**2 ЛАБОРАТОРНАЯ РАБОТА №2**

**«Использование программного интерфейса Win API. Процессы и потоки в ОС Windows»**

**2.1 Цель работы**

Изучение возможности использования программного интерфейса приложений (API) операционных систем Windows 95-2000, NT, XP, 7. Приобретение практических навыков создания и управления процессами и потоками, используя Win API в средах программирования Borland Delphi, C++ Builder или Visual Studio.

**2.2 Вариант задания – 17**

2.2.1 Написать программу **Sort**, реализующую следующий алгоритм:

2.2.1.1 Зафиксировать время начала **Tstrt** выполнения программы.

2.2.1.2 Вывести на экран время **Tstrt** в формате: минуты:секунды:миллисекунды;

2.2.1.3 Зафиксировать текущий момент времени t1;

2.2.1.4 Для **i** от 1 до 100 (количество повторений может быть изменено в зависимости от быстродействия процессора) повторять:

2.2.1.4.1 Заполнить массив целых чисел случайными значениями из диапазона 0-10000;

2.2.1.4.2 Отсортировать массив;

2.2.1.5 Зафиксировать текущий момент времени t2;

2.2.1.6 Определить среднее время одной сортировки: (t2-t1)/100;

2.2.1.7 Вывести на экран среднее время одной сортировки (в миллисекундах);

2.2.1.8 Зафиксировать время окончания **Tend** выполнения программы;

2.2.1.9 Вывести на экран время **Tend** в формате: минуты:секунды:миллисекунды;

Язык программирования – Си, метод сортировки – сортировка выбора, направление сортировки – убывающая, количество элементов массива N = 3800.

2.2.2 Написать программу Master, выполняющую следующие действия:

2.2.2.1 Для **i** от 1 до 3 повторять:

2.2.2.2.1 Используя системные вызовы **CreateProcess**, создать два процесса **Sort** с классами приоритетов: IDLE\_PRIORITY\_CLASS - NORMAL\_PRIORITY\_CLASS, IDLE\_PRIORITY\_CLASS - HIGH\_PRIORITY\_CLASS, NORMAL\_PRIORITY\_CLASS - NORMAL\_PRIORITY\_CLASS. Необходимо, чтобы каждый процесс имел собственную консоль и окно консоли имело заголовок: "Process: NP; Prioritet: PP", где NP – номер процесса (1 или 2), а PP – приоритет соответствующего процесса. (Для изменения свойств окна консоли использовать структуру STARTUPINFO).

2.2.2.2.2 Ожидать окончания процессов, созданных в п. 2.2.2.2.1 (использовать функцию WaitForSingleObject).

2.2.3 Зафиксировать для отчета значения времени, получаемые при выполнении процессов в п 2.2.2.1.

2.2.4 Написать программу **Threads**, содержащую процедуру сортировки массива (разработанную при выполнении пункта 2.2.1), содержащего N/50 элементов и процедуру вывода массива на экран – mass\_print. Программа должна выполнять следующие действия:

2.2.4.1 Генерировать случайный массив, содержащий N/50 элементов.

2.2.4.2 Используя системные вызовы **CreateThread**, создать программные потоки **sort** и **mass\_print** в приостановленном состоянии.

2.2.4.3 Установить приоритеты потоков в THREAD\_PRIORITY\_NORMAL, используя системный вызов **SetThreadPriority**.

2.2.4.4 Активизировать потоки, используя системные вызовы **ResumeThread**;

2.2.4.5 Изменяя приоритеты потоков в разработанной программе в различных сочетаниях фиксировать получаемые результаты.

