _____.....

Львів — 2022

Тема. Багатопоточність в операційній сисемі WINDOWS. Створення, керування та синхронізація потоків.

Мета. Ознайомитися з багатопоточністю в ОС Windows. Навчитись реалізовувати розпаралелення алгоритмів за допомогою багатопоточності в ОС Windows з використанням функцій WinAPI. Навчитись використовувати різні механізми синхронізації потоків.

Лабораторне завдання

1. Реалізувати заданий алгоритм в окремому потоці
2. Виконати розпаралелення заданого алгоритму на 2, 4, 8, 16 потоків
3. Реалізувати можливість зупинку роботи і відновлення, зміни пріоритету певного потоку
4. Реалізувати можливість завершення потоку
5. Застосувати різні механізми синхронізації потоків. (Згідно запропонованих варіантів)
6. Зобразити залежність час виконання – кількість потоків (для випадку без синхронізації і зі синхронізацією кожного виду).
7. Результати виконання роботи відобразити у звіті

Семафор, Інтерлок-функції

Хід роботи

```
#include "mainwindow.h"
#include "./ui_mainwindow.h"
#include <windows.h>
#include <stdio.h>
#include <iostream>
#include "QDebug"
#include "QLineEdit"
#include "QTimer"

struct PARAMETERS {
    int i;
};

struct threadStatus {
    DWORD threadID;
    int task;
};

HANDLE ghSemaphore;
DWORD WINAPI ThreadProc(LPVOID);

int* array;
int N;
int maxSemCount;
int threadCount = 0;
LONG sum = 0;
HANDLE* aThread;
DWORD ThreadID;
threadStatus* s;

MainWindow::MainWindow(QWidget *parent)
: QMainWindow(parent)
, ui(new Ui::MainWindow)
{
    ui->setupUi(this);
```

```
    QTimer *timer = new QTimer(this);
    connect(timer, &QTimer::timeout, this, &MainWindow::abc);
    timer->start(100);
}

MainWindow::~MainWindow() {
    for (int i = 0; i < threadCount; i++)
    CloseHandle(aThread[i]);

    CloseHandle(ghSemaphore);
    delete ui;
}

void MainWindow::on_pushButton_clicked() {
    ui->pushButton->setDisabled(true);
    N = ui->le_length->text().toInt();
    threadCount = ui->le_threads->text().toInt();
    s = new threadStatus[threadCount];
    maxSemCount = ui->lineEdit->text().toInt();
    aThread = new HANDLE[threadCount];
    array = new int[N];

    ghSemaphore = CreateSemaphore(
    NULL,
    maxSemCount,
    maxSemCount,
    NULL);

    if (ghSemaphore == NULL) {
        qDebug() << "CreateSemaphore error: " << GetLastError();
        return;
    }
    QStringList l;
    l << "IDLE" << "LOWEST" << "BELOW_NORMAL" << "NORMAL" << "ABOVE_NORMAL" << "HIGHEST";
    ui->prio2->addItems(l);
    QStringList list;
    for (int i = 0; i < threadCount; i++) {
        list << QString::number(i);

        PARAMETERS* p = new PARAMETERS{ i };
        s[i] = threadStatus { 0,0 };
        aThread[i] = CreateThread(
        NULL,
        0,
        (LPTHREAD_START_ROUTINE)ThreadProc,
        p,
        NULL,
        &ThreadID);
        if( aThread[i] == NULL ) {
            qDebug() << "CreateThread error: " << GetLastError();
            return;
        }
    }
    ui->sus->addItems(list);
    ui->res->addItems(list);
    ui->kill->addItems(list);
    ui->prio1->addItems(list);
}

void MainWindow::abc() {
    ui->le_sum->setText(QString::number(sum));
```

```
    ui->tableWidget->setColumnCount(3);
    ui->tableWidget->setRowCount(0);

    for (int i = 0; i < threadCount; i++) {
        ui->tableWidget->insertRow(i);
        for (int j = 0; j < 3; j++) {
            QTableWidgetItem * item = new QTableWidgetItem();
            item->setTextAlignment(Qt::AlignCenter);
            switch (j) {
                case 0:
                item->setText(QString::number(s[i].threadID));
                break;
                case 1:
                if (s[i].task == 0)
                item->setText(QString::fromStdString("Not Created"));
                else if (s[i].task == 1)
