ГУАП

КАФЕДРА № 43

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

ассистент		Д.А. Кочин
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №2

Разработка многопоточного приложения средствами POSIX в ОС Linux или Mac OS

по курсу: ОПЕРАЦИОННЫЕ СИСТЕМЫ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 4231		К.А. Чистякова
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2024

Цель работы: Знакомство с многопоточным программированием и методами синхронизации потоков средствами POSIX.

Задание и последовательность выполнения работы:

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:
    i. выводить букву имени потока в консоль;
    ii. 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 1 и 2). 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 и задачу производителя и потребителя). 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.), а также функции приостановки выполнения программы за исключением pthread_yield(). Для этого необходимо написать код в файле lab2.cpp:
    i. Функция unsigned int lab2_thread_graph_id() должна возвращать номер графа запуска потоков, полученный из таблицы вариантов заданий.
    ii. Функция const char* lab2_unsynchronized_threads() должна возвращать строку, состоящую из букв потоков, выполняющихся параллельно без синхронизации (см. примеры в файлах lab2.cpp и lab2_ex.cpp).
    iii. 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 и lab2_ex.cpp).

iv. 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.

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

vi. Создавать функцию main() не нужно. В проекте уже имеется готовая функция main(), изменять ее нельзя. Она выполняет единственное действие: вызывает функцию lab2_init().

vii. 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.

4. 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.

Вариант задания

Номер варианта	**Номер графа запуска потоков**	**Интервалы с несинхронизированными потоками**	**Интервалы с чередованием потоков**
26	10	bcde	dghi

Граф запуска потоков

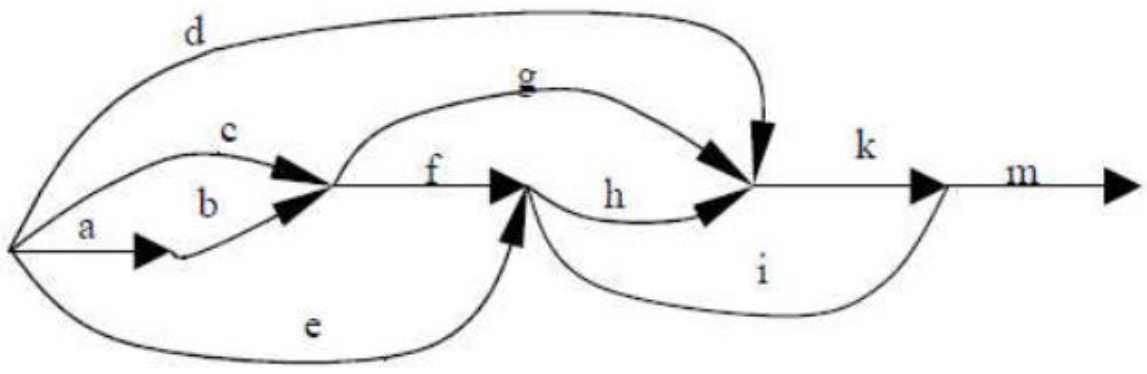


Схема1 - Граф запуска потоков

Результат выполнения работы

Исходный код программы с комментариями:

```
#include "lab2.h"
#include <pthread.h>
#include <semaphore.h>
#include <string.h>

// Объявление массива потоков и синхронизирующих примитивов
pthread_t tid[11];
pthread_mutex_t lock;
sem_t sem_c, sem_d, sem_e, sem_g, sem_h, sem_i, sem_dc, sem_end;
int err;

// Функция идентификации потока в графе
unsigned int lab2_thread_graph_id() { return 10; }

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

// Последовательные потоки
const char *lab2_sequential_threads() { return "dghi"; }

// Функция логирования выполнения потока с блокировкой мьютекса
void log_t(const char *t) {
  pthread_mutex_lock(&lock);
  std::cout << t << std::flush;
  pthread_mutex_unlock(&lock);
  computation(); // Симуляция вычислений
}

// Функция создания потока
int start_t(pthread_t *t, void *(*f)(void *)) {
  return pthread_create(t, NULL, f, NULL);
}

// Функция ожидания завершения потока
void wait_t(pthread_t t) { pthread_join(t, NULL); }

// Прототипы функций потоков
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);
```

