Министерство образования и науки РФ

Севастопольский государственный университет

Отчёт

по лабораторной №3

**«Использование программного интерфейса Win API.**

**Синхронизация процессов и потоков в ОС Windows»**

Выполнил:

ст.гр.ИСб-22д

Воронин И.Ю.

Проверил:

Забаштанский А.К.

Севастополь

2015

1.Цель работы

Изучение программного интерфейса приложений(API) операционных систем Windows 9x, NT, ME, XP. Приобретение практических навыков синхронизации процессов и потоков, используя Win API в средах программирования Borland Delphi, C++ Builder или Visual Studio.

2.Вариант задания

3.1. Написать программу **Sort3**, реализующую следующий алгоритм:

* + 1. Получить в качестве параметра командной строки номера процесса N\_PRC;
    2. Заполнить массив N целых чисел случайными значениями из диапазона 0-100;
    3. Отсортировать массив;
    4. Вывести на экран отсортированный массив в формате N\_PRC:A[i](то есть перед выводом каждого элемента массива необходимо выводить значение параметра N\_PRC);

Язык программирования, метод и направление сортировки, а также количество элементов массива N выбирается в соответствии с вариантом задания, приведенным в таблице 3.1.

3.2. Написать программу **Master3**, создающую процессы **Sort3**, используя системные вызовы **CreateProcess,** передавая в качестве параметра командной строки номер процесса – **N\_PRC**. Необходимо, чтобы все процессы использовали одну консоль и имели класс приоритета NORMAL\_PRIORITY\_CLASS. Количество процессов выбирается исходя из варианта задания.

Так как процессы имеют одинаковый приоритет, то при выводе элементов массива на экран, используя одну консоль, процессы будут прерывать друг друга. Таким образом, выводимые массивы различных процессов будут «перемешаны». Убедитесь в этом запуская программу **Master3**.

В данном случае консоль выступает в качестве разделяемого ресурса и очевидна необходимость использования механизмов взаимоисключения, для предотвращения “смешивания” массивов при выводе.

3.3. Переписать программы **Sort3** и **Master3**, с использованием механизма взаимоисключений (необходимо использовать объект Mutex). Полученная программа **Master3\_mtx** должна создавать мьютекс (с помощью вызова **CreateMutex)** перед созданием процессов, а программа **Sort3\_mtx** должна получать идентификатор (handle) мьютекса (с помощью вызова **OpenMutex)**, занимать мьютекс перед выводом отсортированного массива на экран (с помощью вызова **WaitForSingleObject)**, а после вывода массива на экран освобождать мьютекс (используя вызов **ReleaseMutex)**.

3.4 Разработать программу **Master3\_Crit** для исследования механизма критических секций.

Программа должна содержать функцию **Sort**, запускаемую несколько раз в виде отдельных потоков, с помощью вызовов CreateThread. Функция **Sort** должна: генерировать случайный массив из N элементов, сортировать его и выводить его на экран.

Убедиться в том, что при выполнении программы без использования механизма критических секций при выводе на экран из различных потоков отсортированные массивы будут «перемешиваться».

Добавить вызов InitializeCriticalSection перед созданием потоков и EnterCriticalSection, LeaveCriticalSection, соответственно, перед и после вывода массива на экран в функции **Sort**. Убедиться в том, что отсортированные массивы выводятся на экран не “перемешиваясь”.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 4 | Visual Studio | Быстрая | Убыв. | 6 | 50 |

3.Текст программы

**Использование мьютексов:**

**MASTER**

#include <windows.h>

#include <iostream>

#include <string.h>

#define N 6

using namespace std;

int main(){

char sort\_file[14] = "Sort3.exe";

char cmd[80] = "Sort3.exe ";

STARTUPINFO info1;

memset(&info1, 0, sizeof(info1));

info1.cb = sizeof(info1);

PROCESS\_INFORMATION pinfo1;

//создание мьютекса

HANDLE mtx = CreateMutex(NULL,FALSE,"MyMutex");

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

cmd[strlen(sort\_file)+1] = ' ';

cmd[strlen(sort\_file)+2] = (char)((int)'0'+i);

cmd[strlen(sort\_file)+3] = '\0';

CreateProcess(NULL,cmd,NULL,NULL,TRUE,

NORMAL\_PRIORITY\_CLASS,NULL,NULL,&info1,&pinfo1);

}

return 0;

}

**Sort**

#include <iostream>

#include <windows.h>

#include <algorithm>

#define N 50

using namespace std;

//сортировка

void qs(int\* s\_arr, int first, int last);