                item->setText(QString::fromStdString("Running"));
                else if (s[i].task == 3)
                item->setText(QString::fromStdString("Waiting"));
                else if (s[i].task == 4)
                item->setText(QString::fromStdString("Done"));
                else if (s[i].task == 5)
                item->setText(QString::fromStdString("Suspended"));
                else if (s[i].task == 6)
                item->setText(QString::fromStdString("Killed"));
                break;
                case 2:
                if (GetThreadPriority(aThread[i]) == THREAD_PRIORITY_IDLE)
                item->setText(QString::fromStdString("IDLE"));
                else if (GetThreadPriority(aThread[i]) == THREAD_PRIORITY_LOWEST)
                item->setText(QString::fromStdString("LOWEST"));
                else if (GetThreadPriority(aThread[i]) == THREAD_PRIORITY_BELOW_NORMAL)
                item->setText(QString::fromStdString("BELOW_NORMAL"));
                else if (GetThreadPriority(aThread[i]) == THREAD_PRIORITY_NORMAL)
                item->setText(QString::fromStdString("NORMAL"));
                else if (GetThreadPriority(aThread[i]) == THREAD_PRIORITY_ABOVE_NORMAL)
                item->setText(QString::fromStdString("ABOVE_NORMAL"));
                else if (GetThreadPriority(aThread[i]) == THREAD_PRIORITY_HIGHEST)
                item->setText(QString::fromStdString("HIGHEST"));
                break;
            }

            ui->tableWidget->setItem(i, j, item);
        }
    }
}

DWORD WINAPI ThreadProc(LPVOID lpParam) {
    PARAMETERS* p = (PARAMETERS*)lpParam;
    int i = p->i;
    int start, end;

    DWORD dwWaitResult;
    BOOL bContinue = TRUE;

    s[i].threadID = GetCurrentThreadId();

    while (bContinue) {
        dwWaitResult = WaitForSingleObject(
        ghSemaphore,
        0L);
```

```
        switch (dwWaitResult) {
            case WAIT_OBJECT_0:
            bContinue = FALSE;

            start = (N / threadCount) * i;
            end = (N / threadCount) * (i + 1);

            std::srand(static_cast<unsigned int>(std::time(nullptr)));
            s[i].task = 1;
            for (int j = start; j < end; j++)
            array[j] = rand();
            for (int j = start; j < end; j++) {
                InterlockedAdd(&sum, (LONG)array[j]);
            }
            s[i].task = 4;

            if (!ReleaseSemaphore(ghSemaphore, 1, NULL)) {
                qDebug() << "ReleaseSemaphore error: " << GetLastError();
            }
            break;

            case WAIT_TIMEOUT:
            s[i].task = 3;
            break;
        }
    }
    return TRUE;
}

void MainWindow::on_sus_activated(int index) {
    SuspendThread(aThread[index]);
    if (s[index].task != 6)
    s[index].task = 5;
}

void MainWindow::on_res_activated(int index) {
    ResumeThread(aThread[index]);
    if (s[index].task != 6 || s[index].task != 4 || s[index].task != 3)
    s[index].task = 1;
}

void MainWindow::on_kill_activated(int index) {
    TerminateThread(aThread[index], 0);
    s[index].task = 6;
}

void MainWindow::on_prio1_activated(int index) {
    if (ui->prio2->currentIndex() == 0)
    SetThreadPriority(aThread[index], THREAD_PRIORITY_IDLE);
    else if (ui->prio2->currentIndex() == 1)
    SetThreadPriority(aThread[index], THREAD_PRIORITY_LOWEST);
    else if (ui->prio2->currentIndex() == 1)
    SetThreadPriority(aThread[index], THREAD_PRIORITY_BELOW_NORMAL);
    else if (ui->prio2->currentIndex() == 1)
    SetThreadPriority(aThread[index], THREAD_PRIORITY_NORMAL);
    else if (ui->prio2->currentIndex() == 1)
    SetThreadPriority(aThread[index], THREAD_PRIORITY_ABOVE_NORMAL);
    else if (ui->prio2->currentIndex() == 1)
    SetThreadPriority(aThread[index], THREAD_PRIORITY_HIGHEST);
}
```

