МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»
(БГТУ им. В.Г.Шухова)

Лабораторная работа № 3
дисциплина «Операционные системы»
по теме «Синхронизация потоков»

Выполнил: студент группы ВТ-31 Макаров Д.С.
Проверил: Михелев В.М.

Белгород 2019

Лабораторная работа № 3

«Синхронизация потоков»

Цель работы: получение практических навыков по использованию Win32 API для синхронизации процессов и потоков.

Вариант 9

Содержание отчета:

1. Наименование лабораторной работы, ее цель.
2. Краткое изложение теоретических основ о потоках в ОС Windows.
3. Составить программу, которая включает функцию, с помощью которой приложение

   приостановит свою работу до завершения другого приложения.
4. Исследовать использование мьютексов и событий на конкретном примере. Использовать приложения *LAB_OC_4_1* и *LAB_OC_4_2*, приведенные в папке *ЛАБОС4*. Разработать аналогичные приложения с использованием следующих методов синхронизации потоков. Вариант задания: **Ждущие таймеры**
5. Примеры разработанных приложений (результаты и тексты программ).

Ход работы

Краткие теоритеческие сведения

Процесс — это исполнение программы. Операционная система использует процессы для разделения исполняемых приложений.

Поток — это основная единица, которой операционная система выделяет время процессора. Каждый поток имеет приоритет планирования и набор структур, в которых система сохраняет контекст потока, когда выполнение потока приостановлено.

Синхронизация потоков. Синхронизация потоков необходима для того чтобы исключить состояния гонок и дедлоков. > Гонка потоков - (неопределенность параллелизма) ошибка в проектировании приложения, при которой работа системы или приложения зависит от того, в каком порядке выполняются части кода.

> Дедлок - ситуация , при которой несколько процессов находятся в состоянии ожидания ресурсов, занятых друг другом, и ни один из них не может продолжать свое выполнение.

Синхронизация в ОС Windows В операционной системе Windows существуют несколько объектов ядра для осуществления синхронизации потоков в системе.

1. Мьютекс
2. Семафор.
3. Ждущий таймер.
4. Критическая секция
5. Событие

Любой объект ядра может находится в двух состояних: * сигнальное (свободен) * не сигнально (занят)

Идея синхронизации заключается в том чтобы поток мог останавливать свою работу до освобождения необходимых для его выполнения ресурсов.

Используя API функцию **WaitForSingleObject**(*objectHandle*,*timeout*) мы можем приостановить работу процесса до освобождения ресурса *objectHandle*.

Ждущие таймеры - объект синхронизации переходящий в сигнальное состояние после определенной задержки (*dueTime*).

API функции для работы с ждущими таймерами.

- **CreateWaitableTimer**() - создание ждущего таймера, атрибуты функции принимают имя таймера, атрибуты защиты, и флаг ручного сброса.
- **OpenWaitableTimer**() - открытие дескриптора существующего ждущего таймера по имени или адресу.
- **SetWaitableTimer**() - установка параметров ждущего таймера можно указать задержку (*dueTime*) и период и флаг пробуждения при установке которого таймер будет выводить систему из состояния сна.

Ждущий таймер может работать в 3 режимах.

- *Режим ручного сброса* - после установки параметров таймера и последующего захвата функцией **WaitForSingleObject** таймер остается в сигнальном состоянии до следующей установки параметров.
- *Режим автоматического сброса* - после захвата фукцией **WaitForSingleObject** таймер автоматически переходит в несигнальное состояние.
- *Режим интервального таймера* - после установки параметров таймера, по прошествию задержки таймер переходит в сигнальное состояние, далее после захвата таймера другим потоком таймер сбрасывается после прошествия времени указанном в периода.

