# Лабораторная работа №3

Использование программного интерфейса Win API. Синхронизация процессов и потоков в ОС Windows

## Цель работы

Изучение программного интерфейса приложений (API) операционных систем Windows 9x, NT, ME, XP. Приобретение практических навыков синхронизации процессов и потоков, используя WinAPI в средах программирования Borland Delphi, C++ Builder или Visual Studio.

## Постановка задачи

Получить навыки работы с Мьютексами и Критическими секциями, используя WinAPI.

## Ход выполнения работы

## Тексты программ

В соответствии с выбранным вариантом (Вариант 19), программа была написана в среде программирования Visual Studio.

## Программы работы с мьютексами

##### Текст исходного файла Sort3.old.c

#include <stdio.h>

#include <stdlib.h>

#include <windows.h>

#include <time.h>

#define N 65

// Функция заполнения массива массива случайными числами

void fill\_array (int \*arr, size\_t size)

{

srand(time(NULL));

for (size\_t i = 0; i < size; i++)

arr[i] = rand();

}

// Сортировка массива

void sort\_quick(int \*arr, int f, int l)

{

if (f < l)

{

int left = f, right = l, middle = arr[(left + right) / 2];

do

{

while (arr[left] < middle) left++;

while (arr[right] > middle) right--;

if (left <= right)

{

int tmp = arr[left];

arr[left] = arr[right];

arr[right] = tmp;

left++;

right--;

}

} while (left <= right);

sort\_quick(arr, f, right);

sort\_quick(arr, left, l);

}

}

// Печать массива

void print\_arr (int \*arr, size\_t size, char \*string)

{

for (size\_t i = 0; i < size; i += 2)

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

}

int main (int argc, char \*\*argv)

{

if (argc != 2) return 1;

int arr[N];

// Заполнить и Сортировать массив

fill\_array(arr, N);

sort\_quick(arr, 0, N - 1);

// Печать массива

print\_arr(arr, N, argv[1]);

return 0;

}

##### Текст исходного файла Master3.old.c

#include <stdio.h>

#include <stdlib.h>

#include <windows.h>

#define PROC\_COUNT 5

int main (void)

{

PROCESS\_INFORMATION p[PROC\_COUNT];

STARTUPINFO s[PROC\_COUNT];

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

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

char tmp[1024];

for (size\_t i = 0; i < PROC\_COUNT; i++)

{

s[i].cb = sizeof(STARTUPINFO);

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

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

};

for (size\_t i = 0; i < PROC\_COUNT; i++)

{

sprintf(tmp, ".\\Sort3.old.exe Proc%02d", i);

CreateProcess(NULL, tmp, NULL, NULL, FALSE, CREATE\_SUSPENDED, NULL, NULL, &s[i], &p[i]);

}

for (size\_t i = 0; i < PROC\_COUNT; i++)

ResumeThread(p[i].hThread);

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

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

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

WaitForSingleObject(p[3].hProcess, INFINITE);

WaitForSingleObject(p[4].hProcess, INFINITE);

}

##### Текст исходного файла Sort3.mtx.c

#include <stdio.h>

#include <stdlib.h>

#include <windows.h>

#include <time.h>

#define N 65

// Функция заполнения массива массива случайными числами

void fill\_array (int \*arr, size\_t size)

{

srand(time(NULL));

for (size\_t i = 0; i < size; i++)

arr[i] = rand() % 1000;

}

// Сортировка массива

void sort\_quick(int \*arr, int f, int l)

{

if (f < l)

{

int left = f, right = l, middle = arr[(left + right) / 2];

do

{

while (arr[left] < middle) left++;

while (arr[right] > middle) right--;

if (left <= right)

{

int tmp = arr[left];

arr[left] = arr[right];

arr[right] = tmp;

left++;

right--;

}

} while (left <= right);

sort\_quick(arr, f, right);

sort\_quick(arr, left, l);

}

}

// Печать массива

void print\_arr (int \*arr, size\_t size, char \*string)

{

for (size\_t i = 0; i < size; i += 2)

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

}

int main (int argc, char \*\*argv)

{

if (argc != 2) return 1;

int arr[N];

HANDLE mtx = OpenMutex(MUTEX\_ALL\_ACCESS, FALSE, "MyMutexTypeSort");

// Заполнить и Сортировать массив

fill\_array(arr, N);

sort\_quick(arr, 0, N - 1);

// Ожидать свободный мютекс

WaitForSingleObject(mtx, INFINITE);

// Печать массива разрешена

print\_arr(arr, N, argv[1]);

// Освободить мютекс и завершить программу

ReleaseMutex(mtx);

return 0;

