Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные среды и

системное программирование

ОТЧЁТ

к лабораторной работе №4

на тему

Управление процессами и потоками (Windows)

Выполнила: студентка группы 153502

Шайкова Вероника Андреевна

Проверил: Гриценко Никита Юрьевич

Минск 2023

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc147885133)

[2 Теоретические сведения 4](#_Toc147885134)

[3 Результаты выполнения лабораторной работы 5](#_Toc147885135)

[Выводы](#_Toc147885136) 7

[Список использованных источников](#_Toc147885137) 8

[Приложение А (обязательное) Листинг кода](#_Toc147885138) 9

## 1 ПОСТАНОВКА ЗАДАЧИ

Целью выполнения лабораторной работы является создание оконного приложения на Win32 API, реализующее возможость наблюдения за одновременной работой нескольких потоков, а также возможность манипуляций над ними (создание, приостановка, возобновление работы, блокировка при помощи объектов Mutex или Semaphore, уничтожение).

## 2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Многозадачность осуществляется в Windows при помощи процессов и потоков. Любое приложение Windows после запуска реализуется как процесс (process), т.е. процесс можно представить как совокупность программного кода и выделенных для его исполнения системных ресурсов. При инициализации процесса система всегда создает первичный (основной) поток (thread), который исполняет код программы, манипулируя данными в адресном пространстве процесса. Из основного потока при необходимости могут быть запущены один или несколько вторичных потоков, которые выполняются одновременно с основным потоком. На самом деле истинный параллелизм возможен только при исполнении программы на многопроцессорной компьютерной системе, когда есть возможность распределить потоки между разными процессорами. В случае обычного однопроцессорного компьютера операционная система выделяет по очереди некоторый квант времени каждому потоку.

Для создания потока предназначена функция CreateThread, которая создает новый поток для процесса. Создающий поток должен указать начальный адрес кода, который должен выполнить новый поток. Как правило, начальным адресом является имя функции, определенной в программном коде, которая принимает один параметр и возвращает значение DWORD. Процесс может иметь несколько потоков, одновременно выполняющих одну и ту же функцию.

Поток может быть приостановлен при помощи функций Sleep(int ms) на некоторое количество миллисекунд или SuspentThread(HANDLE hThread) до поступления команды возобновления потока (ResumeThread(HANDLE hThread)). Поток заканчивает работу при вызове функций ExitThread(int id) или TerminateThread. Последнюю использовать рекоммендуется только в крайнем случае, поскольку ее вызов может нарушить процесс высвобождения занятых потоком данных.

Mutex можно использовать для защиты общего ресурса от одновременного доступа нескольких потоков или процессов. Каждый поток должен ожидать владения мьютексом, прежде чем он сможет выполнить код, который обращается к общему ресурсу. Для создания объекта мьютекса используется функция [CreateMutex](https://learn.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-createmutexa).

Когда поток должен использовать общие данные для нескольких потоков, он сначала запрашивает владение мьютексом с помощью функции [WaitForSingleObject](https://learn.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-waitforsingleobject). Если поток получает право собственности на мьютекс, производит необходимые операции над данными, а затем освобождает свое владение мьютексом с помощью функции [ReleaseMutex](https://learn.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-releasemutex).

## 3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

В ходе выполнения лабораторной работы было создано приложение, позволяющее наблюдать за работой двух потоков (рисунок 1). Первый поток итерирует переменную от 0 до 10000, а второй записывает ее значение в файл, предварительно достав предыдущее, которое и выводится на экран, после чего приостанавливается на некоторое время (функция Sleep(ms)).

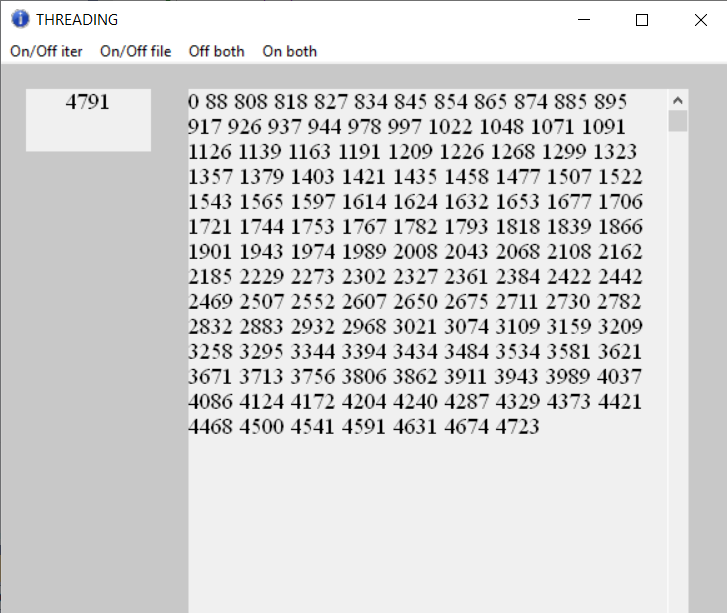


Рисунок 1 ─ Главное окно

Значение предварительно записанной итерируемой переменной находится в файле(рисунок 2), пока второй поток не обновит данные на экране и не считает следующее значение:

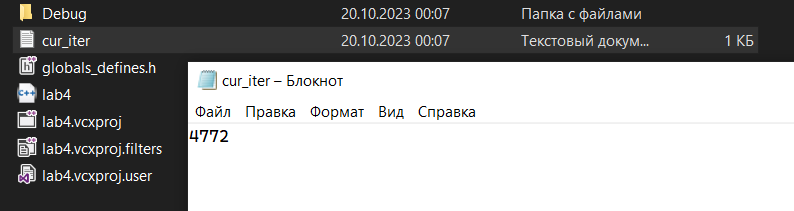
****

Рисунок 2 ─ файл сохранения

Управление потоками происходит при помощи оконного меню, которое позволяет приостановить либо возобновить работу одного или двух потоков, а также объекта Mutex.

## ВЫВОДЫ

В ходе выполнения лабораторной работы было создано приложение на Win32 API, основанное на многопоточном выполнении программы. Данно приложение позволяет оценить и сравнить скорость двух потоков, визуально проследить за их работой. Пользователю также доступна возможность поучаствовать в управлении потоками при помощи меню, которое позволяет приостанавливать и возобновлять работу потоков.

Кроме того, оба потока используют одну итерируемую переменную (первый для итерации, второй для записи в файл). Чтобы между потоками не возникло конфликта из-за использования общего ресурса, их работа дополнительно котролизуется при помощи объекта Mutex, что обеспечивает возможность только одному потоку использовать значение переменной в каждый момент времени.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Щупак Ю. Win32 API. Разработка приложений для Windows. ─ СПб: Питер, 2008. ─ 592 с.: ип.

[2] Создание классических приложений для Windows с использованием API Win32 [Электронный ресурс]. ─ Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/api ─ Дата доступа 20.09.2023

## ПРИЛОЖЕНИЕ А

## (обязательное)

## Листинг кода

**Lab4.cpp**

#include <windows.h>

#include <fileapi.h>

#include <string>

#include "globals\_defines.h"

void save\_to\_file()

{

WaitForSingleObject(iterMutex, INFINITE);

std::string s\_iter = std::to\_string(iter);

ReleaseMutex(iterMutex);

HANDLE wfile = CreateFileA(

"cur\_iter.txt", GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_COMPRESSED, NULL);

DWORD res;

WriteFile(wfile, s\_iter.c\_str(), s\_iter.size(), &res, NULL);

CloseHandle(wfile);

}

std::string load\_from\_file() {

HANDLE rfile = CreateFileA(

"cur\_iter.txt", GENERIC\_READ, 0, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_COMPRESSED, NULL);

DWORD deres;

char fromfile[6];

bool success = ReadFile(rfile, fromfile, sizeof(fromfile), &deres, NULL);

CloseHandle(rfile);

std::string res = std::to\_string(0);

if (success) {

res = std::string(fromfile).substr(0, deres);

}

return res;

}

WNDCLASS createWindowClass(HBRUSH background\_color, HCURSOR cursor\_type, HINSTANCE hInstance, HICON Icon, LPCWSTR window\_name, WNDPROC procedure) {

// создание пустого класса

WNDCLASS new\_window = { 0 };

// наполняем пустой класс параметрами, требуемыми для конкретного окна

new\_window.hbrBackground = background\_color;

new\_window.hCursor = cursor\_type;

new\_window.hInstance = hInstance;

new\_window.hIcon = Icon;

new\_window.lpszClassName = window\_name;

new\_window.lpfnWndProc = procedure;

return new\_window;

}

void CreateMenus(HWND wnd) {

HMENU root = CreateMenu();

AppendMenu(root, MF\_STRING, MENU\_ITER\_PAUSE\_RESUME, L"On/Off iter");

AppendMenu(root, MF\_STRING, MENU\_FILE\_PAUSE\_RESUME, L"On/Off file");

AppendMenu(root, MF\_STRING, MENU\_BOTH\_PAUSE, L"Off both");

AppendMenu(root, MF\_STRING, MENU\_BOTH\_RESUME, L"On both");

SetMenu(wnd, root);

}

void CreateWidgets(HWND wnd) {

itertext = CreateWindowA("static", "0", WS\_VISIBLE | WS\_CHILD | ES\_CENTER, 20, 20, 100, 50, wnd, NULL, NULL, NULL);

filetext = CreateWindowA("static", "0", WS\_VISIBLE | WS\_CHILD | BS\_MULTILINE | WS\_VSCROLL, 150, 20, 400, 800, wnd, NULL, NULL, NULL);

