НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



**ЗВІТ**

**Про виконання лабораторної роботи № 5**

**з дисципліни «Операційні системи»**

**Лектор:**

Ст. викладач Грицай О.Д.

**Виконав:**

студ. групи ПЗ-23

Михалевич П.-І.В.

**Прийняла:**

Ст. викладач Грицай О.Д.

«\_\_\_» \_\_\_\_\_\_ 2021 р.

∑ = \_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_

Львів – 2021

**Тема.** Виконання задачі в декількох потоках. Створення та керування потоками в операційній системі WINDOWS.

**Мета.** Ознайомитися з багатопоточністю в ОС Windows. Навчитись реалізовувати розпаралелювання алгоритмів за допомогою багатопоточності в ОС Windows з використанням функцій WinAPI.

**Теоретичні відомості**

Розглядаючи поняття процесу, визначають ще одну абстракцію для

запущеного процесу: потік. У класичному уявленні існує єдина точка виконання в рамках програми (тобто єдиний потік контролю, на якому збираються та виконуються інструкції), багатопотокова програма має більш ніж одну точкувиконання (тобто кілька потоків контролю, кожен з яких який отримується та виконується).

Кожен потік дуже схожий на окремий процес, за винятком однієї відмінності:

вони мають спільний адресний простір і, отже, мають доступ до одних і тих же

даних. Таким чином, стан одного потоку дуже подібний до стану процесу. Він

має лічильник програм (PC), який відстежує, звідки програма отримує інструкції.

Кожен потік має свій власний приватний набір реєстрів, який він використовує

для обчислень; таким чином, якщо на одному процесорі працюють два потоки,

при переході від запуску одного (T1) до запуску іншого (T2) має відбутися

перемикання контексту. Контекстний перемикач між потоками дуже подібний до перемикання контекстів між процесами, оскільки перед запуском Т2 необхідно зберегти регістр стану Т1 і відновити стан реєстру Т2. За допомогою процесів ми зберегли стан до блоку управління процесами (PCB); тепер нам знадобиться один або кілька блоків управління потоками (TCB) для збереження стану кожного потоку процесу. Однак є одна істотна відмінність у перемиканні контексту, який ми виконуємо між потоками порівняно з процесами: адресний простір залишається незмінним (тобто немає необхідності змінювати, яку таблицю сторінок ми використовуємо).

Ще одна істотна відмінність між потоками та процесами стосується стека. У

простій моделі адресного простору класичного процесу (однопотокового) є

єдиний стек, який зазвичай знаходиться внизу адресного простору. Однак у

багатопотоковому процесі кожен потік працює окремо і, звичайно, може залучати різні підпрограми для виконання будь -якої роботи. Замість одного стека в адресному просторі буде по одному на кожен потік.

**Хід роботи**

**Завдання**

1. Реалізувати заданий алгоритм в окремому потоці.

2. Виконати розпаралелювання 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. Зобразити залежність час виконання – кількість потоків.

6. Результати виконання роботи відобразити у звіті.

**Індивідуальне завдання**

10. Обчислити суму елементів заданого масиву (кількість елементів >10000, елементи масиву задаються формулою a(0)=2, a(i)=a(i-1)\*i+exp(i), i -індекс елементу.

**Код програми:**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <vector>

#include <iostream>

#include <thread>

#include <math.h>

#include <ctime>

#include <windows.h>

#include <mutex>

#include <sstream>

#include <QMessageBox>

#define N 16

double **GetCpuTime**(HANDLE &pi){

FILETIME createTime;

FILETIME exitTime;

FILETIME kernelTime;

FILETIME userTime;

*if* ( GetThreadTimes( pi,

&createTime, &exitTime, &kernelTime, &userTime ) != -1 )

{

SYSTEMTIME userSystemTime;

*if* ( FileTimeToSystemTime( &userTime, &userSystemTime ) != -1 )

*return* (double)userSystemTime.wHour \* 3600.0 +

(double)userSystemTime.wMinute \* 60.0 +

(double)userSystemTime.wSecond +

(double)userSystemTime.wMilliseconds / 1000.0;

*else* *return* -2;

} *else* *return* -1;

}

std::mutex mu;

int sum = 0;

int myindex = 0;

std::vector<int> array(10000,0);

*//std::vector<std::thread>* *mythreads;*

std::vector<HANDLE> myhandle(N, 0);

std::vector<DWORD> mythreadid(N, 0);

MainWindow::**MainWindow**(QWidget \*parent)

: QMainWindow(*parent*)

, ui(*new* Ui::MainWindow)

{

ui->setupUi(*this*);

