

ГУАП

КАФЕДРА № 43

ОТЧЕТ  
ЗАЩИЩЕН С ОЦЕНКОЙ  
ПРЕПОДАВАТЕЛЬ

старший преподаватель  
\_\_\_\_\_  
должность, уч. степень, звание

\_\_\_\_\_  
подпись, дата

М.Д. Поляк  
\_\_\_\_\_  
инициалы, фамилия

## ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №2

Синхронизация потоков средствами POSIX

по курсу: ОПЕРАЦИОННЫЕ СИСТЕМЫ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ гр. №

4131  
\_\_\_\_\_

\_\_\_\_\_  
подпись, дата

Б.А. Гусев  
\_\_\_\_\_  
инициалы, фамилия

Санкт-Петербург 2024

Цель работы: Знакомство с многопоточным программированием и методами синхронизации потоков средствами POSIX.

### **Задание на лабораторную работу**

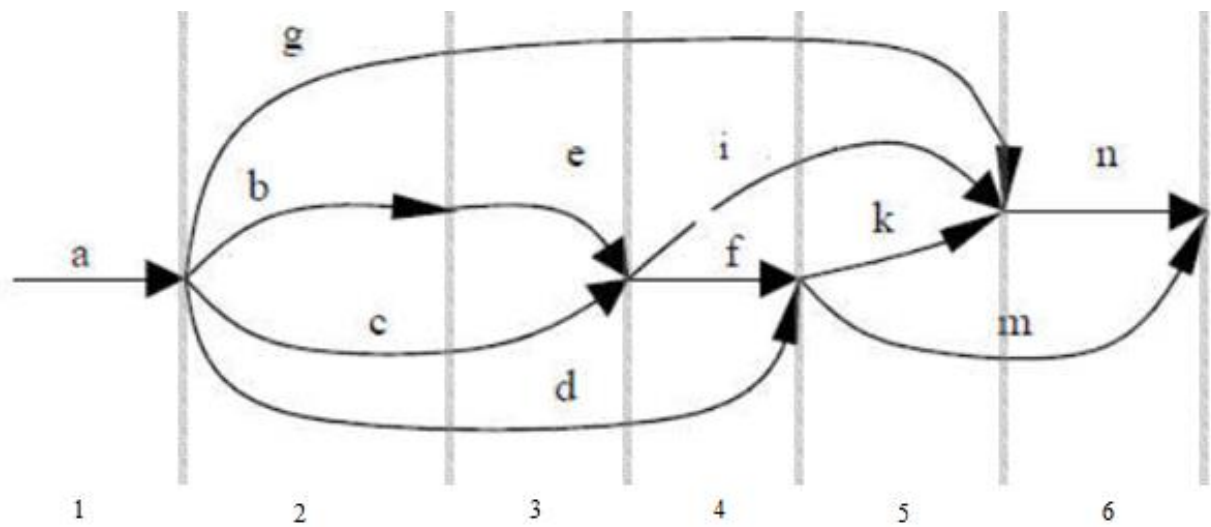
С помощью таблицы вариантов заданий выбрать граф запуска потоков в соответствии с номером варианта. Вершины графа являются точками запуска/завершения потоков, дугами обозначены сами потоки. Длину дуги следует интерпретировать как ориентировочное время выполнения потока. В процессе своей работы каждый поток должен в цикле выполнять два действия:

- выводить букву имени потока в консоль;
- вызывать функцию `computation()` для выполнения вычислений, требующих задействования ЦП на длительное время. Эта функция уже написана и подключается из заголовочного файла `lab2.h`, изменять ее не следует.

