ГУАП

КАФЕДРА № 43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Стар преп |  |  |  | М.Д. Поляк |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №1 |
| «Разработка многопоточного приложения средствами POSIX в ОС Linux или Mac OS» |
| по курсу: ОПЕРАЦИОННЫЕ СИСТЕМЫ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4831 |  | 10.03.2021 |  | К.А.Корнющенков |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2021

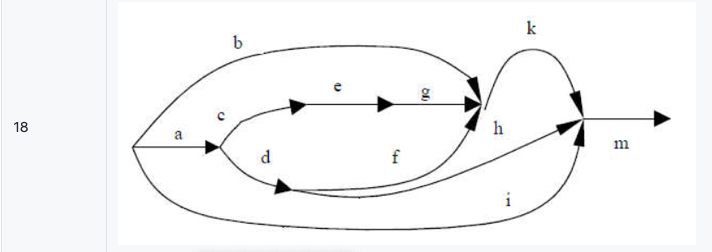
1. **Цель работы**

Знакомство с многопоточным программированием и методами синхронизации потоков средствами POSIX.

1. **Задание на лабораторную работу**

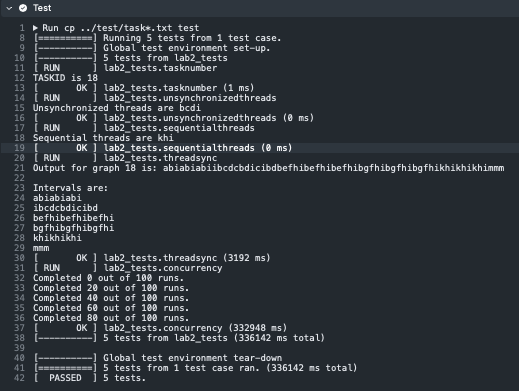
****

1. **Граф запуска потоков**

****

1. **Результат выполнения работы**

****

****

1. **Исходный код программы с комментариями**

#include "lab2.h"

#include <cstring>

#include <fcntl.h>

#include <unistd.h>

#include <iostream>

#include <pthread.h>

#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 \*step1, \*semA1, \*semB1, \*semI1;

sem\_t \*step2;

sem\_t \*step3, \*semB3, \*semE3, \*semF3, \*semH3, \*semI3;

sem\_t \*step4, \*semB4, \*semG4, \*semF4, \*semH4, \*semI4;

sem\_t \*step5, \*semK5, \*semH5, \*semI5;

int err;

int flagFor2 = 12;

void flagFor2Func() {

flagFor2 --;

if (flagFor2 == 0) {

sem\_post(step2);

sem\_post(step2);

sem\_post(step2);

sem\_post(step2);

sem\_post(step2);

flagFor2 = 12;

}

}

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

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

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

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

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

unsigned int lab2\_thread\_graph\_id() { return 18; }

const char\* lab2\_unsynchronized\_threads() { return "bcdi"; }

const char\* lab2\_sequential\_threads() { return "khi"; }

//MARK: - thread\_m

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

//Шаг 6

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

pthread\_mutex\_lock(&lock);

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

pthread\_mutex\_unlock(&lock);

computation();

}

return ptr;

}

//MARK: - thread\_a

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

//Шаг 1

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

sem\_wait(semA1);

pthread\_mutex\_lock(&lock);

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

pthread\_mutex\_unlock(&lock);

computation();

sem\_post(semB1);

}

//Шаг 2

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

if (err != 0)

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

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

if (err != 0)

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

return ptr;

}

//MARK: - thread\_k

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

//Шаг 5

sem\_wait(step4);

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

sem\_wait(semK5);

pthread\_mutex\_lock(&lock);

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

pthread\_mutex\_unlock(&lock);

computation();

sem\_post(semH5);

}

return ptr;

}

//MARK: - thread\_g

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

//Шаг 4

sem\_wait(step3);

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

sem\_wait(semG4);

pthread\_mutex\_lock(&lock);

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

pthread\_mutex\_unlock(&lock);

computation();

sem\_post(semF4);

}

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

if (err != 0)

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

return ptr;

}

//MARK: - thread\_e

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

//Шаг 3

sem\_wait(step2);

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

sem\_wait(semE3);

pthread\_mutex\_lock(&lock);

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

pthread\_mutex\_unlock(&lock);

computation();

sem\_post(semF3);

}

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

if (err != 0)

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

return ptr;

}

//MARK: - thread\_c

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

//Шаг 2

sem\_wait(step1);

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

pthread\_mutex\_lock(&lock);

std::cout << "c" << std::flush;

flagFor2Func();

pthread\_mutex\_unlock(&lock);

computation();

}

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

if (err != 0)

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

return ptr;

}

//MARK: - thread\_d

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

//Шаг 2

sem\_wait(step1);

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

pthread\_mutex\_lock(&lock);

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

flagFor2Func();

pthread\_mutex\_unlock(&lock);

computation();

