МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Санкт-Петербургский государственный университет
аэрокосмического приборостроения»

КАФЕДРА 43

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

Ст. преподаватель		М.Д. Поляк
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. № 4134К		Самарин Д.В
	подпись, дата	инициалы, фамилия

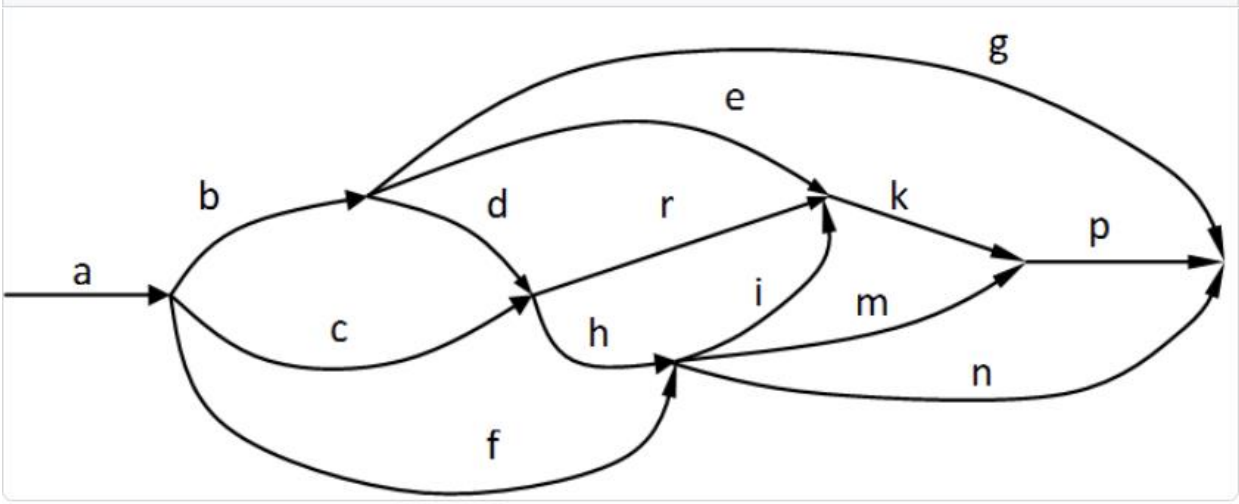
Санкт-Петербург 2024

Цель работы:

Знакомство с многопоточным программированием и методами синхронизации потоков средствами POSIX.

Индивидуальное задание:

Номер варианта	Номер графа запуска потоков	Интервалы с несинхронизированными потоками	Интервалы с чередованием потоков
16	20	bcf	gkmn



