НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



**ЗВІТ**

**Про виконання лабораторної роботи № 8**

**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 LINUX.

**Мета.** Ознайомитися з особливостями синхронізації потоків в ОС Linux. Навчитися організовувати багатопоточність з використанням синхронізації в ОС Linux.

**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, умовна змінна, бар'єри

      б) семафор, спінблокування, [монітори](https://vns.lpnu.ua/mod/page/view.php?id=74781)

   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 <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);

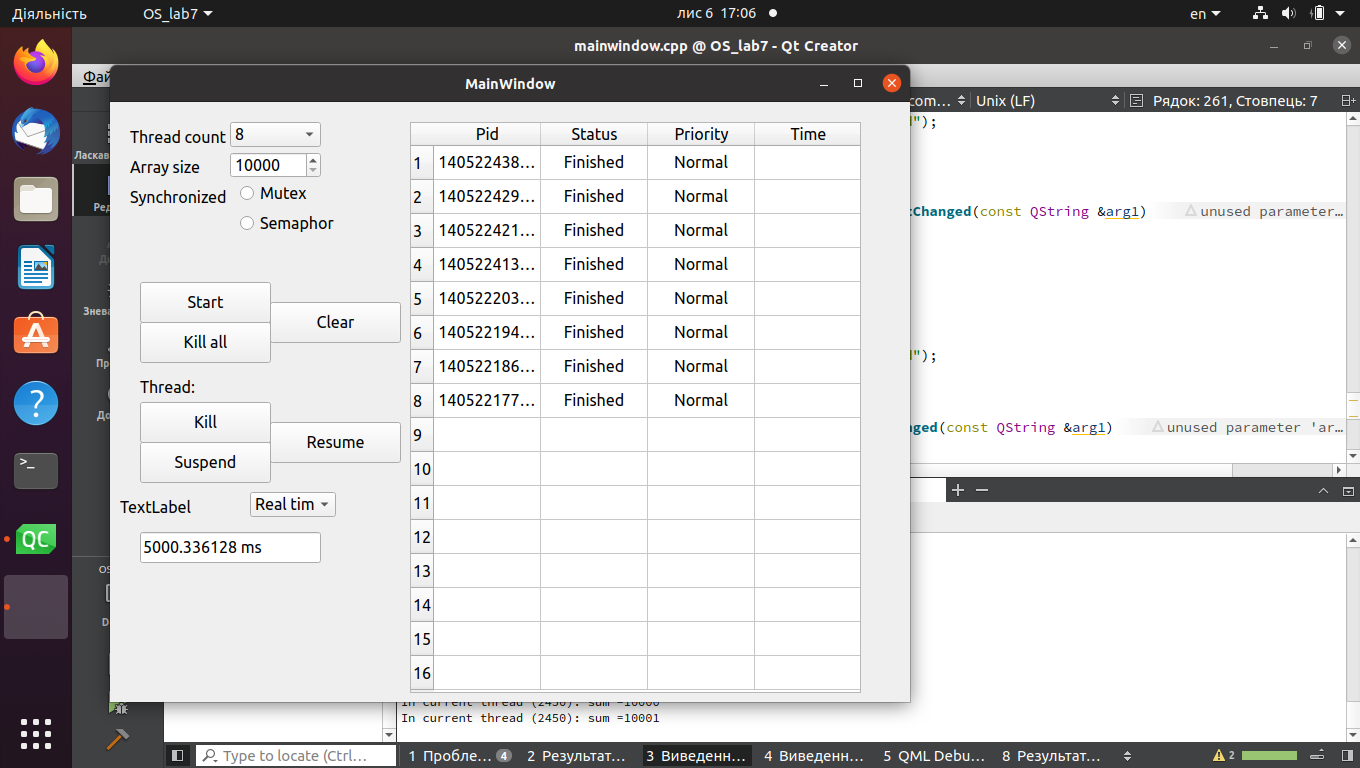
ui->tableWidget->item(ui->tableWidget->currentRow(),2)->setText(ui->comboBox\_priority->currentText());

break;

}

}

**Результат програми:**

****

##### Висновок

На даній лабораторні роботі я ознайомився з синхронізацією процесів в ОС Linux. Навчився реалізувати методи синхронізаці mutex і semaphore. В порівнянні з ОС Windows, ці методи мають аналогічні способи реалізації. За допомогою цих методів здійснив синхронізацію потоків у програмі реалізації завдання.