Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №2 по курсу**

**«Операционные системы»**

**Управление потоками**

Студент: Куценко Максим Дмитриевич

Группа: М8О–212Б-22

Вариант: 18

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2023.

**Постановка задачи**

**Цель работы**

Целью является приобретение практических навыков в:

1. Управление потоками в ОС
2. Обеспечение синхронизации между потоками

**Задание**

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработки использовать стандартные средства создания потоков операционной системы. Ограничение максимального количества потоков, работающих в один момент времени, 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 входных данных и количества потоков. Получившиеся результаты необходимо объяснить.

*Вариант задания 18:*

Найти образец в строке наивным алгоритмом

**Общие сведения о программе**

Программа компилируется из файла naive.c. Также используется заголовочные файлы: "stdlib.h", "stdio.h", "stdbool.h", "pthread.h", "time.h".

В программе используются следующие функции из библиотеки pthread:

1. Инициализация мьютекса.
2. Инициализация базовых параметров.
3. Создание потока.
4. Ожидание завершения работы потоков.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить отличия нитей от процессов, принципы их взаимодействия.

Так, нити не имеют своего адресного пространства, поэтому они куда легковеснее и обеспечивают более быстрое переключение контекста.

1. Составить алгоритм разделения задачи на потоки.

Так, каждый поток сравнивает начинает сравнение строки и образца со следующего символа строки — первый поток начинает с 1-го символа строки, второй поток — со второго, и т.д.

1. Написать структуру, которую потоки используют и изменяют.

Так, имеем две структуры. Первая из них — data, где хранятся указатели на образец, строку и результирующий массив индексов, и переменные, содержащие размеры трёх указанных структур. В эту структуру вводятся данные пользователя и в неё заносится результат.

Вторая структура — data\_wrapper, где хранится указатель на data и позиция в строке, с которой начинаем сравнение с образцом. Этак структура создаётся и передаётся в каждый поток.

1. Обеспечить правильный доступ к критической области — массиву индексов и его размеру.

Для этого используется двоичный мьютекс библиотеки.

1. Обеспечить работу программы с определённым числом потоков в зависимости от ввода пользователя.

Для этого используются argc/argv. Пользователь указывает число потоков, которое затем контролируется через join.

**Основные файлы программы**

**CmakeLists.txt**

cmake\_minimum\_required(VERSION 3.0)

project(Pthreads)

add\_executable(naive naive.c)

**naive.c:**

#include "pthread.h"

#include "stdlib.h"

#include "stdio.h"

#include "stdbool.h"

#include "time.h"

pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;

typedef struct data {

char\* sample;

int sample\_size;

char\* str;

int str\_size;

int\* result;

int result\_size;

} data;

typedef struct data\_wrapper {

data\* pointer;

int pos;

} data\_wrapper;

void\* comparer(void\* wrapper) {

bool same = true;

int pos = ((data\_wrapper\*)wrapper)->pos;

data\* arg = ((data\_wrapper\*)wrapper)->pointer;

for (int i = 0; i < arg->sample\_size; ++i) {

if (arg->str[pos + i] != arg->sample[i]) {

same = false;

break;

}

}

if (same) {

pthread\_mutex\_lock(&mutex);

arg->result = realloc(arg->result, (arg->result\_size + 1)\*sizeof(int));

arg->result[arg->result\_size++] = pos;

pthread\_mutex\_unlock(&mutex);

}

pthread\_exit(0);

}

int naive\_search(data\* arg, int max\_thr) {

pthread\_t tid[max\_thr];

pthread\_attr\_t attr;

pthread\_attr\_init(&attr);

int max\_ind = arg->str\_size - arg->sample\_size;

for (int i=0; i <= max\_ind;) {

int cur\_thr = 0;

while (cur\_thr < max\_thr && i <= max\_ind) {

data\_wrapper\* wrapper = malloc(sizeof(data\_wrapper));

wrapper->pointer = arg;

wrapper->pos = i;

pthread\_create(&tid[cur\_thr], &attr, comparer, wrapper);

++cur\_thr;

++i;

}

for (size\_t j = 0; j < cur\_thr; ++j) {

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

}

}

}

void value\_getter(data\* arg) {

char\* sample = NULL;

char\* str = NULL;

printf("Формат ввода: <образец> \\n <строка>\n");

char c = getchar();

int sample\_size = 0, str\_size = 0;

while (c!='\n') {

sample = realloc(sample, (sample\_size+1)\*sizeof(char));

sample[sample\_size++] = c;

c = getchar();

