Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №3 по курсу**

**«Операционные системы»**

**Управление потоками в ОС. Обеспечение синхронизации между потоками.**

Студент: Моисеенков Илья Павлович

Группа: М80 – 208Б-19

Вариант: 19

Преподаватель: Миронов Евгений Сергеевич

Дата: 19.10.2020

Оценка: отлично

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2020

1. **Постановка задачи**

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработке использовать стандартные средства создания потоков операционной системы (Windows/Unix). Ограничение потоков должно быть задано ключом запуска программы.

Необходимо уметь продемонстрировать количество потоков, используемых программой, с помощью стандартных средств операционной системы.

Привести исследование зависимости ускорения и эффективности алгоритма от входящих данных и количества потоков. Объяснить получившиеся результаты.

Необходимо реализовать проверку числа на простоту при помощи алгоритма «решето Эратосфена».

1. **Общие сведения о программе**

Программа написана на языке Си в UNIX-подобной операционной системе (Ubuntu). Для компиляции программы требуется указать ключ –pthread. Для запуска программы в качестве аргумента командной строки необходимо указать количество потоков, которые могут быть использованы программой.

Программа содержит две глобальные переменные – массив для решета Эратосфена и число, простоту которого проверяем. Переменные объявлены глобальными, чтобы любой поток имел к ним доступ.

Программа включает в себя потоковую функцию void\* sieve\_step(void\* i), в которой помечаются все числа решета, кратные i. Так как все потоки программы работают в одном и том же пространстве памяти, аргументы для передачи потоковой функции хранятся по разным адресам (в массиве, размер которого равен количеству потоков).

В программе предусмотрена проверка на системные ошибки – ошибки выделения памяти, ошибки запуска.

1. **Общий метод и алгоритм решения**

При запуске программы у пользователя запрашивается число num, которое необходимо проверить на простоту. Проверить на простоту можно только неотрицательное число.

Из аргументов командной строки берётся количество потоков, которое может использовать программа. Производится выделение памяти для массива потоков, для массива аргументов потоковой функции и для самого решета. Решето представляет собой массив символов sieve (т.к. размер символьного типа char минимальный). sieve[i] равно нулю, если число простое и единице в противном случае.

По определению числа 0 и 1 не являются простыми, поэтому сразу помечаем их единицами в решете. Необходимо проверить все числа от 2 до num включительно. Если ячейка решета, соответствующая числу i, равна нулю, то это число простое и требуется «вычеркнуть» (пометить единицей) все числа, кратные i. Эта задача и делегируется другим потокам.

Потоковая функция sieve\_step принимает на вход число i и помечает единицами все числа, кратные i. Заметим, что первое число, кратное i и которое еще НЕ было помечено единицей – это чило i2. Для ускорения алгоритма начнём проверку именно с этого числа и будем помечать каждое i-ое число, начиная с i2. По этой же причине в главной функции перебор элементов решета будет вестись от 2 до корня из i. Потоки не смогут повлиять на работу друг друга, поэтому mutex не используется.

Укажем правило, по которому будет выбираться поток для выполнения функции. Заведём переменную cur\_thread, изначально равную нулю. Для выполнения функции будет создаваться поток с индексом cur\_thred(mod therads\_num), где threads\_num – общее количество потоков. Таким образом, потоки будут использоваться в порядке закольцованной очереди. Когда cur\_thread становится больше количества потоков, потоки начинают использоваться повторно. Во избежание ситуации, когда задача будет делегирована потоку, работа которого еще не окончена, будем дожидаться окончания работы потока. После делегирования задач, переменная cur\_thread инкрементируется.

После обработки всего решета необходимо дождаться окончания работы всех активных потоков. После этого необходимо посмотреть число в sieve[num] и сделать вывод о простоте этого числа.

