МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА № 43

ОТЧЁТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Старший преподаватель | |  |  | | | |  | | М. Д. Поляк |
| должность, уч. степень, звание | |  | подпись, дата | | | |  | | инициалы, фамилия |
| ОТЧЁТ О ЛАБОРАТОРНОЙ РАБОТЕ №3 | | | | | | | | | | |
| Разработка многопоточного приложения средствами POSIX в ОС Linux или Mac OS | | | | | | | | | | |
| по дисциплине: Операционные системы | | | | | | | | | | |
|  | | | | | | | | | | |
| РАБОТУ ВЫПОЛНИЛА | | | | | | | | | | |
| СТУДЕНТКА ГР. | 4931 | | |  | 09.04.2022 |  | | Е.Ю. Ильченко | | |
|  |  | | |  | подпись, дата |  | | инициалы, фамилия | | |
|  |  | | |  |  |  | |  | | |

Санкт-Петербург 2022

## Вариант 11

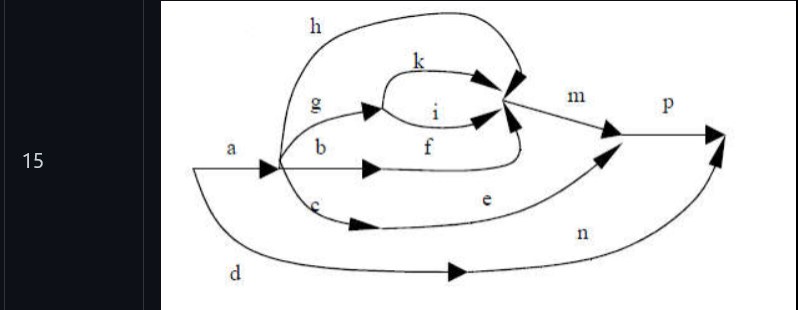
## Цель:

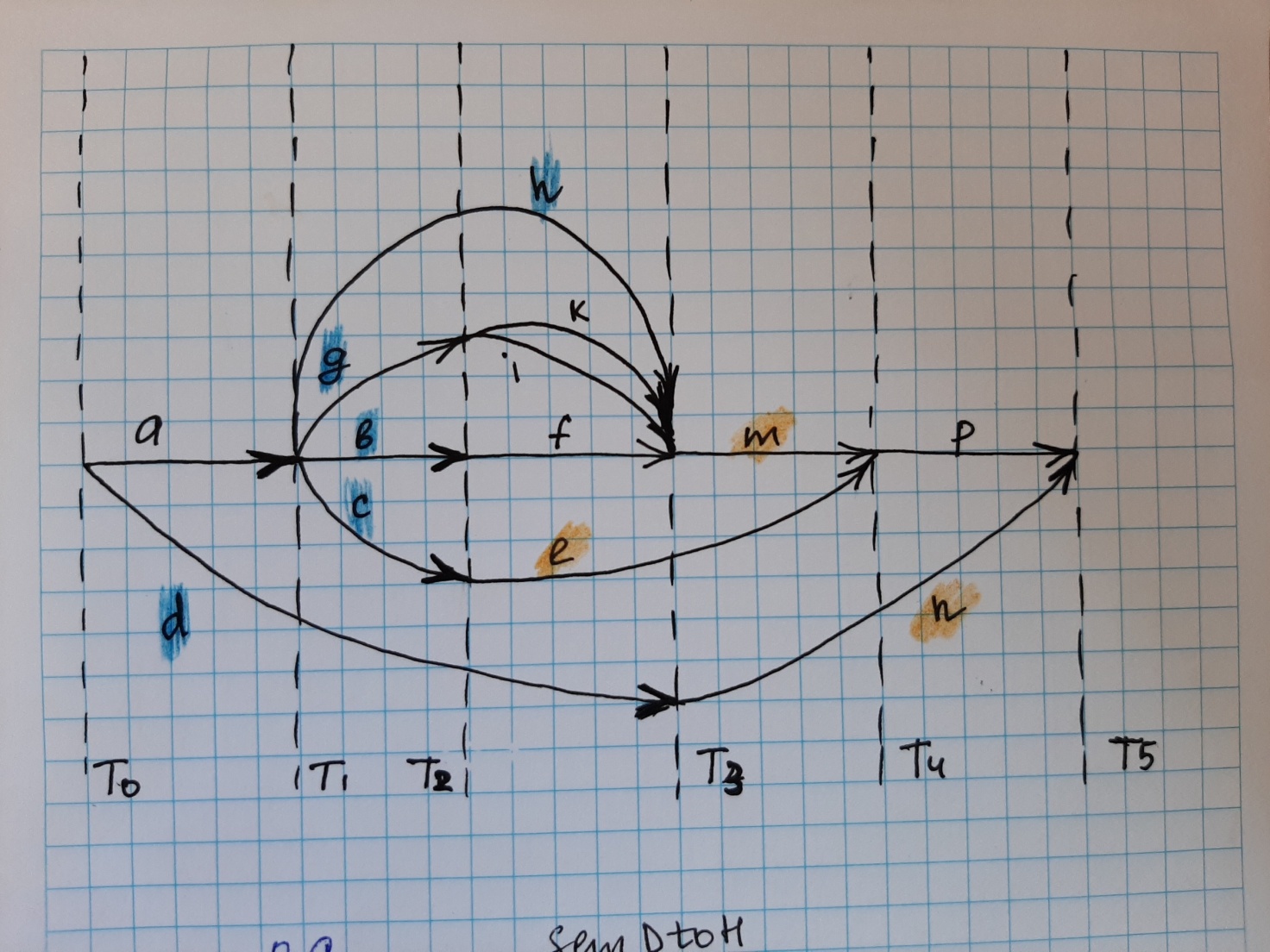
## Знакомство с многопоточным программированием и методами синхронизации потоков средствами POSIX.

