НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



**ЗВІТ**

**Про виконання лабораторної роботи № 7**

**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 LINUX.

**Мета.** Навчитись реалізовувати розпаралелювання алгоритмів за допомогою багатопоточності в ОС Linux з використанням пакету функцій pthread

**Теоретичні відомості**

У ранніх версіях ядра Linux підтримка потоків на рівні системних викликів відбувалась з допомогою системного виклику clone(). Виклик clone(), як і fork() дозволяє створювати новий процес, але вже з повним копіюванням батьківського процесу одразу. Крім того, для нового процесу у цьому виклику створюється власний стек, вказуються прапорці про наслідування. Фактично, тоді створюється нова гілка у процесі, реалізуючи потік користувача, який у ядрі буде відображатися у процес ядра. Послідовність інструкцій ядра, з яким працює процесор, називається не потоками, а процесами. ВІдображення потоків користувача у процеси ядра при цьому відбувається за моделлю 1:1. Пряме використання clone() є складним, так як вимагає задання певних полів у структурі процесу, що в свою чергу може призвести до непередбачуваної ситуації. Крім того, даний системний виклик не підтримується POSIX. У сучасних версіях linux у ядрі на основі розширених можливостей системного виклику clone() підтримуються спеціальні об'єкти ядра - потоки ядра. Підтримка потоків здійснюється через бібліотеку NPTL (Native POSIX Threads Library), що підтримує програмний інтерфейс потоків POSIX. Типи даних і функції, що застосовуються до потоків POSIX, мають префікс pthread\_ і доступні через підключення заголовочного файлу phread.h Для компіляції програми, що використовує потоки необхідно вказати, що ми використовуємо бібліотеку pthread gcc -o main main.c -Wall -pthread;

./main

Для створення потоку використовують функцію phread\_create():

int pthread\_create(pthead\_t \*thread, pthrad\_attr\_t \*attr, void\*(\*thread\_func)(void \*), void \*arg);

thread - вказівник на структуру pthead\_t, дескриптор потоку;

attr - вказівник на структуру атрибутів (розмір стеку, пріоритет, ...)

thread\_func - вказівник на функцію потоку void \*mythread\_fun(void \*value){ }

arg- дані, що передаються у функцію

Наявність вказівникa void як аргументу для функції потоку дозволяє передавати аргументи будь -якого типу; маючи вказівник void як повернене значення, дозволяє потоку повертати будь -який результат. Якщо функція вимагає int аргументу

int pthread\_create(..., void\*(\*start\_routine)(int), int arg);

Якщо функція повертає int

int pthread\_create(..., int(\*start\_routine)(void\*), void \* arg);

**Хід роботи**

**Завдання**

1. Реалізувати заданий алгоритм в окремому потоці.

2. Виконати розпаралелювання заданого алгоритму на 2, 4, 8, 16 потоків.

3. Реалізувати можливість зміни пріоритету певного потоку.

4. Реалізувати можливість завершення потоку

5. Зобразити залежність час виконання – кількість потоків. Порівняти з отриманими результатами в ОС Windows

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 <algorithm>

#include <pthread.h>

#include <unistd.h>

#include <math.h>

#include <signal.h>

#include <ctime>

#include <chrono>

#include <mutex>

#include <sys/types.h>

#include <sstream>

#include <QMessageBox>

#include <semaphore.h>

#define N 16

pthread\_mutex\_t mu;

sem\_t semaphor;

double time\_threads = 0;

int sum = 0;

int myindex = 0;

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

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

std::vector<pthread\_t> pthread(N,0);

struct MyParam{

int first;

int second;

};

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()));

timer = new QTimer();

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

timer->start(100);

pthread\_mutex\_init(&mu,nullptr);

sem\_init(&semaphor,0,1);

}

MainWindow::~MainWindow()

{

for(size\_t i = 0; i < pthread.size(); i++){

if(pthread[i] == 0){

continue;

}

pthread\_join(pthread[i], nullptr);

pthread[i] = 0;