}

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

if (err != 0)

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

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

if (err != 0)

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

return ptr;

}

//MARK: - thread\_f

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

//Шаг 3

sem\_wait(step2);

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

sem\_wait(semF3);

pthread\_mutex\_lock(&lock);

std::cout << "f" << std::flush;

pthread\_mutex\_unlock(&lock);

computation();

sem\_post(semH3);

}

//Шаг 4

sem\_wait(step3);

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

sem\_wait(semF4);

pthread\_mutex\_lock(&lock);

std::cout << "f" << std::flush;

pthread\_mutex\_unlock(&lock);

computation();

sem\_post(semH4);

}

return ptr;

}

//MARK: - thread\_h

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

//Шаг 3

sem\_wait(step2);

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

sem\_wait(semH3);

pthread\_mutex\_lock(&lock);

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

pthread\_mutex\_unlock(&lock);

computation();

sem\_post(semI3);

}

//Шаг 4

sem\_wait(step3);

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

sem\_wait(semH4);

pthread\_mutex\_lock(&lock);

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

pthread\_mutex\_unlock(&lock);

computation();

sem\_post(semI4);

}

//Шаг 5

sem\_wait(step4);

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

sem\_wait(semH5);

pthread\_mutex\_lock(&lock);

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

pthread\_mutex\_unlock(&lock);

computation();

sem\_post(semI5);

}

return ptr;

}

//MARK: - thread\_b

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

//Шаг 1

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

sem\_wait(semB1);

pthread\_mutex\_lock(&lock);

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

pthread\_mutex\_unlock(&lock);

computation();

sem\_post(semI1);

}

//Шаг 2

sem\_wait(step1);

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

pthread\_mutex\_lock(&lock);

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

flagFor2Func();

pthread\_mutex\_unlock(&lock);

computation();

}

//Шаг 3

sem\_wait(step2);

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

sem\_wait(semB3);

pthread\_mutex\_lock(&lock);

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

pthread\_mutex\_unlock(&lock);

computation();

sem\_post(semE3);

}

//Шаг 4

sem\_wait(step3);

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

sem\_wait(semB4);

pthread\_mutex\_lock(&lock);

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

pthread\_mutex\_unlock(&lock);

computation();

sem\_post(semG4);

}

return ptr;

}

//MARK: - thread\_i

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

//Шаг 1

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

sem\_wait(semI1);

pthread\_mutex\_lock(&lock);

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

pthread\_mutex\_unlock(&lock);

computation();

sem\_post(semA1);

}

//сообщаем, что закончили шаг1 и можно начинать шаг 2

sem\_post(step1);

sem\_post(step1);

sem\_post(step1);

sem\_post(step1);

//Шаг 2

sem\_wait(step1);

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

pthread\_mutex\_lock(&lock);

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

flagFor2Func();

pthread\_mutex\_unlock(&lock);

computation();

}

//Шаг 3

sem\_wait(step2);

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

sem\_wait(semI3);

pthread\_mutex\_lock(&lock);

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

pthread\_mutex\_unlock(&lock);

computation();

sem\_post(semB3);

}

sem\_post(step3);

sem\_post(step3);

sem\_post(step3);

sem\_post(step3);

sem\_post(step3);

//Шаг 4

sem\_wait(step3);

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

sem\_wait(semI4);

pthread\_mutex\_lock(&lock);

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

pthread\_mutex\_unlock(&lock);

computation();

sem\_post(semB4);

}

sem\_post(step4);

sem\_post(step4);

sem\_post(step4);

//Шаг 5

sem\_wait(step4);

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

sem\_wait(semI5);

pthread\_mutex\_lock(&lock);

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

pthread\_mutex\_unlock(&lock);

computation();

sem\_post(semK5);

}

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

if (err != 0)

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

return ptr;

}

void clearData() {

// free resources

sem\_unlink("step1");

sem\_unlink("semA1");

sem\_unlink("semB1");

sem\_unlink("semI1");

sem\_unlink("step2");

sem\_unlink("step3");

sem\_unlink("semB3");

sem\_unlink("semE3");

sem\_unlink("semF3");

sem\_unlink("semH3");

sem\_unlink("semI3");

sem\_unlink("step4");

sem\_unlink("semB4");

sem\_unlink("semG4");

sem\_unlink("semF4");

sem\_unlink("semH4");

sem\_unlink("semI4");

sem\_unlink("step5");

sem\_unlink("semK5");

sem\_unlink("semH5");

sem\_unlink("semI5");

}

//MARK: - lab2\_init

int lab2\_init() {

clearData();

// initilize mutex

if (pthread\_mutex\_init(&lock, NULL) != 0) { std::cerr << "Mutex init failed" << std::endl; return 1; }