#### Задание:

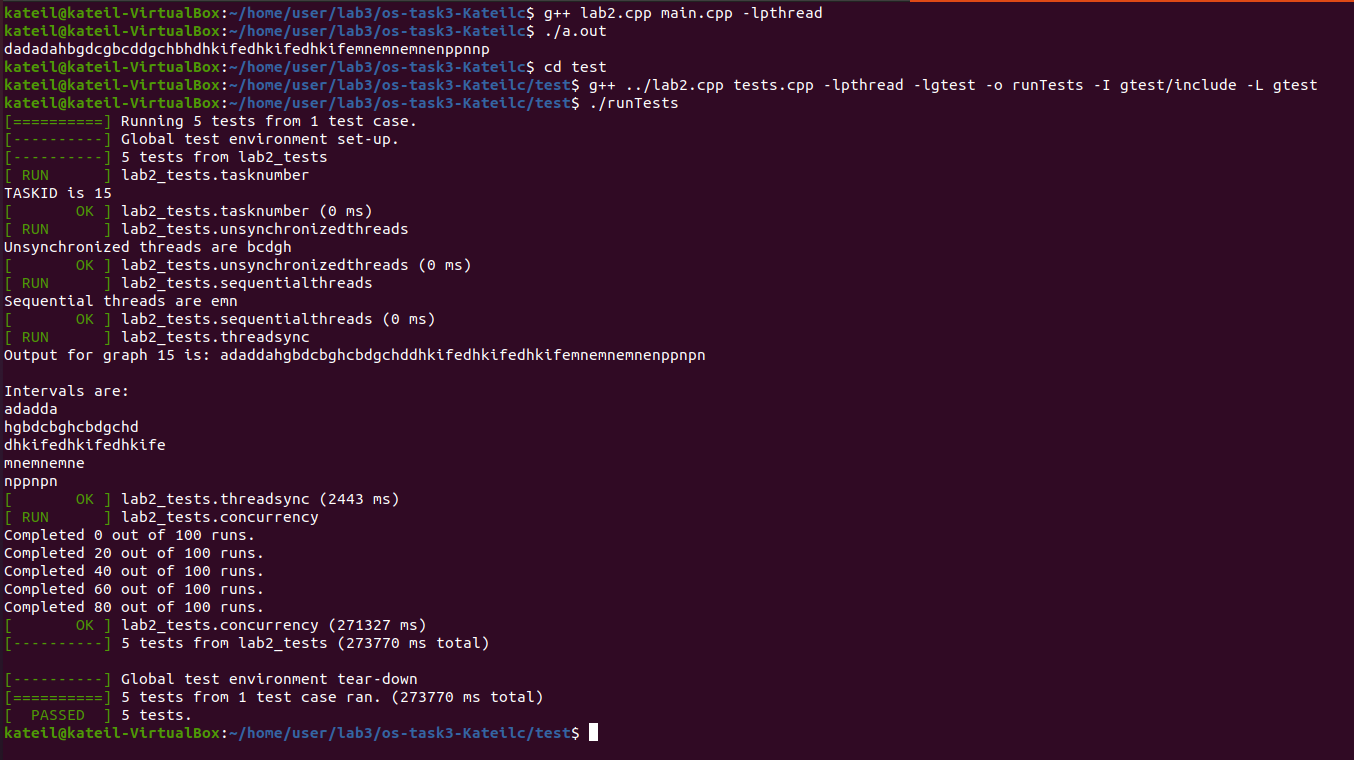
1. С помощью таблицы вариантов заданий выбрать граф запуска потоков в соответствии с номером варианта. Вершины графа являются точками запуска/завершения потоков, дугами обозначены сами потоки. Длину дуги следует интерпретировать как ориентировочное время выполнения потока. В процессе своей работы каждый поток должен в цикле выполнять два действия:
   1. выводить букву имени потока в консоль;
   2. 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.
2. 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](https://github.com/suai-os-2022/os-task3-Kateilc/blob/master/examples/README.md) [1](https://github.com/suai-os-2022/os-task3-Kateilc/blob/master/examples/README.md#%D0%9E%D1%80%D0%B3%D0%B0%D0%BD%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F-%D0%BF%D0%B0%D1%80%D0%B0%D0%BB%D0%BB%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D0%B3%D0%BE-%D0%B2%D1%8B%D0%BF%D0%BE%D0%BB%D0%BD%D0%B5%D0%BD%D0%B8%D1%8F-%D0%BF%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%B2-%D0%B) и [2](https://github.com/suai-os-2022/os-task3-Kateilc/blob/master/examples/README.md#%D0%98%D1%81%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5-%D0%BC%D1%8C%D1%8E%D1%82%D0%B5%D0%BA%D1%81%D0%B0-%D0%B4%D0%BB%D1%8F-%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D1%8B-%D1%81-%D0%BE%D0%B1%D1%89%D0%B8%D0%BC-%D1%80%D0%B5%D1%81%D1%83%D1%80%D1)). Вторая группа синхронизирована семафорами и потоки внутри группы выполняются в строго зафиксированном порядке: входящий в групу поток передает управление другому потоку после каждой итерации цикла (см. [пример 3](https://github.com/suai-os-2022/os-task3-Kateilc/blob/master/examples/README.md#%D0%98%D1%81%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5-%D1%81%D0%B5%D0%BC%D0%B0%D1%84%D0%BE%D1%80%D0%BE%D0%B2-%D0%B4%D0%BB%D1%8F-%D1%81%D0%B8%D0%BD%D1%85%D1%80%D0%BE%D0%BD%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D0%B8-%D0%BF%D0%BE%D1%82%D0%B) и [задачу производителя и потребителя](https://en.wikipedia.org/wiki/Producer%E2%80%93consumer_problem)). Таким образом потоки во второй группе выполняются в строгой очередности.