**2.3 Ход работы**

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

2.3.1.1 Программа Master

#include <stdio.h>

#include <locale.h>

#include <windows.h>

#include <string.h>

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

setlocale(LC\_ALL, "Rus");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

SYSTEMTIME Time;

printf("\nMASTER: Здравствуйте!\n\n");

// Если в качестве аргумента командной строки передовалось что то, то то-то...

char cmd[256];

if (argc!=1) {

strcpy(cmd, argv[1]);

} else {

strcpy(cmd, "sort.exe");

}

int procId = GetCurrentProcessId(); // Взять идентификатор вызывающего процесса

sprintf(cmd+strlen(cmd), " %d", procId); // cmd+идентификатор

printf("Мастер стартует: %s\n", cmd);

fflush(stdout); // Очистить буфер вывода

Sleep(3000); //спать 3 сек

STARTUPINFO si; // lpStartupInfo

memset(&si, 0, sizeof(si)); // Заполнение в info каждого байта 0

si.cb = sizeof(si); // должно быть записано, какой размер имеет стуктура

PROCESS\_INFORMATION pi; // lpProcessInformation

//=============================================--1--

si.lpTitle = "Process: 1.1; Prioritet: IDLE\_PRIORITY\_CLASS";

if (!CreateProcess (NULL, // имя приложения

cmd, // коммандная строка

NULL, // атрибуты доступа процесса

NULL, // атрибуты доступа потока

FALSE, // наследование дескрипторов (не наследовать)

CREATE\_NEW\_CONSOLE | IDLE\_PRIORITY\_CLASS, // класс приоритета

NULL, // наследует среду окружения родительского процесса

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

&si,

&pi) ) { printf("Master: 1.1 Процесс Sort не запущен\a\n"); }

si.lpTitle = "Process: 1.2; Prioritet: NORMAL\_PRIORITY\_CLASS";

if (!CreateProcess (NULL, // имя приложения

cmd, // коммандная строка

NULL, // атрибуты доступа процесса

NULL, // атрибуты доступа потока

FALSE, // наследование дескрипторов (не наследовать)

CREATE\_NEW\_CONSOLE | NORMAL\_PRIORITY\_CLASS, // класс приоритета

NULL, // наследует среду окружения родительского процесса

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

&si,

&pi) ) { printf("Master: 1.2 Процесс Sort не запущен\a\n");}

//pi->hProcess = Дескриптор главного потока созданного процесса

WaitForSingleObject(pi.hProcess, INFINITE);

GetSystemTime(&Time);

printf("Время выполнения программы после второй пары процессов: минуты:секунды:миллисекунды -> %02d:%02d:%03d \n\n",

Time.wMinute, Time.wSecond, Time.wMilliseconds);

//=============================================--2--

si.lpTitle = "Process: 2.1; Prioritet: IDLE\_PRIORITY\_CLASS";

if (!CreateProcess (NULL, // имя приложения

cmd, // коммандная строка

NULL, // атрибуты доступа процесса

NULL, // атрибуты доступа потока

FALSE, // наследование дескрипторов (не наследовать)

CREATE\_NEW\_CONSOLE | IDLE\_PRIORITY\_CLASS, // класс приоритета

NULL, // наследует среду окружения родительского процесса

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

&si,

&pi) ) { printf("Master: 2.1 Процесс Sort не запущен\a\n"); }

si.lpTitle = "Process: 2.2; Prioritet: HIGH\_PRIORITY\_CLASS";

if (!CreateProcess (NULL, // имя приложения

cmd, // коммандная строка

NULL, // атрибуты доступа процесса

NULL, // атрибуты доступа потока

FALSE, // наследование дескрипторов (не наследовать)

CREATE\_NEW\_CONSOLE | HIGH\_PRIORITY\_CLASS, // класс приоритета

NULL, // наследует среду окружения родительского процесса

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

&si,

&pi) ) {printf("Master: 2.2 Процесс Sort не запущен\a\n");}

//pi->hProcess = Дескриптор главного потока созданного процесса

WaitForSingleObject(pi.hProcess, INFINITE);

GetSystemTime(&Time);

printf("Время выполнения программы после второй пары процессов: минуты:секунды:миллисекунды -> %02d:%02d:%03d \n\n",

Time.wMinute, Time.wSecond, Time.wMilliseconds);

//=============================================--3--

si.lpTitle = "Process: 3.1; Prioritet: NORMAL\_PRIORITY\_CLASS";

if (!CreateProcess (NULL, // имя приложения

cmd, // коммандная строка

NULL, // атрибуты доступа процесса

NULL, // атрибуты доступа потока

FALSE, // наследование дескрипторов (не наследовать)

