МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Московский Авиационный Институт
(Национальный Исследовательский Университет)

Институт №8 "Информационные технологий и прикладная математика"
Кафедра 806 "Вычислительная математика и программирование"

Лабораторная работа №3
по курсу "Операционные системы"
3 семестр

Студент: Леухин М. В.
Группа: М8О-206Б-20
Преподаватель: Соколов А. А.
Дата: 13.11.21
Оценка: 5
Подпись: _____

Москва, 2021

Содержание

1 Постановка задачи

Цель работы: приобретение практических навыков в управлении потоками в операционной системе и обеспечении синхронизации между ними.

Задание: 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. В отчете привести исследование зависимости ускорения и эффективности алгоритма от входящих данных и количества потоков. Получившиеся результаты необходимо объяснить.

Вариант 19: необходимо реализовать проверку числа на простоту при помощи алгоритма "решето Эратосфена".

2 Основная часть

2.1 Листинг программы

Файл main.c:

```c
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <sys/time.h>

typedef struct {
    bool* sieve;
    long long size;
    long long prime;
} arg_t;

void handle_error(bool expr, char* msg){
    if (expr){
        perror(msg);
        exit(-1);
    }
}

void* func(void* args){
    arg_t* arg = (arg_t*) args;
    printf("# thread %lu work with prime %lld\n", pthread_self(),
       arg->prime);
    for (int i = 2; i * arg->prime <= arg->size; ++i){
        arg->sieve[i * arg->prime] = true;
    }
    free(arg); arg = NULL;
    return NULL;
}


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

    handle_error(argc < 2, "specify number of threads");

    long long threads_max = strtol(argv[1], NULL, 10);
    handle_error(threads_max < 1, "number of threads can't be
       less than 1");
    if (threads_max > 8){
        printf("The number of threads can't be more than 8. Set
           to 8.\n");
        threads_max = 8;
```

```
41      }
42
43      printf("Enter a natural number: ");
44      long long n; char* cmd = malloc(64 * sizeof(char));
45      fgets(cmd, 1024, stdin);
46      for (int i = 0; i < 1024; ++i){
47          if (cmd[i] == '\n'){ cmd[i] = '\0'; break; }
48          if (cmd[i] == '\0'){ break; }
49          if (cmd[i] < '0' || cmd[i] > '9'){
50              printf("invalid number\n");
51              return 0;
52          }
53      }
54
55      n = strtol(cmd, NULL, 10);
56
57      bool* sieve = malloc((n + 1) * sizeof(bool));
58      memset(sieve, false, n + 1);
59
60      pthread_t* threads = calloc(threads_max, sizeof(pthread_t));
61
62      struct timeval start, stop;
63      gettimeofday(&start, NULL);
64
65
66      long long primes[9] = {2, 3, 5, 7, 11, 13, 17, 19, 23};
67
68      printf("1 iteration:\n");
69
70      for (int i = 0; i < threads_max; ++i){
71          arg_t* arg = malloc(sizeof(arg_t));
72          arg->sieve = sieve;
73          arg->size = n;
74          arg->prime = primes[i];
75          if (pthread_create(&threads[i], NULL, func, arg) != 0){
76              printf("Unable to create %d-th thread\n", i + 1);
77              free(threads);
78              free(arg);
79              exit(-1);
80          }
81      }
82
83      for (int j = 0; j < threads_max; ++j){
84          if (pthread_join(threads[j], NULL) != 0){
85              printf("Unable to join %d-th thread\n", j + 1);
86              free(threads);
87              exit(-1);
88          };
89      }
90
```

```
91      long long prime = primes[threads_max]; int counter = 2;
92
93      while (prime * prime <= n){
94
95          printf("%d iteration:\n", counter);
96          ++counter;
97          int threads_num = 0;
98          while (threads_num < threads_max){
99              if (sieve[prime]){ ++prime; continue; }
100             arg_t* arg = malloc(sizeof(arg_t));
101             arg->sieve = sieve;
102             arg->size = n;
103             arg->prime = prime;
104             if (pthread_create(&threads[threads_num], NULL, func,
                    arg) != 0){
105                 printf("Unable to create %d-th thread\n",
                        threads_num + 1);
106                 free(threads);
107                 free(arg);
108                 exit(-1);
109             }
110             ++threads_num;
111             ++prime;
112         }
113
114         for (int j = 0; j < threads_max; ++j){
115             if (pthread_join(threads[j], NULL) != 0){
116                 printf("Unable to join %d-th thread\n", j + 1);
117                 free(threads);
118                 exit(-1);
119             };
120         }
121
122     }
123
124     gettimeofday(&stop, NULL);
125
126     sieve[n] == 0 ? printf("Result: %lld is prime ", n) :
            printf("Result: %lld is composite ", n);
127     printf("(in %lu ms)\n", stop.tv_sec - start.tv_sec);
128
129     free(threads); free(sieve); free(cmd);
130
131     return 0;
132
133 }
```