1. **Основные файлы программы**

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

char\* sieve;

long long num;

*// thread function - marking numbers that are multiple of i*

void\* sieve\_step(void\* i\_void) {

    long long i = \*(long long\*)i\_void;

    for (long long j = i \* i; j <= num; j += i) {

        sieve[j] = 1;

    }

    pthread\_exit(NULL);

}

int main(int argc, char\* argv[]) {

    if (argc != 2) {

        printf("Syntax: ./main Number\_of\_threads\n");

        exit(1);

    }

    int threads\_num = atoi(argv[1]);

    pthread\_t\* threads = (pthread\_t\*)calloc(threads\_num, sizeof(pthread\_t));

    if (threads == NULL) {

        printf("Can't allocate space for threads\n");

        exit(2);

    }

*// array for arguments which will be passed into thread function*

    long long\* args = (long long\*)malloc(threads\_num \* sizeof(long long));

    if (args == NULL) {

        printf("Can't create an array for arguments for threads\n");

        exit(3);

    }

    printf("Enter a number you want to check: ");

    scanf("%lld", &num);

*// creating array filled with 0 for sieve*

*// 0 - prime number, 1 - nonprime number*

    sieve = (char\*)calloc((num + 1), sizeof(char));

    if (sieve == NULL) {

        printf("Can't create an array for sieve\n");

        exit(3);

    }

*// marking numbers which are not prime by defenition*

    sieve[0] = 1;

    sieve[1] = 1;

    int cur\_thread = 0; *// id of current thread*

    for (long long i = 2; i \* i <= num; ++i) {

        if (sieve[i] == 1) { *// skipping not prime numbers*

            continue;

        }

        if (cur\_thread >= threads\_num) {

*// we should wait while necessery thread is working*

            pthread\_join(threads[cur\_thread % threads\_num], NULL);

        }

        args[cur\_thread % threads\_num] = i; *// copying argument for thread function to a special array*

        pthread\_create(&threads[cur\_thread % threads\_num], NULL, sieve\_step, &args[cur\_thread % threads\_num]);

        ++cur\_thread;

    }

*// waiting for all threads*

    for (int i = 0; i < threads\_num; ++i) {

        pthread\_join(threads[i], NULL);

    }

    if (sieve[num] == 1) {

        printf("%lld is not a prime number\n", num);

    }

    else {

        printf("%lld is a prime number\n", num);

    }

    free(sieve);

    free(threads);

    free(args);