3. 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.). Для этого необходимо написать код в файле lab2.cpp:
   1. Функция unsigned int lab2\_thread\_graph\_id() должна возвращать номер графа запуска потоков, полученный из таблицы вариантов заданий.
   2. Функция const char\* lab2\_unsynchronized\_threads() должна возвращать строку, состоящую из букв потоков, выполняющихся параллельно без синхронизации (см. примеры в файлах [lab2.cpp](https://github.com/suai-os-2022/os-task3-Kateilc/blob/master/lab2.cpp) и [lab2\_ex.cpp](https://github.com/suai-os-2022/os-task3-Kateilc/blob/master/lab2_ex.cpp)).
   3. Функция const char\* lab2\_sequential\_threads() должна возвращать строку, состоящую из букв потоков, выполняющихся параллельно в строгой очередности друг за другом (см. примеры в файлах [lab2.cpp](https://github.com/suai-os-2022/os-task3-Kateilc/blob/master/lab2.cpp) и [lab2\_ex.cpp](https://github.com/suai-os-2022/os-task3-Kateilc/blob/master/lab2_ex.cpp)).
   4. Функция int lab2\_init() заменяет собой функцию main(). В ней необходимо реализовать запуск потоков, инициализацию вспомогательных переменных (мьютексов, семафоров и т.п.). Перед выходом из функции lab2\_init() необходимо убедиться, что все запущенные потоки завершились. Возвращаемое значение: 0 - работа функции завершилась успешно, любое другое числовое значение - при выполнении функции произошла критическая ошибка.
   5. Добавить любые другие необходимые для работы программы функции, переменные и подключаемые файлы.
   6. Создавать функцию main() не нужно. В проекте уже имеется готовая функция main(), изменять ее нельзя. Она выполняет единственное действие: вызывает функцию lab2\_init().
   7. Не следует изменять какие-либо файлы, кроме lab2.cpp. Также не следует создавать новые файлы и писать в них код, поскольку код из этих файлов не будет использоваться во время тестирования.