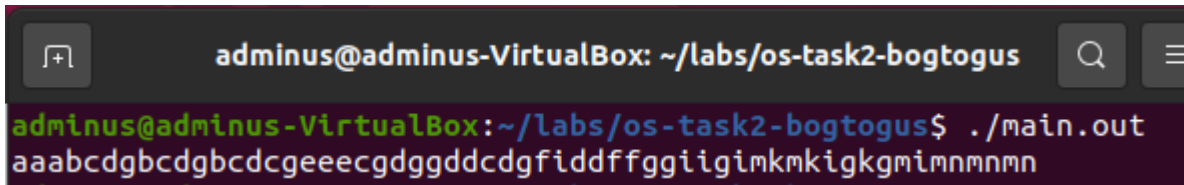
В соответствии с вариантом выделить на графе две группы с выполняющимися параллельно потоками. В первой группе потоки не синхронизированы, параллельное выполнение входящих в группу потоков происходит за счет планировщика задач (см. примеры 1 и 2). Вторая группа синхронизирована семафорами и потоки внутри группы выполняются в строго зафиксированном порядке: входящий в группу поток передает управление другому потоку после каждой итерации цикла (см. пример 3 и задачу производителя и потребителя). Таким образом потоки во второй группе выполняются в строгой очередности.

С использованием средств POSIX реализовать программу для последовательно-параллельного выполнения потоков в ОС Linux или Mac OS X. Запрещается использовать какие-либо библиотеки и модули, решающие задачу кроссплатформенной разработки многопоточных приложений (`std::thread`, `Qt Thread`, `Boost Thread` и т.п.), а также функции приостановки выполнения программы за исключением `pthread_yield()`.

Номер варианта	Номер графа запуска потоков	Интервалы с несинхронизированными потоками	Интервалы с чередованием потоков
3	7	dfgi	bcdg