}

c = getchar();

while (c!='\n') {

str = realloc(str, (str\_size+1)\*sizeof(char));

str[str\_size++] = c;

c = getchar();

}

arg->sample = sample;

arg->sample\_size = sample\_size;

arg->str = str;

arg->str\_size = str\_size;

arg->result = NULL;

arg->result\_size = 0;

}

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

if (argc != 2) {

printf("Неверный ввод! При вызове необходимо указать число нитей.\n");

return 1;

}

data\* arg = malloc(sizeof(data));

value\_getter(arg);

clock\_t t = clock();

naive\_search(arg, atoi(argv[1]));

t = clock() - t;

double time\_taken = ((double)t)/CLOCKS\_PER\_SEC;

printf("\nЗатраченное время: %.3f мс\n", time\_taken\*1000);

printf("Число совпадений: %d\n", arg->result\_size);

if (arg->result\_size) printf("Индексы совпадений: ");

for (int i = 0; i < arg->result\_size; ++i) {

printf("%d ", arg->result[i]);

}

if (arg->result\_size) printf("\n");

}

**Пример работы**

**[Maxim@HONOR-MB15 src]$ ./build/naive 1**

Формат ввода: <образец> \n <строка>

aaaaa

aaaaaaaaaaaaaaaaaaaaaaaaa

Затраченное время: 2.070 мс

Число совпадений: 21

Индексы совпадений: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

**[Maxim@HONOR-MB15 src]$ ./build/naive 3**

Формат ввода: <образец> \n <строка>

aaaaa

aaaaaaaaaaaaaaaaaaaaaaaaa

Затраченное время: 2.244 мс

Число совпадений: 21

Индексы совпадений: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 16 18 19 20

**[Maxim@HONOR-MB15 src]$ ./build/naive 12**

Формат ввода: <образец> \n <строка>

aaaaa

aaaaaaaaaaaaaaaaaaaaaaaaa

Затраченное время: 2.803 мс

Число совпадений: 21

Индексы совпадений: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

**[Maxim@HONOR-MB15 src]$ ./build/naive 1**

Формат ввода: <образец> \n <строка>

abc

abdabeabcabtabuabkaboabmabc

Затраченное время: 2.138 мс

Число совпадений: 2

Индексы совпадений: 6 24  
**[Maxim@HONOR-MB15 src]$ ./build/naive 5**

Формат ввода: <образец> \n <строка>

abc

abdabeabcabtabuabkaboabmabc

Затраченное время: 2.335 мс

Число совпадений: 2

Индексы совпадений: 6 24

**[Maxim@HONOR-MB15 src]$ ./build/naive 3**

Формат ввода: <образец> \n <строка>

a

bcde

Затраченное время: 1.408 мс

Число совпадений: 0

**[Maxim@HONOR-MB15 src]$ ./build/naive 3**

Формат ввода: <образец> \n <строка>

aa

a

Затраченное время: 0.014 мс

Число совпадений: 0

* [Maxim@HONOR-MB15 src]$ ./build/naive

Неверный ввод! При вызове необходимо указать число нитей.

**Вывод**

Изучил, как устроены pipe и как они могут быть полезны при взаимодействии между процессами, условия для которого были созданы при помощи команд fork и exec. Выяснил, что из себя представляют дескрипторы и как на практике работает вывод в языке Си.

Также в процессе работы над программой определил, что лабораторные работы желательно писать, используя исключительно инструментарий языка Си, потому что системные вызовы ориентированы именно на него.

При записи в файл и вывод понял, что числа печатаются для пользователя в текстовом виде, а не в бинарном. Бинарные же символы в файлах формата .txt и в терминале отображаются некорректно.

Встретил ошибку, связанную со считыванием символов в char\* из пользовательского ввода и выяснил важность использования „\0“.

В процессе работы над программой изучил, что из себя представляют нити и как можно осуществлять их взаимодействие и синхронизацию друг с другом. Так, множество потоков могут одновременно читать одни и те же величины, но для их изменения необходимо пользоваться мьютексом и/или функцией join. Кроме того, потокам можно указывать приоритеты выполнения, но этим я в лабораторной работе не пользовался.

Для замера времени работы участка кода в зависимости от различного числом потоков я воспользовался библиотекой time.h. Время решил выводить в милисекундах для большего удобства.

Замеры времени дали довольно интересный результат — большее число потоков давало в среднем большее время выполнения участка кода. Впрочем, это довольно просто объяснить — каждый поток занимается тем, что посимвольно сравнивает строки, на что уходит достаточно мало времени, и при этом их создаётся достаточно много, поэтому многопоточность в данном случае нецелесообразна.