

---

КАФЕДРА

ОТЧЕТ  
ЗАЩИЩЕН С ОЦЕНКОЙ  
РУКОВОДИТЕЛЬ

---

должность, уч. степень, звание

---

подпись, дата

---

инициалы, фамилия

Отчет о лабораторной работе №2  
**Разработка многопоточного приложения средствами POSIX**

По дисциплине: ОПЕРАЦИОННЫЕ СИСТЕМЫ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. №

---

подпись, дата

---

инициалы, фамилия

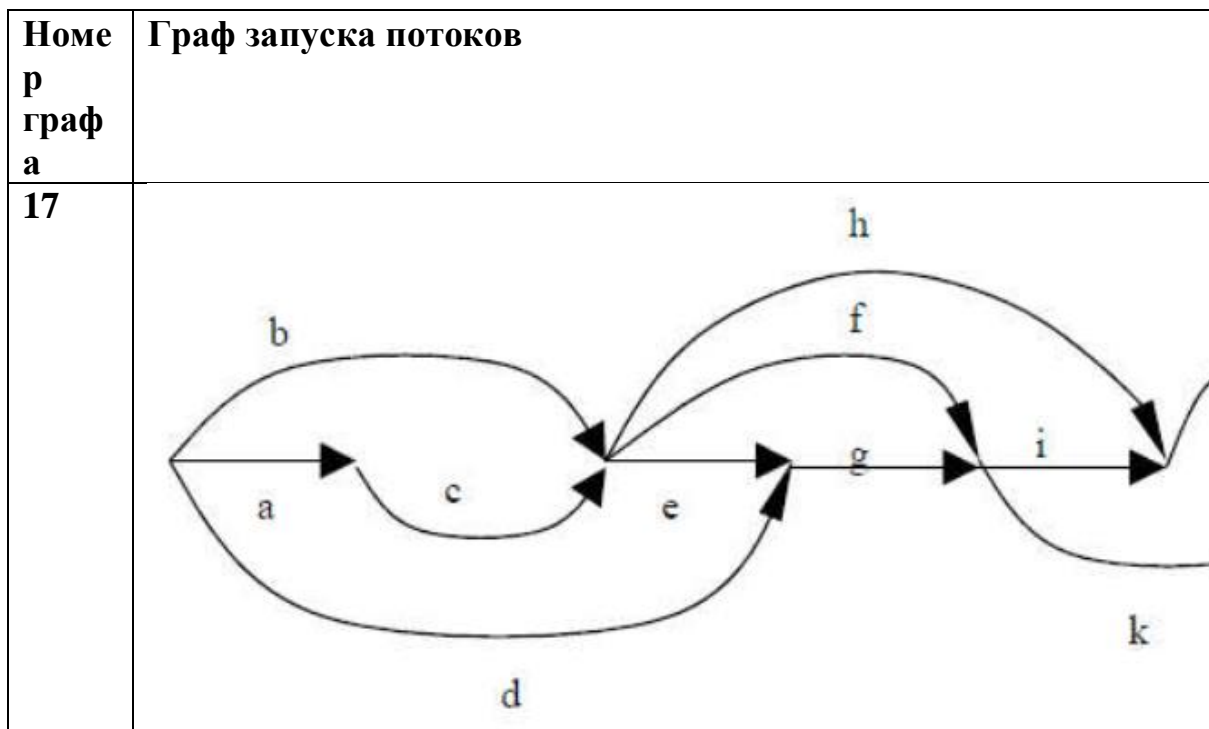
Санкт-Петербург 2024

### Цель работы:

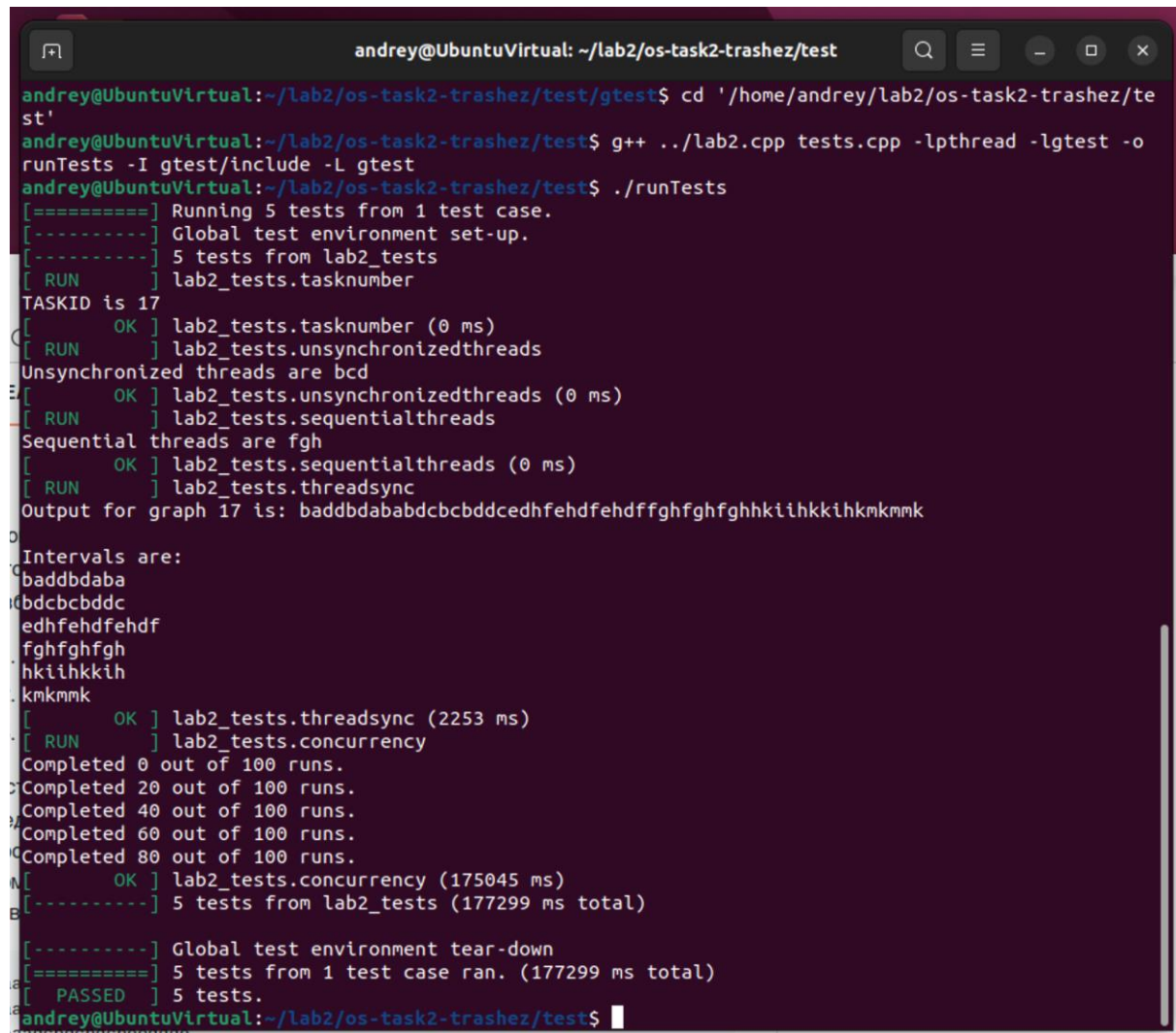
Знакомство с многопоточным программированием и методами синхронизации потоков средствами POSIX.