Результат выполнения работы

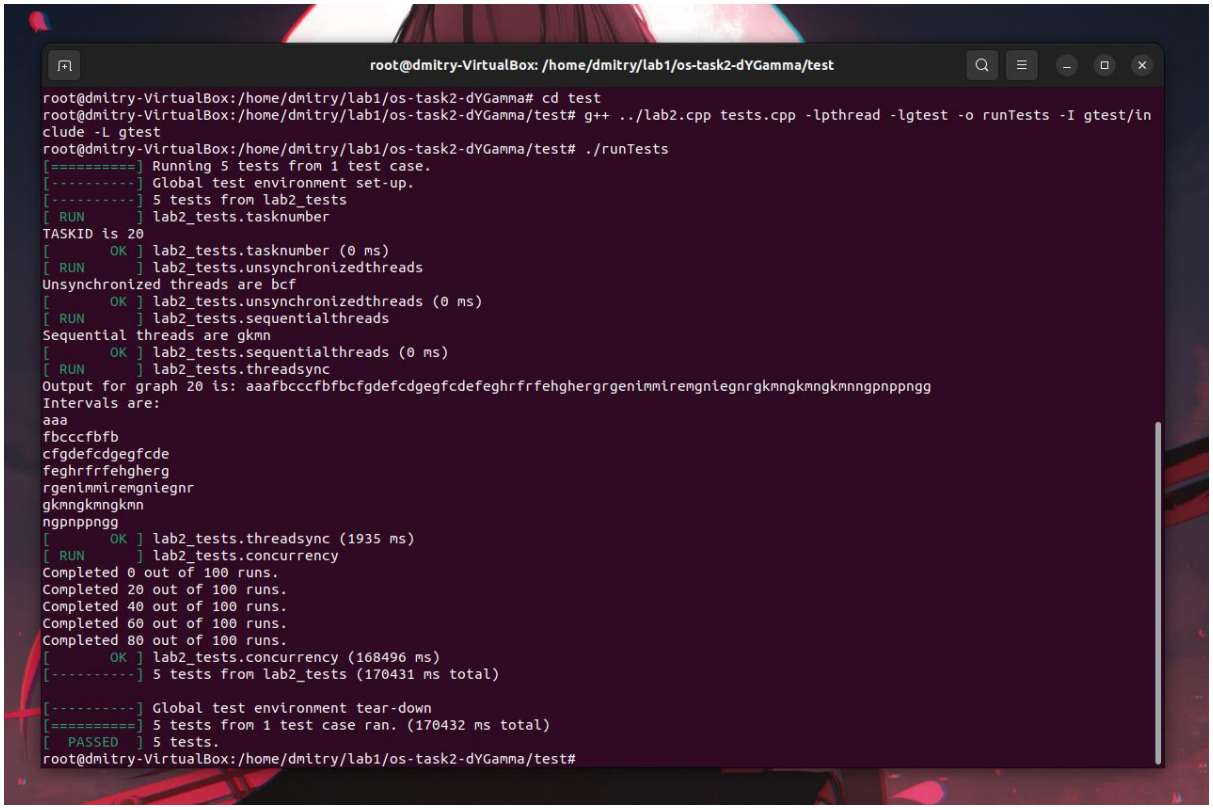


Рисунок 1 – результат прохождения тестирования

Исходный код программы с комментариями

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

pthread_mutex_t mutex;
sem_t threadSemSyncM, threadSemSyncK, threadSemSyncG, threadSemSyncN;
sem_t threadSemB, threadSemC, threadSemF, threadSemD, threadSemE,
threadSemI,
    threadSemM, threadSemN, threadSemR, threadSemH, threadSemP, threadSemG;

unsigned int lab2_thread_graph_id() {
    return 20;
}

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

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

pthread_t executeJob(void *(*jobFunc)(void *)) {
```

```
    pthread_t jobId;
    pthread_create(&jobId, NULL, jobFunc, NULL);
    return jobId;
}

void printWithMutex(const std::string &data) {
    for (int i = 0; i < 3; ++i) {
        pthread_mutex_lock(&mutex);
        std::cout << data << std::flush;
        pthread_mutex_unlock(&mutex);
        computation();
    }
}

void printSem(const std::string &data, sem_t &actualSem, sem_t
&actualFolowing) {
    for (int i = 0; i < 3; ++i) {
        sem_wait(&actualSem);
        std::cout << data << std::flush;
        sem_post(&actualFolowing);
        computation();
    }
}

void *jobF(void *p);
void *jobA(void *p) {
    printWithMutex("a");
    pthread_join(executeJob(jobF), NULL);
    return p;
}

void *jobG(void *p);
void *jobB(void *p) {
    printWithMutex("b");

    sem_post(&threadSemC);
    sem_post(&threadSemF);
    sem_wait(&threadSemB); //wait f
    sem_wait(&threadSemB); //wait c

    executeJob(jobG);
    return p;
}

void *jobH(void *p);
void *jobC(void *p) {
    printWithMutex("c");

    sem_post(&threadSemB);
    sem_post(&threadSemF);
```

```
    sem_wait(&threadSemC); //wait f
    sem_wait(&threadSemC); //wait b

    printWithMutex("c");
    sem_post(&threadSemF);
    sem_post(&threadSemD);
    sem_post(&threadSemE);
    sem_post(&threadSemG);
    sem_wait(&threadSemC); //wait g
    sem_wait(&threadSemC); //wait e
    sem_wait(&threadSemC); //wait d
    sem_wait(&threadSemC); //wait f
    executeJob(jobH);
    return p;
}

void *jobD(void *p) {
    printWithMutex("d");

