Министерство науки и высшего образования

Московский Авиационный Институт
(национальный исследовательский университет)

ЛАБОРАТОРНАЯ РАБОТА №3
по курсе операционные системы I семестр 2022/2023

Студент: Катин Иван Вячеславович
Группа: М8О-210Б-21
Вариант: 17
Преподователь : Миронов Евгений Сергеевич
Оценка:
Дата:
Подпись:

Москва, 2022

Содержание

Постановка задачи

Вариант 17.
Найти в большом целочисленном массиве минимальный элемент

Цель работы

Целью является приобретение практических навыков в:

- Управление потоками в ОС
- Обеспечение синхронизации между потоками

Алгоритм решения

Найти максимальное число потоков. Читаем количество потоков, если их больше максимального числа потоков, устанавливаем это максимальное значение (от большего числа потоков не будет смысла), иначе устанавливаем заданное количество потоков. Заполняем массив рандомными числами. Создаем структуру **thread_data**, содержащую исходный массив, левую и правую границу участка массива, ссылка на переменную минимального числа. Эта структура данных будет аргументом в функции потока. Создаем массив **thread_data** длиной равной количеству потоков. Дальше делим поровну длину массива на количество потоков, остаток оставляем для последнего потока(записываем в структуру данных). Получается что каждый поток ищет минимальное значение на своём участке массива. Чтобы не получить **race condition**, создаем **mutex**, который будет лочить ссылку на минимальное значение, в которую записывается минимальное значение переменной. Создаем массив потоков, получаем время и запускаем потоки. Ожидаем, когда потоки закончат свою работу, и снова получаем время. Разность времени — время работы программы. Ищем просто минимальное число, чтобы проверить результат.

Код программы

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

typedef struct {
    int* array;
    int start_array;
    int end_array;
    int* minimum;
} thread_data;

pthread_mutex_t mutex;

void handle_error(bool expression, char* message) {
    if (expression) {
        write(fileno(stdout), message, strlen(message) * sizeof(char));
        exit(-1);
    }
}

void print_result(int minimum, int proof, int time, char* number_threads){
    char minimum_string[50];
    char proof_string[50];
    char time_string[50];
    sprintf(minimum_string, "%d", minimum);
    sprintf(proof_string, "%d", proof);
    sprintf(time_string, "%d", time);
    write(fileno(stdout), "number of thread = ", strlen("number of thread = "));
    write(fileno(stdout), number_threads, strlen(number_threads));
    write(fileno(stdout), "\n", 1);
    write(fileno(stdout), "time = ", strlen("time = "));
    write(fileno(stdout), time_string, strlen(time_string));
    write(fileno(stdout), "\n", 1);
    write(fileno(stdout), "minimum = ", strlen("minimum = "));
    write(fileno(stdout), minimum_string, strlen(minimum_string));
    write(fileno(stdout), "\n", 1);
    write(fileno(stdout), "proof = ", strlen("proof = "));
    write(fileno(stdout), proof_string, strlen(proof_string));
    write(fileno(stdout), "\n", 1);
}

void* function(void* parametrs){
    thread_data* data = (thread_data*) parametrs;
    int minimum = data->array[data->start_array];
    for(int i = data->start_array; i < data->end_array; i++){
        if(data->array[i] < minimum){
            minimum = data->array[i];
        }
    }
    pthread_mutex_lock(&mutex);
    if(minimum < *(data->minimum)){
        *(data->minimum) = minimum;
    }
    pthread_mutex_unlock(&mutex);
```

```
59      return NULL;
60  }
61
62  int main(int argc, char** argv){
63      int max_thread = sysconf(_SC_NPROCESSORS_ONLN);
64      handle_error(argc < 2, "write the number of threads");
65      int number_threads = atoi(argv[1]);
66      char max_threads_string[50];
67      sprintf(max_threads_string, "%d", number_threads);
68      if(number_threads > max_thread){
69          number_threads = max_thread;
70          sprintf(max_threads_string, "%d", number_threads);
71          write(fileno(stdout), "flow limit exceeded, installed ", strlen("flow limit exceeded, installed
                ));
72          write(fileno(stdout), max_threads_string, strlen(max_threads_string));
73          write(fileno(stdout), "\n", 1);
74      }
75      pthread_t* threads = (pthread_t*) malloc(number_threads * sizeof(pthread_t));
76      pthread_mutex_init(&mutex,NULL);
77      int size_array = 1000000 + rand() % 1000000;
78      int* array = (int*) malloc(size_array * sizeof(int));
79      for (int i = 0; i < size_array; ++i) {
80          array[i] = rand();
81      }
82      int minimum = array[0];
83      thread_data* array_data = (thread_data*) malloc(number_threads * sizeof(thread_data));
84      for (int i = 0; i < number_threads; ++i) {
85          array_data[i].array = array;
86          array_data[i].start_array = size_array / number_threads * i;
87          array_data[i].end_array = size_array / number_threads * (i+1);
88          array_data[i].minimum = &minimum;
89          if(i == number_threads - 1){
90              array_data[i].end_array += size_array % number_threads;
91          }
92      }
93      struct timeval start_time, stop_time;
94      handle_error(gettimeofday(&start_time, NULL), "Time set error");
95      for (int i = 0; i < number_threads; ++i) {
96          handle_error(pthread_create(&(threads[i]), NULL, function, &array_data[i]) != 0, "Error. Don't
                create thread");
97      }
98      for (int i = 0; i < number_threads; ++i) {
99          pthread_join(threads[i], NULL);
100     }
101     handle_error(gettimeofday(&stop_time, NULL) == -1, "Time set error");
102     pthread_mutex_destroy(&mutex);
103     int proof = array[0];
104     for (int i = 0; i < size_array; ++i) {
105         if(proof > array[i]){
106             proof = array[i];
107         }
108     }
109     free(threads);
110     free(array_data);
111     free(array);
112     int time = (stop_time.tv_sec - start_time.tv_sec) * 1000 + stop_time.tv_usec - start_time.tv_usec;
113     print_result(minimum,proof,time, max_threads_string);
114     return 0;
115 }
```

Тест кейсы

```
machine@Turing:~/CLionProjects/oos/lab3$ ./run 7
number of thread = 7
time = 660
minimum = 1210
proof = 1210


machine@Turing:~/CLionProjects/oos/lab3$ ./run 1
number of thread = 1
time = 2301
minimum = 1210
proof = 1210


machine@Turing:~/CLionProjects/oos/lab3$ ./run 8
number of thread = 8
time = 610
minimum = 1210
proof = 1210
```

Вывод

С помощью потоков можно ускорить программу в несколько раз, хотя, если создать потоков больше чем ядер процессоров, то от этого будет не особо много пользы, так как количество потоков, которые могут работать одновременно не больше количества ядер, однако есть некоторый тип задач, для которых планирование нескольких ожидающих потоков на одно ядро может обеспечить гораздо большую производительность программы.