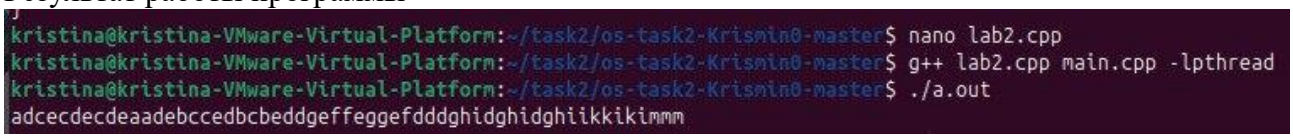
```
void *thread_h(void *ptr);
void *thread_i(void *ptr);
void *thread_k(void *ptr);
void *thread_m(void *ptr);

// Функция потока A
void *thread_a(void *ptr) {
  for (int i = 0; i < 3; i++) {
    log_t("a");
  }
  return ptr;
}

// Функция потока B
void *thread_b(void *ptr) {
  for (int i = 0; i < 3; i++) {
    log_t("b");
  }
  return ptr;
}

// Функция потока C, использует семафоры для синхронизации
void *thread_c(void *ptr) {
  for (int i = 0; i < 3; i++) {
    log_t("c");
  }
  wait_t(tid[0]); // Ожидание завершения потока A
  sem_post(&sem_dc);
  sem_wait(&sem_c);
  for (int i = 0; i < 3; i++) {
    log_t("c");
  }
  return ptr;
}

// Функция потока D
void *thread_d(void *ptr) {
  for (int i = 0; i < 3; i++) {
    log_t("d");
  }
  sem_wait(&sem_dc);
  sem_post(&sem_e);
  sem_wait(&sem_d);
  for (int i = 0; i < 3; i++) {
    log_t("d");
  }
  wait_t(tid[2]); // Ожидание завершения потока C
  wait_t(tid[1]); // Ожидание завершения потока B
  sem_post(&sem_e);
  sem_wait(&sem_d);
  for (int i = 0; i < 3; i++) {
    log_t("d");
```

```
  }

  // Дополнительные итерации с семафорами
  for (int i = 0; i < 3; i++) {
    sem_wait(&sem_d);
    log_t("d");
    sem_post(&sem_g);
  }
  return ptr;
}

// Функция потока E
void *thread_e(void *ptr) {
  for (int i = 0; i < 3; i++) {
    log_t("e");
  }
  sem_post(&sem_d);
  sem_wait(&sem_e);
  sem_post(&sem_c);

  int err = start_t(&tid[1], thread_b);
  if (err != 0)
    std::cerr << "Can't create thread. Error: " << strerror(err) << std::endl;

  for (int i = 0; i < 3; i++) {
    log_t("e");
  }
  sem_post(&sem_d);
  sem_wait(&sem_e);

  err = start_t(&tid[5], thread_f);
  if (err != 0)
    std::cerr << "Can't create thread. Error: " << strerror(err) << std::endl;
  err = start_t(&tid[6], thread_g);
  if (err != 0)
    std::cerr << "Can't create thread. Error: " << strerror(err) << std::endl;

  for (int i = 0; i < 3; i++) {
    log_t("e");
  }
  return ptr;
}

// Функция потока F
void *thread_f(void *ptr) {
  for (int i = 0; i < 3; i++) {
    log_t("f");
  }
  return ptr;
}

// Функция потока G
```