Примеры работы программы

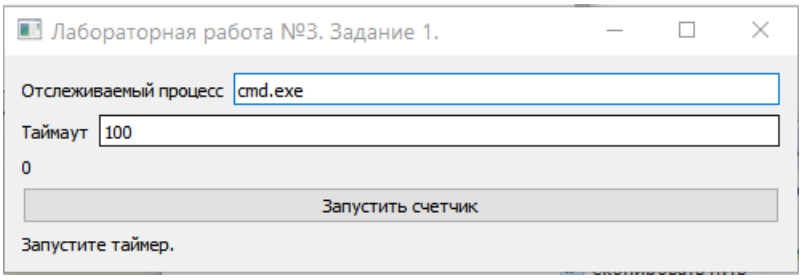

Рис. 1: Задание 1. Приложение останавливающее счетчик во время запущенного процесса

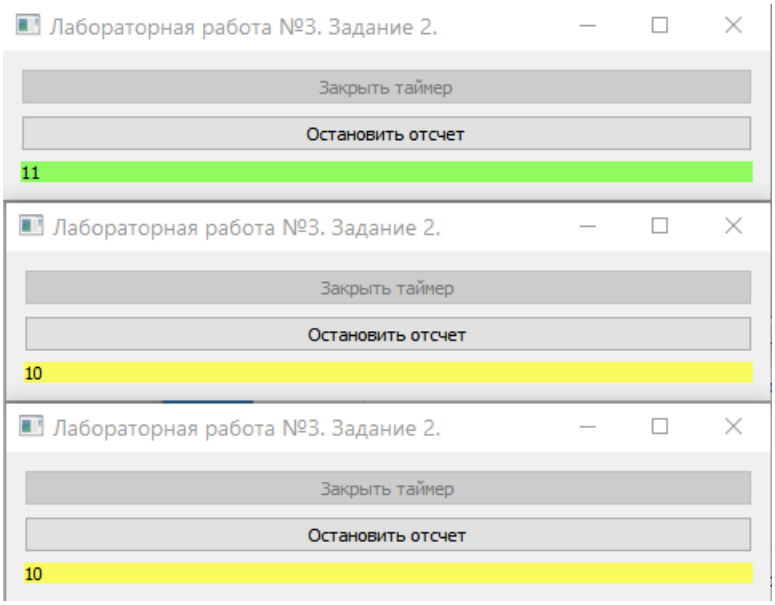

Рис. 2: Задание 2. Приложение использующее именованный ждущий таймер

Приложение

Содержимое файла mainwindow.cpp

```
#include "mainwindow.hpp"

QColor normalColor;
QColor countColor = QColor("#fbfb46");

MainWindow::MainWindow(){
    layout = new QVBoxLayout;

    status = new QLabel("0");
    normalColor = status->palette().color(status->backgroundRole());

    timerButton = new QPushButton("Создать таймер");
    processButton = new QPushButton("Запустить отсчет");

    layout->addWidget(timerButton);
    layout->addWidget(processButton);
    layout->addWidget(status);
    this->setLayout(layout);
    connect(this->processButton,SIGNAL(pressed()),
            this,SLOT(processBClick()));
    connect(this->timerButton,SIGNAL(pressed()),
            this,SLOT(timerBClick()));
    connect(&runTimer, SIGNAL(timeout()), this, SLOT(runTimerTick()));
    processButton->setEnabled(false);

};

void MainWindow::timerBClick(){
    timerState = !timerState;
    if(timerState){
        h = CreateWaitableTimerA(NULL, FALSE, "named_timer");
        if(h != NULL){
            if (GetLastError() == ERROR_ALREADY_EXISTS){
                status->setText("Таймер \"named_timer\" уже создан другим процессом.
                ↪  Дескриптор таймера получен 0x"+ QString::number((long long)h, 16));
            }else{
                status->setText("Таймер \"named_timer\" создан текущим процессом. Дескриптор
                ↪  таймера 0x"+ QString::number((long long)h, 16));
                LARGE_INTEGER l_i;
                //l_i.QuadPart = -1000LL;
                //SetWaitableTimer(h, &l_i, 10000, NULL, NULL, FALSE);
                l_i.QuadPart = 0;
                SetWaitableTimer(h, &l_i, 0, NULL, NULL, FALSE);
            };
            timerButton->setText("Закрыть таймер");
            processButton->setEnabled(true);
        }else{
            status->setText("Ошибка создания таймера");
            timerState = false;
        };
    }else{
        timerButton->setText("Создать таймер");
        processButton->setEnabled(false);
        runTimer.stop();
        CloseHandle(h);
    };
```

```
};

void MainWindow::processBClick(){
    runState = !runState;
    if(runState){
        LARGE_INTEGER l_i;
        l_i.QuadPart = -1000LL;
        SetWaitableTimer(h, &l_i, 10000, NULL, NULL, FALSE);
        WaitForSingleObject(h, INFINITE);
        runTimer.start(1000);
        timerButton->setEnabled(false);
        processButton->setText("Остановить отсчет");

    }else{
        runTimer.stop();
        timerButton->setEnabled(true);
        processButton->setText("Запустить отсчет");
    }
};

void MainWindow::runTimerTick(){
    if(!(count % 10)){
        runTimer.stop();
        status->setStyleSheet("background-color: rgb(250,250,95)");
        repaint();
        WaitForSingleObject(h, INFINITE);
        count++;
        status->setStyleSheet("background-color: rgb(144,250,95)");
        status->setText(QString::number(count));
        runTimer.start(1000);
        repaint();
    }else{
        count++;
        status->setText(QString::number(count));
        repaint();
    }
};
```

