# ЛАБОРАТОРНАЯ РАБОТА №3

# ИСПОЛЬЗОВАНИЕ ПРОГРАММНОГО ИНТЕРФЕЙСА WIN API. СИНХРОНИЗАЦИЯ ПРОЦЕССОВ И ПОТОКОВ В ОС WINDOWS

Цель работы

Изучение программного интерфейса приложений (API) операционных систем семейства Win. Приобретение практических навыков синхронизации процессов и потоков при использовании Win API в различных средах программирования.

Задачи

1. Написать программу Sort3, реализующую следующий алгоритм:
   1. Получить в качестве параметра командной строки номера процесса N\_PRC;
   2. Заполнить массив N целых чисел случайными значениями из диапазона 0-100 и отсортировать его;
   3. Вывести на экран отсортированный массив в формате N\_PRC: A[i];

Язык программирования, метод и направление сортировки, а также количество элементов массива N выбрать в соответствии с вариантом задания (Таблица 1).

1. Написать программу Master3, создающую процессы Sort3, передавая в качестве параметра командной строки номер процесса. Необходимо, чтобы все процессы использовали одну консоль и имели класс приоритета NORMAL\_PRIORITY\_CLASS. Количество процессов выбирается исходя из варианта задания (Таблица 1);
2. Переписать программы Sort3 и Master3, с использованием механизма взаимоисключений. Полученная программа Master3\_mtx должна создавать мьютекс перед созданием процессов, а программа Sort3\_mtx должна получать идентификатор мьютекса, занимать мьютекс перед выводом отсортированного массива на экран, а после вывода массива на экран освобождать мьютекс;
3. Разработать программу Master3\_Crit для исследования механизма критических секций. Программа должна содержать функцию Sort, запускаемую несколько раз в виде отдельных потоков. Функция Sort должна: генерировать случайный массив из N элементов, сортировать его и выводить его на экран;

Вариант задания

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № варианта | Среда программирования | Метод сортировки | Направление сортировки | Количество процессов | N |
| 11 | Lazarus | Выбор | Убыв. | 5 | 75 |

Таблица 1 – Вариант задания

Ход работы

Программа Master3 была вызвана при отсутствии файла Sort3.exe. В результате в консоль были выведены сообщения об ошибке (Рисунок 1).

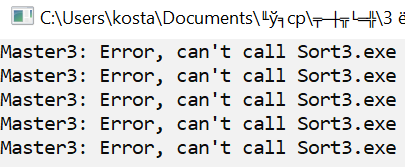


Рисунок 1 – Сообщение об ошибке (Master3)

Затем была создана программа Sort3. При запуске Master3 было вызвано 5 процессов Sort3.exe с различным значением переменной N\_PRC. Однако результаты сортировок были выведены в консоль хаотично (Рисунок 2).

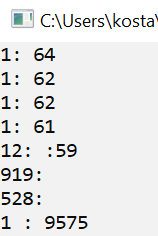


Рисунок 2 – Результат вызова потоков в Master3

Это обусловлено тем, что все процессы работают с одной консолью и имеют одинаковый приоритет. По этой причине они прерывают друг друга, из-за чего данные перемешиваются при выводе на экран.

Программы Master3 и Sort3 были переписаны с использованием мьютексов. В результате в консоль по очереди без перемешивания были выведены результаты всех пяти запущенных процессов Sort3\_mtx (Рисунок 3).

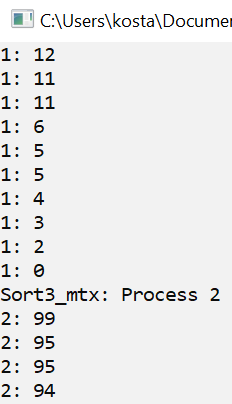


Рисунок 3 – Результат вызова Master3\_mtx

Программа Master3\_Crit сначала была написана без использования механизма критических секций. В результате данные, выводимые в консоль, перемешались (Рисунок 4).

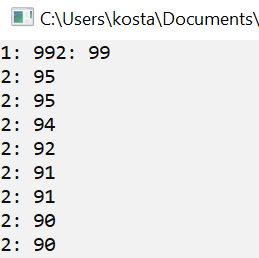


Рисунок 4 – Вызов Master3\_Crit без критических секций

После того, как программа Master3\_Crit была переписана так, чтобы вывод массивов находился в критической секции, проблема с перемешиванием при выводе исчезла (Рисунок 5).

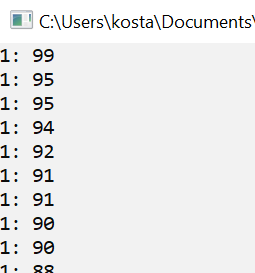


Рисунок 5 – Вызов Master3\_Crit

Текст программы

Программа Sort3:

#include <windows.h>

#include <iostream>

#include <cstdlib>

#include <ctime>

using namespace std;

const int N = 75;

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

{

// Проверка наличия параметра N\_PRC

if (argc != 2)

{

cout << "Sort3: Program must be called by Master3.exe" << endl;

return 0;

}

// N\_PRC - номер процесса, переданный через Master.exe

int N\_PRC = atoi(argv[1]);

// Заполнение массива

int arr[N];

srand(time(NULL));

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

arr[i] = rand() % 101;

// Сортировка (прямой выбор)

int swap;

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

{

int max = i;

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

if (arr[j] > arr[max]) max = j;

swap = arr[i];

arr[i] = arr[max];

arr[max] = swap;

}

// Вывод массива

cout << "Sort3: Process " << N\_PRC << endl;