### Индивидуальное задание:

Номер варианта	Номер графа запуска потоков	Интервалы несинхронизированными потоками	Интервалы с чередованием потоков
13	17	bcd	fgh



## Результат выполнения работы



```
andrey@UbuntuVirtual: ~/lab2/os-task2-trashez/test
andrey@UbuntuVirtual:~/lab2/os-task2-trashez/test/gtest$ cd '/home/andrey/lab2/os-task2-trashez/test'
andrey@UbuntuVirtual:~/lab2/os-task2-trashez/test$ g++ ../lab2.cpp tests.cpp -lpthread -lgtest -o runTests -I gtest/include -L gtest
andrey@UbuntuVirtual:~/lab2/os-task2-trashez/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 17
[ OK       ] lab2_tests.tasknumber (0 ms)
[ RUN      ] lab2_tests.unsynchronizedthreads
Unsynchronized threads are bcd
[ OK       ] lab2_tests.unsynchronizedthreads (0 ms)
[ RUN      ] lab2_tests.sequentialthreads
Sequential threads are fgh
[ OK       ] lab2_tests.sequentialthreads (0 ms)
[ RUN      ] lab2_tests.threadsync
Output for graph 17 is: baddbdababdbcbddcedhfehdffghfghfghhkihkkihknkmmk
Intervals are:
baddbdaba
bdbcdbddc
edhfehdfehd
fghfghfgh
hkihkkih
knkmmk
[ OK       ] lab2_tests.threadsync (2253 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 (175045 ms)
[-----] 5 tests from lab2_tests (177299 ms total)

[-----] Global test environment tear-down
[=====] 5 tests from 1 test case ran. (177299 ms total)
[ PASSED   ] 5 tests.
andrey@UbuntuVirtual:~/lab2/os-task2-trashez/test$
```

Рисунок 1 – результат прохождения тестирования

## Исходный код программы с комментариями

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

#define NUMBER_OF_THREADS 11

// Идентификаторы потоков
pthread_t tid[NUMBER_OF_THREADS];
// Мьютекс для критической секции
pthread_mutex_t lock;
// Семафоры для синхронизированных потоков
sem_t semB, semC, semD, semE, semF, semG, semH, semI, semK;
```

```

int err;

unsigned int lab2_thread_graph_id() {
    return 17; // Номер графа запуска потоков
}

const char* lab2_unsynchronized_threads() {
    return "bcd"; // Потоки, выполняющиеся параллельно без синхронизации
}

const char* lab2_sequential_threads() {
    return "fgh"; // Потоки, выполняющиеся последовательно с
    синхронизацией
}

// Функции потоков (заготовки для потоков a, b, c, d, e, g, f, h, i, k и m)
void *thread_a(void *ptr);
void *thread_b(void *ptr);
void *thread_c(void *ptr);
void *thread_d(void *ptr);
void *thread_e(void *ptr);
void *thread_g(void *ptr);
void *thread_f(void *ptr);
void *thread_h(void *ptr);
void *thread_i(void *ptr);
void *thread_k(void *ptr);
void *thread_m(void *ptr);

// Функции потоков

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();
    }

    sem_post(&semB);

    pthread_exit(NULL);
    return ptr;
}

```

```

}

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

    sem_wait(&semB);
    sem_post(&semD);

    for (int i = 0; i < 3; ++i) {
        pthread_mutex_lock(&lock);
        std::cout << "b" << std::flush;
        pthread_mutex_unlock(&lock);
        computation();
    }

    sem_post(&semC);

    pthread_exit(NULL);
    return ptr;
}

void *thread_c(void *ptr) {
    for (int i = 0; i < 3; ++i) {
        pthread_mutex_lock(&lock);
        std::cout << "c" << std::flush;
        pthread_mutex_unlock(&lock);
        computation();
    }

    sem_wait(&semC);
    sem_post(&semD);

    pthread_exit(NULL);
    return ptr;
}

void *thread_d(void *ptr) {
    for (int i = 0; i < 3; ++i) {
        pthread_mutex_lock(&lock);
        std::cout << "d" << std::flush;

```

```

        pthread_mutex_unlock(&lock);
        computation();
    }

    sem_wait(&semD);
    pthread_create(&tid[2], NULL, thread_c, NULL);

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

    sem_wait(&semD);
    pthread_create(&tid[4], NULL, thread_e, NULL);
    pthread_create(&tid[6], NULL, thread_f, NULL);
    pthread_create(&tid[7], NULL, thread_h, NULL);

