

**Московский авиационный институт (национальный
исследовательский университет)**

Факультет информационных технологий и прикладной
математики Кафедра вычислительной математики и
программирования

Лабораторная работа по предмету "операционные системы"
№3

Студент: Мокиева С.А.

Преподаватель: Соколов А.А.

Группа: М8О-206Б-20

Дата: 12.04.2022

Оценка:

Подпись:

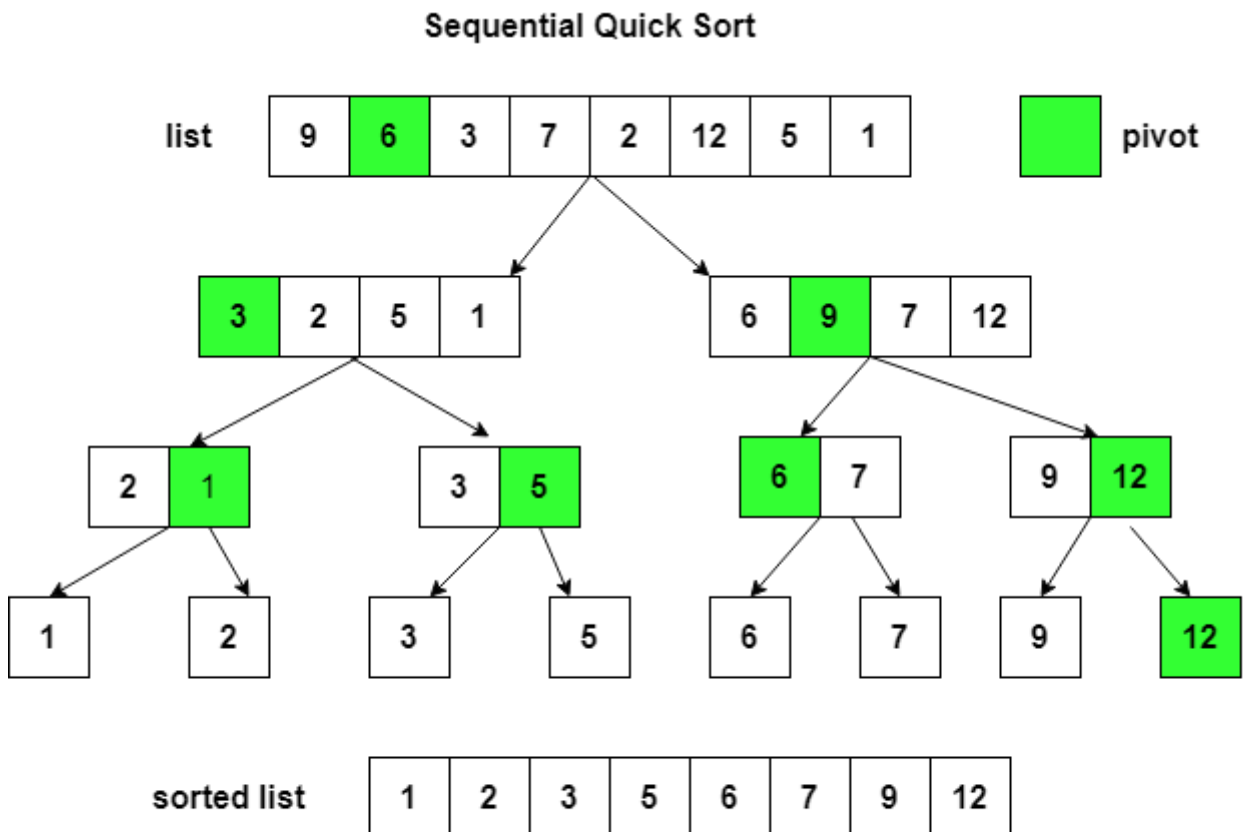
Москва 2022г.

Вариант 2

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.