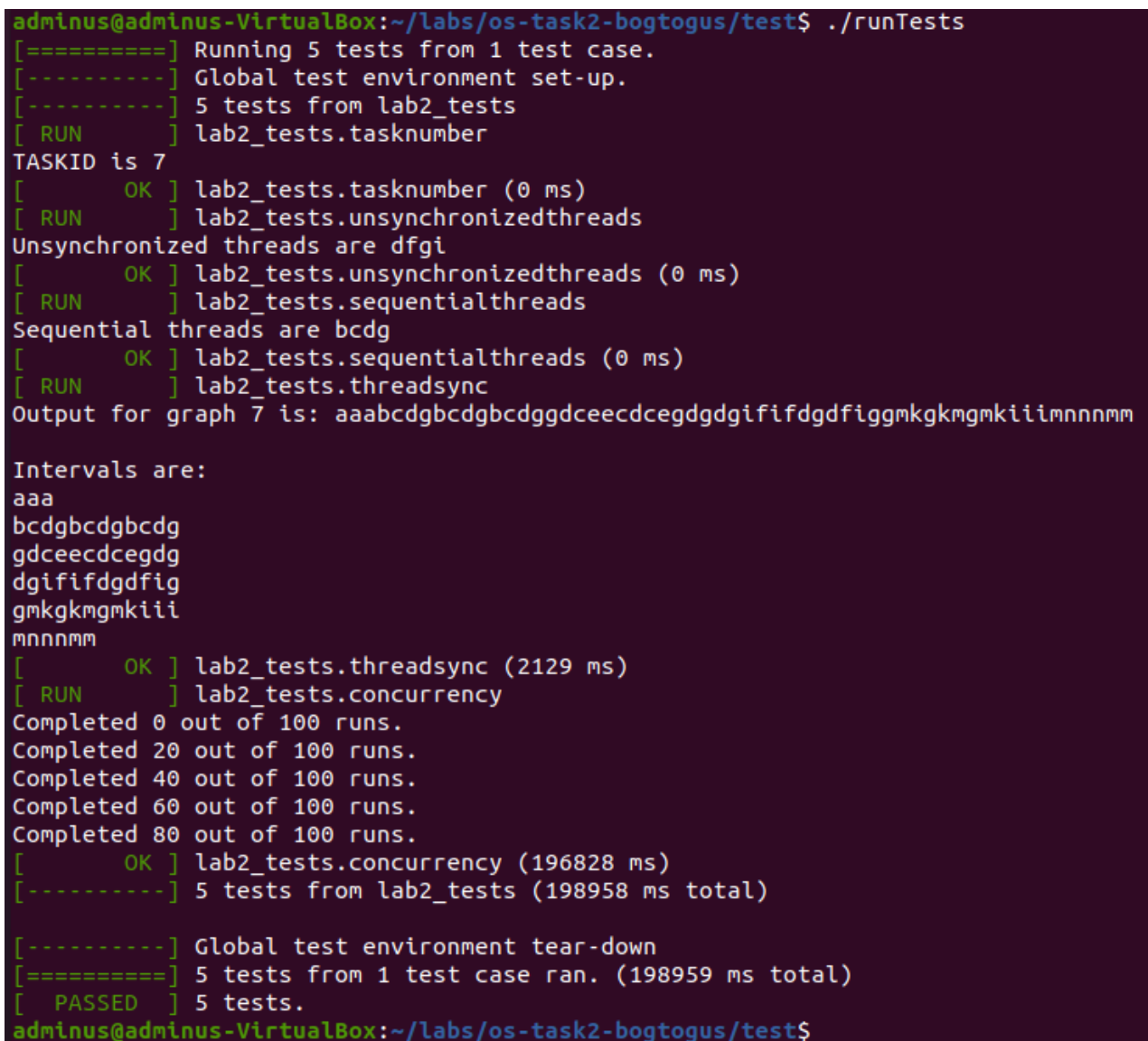


## Результат выполнения



```
adminus@adminus-VirtualBox: ~/labs/os-task2-bogtodus
adminus@adminus-VirtualBox:~/labs/os-task2-bogtodus$ ./main.out
aaabcdgbcdbcdgcgeecgdggddcdgfidffggigimkmgkigmimmmmm
```

Рисунок 1 - вывод последовательности символов потоков.



```
adminus@adminus-VirtualBox:~/labs/os-task2-bogtodus/test$ ./runTests
[=====] Running 5 tests from 1 test case.
[-----] Global test environment set-up.
[-----] 5 tests from lab2_tests
[ RUN    ] lab2_tests.tasknumber
TASKID is 7
[      OK ] lab2_tests.tasknumber (0 ms)
[ RUN    ] lab2_tests.unsynchronizedthreads
Unsynchronized threads are dfgi
[      OK ] lab2_tests.unsynchronizedthreads (0 ms)
[ RUN    ] lab2_tests.sequentialthreads
Sequential threads are bcdg
[      OK ] lab2_tests.sequentialthreads (0 ms)
[ RUN    ] lab2_tests.threadsync
Output for graph 7 is: aaabcdgbcdbcdggdceecdcegdgdgifdgdfiggmkgkmgmkiiimnnmm

Intervals are:
aaa
bcdgbcdbcdg
gdceecdcegdg
dgifidgdfig
gmkgkmgmkiii
mnnmmmm
[      OK ] lab2_tests.threadsync (2129 ms)
[ RUN    ] lab2_tests.concurrency
Completed 0 out of 100 runs.
Completed 20 out of 100 runs.
Completed 40 out of 100 runs.
Completed 60 out of 100 runs.
Completed 80 out of 100 runs.
[      OK ] lab2_tests.concurrency (196828 ms)
[-----] 5 tests from lab2_tests (198958 ms total)

[-----] Global test environment tear-down
[=====] 5 tests from 1 test case ran. (198959 ms total)
[ PASSED ] 5 tests.
adminus@adminus-VirtualBox:~/labs/os-task2-bogtodus/test$
```

Рисунок 2 - успешно пройденные тесты.