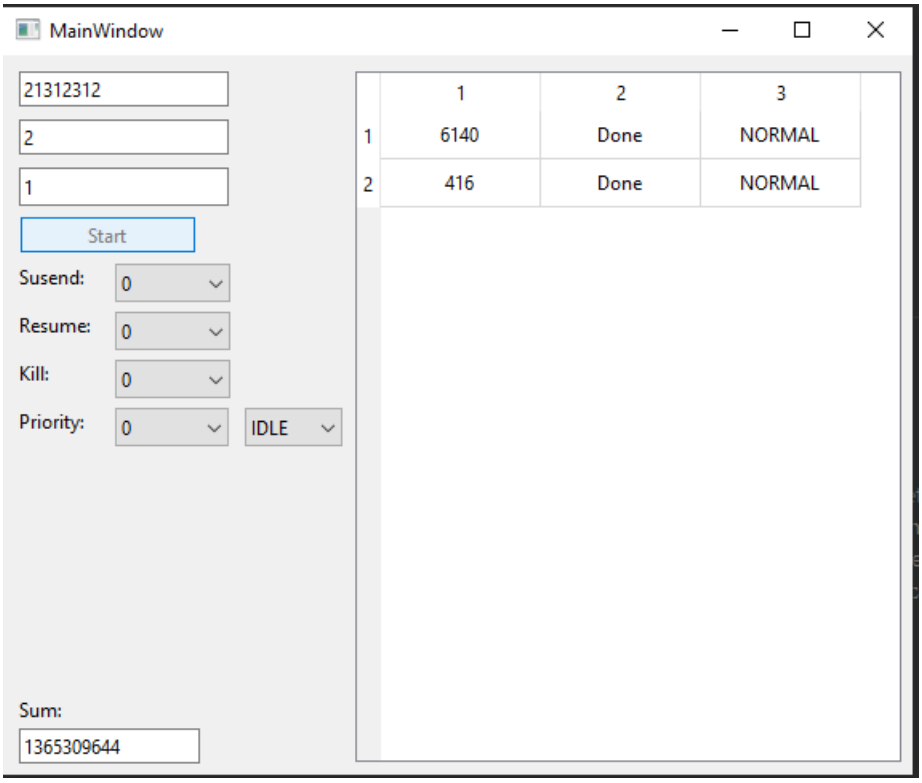

Рис. 1: Виконання програми

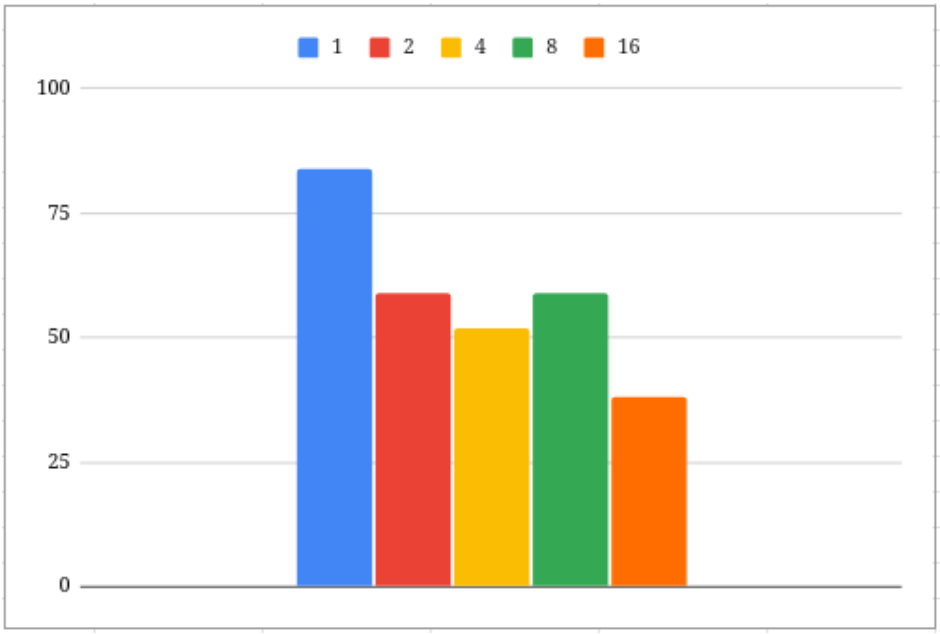

Рис. 2: Залежність часу виконання від кількості потоків

Висновок

Під час виконання лабораторної роботи я ознайомився з багатопоточністю в ОС Windows. Навчивсь реалізовувати розпаралелення алгоритмів за допомогою багатопоточності в ОС Windows з використанням функцій WinAPI. Навчивсь використовувати різні механізми синхронізації потоків