ГУАП

КАФЕДРА № 43

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

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

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №3

Разработка многопоточного приложения в ОС Windows

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 Windows API.

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 lab3.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. 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. задачу производителя и потребителя). Таким образом потоки во второй группе выполняются в строгой очередности.
3. С использованием средств Windows API реализовать программу для последовательно-параллельного выполнения потоков в ОС Windows. 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 (например, Sleep(), SwitchToThread() и подобные). Для этого необходимо написать код в файле lab3.cpp:
    i. Функция unsigned int lab3_thread_graph_id() должна возвращать номер графа запуска потоков, полученный из таблицы вариантов заданий.
    ii. Функция const char* lab3_unsynchronized_threads() должна возвращать строку, состоящую из букв потоков, выполняющихся параллельно без синхронизации.
    iii. Функция const char* lab3_sequential_threads() должна возвращать строку, состоящую из букв потоков, выполняющихся параллельно в строгой очередности друг за другом.

iv. Функция int lab3_init() заменяет собой функцию main(). В ней необходимо реализовать запуск потоков, инициализацию вспомогательных переменных (мьютексов, семафоров и т.п.). Перед выходом из функции lab3_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 lab3_init().

vii. Не следует изменять какие-либо файлы, кроме lab3.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

Номер варианта	Номер графа запуска потоков	Несинхронизированные потоки	Потоки с чередованием
26	6	bcd	hik

Граф запуска потоков

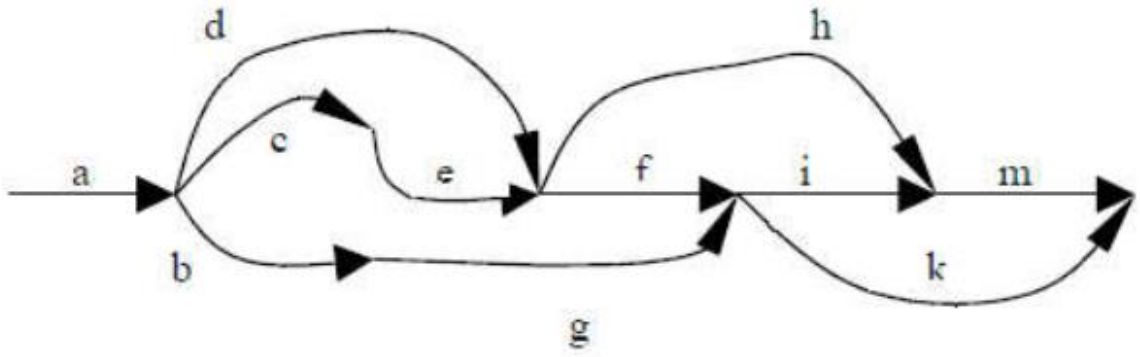


Схема 1 – Граф запуска потоков

Результат выполнения работы

Исходный код программы с комментариями

```
#include "lab3.h"
#include <windows.h>
#include <iostream>

#include <stdio.h>

HANDLE tid[11];
HANDLE lock;
HANDLE sem_h, sem_i, sem_k, sem_a, sem_b, sem_end;

// 26
unsigned int lab3_thread_graph_id() { return 6; }
const char *lab3_unsynchronized_threads() { return "bcd"; }
const char *lab3_sequential_threads() { return "hik"; }

bool start_t(HANDLE *t, DWORD (*f)(LPVOID)) {
  *t = CreateThread(0, 0, f, NULL, 0, 0);
  return *t != NULL;
}
void wait(HANDLE h) { WaitForSingleObject(h, INFINITE); }

void sem_post(HANDLE s) { ReleaseSemaphore(s, 1, NULL); }

void log_t(const char *t) {
  wait(lock);
  std::cout << t << std::flush;
  ReleaseMutex(lock);
  computation();
}

DWORD WINAPI thread_a(LPVOID);
DWORD WINAPI thread_b(LPVOID);
DWORD WINAPI thread_c(LPVOID);
DWORD WINAPI thread_d(LPVOID);
DWORD WINAPI thread_e(LPVOID);
DWORD WINAPI thread_f(LPVOID);
DWORD WINAPI thread_g(LPVOID);
DWORD WINAPI thread_h(LPVOID);
DWORD WINAPI thread_i(LPVOID);
DWORD WINAPI thread_k(LPVOID);
DWORD WINAPI thread_m(LPVOID);

DWORD WINAPI thread_a(LPVOID) {
  for (int i = 0; i < 3; i++) {
    log_t("a");
  }
  if (!start_t(&tid[1], thread_b)) {
    std::cerr << "start_t error";
```