}

##### Текст исходного файла Master3.mtx.c

#include <stdio.h>

#include <stdlib.h>

#include <windows.h>

#define PROC\_COUNT 5

int main (void)

{

char tmp[1024];

PROCESS\_INFORMATION p[PROC\_COUNT];

STARTUPINFO s[PROC\_COUNT];

HANDLE mtx = CreateMutex(NULL, TRUE, "MyMutexTypeSort");

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

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

for (size\_t i = 0; i < PROC\_COUNT; i++)

{

s[i].cb = sizeof(STARTUPINFO);

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

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

};

for (size\_t i = 0; i < PROC\_COUNT; i++)

{

sprintf(tmp, ".\\Sort3.mtx.exe Proc%02d", i);

CreateProcess(NULL, tmp, NULL, NULL, FALSE, CREATE\_SUSPENDED, NULL, NULL, &s[i], &p[i]);

}

for (size\_t i = 0; i < PROC\_COUNT; i++)

ResumeThread(p[i].hThread);

ReleaseMutex(mtx);

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

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

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

WaitForSingleObject(p[3].hProcess, INFINITE);

WaitForSingleObject(p[4].hProcess, INFINITE);

CloseHandle(mtx);

}

## Программы работы с критическими секциями

##### Текст исходного файла Master3.crit.old.c

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <windows.h>

#define N 65

#define THREAD\_COUNT 5

void fill\_array (int \*, size\_t);

void sort\_quick(int \*, int, int);

void print\_arr (int \*, size\_t, char \*);

DWORD WINAPI Sort (LPVOID);

int main (void);

int main (void)

{

char tmp[THREAD\_COUNT][1024];

DWORD th[THREAD\_COUNT];

HANDLE hth[THREAD\_COUNT];

for (size\_t i = 0; i < THREAD\_COUNT; i++)

{

sprintf(tmp[i], "Thr%02d", i + 1);

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

}

for (size\_t i = 0; i < THREAD\_COUNT; i++) ResumeThread(hth[i]);

WaitForMultipleObjects(THREAD\_COUNT, hth, TRUE, INFINITE);

return 0;

}

DWORD WINAPI Sort (LPVOID p)

{

int string[N];

fill\_array(string, N);

sort\_quick(string, 0, N - 1);

print\_arr(string, N, (char \*) p);

return 0;

}

void fill\_array (int \*arr, size\_t size)

{

srand(time(NULL));

for (size\_t i = 0; i < size; i++)

arr[i] = rand() % 1000;

}

void sort\_quick(int \*arr, int f, int l)

{

if (f < l)

{

int left = f, right = l, middle = arr[(left + right) / 2];

do

{

while (arr[left] < middle) left++;

while (arr[right] > middle) right--;

if (left <= right)

{

int tmp = arr[left];

arr[left] = arr[right];

arr[right] = tmp;

left++;

right--;

}

} while (left <= right);

sort\_quick(arr, f, right);

sort\_quick(arr, left, l);

}

}

void print\_arr (int \*arr, size\_t size, char \*string)

{

for (size\_t i = 0; i < size; i += 2)

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

}

##### Текст исходного файла Master3.crit.new.c

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <windows.h>

#define N 66

#define THREAD\_COUNT 5

CRITICAL\_SECTION c;

void fill\_array (int \*, size\_t);

void sort\_quick(int \*, int, int);

void print\_arr (int \*, size\_t, char \*);

DWORD WINAPI Sort (LPVOID);

int main (void);

int main (void)

{

char tmp[THREAD\_COUNT][1024];

DWORD th[THREAD\_COUNT];

HANDLE hth[THREAD\_COUNT];

// Создание критической секции

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

InitializeCriticalSection(&c);

for (size\_t i = 0; i < THREAD\_COUNT; i++)

{

sprintf(tmp[i], "Thr%02d", i + 1);

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

}

for (size\_t i = 0; i < THREAD\_COUNT; i++) ResumeThread(hth[i]);

WaitForMultipleObjects(THREAD\_COUNT, hth, TRUE, INFINITE);

DeleteCriticalSection(&c);

return 0;

}

DWORD WINAPI Sort (LPVOID p)

{

int string[N];

fill\_array(string, N);

sort\_quick(string, 0, N - 1);

EnterCriticalSection(&c);

print\_arr(string, N, (char \*) p);

LeaveCriticalSection(&c);

return 0;

}

void fill\_array (int \*arr, size\_t size)

{

srand(time(NULL));

for (size\_t i = 0; i < size; i++)

arr[i] = rand() % 1000;

}

void sort\_quick(int \*arr, int f, int l)

{

if (f < l)