Содержимое файла task1.cpp

```
#include "task1.hpp"
#include <QDebug>

Task1::Task1(){
    l1 = new QHBoxLayout;
    l2 = new QHBoxLayout;
    timeout = new QLineEdit(QString::number(timeoutValue));
    process = new QLineEdit(processName);
    counter = new QLabel("0");
    processLabel = new QLabel("Отслеживаемый процесс");
    timeoutLabel = new QLabel("Таймаут");
    button = new QPushButton("Запустить счетчик");
    QVBoxLayout *layout = new QVBoxLayout;
    statusLabel = new QLabel("Запустите таймер.");

    l1->addWidget(timeoutLabel);
    l1->addWidget(timeout);
    l2->addWidget(processLabel);
    l2->addWidget(process);
```

```
    layout->addLayout(l2);
    layout->addLayout(l1);
    layout->addWidget(counter);
    layout->addWidget(button);
    layout->addWidget(statusLabel);
    this->setLayout(layout);


    getProcIDByName(processName,&processId);
    qDebug() << processId;
    h = OpenProcess(PROCESS_QUERY_INFORMATION | PROCESS_VM_READ |
    ↪  SYNCHRONIZE,false,processId);
    connect(&timer,SIGNAL(timeout()),
            this,SLOT(timerTick()));
    connect(button,SIGNAL(pressed()),
            this,SLOT(buttonClick()));
    connect(timeout,SIGNAL(textChanged(const QString)),
            this,SLOT(timeoutChange(const QString)));
    connect(process,SIGNAL(textChanged(const QString)),
            this,SLOT(processChange(const QString)));
};

void Task1::buttonClick(){
    timerFlag = !timerFlag;
    if(timerFlag){
        button->setText(QString("Остановить счетчик"));
        timer.start(2000);
        statusLabel->setText(QString("Запустите приложение ")+processName);
    }else{
        button->setText(QString("Запустить счетчик"));
        timer.stop();
        statusLabel->setText("Запустите таймер.");
    };
};

void Task1::timerTick(){
    count++;
    counter->setStyleSheet("background-color: rgb(250,250,95)");
    repaint();
    DWORD r_value = WaitForSingleObject(h,timeoutValue);
    counter->setStyleSheet("background-color: rgb(144,250,95)");
    counter->setText(QString::number(count));
    switch (r_value) {
        case WAIT_OBJECT_0:
            statusLabel->setStyleSheet("color: rgb(0, 0, 0)");
            statusLabel->setText(QString("Приложение ")+processName+QString(" закрыто. Счетчик
            ↪  работает дальше."));
            break;
        case WAIT_TIMEOUT:
            statusLabel->setStyleSheet("color: rgb(0, 0, 0)");
            statusLabel->setText(QString("Время ожидание вышло. Счетчик работает дальше."));
            break;
        case WAIT_FAILED:
            counter->setStyleSheet("background-color: rgb(120,0,0)");
            statusLabel->setStyleSheet("color: rgb(255, 100, 100)");
            statusLabel->setText("Приложение не найдено или произошла ошибка.");
    }
    CloseHandle(h);
    getProcIDByName(processName,&processId);
    h = OpenProcess(PROCESS_QUERY_INFORMATION | PROCESS_VM_READ |
    ↪  SYNCHRONIZE,false,processId);
```

```
};

void Task1::timeoutChange(const QString & text){
    timeoutValue = text.toInt();
};

void Task1::processChange(const QString & text){
    timer.stop();
    button->setText(QString("Запустить счетчик"));
    timerFlag = false;
    processName = text;
    CloseHandle(h);
    getProcIDByName(processName,&processId);
    qDebug() << processId;
    h = OpenProcess(PROCESS_QUERY_INFORMATION | PROCESS_VM_READ |
    ↪  SYNCHRONIZE,false,processId);
};

bool getProcIDByName(QString pr_name, DWORD *ID){
    DWORD len = 1000, size_needed, ID_arr[len];
    HANDLE h;
    WCHAR w_name[255];
    QString name;
    EnumProcesses(ID_arr, len*sizeof(DWORD), &size_needed);
    for (uint i = 0; i < size_needed/sizeof(DWORD); i++){
        h = OpenProcess(PROCESS_QUERY_INFORMATION | PROCESS_VM_READ, false, ID_arr[i]);
        if (!GetModuleBaseNameW(h, NULL, w_name, 255)) name = "";
        else name = QString::fromWCharArray(w_name);
        if (name.toLower() == pr_name.toLower()){
            CloseHandle(h);
            if (ID != nullptr) *ID = ID_arr[i];
            return true;
        }
        CloseHandle(h);
    }
    return false;
}
```