}

1. **Демонстрация работы программы**

**mosik@LAPTOP-69S778GL**:~/os\_lab3$ gcc main.c -o main -pthread

**mosik@LAPTOP-69S778GL**:~/os\_lab3$ ./main

Syntax: ./main Number\_of\_threads

**mosik@LAPTOP-69S778GL**:~/os\_lab3$ ./main 1

Enter a number you want to check: 13

13 is a prime number

**mosik@LAPTOP-69S778G**L:~/os\_lab3$ ./main 1

Enter a number you want to check: 14

14 is not a prime number

**mosik@LAPTOP-69S778GL**:~/os\_lab3$ ./main 1

Enter a number you want to check: 1000000

1000000 is not a prime number

**mosik@LAPTOP-69S778GL**:~/os\_lab3$ ./main 1

Enter a number you want to check: 983

983 is a prime number

**mosik@LAPTOP-69S778GL**:~/os\_lab3$ ./main 1

Enter a number you want to check: 3

3 is a prime number

**mosik@LAPTOP-69S778GL**:~/os\_lab3$ ./main 1

Enter a number you want to check: 1

1 is not a prime number

**mosik@LAPTOP-69S778GL**:~/os\_lab3$ ./main 1

Enter a number you want to check: -12

Can't create an array for sieve

**mosik@LAPTOP-69S778GL:**~/os\_lab3$ ./main 1

Enter a number you want to check: 0

0 is not a prime number

**mosik@LAPTOP-69S778GL**:~/os\_lab3$ cat test.txt

123123123

**mosik@LAPTOP-69S778GL**:~/os\_lab3$ time ./main 1 < test.txt

Enter a number you want to check: 123123123 is not a prime number

real 0m2.615s

user 0m2.188s

sys 0m0.141s

**mosik@LAPTOP-69S778GL**:~/os\_lab3$ time ./main 2 < test.txt

Enter a number you want to check: 123123123 is not a prime number

real 0m1.700s

user 0m3.047s

sys 0m0.250s

**mosik@LAPTOP-69S778GL**:~/os\_lab3$ time ./main 3 < test.txt

Enter a number you want to check: 123123123 is not a prime number

real 0m1.425s

user 0m3.844s

sys 0m0.344s

**mosik@LAPTOP-69S778GL**:~/os\_lab3$ time ./main 4 < test.txt

Enter a number you want to check: 123123123 is not a prime number

real 0m1.314s

user 0m4.656s

sys 0m0.438s

**mosik@LAPTOP-69S778GL**:~/os\_lab3$ time ./main 5 < test.txt

Enter a number you want to check: 123123123 is not a prime number

real 0m1.307s

user 0m5.297s

sys 0m0.844s

**mosik@LAPTOP-69S778GL**:~/os\_lab3$ time ./main 6 < test.txt

Enter a number you want to check: 123123123 is not a prime number

real 0m1.458s

user 0m6.313s

sys 0m1.250s

**mosik@LAPTOP-69S778GL**:~/os\_lab3$ time ./main 7 < test.txt

Enter a number you want to check: 123123123 is not a prime number

real 0m1.300s

user 0m6.391s

sys 0m1.328s

**mosik@LAPTOP-69S778GL**:~/os\_lab3$ time ./main 8 < test.txt

Enter a number you want to check: 123123123 is not a prime number

real 0m1.336s

user 0m6.984s

sys 0m1.453s

**mosik@LAPTOP-69S778GL**:~/os\_lab3$ time ./main 9 < test.txt

Enter a number you want to check: 123123123 is not a prime number

real 0m1.369s

user 0m7.016s

sys 0m1.266s

**mosik@LAPTOP-69S778GL**:~/os\_lab3$ time ./main 100 < test.txt

Enter a number you want to check: 123123123 is not a prime number

real 0m1.125s

user 0m5.641s

sys 0m1.328s

**mosik@LAPTOP-69S778GL**:~/os\_lab3$ time ./main 100000 < test.txt

Enter a number you want to check: 123123123 is not a prime number

real 0m1.247s

user 0m8.234s

sys 0m0.922s

**mosik@LAPTOP-69S778GL**:~/os\_lab3$ time ./main 10000000 < test.txt

Enter a number you want to check: 123123123 is not a prime number

real 0m1.234s

user 0m6.859s

sys 0m1.031s

**mosik@LAPTOP-69S778GL:~/os\_lab3$** strace -f -e trace="%process,write" -o strace\_log.txt ./main 3

Enter a number you want to check: 100

100 is not a prime number

**mosik@LAPTOP-69S778GL:~/os\_lab3$** cat strace\_log.txt

785 execve("./main", ["./main", "3"], 0x7fffc550ab10 /\* 30 vars \*/) = 0

785 arch\_prctl(ARCH\_SET\_FS, 0x7f1970de0740) = 0

785 write(1, "Enter a number you want to check"..., 34) = 34

*785 clone(child\_stack=0x7f19703cffb0, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tidptr=0x7f19703d09d0, tls=0x7f19703d0700, child\_tidptr=0x7f19703d09d0) = 795*

*785 clone(child\_stack=0x7f196fbbffb0, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tidptr=0x7f196fbc09d0, tls=0x7f196fbc0700, child\_tidptr=0x7f196fbc09d0) = 796*

*785 clone(child\_stack=0x7f196f3affb0, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tidptr=0x7f196f3b09d0, tls=0x7f196f3b0700, child\_tidptr=0x7f196f3b09d0) = 797*

795 exit(0) = ?

795 +++ exited with 0 +++

796 exit(0 <unfinished ...>

*785 clone( <unfinished ...>*

796 <... exit resumed>) = ?

796 +++ exited with 0 +++

*785 <... clone resumed> child\_stack=0x7f19703cffb0, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tidptr=0x7f19703d09d0, tls=0x7f19703d0700, child\_tidptr=0x7f19703d09d0) = 798*

797 exit(0) = ?

797 +++ exited with 0 +++

798 exit(0) = ?

798 +++ exited with 0 +++

785 write(1, "100 is not a prime number\n", 26) = 26

785 exit\_group(0) = ?