|  |  |  |  |
| --- | --- | --- | --- |
| Номер варианта | Номер графа запуска потоков | Интервалы с несинхронизированными потоками | Интервалы с чередованием потоков |
| 11 | 15 | bcdgh | emn |





**Результат выполнения работы программы:**

****

**Исходный код программы:**

#include "lab2.h"

#include <cstring>

#include <semaphore.h>

#define NUMBER\_OF\_THREADS 13

// thread identifiers

pthread\_t tid[NUMBER\_OF\_THREADS];

// critical section lock

pthread\_mutex\_t lock;

// semaphores for sequential threads

sem\_t semM, semE, semN, semK, semD, semH, semI, semF, semL, semT;

int err;

unsigned int lab2\_thread\_graph\_id()

{

return 15;

}

const char\* lab2\_unsynchronized\_threads()

{

return "bcdgh";;

}

const char\* lab2\_sequential\_threads()

{

return "emn";;

}

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\_f(void \*ptr);

void \*thread\_g(void \*ptr);

void \*thread\_h(void \*ptr);

void \*thread\_i(void \*ptr);

void \*thread\_k(void \*ptr);

void \*thread\_m(void \*ptr);

void \*thread\_n(void \*ptr);

void \*thread\_p(void \*ptr);

void \*thread\_d(void \*ptr)

{

//1-й интервал

for (int i = 0; i < 3; ++i) {

pthread\_mutex\_lock(&lock);

std::cout << "d" << std::flush;

pthread\_mutex\_unlock(&lock);

computation();

}

//ожидание завершения потока A

pthread\_join(tid[0], NULL);

//создание потока h

err = pthread\_create(&tid[2], NULL, thread\_h, NULL);

if (err != 0)

std::cerr << "Can't create thread. Error: " << strerror(err) <<std::endl;

//создание потока g

err = pthread\_create(&tid[3], NULL, thread\_g, NULL);

if (err != 0)

std::cerr << "Can't create thread. Error: " << strerror(err) << std::endl;

//создание потока b

err = pthread\_create(&tid[4], NULL, thread\_b, NULL);

if (err != 0)

std::cerr << "Can't create thread. Error: " << strerror(err) << std::endl;

//создание потока c

err = pthread\_create(&tid[5], NULL, thread\_c, NULL);

if (err != 0)

std::cerr << "Can't create thread. Error: " << strerror(err) << std::endl;

//2-й интервал

for (int i = 0; i < 3; ++i) {

pthread\_mutex\_lock(&lock);

std::cout << "d" << std::flush;

pthread\_mutex\_unlock(&lock);

computation();

}

//ожидание завершения потока g

pthread\_join(tid[3], NULL);

//ожидание завершения потока b

pthread\_join(tid[4], NULL);

//ожидание завершения потока c

pthread\_join(tid[5], NULL);

sem\_wait(&semL);