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

    sem_post(&semE);

    pthread_exit(NULL);
    return ptr;
}

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

    sem_wait(&semE);
    sem_post(&semH);

    pthread_exit(NULL);
    return ptr;
}

```

```
}
```

```
void *thread_g(void *ptr) {  
    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(&semH);  
    }  
  
    pthread_exit(NULL);  
    return ptr;  
}
```

```
void *thread_f(void *ptr) {  
    for (int i = 0; i < 3; ++i) {  
        pthread_mutex_lock(&lock);  
        std::cout << "f" << std::flush;  
        pthread_mutex_unlock(&lock);  
        computation();  
    }  
  
    for (int i = 0; i < 3; ++i) {  
        sem_wait(&semF);  
        pthread_mutex_lock(&lock);  
        std::cout << "f" << std::flush;  
        pthread_mutex_unlock(&lock);  
        computation();  
        sem_post(&semG);  
    }  
  
    pthread_exit(NULL);  
    return ptr;  
}
```

```
void *thread_h(void *ptr) {  
    for (int i = 0; i < 3; ++i) {  
        pthread_mutex_lock(&lock);  
        std::cout << "h" << std::flush;
```

```

    pthread_mutex_unlock(&lock);
    computation();
}

sem_wait(&semH);
sem_post(&semF);
pthread_create(&tid[5], NULL, thread_g, NULL);

for (int i = 0; i < 3; ++i) {
    sem_wait(&semH);
    pthread_mutex_lock(&lock);
    std::cout << "h" << std::flush;
    pthread_mutex_unlock(&lock);
    computation();
    sem_post(&semF);
}

pthread_create(&tid[8], NULL, thread_i, NULL);
pthread_create(&tid[9], NULL, thread_k, NULL);

for (int i = 0; i < 3; ++i) {
    pthread_mutex_lock(&lock);
    std::cout << "h" << std::flush;
    pthread_mutex_unlock(&lock);
    computation();
}

sem_post(&semI);

pthread_exit(NULL);
return ptr;
}

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

    sem_wait(&semI);
    sem_post(&semK);
}

```



```

    pthread_exit(NULL);
    return ptr;
}

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

    sem_wait(&semK);
    pthread_create(&tid[10], NULL, thread_m, NULL);

    for (int i = 0; i < 3; ++i) {
        pthread_mutex_lock(&lock);
        std::cout << "k" << std::flush;
        pthread_mutex_unlock(&lock);
        computation();
    }

    pthread_exit(NULL);
    return ptr;
}

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

    pthread_exit(NULL);
    return ptr;
}

int lab2_init()
{
    pthread_mutex_init(&lock, NULL);

    sem_init(&semB, 0, 0);

```

```

sem_init(&semC, 0, 0);
sem_init(&semD, 0, 0);
sem_init(&semE, 0, 0);
sem_init(&semF, 0, 0);
sem_init(&semG, 0, 0);

sem_init(&semH, 0, 0);
sem_init(&semI, 0, 0);
sem_init(&semK, 0, 0);

// Запуск первых потоков
pthread_create(&tid[0], NULL, thread_a, NULL);
pthread_create(&tid[1], NULL, thread_b, NULL);
pthread_create(&tid[3], NULL, thread_d, NULL);

// Ожидание завершения всех потоков
for (int i = 0; i < NUMBER_OF_THREADS; i++) {
    pthread_join(tid[i], NULL);
}

// Освобождение ресурсов
pthread_mutex_destroy(&lock);

sem_destroy(&semB);
sem_destroy(&semC);
sem_destroy(&semD);
sem_destroy(&semE);
sem_destroy(&semF);
sem_destroy(&semG);

sem_destroy(&semH);
sem_destroy(&semI);
sem_destroy(&semK);

std::cout << std::endl;
return 0;
}

```

## Выводы

В ходе работы мы познакомились с основами многопоточного программирования и практически применили методы синхронизации потоков, используя мьютексы и семафоры в соответствии со стандартом POSIX. Это позволило нам управлять порядком выполнения потоков и изучить влияние синхронизации на параллельное выполнение задач.