Задание: Отсортировать массив целых чисел при помощи параллельного алгоритма быстрой сортировки.



q.opengenus.org

Реализация

main.cpp

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

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
//инициализация mutex по умолчанию, статика
int n_threads_creatable;
```

```

typedef struct quicksort_starter {
    int * arr; //массив
    int low; //номер первого элемента
    int high; //номер последнего элемента
}
quicksort_parameters;

void swap(int * a, int * b) { //функция, меняющая значения
элементов местами
    int t = * a;
    * a = * b;
    * b = t;
}

int partition(int * arr, int low, int high, int pivot) {
//операция по разделению
    int pivotValue = arr[pivot];
    swap( & arr[pivot], & arr[high]); // меняем местами опорный и
последний элементы
    int s = low; // индекс наименьшего элемента
    for (int i = low; i < high; i++) {
        // И если текущий элемент не больше, чем опорный, то меняем
местами текущий элемент и тот, который меньше
        if (arr[i] <= pivotValue) {
            swap( & arr[i], & arr[s]);
            s++;
        }
    }
    swap( & arr[s], & arr[high]);
    return s;
}

void quicksort_nonparallel(int * arr, int low, int high) {
//непарраллельная сортировка
    if (low < high) {
        int pivotPosition = low + (high - low) / 2;
        pivotPosition = partition(arr, low, high, pivotPosition);
        quicksort_nonparallel(arr, low, pivotPosition - 1);
        quicksort_nonparallel(arr, pivotPosition + 1, high);
    }
}

void quicksort_parallel(int * arr, int low, int high);

void * quicksort_parallel_runner(void * initialValues) {
    quicksort_parameters * parameters = initialValues;
    quicksort_parallel(
        parameters -> arr, parameters -> low, parameters -> high
    );
    return NULL;
}

void quicksort_parallel(int * arr, int low, int high) {

```

```

// Сортируем arr[low]..arr[high].
if (low >= high) { //если нечего сортировать, выходим
    return;
}

int pivotPos = low + (high - low) / 2; //находим номер
среднего элемента
pivotPos = partition(arr, low, high, pivotPos); //выполняем
операцию по разделению
// Теперь все элементы в arr[low]..arr[pivotPos-1] не
превышают все элементы
// в arr[pivotPos]..arr[high], поэтому каждую из этих двух
частей можно
// отсортировать независимо.

//необходимо защититься от случая, когда переменная
n_threads_creatable одновременно в разных потоках модифицируется
//это можно сделать с помощью mutex

pthread_mutex_lock(&mutex);
char is_ok = n_threads_creatable > 0;
if (is_ok) {
    n_threads_creatable--;
}
pthread_mutex_unlock(&mutex);

if (! is_ok) { //если потоков 0, дальше сортируем и выходим
    // Больше потоков не создать, так что продолжаем сортировку
    обеих частей
    // в текущем потоке.
    quicksort_nonparallel(arr, low, pivotPos - 1);
    quicksort_nonparallel(arr, pivotPos + 1, high);
    return;
}

//если нельзя создать, делаем это в том же потоке
//если можно, создаём новый поток и там вызываем сортировку
для левой части,
//а правую часть сортируем здесь и ждём, когда сортировка
левой части закончится

// Сортируем левую часть в новом потоке:

quicksort_parameters thread_param = {
    arr,
    low, //номер первого элемента
    pivotPos - 1 //номер крайнего левого от середины элемента
};
pthread_t thread;
pthread_create( & thread, NULL, quicksort_parallel_runner, &
thread_param);

```

```

    // Пока левая часть сортируется, сортируем правую в текущем
    потоке:
    quicksort_parallel(arr, pivotPos + 1, high);

    // Когда правая часть отсортируется, ожидаем сортировки
    левой части
    // (если она ещё не завершена).
    pthread_join(thread, NULL);
}

int main(int argc, char ** argv) {
    if (argc != 2) { //если аргументов не 2, выводим сообщение об
    ошибке
        printf("Usage: %s n_threads_creatable\n", argv[0]);
        return 1;
    }

    n_threads_creatable = strtol(argv[1], NULL, 10); //строка в
    long

    int size;
    scanf("%d", & size); //считываем, сколько элементов

    int * elements = malloc(size * sizeof(int)); //выделяем память
    на эти элементы

    for (int i = 0; i < size; i++) { //считываем все эти элементы
        scanf("%d", & elements[i]);
    }

    quicksort_parallel(elements, 0, size - 1); //вызываем
    параллельную быструю сортировку

    for (int i = 0; i < size; i++) { //выводим отсортированный
    массив элементов
        printf("%d\n", elements[i]);
    }