//k

err = pthread\_create(&tid[6], NULL, thread\_k, NULL);

if (err != 0)

std::cerr << "Can't create thread. Error: " << strerror(err) << std::endl;

//i

err = pthread\_create(&tid[7], NULL, thread\_i, NULL);

if (err != 0)

std::cerr << "Can't create thread. Error: " << strerror(err) << std::endl;

//f

err = pthread\_create(&tid[8], NULL, thread\_f, NULL);

if (err != 0)

std::cerr << "Can't create thread. Error: " << strerror(err) << std::endl;

//e

err = pthread\_create(&tid[9], NULL, thread\_e, NULL);

if (err != 0)

std::cerr << "Can't create thread. Error: " << strerror(err) << std::endl;

//3-й интервал

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(&semH);

}

//ожидание завершения потока e

pthread\_join(tid[9], NULL);

//ожидание завершения потока h

pthread\_join(tid[2], NULL);

return ptr;

}

void \*thread\_a(void \*ptr)

{

//1-й интервал

for (int i = 0; i < 3; ++i) {

pthread\_mutex\_lock(&lock);

std::cout << "a" << std::flush;

pthread\_mutex\_unlock(&lock);

computation();

}

return ptr;

}

void \*thread\_g(void \*ptr)

{

//2-й интервал

for (int i = 0; i < 3; ++i) {

pthread\_mutex\_lock(&lock);

std::cout << "g" << std::flush;

pthread\_mutex\_unlock(&lock);

computation();

}

return ptr;

}

void \*thread\_b(void \*ptr)

{

//2-й интервал

for (int i = 0; i < 3; ++i) {

pthread\_mutex\_lock(&lock);

std::cout << "b" << std::flush;

pthread\_mutex\_unlock(&lock);

computation();

}

return ptr;

}

void \*thread\_c(void \*ptr)

{

//2-й интервал

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\_h(void \*ptr)

{

//2-й интервал

for (int i = 0; i < 3; ++i) {

pthread\_mutex\_lock(&lock);

std::cout << "h" << std::flush;

pthread\_mutex\_unlock(&lock);

computation();

}

sem\_post(&semL);

//3-й интервал

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(&semK);

}

//ожидание завершения потока k

pthread\_join(tid[6], NULL);

return ptr;

}

void \*thread\_k(void \*ptr)

{

//3-й интервал

for (int i = 0; i < 3; ++i) {

sem\_wait(&semK);

pthread\_mutex\_lock(&lock);

std::cout << "k"<< std::flush;

pthread\_mutex\_unlock(&lock);

computation();

sem\_post(&semI);

}

//ожидание завершения потока i

pthread\_join(tid[7], NULL);

return ptr;

}

void \*thread\_i(void \*ptr)

{

//3-й интервал

for (int i = 0; i < 3; ++i) {

sem\_wait(&semI);

pthread\_mutex\_lock(&lock);

std::cout << "i"<< std::flush;

pthread\_mutex\_unlock(&lock);

computation();

sem\_post(&semF);

}

//ожидание завершения потока f

pthread\_join(tid[8], NULL);

return ptr;

}

void \*thread\_f(void \*ptr)

{

//3-й интервал

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(&semE);

}

return ptr;

}

void \*thread\_e(void \*ptr)

{

//3-й интервал

for (int i = 0; i < 3; ++i) {

sem\_wait(&semE);

pthread\_mutex\_lock(&lock);

std::cout << "e" << std::flush;

pthread\_mutex\_unlock(&lock);

computation();

sem\_post(&semD);

}

// start thread M

err = pthread\_create(&tid[10], NULL, thread\_m, NULL);

if (err != 0)

std::cerr << "Can't create thread. Error: " << strerror(err) << std::endl;

// start thread N

err = pthread\_create(&tid[11], NULL, thread\_n, NULL);

if (err != 0)

std::cerr << "Can't create thread. Error: " << strerror(err) << std::endl;

//4-й интервал

for (int i = 0; i < 3; ++i) {

sem\_wait(&semE);

pthread\_mutex\_lock(&lock);

std::cout << "e" << std::flush;

pthread\_mutex\_unlock(&lock);

computation();

sem\_post(&semM);

