# ЛАБОРАТОРНАЯ РАБОТА №2

# ИСПОЛЬЗОВАНИЕ ПРОГРАММНОГО ИНТЕРФЕЙСА WIN API.

# ПРОЦЕССЫ И ПОТОКИ В ОС WINDOWS

Цель работы

Изучение возможностей использования программного интерфейса приложений (API) операционных систем Windows 95-2000, NT, XP, 7. Приобретение практических навыков создания и управления процессами и потоками с помощью использования Win API в среде программирования Lazarus, Dev C++ или Visual Studio.

Задачи

1. Написать программу Sort, реализующую следующий алгоритм:
   1. Зафиксировать время начала Tstrt выполнения программы и вывести его на экран в формате минуты: секунды: миллисекунды;
   2. Зафиксировать текущий момент времени t1;
   3. Для i от 1 до 100 (количество повторений может быть изменено в зависимости от быстродействия процессора) повторять:
      1. Заполнить массив целых чисел случайными значениями из диапазона 0-10000;
      2. Отсортировать массив;
   4. Зафиксировать текущий момент времени t2;
   5. Определить среднее время одной сортировки: (t2-t1)/100 и вывести его на экран в миллисекундах;
   6. Зафиксировать время окончания Tend выполнения программы и вывести его на экран в формате: минуты: секунды: миллисекунды;

Язык программирования, метод и направление сортировки, а также количество элементов массива N выбрать в соответствии с вариантом задания (Таблица 11).

1. Написать программу Master, для i от 1 до 3 повторяющую следующие действия:
   1. Используя системные вызовы CreateProcess, создать два процесса Sort с классами приоритетов в соответствии с вариантом задания. Необходимо, чтобы каждый процесс имел собственную консоль и окно консоли имело заголовок: "Process: NP; Priority: PP", где NP – номер процесса (1 или 2), а PP – приоритет соответствующего процесса (для изменения свойств окна консоли использовать структуру STARTUPINFO);
   2. Ожидать окончания созданных процессов (использовать функцию WaitForSingleObject);
   3. Зафиксировать для отчета значения времени, получаемые при выполнении процессов;
2. Написать программу Threads, содержащую процедуру сортировки массива (разработанную при выполнении пункта 1), содержащего N/50 элементов и процедуру вывода массива на экран – mass\_print. Программа должна выполнять следующие действия:
   1. Генерировать случайный массив, содержащий N/50 элементов;
   2. Используя системные вызовы CreateThread, создать программные потоки sort и mass\_print в приостановленном состоянии;
   3. Установить приоритеты потоков в THREAD\_PRIORITY\_NORMAL, используя системный вызов SetThreadPriority;
   4. Активизировать потоки, используя системные вызовы ResumeThread;
   5. Изменяя приоритеты потоков в разработанной программе в различных сочетаниях фиксировать получаемые результаты;

Вариант задания

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № варианта | Среда программирования | Метод сортировки | Направление сортировки | N | Приоритеты\* |
| 11 | Dev C++ | Пузырёк | Возр. | 3200 | 3-2, 3-3, 1-3 |

Таблица 1 – Вариант задания

Приоритеты: 1 − IDLE\_PRIORITY\_CLASS; 2 − NORMAL\_PRIORITY\_CLASS; 3 − HIGH\_PRIORITY\_CLASS.

Ход работы

Перед началом работы были изучены принципы работы с процессами и потоками в операционной системе Windows, их обработка на языке C++.

Программа Sort была протестирована (Рисунок 1). В результате было выведено среднее время одной сортировки массива из 3200 чисел методом пузырька (22 мс.).

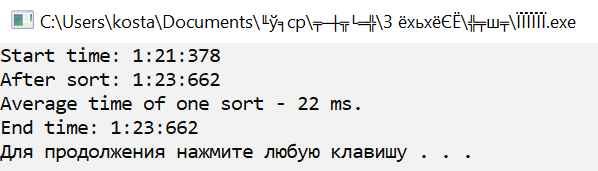


Рисунок 1 – Запуск программы Sort

Была запущена программа Master. Сначала было выполнено сравнение аналогичных процессов с приоритетами HIGH и NORMAL. В результате процесс с более высоким приоритетом был завершён раньше, чем с обычным (Рисунок 2).