CREATE\_NEW\_CONSOLE | NORMAL\_PRIORITY\_CLASS, // класс приоритета

NULL, // наследует среду окружения родительского процесса

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

&si,

&pi) ) { printf("Master: 3.1 Процесс Sort не запущен\a\n"); }

si.lpTitle = "Process: 3.2; Prioritet: NORMAL\_PRIORITY\_CLASS";

if (!CreateProcess (NULL, // имя приложения

cmd, // коммандная строка

NULL, // атрибуты доступа процесса

NULL, // атрибуты доступа потока

FALSE, // наследование дескрипторов (не наследовать)

CREATE\_NEW\_CONSOLE | NORMAL\_PRIORITY\_CLASS, // класс приоритета

NULL, // наследует среду окружения родительского процесса

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

&si,

&pi) ) {printf("Master: 3.2 Процесс Sort не запущен\a\n"); }

//pi->hProcess = Дескриптор главного потока созданного процесса

WaitForSingleObject(pi.hProcess, INFINITE);

GetSystemTime(&Time);

printf("Время выполнения программы после третьей пары процессов: минуты:секунды:миллисекунды -> %02d:%02d:%03d \n\n",

Time.wMinute, Time.wSecond, Time.wMilliseconds);

system("pause");

return 0;

}

2.3.1.2 Программа сортировки

#include <stdio.h>

#include <locale.h>

#include <windows.h>

#include <time.h>

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

setlocale(LC\_ALL, "Rus");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

system("color B0");

srand(time(0));

unsigned int kolRepetitions = 100; //кол-во повторений

unsigned int numElementsArray = 3800; //кол-во элементов массива

// объявляем перменные в которых будем хранить время 0w0 !

SYSTEMTIME TimeStart, TimeEnd, time1, time2;

// Зафиксировать время начала выполенения программы

GetSystemTime(&TimeStart);

// Вывести на экран время Tstrt в формате минуты:секунды:миллисекунды

printf("Время начала выполнения программы: минуты:секунды:миллисекунды -> %02d:%02d:%03d \n\n",

TimeStart.wMinute, TimeStart.wSecond, TimeStart.wMilliseconds);

// Зафиксировать текущий момент времени t1

time1 = TimeStart;

int array[numElementsArray];

// Для i от 1 до 100 повторять

int i, i2, j, temp;

for (i = 0; i < kolRepetitions; i++){

// Заполнить массив целых чисел случайными числами из диапазона 0-10000

for (j = 0; j < numElementsArray; j++) {

array[j] = rand()%10000;

}

// Отсортировать массив (сортировка Выбора по убыванию)

int max;

for (i2 = 0; i2 < numElementsArray-1; i2++) {

for (max = i2, j = i2+1; j < numElementsArray; j++) {

if (array[j] > array[max]) {

max = j;

}

temp = array[max];

array[max] = array[i2];

array[i2] = temp;

}

}

}

// Зафиксировать текущий момент времени t2

GetSystemTime(&time2);

printf("Время выполнения программы после сортировок: минуты:секунды:миллисекунды -> %02d:%02d:%03d \n\n",

time2.wMinute, time2.wSecond, time2.wMilliseconds);

// Определить среднее вермя одной сортировки: (t2-t1)/100 и вывести на экран в мс

FILETIME fileTime1, fileTime2;

ULARGE\_INTEGER li1, li2, dif;

// Преобразовать SystemTime -> FileTime

SystemTimeToFileTime(&time1, &fileTime1);

SystemTimeToFileTime(&time2, &fileTime2);

// Преобразовать FileTime -> ULARGE\_INTEGER

li1.u.LowPart = fileTime1.dwLowDateTime;

li1.u.HighPart = fileTime1.dwHighDateTime;

li2.u.LowPart = fileTime2.dwLowDateTime;

li2.u.HighPart = fileTime2.dwHighDateTime;

// Вычитание t2-t1 и делим на кол-во сортировок

dif.QuadPart = (li2.QuadPart - li1.QuadPart) / 100;

printf("Для кол-ва элементов массива = %d и кол-ву сортировок = %d:\n\n", kolRepetitions, numElementsArray);

printf("среднее время выполнения одной сортировки = %03dms \n\n", dif.QuadPart/100);