}

sem\_post(&semT);

//ожидание завершения потока N

pthread\_join(tid[11], NULL);

return ptr;

}

void \*thread\_m(void \*ptr)

{

//4-й интервал

for (int i = 0; i < 3; ++i) {

sem\_wait(&semM);

pthread\_mutex\_lock(&lock);

std::cout << "m" << std::flush;

pthread\_mutex\_unlock(&lock);

computation();

sem\_post(&semN);

}

return ptr;

}

void \*thread\_n(void \*ptr)

{

//4-й интервал

for (int i = 0; i < 3; ++i) {

sem\_wait(&semN);

pthread\_mutex\_lock(&lock);

std::cout << "n" << std::flush;

pthread\_mutex\_unlock(&lock);

computation();

sem\_post(&semE);

}

sem\_wait(&semT);

// start thread P

err = pthread\_create(&tid[12], NULL, thread\_p, NULL);

if (err != 0)

std::cerr << "Can't create thread. Error: " << strerror(err) << std::endl;

//5-й интервал

for (int i = 0; i < 3; ++i) {

pthread\_mutex\_lock(&lock);

std::cout << "n" << std::flush;

pthread\_mutex\_unlock(&lock);

computation();

}

//ожидание завершения потока p

pthread\_join(tid[12], NULL);

return ptr;

}

void \*thread\_p(void \*ptr)

{

//5-й интервал

for (int i = 0; i < 3; ++i) {

pthread\_mutex\_lock(&lock);

std::cout << "p" << std::flush;

pthread\_mutex\_unlock(&lock);

computation();

}

return ptr;

}

int lab2\_init()

{

// initilize mutex

if (pthread\_mutex\_init(&lock, NULL) != 0) {

std::cerr << "Mutex init failed" << 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(&semH, 0, 0) != 0 ) {

std::cerr << "Semaphore #1 init failed" << std::endl;

return 1;

}

if ( sem\_init(&semK, 0, 0) != 0 ) {

std::cerr << "Semaphore #2 init failed" << std::endl;

return 1;

}

if ( sem\_init(&semI, 0, 0) != 0 ) {

std::cerr << "Semaphore #3 init failed" << std::endl;

return 1;

}

if ( sem\_init(&semF, 0, 0) != 0 ) {

std::cerr << "Semaphore #4 init failed" << std::endl;

return 1;

}

if ( sem\_init(&semE, 0, 0) != 0 ) {

std::cerr << "Semaphore #5 init failed" << std::endl;

return 1;

}

if ( sem\_init(&semD, 0, 1) != 0 ) {

std::cerr << "Semaphore #6 init failed" << std::endl;

return 1;

}

if ( sem\_init(&semM, 0, 1) != 0 ) {

std::cerr << "Semaphore #7 init failed" << std::endl;

return 1;

}

if ( sem\_init(&semN, 0, 0) != 0 ) {

std::cerr << "Semaphore #8 init failed" << std::endl;

return 1;

}

if ( sem\_init(&semL, 0, 0) != 0 ) {

std::cerr << "Semaphore #9 init failed" << std::endl;

return 1;

}

if ( sem\_init(&semT, 0, 0) != 0 ) {

std::cerr << "Semaphore #10 init failed" << std::endl;

return 1;

}

// start the a thread

err = pthread\_create(&tid[0], NULL, thread\_a, NULL);

if (err != 0)

std::cerr << "Can't create thread. Error: " << strerror(err) << std::endl;

// start the d thread

err = pthread\_create(&tid[1], NULL, thread\_d, NULL);

if (err != 0)

std::cerr << "Can't create thread. Error: " << strerror(err) << std::endl;

// ... and wait for it to finish

pthread\_join(tid[1], NULL);

// free resources

pthread\_mutex\_destroy(&lock);

sem\_destroy(&semE);

sem\_destroy(&semN);

sem\_destroy(&semM);

std::cout << std::endl;

// success

return 0;

}

**Выводы:** В процессе выполнения лабораторной работы ознакомились с многопоточным программированием и методами синхронизации потоков средствами POSIX.