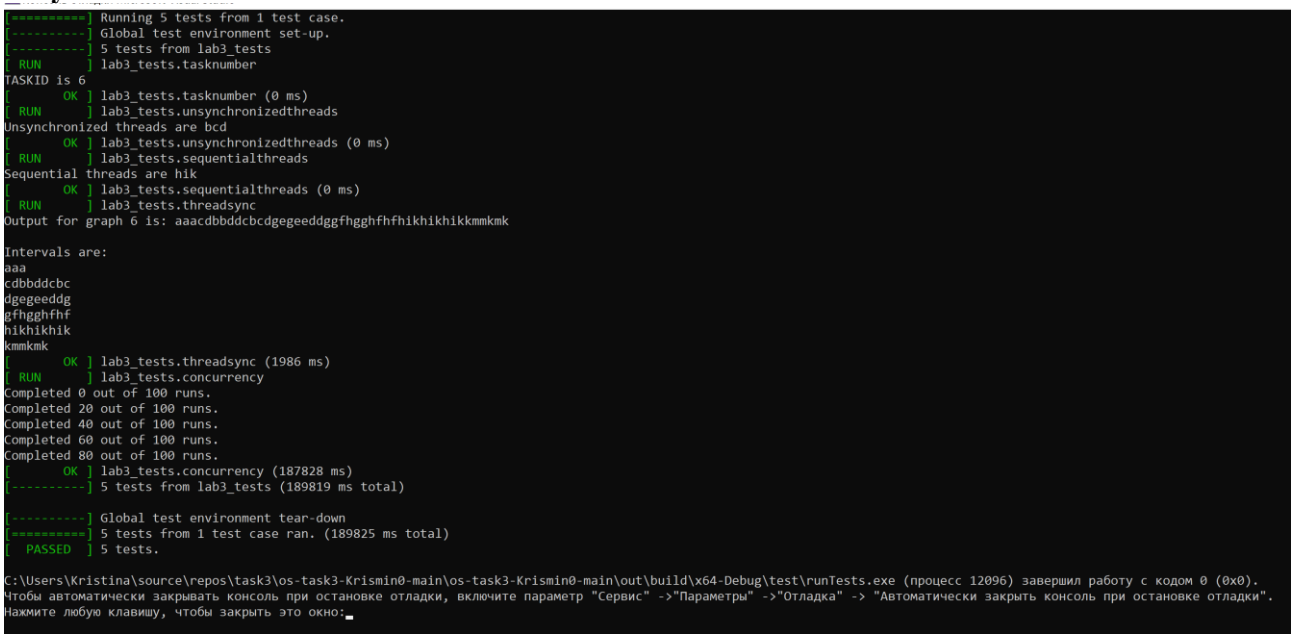
```
  }
  if (!start_t(&tid[2], thread_c)) {
    std::cerr << "start_t error";
  }
  if (!start_t(&tid[3], thread_d)) {
    std::cerr << "start_t error";
  }
  return 0;
}

DWORD WINAPI thread_b(LPVOID) {
  for (int i = 0; i < 3; i++) {
    log_t("b");
  }
  return 0;
}

DWORD WINAPI thread_c(LPVOID) {
  for (int i = 0; i < 3; i++) {
    log_t("c");
  }
  return 0;
}

DWORD WINAPI thread_d(LPVOID) {
  for (int i = 0; i < 3; i++) {
    log_t("d");
  }
  wait(tid[1]);
  wait(tid[2]);

  if (!start_t(&tid[4], thread_e)) {
    std::cerr << "start_t error";
  }
  if (!start_t(&tid[6], thread_g)) {
    std::cerr << "start_t error";
  }
  for (int i = 0; i < 3; i++) {
    log_t("d");
  }
  return 0;
}

DWORD WINAPI thread_e(LPVOID) {
  for (int i = 0; i < 3; i++) {
    log_t("e");
  }
  return 0;
}

DWORD WINAPI thread_f(LPVOID) {
  for (int i = 0; i < 3; i++) {
```

```
    log_t("f");
  }
  return 0;
}

DWORD WINAPI thread_g(LPVOID) {
  for (int i = 0; i < 3; i++) {
    log_t("g");
  }
  wait(tid[3]);
  wait(tid[4]);

  if (!start_t(&tid[5], thread_f)) {
    std::cerr << "start_t error";
  }
  if (!start_t(&tid[7], thread_h)) {
    std::cerr << "start_t error";
  }
  for (int i = 0; i < 3; i++) {
    log_t("g");
  }
  return 0;
}

DWORD WINAPI thread_h(LPVOID) {
  for (int i = 0; i < 3; i++) {
    log_t("h");
  }
  wait(tid[6]);
  wait(tid[5]);

  if (!start_t(&tid[8], thread_i)) {
    std::cerr << "start_t error";
  }
  if (!start_t(&tid[9], thread_k)) {
    std::cerr << "start_t error";
  }

  for (int i = 0; i < 3; i++) {
    wait(sem_h);
    log_t("h");
    sem_post(sem_i);
  }
  return 0;
}

DWORD WINAPI thread_i(LPVOID) {
  for (int i = 0; i < 3; i++) {
    wait(sem_i);
    log_t("i");
    sem_post(sem_k);
  }
```

```
  return 0;
}

DWORD WINAPI thread_k(LPVOID) {
  for (int i = 0; i < 3; i++) {
    wait(sem_k);
    log_t("k");
    sem_post(sem_h);
  }
  if (!start_t(&tid[10], thread_m)) {
    std::cerr << "start_t error";
  }
  for (int i = 0; i < 3; i++) {
    log_t("k");
  }
  wait(tid[10]);
  sem_post(sem_end);
  return 0;
}

DWORD WINAPI thread_m(LPVOID) {
  for (int i = 0; i < 3; i++) {
    log_t("m");
  }
  return 0;
}

int lab3_init() {
  lock = CreateMutex(NULL, false, NULL);
  if (lock == NULL) {
    std::cerr << "CreateMutex error: " << GetLastError() << std::endl;