}

delete ui;

pthread\_mutex\_destroy(&mu);

sem\_destroy(&semaphor);

}

void MainWindow::MyTimer(){

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

}

void\* ArraySum(void \* data)

{

auto param = \*(MyParam\*)(data);

//sleep(10);

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

for(int i = param.first; i< param.second; i++){

if(i == 0){

array[i] = 2;

sum+=array[i];

continue;

}

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

//sum+=array[i];

//pthread\_mutex\_lock(&mu);

sum+=1;

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

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

//pthread\_mutex\_unlock(&mu);

}

//std::stringstream ss;

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

//textEdit->append(QString::fromStdString(ss.str()));

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

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

return nullptr;

}

void \*MutexArraySum(void \* data)

{

auto param = \*(MyParam\*)(data);

//sleep(10);

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

for(int i = param.first; i< param.second; i++){

if(i == 0){

array[i] = 2;

sum+=array[i];

continue;

}

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

//sum+=array[i];

pthread\_mutex\_lock(&mu);

sum+=1;

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

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

pthread\_mutex\_unlock(&mu);

}

//std::stringstream ss;

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

//textEdit->append(QString::fromStdString(ss.str()));

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

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

return nullptr;

}

void \*SemaphorArraySum(void \* data)

{

auto param = \*(MyParam\*)(data);

//sleep(10);

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

for(int i = param.first; i< param.second; i++){

if(i == 0){

array[i] = 2;

sum+=array[i];

continue;

}

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

//sum+=array[i];

sem\_wait(&semaphor);

sum+=1;

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

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

sem\_post(&semaphor);

}

//std::stringstream ss;

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

//textEdit->append(QString::fromStdString(ss.str()));

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

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

return nullptr;

}

void MainWindow::MySlot(){

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

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

connect(timer,SIGNAL(timeout()),this,SLOT(CheckFinish()));

sum = 0;

myindex = 0;

time\_threads = 0;

for(size\_t i = 0; i < pthread.size(); i++){

if(pthread[i] == 0){

continue;

}

pthread\_join(pthread[i], nullptr);

pthread[i] = 0;

}

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);

}

}

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

MyParam\* param = new MyParam;

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

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

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

if(ui->radioButton\_mut->isChecked()){

pthread\_create(&pthread[i], nullptr, MutexArraySum, (void\*)param);

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

pthread\_create(&pthread[i], nullptr, SemaphorArraySum, (void\*)param);

}else{

pthread\_create(&pthread[i], nullptr, ArraySum, (void\*)param);

}

//pthread\_kill(pthread[i],SIGSTOP);

//}

std::stringstream ss;

//pthread\_id\_np\_t t;

//pthread\_getunique\_np(&pthread[i],&t);

ss << pthread[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"){

pthread\_kill(pthread[ui->tableWidget->currentRow()],SIGSTOP);

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

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

pthread\_kill(pthread[ui->tableWidget->currentRow()],SIGCONT);

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

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

pthread\_kill(pthread[ui->tableWidget->currentRow()],SIGKILL);

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

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

}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("");

}

}

}

}

void MainWindow::CheckFinish()

{

for(size\_t i = 0; i < pthread.size(); i++){

if(pthread[i] == 0){

continue;

}

pthread\_join(pthread[i], nullptr);

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

pthread[i] = 0;

}

}

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

{

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

for(size\_t i = 0; i < pthread.size(); i++){

if(pthread[i] == 0){

continue;

}

pthread\_join(pthread[i], nullptr);

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

pthread[i] = 0;

}

}

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

{

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

case 0:

pthread\_setschedprio(pthread[ui->tableWidget->currentRow()], 19);

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

break;

case 1:

pthread\_setschedprio(pthread[ui->tableWidget->currentRow()], 10);

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

break;

case 2:

pthread\_setschedprio(pthread[ui->tableWidget->currentRow()], 5);

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

break;

case 3:

pthread\_setschedprio(pthread[ui->tableWidget->currentRow()], 0);

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

break;

case 4:

pthread\_setschedprio(pthread[ui->tableWidget->currentRow()], -5);