//печать массива

void mass\_print(int \*arr,int n,char\* N\_PRC);

int main(int argc, char \*argv[]){

int arr[N];

char\* N\_PRC="";

if(argc>1){

N\_PRC= argv[1];

}

cout << (argv[1]) << endl;

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

arr[i] = rand()%100;

}

qs(arr,0,N);

mass\_print(arr,N,N\_PRC);

return 0;

}

void mass\_print(int \*arr,int n,char\* N\_PRC){

cout << endl;

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

cout << N\_PRC << ": "<<arr[i] << endl;

}

cout << endl;

}

void qs(int\* s\_arr, int first, int last){

int i = first, j = last, x = s\_arr[(first + last) / 2];

do {

while (s\_arr[i] > x) i++;

while (s\_arr[j] < x) j--;

if (i <= j) {

if (s\_arr[i] < s\_arr[j]) swap(s\_arr[i], s\_arr[j]);

i++;

j--;

}

} while (i <= j);

if (i < last)

qs(s\_arr, i, last);

if (first < j)

qs(s\_arr, first, j);

}

**Использование критической секции:**

#include <iostream>

#include <windows.h>

#include <algorithm>

CRITICAL\_SECTION critsec;

const int N=50;

int amount=0;

//глобальная переменная

int arr[N];

using namespace std;

DWORD WINAPI TH(const LPVOID param);

//сортировка

void qs(int\* s\_arr, int first, int last);

//печать массива

void mass\_print(int \*arr,int n,char\* N\_PRC);

DWORD WINAPI TH(const LPVOID param){

EnterCriticalSection(&critsec);

int counter = (int)param;

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

arr[i] = rand()%100;

}

qs(arr,0,N);

mass\_print(arr,N,(char\*)('0'+counter));

LeaveCriticalSection(&critsec);

}

int main(int argc, char \*argv[]){

InitializeCriticalSection(&critsec);

HANDLE T1 = CreateThread(NULL,0,&TH,0,CREATE\_SUSPENDED,0);

HANDLE T2 = CreateThread(NULL,0,&TH,0,CREATE\_SUSPENDED,0);

HANDLE T3 = CreateThread(NULL,0,&TH,0,CREATE\_SUSPENDED,0);

HANDLE T4 = CreateThread(NULL,0,&TH,0,CREATE\_SUSPENDED,0);

HANDLE T5 = CreateThread(NULL,0,&TH,0,CREATE\_SUSPENDED,0);

HANDLE T6 = CreateThread(NULL,0,&TH,0,CREATE\_SUSPENDED,0);

SetThreadPriority(T1,THREAD\_PRIORITY\_NORMAL);

SetThreadPriority(T2,THREAD\_PRIORITY\_NORMAL);

SetThreadPriority(T3,THREAD\_PRIORITY\_NORMAL);

SetThreadPriority(T4,THREAD\_PRIORITY\_NORMAL);

SetThreadPriority(T5,THREAD\_PRIORITY\_NORMAL);

SetThreadPriority(T6,THREAD\_PRIORITY\_NORMAL);

ResumeThread(T1);

ResumeThread(T2);

ResumeThread(T3);

ResumeThread(T4);

ResumeThread(T5);

ResumeThread(T6);

return 0;

}

void mass\_print(int \*arr,int n,char\* N\_PRC){

cout << endl;

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

cout << amount << ": "<<arr[i] << endl;

}

cout << endl;

amount++;

}

void qs(int\* s\_arr, int first, int last){

int i = first, j = last, x = s\_arr[(first + last) / 2];

do {

while (s\_arr[i] > x) i++;

while (s\_arr[j] < x) j--;

if (i <= j) {

if (s\_arr[i] < s\_arr[j]) swap(s\_arr[i], s\_arr[j]);

i++;

j--;

}

} while (i <= j);

if (i < last)

qs(s\_arr, i, last);

if (first < j)

qs(s\_arr, first, j);

}

4.Тест программы

Тест программы до использования мьютексов(рис.4.1)