//шаг 1

step1 = sem\_open("step1", O\_CREAT, 0, 0);

if (step1 == SEM\_FAILED) { std::cerr << "Semaphore #step1 init failed" << std::endl; return 1; }

semA1 = sem\_open("semA1", O\_CREAT, 0, 1);

if (semA1 == SEM\_FAILED) { std::cerr << "Semaphore #semA1 init failed" << std::endl; return 1; }

semB1 = sem\_open("semB1", O\_CREAT, 0, 0);

if (semB1 == SEM\_FAILED) { std::cerr << "Semaphore #semB1 init failed" << std::endl; return 1; }

semI1 = sem\_open("semI1", O\_CREAT, 0, 0);

if (semI1 == SEM\_FAILED) { std::cerr << "Semaphore #semI1 init failed" << std::endl; return 1; }

//шаг 2

step2 = sem\_open("step2", O\_CREAT, 0, 0);

if (step2 == SEM\_FAILED) { std::cerr << "Semaphore #step2 init failed" << std::endl; return 1; }

//шаг 3

step3 = sem\_open("step3", O\_CREAT, 0, 0);

if (step3 == SEM\_FAILED) { std::cerr << "Semaphore #step3 init failed" << std::endl; return 1; }

semB3 = sem\_open("semB3", O\_CREAT, 0, 1);

if (semB3 == SEM\_FAILED) { std::cerr << "Semaphore #semB3 init failed" << std::endl; return 1; }

semE3 = sem\_open("semE3", O\_CREAT, 0, 0);

if (semE3 == SEM\_FAILED) { std::cerr << "Semaphore #semE3 init failed" << std::endl; return 1; }

semF3 = sem\_open("semF3", O\_CREAT, 0, 0);

if (semF3 == SEM\_FAILED) { std::cerr << "Semaphore #semF3 init failed" << std::endl; return 1; }

semH3 = sem\_open("semH3", O\_CREAT, 0, 0);

if (semH3 == SEM\_FAILED) { std::cerr << "Semaphore #semH3 init failed" << std::endl; return 1; }

semI3 = sem\_open("semI3", O\_CREAT, 0, 0);

if (semI3 == SEM\_FAILED) { std::cerr << "Semaphore #semI3 init failed" << std::endl; return 1; }

//шаг 4

step4 = sem\_open("step4", O\_CREAT, 0, 0);

if (step4 == SEM\_FAILED) { std::cerr << "Semaphore #step4 init failed" << std::endl; return 1;}

semB4 = sem\_open("semB4", O\_CREAT, 0, 1);

if (semB4 == SEM\_FAILED) { std::cerr << "Semaphore #semB4 init failed" << std::endl; return 1;}

semG4 = sem\_open("semG4", O\_CREAT, 0, 0);

if (semG4 == SEM\_FAILED) { std::cerr << "Semaphore #semG4 init failed" << std::endl; return 1;}

semF4 = sem\_open("semF4", O\_CREAT, 0, 0);

if (semF4 == SEM\_FAILED) { std::cerr << "Semaphore #semF4 init failed" << std::endl; return 1;}

semH4 = sem\_open("semH4", O\_CREAT, 0, 0);

if (semH4 == SEM\_FAILED) { std::cerr << "Semaphore #semH4 init failed" << std::endl; return 1;}

semI4 = sem\_open("semI4", O\_CREAT, 0, 0);

if (semI4 == SEM\_FAILED) { std::cerr << "Semaphore #semI4 init failed" << std::endl; return 1;}

//шаг 5

step5 = sem\_open("step5", O\_CREAT, 0, 1);

if (step5 == SEM\_FAILED) { std::cerr << "Semaphore #step5 init failed" << std::endl; return 1;}

semK5 = sem\_open("semK5", O\_CREAT, 0, 1);

if (semK5 == SEM\_FAILED) { std::cerr << "Semaphore #semK5 init failed" << std::endl; return 1;}

semH5 = sem\_open("semH5", O\_CREAT, 0, 0);

if (semH5 == SEM\_FAILED) { std::cerr << "Semaphore #semH5 init failed" << std::endl; return 1;}

semI5 = sem\_open("semI5", O\_CREAT, 0, 0);

if (semI5 == SEM\_FAILED) { std::cerr << "Semaphore #semI5 init failed" << std::endl; return 1;}

// start the first thread

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

if (err != 0){std::cerr << "Can't create thread. Error: " << strerror(err) << std::endl;}

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

if (err != 0){ std::cerr << "Can't create thread. Error: " << strerror(err) << std::endl;}

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

if (err != 0) {std::cerr << "Can't create thread. Error: " << strerror(err) << std::endl;}

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

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

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

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

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

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

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

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

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

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

pthread\_join(tid[10], NULL);

pthread\_mutex\_destroy(&lock);

clearData();

std::cout << std::endl;

// success

return 0;

}

1. **Вывод**

В ходе выполнения лабораторной работы были получены навыки работы с многопоточным программирование и методами синхронизации потоков средствами POSIX.