## Листинг

```
#include "lab2.h"
#include <cstring>
#include <semaphore.h>

#define NUMBER_OF_THREADS 11

// thread identifiers
pthread_t tid[NUMBER_OF_THREADS];
// critical section lock
pthread_mutex_t lock;
// semaphores for sequential threads
sem_t semaphor_1, semaphor_2, semaphor_4, semAllow_4,
semaphor_5, semaphor_6, semB, semC, semD, semG;

int err;

unsigned int lab2_thread_graph_id() {
    return 7;
}

const char* lab2_unsynchronized_threads() {
    return "dfgi";
}

const char* lab2_sequential_threads() {
    return "bcdg";
}

void *thread_b(void *ptr);
void *thread_c(void *ptr);
void *thread_d(void *ptr);
void *thread_f(void *ptr);
void *thread_e(void *ptr);
void *thread_g(void *ptr);
void *thread_k(void *ptr);
void *thread_i(void *ptr);
void *thread_m(void *ptr);
void *thread_n(void *ptr);

/*
0  a
1  b
```

```
2  c
3  d
4  e
5  f
6  g
7  i
8  k
9  m
10 n
*/
```

```
void *thread_a(void *ptr) {
    for (int i = 0; i < 3; ++i) {
        pthread_mutex_lock(&lock);
        std::cout << "a" << std::flush;
        pthread_mutex_unlock(&lock);
        computation();
    }
    err = pthread_create(&tid[1], NULL, thread_b, NULL);
    if (err != 0)
        std::cerr << "Невозможно создать поток. Ошибка:" <<
        strerror(err) << std::endl;
    err = pthread_create(&tid[2], NULL, thread_c, NULL);
    if (err != 0)
        std::cerr << "Невозможно создать поток. Ошибка:" <<
        strerror(err) << std::endl;
    err = pthread_create(&tid[3], NULL, thread_d, NULL);
    if (err != 0)
        std::cerr << "Невозможно создать поток. Ошибка:" <<
        strerror(err) << std::endl;
    err = pthread_create(&tid[6], NULL, thread_g, NULL);
    if (err != 0)
        std::cerr << "Невозможно создать поток. Ошибка:" <<
        strerror(err) << std::endl;
    // wait for thread D to finish
    return ptr;
}
```

```
void *thread_b(void *ptr) {
    for (int i = 0; i < 3; ++i) {
        sem_wait(&semB);
        pthread_mutex_lock(&lock);
        std::cout << "b" << std::flush;
        pthread_mutex_unlock(&lock);
    }
}
```

```

        computation();
        sem_post(&semC);
    }
    return ptr;
}

void *thread_c(void *ptr) {
    for (int i = 0; i < 3; ++i) {
        sem_wait(&semC);
        pthread_mutex_lock(&lock);
        std::cout << "c" << std::flush;
        pthread_mutex_unlock(&lock);
        computation();
        sem_post(&semD);
    }
    pthread_join(tid[1], NULL);
    sem_post(&semaphor_1);
    sem_wait(&semaphor_4);
    for (int i = 0; i < 3; ++i) {
        pthread_mutex_lock(&lock);
        std::cout << "c" << std::flush;
        pthread_mutex_unlock(&lock);
        computation();
    }
    return ptr;
}

void *thread_d(void *ptr) {
    for (int i = 0; i < 3; ++i) {
        sem_wait(&semD);
        pthread_mutex_lock(&lock);
        std::cout << "d" << std::flush;
        pthread_mutex_unlock(&lock);
        computation();
        sem_post(&semG);
    }
    sem_post(&semaphor_2);
    sem_wait(&semAllow_4);
    for (int i = 0; i < 3; ++i) {
        pthread_mutex_lock(&lock);
        std::cout << "d" << std::flush;
        pthread_mutex_unlock(&lock);
        computation();
    }
}

```

```

    sem_wait(&semaphor_5);
    sem_post(&semaphor_2);
    for (int i = 0; i < 3; ++i) {
        pthread_mutex_lock(&lock);
        std::cout << "d" << std::flush;
        pthread_mutex_unlock(&lock);
        computation();
    }
    return ptr;
}

void *thread_g(void *ptr) {
    // perform computations
    for (int i = 0; i < 3; ++i) {
        sem_wait(&semG);
        pthread_mutex_lock(&lock);
        std::cout << "g" << std::flush;
        pthread_mutex_unlock(&lock);
        computation();
        sem_post(&semB);
    }
    sem_wait(&semaphor_1);
    sem_wait(&semaphor_2);
    sem_post(&semaphor_4);
    sem_post(&semAllow_4);
    err = pthread_create(&tid[4], NULL, thread_e, NULL);
    if (err != 0)
        std::cerr << "Невозможно создать поток. Ошибка:" <<
strerror(err) << std::endl;
    for (int i = 0; i < 3; ++i) {
        pthread_mutex_lock(&lock);
        std::cout << "g" << std::flush;
        pthread_mutex_unlock(&lock);
        computation();
    }
    pthread_join(tid[2], NULL);
    pthread_join(tid[4], NULL);
    sem_post(&semaphor_5);
    sem_wait(&semaphor_2);
    err = pthread_create(&tid[5], NULL, thread_f, NULL);
    if (err != 0)
        std::cerr << "Невозможно создать поток. Ошибка:" <<
strerror(err) << std::endl;
    err = pthread_create(&tid[7], NULL, thread_i, NULL);