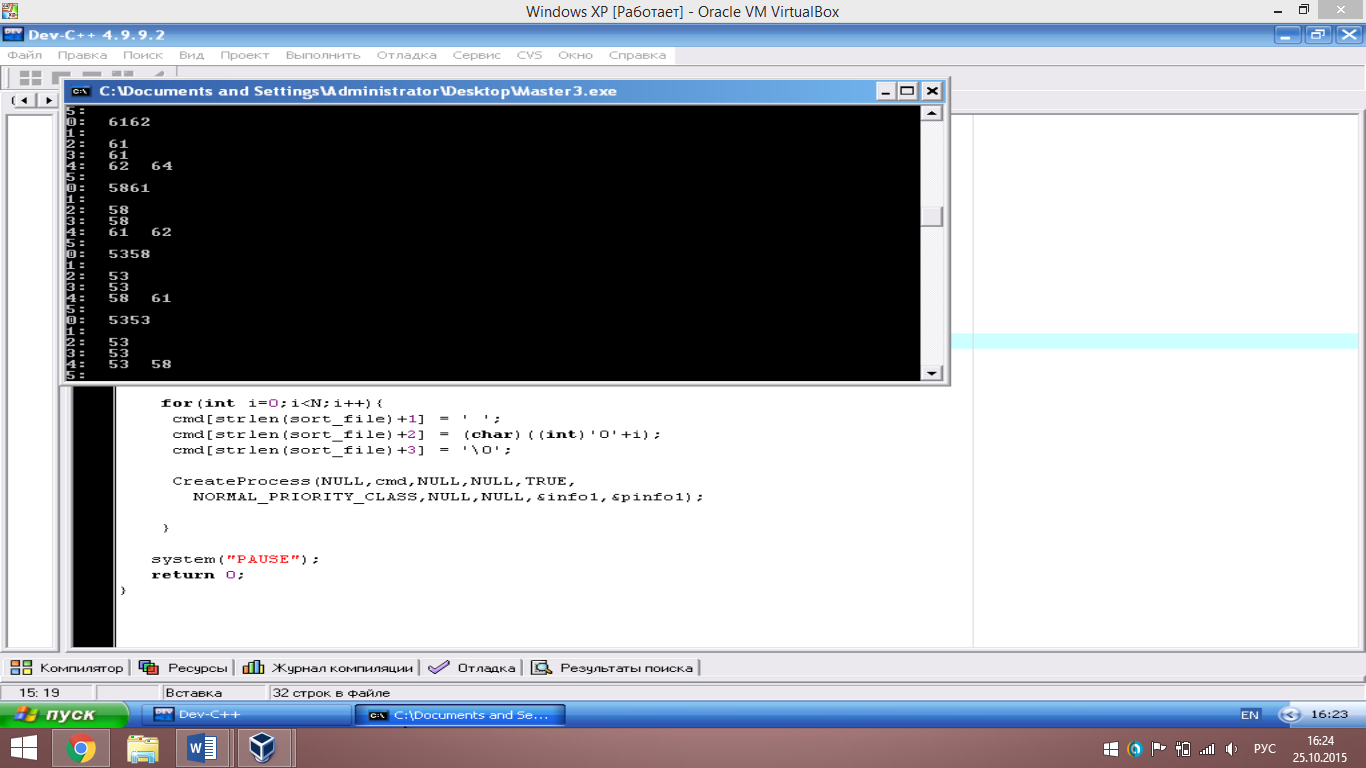


Рис.4.1-Тест №1.

Тест программы с использованием мьютексов(рис.4.2.)