{

int left = f, right = l, middle = arr[(left + right) / 2];

do

{

while (arr[left] < middle) left++;

while (arr[right] > middle) right--;

if (left <= right)

{

int tmp = arr[left];

arr[left] = arr[right];

arr[right] = tmp;

left++;

right--;

}

} while (left <= right);

sort\_quick(arr, f, right);

sort\_quick(arr, left, l);

}

}

void print\_arr (int \*arr, size\_t size, char \*string)

{

for (size\_t i = 0; i < size; i++)

printf("%s: A[%2u] = %3d. ", string, i, arr[i]);

}

### Тестовые примеры

Были написаны основная программа Master3 и дополнительная программа Sort3, запускаемая основной программой как дополнительный процесс. После запуска было обнаружено, что выводы 5 процессов «перемешиваются» друг с другом (Рисунок 3.1). После чего программы были переписаны с использованием мьютексов. После запуска Master3 на экран были выведены данные, не «перемешанные» между собой, что продемонстрировано на Рисунке 3.2.

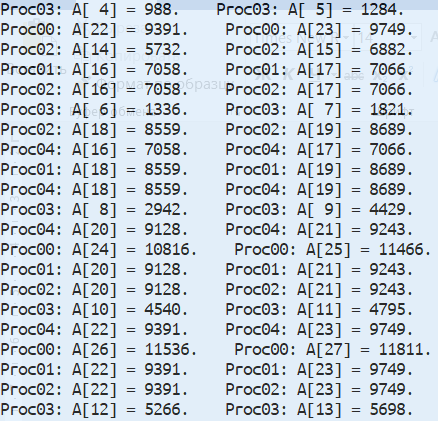


Рисунок 3.1 – Вывод программы Master3 (без мьютексов)

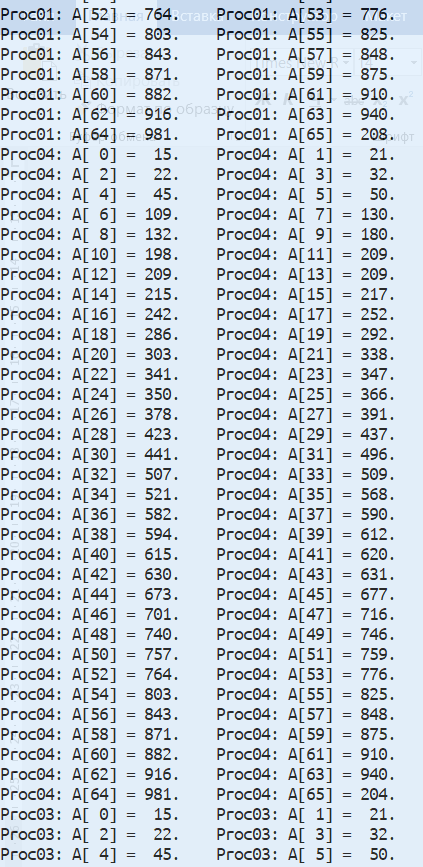


Рисунок 3.2 – Вывод программы Master3 (с использованием мьютексов)

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

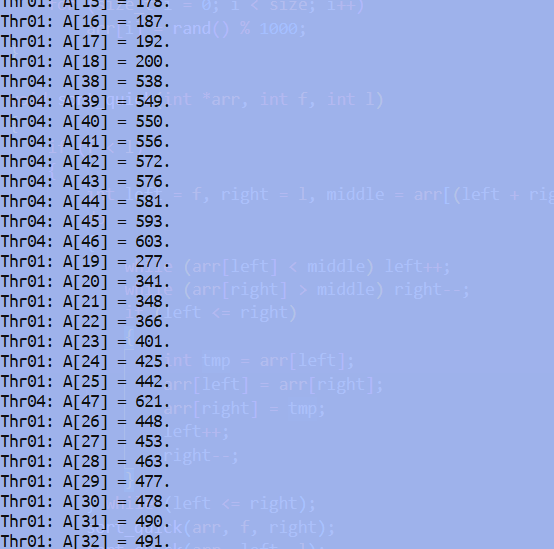


Рисунок 3.3 – Вывод программы, скомпилированной из файла Master.crit.old.c

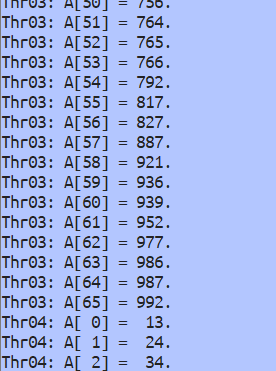


Рисунок 3.4 – Вывод программы, скомпилированной из файла Master.crit.new.c

## Вывод

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.