# ЛАБОРАТОРНАЯ РАБОТА №5

# ИССЛЕДОВАНИЕ ВОЗМОЖНОСТЕЙ УПРАВЛЕНИЯ ПАМЯТЬЮ И ОБМЕНА ДАННЫМИ МЕЖДУ ПРОЦЕССАМИ В ОС WINDOWS

Цель работы

Изучить возможности программного интерфейса приложений операционных систем Windows по управления памятью и обмена данными между процессами. Приобрести практические навыки использования для управления памятью и обмена данными между процессами.

Задачи

1. Написать программу MemShare, выполняющую следующие действия:
   1. Создание (запуск) процесса MemSort (действия, выполняемые этим процессом описаны в п. 2) с помощью вызова CreateProcess;
   2. Выделение с помощью вызова VirtualAlloc заданного по варианту задания количества страниц памяти (страница 4096 байт);
   3. Заполнение выделенной памяти случайными числами в диапазоне от 0 до MAX (значение MAX выбирается исходя из типа данных, заданного по варианту задания);
   4. Перевод режима доступа выделенной памяти в ReadOnly (с помощью вызова VirtualProtect);
   5. Создание памяти общего доступа с размером, соответствующим размеру выделенной с помощью VirtualAlloc памяти (задать имя создаваемого объекта отображения "memshare"), при помощи механизма отображения файлов на память (функции CreateFileMapping и MapViewOfFile);
   6. Копирование данных из памяти, выделенной с помощью VirtualAlloc, в память общего доступа (например, с помощью вызова CopyMemory);
   7. Ожидание процесса MemSort (используя заданный по варианту задания объект синхронизации необходимо ожидать пока процесс MemSort не подготовит данные);
   8. Перевод режима доступа выделенной в п. 1.1 памяти в ReadWrite (с помощью вызова VirtualProtect);
   9. Копирование данных из памяти общего доступа в память, выделенную с помощью VirtualAlloc (например, с помощью вызова CopyMemory);
   10. Вывод на экран данных из памяти, выделенной с помощью VirtualAlloc;
   11. Освобождение выделенной памяти (с помощью вызова VirtualFree);
   12. Освобождение памяти общего доступа (с помощью вызовов UnmapViewOfFile и СloseHandle);
2. Написать программу MemSort выполняющую следующие действия:
   1. Ожидание пока процесс MemShare не подготовит данные в памяти общего доступа (использовать заданный по варианту задания тип объекта синхронизации);
   2. Открытие памяти общего доступа "memshare" (используя вызовы OpenFileMapping и MapViewOfFile);
   3. Сортировка данных в памяти общего доступа методом, указанным в варианте задания;

Вариант задания

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № варианта | Количество страниц памяти | Тип данных для заполнения памяти | Тип сортировки | Объект синхронизации |
| 11 | 3 | short | Вставка | Мьютекс |

Таблица 1 – Вариант задания

Ход работы

Была запущена программа MemShare. В результате на экран был выведен отсортированный массив чисел в указанном вариантом диапазоне (Рисунок 1).

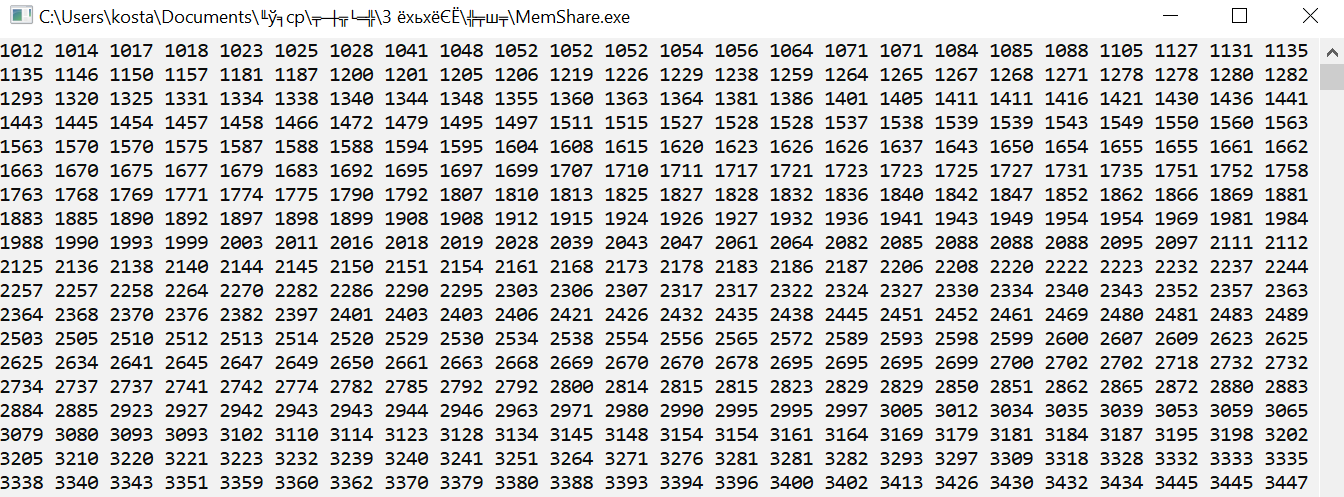


Рисунок 1 – Тестирование программы

Числа в массиве не превышают 32.768, что соответствуют положительной части типа short (2 байта). Также в ходе проверок было выяснено, что длина массива составила 6144 элемента, что также соответствуют поставленным условиям (3 страницы, 2 байта на элемент).

