**БУ ВО Ханты-Мансийского автономного округа – Югры**

**«Сургутский государственный университет»**

**Политехнический институт**

**Кафедра автоматики и компьютерных систем**

**ОТЧЁТ**

**по лабораторной работе №3**

**по дисциплине: «Операционные системы»**

**Выполнил: студент группы №609-21**

**Шумилов И. Д.**

**Принял: старший преподаватель кафедры АиКС**

**Кривицкая М. А.**

**Сургут**

**2024г.**

Цель работы: изучить функции и структуры данных Windows API, используемые для управления потоками, исследовать базовые механизмы взаимодействия потоков, ознакомиться с принципами управления приоритетами потоков.

**Задание на лабораторную работу:**

1. Разработать и отладить прогрпмму, выполняющую следующие функции:
   1. Заполнение некоторого массива случайными числами
   2. Создание в основном потоке двух дополнительных потоков, выполняющих сортировку массива двумя различными методами и сохраняющих результаты в отдельный массив.
   3. Основной поток ожидает завершения сортировки, сравнивает результаты сортировки, получает значения временных характеристик дополнительных потоков, основного потока и всего процесса, результаты выводит на экран
2. Ипользуя реализацию согласно п. 1, создать программу с учетом следующих требований:
   1. После создания дополнительных потоков изменить их базовые приоритеты в максимально большую сторону для одного и в меньшую для другого
   2. Сравнить полученные временные характеристики выполнения потоков с результатами, полученными согласно п. 1.3
   3. Выполнить п 2.1 и 2.2 поменяв более приоритетный и менее приоритетный потоки местами
3. Дополнить программу, разработанную согласно п. 2, блокированием динамических изменений приоритетов потоков после их запуска и сравнить полученные временные характеристики выполнения потоков с полученными раннее.
4. По результатам работы заполнить таблицу и построить временные диаграммы, схематично иллюстрирующие псевдопараллельное выполнение потоков при различных уровнях приоритетов.

Листинг 1. Код программы на С++:  
#include <Windows.h>

#include <iostream>

struct ARGS {

int\* arr;

int size;

};

enum Priority {

EQUAL, FIRST, SECOND

};

void GenArr(int\*, int);

void CreateAndMeasureThreads(Priority, const int);

DWORD WINAPI SortA(LPVOID);

DWORD WINAPI SortB(LPVOID);

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

const int size = std::pow(10, 5);

CreateAndMeasureThreads(EQUAL, size);

CreateAndMeasureThreads(FIRST, size);

CreateAndMeasureThreads(SECOND, size);

return 1;

}

void GenArr(int\* arr, int size) {

srand((unsigned)time(NULL));

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

arr[i] = rand() % 101; // from 0 to 100

}

}

void CreateAndMeasureThreads(Priority prio, const int size) {

int aPrio, bPrio;

switch (prio) {

case EQUAL:

aPrio = THREAD\_PRIORITY\_NORMAL;

bPrio = THREAD\_PRIORITY\_NORMAL;

break;

case FIRST:

aPrio = THREAD\_PRIORITY\_HIGHEST;

bPrio = THREAD\_PRIORITY\_LOWEST;

break;

case SECOND:

aPrio = THREAD\_PRIORITY\_LOWEST;

bPrio = THREAD\_PRIORITY\_HIGHEST;

break;

}

int\* arrA = (int\*)malloc(sizeof(int) \* size), \* arrB = (int\*)malloc(sizeof(int) \* size);

ARGS argsA = { arrA, size };

ARGS argsB = { arrB, size };

GenArr(arrA, size);

GenArr(arrB, size);

DWORD threadAId, threadBId;

FILETIME creationTimeA, exitTimeA, kernelTimeA, userTimeA,

creationTimeB, exitTimeB, kernelTimeB, userTimeB;

SetProcessPriorityBoost(GetCurrentProcess(), FALSE);

HANDLE hThreadA = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)SortA, &argsA, 0, &threadAId);

SetThreadPriority(hThreadA, aPrio);

HANDLE hThreadB = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)SortB, &argsB, 0, &threadBId);

SetThreadPriority(hThreadB, bPrio);

WaitForSingleObject(hThreadA, INFINITE);

WaitForSingleObject(hThreadB, INFINITE);

GetThreadTimes(hThreadA, &creationTimeA, &exitTimeA, &kernelTimeA, &userTimeA);

