Министерство образования и науки Российской Федерации

Севастопольский государственный университет

Институт информационных технологий и управления в технических системах

ОТЧЕТ

по лабораторной работе №3

Использование программного интерфейса Win API.

Синхронизация процессов и потоков в ОС Windows.

Выполнил: ст. гр. ИС/б-21

Куркчи А.Е.

Проверил:

Шишкевич В.Е.

Севастополь

2015

# Цель

# 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.

1. Постановка задачи

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

* + 1. Полученить в качестве параметра командной строки номера процесса N\_PRC;
    2. Заполненить 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 «перемешившись».

1. Текст программ

Программа Master\_mtx:

#include "stdafx.h"

**using** **namespace** std**;**

const int T **=** 6**;**

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

int i**;**

char cmd**[**128**],**

window**[**64**];**

DWORD flags**;**

int pid **=** GetCurrentProcessId**();**

cout **<<** "Master: Starting PID: " **<<** pid **<<** endl**;**

cout**.**flush**();**

STARTUPINFO info**;**

memset**(&**info**,** 0**,** **sizeof(**info**));**

info**.**cb **=** **sizeof(**info**);**

HANDLE mtx **=** CreateMutex**(NULL,**0**,**"lab.03.mtx"**);**

**for(**i**=**1**;**i**<=**T**;**i**++)** **{**

sprintf**(**cmd**,**"%s %d"**,**"Lab.03.Sort\_mtx.exe"**,**i**);**

sprintf**(**window**,**"Process: %d"**,**i**);**

flags **=** NORMAL\_PRIORITY\_CLASS**;**

info**.**lpTitle **=** window**;**

PROCESS\_INFORMATION pinfo**;**

CreateProcess**(NULL,**cmd**,NULL,NULL,**TRUE**,**flags**,NULL,NULL,&**info**,&**pinfo**);**

**}**

CloseHandle**(**mtx**);**

Sleep**(**1000**);**

system**(**"pause"**);**

cout **<<** "Master: Exiting\n"**;**

**return** 0**;**

**}**

Программа Sort\_mtx:

#include "stdafx.h"

**using** **namespace** std**;**

const int N **=** 45**;**

void fill**(**int**\*** a**,**int n**)** **{**

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

a**[**i**]** **=** rand**()%**10000**;**

**}**

**}**

void sort**(**int**\*** a**,** int n**)** **{**

int i**,**j**,**k**;**

int t**;**

**for(**k **=** n**/**2**;** k **>** 0**;** k **/=**2**)**

**for(**i **=** k**;** i **<** n**;** i**++)** **{**

t **=** a**[**i**];**

**for(**j **=** i**;** j**>=**k**;** j**-=**k**)** **{**

**if(**t **<** a**[**j**-**k**])**

a**[**j**]** **=** a**[**j**-**k**];**

**else**

**break;**

**}**

a**[**j**]** **=** t**;**

**}**

**}**

void print**(**int**\*** a**,**int n**,**int p**)** **{**

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

cout **<<** p **<<** ":" **<<** a**[**i**]** **<<** " "**;**

**}**

cout **<<** endl**;**

**}**

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

**if(**argc **!=** 2**)** **{**

**return** 1**;**

**}**

HANDLE mtx **=** CreateMutex**(NULL,**0**,**"lab.03.mtx"**);**

int pn **=** atoi**(**argv**[**1**]);**

int**\*** a **=** **new** int**[**N**+**5**];**

fill**(**a**,**N**);**

sort**(**a**,**N**);**

WaitForSingleObject**(**mtx**,**INFINITE**);**

print**(**a**,**N**,**pn**);**

ReleaseMutex**(**mtx**);**

**return** 0**;**

**}**

Программа Master\_crt:

#include "stdafx.h"

