Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра информационных технологий автоматизированных систем

Факультет ИТиУ
Специальность АСОИ

Индивидуальная практическая работа по модулю 4 по дисциплине «Системное программное обеспечение», часть 1 «Управление процессами и потоками» Вариант №1

Выполнил:
Ст. Гр. 820601
Шведов А.Р
Зачетная книжка No 82060145

Минск 2020

1. Задание

Во всех вариантах заданий требуется синхронизировать потоки с помощью одного из следующих методов синхронизации:

- критическая секция;
- Mutex;
- событие;
- семафоры.

Имеется массив элементов типа Date в виде структуры

```
struct Dates {
int count = 0;//количество имеющихся элементов в массиве Date
dates[100];
} dts = {0};
```

Главный поток программы (функция main()) создает вторичный поток, передав в него указатель на структуру dts. Вторичный поток запоминает значение из поля count, открывает файл и затем в цикле, если значение count изменилось, то записывает последний элемент массива dates в файл. Так продолжается до тех пор, пока count не достигнет некоторого максимального значения, после этого поток закрывает файл и завершается; Далее главный поток организует цикл ввода дат следующим образом:

- инициализируется временная переменная tmp типа Date (ввод с клавиатуры);
- с помощью функции SuspendThread() приостанавливается поток;
- значение временной переменной заносится в массив dts.dates[dts.count] = tmp; dts.count++;
- с помощью функции ResumeTread() поток запускается на выполнение;

Так продолжается до тех пор, пока count не достигнет некоторого максимального значения.

2. Ход работы

2.1.Теоретические сведения

Для программ, использующих несколько потоков или процессов, необходимо, чтобы все они выполняли возложенные на них функции в нужной последовательности. В среде *Windows 9x* для этой цели предлагается использовать несколько механизмов, обеспечивающих слаженную работу потоков. Эти механизмы называют механизмами синхронизации.

Средства синхронизации в *ОС Windows*:

- Критическая секция (Critical Section) – это объект, который принадлежи

процессу, а не ядру. А значит, не может синхронизировать потоки из разных процессов. Критический раздел анализирует значение специальной переменной процесса, которая используется как флаг, предотвращающий исполнение некоторого участка кода несколькими потоками одновременно. Среди синхронизирующих объектов критические разделы наиболее просты;

- *Mutex*. Это объект ядра, у него есть имя, а значит с их помощью можно синхронизировать доступ к общим данным со стороны нескольких процессов, точнее, со стороны потоков разных процессов. Ни один другой поток не может завладеть мьютексом, который уже принадлежит одному из по-токов. Если мьютекс защищает какие-то совместно используемые данные, он сможет выполнить свою функцию только в случае, если перед обращением к этим данным каждый из потоков будет проверять состояние этого мьютекса. Windows расценивает мьютекс как объект общего доступа, который можно пере-вести в сигнальное состояние или сбросить;
- Семафор – *semaphore*. Объект ядра “семафор” используются для учёта ресурсов и служат для ограничения одновременного доступа к ресурсу нескольких потоков. Используя семафор, можно организовать работу программы таким образом, что к ресурсу одновременно смо-гут получить доступ несколько потоков, однако количество этих потоков будет ограничено;
- Событие – *event*. События обычно просто оповещают об окончании какой-либо операции, они также являются объектами ядра. Можно не просто явным образом освободить, но также есть операция установки события. События могут быть мануальными (*manual*) и единичными (*single*). Единичное событие (*single event*) – это скорее общий флаг. Событие находится в сигнальном состоянии, если его установил какой-нибудь поток. Если для работы программы требуется, чтобы в случае возникновения события на него реа-гировал только один из потоков, в то время как все остальные потоки продолжали ждать, то используют единичное событие. Мануальное событие (*manual event*) — это не про-сто общий флаг для нескольких потоков. Оно выполняет несколько более сложные функции. Любой поток может установить это событие или сбросить (очистить) его. Если событие установлено, оно останется в этом состоянии сколь угодно долгое время, вне зависимости от того, сколько потоков ожидают установки этого события.

Для синхронизации потоков был использован *Mutex*.

2.2. Листинг Программы

Файл “main.cpp”

```
#include <thread>
#include <iostream>
#include <fstream>
#include <mutex>
#include <condition_variable>

using namespace std;

static const int MAX_COUNT = 5;

struct Date {
    int year;
    int month;
    int day;
};

struct Dates {
    int count = 0;
    Date dates[100]{};
} dts = {0};

mutex kLock;
condition_variable kCv;
bool kReady = false;
bool kProcessed = false;

void secondThread(Dates* dates){
    ofstream datesFile("dates.txt");

    int currentCount = dates->count;

    while (currentCount < MAX_COUNT) {
        // дождаться передачи управления от главного потока
        {
            unique_lock<std::mutex> lk(kLock);
            kCv.wait(lk, [] { return kReady; } );
        }

        if (dates->count != currentCount) {
            currentCount = dates->count;
```

```
            Date date = dates->dates[currentCount-1];
            datesFile << "date[" << currentCount << "]: { day: " << date.day << ", month:
" << date.month << ", year: " << date.year << " } \n" << endl;
        }

        {
            std::lock_guard<std::mutex> lk(kLock);
            kProcessed = true;
            kCv.notify_one();
        }
    }

    datesFile.close();
}

int enter_int(){
    int n;

    while(true){
        cin >> n;
        if (!cin){  // == cin.fail()
            cout << "error. Try again\n";
            cin.clear();
            while(cin.get()!='\n');
        }else break;
    }
    return n;
}

int main(int argc, const char * argv[]) {
    thread secondThr(secondThread, &dts);
    secondThr.detach();

    while (dts.count < MAX_COUNT) {
        Date tmp{};

        cout << "Enter year: ";
        tmp.year = enter_int();
        cout << endl << "Enter month: ";
        tmp.month = enter_int();
        cout << endl << "Enter day: ";
        tmp.day = enter_int();
        cout << endl;
```

```
        dts.dates[dts.count] = tmp;
        dts.count++;

        {
            // передать управление второму потоку
            std::lock_guard<std::mutex> lk(kLock);
            kReady = true;
            kCv.notify_one();
        }

        // дождаться выполнения второго потока
        std::unique_lock<std::mutex> lk(kLock);
        kCv.wait(lk, []{return kProcessed;});
    }

    return 0;
}
```

3. Выводы

В ходе выполнения данной работы были изучены средства для синхронизации потоков и обработки ошибок в C++.