Результаты тестирования полностью соответствуют ожиданиям, что свидетельствует о правильности выполнения программы.

Текст программы

Программа MemShare:

#include <iostream>

#include <windows.h>

#include <climits>

using namespace std;

// Константы

const int N\_PAGE = 3; // кол-во страниц (задано вариантом)

const int PAGE\_SIZE = 4096; // размер страницы в байтах

const int TOTAL\_SIZE = N\_PAGE \* PAGE\_SIZE; // память под все страницы

// Начальные данные о типе массива

typedef short arr\_type;

const int MAX = SHRT\_MAX;

const int ARR\_LEN = TOTAL\_SIZE / sizeof(arr\_type);

// Гл. функция заполняет массив, вызывает процесс сортировки и выводит на экран

int main()

{

HANDLE mutex = CreateMutex(NULL, TRUE, "MyMytex");

// запуск процесса MemSort (в режиме ожидания)

STARTUPINFO si = { 0 };

si.cb = sizeof(STARTUPINFO);

PROCESS\_INFORMATION pi = { 0 };

CreateProcess(NULL, "MemSort.exe", 0, NULL, FALSE, 0, NULL, NULL, &si, &pi);

// выделение памяти под массив и его заполнение

LPVOID base\_mem = VirtualAlloc(NULL, TOTAL\_SIZE, MEM\_COMMIT, PAGE\_READWRITE);

arr\_type\* arr = (arr\_type\*)base\_mem;

for (int i = 0; i < ARR\_LEN; i++)

{

arr[i] = rand() % MAX;

}

// перевод режима доступа (только для чтения)

DWORD old\_protec;

VirtualProtect(base\_mem, TOTAL\_SIZE, PAGE\_READONLY, &old\_protec);

// создание объекта отображения для памяти общего доступа

HANDLE file\_map = CreateFileMapping(INVALID\_HANDLE\_VALUE, NULL, PAGE\_READWRITE, 0, TOTAL\_SIZE, "memshare");

LPVOID pub = MapViewOfFile(file\_map, FILE\_MAP\_ALL\_ACCESS, 0, 0, TOTAL\_SIZE);

// перенос данных из массива

CopyMemory(pub, base\_mem, TOTAL\_SIZE);

// запуск сортировки MemSort

ReleaseMutex(mutex);

WaitForSingleObject(pi.hProcess, INFINITE);

// вернуть возможность записи в массив

VirtualProtect(base\_mem, TOTAL\_SIZE, PAGE\_READWRITE, &old\_protec);

CopyMemory(base\_mem, pub, TOTAL\_SIZE);

// вывод массива

for (int i = 0; i < ARR\_LEN; i++)

{

cout << arr[i] << " ";

}

// освобождение памяти

VirtualFree(base\_mem, TOTAL\_SIZE, MEM\_RELEASE);

UnmapViewOfFile(pub);

CloseHandle(file\_map);

return 1;

}

Программа MemSort:

#include <windows.h>

#include <climits>

// Константы

const int N\_PAGE = 3; // кол-во страниц (задано вариантом)

const int PAGE\_SIZE = 4096; // размер страницы в байтах

const int TOTAL\_SIZE = N\_PAGE \* PAGE\_SIZE; // память под все страницы

// Начальные данные о типе массива

typedef short arr\_type;

const int MAX = SHRT\_MAX;

const int ARR\_LEN = TOTAL\_SIZE / sizeof(arr\_type);

// Функция сортировки массива (метод вставки)

void sort(arr\_type\* arr, int len)

{

for (int i = 1; i < len; i++)

{

int temp = arr[i];

int j = i - 1;

while ((j >= 0) && (arr[j] > temp))

{

arr[j + 1] = arr[j];

arr[j] = temp;

j--;

}

}

}

// Главная функция

int main()

{

// ожидание освобождения мьютекса

HANDLE mutex = OpenMutex(MUTEX\_ALL\_ACCESS, FALSE, "MyMutex");

WaitForSingleObject(mutex, INFINITE);

// открыть память общего доступа

HANDLE shared = OpenFileMapping(FILE\_MAP\_ALL\_ACCESS, FALSE, "memshare");

arr\_type\* arr = (arr\_type\*)MapViewOfFile(shared, FILE\_MAP\_ALL\_ACCESS, 0, 0, TOTAL\_SIZE);

sort(arr, ARR\_LEN);

// освобождение мьютекса и памяти

ReleaseMutex(mutex);

UnmapViewOfFile((LPVOID)arr);

CloseHandle(shared);

return 1;

}

Вывод

В ходе работы были изучены возможности программного интерфейса приложений Win API управления памятью и обмена данными между процессами. Были освоены навыки выделения и резервирования памяти, создания и работы с общей памятью при написании многопоточных приложений.