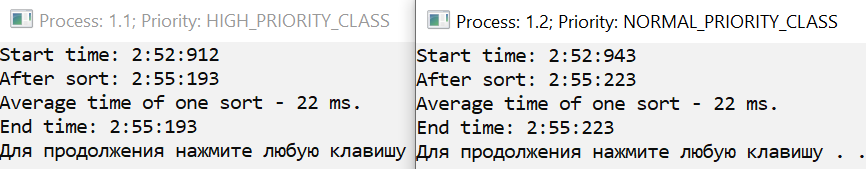


Рисунок 2 – Сравнение процессов с приоритетом HIGH и NORMAL

Затем были запущены два процесса с приоритетом HIGH. В результате скорость выполнения процессов не зависела от порядка их запуска (Рисунок 3).

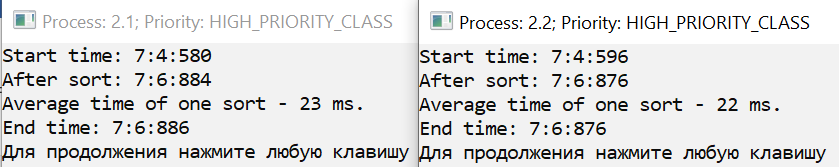


Рисунок 3 - Сравнение двух процессов с приоритетом HIGH

При сравнении процессов с приоритетами HIGH и IDLE был сделан вывод, что процессы с приоритетом IDLE происходят с большей задержкой, чем с приоритетом HIGH (Рисунок 4).

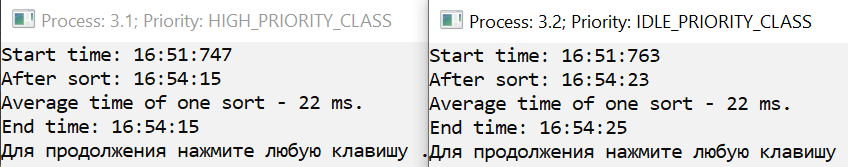


Рисунок 4 - Сравнение процессов с приоритетом HIGH и IDLE

Была запущена программа Threads. Сперва оба потока были запущены с приоритетом NORMAL (Рисунок 5). В результате процессы первой была выполнена та функция, вызов которой произошёл раньше (sort).

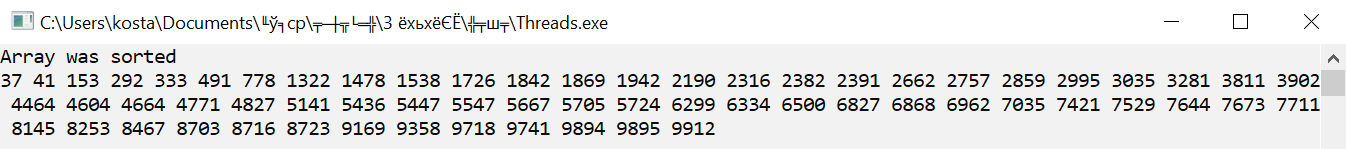


Рисунок 5 – Потоки с приоритетами NORMAL

Затем для потока функции sort был установлен приоритет IDLE. В результате функция sort успела отсортировать массив до вызова функции mass\_print, однако надпись о результате сортировки была выведена позже (Рисунок 6). Значит, в некоторый момент потоки начали выполняться одновременно.