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

break;

case 5:

pthread\_setschedprio(pthread[ui->tableWidget->currentRow()], -10);

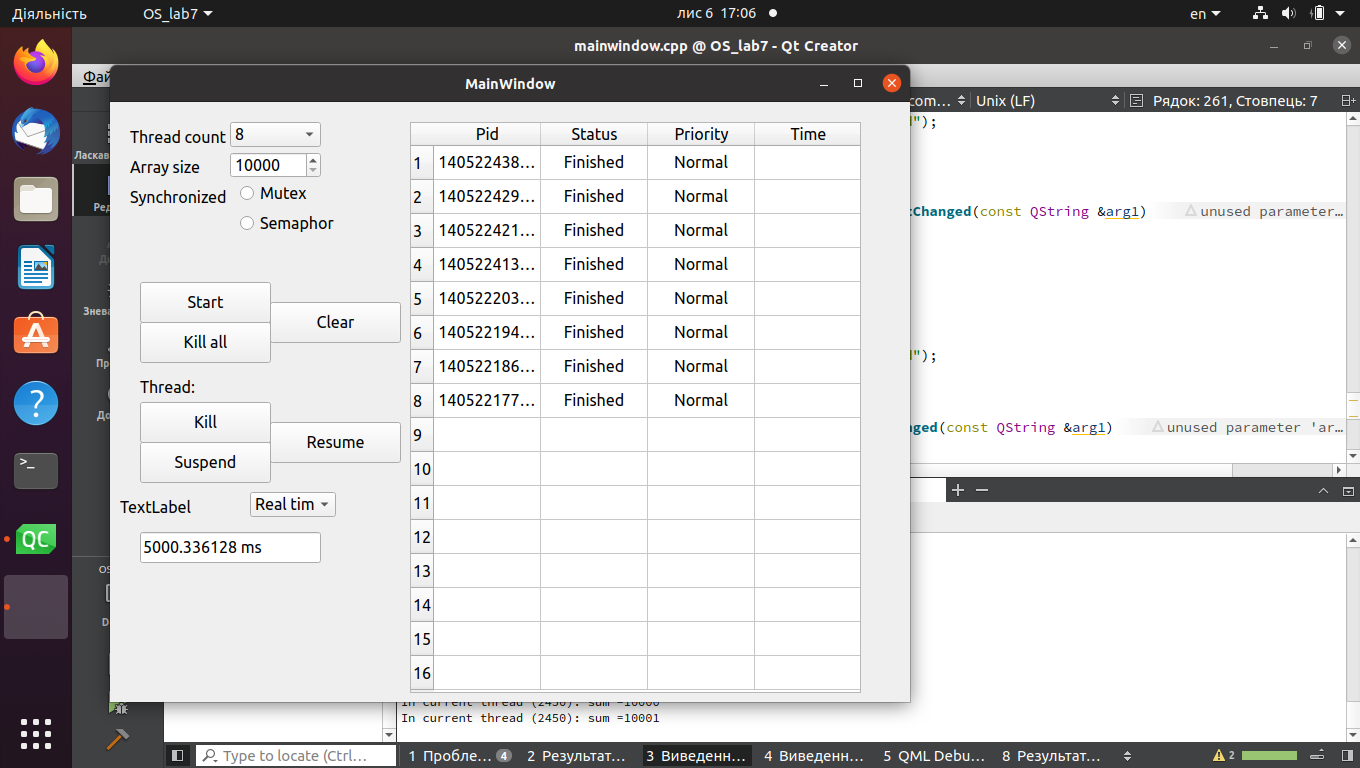
ui->tableWidget->item(ui->tableWidget->currentRow(),2)->setText(ui->comboBox\_priority->currentText());

break;

}

}

**Результат програми:**

****

##### Висновок

На цій лабораторні роботі я навчивсь реалізовувати розпаралелювання алгоритмів за допомогою багатопоточності в ОС Linux з використанням пакету функцій pthread. Навчився працювати з потоками в ОС Linux. Створив програму, для посимвольного виведення даних.