GetThreadTimes(hThreadB, &creationTimeB, &exitTimeB, &kernelTimeB, &userTimeB);

std::cout << "A." << std::endl;

std::cout << "Creation time: " << creationTimeA.dwLowDateTime / 100000.0 << "." << std::endl;

std::cout << "Exit time: " << exitTimeA.dwLowDateTime / 100000.0 << "." << std::endl;

std::cout << "Kernel time: " << kernelTimeA.dwLowDateTime / 100000.0 << "." << std::endl;

std::cout << "User time: " << userTimeA.dwLowDateTime / 100000.0 << "." << std::endl;

std::cout << "\nWork time: " << (exitTimeA.dwLowDateTime - creationTimeA.dwLowDateTime) / 100000.0 << ".\n\n\n";

std::cout << "B." << std::endl;

std::cout << "Creation time: " << creationTimeB.dwLowDateTime / 100000.0 << "." << std::endl;

std::cout << "Exit time: " << exitTimeB.dwLowDateTime / 100000.0 << "." << std::endl;

std::cout << "Kernel time: " << kernelTimeB.dwLowDateTime / 100000.0 << "." << std::endl;

std::cout << "User time: " << userTimeB.dwLowDateTime / 100000.0 << "." << std::endl;

std::cout << "\nWork time: " << (exitTimeB.dwLowDateTime - creationTimeB.dwLowDateTime) / 100000.0 << ".\n\n\n\n";

}

// bubble sort

DWORD WINAPI SortA(LPVOID lpParam) {

ARGS args = \*((ARGS\*)lpParam);

int\* arr = args.arr;

int size = args.size;

int\* tmpA = (int\*)(malloc(sizeof(int) \* size));

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

tmpA[i] = arr[i];

}

int tmpV;

bool flag;

do {

flag = true;

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

if (tmpA[i] > tmpA[i + 1]) {

tmpV = tmpA[i];

tmpA[i] = tmpA[i + 1];

tmpA[i + 1] = tmpV;

flag = false;

}

}

} while (!flag);

return NULL;

}

// insertion sort

DWORD WINAPI SortB(LPVOID lpParam) {

ARGS args = \*((ARGS\*)lpParam);

int\* arr = args.arr;

int size = args.size;

int\* tmpA = (int\*)(malloc(sizeof(int) \* size));

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

tmpA[i] = arr[i];

}

int tmpV, j;

for (int i = 1; i < size; i++) {

tmpV = tmpA[i];

j = i - 1;

while (j >= 0 && tmpA[j] > tmpV) {

tmpA[j + 1] = tmpA[j];

j = j - 1;

}

tmpA[j + 1] = tmpV;

}

return NULL;

}

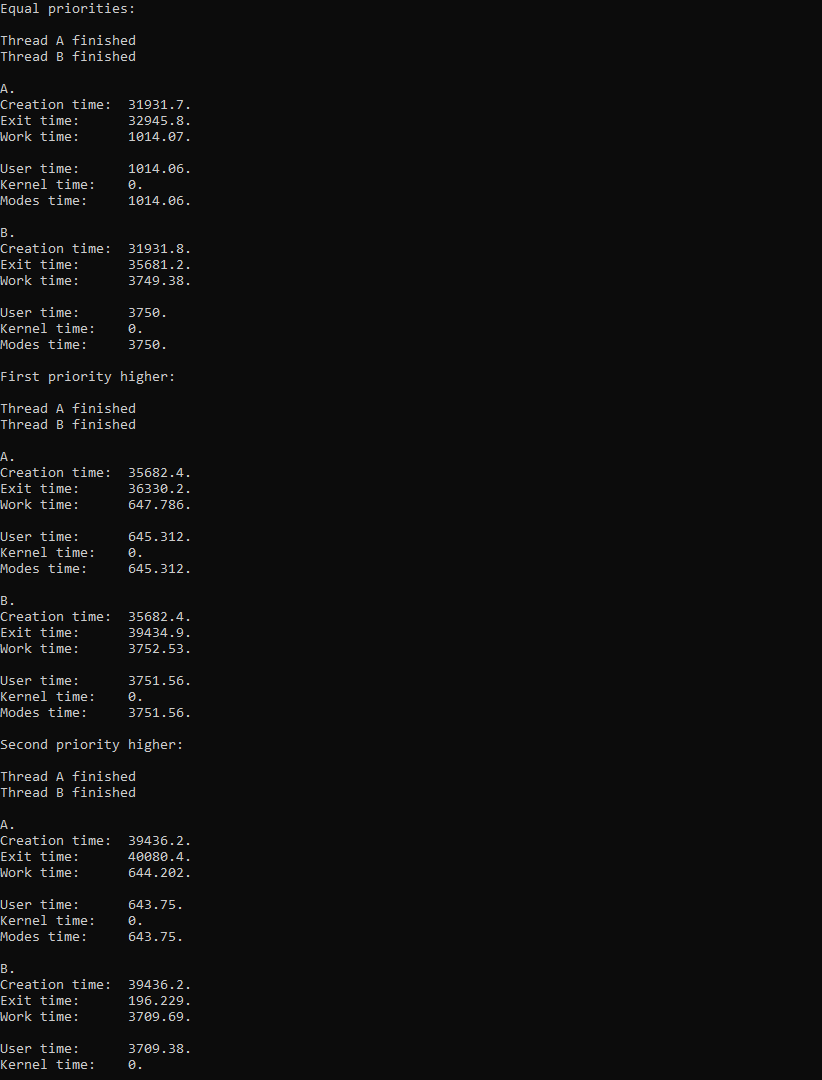


Рис. 1 - результат выполнения программы без блокирования динамических изменений приоритетов