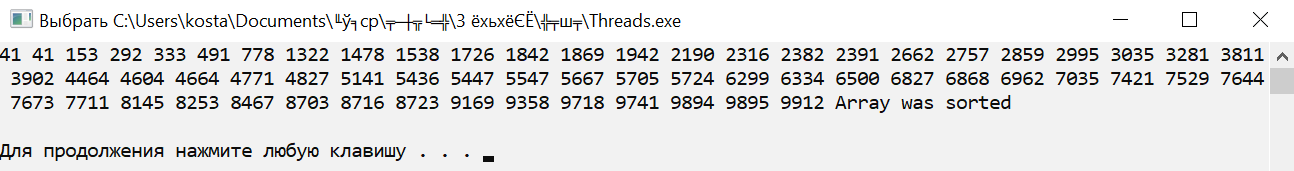


Рисунок 6 – Потоки с приоритетами IDLE и NORRMAL

При установке потоку функции mass\_print приоритета TIME\_CRITICAL потоки выполнялись практически одновременно (Рисунок 7).

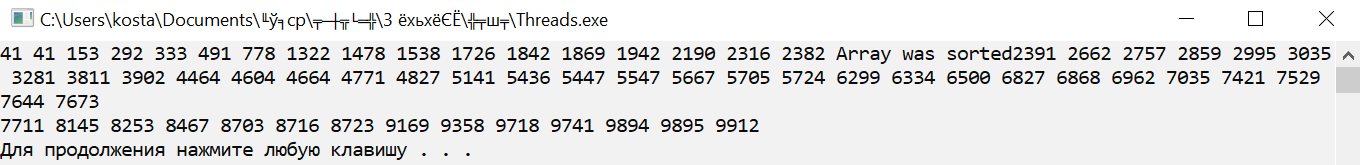


Рисунок 7 – Потоки с приоритетами IDLE и TIME\_CRITICAL

Текст программы

Программа Sort:

#include <windows.h>

#include <iostream>

using namespace std;

// Длина массива, задана вариантом

const int N = 3200;

// Функция сортировки массива методом пузырька

void SortIntArr(int mas[N]) {

int change;

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

for (int j = 0; j < N - i - 1; j++) {

if (mas[j] > mas[j + 1]) {

change = mas[j + 1];

mas[j + 1] = mas[j];

mas[j] = change;

}

}

}

return;

}

int main() {

// Переменные для хранения времени

SYSTEMTIME Tsrt, t1, t2, Tend;

// Время начала выполнения программы

GetSystemTime(&Tsrt);

cout << "Start time: " << Tsrt.wMinute << ":";

cout << Tsrt.wSecond << ":" << Tsrt.wMilliseconds << '\n';

// Время начала выполнения сортировок

GetSystemTime(&t1);

int arr[N];

// Заполнение массива случайными значениями и его сортировка (100 повторов)

for (int j = 0; j < 100; j++) {

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

arr[i] = rand() % 10000;

}

SortIntArr(arr);

}

// Время окончания цикла сортировок

GetSystemTime(&t2);

cout << "After sort: ";

cout << t2.wMinute << ":" << t2.wSecond << ":" << t2.wMilliseconds << '\n';

// Перевод t1 и t2 в 64-битовый формат

FILETIME ft1, ft2;

ULARGE\_INTEGER li1, li2, delta;

SystemTimeToFileTime(&t1, &ft1);

SystemTimeToFileTime(&t2, &ft2);

li1.u.LowPart = ft1.dwLowDateTime;

li1.u.HighPart = ft1.dwHighDateTime;

li2.u.LowPart = ft2.dwLowDateTime;

li2.u.HighPart = ft2.dwHighDateTime;

// Поиск среднего времени одной сортировки в мсек. (t2 - t1) / 100

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

cout << "Average time of one sort - " << delta.QuadPart / 10000 << " ms.\n";

// Время окончания выполнения программы

GetSystemTime(&Tend);

cout << "End time: " << Tend.wMinute << ":";

cout << Tend.wSecond << ":" << Tend.wMilliseconds << '\n';

return 0;

}

Программа Master:

#include <windows.h>

#include <iostream>

#include <string.h>

using namespace std;