// Зафиксировать время окончания TimeEnd выполнения программы и вывести на экран

GetSystemTime(&TimeEnd);

printf("Время окончания выполнения программы: минуты:секунды:миллисекунды -> %02d:%02d:%03d \n\n",

TimeEnd.wMinute, TimeEnd.wSecond, TimeEnd.wMilliseconds);

printf("\n");

Sleep(10000);

return 0;

}

2.3.1.3 Программа Threads

// LPSECURITY\_ATTRIBUTES lpThreadAttributes, //атрибуты доступа

// DWORD dwStackSize, //размер стека потока

// LPTHREAD\_START\_ROUTINE lpStartAddress, //функция потока

// LPVOID lpParameter, //параметр функции

// DWORD dwCreationFlags, //состояние потока

// LPDWORD lpThreadId; //идентификатор потока

#include <stdio.h>

#include <locale.h>

#include <windows.h>

#include <time.h>

#define numElementsArray (3800/50)

void setArray(int array[numElementsArray]) {

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

array[i] = rand()%10000;

}

}

void sort(int array[numElementsArray]) {

int max, j, i, temp;

for (i = 0; i < numElementsArray-1; i++) {

for (max = i, j = i+1; j < numElementsArray; j++) {

if (array[j] > array[max]) {

max = j;

}

temp = array[max];

array[max] = array[i];

array[i] = temp;

}

}

printf(" сортировка окончена ");

}

void mass\_print(int array[numElementsArray]) {

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

printf("%d ", array[i]);

}

printf("\n\n");

}

//==============================================================================

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

setlocale(LC\_ALL, "Rus");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

srand(time(0));

DWORD ti1, ti2; //идентификаторы потоков

HANDLE th1, th2; //дескрипторы потоков

int array[numElementsArray];

setArray(array);

th1 = CreateThread(NULL, 0, mass\_print, &array, CREATE\_SUSPENDED, &ti1); //CREATE\_SUSPENDED - создать приостановленный

th2 = CreateThread(NULL, 0, sort, &array, CREATE\_SUSPENDED, &ti2); //CREATE\_SUSPENDED - создать приостановленный

SetThreadPriority(th1, THREAD\_PRIORITY\_NORMAL); //печать массива

SetThreadPriority(th2, THREAD\_PRIORITY\_NORMAL); //сортировка

ResumeThread(th1);

ResumeThread(th2);

WaitForSingleObject(th1, INFINITE);

WaitForSingleObject(th2, INFINITE);

printf("\n\n");

system("pause");

return 0;

}

2.3.2 Тестовые примеры продемонстрированы на рисунках 2.1 – 2.3 (работа с процессами) и на рисунках 2.4 – 2.5 (работа с потоками).

На Рисунке 2.1 продемонстрировано выполнение программ с приоритетами IDLE (слева) и NORMAL (справа). Был сделан вывод, что выполнение процесса с приоритетом IDLE происходит чуть более медленно чем с приоритетом NORMAL.