*for*(int i = 0; i < ui->tableWidget->rowCount(); i++)

{

*for*(int j = 0; j < ui->tableWidget->columnCount(); j++)

{

QTableWidgetItem\* item = *new* QTableWidgetItem;

item->setText("");

item->setTextAlignment(Qt::*AlignCenter*);

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

}

}

connect(ui->pushButton\_start,SIGNAL(clicked()),*this*,SLOT(MySlot()));

connect(ui->pushButton\_suspend,SIGNAL(clicked()),*this*,SLOT(MySlot()));

connect(ui->pushButton\_resume,SIGNAL(clicked()),*this*,SLOT(MySlot()));

connect(ui->pushButton\_kill,SIGNAL(clicked()),*this*,SLOT(MySlot()));

connect(ui->pushButton\_killall,SIGNAL(clicked()),*this*,SLOT(MySlot()));

timer = *new* QTimer();

connect(timer,SIGNAL(timeout()),*this*,SLOT(MyTimer()));

timer->start(100);

}

MainWindow::~***MainWindow***()

{

*for*(size\_t i = 0; i < std::size(myhandle); i++){

DWORD dwCode;

GetExitCodeThread(myhandle[i],&dwCode);

*if*(dwCode == STILL\_ACTIVE){

WaitForSingleObject(myhandle[i], INFINITE);

CloseHandle(myhandle[i]);

}

}

*delete* ui;

}

void MainWindow::**MyTimer**(){

*for*(int i = 0; i<ui->comboBox\_thread\_count->currentText().toInt();i++){

*if*(ui->tableWidget->item(i,0)->text()!=""){

ui->tableWidget->item(i,3)->setText(QString::number(GetCpuTime(*myhandle[i]*)));

DWORD dwCode;

GetExitCodeThread(myhandle[i],&dwCode);

*if*(dwCode != STILL\_ACTIVE && ui->tableWidget->item(i,1)->text() != "Killed"){

ui->tableWidget->item(i,1)->setText("Killed");

}

}

}

}

void **ArraySum**(int\* param)

{

Sleep(1000);

system("pause");

*for*(int i = param[0]; i< param[1]; i++){

*if*(i == 0){

array[i] = 2;

sum+=array[i];

*continue*;

}

array[i] = array[i-1]\*i + exp(i);

*//sum+=array[i];*

sum+=1;

std::cout<<myindex++<< " / " << std::endl;

}

std::cout<<"In current thread ("<<GetCurrentThreadId()<<"): sum =" << sum << std::endl;

}

void **MutexArraySum**(int\* param)

{

mu.lock();

Sleep(1000);

system("pause");

*for*(int i = param[0]; i< param[1]; i++){

*if*(i == 0){

array[i] = 2;

sum+=array[i];

*continue*;

}

array[i] = array[i-1]\*i + exp(i);

*//sum+=array[i];*

sum+=1;

std::cout<<myindex++<< " // " << std::endl;

}

std::cout<<"In current thread ("<<GetCurrentThreadId()<<"): sum =" << sum << std::endl;

mu.unlock();

}

void MainWindow::**MySlot**(){

QPushButton\* btn = (QPushButton\*) sender();

*if*(btn->text() == "Start"){

*for*(size\_t i = 0; i < std::size(myhandle); i++){

*if*(myhandle[i] != 0){

DWORD dwCode;

GetExitCodeThread(myhandle[i],&dwCode);

*if*(dwCode == STILL\_ACTIVE){

WaitForSingleObject(myhandle[i], INFINITE);

CloseHandle(myhandle[i]);

}

}

}

*for*(int i = 0; i < ui->tableWidget->rowCount(); i++)

{

*for*(int j = 0; j < ui->tableWidget->columnCount(); j++)

{

QTableWidgetItem\* item = *new* QTableWidgetItem;

item->setText("");

item->setTextAlignment(Qt::*AlignCenter*);

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

}

}

*/\*std::thread* *array\_thread[ui->comboBox\_thread\_count->currentText().toInt()];*

*for* *(int* *i* *=* *0;* *i<ui->comboBox\_thread\_count->currentText().toInt();* *i++)* *{*

*array\_thread[i]* *=* *std::thread(ArraySum,(i)\*(ui->spinBox\_array\_size->value()/ui->comboBox\_thread\_count->currentText().toInt()),*

*(i+1)\*(ui->spinBox\_array\_size->value()/ui->comboBox\_thread\_count->currentText().toInt()));*