// Приоритеты процессов, заданные вариантом

const int prts[3][3] = { {3, 2}, {3, 3}, {1, 3} };

int main() {

char cmd[10] = "Sort.exe"; // имя запускаемой программы

// Данные запускаемого процесса

STARTUPINFO si, sy;

PROCESS\_INFORMATION pi;

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

si.cb = sizeof(si);

// Массив с названиями классов приоритетов

char prtClass[3][30];

strcpy(prtClass[0], "IDLE\_PRIORITY\_CLASS");

strcpy(prtClass[1], "NORMAL\_PRIORITY\_CLASS");

strcpy(prtClass[2], "HIGH\_PRIORITY\_CLASS");

SYSTEMTIME time;

// Цикл запуска процессов Sort.exe

for (int i = 0; i < 3; i++) {

for (int j = 0; j <= 1; j++) {

// Заполнение заголовка процесса

char t[200] = "Process: ";

sprintf(t + strlen(t), "%d.%d; Priority: %s", i + 1, j + 1, prtClass[prts[i][j] - 1]);

si.lpTitle = t;

// Вызов процесса с определённым классом приоритета

switch (prts[i][j]) {

case 1: CreateProcess(NULL, cmd, NULL, NULL, FALSE, CREATE\_NEW\_CONSOLE | IDLE\_PRIORITY\_CLASS, NULL, NULL, &si, &pi); break;

case 2: CreateProcess(NULL, cmd, NULL, NULL, FALSE, CREATE\_NEW\_CONSOLE | NORMAL\_PRIORITY\_CLASS, NULL, NULL, &si, &pi); break;

case 3: CreateProcess(NULL, cmd, NULL, NULL, FALSE, CREATE\_NEW\_CONSOLE | HIGH\_PRIORITY\_CLASS, NULL, NULL, &si, &pi); break;

}

}

// Ожидание конца процессов и вывод текущего времени

WaitForSingleObject(pi.hProcess, INFINITE);

GetSystemTime(&time);

cout << "End of processes " << i << ": ";

cout << time.wMinute << "." << time.wSecond << ".";

cout << time.wMilliseconds << endl;

}

}

Программа Threads:

#include <windows.h>

#include <iostream>

using namespace std;

// Длина массива, заданная вариантом

const int N = 3200 / 50;

// Функция сортировки массива

LPTHREAD\_START\_ROUTINE sort(int\* mas) {

int swap;

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

for (int j = 0; j < N - i - 1; j++) {

if (mas[j] > mas[j + 1]) {

swap = mas[j + 1];

mas[j + 1] = mas[j];

mas[j] = swap;

}

}

}

cout << "Array was sorted" << endl;

return 0;

}

// Функция вывода массива на экран

LPTHREAD\_START\_ROUTINE mass\_print(int\* mas) {

for (int i = 0; i < N - 1; i++)

cout << mas[i] << " ";

cout << endl;

return 0;

}

int main() {

int arr[N];

// Заполнение массива случайными значениями

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

arr[i] = rand() % 10000;

DWORD id1, id2; // идентификаторы потоков

HANDLE th1, th2; // дескрипторы потоков

// Создание приостановленных потоков процедур sort и mass\_print

th1 = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)sort, (LPVOID)arr, CREATE\_SUSPENDED, &id1);

th2 = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)mass\_print, (LPVOID)arr, CREATE\_SUSPENDED, &id2);

// Задание приоритетов потоков

SetThreadPriority(th1, THREAD\_PRIORITY\_NORMAL); // сортировка

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

// Активирование потоков

ResumeThread(th1);

ResumeThread(th2);

// Ожидание окончания потоков

WaitForSingleObject(th1, INFINITE);

WaitForSingleObject(th2, INFINITE);

system("pause");

return 0;

}

Вывод

В ходе работы были исследованы системные услуги программного интерфейса приложений Win API. Изучены методы управления процессами и потоками при написании программ. В результате был написан ряд программ на языке C++, вызывающих процессы и потоки с различными приоритетами и вычисляющих время их выполнения.