    sem_post(&threadSemG);
    sem_post(&threadSemE);
    sem_post(&threadSemC);
    sem_post(&threadSemF);
    return p;
}

void *jobE(void *p) {
    printWithMutex("e");

    sem_post(&threadSemG);
    sem_post(&threadSemD);
    sem_post(&threadSemC);
    sem_post(&threadSemF);

    sem_wait(&threadSemE); //wait g
    sem_wait(&threadSemE); //wait d
    sem_wait(&threadSemE); //wait c
    sem_wait(&threadSemE); //wait f

    printWithMutex("e");

    sem_post(&threadSemG);
    sem_post(&threadSemR);
    sem_post(&threadSemH);
    sem_post(&threadSemF);

    sem_wait(&threadSemE); //wait g
    sem_wait(&threadSemE); //wait r
    sem_wait(&threadSemE); //wait h
    sem_wait(&threadSemE); //wait f
```

```
    printWithMutex("e");

    sem_post(&threadSemG);
    sem_post(&threadSemR);
    sem_post(&threadSemI);
    sem_post(&threadSemM);
    sem_post(&threadSemN);
    return p;
}

void *jobG(void *p) {
    executeJob(jobE);
    executeJob(jobD);

    printWithMutex("g");

    sem_post(&threadSemE);
    sem_post(&threadSemD);
    sem_post(&threadSemC);
    sem_post(&threadSemF);

    sem_wait(&threadSemG); //wait e
    sem_wait(&threadSemG); //wait d
    sem_wait(&threadSemG); //wait c
    sem_wait(&threadSemG); //wait f

    printWithMutex("g");

    sem_post(&threadSemE);
    sem_post(&threadSemR);
    sem_post(&threadSemH); // not used
    sem_post(&threadSemF);

    sem_wait(&threadSemG); //wait e
    sem_wait(&threadSemG); //wait r
    sem_wait(&threadSemG); //wait h
    sem_wait(&threadSemG); //wait f

    printWithMutex("g");

    sem_post(&threadSemR);
    sem_post(&threadSemI);
    sem_post(&threadSemM);
    sem_post(&threadSemN);

    sem_wait(&threadSemG); //wait e
    sem_wait(&threadSemG); //wait r
    sem_wait(&threadSemG); //wait i
    sem_wait(&threadSemG); //wait m
```

```
    sem_wait(&threadSemG); //wait n

    printSem("g", threadSemSyncG, threadSemSyncK);

    sem_wait(&threadSemG);

    printWithMutex("g");

    sem_post(&threadSemN);
    return p;
}

void *jobN(void *p);
void *jobF(void *p) {
    executeJob(jobB);
    executeJob(jobC);

    printWithMutex("f");

    sem_post(&threadSemB);
    sem_post(&threadSemC);
    sem_wait(&threadSemF); //wait c
    sem_wait(&threadSemF); //wait b

    printWithMutex("f");

    sem_post(&threadSemC);
    sem_post(&threadSemD);
    sem_post(&threadSemE);
    sem_post(&threadSemG);

    sem_wait(&threadSemF); //wait g
    sem_wait(&threadSemF); //wait e
    sem_wait(&threadSemF); //wait d
    sem_wait(&threadSemF); //wait c

    printWithMutex("f");

    sem_post(&threadSemH);
    sem_post(&threadSemR);
    sem_post(&threadSemE);
    sem_post(&threadSemG);

    sem_wait(&threadSemF); //wait g
    sem_wait(&threadSemF); //wait e
    sem_wait(&threadSemF); //wait r
    sem_wait(&threadSemF); //wait h

    pthread_join(executeJob(jobN), NULL);
    return p;
```

```
}

void *jobK(void *p) {
    printSem("k", threadSemSyncK, threadSemSyncM);
    return p;
}

void *jobI(void *p) {
    printWithMutex("i");

    sem_post(&threadSemG);
    sem_post(&threadSemR);
    sem_post(&threadSemN);
    sem_post(&threadSemM);

    sem_wait(&threadSemI); //wait g
    sem_wait(&threadSemI); //wait e
    sem_wait(&threadSemI); //wait r
    sem_wait(&threadSemI); //wait m
    sem_wait(&threadSemI); //wait n

