МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ



**Дніпровський національний університет  
залізничного транспорту імені академіка В. Лазаряна**

Кафедра «Комп’ютерні інформаційні технології»

**Лабораторна робота №1**

**з дисципліни «Операційні системи»**

**на тему: «Створення потоків і їх синхронізація в режимі користувача»**

Виконав: студент гр. ПЗ1911

Сіньков Г.О.

Прийняла: ас. каф. КІТ

Адрющенко В. О.

Дніпро, 2020

**Лабораторна робота №1**

**Тема.** Створення потоків і їх синхронізація в режимі користувача.

**Завдання.** Розробити програму, яка створює два або більше потоки керування (системний виклик CreateThread), що отримують доступ до спільних ресурсів за принципом взаємовиключного доступу. Для синхронізації застосовувати системні структури CRITICAL\_SECTION. Характеристики ресурсів і спосіб їх використання потоками відповідає індивідуальному завданню.

**Варіант 12.**

Характеристики ресурсу: 4) Стек на основі динамічного списку.

Спосіб використання ресурсу: 1) Потоки першого виду додають інформацію, потоки другого виду видаляють інформацію.

**Текст програми**

main.cpp

#include <windows.h>

#include <process.h>

#include <iostream>

#include <conio.h>

#include <random>

#include "nodeStack.h"

// Stack based on a dynamic list

CRITICAL\_SECTION critical\_section;

StackList<int> SL;

void fullStack(void \*);

void print(void \*);

void empty(void \*);

int main() {

InitializeCriticalSection(&critical\_section); {

HANDLE FullStack = (HANDLE)\_beginthread(fullStack, 1024, NULL);

HANDLE Print = (HANDLE)\_beginthread(print, 1024, NULL);

HANDLE Empty = (HANDLE)\_beginthread(empty, 1024, NULL);

Sleep(5000);

}

DeleteCriticalSection(&critical\_section);

}

void fullStack(void \*) {

EnterCriticalSection(&critical\_section);

std::cout << "Full Stack!" << std::endl;

std::random\_device generator;

std::uniform\_int\_distribution<int> distribution(0, 100);

for (int i = 0; i < 10; i++) {

int number = distribution(generator);

SL.Push(number);

}

std::cout << std::endl;

LeaveCriticalSection(&critical\_section);

\_endthread();

}

void print(void \*) {

EnterCriticalSection(&critical\_section);

std::cout << "Print Stack!" << std::endl;

SL.Print();

std::cout << std::endl;

LeaveCriticalSection(&critical\_section);

\_endthread();

}

void empty(void \*) {

EnterCriticalSection(&critical\_section);

std::cout << "Empty Stack!" << std::endl;

SL.Empty();

LeaveCriticalSection(&critical\_section);

\_endthread();

}

nodeStack.h

#ifndef \_\_NodeStack\_\_

#define \_\_NodeStack\_H\_\_

#include <windows.h>

#include <process.h>

#include <iostream>

#include <conio.h>

#include <random>

template <typename T>

struct NodeStack {

T item;

NodeStack<T>\* next;

};

template <typename T>

class StackList {

private:

NodeStack<T>\* pTop; // pointer to the top of the stack

public:

StackList() {

pTop = nullptr;

}

void Push(T item) {

NodeStack<T>\* p;

// 1. Form element

try {

p = new NodeStack<T>; // trying to allocate memory

}

catch (const std::bad\_alloc& e) {

// if memory is not allocated, then exit

std::cout << e.what() << std::endl;

return;

}

p->item = item;

p->next = pTop; // p points to 1st element

// 2. Redirect pTop to p

pTop = p;

}

void Empty() {

NodeStack<T>\* p;

NodeStack<T>\* p2;

p = pTop;

while (p != nullptr) {

p2 = p; // make a copy of p

p = p->next; // go to the next element of the stack

delete p2; // delete memory allocated for the previous item

}

pTop = nullptr; // fix the top of the stack

}

void Print() {

if (pTop == nullptr)

std::cout << "stack is empty." << std::endl;

else {

NodeStack<T>\* p;

p = pTop;

while (p != nullptr) {

std::cout << p->item << "\t";

p = p->next;

}

std::cout << std::endl;

}

}

~StackList() {

Empty();

}

};

#endif



Висновок: Багатопотоковість - це принцип побудови програми, при якому кілька блоків можуть виконуватися одночасно і не заважати один одному. Припустимо, нам потрібно, щоб якісь операції виконувались одночасно.

Наприклад, у нас є два об'єкти, які виконують конкретні операції незалежно один від одного. Але вони використовують той самий метод, який додає до статичної змінної число, що передається методу в параметрі. Однак якщо ці операції виконуватимуться одночасно, то незрозуміло буде, що запишеться у цій змінній, адже ми одночасно додаємо до вихідного значення змінної два різні числа.