МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА №  43

ОТЧЁТ

ЗАЩИЩЁН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

### Старший преподаватель                                                      Поляк М.Д.

должность, уч. Степень, звание   подпись, дата                    инициалы, фамилия

ОТЧЁТ О ЛАБОРАТОРНОЙ РАБОТЕ №2.

Синхронизация потоков средствами POSIX.

по курсу: [Операционные системы](https://pro.guap.ru/inside/students/subjects/3154495)

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. 4136                                                                                Бобрович Н. С.

                                                                         подпись, дата                      инициалы, фамилия

Санкт-Петербург 2024

1. **Цель работы:**

Знакомство с многопоточным программированием и методами синхронизации потоков средствами POSIX.

1. **Индивидуальное задание:**
2. 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.

1. 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-2024/os-task2-OskolockKoli/blob/master/examples/README.md) [1](https://github.com/suai-os-2024/os-task2-OskolockKoli/blob/master/examples/README.md" \l "%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%B1%D0%B5%D0%B7-%D0%B8%D1%81%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F-%D1%81%D1%80%D0%B5%D0%B4%D1%81%D1%82%D0%B2-%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) и [2](https://github.com/suai-os-2024/os-task2-OskolockKoli/blob/master/examples/README.md" \l "%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%81%D0%BE%D0%BC)). 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-2024/os-task2-OskolockKoli/blob/master/examples/README.md" \l "%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%BE%D0%BA%D0%BE%D0%B2) 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.
2. 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(). Для этого необходимо написать код в файле lab2.cpp:

Функция unsigned int lab2\_thread\_graph\_id() должна возвращать номер графа запуска потоков, полученный из таблицы вариантов заданий.

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-2024/os-task2-OskolockKoli/blob/master/lab2.cpp) и [lab2\_ex.cpp](https://github.com/suai-os-2024/os-task2-OskolockKoli/blob/master/lab2_ex.cpp)).

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-2024/os-task2-OskolockKoli/blob/master/lab2.cpp) и [lab2\_ex.cpp](https://github.com/suai-os-2024/os-task2-OskolockKoli/blob/master/lab2_ex.cpp)).

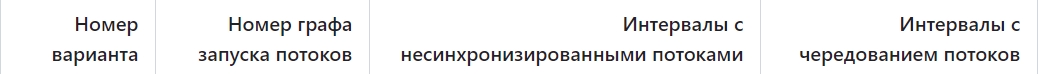
Функция int lab2\_init() заменяет собой функцию main(). В ней необходимо реализовать запуск потоков, инициализацию вспомогательных переменных (мьютексов, семафоров и т.п.). Перед выходом из функции lab2\_init() необходимо убедиться, что все запущенные потоки завершились. Возвращаемое значение: 0 - работа функции завершилась успешно, любое другое числовое значение - при выполнении функции произошла критическая ошибка.

Добавить любые другие необходимые для работы программы функции, переменные и подключаемые файлы.

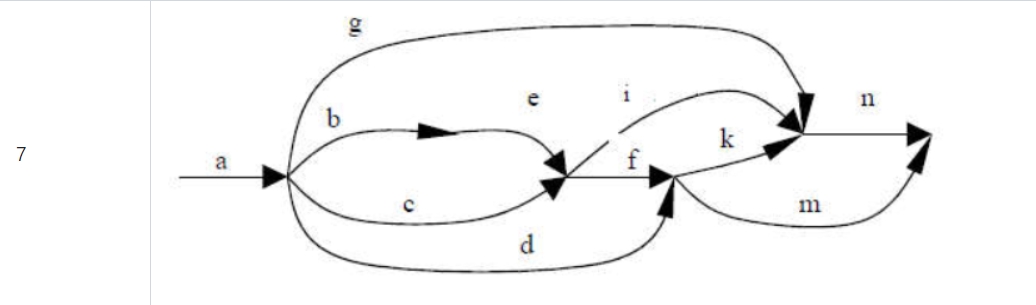
Создавать функцию main() не нужно. В проекте уже имеется готовая функция main(), изменять ее нельзя. Она выполняет единственное действие: вызывает функцию lab2\_init().

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.