*std::stringstream* *ss;*

*ss* *<<* *array\_thread[i].get\_id();*

*std::string* *mystring* *=* *ss.str();*

*ui->tableWidget->item(i,0)->setText(QString::fromStdString(mystring));*

*}*

*for* *(int* *i* *=* *0;* *i<ui->comboBox\_thread\_count->currentText().toInt();* *i++)* *{*

*array\_thread[i].join();*

*}*

*std::cout* *<<* *sum* *<<* *std::endl;\*/*

*for*(int i = 0; i<ui->comboBox\_thread\_count->currentText().toInt(); i++){

int\* param = *new* int[2];

param[0] = (i)\*(ui->spinBox\_array\_size->value()/ui->comboBox\_thread\_count->currentText().toInt());

param[1] = (i+1)\*(ui->spinBox\_array\_size->value()/ui->comboBox\_thread\_count->currentText().toInt());

*if*(ui->checkBox->isChecked()){

myhandle[i] = CreateThread(0, 0, (LPTHREAD\_START\_ROUTINE)&MutexArraySum, param, 0, &mythreadid[i]);

}*else*{

myhandle[i] = CreateThread(0, 0, (LPTHREAD\_START\_ROUTINE)&ArraySum, param, 0, &mythreadid[i]);

}

std::stringstream ss;

ss << mythreadid[i];

std::string mystring = ss.str();

ui->tableWidget->item(i,0)->setText(QString::fromStdString(mystring));

ui->tableWidget->item(i,1)->setText("Running");

ui->tableWidget->item(i,2)->setText("Normal");

}

}*else* *if*(btn->text() == "Suspend"){

*if*(myhandle[ui->tableWidget->currentRow()] != 0){

DWORD dwCode;

GetExitCodeThread(myhandle[ui->tableWidget->currentRow()],&dwCode);

*if*(ui->tableWidget->item(ui->tableWidget->currentRow(),1)->text() != "Suspended" &&

ui->tableWidget->item(ui->tableWidget->currentRow(),1)->text() != "Killed" &&

dwCode == STILL\_ACTIVE){

SuspendThread(myhandle[ui->tableWidget->currentRow()]);

ui->tableWidget->item(ui->tableWidget->currentRow(),1)->setText("Suspended");

ui->tableWidget->item(ui->tableWidget->currentRow(),3)->setText(QString::number(GetCpuTime(*myhandle[ui->tableWidget->currentRow()]*)));

}*else*{

*if*(dwCode != STILL\_ACTIVE){

ui->tableWidget->item(ui->tableWidget->currentRow(),1)->setText("Killed");

ui->tableWidget->item(ui->tableWidget->currentRow(),3)->setText(QString::number(GetCpuTime(*myhandle[ui->tableWidget->currentRow()]*)));

}

QMessageBox::information(*this*,"Info","You can't suspend the process!");

}

}

}*else* *if*(btn->text() == "Resume"){

*if*(myhandle[ui->tableWidget->currentRow()] != 0){

DWORD dwCode;

GetExitCodeThread(myhandle[ui->tableWidget->currentRow()],&dwCode);

*if*(ui->tableWidget->item(ui->tableWidget->currentRow(),1)->text() != "Running" &&

ui->tableWidget->item(ui->tableWidget->currentRow(),1)->text() != "Killed" &&

dwCode == STILL\_ACTIVE){

ResumeThread(myhandle[ui->tableWidget->currentRow()]);

ui->tableWidget->item(ui->tableWidget->currentRow(),1)->setText("Running");

ui->tableWidget->item(ui->tableWidget->currentRow(),3)->setText(QString::number(GetCpuTime(*myhandle[ui->tableWidget->currentRow()]*)));

}*else*{

*if*(dwCode != STILL\_ACTIVE){

ui->tableWidget->item(ui->tableWidget->currentRow(),1)->setText("Killed");

ui->tableWidget->item(ui->tableWidget->currentRow(),3)->setText(QString::number(GetCpuTime(*myhandle[ui->tableWidget->currentRow()]*)));

}

QMessageBox::information(*this*,"Info","You can't resume the process!");

}

}

}*else* *if*(btn->text() == "Kill"){

*if*(myhandle[ui->tableWidget->currentRow()] != 0){

DWORD dwCode;

GetExitCodeThread(myhandle[ui->tableWidget->currentRow()],&dwCode);

*if*(ui->tableWidget->item(ui->tableWidget->currentRow(),1)->text() != "Killed" &&

dwCode == STILL\_ACTIVE){

TerminateThread(myhandle[ui->tableWidget->currentRow()], 0);

ui->tableWidget->item(ui->tableWidget->currentRow(),1)->setText("Killed");

ui->tableWidget->item(ui->tableWidget->currentRow(),3)->setText(QString::number(GetCpuTime(*myhandle[ui->tableWidget->currentRow()]*)));

