Министерство образования и науки Российской Федерации

Севастопольский государственный университет

Институт информационных технологий и управления в технических системах

ОТЧЕТ

по лабораторной работе №5

Исследование возможностей управления памятью и обмена данными между процессами в ОС Windows

Выполнил: ст. гр. ИС/б-21

Куркчи А.Э.

Проверил:

Шишкевич В.Е.

Севастополь

2015

# Цель

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

1. Постановка задачи

2.1. Написать программу MemShare, выполняющую следующие действия:

2.1.1. Создание(запуск) процесса MemSort(действия, выполняемые этим процессом описаны в п. 3.2), используя вызов CreateProcess;

2.1.2. Выделение с помощью вызова VirtualAlloc заданного по варианту задания количества страниц памяти (страница 4096 байт);

2.1.3. Заполнение выделенной памяти случайными числами в диапазоне от 0 до MAX (значение MAX выбирается исходя из типа данных, заданному по варианту задания);

2.1.4. Перевод режима доступа выделенной памяти в ReadOnly (с помощью вызова VirtualProtect);

2.1.5. Используя механизм отображения файлов на память (функции CreateFileMapping и MapViewOfFile), создание памяти общего доступа, с размером соответствующим размеру выделенной с помощью VirtualAlloc памяти (задать имя создаваемого объекта отображения "memshare");

2.1.6. Копирование данных из памяти, выделенной с помощью VirtualAlloc, в память общего доступа (например, с помощью вызова CopyMemory);

2.1.7. Ожидание процесса MemSort (используя заданный по варианту задания объект синхронизации необходимо ожидать пока процесс MemSort не подготовит данные);

2.1.8. Перевод режима доступа выделенной в п. 3.1.1 памяти в ReadWrite(с помощью вызова VirtualProtect);

2.1.9. Копирование данных из памяти общего доступа в память, выделенную с помощью VirtualAlloc (например, с помощью вызова CopyMemory);

2.1.10. Вывод на экран данных из памяти, выделенной с помощью VirtualAlloc;

2.1.11. Освобождение выделенной памяти (с помощью вызова VirtualFree);

2.1.12. Освободить память общего доступа (используя вызовы UnmapViewOfFile и СloseHandle);

2.2. Написать программу MemSort, выполняющую следующие действия:

2.2.1. Ожидание, пока процесс MemShare не подготовит данные в памяти общего доступа (использовать заданный по варианту задания тип объекта синхронизации);

2.2.2. Открытие памяти общего доступа "memshare" (используя вызовы OpenFileMapping и MapViewOfFile);

2.2.3. Сортировка данных в памяти общего доступа методом, указанным в варианте задания

Вариант №12

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № варианта | Количество страниц памяти | Тип данных для заполнения памяти | Тип сортировки | Объект синхронизации |
| 12 | 4 | int | Быстрая | Бинарный семафор |

1. Текст программ

Программа MemShare:

#include <Windows.h>

#include <iostream>

#include <conio.h>

#include <ctime>

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

const int PAGE\_SIZE**=**4096**;**

const int N\_PAGES**=**4**;**

int main**()** **{**

HANDLE hSem**;**

HANDLE hMapFile**;**

SECURITY\_ATTRIBUTES sAttr**;**

STARTUPINFO stInfo**;**

PROCESS\_INFORMATION pInfo**;**

char strCmd**[]** **=** "MemSort.exe"**;**

int **\***pVirtAlloc**;**

int **\***pBuf**;**

int nEl**;**

int i**;**

sAttr**.**bInheritHandle **=** TRUE**;**

sAttr**.**lpSecurityDescriptor **=** **NULL;**

sAttr**.**nLength **=** **sizeof(**SECURITY\_ATTRIBUTES**);**

hSem **=** CreateSemaphore**(&**sAttr**,**0**,**1**,**"sem"**);**

**if(**hSem**)** **{**

printf**(**"[MemShare] Successfully created semaphore\n"**);**

**}** **else** **{**

fprintf**(**stderr**,** "[MemShare] Failed to create semaphore %d\n"**,**GetLastError**());**