1. Подготовить отчет о выполнении лабораторной работы и загрузить его под именем report.pdf в репозиторий. В случае использования системы компьютерной верстки LaTeX также загрузить исходный файл report.tex.
2. **Граф запуска потоков:**

****

**Скриншот 29-05-2024 134422**

****

1. **Результат выполнения работы:**

aaabcdgbcdgbcdggedcgcdcdgeegdifggfifiddigkmkgigimkmmnnmnm

1. **Исходный код программы с комментариями:**

#include "lab2.h"

#include <cstring>

#include <semaphore.h>

#include <pthread.h>

// Функция, возвращающая уникальный идентификатор графа потоков

unsigned int lab2\_thread\_graph\_id()

{

return 7;

}

// Функция, возвращающая строку с именами несинхронизированных потоков

const char\* lab2\_unsynchronized\_threads()

{

return "dfgi";

}

// Функция, возвращающая строку с именами последовательных потоков

const char\* lab2\_sequential\_threads()

{

return "bcdg";

}

// Мьютекс, используемый для защиты вывода данных в функции productAndCompute

pthread\_mutex\_t unsynchronizedLock;

// Функция, печатающая данные 3 раза с вычислениями между ними

void productAndCompute(const std::string& data) {

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

pthread\_mutex\_lock(&unsynchronizedLock); // Блокируем мьютекс, чтобы обеспечить исключительный доступ к выводу

std::cout << data << std::flush; // Выводим данные

pthread\_mutex\_unlock(&unsynchronizedLock); // Разблокируем мьютекс, чтобы другие потоки могли получить доступ

computation(); // Выполняем вычисления

}

}

// Функция, обеспечивающая последовательное выполнение потоков с выводом данных между ними

void handleSequentialThreads(sem\_t& cur, sem\_t& next, const std::string& data) {

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

sem\_wait(&cur); // Ждем своей очереди

std::cout << data << std::flush; // Выводим данные

sem\_post(&next); // Сигнализируем следующему потоку, что он может начать

computation(); // Выполняем вычисления

}

}

// Объявления функций для различных задач

void\* taskG(void\* ptr);

void\* taskA(void\* ptr);

void\* taskB(void\* ptr);

void\* taskC(void\* ptr);

void\* taskD(void\* ptr);

void\* taskN(void\* ptr);

// Семафоры, используемые для синхронизации выполнения задач

sem\_t taskBSSync, taskGSSync;

sem\_t taskGS, taskBS, taskES, taskCS, taskDS, taskIS, taskFS, taskMS;

// Реализация функции taskG

void\* taskG(void\* ptr)

{

// Создаем и запускаем потоки задач taskB, taskC и taskD

pthread\_t pthrB;

pthread\_create(&pthrB, NULL, taskB, NULL);

pthread\_t pthrC;

pthread\_create(&pthrC, NULL, taskC, NULL);

pthread\_t pthrD;

pthread\_create(&pthrD, NULL, taskD, NULL);

// Выполняем последовательные потоки, передавая данные "g" для вывода

handleSequentialThreads(taskGSSync, taskBSSync, "g");

sem\_post(&taskBS); // Сигнализируем потоку taskB, что он может начать

sem\_wait(&taskGS); // Ждем, пока потоки taskB, taskC и taskD завершатся

// Выполняем функцию productAndCompute три раза с данными "g"

productAndCompute("g");

sem\_post(&taskES);

sem\_post(&taskDS);

sem\_wait(&taskGS);

sem\_wait(&taskGS);

productAndCompute("g");

sem\_post(&taskIS);

sem\_post(&taskFS);

sem\_wait(&taskGS);

sem\_wait(&taskGS);

sem\_wait(&taskGS);

productAndCompute("g");

sem\_post(&taskMS);

sem\_wait(&taskGS);

sem\_wait(&taskGS);

sem\_wait(&taskGS);

// Создаем и запускаем поток задачи taskN

pthread\_t pthrN;

pthread\_create(&pthrN, NULL, taskN, NULL);

pthread\_join(pthrN, NULL); // Ждем, пока поток taskN завершится

return ptr;

}

void\* taskE(void\* ptr);

sem\_t tasbCSSync;

void\* taskB(void\* ptr)

{

handleSequentialThreads(taskBSSync, tasbCSSync, "b");

sem\_wait(&taskBS);

pthread\_t pthrE;

pthread\_create(&pthrE, NULL, taskE, NULL);

sem\_post(&taskGS);

sem\_post(&taskCS);

sem\_post(&taskDS);

return ptr;

