НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



**ЗВІТ**

**Про виконання лабораторної роботи № 6**

**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.** Синхронізація потоків в операційній системі WINDOWS.

**Мета.** Ознайомитися зі способами синхронізації потоків. Навчитися організовувати багатопоточність з використанням синхронізації за допомогою функцій WinAPI.

**Теоретичні відомості**

Існує три основні проблеми організації міжпоточної взаємодії: 1. Обмін даними. 2. Доступ до загальних ресурсів. 3. Узгодження дій. Розглянемо міжпоточної взаємодія на простому прикладі: спулер. Коли потоку потрібно надрукувати файл, він поміщає його ім'я до спеціального каталогу спулера. Інший потік - демон друку періодично перевіряє наявність файлів, друкує їх і видаляє імена з каталогу. Нехай каталог спулера складається з великої кількості сегментів, пронумерованих 0, 1, 2,:, в кожному з яких може зберігатися ім'я файлу. Також існує спільно використовувана змінна in, яка вказує на наступний вільний сегмент. Припустимо, сегменти з 0 по 6 зайняті - файли чекають своєї черги на друк, отже in = 7. Більш-менш одночасно два потоку А і В вирішили поставити свої файли в чергу на друк. Можливе виникнення наступної ситуації. Потік А зчитує значення in (7) у своїй локальній змінній, після чого закінчується його квант часу і операційна система перемикає процесор на виконання потоку В. В свою чергу, потік В теж зчитує і локально зберігає значення змінної in, теж рівне 7, тобто обидва потоки вважають, що вільний сегмент - 7. Потік В зберігає в 7 сегменті спулера ім'я файлу, і змінює значення змінної in на 8, після чого продовжує займатися своїми справами, не пов'язаними з друком. Нарешті управління переходить до потоку А, і він продовжує з того місця, на якому зупинився. Потік А зберігає ім'я свого файлу в сегменті спулера, з номером, збереженому в локальній змінній, тобто теж в сьомому сегменті. Природно, заміщаючи ім'я файлу, збереженого потоком В. Потім, потік збільшує на одиницю значення своєї внутрішньої змінної (7 + 1 = 8) і записує в змінну in значення 8. Структура каталогу спулера не порушена, тому демон друку нічого не запідозрить, однак файл, відправлений на друк потоком В ніколи надрукований не буде. Ситуація, при якій кілька потоків борються за доступ до загального ресурсу, а результат залежить від того, хто цей доступ отримав першим, називається станом перегонів (race condition). Налагодження програм з такими ситуаціями складна, оскільки більшість прогонів будуть хорошими, але зрідка будуть відбуватися дивні й незрозумілі речі.

**Хід роботи**

**Завдання**

1. Реалізувати заданий алгоритм в кількох (2, 4, 8, 16 ) потоках.

   2. Використати два різні методи для синхронізації потоків із наведених у переліку:

      а) м'ютекс, критична секція, wait-функції

      б) семафор, спінблокування, події

   3. Реалізувати можливість завершення потоку

   4. Зобразити залежність час виконання – кількість потоків.

   5. Порівняти виконання задачі відносно методів синхронізації

   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 <chrono>

#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;

HANDLE semaphore;

HANDLE WinMutex;

double time\_threads = 0;

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()));

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

connect(ui->pushButton\_get\_sum,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]);

}

}

CloseHandle(semaphore);

CloseHandle(WinMutex);

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");

}

}

}

ui->lineEdit->setText(QString::number(time\_threads,'f',6)+ " ms" );

}

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

{

//Sleep(1000);

//system("pause");

Sleep(10000);

const auto start = std::chrono::high\_resolution\_clock::now();

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;

const auto end = std::chrono::high\_resolution\_clock::now();

time\_threads += std::chrono::duration\_cast<std::chrono::nanoseconds>(end - start).count() \* 1e-6;

}

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

{

mu.lock();

Sleep(1000);

const auto start = std::chrono::high\_resolution\_clock::now();

//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()<<")(mutex): sum =" << sum << std::endl;

const auto end = std::chrono::high\_resolution\_clock::now();

time\_threads += std::chrono::duration\_cast<std::chrono::nanoseconds>(end - start).count() \* 1e-6;

mu.unlock();

}

void **SemaphorArraySum**(int\* param)

{

WaitForSingleObject(semaphore, INFINITE);

Sleep(10000);

const auto start = std::chrono::high\_resolution\_clock::now();