}*else*{

*if*(dwCode != STILL\_ACTIVE){

ui->tableWidget->item(ui->tableWidget->currentRow(),1)->setText("Killed");

ui->tableWidget->item(ui->tableWidget->currentRow(),3)->setText(QString::number(GetCpuTime(*myhandle[ui->tableWidget->currentRow()]*)));

}

QMessageBox::information(*this*,"Info","The process has been killed yet!");

}

}

}*else* *if*(btn->text() == "Kill all"){

*for*(int i = 0; i < ui->tableWidget->rowCount(); i++){

*if*(myhandle[i] != 0){

DWORD dwCode;

GetExitCodeThread(myhandle[i],&dwCode);

*if*(ui->tableWidget->item(i,1)->text() != "Killed" &&

dwCode == STILL\_ACTIVE){

TerminateThread(myhandle[i], 0);

ui->tableWidget->item(i,1)->setText("Killed");

ui->tableWidget->item(i,3)->setText(QString::number(GetCpuTime(*myhandle[i]*)));

}*else*{

*if*(dwCode != STILL\_ACTIVE){

ui->tableWidget->item(i,1)->setText("Killed");

ui->tableWidget->item(i,3)->setText(QString::number(GetCpuTime(*myhandle[i]*)));

}

}

}

}

}

}

void MainWindow::**on\_comboBox\_thread\_count\_currentTextChanged**(*const* QString &arg1)

{

*//ui->tableWidget->setRowCount(arg1.toInt());*

*for*(size\_t i = 0; i < std::size(myhandle); i++){

*if*(myhandle[i] != 0){

DWORD dwCode;

GetExitCodeThread(myhandle[i],&dwCode);

*if*(dwCode == STILL\_ACTIVE){

WaitForSingleObject(myhandle[i], INFINITE);

CloseHandle(myhandle[i]);

}

}

}

}

void MainWindow::**on\_comboBox\_priority\_currentTextChanged**(*const* QString &arg1)

{

*switch*(ui->comboBox\_priority->currentIndex()){

*case* 0:

SetThreadPriority(myhandle[ui->tableWidget->currentRow()],THREAD\_PRIORITY\_TIME\_CRITICAL);

ui->tableWidget->item(ui->tableWidget->currentRow(),2)->setText(ui->comboBox\_priority->currentText());

ui->tableWidget->item(ui->tableWidget->currentRow(),3)->setText(QString::number(GetCpuTime(*myhandle[ui->tableWidget->currentRow()]*)));

*break*;

*case* 1:

SetThreadPriority(myhandle[ui->tableWidget->currentRow()],THREAD\_PRIORITY\_HIGHEST);

ui->tableWidget->item(ui->tableWidget->currentRow(),2)->setText(ui->comboBox\_priority->currentText());

ui->tableWidget->item(ui->tableWidget->currentRow(),3)->setText(QString::number(GetCpuTime(*myhandle[ui->tableWidget->currentRow()]*)));

*break*;

*case* 2:

SetThreadPriority(myhandle[ui->tableWidget->currentRow()],THREAD\_PRIORITY\_ABOVE\_NORMAL);

ui->tableWidget->item(ui->tableWidget->currentRow(),2)->setText(ui->comboBox\_priority->currentText());

ui->tableWidget->item(ui->tableWidget->currentRow(),3)->setText(QString::number(GetCpuTime(*myhandle[ui->tableWidget->currentRow()]*)));

*break*;

*case* 3:

SetThreadPriority(myhandle[ui->tableWidget->currentRow()],THREAD\_PRIORITY\_NORMAL);

ui->tableWidget->item(ui->tableWidget->currentRow(),2)->setText(ui->comboBox\_priority->currentText());

ui->tableWidget->item(ui->tableWidget->currentRow(),3)->setText(QString::number(GetCpuTime(*myhandle[ui->tableWidget->currentRow()]*)));

*break*;

*case* 4:

SetThreadPriority(myhandle[ui->tableWidget->currentRow()],THREAD\_PRIORITY\_BELOW\_NORMAL);

ui->tableWidget->item(ui->tableWidget->currentRow(),2)->setText(ui->comboBox\_priority->currentText());

ui->tableWidget->item(ui->tableWidget->currentRow(),3)->setText(QString::number(GetCpuTime(*myhandle[ui->tableWidget->currentRow()]*)));

*break*;

*case* 5:

SetThreadPriority(myhandle[ui->tableWidget->currentRow()],THREAD\_PRIORITY\_IDLE);

ui->tableWidget->item(ui->tableWidget->currentRow(),2)->setText(ui->comboBox\_priority->currentText());

