**3 ЛАБОРАТОРНАЯ РАБОТА №3**

**«Использование программного интерфейса Win API.**

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

**3.1 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.

**3.2 Вариант задания – 17**

3.2.1 Написать программу **Sort3**, реализующую следующий алгоритм:

3.2.1.1 Получить в качестве параметра командной строки номер процесса N\_PRC;

3.2.1.2 Заполнить массив **N = 95** целыми числами случайными значениями из диапазона 0-100;

3.2.1.3 Отсортировать массив методом **вставки** по **возрастанию**;

3.2.1.4 Вывести на экран отсортированный массив в формате N PRC:A[i] (то есть перед выводом каждого элемента массива необходимо выводить значение параметра N\_PRC);

3.2.2 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 = 3**.

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.

3.2.3 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**).

3.2.4 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”.

**3.3 Ход работы**

3.3.1 Написаны программы на языке си и их тексты представлены ниже.

3.3.1.1 Программа Sort3.c

#include <stdio.h>

#include <windows.h>

#include <time.h>

#define N 95 // по варианту N = 95

//заполнить массив целыми числами из диапазона 0-100

void setArray(int \*arr) {

srand(time(NULL));

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

arr[i] = rand() % 101;

}

//отсортировать массив методом вставки по возрастанию

void sortArray(int \*arr) {

int counter = 0;

for (int i = 1; i < N; i++) {

for (int j = i; (j > 0) && (arr[j-1] > arr[j]); j--) {

counter++;

int tmp = arr[j-1];

arr[j-1] = arr[j];

arr[j] = tmp;

}

}

}

//вывести массив на экран

void showArray(int \*arr, char const \*string) {

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

printf("%s: A[%2d] = %3d;\n", string, i+1, arr[i]);

}

}

//=====MAIN=====

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

// если в качестве аргумента коммандной строки нет ничего то программа завершается

if (argc != 2) return 1;

//создаем массив целых чисел размером N (95)

int arr[N];

//заполнить массив целыми числами из диапазона 0-100

setArray(arr);

//отсортировать массив методом вставки по возрастанию

sortArray(arr);

//вывести массив на экран

showArray(arr, argv[1]);

// system("pause");

return 0;

}

3.3.1.2 Программа Master.c

#include <stdio.h>

#include <windows.h>

#define COL\_PRC 3 //кол-во создаваемых процессов по варианту

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

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

// Создать 3 процесса Sort3 передавая в качестве параметра

// командной строки номер процесса – N\_PRC

// Необходимо, чтобы все процессы использовали одну консоль

// и имели класс приоритета NORMAL\_PRIORITY\_CLASS

char cmd[255];

STARTUPINFO s[COL\_PRC];

PROCESS\_INFORMATION p[COL\_PRC];

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

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

for (int i = 0; i < COL\_PRC; i++) {

s[i].cb = sizeof(STARTUPINFO); // должно быть записано, какой размер имеет стуктура

s[i].lpTitle = calloc(sizeof(char), sprintf(cmd, "Proc%d", i + 1));

strcpy(s[i].lpTitle, cmd);

}

for (int i = 0; i < COL\_PRC; i++) {

sprintf(cmd, ".\\sort3.exe Proc\_%d", i+1);

if (!CreateProcess (NULL, // имя приложения

cmd, // коммандная строка

NULL, // атрибуты доступа процесса

NULL, // атрибуты доступа потока

FALSE, // наследование дескрипторов (не наследовать)

NORMAL\_PRIORITY\_CLASS, // класс приоритета

NULL, // наследует среду окружения родительского процесса

NULL, // текущая папка родительского процесса будет унаследована порождаемым процессом

&s[i],

&p[i]) ) {

printf("Master: %d Процесс Sort не запущен\a\n", i+1);

}

}

WaitForSingleObject(p[0].hProcess, INFINITE);

WaitForSingleObject(p[1].hProcess, INFINITE);

WaitForSingleObject(p[2].hProcess, INFINITE);

system("pause");

return 0;

}

3.3.1.3 Программа SORT3\_mtx.c

#include <stdio.h>

#include <windows.h>

#include <time.h>

#define N 95 // по варианту N = 95

//заполнить массив целыми числами из диапазона 0-100

void setArray(int \*arr) {

srand(time(NULL));

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

arr[i] = rand() % 101;

}

//отсортировать массив методом вставки по возрастанию

void sortArray(int \*arr) {

int counter = 0;

for (int i = 1; i < N; i++) {

for (int j = i; (j > 0) && (arr[j-1] > arr[j]); j--) {

counter++;

int tmp = arr[j-1];

arr[j-1] = arr[j];

arr[j] = tmp;

}

}

}

//вывести массив на экран

void showArray(int \*arr, char const \*string) {

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

printf("%s: A[%2d] = %3d;\n", string, i+1, arr[i]);

}

}

//=====MAIN=====

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

// если в качестве аргумента коммандной строки нет ничего то программа завершается

if (argc != 2) return 1;

//создаем массив целых чисел размером N (95)

int arr[N];