785 +++ exited with 0 +++

**mosik@LAPTOP-69S778GL:**~/os\_lab3$ strace -f -e trace="%process,write" -o strace\_log1.txt ./main 2

Enter a number you want to check: 1000

1000 is not a prime number

**mosik@LAPTOP-69S778GL**:~/os\_lab3$ cat strace\_log1.txt

812 execve("./main", ["./main", "2"], 0x7fffcd086980 /\* 30 vars \*/) = 0

812 arch\_prctl(ARCH\_SET\_FS, 0x7fa973c50740) = 0

812 write(1, "Enter a number you want to check"..., 34) = 34

*812 clone(child\_stack=0x7fa9733cffb0, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tidptr=0x7fa9733d09d0, tls=0x7fa9733d0700, child\_tidptr=0x7fa9733d09d0) = 822*

*812 clone(child\_stack=0x7fa972bbffb0, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tidptr=0x7fa972bc09d0, tls=0x7fa972bc0700, child\_tidptr=0x7fa972bc09d0) = 823*

823 exit(0 <unfinished ...>

822 exit(0 <unfinished ...>

823 <... exit resumed>) = ?

822 <... exit resumed>) = ?

823 +++ exited with 0 +++

822 +++ exited with 0 +++

*812 clone(child\_stack=0x7fa9733cffb0, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tidptr=0x7fa9733d09d0, tls=0x7fa9733d0700, child\_tidptr=0x7fa9733d09d0) = 824*

*812 clone(child\_stack=0x7fa972bbffb0, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tidptr=0x7fa972bc09d0, tls=0x7fa972bc0700, child\_tidptr=0x7fa972bc09d0) = 825*

824 exit(0) = ?

824 +++ exited with 0 +++

*812 clone( <unfinished ...>*

825 exit(0) = ?

*812 <... clone resumed> child\_stack=0x7fa9733cffb0, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tidptr=0x7fa9733d09d0, tls=0x7fa9733d0700, child\_tidptr=0x7fa9733d09d0) = 826*

825 +++ exited with 0 +++

*812 clone(child\_stack=0x7fa972bbffb0, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tidptr=0x7fa972bc09d0, tls=0x7fa972bc0700, child\_tidptr=0x7fa972bc09d0) = 827*

826 exit(0 <unfinished ...>

827 exit(0 <unfinished ...>

826 <... exit resumed>) = ?

827 <... exit resumed>) = ?

826 +++ exited with 0 +++

827 +++ exited with 0 +++

812 *clone(child\_stack=0x7fa9733cffb0, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tidptr=0x7fa9733d09d0, tls=0x7fa9733d0700, child\_tidptr=0x7fa9733d09d0) = 828*

*812 clone(child\_stack=0x7fa972bbffb0, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tidptr=0x7fa972bc09d0, tls=0x7fa972bc0700, child\_tidptr=0x7fa972bc09d0) = 829*

828 exit(0) = ?

828 +++ exited with 0 +++

*812 clone( <unfinished ...>*

829 exit(0 <unfinished ...>

*812 <... clone resumed> child\_stack=0x7fa9733cffb0, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tidptr=0x7fa9733d09d0, tls=0x7fa9733d0700, child\_tidptr=0x7fa9733d09d0) = 830*

829 <... exit resumed>) = ?

829 +++ exited with 0 +++

*812 clone(child\_stack=0x7fa972bbffb0, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tidptr=0x7fa972bc09d0, tls=0x7fa972bc0700, child\_tidptr=0x7fa972bc09d0) = 831*

830 exit(0) = ?

830 +++ exited with 0 +++

*812 clone( <unfinished ...>*

831 exit(0 <unfinished ...>

*812 <... clone resumed> child\_stack=0x7fa9733cffb0, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tidptr=0x7fa9733d09d0, tls=0x7fa9733d0700, child\_tidptr=0x7fa9733d09d0) = 832*

831 <... exit resumed>) = ?

831 +++ exited with 0 +++

832 exit(0) = ?

832 +++ exited with 0 +++

812 write(1, "1000 is not a prime number\n", 27) = 27

812 exit\_group(0) = ?

812 +++ exited with 0 +++

1. **Исследование ускорения и эффективности**

Для исследования ускорения и эффективности параллельного решета Эратосфена замерим время работы программы для следующих входных данных:

1. 3257
2. 10000079
3. 1000000007

Время работы программы будет замеряться при помощи функций стандартной библиотеки time.h. Нужно учитывать, что время работы может варьироваться в небольших пределах из-за постоянной работы фоновых процессов. Результаты будут занесены в таблицы.

