Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт № 8 «Компьютерные науки и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

**Лабораторная работа №3 по курсу**

**«Операционные системы»**

Студент: Воробьева К.Н.

Группа: М8О-201Б-21

Вариант: №17

Преподаватель: Миронов Е.С.

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2022

**Содержание**

1. Цель работы
2. Постановка задачи
3. Общие сведения о программе
4. Исходный код
5. Демонстрация работы программы
6. Выводы
   * + 1. **Цель работы**

Целью является приобретение практических навыков в:

* Управление потоками в ОС
* Обеспечение синхронизации между потоками
  + - 1. **Постановка задачи**

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработки использовать стандартные средства создания потоков операционной системы (Windows/Unix). Ограничение потоков должно быть задано ключом запуска вашей программы. Так же необходимо уметь продемонстрировать количество потоков, используемое вашей программой с помощью стандартных средств операционной системы. Получившиеся результаты необходимо объяснить.

**Вариант №17:** Найти в большом целочисленном массиве минимальный элемент

Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

* + - 1. **Общие сведения о программе**

Программа представляет из себя файл lab3.c

В программе используются такие команды, как:

**pthread\_create**(pthread\_t \*tid, const pthread\_attr\_t \*attr, void\*(\*function)(void\*), void\* arg) – создание потока

**pthread\_join**(pthread\_t \*tid, const pthread\_attr\_t \*attr) – ожидание завершение потока

* + - 1. **Исходный код**

**lab3.c**

*#include "lab3.h"*

*#include "utils.h"*

*#include <pthread.h>*

*#include <iostream>*

*namespace*

*{*

*void MinVectorRows(const TVector &lhs, TVector &result, int firstRow, int lastRow, int iterator)*

*{*

*int min1;*

*min1 = lhs[firstRow];*

*for (int j = firstRow; j < lastRow; ++j)*

*{*

*if (min1 > lhs[j])*

*{*

*min1 = lhs[j];*

*}*

*}*

*result[iterator] = min1;*

*}*

*}*

*void \*MinVectorRowsRoutine(void \*arg)*

*{*

*auto \*token = (TThreadToken \*)arg;*

*MinVectorRows(\*token->lhs, \*token->result, token->firstRow, token->lastRow, token->iterator);*

*return nullptr;*

*}*

*int MinVector(const TVector &lhs, int threadCount)*

*{*

*int min;*

*int actualThreads = std::min(threadCount, isize(lhs));*

*TVector result(actualThreads);*

*if (threadCount > 1)*

*{*

*int iterator = 0;*

*int rowsPerThread = isize(lhs) / actualThreads;*

*std::vector<pthread\_t> threads(actualThreads);*

*std::vector<TThreadToken> tokens(actualThreads);*

*for (int i = 0; i < isize(lhs); i += rowsPerThread)*

*{*

*tokens[iterator].lhs = &lhs;*

*tokens[iterator].result = &result;*

*tokens[iterator].firstRow = i;*

*tokens[iterator].iterator = iterator;*

*if (i + rowsPerThread >= isize(result))*

*{*

*tokens[iterator].lastRow = isize(lhs);*

*pthread\_create(&threads[iterator], nullptr, &MinVectorRowsRoutine, &tokens[iterator]);*

*}*

*else*

*{*

*tokens[iterator].lastRow = (i + rowsPerThread - 1);*

*pthread\_create(&threads[iterator], nullptr, &MinVectorRowsRoutine, &tokens[iterator]);*

*}*

*++iterator;*

*}*

*for (int i = 0; i < actualThreads; i++)*

*{*

*pthread\_join(threads[i], nullptr);*

*}*

*}*

*else*

*{*

*MinVectorRows(lhs, result, 0, isize(lhs), 0);*

*}*

*min = result[0];*

*for (int j = 0; j < isize(result); ++j)*

*{*

*if (min > result[j])*

*{*

*min = result[j];*

*}*

*}*

*return min;*

*}*

**main.cpp**

*#include "lab3.h"*

*#include <iostream>*

*int main()*

*{*

*int m;*

*int threadCount;*

*std::cin >> m >> threadCount;*

*if (threadCount == 0)*

*{*

*threadCount = 1;*

*}*

*TVector lhs(m);*

*for (int i = 0; i < m; ++i)*

*{*

*std::cin >> lhs[i];*

*}*

*int min = MinVector(lhs, threadCount);*

*std::cout << "Minimum:" << min;*

*std::cout << '\n';*

*}*

**lab3.h**

*#ifndef OS\_LABS\_LAB3\_H*

*#define OS\_LABS\_LAB3\_H*

*#include <vector>*

*using TVector = std::vector<int>;*

*int MinVector(const TVector &lhs, int threadCount);*

*struct TThreadToken {*

*const TVector\* lhs;*

*TVector\* result;*

*int firstRow;*

*int lastRow;*

*int iterator;*

*};*

*#endif // OS\_LABS\_LAB3\_H*

**utils.h**

*#ifndef OS\_LABS\_UTILS\_H*

*#define OS\_LABS\_UTILS\_H*

*template <typename Container>*

*inline int isize(const Container &c)*

*{*

*return static\_cast<int>(c.size());*

*}*

*#endif // OS\_LABS\_UTILS\_H*

* + - 1. **Демонстрация работы программы**

**Тест:**

*karina@MSI:~/projects/OS/build/lab3$ ./lab3*

*1000000*

*4*

*Minimum:1*

*Number of threads:4*

*Time microseconds:3372*

*karina@MSI:~/projects/OS/build/lab3$ ./lab3*

*100000*

*4*

*Minimum:1*

*Number of threads:4*

*Time microseconds:524*

*karina@MSI:~/projects/OS/build/lab3$ ./lab3*

*100000*

*5*

*Minimum:1*

*Number of threads:5*

*Time microseconds:941*

*karina@MSI:~/projects/OS/build/lab3$*

* + - 1. **Выводы**

Проделав лабораторную работу, я приобрела практические навыки в управлении потоками в ОС и обеспечила синхронизацию между ними. Однако, оказывается, что создание потоков не так просто, как кажется на первый взгляд, ведь необходимо воплотить специальную функцию, которая будет выполняться в отдельном потоке исполнения, поэтому необходимо с особой осторожностью подходить к реализации данной функции из-за доступа к одной и той же памяти родительского процесса многими потоками. Затем создать специальную структуру для данных потока, и лишь потом возможно будет создать потоки. Также я узнала, что использование потоков может пригодиться в любой системе: в однопроцессорной, в которой достаточная часть времени уходит на ожидание ввода-вывода и в многопроцессорных, где задачи могут выполняться параллельно на разных процессорах, что даёт рост производительности программы. К сожалению, не любой алгоритм хорошо выполняется как многопоточная программа, однако для некоторых из них есть параллельные реализации, которые ускоряют работу программы.