    return 1;
  }

  sem_end = CreateSemaphore(NULL, 0, 1, NULL);
  if (sem_end == NULL) {
    std::cerr << "CreateSemaphore error: " << GetLastError() << std::endl;
    return 1;
  }
  sem_h = CreateSemaphore(NULL, 1, 1, NULL);
  if (sem_h == NULL) {
    std::cerr << "CreateSemaphore error: " << GetLastError() << std::endl;
    return 1;
  }
  sem_i = CreateSemaphore(NULL, 0, 1, NULL);
  if (sem_i == NULL) {
    std::cerr << "CreateSemaphore error: " << GetLastError() << std::endl;
    return 1;
  }
  sem_k = CreateSemaphore(NULL, 0, 1, NULL);
  if (sem_k == NULL) {
    std::cerr << "CreateSemaphore error: " << GetLastError() << std::endl;
```

```
    return 1;
  }

  if (!start_t(&tid[0], thread_a)) {
    std::cerr << "start_t error";
    return 1;
  }

  wait(sem_end);

  for (int i = 0; i < 11; i++) {
    CloseHandle(tid[i]);
  }
  CloseHandle(lock);
  CloseHandle(sem_h);
  CloseHandle(sem_i);
  CloseHandle(sem_k);
  CloseHandle(sem_end);

  std::cout << std::endl;

  return 0;
}
```

Результат выполнения тестов



Вывод:

В ходе выполнения данной лабораторной работы были изучены принципы многопоточного программирования и методы синхронизации потоков с использованием Windows API.

Был выбран граф запуска потоков в соответствии с вариантом задания, после чего реализована программа, обеспечивающая как несинхронизированное параллельное выполнение потоков, так и их строго упорядоченное выполнение с применением семафоров.

Первая группа потоков выполнялась параллельно без синхронизации, что позволило проанализировать работу потоков при управлении со стороны планировщика задач. Вторая группа потоков была синхронизирована с помощью семафоров, что обеспечило их последовательное выполнение в заданном порядке.

Программа была реализована в файле `lab3.cpp` с использованием Windows API без применения сторонних библиотек для многопоточного программирования. В ходе выполнения работы были использованы механизмы синхронизации, такие как мьютексы и семафоры. Также было обеспечено корректное завершение всех запущенных потоков перед выходом из функции `lab3_init()`.

Таким образом, лабораторная работа позволила освоить базовые механизмы работы с потоками в среде Windows, методы их синхронизации и управления, а также принципы организации параллельного и последовательного выполнения задач.