    executeJob(jobK);
    return p;
}

void *jobR(void *p) {
    printWithMutex("r");

    sem_post(&threadSemG);
    sem_post(&threadSemE);
    sem_post(&threadSemF);

    sem_wait(&threadSemR); //wait g
    sem_wait(&threadSemR); //wait e
    sem_wait(&threadSemR); //wait h
    sem_wait(&threadSemR); //wait f

    printWithMutex("r");

    sem_post(&threadSemG);
    sem_post(&threadSemI);
    sem_post(&threadSemM);
    sem_post(&threadSemN);

    return p;
}


void *jobH(void *p) {
    executeJob(jobR);
```

```
    printWithMutex("h");
    sem_post(&threadSemF);
    sem_post(&threadSemR);
    sem_post(&threadSemE);
    sem_post(&threadSemG);
    return p;
}

void *jobP(void *p);
void *jobM(void *p) {
    printWithMutex("m");

    sem_post(&threadSemG);
    sem_post(&threadSemE);
    sem_post(&threadSemR);
    sem_post(&threadSemN);
    sem_post(&threadSemI);

    sem_wait(&threadSemM); //wait g
    sem_wait(&threadSemM); //wait e
    sem_wait(&threadSemM); //wait r
    sem_wait(&threadSemM); //wait i
    sem_wait(&threadSemM); //wait n

    printSem("m", threadSemSyncM, threadSemSyncN);
    sem_wait(&threadSemM); //wait n
    executeJob(jobP);
    return p;
}

void *jobN(void *p) {
    executeJob(jobI);
    executeJob(jobM);

    printWithMutex("n");

    sem_post(&threadSemG);
    sem_post(&threadSemR);
    sem_post(&threadSemI);
    sem_post(&threadSemM);
    sem_wait(&threadSemN); //wait m
    sem_wait(&threadSemN); //wait i
    sem_wait(&threadSemN); //wait r
    sem_wait(&threadSemN); //wait e
    sem_wait(&threadSemN); //wait g

    printSem("n", threadSemSyncN, threadSemSyncG);

    sem_post(&threadSemG);
```

```
    sem_post(&threadSemM);

    printWithMutex("n");

    sem_wait(&threadSemN); //wait p
    sem_wait(&threadSemN); //wait g
    return p;
}

void *jobP(void *p) {
    printWithMutex("p");
    sem_post(&threadSemN);
    return p;
}

int lab2_init() {
    pthread_mutex_init(&mutex, NULL);
    sem_init(&threadSemB, 0, 0);
    sem_init(&threadSemC, 0, 0);
    sem_init(&threadSemD, 0, 0);
    sem_init(&threadSemE, 0, 0);
    sem_init(&threadSemF, 0, 0);
    sem_init(&threadSemI, 0, 0);
    sem_init(&threadSemN, 0, 0);
    sem_init(&threadSemH, 0, 0);
    sem_init(&threadSemG, 0, 0);
    sem_init(&threadSemP, 0, 0);
    sem_init(&threadSemR, 0, 0);
    sem_init(&threadSemSyncM, 0, 0);
    sem_init(&threadSemSyncK, 0, 0);
    sem_init(&threadSemSyncG, 0, 1);
    sem_init(&threadSemSyncN, 0, 0);

    pthread_join(executeJob(jobA), NULL);

    pthread_mutex_destroy(&mutex);
    sem_destroy(&threadSemB);
    sem_destroy(&threadSemC);
    sem_destroy(&threadSemD);
    sem_destroy(&threadSemE);
    sem_destroy(&threadSemF);
    sem_destroy(&threadSemI);
    sem_destroy(&threadSemN);
    sem_destroy(&threadSemH);
    sem_destroy(&threadSemG);
    sem_destroy(&threadSemP);
    sem_destroy(&threadSemR);
    sem_destroy(&threadSemSyncM);
    sem_destroy(&threadSemSyncK);
    sem_destroy(&threadSemSyncG);
```

```
    sem_destroy(&threadSemSyncN);

    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.