**using** **namespace** std**;**

const int T **=** 6**;**

const int N **=** 45**;**

CRITICAL\_SECTION crt**;**

void fill**(**int**\*** a**,**int n**)** **{**

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

a**[**i**]** **=** rand**()%**10000**;**

**}**

**}**

void sort**(**int**\*** a**,** int n**)** **{**

int i**,**j**,**k**;**

int t**;**

**for(**k **=** n**/**2**;** k **>** 0**;** k **/=**2**)**

**for(**i **=** k**;** i **<** n**;** i**++)** **{**

t **=** a**[**i**];**

**for(**j **=** i**;** j**>=**k**;** j**-=**k**)** **{**

**if(**t **<** a**[**j**-**k**])**

a**[**j**]** **=** a**[**j**-**k**];**

**else**

**break;**

**}**

a**[**j**]** **=** t**;**

**}**

**}**

void print**(**int**\*** a**,**int n**,**int p**)** **{**

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

cout **<<** p **<<** ":" **<<** a**[**i**]** **<<** " "**;**

**}**

cout **<<** endl**;**

**}**

DWORD WINAPI test**(**const LPVOID lpParam**)** **{**

int**\*** a **=** **new** int**[**N**+**5**];**

fill**(**a**,**N**);**

sort**(**a**,**N**);**

EnterCriticalSection**(&**crt**);**

print**(**a**,**N**,\*(**int**\*)**lpParam**);**

LeaveCriticalSection**(&**crt**);**

**return** 0**;**

**}**

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

int i**;**

char window**[**64**];**

InitializeCriticalSection**(&**crt**);**

int pid **=** GetCurrentProcessId**();**

cout **<<** "Master: Starting PID: " **<<** pid **<<** endl**;**

cout**.**flush**();**

STARTUPINFO info**;**

memset**(&**info**,** 0**,** **sizeof(**info**));**

info**.**cb **=** **sizeof(**info**);**

**for(**i**=**1**;**i**<=**T**;**i**++)** **{**

int**\*** z **=** **new** int**(**i**);**

CreateThread**(NULL,**0**,&**test**,**z**,**0**,NULL);**

**}**

Sleep**(**5000**);**

system**(**"pause"**);**

cout **<<** "Master: Exiting\n"**;**

**return** 0**;**

**}**

1. Результат

Результат работы программы Master\_mtx показан на рисунке 1.

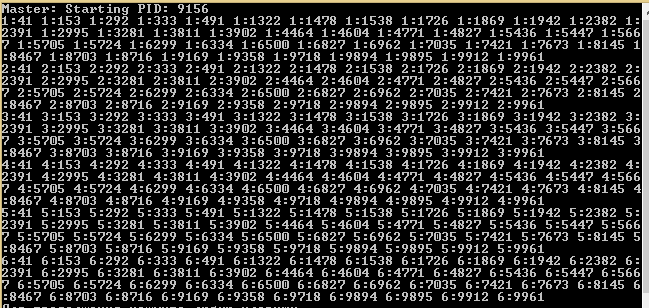


Рисунок 1 – запуск Master\_mtx

Результат работы программы Master\_crt показан на рисунке 2.

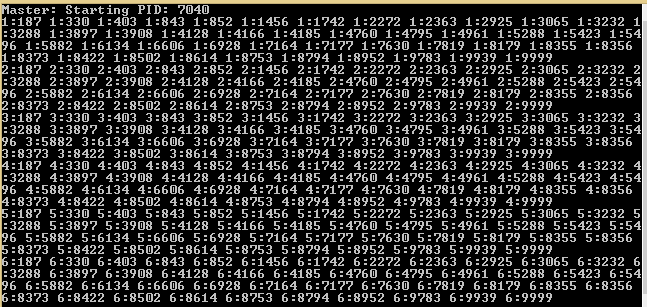


Рисунок 2 – запуск Master\_crt

Выводы

# В ходе лабораторной работы были изучены методы синхронизации между процессами и потоками средствами Mutex и CriticalSection WinAPI и языка Visual C++. Использование Mutex возможно в пределах древа процессов в случае начального создания мьютекса без запрета наследования, в то время как критические секции работают в пределах потоков одного процесса.