}

sem\_t taskDSSync;

void\* taskC(void\* ptr)

{

handleSequentialThreads(tasbCSSync, taskDSSync, "c");

sem\_wait(&taskCS);

productAndCompute("c");

sem\_post(&taskES);

return ptr;

}

void\* taskD(void\* ptr)

{

handleSequentialThreads(taskDSSync, taskGSSync, "d");

sem\_wait(&taskDS);

productAndCompute("d");

sem\_post(&taskES);

sem\_post(&taskGS);

sem\_wait(&taskDS);

sem\_wait(&taskDS);

productAndCompute("d");

sem\_post(&taskFS);

sem\_post(&taskIS);

sem\_post(&taskGS);

return ptr;

}

void\* taskI(void\* ptr);

void\* taskE(void\* ptr)

{

productAndCompute("e");

sem\_wait(&taskES);

sem\_wait(&taskES);

sem\_wait(&taskES);

pthread\_t pthrI;

pthread\_create(&pthrI, NULL, taskI, NULL);

sem\_post(&taskDS);

sem\_post(&taskGS);

return ptr;

}

void\* taskK(void\* ptr);

void\* taskF(void\* ptr)

{

productAndCompute("f");

sem\_post(&taskGS);

sem\_post(&taskIS);

sem\_wait(&taskFS);

sem\_wait(&taskFS);

sem\_wait(&taskFS);

pthread\_t pthrK;

pthread\_create(&pthrK, NULL, taskK, NULL);

return ptr;

}

void\* taskI(void\* ptr)

{

pthread\_t pthrF;

pthread\_create(&pthrF, NULL, taskF, NULL);

productAndCompute("i");

sem\_post(&taskGS);

sem\_post(&taskFS);

sem\_wait(&taskIS);

sem\_wait(&taskIS);

sem\_wait(&taskIS);

productAndCompute("i");

sem\_post(&taskGS);

sem\_post(&taskMS);

return ptr;

}

void\* taskM(void\* ptr);

void\* taskK(void\* ptr)

{

pthread\_t pthrM;

pthread\_create(&pthrM, NULL, taskM, NULL);

productAndCompute("k");

sem\_post(&taskGS);

sem\_post(&taskMS);

return ptr;

}

sem\_t taskNS;

void\* taskM(void\* ptr)

{

productAndCompute("m");

sem\_post(&taskGS);

sem\_wait(&taskMS);

sem\_wait(&taskMS);

sem\_wait(&taskMS);

productAndCompute("m");

sem\_post(&taskNS);

return ptr;

}

void\* taskN(void\* ptr)

{

productAndCompute("n");

sem\_wait(&taskNS);

return ptr;

}

int lab2\_init()

{

pthread\_mutex\_init(&unsynchronizedLock, NULL);

sem\_init(&taskBS, 0, 0);

sem\_init(&taskGS, 0, 0);

sem\_init(&taskCS, 0, 0);

sem\_init(&taskDS, 0, 0);

sem\_init(&taskES, 0, 0);

sem\_init(&taskFS, 0, 0);

sem\_init(&taskMS, 0, 0);

sem\_init(&taskIS, 0, 0);

sem\_init(&taskNS, 0, 0);

sem\_init(&taskBSSync, 0, 1);

sem\_init(&tasbCSSync, 0, 0);

sem\_init(&taskGSSync, 0, 0);

sem\_init(&taskDSSync, 0, 0);

pthread\_t pthrA;

pthread\_create(&pthrA, NULL, taskA, NULL);

pthread\_join(pthrA, NULL);

pthread\_mutex\_destroy(&unsynchronizedLock);

sem\_destroy(&taskGS);

sem\_destroy(&taskCS);

sem\_destroy(&taskDS);

sem\_destroy(&taskES);

sem\_destroy(&taskFS);

sem\_destroy(&taskMS);

sem\_destroy(&taskIS);

sem\_destroy(&taskNS);

sem\_destroy(&taskBSSync);

sem\_destroy(&tasbCSSync);

sem\_destroy(&taskGSSync);

sem\_destroy(&taskDSSync);

return 0;

}

1. **Выводы:**

Познакомился с многопоточным программированием и методами синхронизации потоков средствами POSIX.