//OpenMutex(Задает запрошенный доступ к объекту мьютекса, наследуется ли возвращаемый дескриптор, имя);

HANDLE mtx = OpenMutex(MUTEX\_ALL\_ACCESS, FALSE, "MyMutexSort"); //+

//заполнить массив целыми числами из диапазона 0-100

setArray(arr);

//отсортировать массив методом вставки по возрастанию

sortArray(arr);

// Ожидать свободный мьютекс

WaitForSingleObject(mtx, INFINITE); //+

//вывести массив на экран

showArray(arr, argv[1]);

// Освободить мьютекс и завершить программу

ReleaseMutex(mtx); //+

// system("pause");

return 0;

}

3.3.1.4 Программа MASTER3\_mtx.c

#include <stdio.h>

#include <windows.h>

#define COL\_PRC 3 //кол-во создаваемых процессов по варианту

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

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

// Создать 3 процесса Sort3 передавая в качестве параметра

// командной строки номер процесса – N\_PRC

// Необходимо, чтобы все процессы использовали одну консоль

// и имели класс приоритета NORMAL\_PRIORITY\_CLASS

char cmd[255];

STARTUPINFO s[COL\_PRC];

PROCESS\_INFORMATION p[COL\_PRC];

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

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

HANDLE mtx = CreateMutex(NULL, TRUE, "MyMutexSort"); //+

for (int i = 0; i < COL\_PRC; i++) {

s[i].cb = sizeof(STARTUPINFO); // должно быть записано, какой размер имеет стуктура

s[i].lpTitle = calloc(sizeof(char), sprintf(cmd, "Proc%d", i + 1));

strcpy(s[i].lpTitle, cmd);

}

for (int i = 0; i < COL\_PRC; i++) {

sprintf(cmd, ".\\sort3\_mtx.exe Proc\_%d", i+1);

if (!CreateProcess (NULL, // имя приложения

cmd, // коммандная строка

NULL, // атрибуты доступа процесса

NULL, // атрибуты доступа потока

FALSE, // наследование дескрипторов (не наследовать)

NORMAL\_PRIORITY\_CLASS, // класс приоритета

NULL, // наследует среду окружения родительского процесса

NULL, // текущая папка родительского процесса будет унаследована порождаемым процессом

&s[i],

&p[i]) ) {

printf("Master: %d Процесс Sort не запущен\a\n", i+1);

}

}

ReleaseMutex(mtx); //+

WaitForSingleObject(p[0].hProcess, INFINITE);

WaitForSingleObject(p[1].hProcess, INFINITE);

WaitForSingleObject(p[2].hProcess, INFINITE);

CloseHandle(mtx); //+

system("pause");

return 0;

}

3.3.1.5 Программа Master3\_crit.c

#include <stdio.h>

#include <windows.h>

#include <time.h>

#define N 95 // по варианту N = 95

#define COL\_THREAD 3 //кол-во создаваемых потоков по варианту

CRITICAL\_SECTION c = { 0 }; //Критическая секция

//заполнить массив целыми числами из диапазона 0-100

void setArray(int \*arr) {

srand(time(NULL));

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

arr[i] = rand() % 101;

}

//отсортировать массив методом вставки по возрастанию

void sortArray(int \*arr) {

int counter = 0;

for (int i = 1; i < N; i++) {

for (int j = i; (j > 0) && (arr[j-1] > arr[j]); j--) {

counter++;

int tmp = arr[j-1];

arr[j-1] = arr[j];

arr[j] = tmp;

}

}

}

//вывести массив на экран

void showArray(int \*arr, char const \*string) {

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

printf("%s: A[%2d] = %3d;\n", string, i+1, arr[i]);

}

}

DWORD WINAPI Sort (LPVOID p)

{

int arr[N];

setArray(arr);

sortArray(arr);

EnterCriticalSection(&c);

showArray(arr, (char \*) p);

LeaveCriticalSection(&c);

return 0;

}

int main ()

{

char tmp[COL\_THREAD][1024];

DWORD th[COL\_THREAD];

HANDLE hth[COL\_THREAD];

InitializeCriticalSection(&c);

for (int i = 0; i < COL\_THREAD; i++) {

sprintf(tmp[i], "Thr%d", i + 1);

hth[i] = CreateThread(NULL, 0, Sort, (LPVOID) (tmp + i), CREATE\_SUSPENDED, &th[i]);

}

for (int i = 0; i < COL\_THREAD; i++) ResumeThread(hth[i]);

WaitForMultipleObjects(COL\_THREAD, hth, TRUE, INFINITE); //ожидаем все потоки

DeleteCriticalSection(&c);

system("pause");

return 0;

}

3.3.2 Тестовые примеры продемонстрированы на рисунках 3.1 – 3.4.

Были написаны основная программа Master3 и дополнительная программа Sort3, запускаемая основной программой как дополнительный процесс. После запуска было обнаружено, что выводы 3 процессов «перемешиваются» друг с другом (Рисунок 3.1).