\_getch**();**

exit**(-**1**);**

**}**

ZeroMemory**(&**stInfo**,sizeof(**STARTUPINFO**));**

**if(**CreateProcess**(NULL,**strCmd**,NULL,NULL,**FALSE**,**NORMAL\_PRIORITY\_CLASS**,NULL,NULL,&**stInfo**,&**pInfo**))** **{**

printf**(**"[MemShare] Successfully created process\n"**);**

**}** **else** **{**

fprintf**(**stderr**,**"[MemShare] Failed to create process %d\n"**,**GetLastError**());**

\_getch**();**

exit**(-**1**);**

**}**

pVirtAlloc **=** **(**int**\*)** VirtualAlloc**(NULL,**PAGE\_SIZE**\***N\_PAGES**,**MEM\_COMMIT**,**PAGE\_READWRITE**);**

**if(**pVirtAlloc**)** **{**

printf**(**"[MemShare] Successfully allocated memory\n"**);**

**}** **else** **{**

fprintf**(**stderr**,**"[MemShare] Failed to allocate memory %d\n"**,**GetLastError**());**

\_getch**();**

exit**(-**1**);**

**}**

nEl **=** **(**PAGE\_SIZE**\***N\_PAGES**)/sizeof(**int**);**

srand**(**time**(**0**));**

**for(**i**=**0**;** i**<**nEl**;** **++**i**)** **{**

pVirtAlloc**[**i**]** **=** rand**()** **%** 30000**;**

**}**

printf**(**"[MemShare] Successfully filled array\n"**);**

VirtualProtect**(**pVirtAlloc**,**PAGE\_SIZE**\***N\_PAGES**,**PAGE\_READONLY**,NULL);**

hMapFile **=** CreateFileMapping**(**INVALID\_HANDLE\_VALUE**,NULL,**PAGE\_READWRITE**,**0**,**PAGE\_SIZE**\***N\_PAGES**,**"file"**);**

**if(**hMapFile**)** **{**

printf**(**"[MemShare] Successfully created file mapping\n"**);**

**}** **else** **{**

fprintf**(**stderr**,**"[MemShare] Failed to create file mapping %d\n"**,**GetLastError**());**

\_getch**();**

exit**(-**1**);**

**}**

pBuf **=** **(**int**\*)** MapViewOfFile**(**hMapFile**,**FILE\_MAP\_ALL\_ACCESS**,**0**,**0**,**PAGE\_SIZE**\***N\_PAGES**);**

**if(**pBuf**)** **{**

printf**(**"[MemShare] Successfully mapped file to memory\n"**);**

**}** **else** **{**

fprintf**(**stderr**,**"[MemShare] Failed to map file to memory %d\n"**,**GetLastError**());**

\_getch**();**

exit**(-**1**);**

**}**

CopyMemory**(**pBuf**,**pVirtAlloc**,**PAGE\_SIZE**\***N\_PAGES**);**

ReleaseSemaphore**(**hSem**,** 1**,** **NULL);**

printf**(**"[MemShare] Released semaphore\n"**);**

Sleep**(**50**);**

WaitForSingleObject**(**hSem**,** INFINITE**);**

printf**(**"[MemShare] Got semaphore\n"**);**

VirtualProtect**(**pVirtAlloc**,**PAGE\_SIZE**\***N\_PAGES**,**PAGE\_READWRITE**,NULL);**

CopyMemory**(**pVirtAlloc**,**pBuf**,**PAGE\_SIZE**\***N\_PAGES**);**

printf**(**"[MemShare] Resulted array is:\n"**);**

**for(**i**=**0**;** i**<**nEl**;** **++**i**)** **{**

printf**(**"%5d "**,** pVirtAlloc**[**i**]);**

**}**

printf**(**"\n"**);**

UnmapViewOfFile**(**pVirtAlloc**);**

VirtualFree**(**pVirtAlloc**,**PAGE\_SIZE**\***N\_PAGES**,**MEM\_RELEASE**);**

CloseHandle**(**hSem**);**

CloseHandle**(**hMapFile**);**

CloseHandle**(**pInfo**.**hThread**);**

CloseHandle**(**pInfo**.**hProcess**);**

printf**(**"[MemShare] Exiting!\n"**);**

\_getch**();**

**return** 0**;**

**}**

Программа MemSort:

#include <Windows.h>

#include <iostream>

#include <conio.h>

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

const int PAGE\_SIZE**=**4096**;**

const int N\_PAGES**=**4**;**

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

long i **=** 0**,** j **=** N**-**1**;**

int temp**,** p **=** a**[**N**/**2**];**

**do** **{**

**while(**a**[**i**]** **<** p**)** **{**

i**++;**

**}**

**while(**a**[**j**]** **>** p**)** **{**

j**--;**

**}**

**if(**i **<=** j**)** **{**

temp **=** a**[**i**];**

a**[**i**]** **=** a**[**j**];**

a**[**j**]** **=** temp**;**

i**++;**

j**--;**

**}**

**}** **while(**i**<=**j**);**

**if(**j **>** 0**)** **{**

sort**(**a**,**j**);**

**}**

**if(**N **>** i**)** **{**

sort**(**a**+**i**,**N**-**i**);**

**}**

**}**

int main**()** **{**

HANDLE hMapFile**;**

HANDLE hSem**;**

int **\***pBuf**;**

int nEl**;**

hSem **=** OpenSemaphore**(**SEMAPHORE\_MODIFY\_STATE**,**FALSE**,**"sem"**);**

**if(**hSem**)** **{**

printf**(**"[MemSort] Opened semaphore\n"**);**

**}** **else** **{**

fprintf**(**stderr**,** "[MemSort] Error opening semaphore %d\n"**,** GetLastError**());**

\_getch**();**

exit**(-**1**);**

**}**

WaitForSingleObject**(**hSem**,** INFINITE**);**

hMapFile **=** OpenFileMapping**(**FILE\_MAP\_ALL\_ACCESS**,**FALSE**,**"file"**);**

**if(**hMapFile**)** **{**

printf**(**"[MemSort] Opened File mapping\n"**);**

**}** **else** **{**

fprintf**(**stderr**,** "[MemSort] Error opening file mapping %d\n"**,** GetLastError**());**

\_getch**();**

exit**(-**1**);**

**}**

pBuf **=** **(**int**\*)**MapViewOfFile**(**hMapFile**,** FILE\_MAP\_ALL\_ACCESS**,** 0**,** 0**,** PAGE\_SIZE**\***N\_PAGES**);**

**if(**pBuf**)** **{**

printf**(**"[MemSort] Mapped file\n"**);**

**}** **else** **{**

fprintf**(**stderr**,** "[MemSort] Error mapping file %d\n"**,** GetLastError**());**

\_getch**();**

exit**(-**1**);**

**}**

nEl **=** **(**PAGE\_SIZE**\***N\_PAGES**)/sizeof(**int**);**

sort**(**pBuf**,**nEl**);**

printf**(**"[MemSort] Sorted the array\n"**);**

UnmapViewOfFile**((**PVOID**)**pBuf**);**

CloseHandle**(**hMapFile**);**

ReleaseSemaphore**(**hSem**,**1**,NULL);**

printf**(**"[MemSort] Released semaphore\n"**);**

CloseHandle**(**hSem**);**

**return** 0**;**

**}**

1. Результат

Начало работы программы показано на рисунке 1.

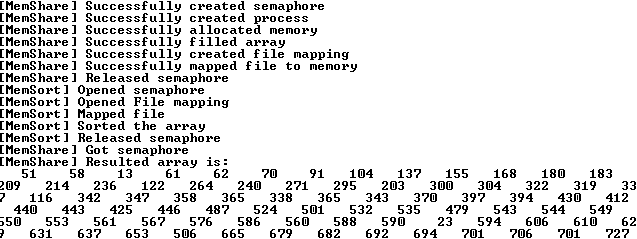


Рисунок 1 – запуск программы

Окончание работы программы показано на рисунке 2.

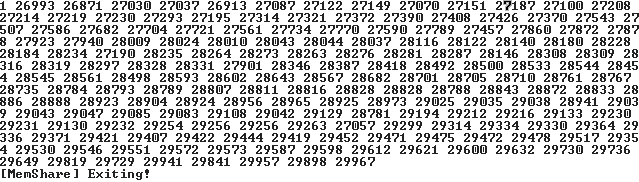


Рисунок 2 – конец программы

Выводы

В ходе выполнения лабораторной работы изучил возможности программного интерфейса приложений (API) операционных систем Windows по управления памятью и обмена данными между процессами

Приобрел практические навыки использования Win API для управления памятью и обмена данными между процессами.