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: Чирикова П. С.
Группа: М8О-201Б-21
Вариант: №17
Преподаватель: Миронов Е.С.
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2022

Содержание

1. **Цель работы**

Целью является приобретение практических навыков в:

- Управление потоками в ОС
- Обеспечение синхронизации между потоками

2. 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.

3. 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

4. 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
```

5. **Демонстрация работы программы**

Тест:

```
polina@polina-Vostro-3400:~/projects/OS/build/lab3$ ./lab3
1000000
4
Minimum:1
Number of threads:4
Time microseconds:3372
polina@polina-Vostro-3400:~/projects/OS/build/lab3$ ./lab3
100000
4
Minimum:1
Number of threads:4
Time microseconds:524
polina@polina-Vostro-3400:~/projects/OS/build/lab3$ ./lab3
100000
5
Minimum:1
Number of threads:5
Time microseconds:941
polina@polina-Vostro-3400:~/projects/OS/build/lab3$
```

6. **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.