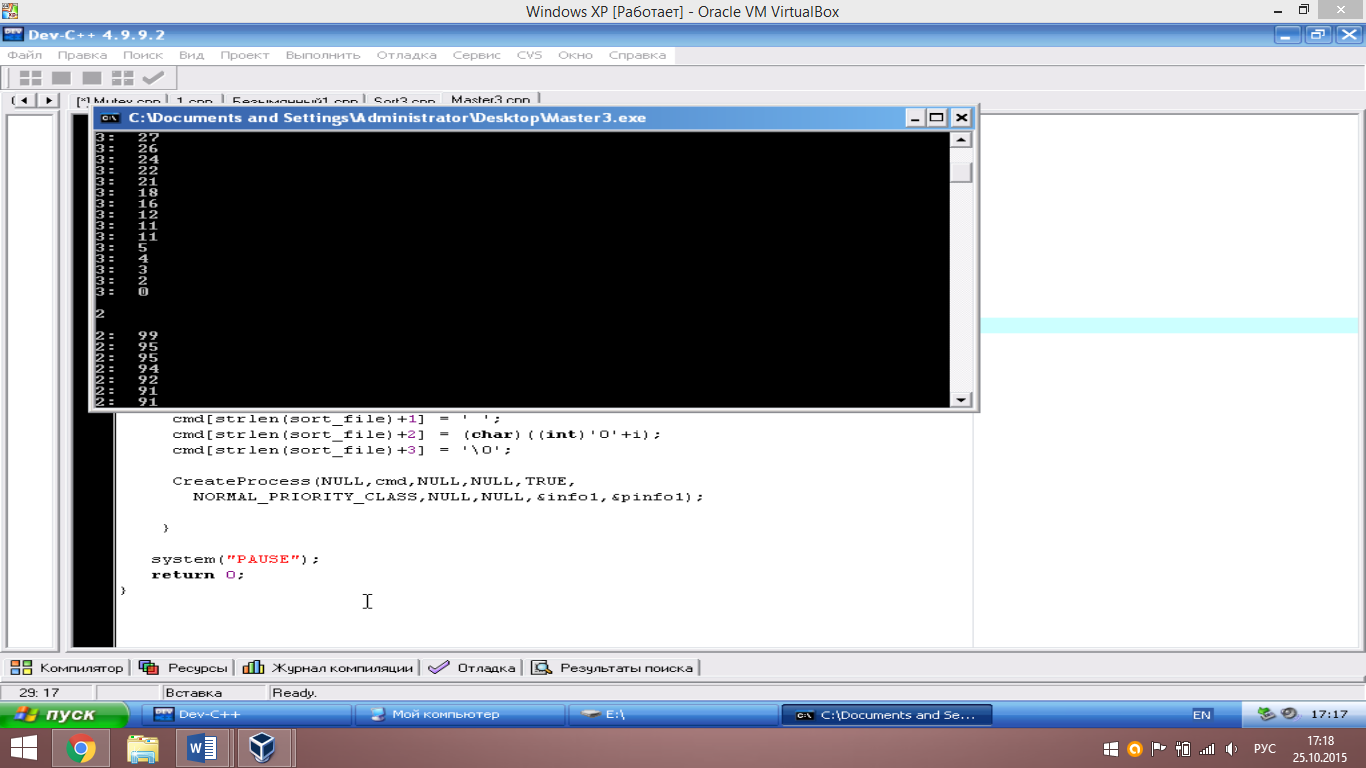
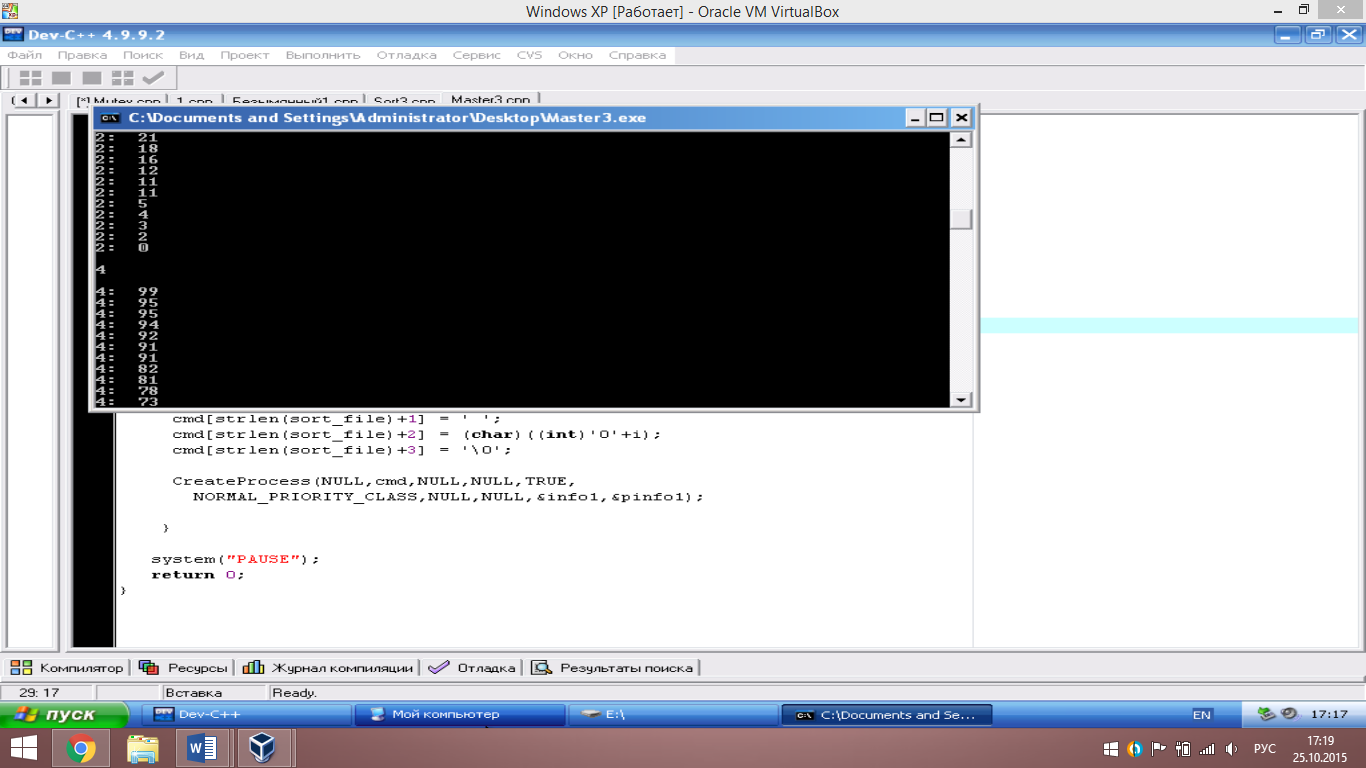
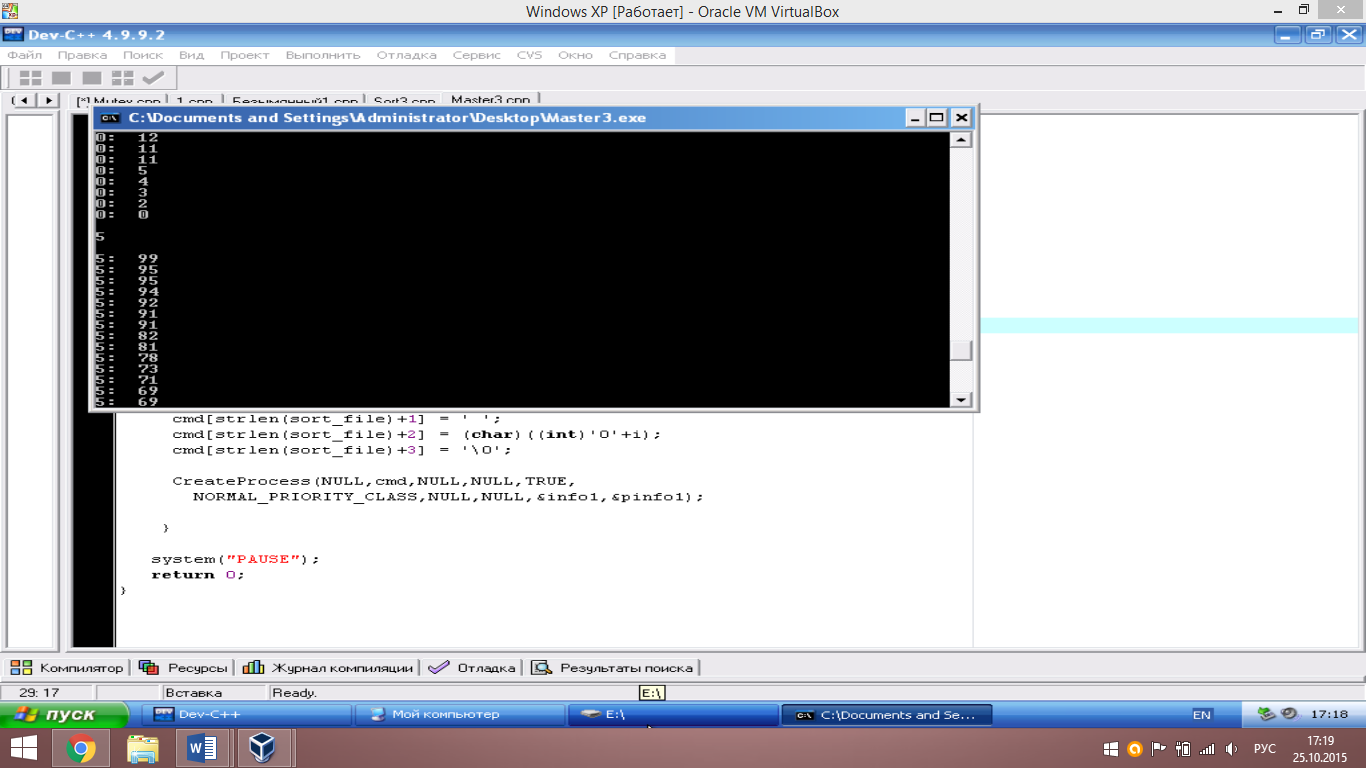


Рис.4.1-Тест №1.

Аналогичные результаты можно увидеть и при использовании критической секции.

ВЫВОДЫ

В данной лабораторной работе были изучены мьютексы и критические секции. Данные технологии используются, когда потокам или процессам необходимо поделить между собой один общий ресурс. В данной случае этим ресурсом было окно командной строки. Были проведены тесты, которые показывали правильность написанной программы.