Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №3 по курсу**

**«Операционные системы»**

Студент: Шумилова Александра

Группа: М8О-207Б-21

Вариант: 18

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2022

**Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. Выводы

**Репозиторий**

https://github.com/tonsoleils/OS

**Постановка задачи**

**Цель работы**

Целью является приобретение практических навыков в:

* Управление потоками в ОС
* Обеспечение синхронизации между потоками

**Задание**

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработке использовать стандартные средства создания потоков операционной системы (Windows/Unix). Ограничение потоков должно быть задано ключом запуска вашей программы.

Так же необходимо уметь продемонстрировать количество потоков, используемое вашей программой с помощью стандартных средств операционной системы.

В отчете привести исследование зависимости ускорения и эффективности алгоритма от входящих данных и количества потоков. Получившиеся результаты необходимо объяснить.

**Вариант 18:** Найти образец в строке наивным алгоритмом

**Общие сведения о программе**

Программа компилируется из файла main.cpp. Также используется заголовочные файлы: iostream, string, cmath. В программе используются следующие системные вызовы:

1. pthread\_mutex\_init(…) – инициализировать мьютекс
2. pthread\_mutex\_lock(…) – «закрыть» мьютекс
3. pthread\_mutex\_unlock(…) – «открыть» мьютекс
4. pthread\_create(…) – создать поток
5. pthread\_join(…) - ожидание завершения потока, заданного параметром

**Общий метод и алгоритм решения**

Для начала считываем количество потоков, переданное на входной поток программе. Далее подсчитываем реальное количество потоков, которое мы будем использовать по следующему алгоритму: если введённое количество потоков меньше или равно размеру строки минус размер образца плюс 1 (это крайний случай, когда каждый поток будет обрабатывать вхождение образца только один раз, иначе остальным потокам нечего будет делать), то мы принимаем введенное количество потоков и вычисляем размер области поиска для каждого из потоков как размер строки делённый на количество потоков. Иначе мы делаем такое количество потоков, какое и описано в крайнем случае выше (размер строки минус размер образца + 1). Далее создаём это количество потоков, в каждом из которых производим поиск образца в строке наивным поиском (посимвольно сравнивая текущий фрагмент строки с образцом, смещаясь на 1 символ при каждой итерации).

**Исходный код**

#include <iostream>

#include <string>

#include <cmath>

**pthread\_mutex\_t** mutex; // Переменная для мьютекса

**void** **naivePatternSearch**(**void**\* inArgs) { // Функция наивного поиска подстроки в строке, принимаем аргументы

pthread\_mutex\_lock(&mutex); // закрываем мьютекс

**struct** threadArgs { // структура принимаемых аргументов

std::string mainString;

std::string pattern;

**int** index;

**size\_t** size;

};

threadArgs args = \*(threadArgs\*)inArgs; // приводим принятые аргументы к типу структуры аргументов

std::string mainString = args.mainString; // и распаковываем каждый аргумент. Это оригинальная строка

std::string pattern = args.pattern; // подстрока

**int** index = args.index; // индекс (по факту порядковый номер текущего потока)

**size\_t** size = args.size; // размер, по которому пробегает поток и ищет подстроку в строке

**size\_t** patLen = pattern.size(); // размер подстроки (образца)

**for** (**int** i = index \* (**int**)size; i < (index \* (**int**)size + (**int**)size); i++) { // index \* size - откуда будет начинать поиск поток

// index \* size + size - где поток закончит поиск

**int** j;

**for** (j = **0**; j < patLen; j++) { // проверяем, совпадает ли каждый символ подстроки с символами самой строки

**if** (mainString[i + j] != pattern[j]) {

**break**; // если какой-то символ не совпал - прерываем цикл и переходим к следующему i

}

}

**if** (j == patLen) { // если всё же у нас подстрока совпала с символами строки - выводим индекс, с которой есть вхождение

std::cout << "[" << index << "] " << "Pattern found at position: " << i << std::endl;