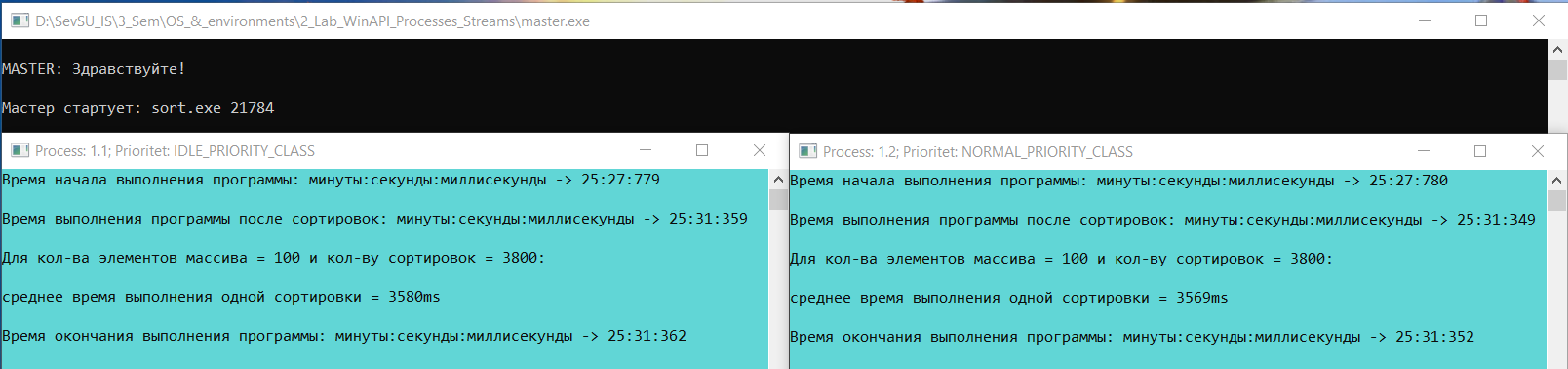


Рисунок 2.1 ­– Процессы с приоритетами IDLE и NORMAL

На Рисунке 2.2 продемонстрировано выполнение программ с приоритетами IDLE и HIGH (слева и справа соответственно). Был сделан вывод, что выполнение процесса с приоритетом IDLE происходит чуть более медленно чем с приоритетом HIGH.

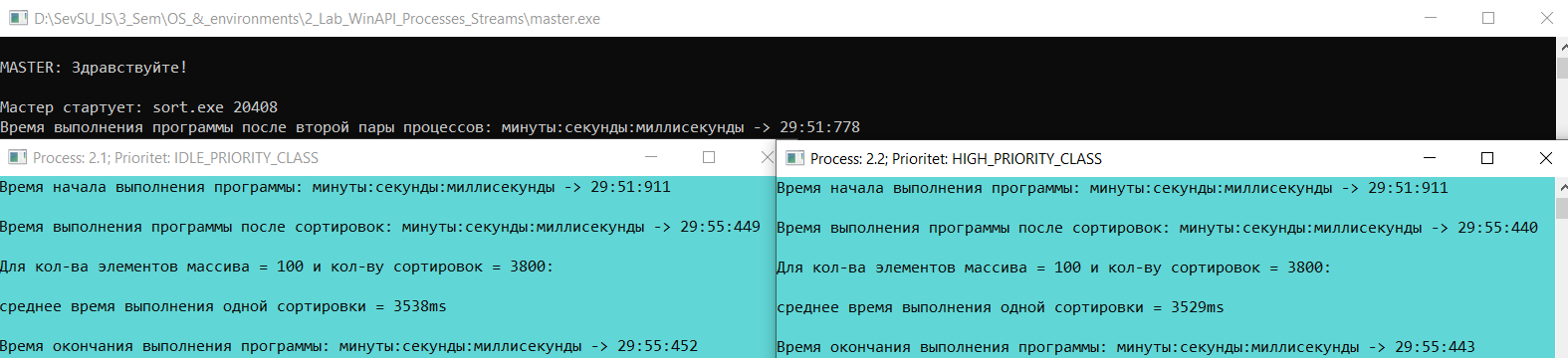


Рисунок 2.1 ­– Процессы с приоритетами IDLE и HIGH

На Рисунке 2.3 продемонстрировано выполнение программ с приоритетами NORMAL и NORMAL. Был сделан вывод, что вследствие одинакового приоритета на выполнение каждого процесса отведено одинаковое количество процессорного времени, а значит они завершат работу в одно и то же время при одинаковом времени работы.