SendMessageA(itertext, WM\_SETFONT, WPARAM(chelovecheskiy\_shrift), TRUE);

SendMessageA(filetext, WM\_SETFONT, WPARAM(chelovecheskiy\_shrift), TRUE);

}

DWORD WINAPI ThreadIterate(LPVOID lpParameter) {

while (cont = true) {

while (iter <= 10000) {

SetWindowTextA(itertext, std::to\_string(iter).c\_str());

WaitForSingleObject(iterMutex, INFINITE);

++iter;

ReleaseMutex(iterMutex);

}

iter = 0;

}

ExitThread(0);

}

DWORD WINAPI ThreadFileWork(LPVOID lpParameter)

{

while (cont == true) {

std::string chariter = load\_from\_file();

int ss = chariter.size();

save\_to\_file();

std::string bigger\_text;

int cur\_size = GetWindowTextA(filetext, filedata, size);

if (ss + cur\_size > size - 2) {

filedata = new char[size];

bigger\_text = chariter;

}

else {

bigger\_text = std::string(filedata).substr(0, cur\_size) + " " + chariter;

}

SetWindowTextA(filetext, bigger\_text.c\_str());

Sleep(10);

}

ExitThread(0);

}

LRESULT CALLBACK createNewProcedure(HWND window, UINT message, WPARAM wp, LPARAM lp) {

switch (message) {

case WM\_CREATE:{

iterthread = CreateThread(NULL, 0, ThreadIterate, NULL, 0, NULL );

filethread = CreateThread(NULL, 0, ThreadFileWork, NULL, 1, NULL);

CreateWidgets(window);

SetTimer(window, 1, 1, NULL);

CreateMenus(window);

break;

}

case WM\_DESTROY:

{

cont = false;

//TerminateThread(iterthread, 0);

//TerminateThread(filethread, 0);

PostQuitMessage(0);

break;

}

case WM\_COMMAND:

switch (wp)

{

case MENU\_ITER\_PAUSE\_RESUME:{

if (iter\_suspended) {

ResumeThread(iterthread);

iter\_suspended = false;

}

else {

SuspendThread(iterthread);

iter\_suspended = true;

}

break;

}

case MENU\_FILE\_PAUSE\_RESUME:{

if (file\_suspended) {

ResumeThread(filethread);

file\_suspended = false;

}

else {

SuspendThread(filethread);

file\_suspended = true;

}

break;

}

case MENU\_BOTH\_PAUSE:{

SuspendThread(iterthread);

SuspendThread(filethread);

iter\_suspended = file\_suspended = true;

break;

}

case MENU\_BOTH\_RESUME:{

ResumeThread(iterthread);

ResumeThread(filethread);

iter\_suspended = file\_suspended = false;

break;

}}

break;

default: return DefWindowProc(window, message, wp, lp);

}

}

int WINAPI WinMain(HINSTANCE hInst, HINSTANCE hPrevInst, LPSTR args, int ncmdshow) {

chelovecheskiy\_shrift = CreateFontA(20, 7, 0, 0, FW\_EXTRALIGHT, FALSE, FALSE, FALSE, DEFAULT\_CHARSET, OUT\_OUTLINE\_PRECIS, CLIP\_DEFAULT\_PRECIS, ANTIALIASED\_QUALITY, FF\_ROMAN, "Font1");

WNDCLASS main\_window = createWindowClass(CreateSolidBrush(RGB(200, 200, 200)), LoadCursor(NULL, IDC\_CROSS), hInst, LoadIcon(NULL, IDI\_ASTERISK), L"MainWindow", createNewProcedure);

if (!RegisterClassW(&main\_window)) {

return -1;

}

MSG main\_window\_message = { 0 };

HWND mainwnd = CreateWindow(L"MainWindow", L"THREADING", WS\_OVERLAPPEDWINDOW | WS\_VISIBLE, 200, 200, 600, 500, NULL, NULL, NULL, NULL);

while (GetMessage(&main\_window\_message, NULL, NULL, NULL)) {

TranslateMessage(&main\_window\_message);

DispatchMessage(&main\_window\_message);

}

return 0;

}

**globals\_defines.h**

#pragma once

volatile int iter = 0;

int size = 1000;

char\* filedata = new char[size];

HWND itertext;

HWND filetext;

HANDLE iterthread;

HANDLE filethread;

HANDLE iterMutex;

HFONT chelovecheskiy\_shrift;

#define MENU\_ITER\_PAUSE\_RESUME 1

#define MENU\_FILE\_PAUSE\_RESUME 2

#define MENU\_BOTH\_PAUSE 3

#define MENU\_BOTH\_RESUME 4

bool iter\_suspended = false;

bool file\_suspended = false;

bool cont = true;