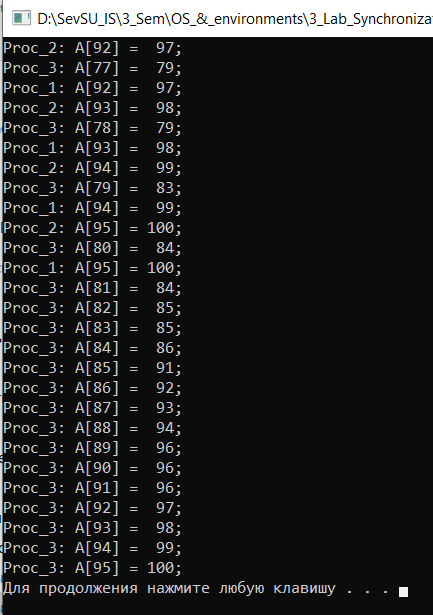


Рисунок 3.1 ­– Вывод программы Master3 (без мьютексов)

После чего программы были переписаны с использованием мьютексов. После запуска Master3\_mtx на экран были выведены данные, не «перемешанные» между собой, что продемонстрировано на рисунке 3.2.

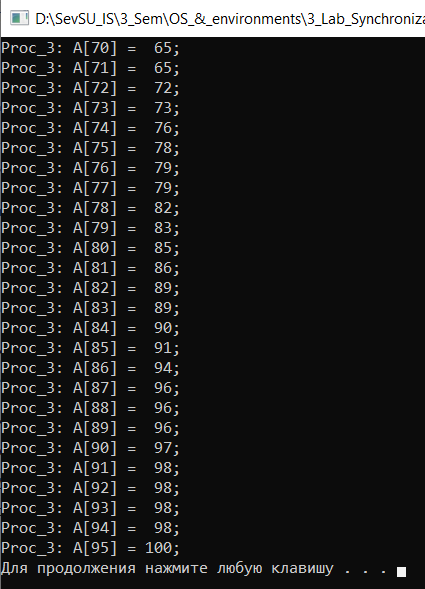


Рисунок 3.2 ­– Вывод программы Master3\_mtx (с использованием мьютексов)

Была написана программа, представленная в файле Master3\_crit.c. После запуска было обнаружено, что данные «перемешиваются» при выводе на экран (Рисунок 3.3). После чего программа была переписана с использованием критических секций, скомпилирована и запущена. Данные были выведена на экран, не «перемешавшись», что продемонстрировано на Рисунке 3.4.

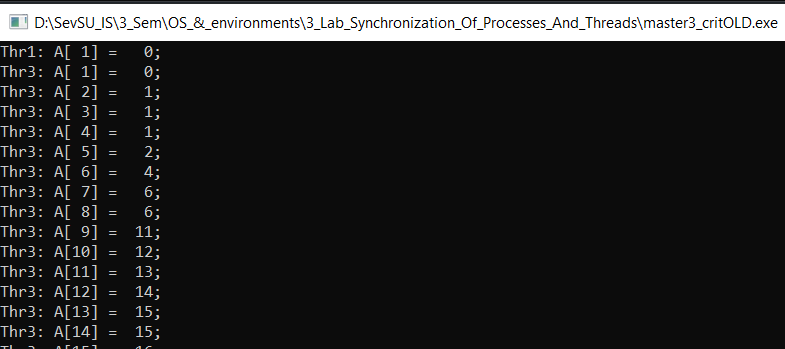


Рисунок 3.3 – Вывод программы без использования критических секций

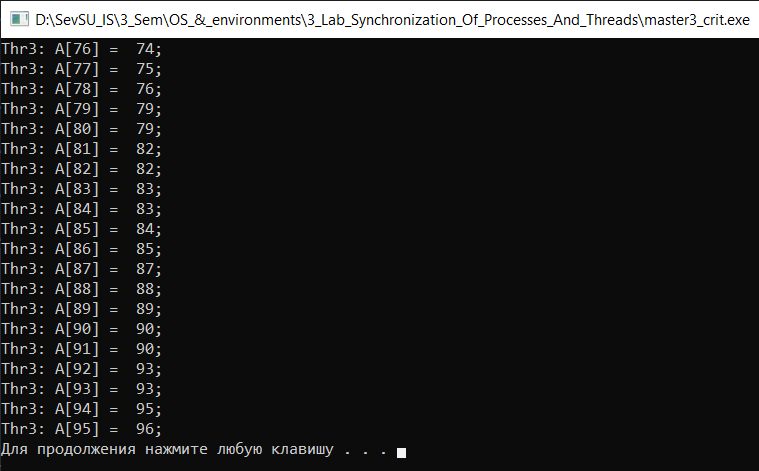


Рисунок 3.3 – Вывод программы с использованием критических секций

Результаты тестирования полностью соответствуют ожиданиям.

**Выводы**

При выполнении данной лабораторной работы были получены навыки разработки программ с мьютексами и критическими секциями с помощью средств WinAPI. Также были повторно закреплены навыки синхронизации процессов и потоков с помощью функции WaitForSingleObject. Полученные знания позволят создавать многопоточные приложения, которые на многоядерных системах будут работать эффективнее.