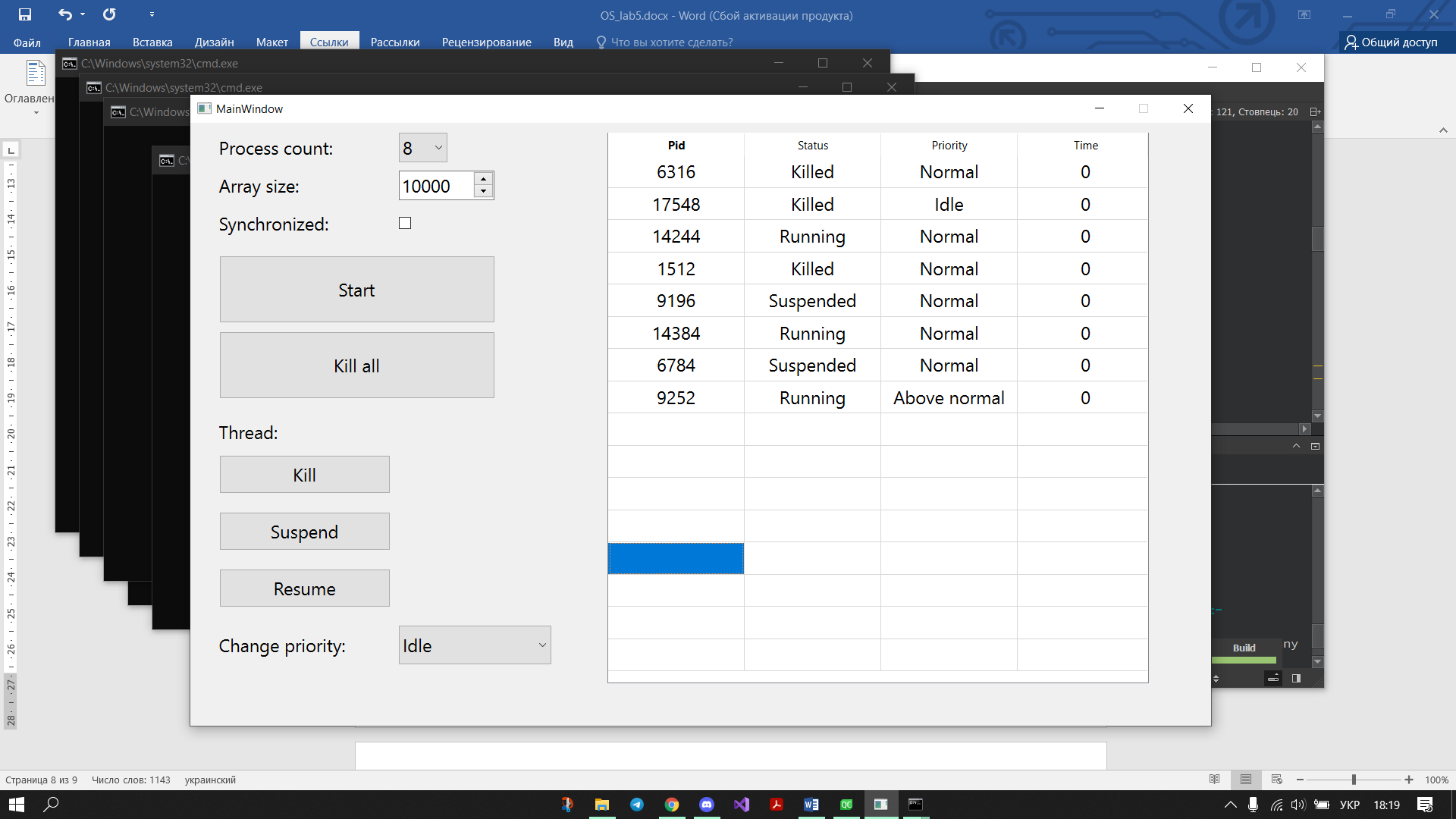
ui->tableWidget->item(ui->tableWidget->currentRow(),3)->setText(QString::number(GetCpuTime(*myhandle[ui->tableWidget->currentRow()]*)));

*break*;

}

}

**Результат програми:**



##### Висновок

На даній лабораторній роботі я навчився працювати з процесами в ОС Linux з використанням функцій API.

За допомогою функції CreateThread я реалізував індивідуальне завдання у 2, 4, 8 і 16 пoтоках. Також навчився використовувати такі функції, як SeThreadPriority, TerminateThread, ResumeThread, GetExitCodeThread, SuspendThread.