}

}

free(inArgs); // освобождаем переданные данные

pthread\_mutex\_unlock(&mutex); // открываем мьютекс

}

**int** **main**(**int** argc, **char**\*\* argv) {

**size\_t** inThreads; // переменная для введенного количества потоков

**if** (argc != **2**) { // если мы не указали количество потоков при запуске

std::cerr << "Incorrect num of arguments, expected thread count" << std::endl; // выводим ошибки

std::cerr << "Setting thread count to 1" << std::endl;

inThreads = **1**; // приравниваем количество потоков к 1

} **else** { // иначе

std::string temp = argv[**1**]; // считваем введенное количество потоков

inThreads = std::stoi(temp); // преобразует в int

}

std::string mainString = "catdogcathousecatdoga"; // Строка, в которой ищем

std::string pattern = "cat"; // подстрока, которую ищем

**size\_t** threadCount, size; // переменная для количества потоков и размера области, в которой каждый поток будет искать вхождение

**if** (inThreads <= mainString.size() - pattern.size() + **1**) { // если кол-во введенных потоков меньше, чем размер строки - размер образца

// + 1 (иначе у нас будут лишние потоки, которым нечего обрабатывать)

size = mainString.size() / inThreads; // вычислаем размер области поиска для каждого из потоков

threadCount = inThreads; // кол-во потоков

} **else** {

threadCount = mainString.size() - pattern.size() + **1**; // иначе мы делаем количество потоков такое, чтобы каждый поток обработал

// только 1 вхождение в строку

size = mainString.size() / threadCount; // и вычисляем размер области поиска (по факту, она всегда будет равна 1)

}

**pthread\_t** th[threadCount]; // переменная для потоков. количество потоков = threadCount

**int** i;

pthread\_mutex\_init(&mutex, nullptr); // инициализируем мьютекс

**for** (i = **0**; i < threadCount; i++) { // цикл, в котором мы запускаем каждый поток

pthread\_mutex\_lock(&mutex); // закрываем мьютекс

**struct** threadArgs { // подготавливаем структуру для аругментов

std::string mainString;

std::string pattern;

**int** index;

**size\_t** size;

};

**auto** \*arguments = **new** threadArgs; // выделяем место в памяти для структуры. для каждого потока будет выделено отдельное место

arguments->mainString = mainString; // заполняем структуру данными

arguments->pattern = pattern;

arguments->index = i;

arguments->size = size;

// запускаем поток, в котором выполняем функцию naivePatternSearch с параметрами arguments

**if** (pthread\_create(&th[i], nullptr, **reinterpret\_cast**<**void** \*(\*)(**void** \*)>(&naivePatternSearch), arguments) != **0**) {

std::cerr << "error while creating pthread" << std::endl; // если поток не создался - выводим ошибку

}

pthread\_mutex\_unlock(&mutex); // открываем мьютекс

}

**for** (i = **0**; i < threadCount; i++) {

**if** (pthread\_join(th[i], nullptr) != **0**) { // тут мы ждём, когда потоки завершатся. Если кто-то вернул код != 0 - выводим ошибку

std::cerr << "error while launching pthread" << std::endl;

}

}

pthread\_mutex\_destroy(&mutex); // удаляем мьютекс

**return** **0**;

}

**Демонстрация работы программы**

Строка, в которой ищем: "catdogcathousecatdoga"  
Образец, который ищем: "cat"

**tonsoleils@LAPTOP-31GE9NQM:/mnt/d/OS/lab1$** ./lab3 4

[2] Pattern found at position: 14

[1] Pattern found at position: 6

[0] Pattern found at position: 0

**tonsoleils@LAPTOP-31GE9NQM:/mnt/d/OS/lab1$** ./lab3 1

[0] Pattern found at position: 0

[0] Pattern found at position: 6

[0] Pattern found at position: 14

**tonsoleils@LAPTOP-31GE9NQM:/mnt/d/OS/lab1$**  ./lab3 1000

[0] Pattern found at position: 0

[6] Pattern found at position: 6

[14] Pattern found at position: 14

**Выводы**

Проделав данную работу, я научилась управлять потоками в ОС, обеспечивать синхронизацию потоков. Сделала вывод, что синхронизация потоков очень важна, так как при работе с «несинхронизированными» потоками мы можем получить неожидаемое поведение. Так, например, до использования мьютексов в моей лабораторной работе, несколько потоков могли начать одновременно выводить данные и вывод данных смешивался, превращаясь в кашу. Так же могли быть проблемы с доступом к памяти (все потоки обращались бы к одним и тем же адресам), но эту проблему удалось решить ещё до использования мьютексов путём выделения памяти для каждого из процессов.