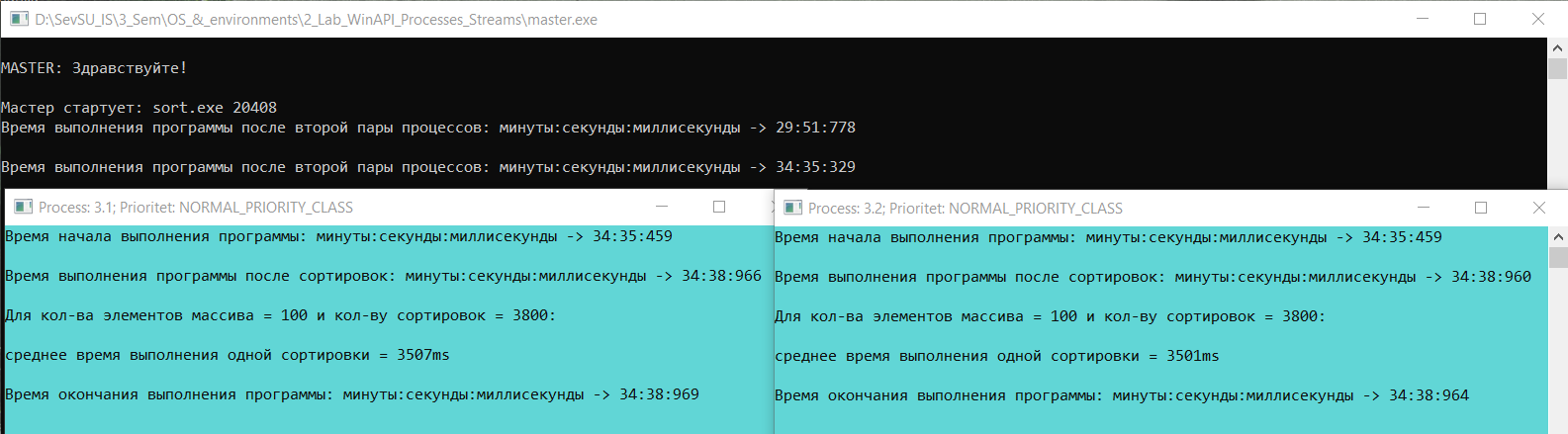


Рисунок 2.3 – Процессы с приоритетами NORMAL и NORMAL

На Рисунке 2.4 оба потока имеют приоритет NORMAL. Из результатов тестирования видно, что функция сортировки завершает свою работу до функции вывода массива на экран.

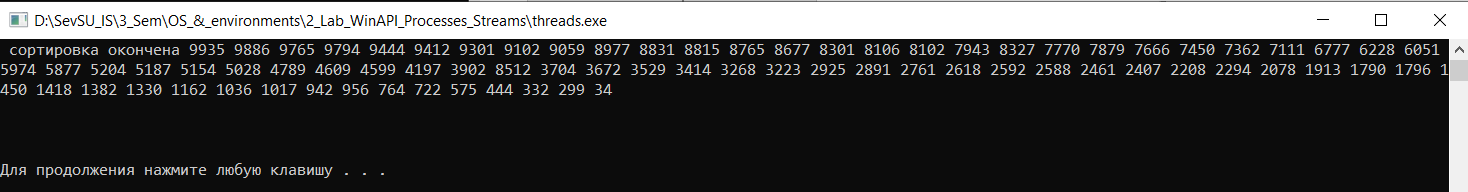


Рисунок 2.4 – Тест с потоками №1

На Рисунке 2.5 потоки имеют приоритеты TIME\_CRITICAL (печать массива) и NORMAL (сортировка массива). Из результатов тестирования видно, что функция сортировки завершает своё выполнение во время вывода массива на экран.

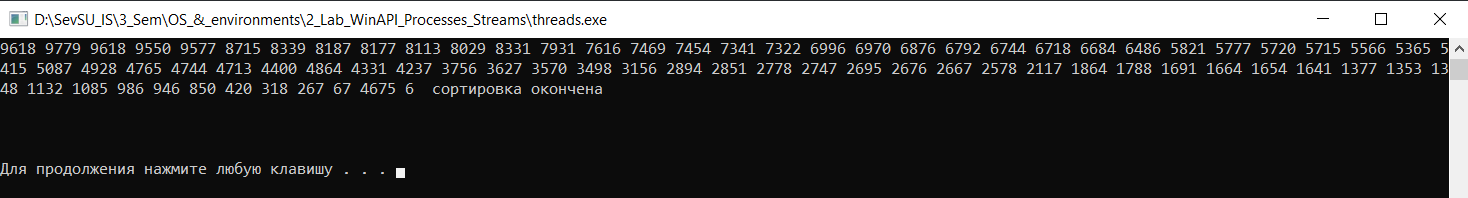


Рисунок 2.5 – Тест с потоками №2

**Выводы**

При выполнении данной лабораторной работы были получены навыки работы с процессами и потоками с помощью библиотеки WinAPI. Также были повторно закреплены навыки написания методов сортировки, работы с командной строкой Windows. Полученные знания позволят создавать многопоточные приложения, которые на многоядерных системах будут работать эффективнее.