```



```

        if (err != 0)
            std::cerr << "Невозможно создать поток. Ошибка:" <<
strerror(err) << std::endl;
        for (int i = 0; i < 3; ++i) {
            pthread_mutex_lock(&lock);
            std::cout << "g" << std::flush;
            pthread_mutex_unlock(&lock);
            computation();
        }
        sem_post(&semaphor_6);
        sem_wait(&semaphor_1);
        for (int i = 0; i < 3; ++i) {
            pthread_mutex_lock(&lock);
            std::cout << "g" << std::flush;
            pthread_mutex_unlock(&lock);
            computation();
        }
        pthread_join(tid[7], NULL);
        return ptr;
    }

```

```

void *thread_e(void *ptr) {
    // perform computations
    for (int i = 0; i < 3; ++i) {
        pthread_mutex_lock(&lock);
        std::cout << "e" << std::flush;
        pthread_mutex_unlock(&lock);
        computation();
    }
    return ptr;
}

```

```

void *thread_f(void *ptr) {
    // perform computations
    for (int i = 0; i < 3; ++i) {
        pthread_mutex_lock(&lock);
        std::cout << "f" << std::flush;
        pthread_mutex_unlock(&lock);
        computation();
    }
    pthread_join(tid[3], NULL);
    return ptr;
}

```

```

void *thread_i(void *ptr) {
    // perform computations
    for (int i = 0; i < 3; ++i) {
        pthread_mutex_lock(&lock);
        std::cout << "i" << std::flush;
        pthread_mutex_unlock(&lock);
        computation();
    }
    pthread_join(tid[5], NULL);
    sem_post(&semaphor_1);
    sem_wait(&semaphor_6);
    err = pthread_create(&tid[8], NULL, thread_k, NULL);
    if (err != 0)
        std::cerr << "Невозможно создать поток. Ошибка:" <<
strerror(err) << std::endl;
    err = pthread_create(&tid[9], NULL, thread_m, NULL);
    if (err != 0)
        std::cerr << "Невозможно создать поток. Ошибка:" <<
strerror(err) << std::endl;
    pthread_join(tid[8], NULL);
    for (int i = 0; i < 3; ++i) {
        pthread_mutex_lock(&lock);
        std::cout << "i" << std::flush;
        pthread_mutex_unlock(&lock);
        computation();
    }
    sem_post(&semaphor_4);
    err = pthread_create(&tid[10], NULL, thread_n, NULL);
    if (err != 0)
        std::cerr << "Невозможно создать поток. Ошибка:" <<
strerror(err) << std::endl;
    pthread_join(tid[9], NULL);
    pthread_join(tid[10], NULL);
    return ptr;
}

void *thread_k(void *ptr) {
    // perform computations
    for (int i = 0; i < 3; ++i) {
        pthread_mutex_lock(&lock);
        std::cout << "k" << std::flush;
        pthread_mutex_unlock(&lock);
        computation();
    }
}

```

```

        return ptr;
    }

void *thread_m(void *ptr) {
    // perform computations
    for (int i = 0; i < 3; ++i) {
        pthread_mutex_lock(&lock);
        std::cout << "m" << std::flush;
        pthread_mutex_unlock(&lock);
        computation();
    }
    sem_wait(&semaphor_4);
    for (int i = 0; i < 3; ++i) {
        pthread_mutex_lock(&lock);
        std::cout << "m" << std::flush;
        pthread_mutex_unlock(&lock);
        computation();
    }
    return ptr;
}

void *thread_n(void *ptr) {
    // perform computations
    for (int i = 0; i < 3; ++i) {
        pthread_mutex_lock(&lock);
        std::cout << "n" << std::flush;
        pthread_mutex_unlock(&lock);
        computation();
    }
    return ptr;
}

int lab2_init() {
    // initilize mutex
    if (pthread_mutex_init(&lock, NULL) != 0) {
        std::cerr << "Mutex не инициализирован" << std::endl;
        return 1;
    }
    // initialize semaphores
    // THIS CODE WILL NOT RUN ON MacOS!
    // MacOS doesn't support unnamed semaphores, so named
    semaphores should be used instead
    if ( sem_init(&semaphor_1, 0, 0) != 0 ) {
        std::cerr << "Семафор #1 не инициализирован" <<

