Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №3 по курсу
«Операционные системы»

Студент: Маринин И.С.
Группа: М8о–208Б–19
Вариант: 19
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2021.

Постановка задачи

Цель работы

Приобретение практических навыков в:

· Управление потоками в ОС

· Обеспечение синхронизации между потоками

Задание

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.

Вариант 19: Необходимо реализовать проверку числа на простоту при помощи алгоритма «решето Эратосфена».

Общие сведения о программе

Программа написана на языке Си в операционной системе UNIX. Для компиляции программы требуется указать ключ -pthread. Для запуска программы в качестве 1 аргумента командной строки необходимо указать количество потоков, которые могут быть использованы программой, а в качестве 2 аргумента — тестируемое число.

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.

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 и само число для проверки. 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 число i^2. Для ускорения алгоритма начнём проверку именно с этого числа и будем помечать каждое i-ое число, начиная с i^2. 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.

Основные файлы программы

task_19.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

char* sieve;
int num;

void* SieveStep(void* i_void){
    int i = *(int*)i_void;
```

```
    for (int j = i + i; j <= num; j += i) {
        sieve[j] = 1;
    }
    pthread_exit(NULL);
}

int main(int argc, char* argv[]) {
    if (argc!= 3) {
        printf("Sitax: ./a.out Threads_num Num_for_test\n");
        return 0;
    }

    int threads_num = atoi(argv[1]);
    num = atoi(argv[2]);

    pthread_t* threads = (pthread_t*)calloc(threads_num,
sizeof(pthread_t));

    // массив для аргументов, которые будут переданы в функцию
потока
    if (threads == NULL) {
        printf("Can't create an array for arguments for
threads\n");
        exit(EXIT_FAILURE);
    }

    // массив для аргументов, которые будут переданы в функцию
потока
    int *args = (int*)malloc(threads_num * sizeof(int));

    // создание массива, заполненного 0 для сита
    // 0 - простое число, 1 - не простое число
    sieve = (char*)calloc((num + 1), sizeof(char));
    if (sieve == NULL) {
        printf("Can't create an array for sieve\n");
        exit(EXIT_FAILURE);
    }

    // маркировка чисел, которые не являются простыми по опреде-
лению
```

```
    sieve[0] = 1;
    sieve[1] = 1;

    int cur_thread = 0; // id текущего потока
    for (int i = 2; i * i <= num; ++i) {
        if (sieve[i] == 1) continue; // пропуск не простых чисел
        if (cur_thread >= threads_num) {
            pthread_join(threads[cur_thread %
threads_num] ,NULL);
        }

        args[cur_thread % threads_num] = i; // копирование аргу-
мента для функции потока в специальный массив
        pthread_create(&threads[cur_thread % threads_num], NULL,
SieveStep, &args[cur_thread % threads_num]);
        ++cur_thread;
    }

    for (int i = 0; i < threads_num; ++i) {
        pthread_join(threads[i], NULL);
    }

    if (sieve[num] == 1)
        printf("%d is not a prime number\n", num);
    else
        printf("%d is a prime number\n", num);

    free(sieve);
    free(threads);
    free(args);
}
```

Пример работы

ivanmarinin@MacBook-Air-Ivan src % gcc task_19.c

ivanmarinin@MacBook-Air-Ivan src % ./a.out 10 10

10 is not a prime number

ivanmarinin@MacBook-Air-Ivan src % ./a.out 1000 1

1 is not a prime number

ivanmarinin@MacBook-Air-Ivan src % ./a.out 1000 19999999

19999999 is a prime number

ivanmarinin@MacBook-Air-Ivan src % ./a.out 1 19999999

19999999 is a prime number

ivanmarinin@MacBook-Air-Ivan src % ./a.out 1 0

0 is not a prime number

ivanmarinin@MacBook-Air-Ivan src % ./a.out 5232 5537429

5537429 is not a prime number

ivanmarinin@MacBook-Air-Ivan src % ./a.out 2 5537429

5537429 is not a prime number

Вывод

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 параллельно.

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.