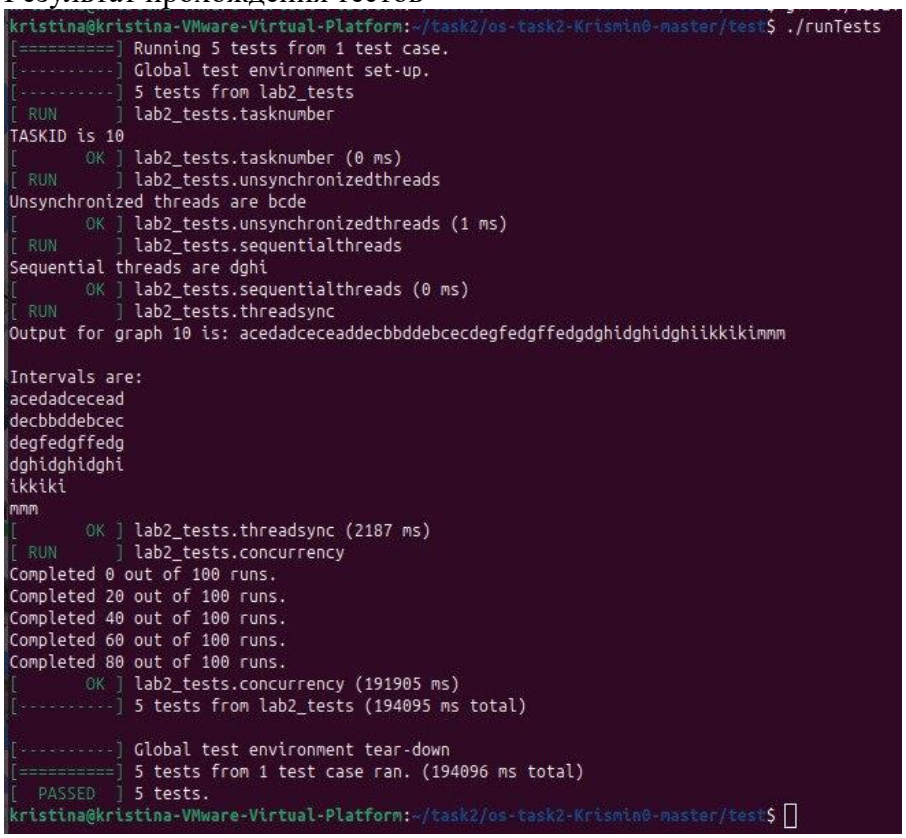
```
void *thread_g(void *ptr) {
  for (int i = 0; i < 3; i++) {
    log_t("g");
  }

  wait_t(tid[5]);
  wait_t(tid[4]);

  int err = start_t(&tid[7], thread_h);
  if (err != 0)
    std::cerr << "Can't create thread. Error: " << strerror(err) << std::endl;
  err = start_t(&tid[8], thread_i);
  if (err != 0)
    std::cerr << "Can't create thread. Error: " << strerror(err) << std::endl;

  sem_post(&sem_d);

  for (int i = 0; i < 3; i++) {
    sem_wait(&sem_g);
    log_t("g");
    sem_post(&sem_h);
  }
  return ptr;
}

// Функция инициализации потоков и семафоров
int lab2_init() {
  if (pthread_mutex_init(&lock, NULL) != 0) {
    std::cerr << "Mutex init failed" << std::endl;
    return 1;
  }

  // Инициализация семафоров
  sem_init(&sem_c, 0, 0);
  sem_init(&sem_d, 0, 0);
  sem_init(&sem_e, 0, 0);
  sem_init(&sem_g, 0, 0);
  sem_init(&sem_h, 0, 0);
  sem_init(&sem_i, 0, 0);
  sem_init(&sem_dc, 0, 0);
  sem_init(&sem_end, 0, 0);

  // Запуск потоков
  start_t(&tid[0], thread_a);
  start_t(&tid[2], thread_c);
  start_t(&tid[3], thread_d);
  start_t(&tid[4], thread_e);

  sem_wait(&sem_end);

  // Очистка ресурсов
  pthread_mutex_destroy(&lock);
```

```
  sem_destroy(&sem_c);
  sem_destroy(&sem_d);
  sem_destroy(&sem_e);
  sem_destroy(&sem_g);
  sem_destroy(&sem_h);
  sem_destroy(&sem_i);
  sem_destroy(&sem_dc);
  sem_destroy(&sem_end);

  return 0;
}
```

Результат работы программы



Результат прохождения тестов



Вывод:

В ходе выполнения данной лабораторной работы было изучено многопоточное программирование и методы синхронизации потоков с использованием средств POSIX. В процессе работы был выбран граф запуска потоков, согласно варианту задания, и реализована программа, обеспечивающая параллельное и последовательно-параллельное выполнение потоков.

Первая группа потоков выполнялась параллельно без синхронизации, что позволило проанализировать особенности работы потоков в условиях конкурентного выполнения.

Вторая группа потоков была синхронизирована с использованием семафоров, что обеспечило их строгое упорядоченное выполнение.

При реализации программы были использованы базовые механизмы синхронизации, включая мьютексы и семафоры. Вся логика многопоточного взаимодействия была реализована в файле lab2.cpp без использования кроссплатформенных библиотек для работы с потоками.

Таким образом, лабораторная работа позволила закрепить знания о принципах работы многопоточных приложений, способах управления потоками и методах их синхронизации в среде POSIX.