```

```

std::endl;
    return 1;
}
if ( sem_init(&semaphor_2, 0, 0) != 0 ) {
    std::cerr << "Семафор #2не инициализирован" <<
std::endl;
    return 1;
}
if ( sem_init(&semaphor_4, 0, 0) != 0 ) {
    std::cerr << "Семафор #1не инициализирован" <<
std::endl;
    return 1;
}
if ( sem_init(&semAllow_4, 0, 0) != 0 ) {
    std::cerr << "Семафор #2не инициализирован" <<
std::endl;
    return 1;
}
if ( sem_init(&semaphor_5, 0, 0) != 0 ) {
    std::cerr << "Семафор #2не инициализирован" <<
std::endl;
    return 1;
}
if ( sem_init(&semaphor_6, 0, 0) != 0 ) {
    std::cerr << "Семафор #2не инициализирован" <<
std::endl;
    return 1;
}
if ( sem_init(&semB, 0, 1) != 0 ) {
    std::cerr << "Семафор #1не инициализирован" <<
std::endl;
    return 1;
}
if ( sem_init(&semC, 0, 0) != 0 ) {
    std::cerr << "Семафор #2не инициализирован" <<
std::endl;
    return 1;
}
if ( sem_init(&semD, 0, 0) != 0 ) {
    std::cerr << "Семафор #3не инициализирован" <<
std::endl;
    return 1;
}
if ( sem_init(&semG, 0, 0) != 0 ) {

```

```

        std::cerr << "Семафор #4 не инициализирован" <<
std::endl;
        return 1;
    }

    // start the first thread
    err = pthread_create(&tid[0], NULL, thread_a, NULL);
    if (err != 0)
        std::cerr << "Невозможно создать поток. Ошибка:" <<
strerror(err) << std::endl;

    // ... and wait for it to finish
    pthread_join(tid[0], NULL);
    pthread_join(tid[6], NULL);

    // free resources
    pthread_mutex_destroy(&lock);
    sem_destroy(&semaphor_1);
    sem_destroy(&semaphor_2);
    sem_destroy(&semaphor_4);
    sem_destroy(&semAllow_4);
    sem_destroy(&semaphor_5);
    sem_destroy(&semaphor_6);
    sem_destroy(&semB);
    sem_destroy(&semC);
    sem_destroy(&semD);
    sem_destroy(&semG);
    std::cout << std::endl;
    // success
    return 0;
}

```

## Исходный код

```
#!/usr/bin/env bash

# edit the code below and add your code
# отредактируйте код ниже и добавьте свой

# Переменная с номером варианта (константа):
TASKID=3

# Дополнительные переменные (должны вычисляться динамически):
VAR_1=$(grep -c . dns-tunneling.log)

# Заголовок файла
echo "<dnslog>" >> results.txt

# Фильтрация первых 20 строк файла dns-tunneling.log и
конвертация в формат XML
grep -m 20 "" dns-tunneling.log | awk -F'\t' '{print
"<row>\n\t<timestamp>" $4 "</timestamp>\n\t<client_ip>" $5
"</client_ip>\n\t<client_port>" $6 "</client_port>\n</row>"}'
>> results.txt

# Окончание файла
echo "</dnslog>" >> results.txt

# Подсчет количества записей с IP-адресами из подсети 10.1.*.*
VAR_2=$(grep -E -c "10\.1\. [0-9]{1,3}\. [0-9]{1,3}"
results.txt)
```

## **Выводы**

В результате выполнения лабораторной работы был изучен синтаксис скриптов `bash`, на практике применены утилиты, позволяющие обрабатывать лог-файл.