Таблица 1. Исследование обработки числа 3257

|  |  |  |  |
| --- | --- | --- | --- |
| Количество потоков (n) | Время работы программы (Tn), сек | Ускорение  (Sn = T1 / Tn) | Эффективность  (Xn = Sn / n) |
| 1 | 0,0026 | - | - |
| 2 | 0,0017 | 1,53 | 0,77 |
| 3 | 0,0014 | 1,86 | 0,62 |
| 4 | 0,0012 | 2,17 | 0,54 |
| 5 | 0,0013 | 2,00 | 0,40 |
| 6 | 0,0012 | 2,17 | 0,36 |
| 7 | 0,0011 | 2,36 | 0,34 |
| 8 | 0,0012 | 2,17 | 0,27 |
| 9 | 0,0013 | 2,00 | 0,22 |
| 10 | 0,0013 | 2,00 | 0,20 |

Таблица 2. Исследование обработки числа 10000079

|  |  |  |  |
| --- | --- | --- | --- |
| Количество потоков (n) | Время работы программы (Tn), сек | Ускорение  (Sn = T1 / Tn) | Эффективность  (Xn = Sn / n) |
| 1 | 0,1764 | - | - |
| 2 | 0,1073 | 1,64 | 0,82 |
| 3 | 0,0935 | 1,88 | 0,63 |
| 4 | 0,0829 | 2,13 | 0,53 |
| 5 | 0,0850 | 2,08 | 0,42 |
| 6 | 0,0869 | 2,03 | 0,34 |
| 7 | 0,0830 | 2,13 | 0,30 |
| 8 | 0,0825 | 2,14 | 0,27 |
| 9 | 0,0850 | 2,08 | 0,23 |
| 10 | 0,0840 | 2,10 | 0,21 |

Таблица 3. Исследование обработки числа 1000000007

|  |  |  |  |
| --- | --- | --- | --- |
| Количество потоков (n) | Время работы программы (Tn), сек | Ускорение  (Sn = T1 / Tn) | Эффективность  (Xn = Sn / n) |
| 1 | 20,60 | - | - |
| 2 | 14,95 | 1,38 | 0,69 |
| 3 | 13,91 | 1,48 | 0,49 |
| 4 | 13,45 | 1,53 | 0,38 |
| 5 | 13,62 | 1,51 | 0,30 |
| 6 | 13,84 | 1,49 | 0,25 |
| 7 | 12,85 | 1,60 | 0,23 |
| 8 | 13,71 | 1,50 | 0,19 |
| 9 | 16,43 | 1,25 | 0,14 |
| 10 | 17,44 | 1,18 | 0,12 |

По этим данным можно увидеть, что значительный выигрыш по времени мы имеем только при обработке действительно больших чисел (порядка 107 и выше). Оптимальнее всего использовать 2-3 потока. Таким образом можно ускорить время работы программы примерно в полтора раза. Средняя эффективность составляет 0,5. Если использовать большее количество потоков, то выигрыш от параллельной обработки будет перекрываться затратами на создание и регулирование потоков.

1. **Выводы**

Язык Си позволяет пользователю взаимодействовать с потоками операционной системы. Для этого на Unix-подобных системах требуется подключить библиотеку pthread.h.

Создание потоков происходит быстрее, чем создание процессов, а все потоки используют одну и ту же область данных. Поэтому многопоточность – один из способов ускорить обработку каких-либо данных: выполнение однотипных, не зависящих друг от друга задач, можно поручить отдельным потокам, которые будут работать парарллельно.

Средствами языка Си можно совершать системные запросы на создание потока, ожидания завершения потока, а также использовать различные примитивы синхронизации.

В данной лабораторной работе был реализован и исследован алгоритм проверки числа на простоту при помощи решета Эратосфена. Установили, что при использовании двух-трёх потоков можно получить выигрыш по времени примерно в полтора раза, что значительно ускоряет обработку большого количества чисел. Но при использовании большего количества потоков ускорение не будет большим, так как операционной системе приходится тратить больше времени на выделение памяти под потоки и на их регулирование.