2.2 Результат работы программы

Первый тест:

```
matvey@matvey-Lenovo-IdeaPad-S340-15API:~/labs/2os/3lab/src$ ./a.out 1
Enter a natural number: 1000000007
1 iteration:
# thread 140316983691008 work with prime 2
2 iteration:
# thread 140316983691008 work with prime 3
3 iteration:
# thread 140316983691008 work with prime 5
...
3400 iteration:
# thread 140316983691008 work with prime 31601
3401 iteration:
# thread 140316983691008 work with prime 31607
3402 iteration:
# thread 140316983691008 work with prime 31627
Result: 1000000007 is prime (in 22 ms)
```

Второй тест:

```
matvey@matvey-Lenovo-IdeaPad-S340-15API:~/labs/2os/3lab/src$ ./a.out 4
Enter a natural number: 1000000007
1 iteration:
# thread 140602044729088 work with prime 5
# thread 140602036336384 work with prime 7
# thread 140602061514496 work with prime 2
# thread 140602053121792 work with prime 3
2 iteration:
# thread 140602036336384 work with prime 11
# thread 140602044729088 work with prime 13
# thread 140602053121792 work with prime 17
# thread 140602061514496 work with prime 19
...
850 iteration:
# thread 140602036336384 work with prime 31567
# thread 140602044729088 work with prime 31573
# thread 140602053121792 work with prime 31583
# thread 140602061514496 work with prime 31601
851 iteration:
# thread 140602061514496 work with prime 31607
# thread 140602053121792 work with prime 31627
# thread 140602044729088 work with prime 31643
# thread 140602036336384 work with prime 31649
Result: 1000000007 is prime (in 12 ms)
```

Третий тест:

```
matvey@matvey-Lenovo-IdeaPad-S340-15API:~/labs/2os/3lab/src$ ./a.out 8
```

```
Enter a natural number: 1000000007
1 iteration:
# thread 139731472094976 work with prime 3
# thread 139731446916864 work with prime 11
# thread 139731480487680 work with prime 2
# thread 139731430131456 work with prime 17
# thread 139731463702272 work with prime 5
# thread 139731455309568 work with prime 7
# thread 139731421738752 work with prime 19
# thread 139731438524160 work with prime 13
# thread 139731463702272 work with prime 47
...
426 iteration:
# thread 139731421738752 work with prime 31607
# thread 139731430131456 work with prime 31627
# thread 139731438524160 work with prime 31643
# thread 139731446916864 work with prime 31649
# thread 139731480487680 work with prime 31657
# thread 139731472094976 work with prime 31663
# thread 139731455309568 work with prime 31687
# thread 139731463702272 work with prime 31667
Result: 1000000007 is prime (in 11 ms)
```

3 Вывод

В ходе выполнения лабораторной работы я познакомился с тем, как создавать новые потоки в контексте процесса и как можно ими управлять. Для этого я изучил инструменты, которые предоставляет операционная система для создания и управдения потоками. Также я реализовал программу, которая позволяет определять простоту введённого натурального числа, причём делает она это с помощью многопоточной обработки. Тесты показывают, что использование нескольких потоков даёт ощутимый прирост в производительности лишь при использования до 4 потоков — в примерах выше выигрыш во времени при использовании 4 потоков был в 2 раза по сравнению с 1 потоком, а разницы между 4 и 8 потоками почти нет. Это связано стем, что при обработке в 8 потоков на самом деле одновременно все 8 потоков выполнятся не будут