    free(elements); //освобождаем место в памяти
}

```

run.sh

```

#!/bin/bash

set -e # exit on error

gcc quicksort.c -o quicksort -pthread

./quicksort 4 <20.txt >/dev/null &

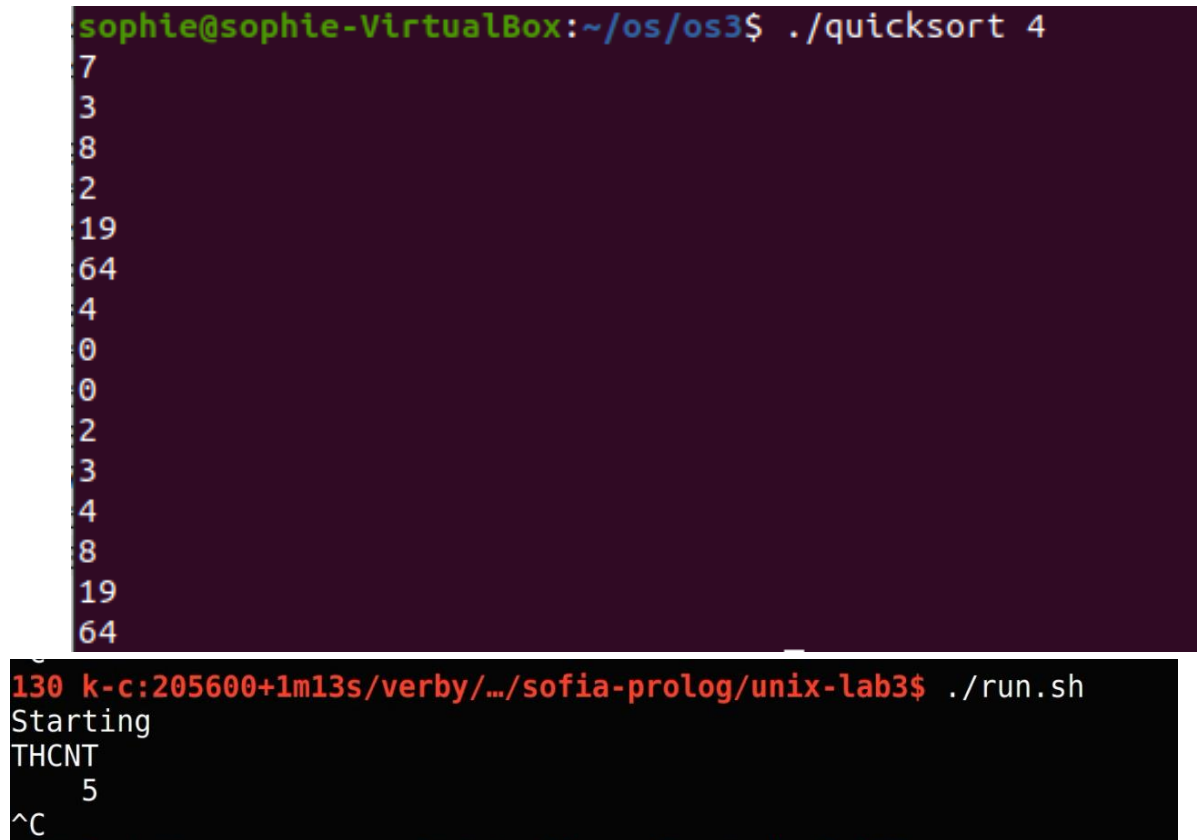
ps $! #просмотр PID последнего процесса в фоновом режиме

```

```
sleep 1
ps -o thcount $!#получить количество потоков для данного
процесса

wait #ожидает завершение процесса
```

Пример работы



The first screenshot shows a terminal window with the prompt 'sophie@sophie-VirtualBox:~/os/os3\$' and the command './quicksort 4'. Below the command, a list of numbers is displayed: 7, 3, 8, 2, 19, 64, 4, 0, 0, 2, 3, 4, 8, 19, 64. The second screenshot shows a terminal window with the prompt '130 k-c:205600+1m13s/verby/.../sofia-prolog/unix-lab3\$' and the command './run.sh'. Below the command, the text 'Starting' and 'THCNT' is displayed, followed by the number 5, and then '^C'.

(второй скриншот с другого компьютера, тк на моём из-за большого количества данных всё зависло)

Вывод

В ходе лабораторной работы я работала с потоками, mutex, изучила понятие параллельной быстрой сортировки. Её эффективность во многом зависит от выбора опорного элемента. Также я узнала, как можно вывести количество потоков, используемых программой с помощью стандартных средств операционной системы.