Таблица 1. Время выполнений функций в потоках (без блокирования динамических изменений приоритетов)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | | Приоритеты равны | | Приоритет 1-го больше | | Приоритет 2-го больше | |
| 1-ый поток | 2-ой поток | 1-ый поток | 2-ой поток | 1-ый поток | 2-ой поток |
| Время | Создания | 31931 | 31931 | 35682 | 35682 | 39436 | 39436 |
| Завершения | 32945 | 35681 | 36330 | 39434 | 40080 | 196 |
| Существования | 1014 | 3749 | 647 | 3751 | 644 | 3709 |
| Время работы | В режиме пользователя | 1014 | 3750 | 645 | 3751 | 643 | 3709 |
| В режиме ядра | 0 | 0 | 0 | 0 | 0 | 0 |
| суммарное | 1014 | 3750 | 645 | 3751 | 643 | 3709 |

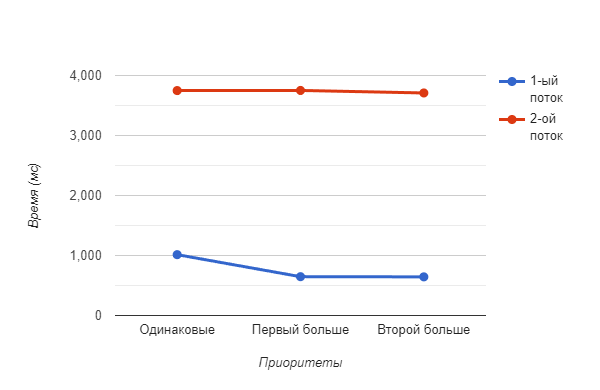


Рис. 2 - график временной зависимости потоков на разных приоритетах

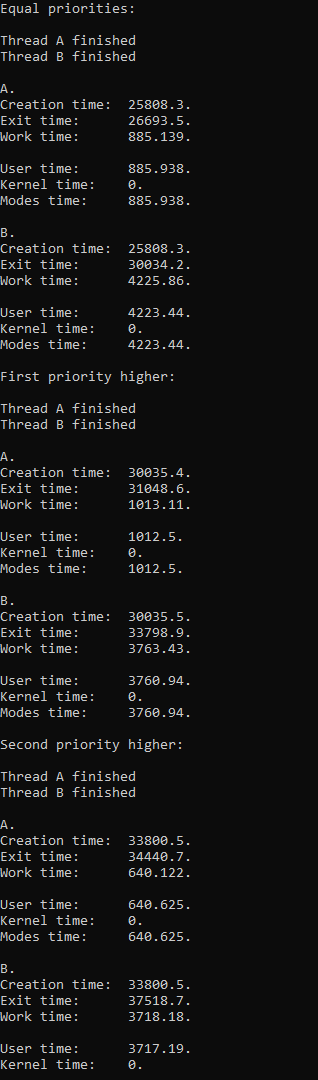


Рис. 3 - результат выполнения программы с блокированием динамических изменений приоритетов

Таблица 2. Время выполнений функций в потоках (c блокированием динамических изменений приоритетов)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | | Приоритеты равны | | Приоритет 1-го больше | | Приоритет 2-го больше | |
| 1-ый поток | 2-ой поток | 1-ый поток | 2-ой поток | 1-ый поток | 2-ой поток |
| Время | Создания | 25808 | 25808 | 30035 | 30035 | 33800 | 33800 |
| Завершения | 26693 | 30034 | 31048 | 33798 | 34440 | 37518 |
| Существования | 885 | 4225 | 1013 | 3763 | 640 | 3718 |
| Время работы | В режиме пользователя | 885 | 4223 | 1012 | 3760 | 640 | 3717 |
| В режиме ядра | 0 | 0 | 0 | 0 | 0 | 0 |
| суммарное | 885 | 4223 | 1012 | 3760 | 640 | 3717 |

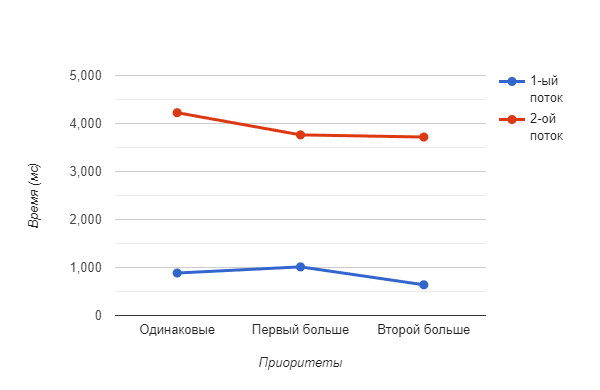


Рис. 4 - график временной зависимости потоков на разных приоритетах

**Вывод:** были изучены некоторые функции и структуры данных библиотеки Windows в С++, используемые для создания и управления потоками. Также были иследованы механизмы взаимодействия потоков и принципы управления приоритетами потоков.