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

cout << N\_PRC << ": " << arr[i] << endl;

return 1;

}

Программа Master3:

#include <windows.h>

#include <iostream>

#include <cstring>

using namespace std;

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

{

// Информация о запуске нового процесса

STARTUPINFO si;

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

si.cb = sizeof(si);

// Идентификатор и дескриптор нового процесса

PROCESS\_INFORMATION pi;

// Запуск процессов

for (int N\_PRC = 1; N\_PRC <= 5; N\_PRC++)

{

char cmd[300] = "Sort3.exe";

sprintf(cmd + strlen(cmd), " %d", N\_PRC);

if (!CreateProcess(NULL, cmd, NULL, NULL, FALSE, NORMAL\_PRIORITY\_CLASS, NULL, NULL, &si, &pi))

cout << "Master3: Error, can't call Sort3.exe" << endl;

}

// Закрытие процессов

WaitForSingleObject(pi.hProcess, INFINITE);

}

Программа Sort3\_mtx:

#include <windows.h>

#include <iostream>

#include <cstdlib>

#include <ctime>

using namespace std;

const int N = 75; // длина массива

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

{

// Проверка наличия параметра N\_PRC

if (argc != 2)

{

cout << "Sort3\_mtx: Program must be called by Master3\_mtx.exe" << endl;

return 0;

}

// Открытие объекта мьютекса (или ошибка)

HANDLE mtx = OpenMutex(SYNCHRONIZE, FALSE, "MyMutex");

if (!mtx) return 0;

// N\_PRC - номер процесса, переданный через Master.exe

int N\_PRC = atoi(argv[1]);

// Заполнение массива

int arr[N];

srand(time(NULL));

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

arr[i] = rand() % 101;

// Сортировка (прямой выбор)

int swap;

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

{

int max = i;

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

if (arr[j] > arr[max]) max = j;

swap = arr[i];

arr[i] = arr[max];

arr[max] = swap;

}

// Захват объекта мьютекса

WaitForSingleObject(mtx, INFINITE);

// Вывод массива

cout << "Sort3\_mtx: Process " << N\_PRC << endl;

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

cout << N\_PRC << ": " << arr[i] << endl;

// Освобождение мьютекса

ReleaseMutex(mtx);

return 1;

}

Программа Master3\_mtx:

#include <windows.h>

#include <iostream>

#include <cstring>

using namespace std;

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

{

HANDLE mtx = CreateMutex(NULL, FALSE, "MyMutex"); // мьютекс

// Информация о запуске нового процесса

STARTUPINFO si;

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

si.cb = sizeof(si);

// Идентификатор и дескриптор нового процесса

PROCESS\_INFORMATION pi;

// Запуск процессов

for (int N\_PRC = 1; N\_PRC <= 5; N\_PRC++)

{

char cmd[300] = "Sort3\_mtx.exe";

sprintf(cmd + strlen(cmd), " %d", N\_PRC);

if (!CreateProcess(NULL, cmd, NULL, NULL, FALSE, NORMAL\_PRIORITY\_CLASS, NULL, NULL, &si, &pi))

cout << "Master3\_mtx: Error, can't call Sort3\_mtx.exe" << endl;

//WaitForSingleObject(pi.hProcess, INFINITE);

}

// Ожидание конца процессов и закрытие мьютекса

WaitForSingleObject(pi.hProcess, INFINITE);

CloseHandle(mtx);

}

Программа Master3\_Crit:

#include <iostream>

#include <windows.h>

#include <cstdlib>

#include <ctime>

using namespace std;

const int N = 75; // длинна массива

CRITICAL\_SECTION crit; // критическая секция

// Заполнение массива

void setArray(int arr[N])

{

srand(time(NULL));

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

arr[i] = rand() % 101;

}

// Сортировка массива

void sortArray(int arr[N])

{

int swap;

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

{

int max = i;

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

if (arr[j] > arr[max]) max = j;

swap = arr[i];

arr[i] = arr[max];

arr[max] = swap;

}

}

// Вывод массива

void showArray(int arr[N], int N\_THR)

{

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

cout << N\_THR << ": " << arr[i] << endl;

}

// Функция потока (ввод, сортировка, вывод)

LPTHREAD\_START\_ROUTINE Sort(int N\_THR)

{

int arr[N];

setArray(arr); // ввод

sortArray(arr); // сортировка

// Обращение к критической секции

EnterCriticalSection(&crit);

cout << "Thread " << N\_THR << endl;

showArray(arr, N\_THR); // вывод

// Освобождение критической секции

LeaveCriticalSection(&crit);

return 0;

}

// Главная функция

int main()

{

DWORD id[5]; // идентификаторы

HANDLE th[5]; // дескрипторы

// Инициализация критической секции

InitializeCriticalSection(&crit);

// Цикл запусков потоков Sort

for (int i = 0; i < 5; i++)

th[i] = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)Sort, (LPVOID)(i + 1), 0, &id[i]);

// Ожидание окончания потоков и удаление критической секции

WaitForMultipleObjects(5, th, TRUE, INFINITE);

DeleteCriticalSection(&crit);

return 0;

}

Вывод

В ходе работы были получены навыки синхронизации процессов и потоков с помощью инструментов программного интерфейса ОС Windows Win API. Был разработан ряд многопоточных программ с использованием мьютексов и критических секций. Благодаря этим средствам в написанных программах удалось организовать параллельную работу процессов и потоков так, чтобы при их работе с одной областью памяти они не перекрывали друг друга и не возникало ошибок.