//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()<<")(semaphor): sum =" << sum << std::endl;

const auto end = std::chrono::high\_resolution\_clock::now();

time\_threads += std::chrono::duration\_cast<std::chrono::nanoseconds>(end - start).count() \* 1e-6;

ReleaseSemaphore(semaphore, 1, nullptr);

}

void **WinMutexArraySum**(int\* param)

{

WaitForSingleObject(WinMutex, INFINITE);

Sleep(1000);

const auto start = std::chrono::high\_resolution\_clock::now();

//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()<<")(semaphor): sum =" << sum << std::endl;

const auto end = std::chrono::high\_resolution\_clock::now();

time\_threads += std::chrono::duration\_cast<std::chrono::nanoseconds>(end - start).count() \* 1e-6;

ReleaseMutex(WinMutex);

}

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

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

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

sum = 0;

myindex = 0;

time\_threads = 0;

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;\*/

semaphore = CreateSemaphore(0, 1, 1, nullptr);

WinMutex = CreateMutex(nullptr, FALSE, nullptr);

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

int\* param = new int[2];

array.resize(ui->spinBox\_array\_size->value());

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->radioButton->isChecked()){

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

}else if(ui->radioButton\_semaphor->isChecked()){

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

}else if(ui->radioButton\_winMutex->isChecked()){

myhandle[i] = CreateThread(0, 0, (LPTHREAD\_START\_ROUTINE)&SemaphorArraySum, 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]*)));

}

}

}

}

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

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

{

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

{

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

}

}

}else if(btn->text() == "Get Sum"){

std::cout << "\nGeneral sum = " << sum << std::endl;

}

}

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());

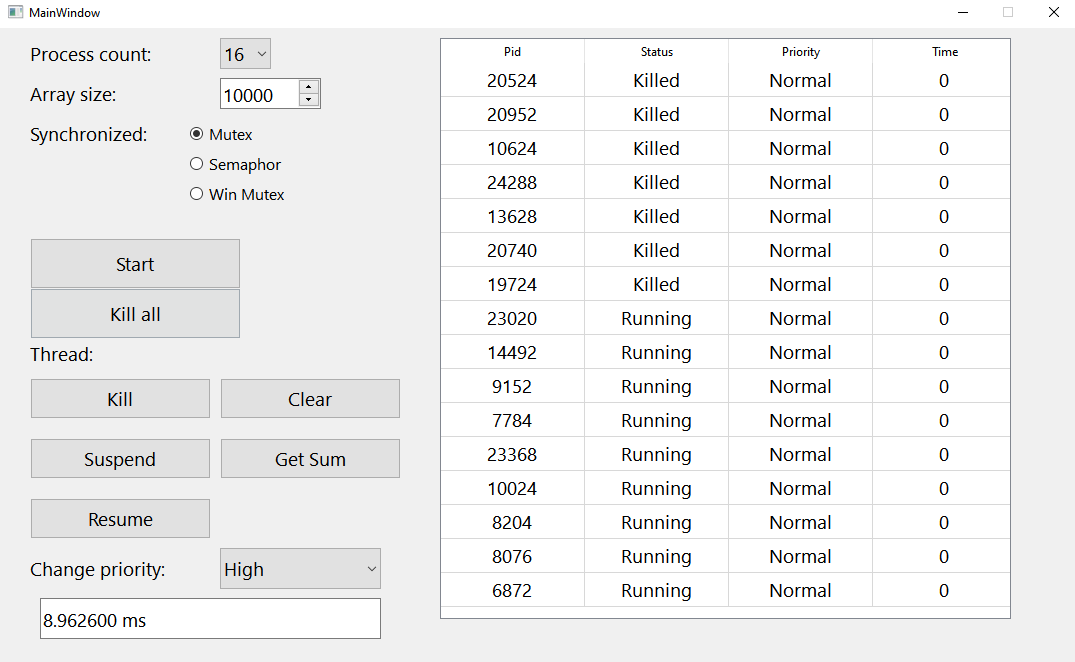
ui->tableWidget->item(ui->tableWidget->currentRow(),3)->setText(QString::number(GetCpuTime(*myhandle[ui->tableWidget->currentRow()]*)));

break;

}

}

**Результат програми:**

****

##### Висновок

На даній лабораторній роботі я навчився працювати з синхронізацією потоків в ОС Windows з використанням функцій API. Використав такі методи синхронізації, як Wait-функції, мютекс і семафор.