Содержимое файла t1main.cpp

```
#include <QtWidgets>
#include <windows.h>
#include <wchar.h>
#include "task1.hpp"

int main (int argc,char* argv[]){
    QApplication* app = new QApplication (argc,argv);
    Task1 *t1 =new Task1;
    t1->setWindowTitle("Лабораторная работа №3. Задание 1.");
    t1->show();
    return app->exec();
}
```

Содержимое файла mainwindow.hpp

```
#include <QtWidgets>
#include <windows.h>
#include <tchar.h>
#include <psapi.h>
#pragma once
```

```
class MainWindow : public QWidget{
    Q_OBJECT
    private:
        HANDLE h;
        int count = 1;
        bool timerState = false;
        bool runState = false;
        QTimer runTimer;
        QVBoxLayout *layout;
        QLabel *status;
        QPushButton *timerButton,*processButton;
    public:
        MainWindow();
    public slots:
        void timerBClick();
        void processBClick();
        void runTimerTick();
};
```

Содержимое файла task1.hpp

```
#include <QtWidgets>
#include <windows.h>
#include <tchar.h>
#include <psapi.h>
#pragma once

bool getProcIDByName(QString pr_name, DWORD *ID);

class Task1 : public QWidget{
    Q_OBJECT
    private:
        int count = 0;
        int timeoutValue = 100;
        QString processName = QString("cmd.exe");
        HANDLE h;
        DWORD processId;
        bool timerFlag = false;
        QTimer timer;
        QHBoxLayout *l1;
        QHBoxLayout *l2;
        QLineEdit *timeout;
        QLineEdit *process;
        QLabel *counter;
        QLabel *processLabel;
        QLabel *timeoutLabel;
        QLabel *statusLabel;
        QPushButton *button;
        QVBoxLayout *layout;
    public:
        Task1();
    public slots:
        void buttonClick();
        void timerTick();
        void timeoutChange(const QString & text);
        void processChange(const QString & text);
};
```

Содержимое файла t2main.cpp

```
#include <QtWidgets>
#include <windows.h>
#include <wchar.h>
#include "mainwindow.hpp"

int main (int argc,char* argv[]){
    QApplication* app = new QApplication (argc,argv);
    MainWindow *t1 = new MainWindow;
    t1->setWindowTitle("Лабораторная работа №3. Задание 2.");
    t1->show();
    return app->exec();
}
```