Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №3 по курсу**

**«Операционные системы»**

**Управление потоками в ОС. Обеспечение синхронизации между потоками.**

Студент: Мариничев И. А.

Группа: М8О-208Б-19

Преподаватель: Миронов Е. С.

Дата: 14.12.2020

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2020

1. **Постановка задачи. Вариант 18**

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 программы.

Необходимо уметь продемонстрировать количество потоков, используемых программой, с помощью стандартных средств операционной системы.

Привести исследование зависимости ускорения и эффективности алгоритма от входящих данных и количества потоков. Объяснить получившиеся результаты.

Найти образец в строке наивным алгоритмом.

1. **Общие сведения о программе**

Программа написана на языке Си в UNIX-подобной операционной системе (Ubuntu). Для компиляции программы требуется указать ключ –pthread. Для запуска программы в качестве аргумента командной строки необходимо указать количество потоков, которые могут быть использованы программой.

Программа содержит содержит структуру ThreadArg которая содержит все необходимые переменные для поиска подстроки в строке, она необходима для того, чтобы любой поток имел к ним доступ.

Программа включает в себя потоковую функцию void\* Search(void\* argument), в которой помечаются все вхождения подстроки. Так как все потоки программы работают в одном и том же пространстве памяти, аргументы для передачи потоковой функции хранятся по разным адресам (в массиве args, размер которого равен количеству потоков).

В программе предусмотрена проверка на системные ошибки – ошибки выделения памяти, ошибки запуска.

1. **Общий метод и алгоритм решения**

При запуске программы у пользователя запрашивается ввести длину строки, длину образца, строку, образец. Образец не может превышать строку, предусмотрен вывод ошибки в данном случае.

Из аргументов командной строки берётся количество потоков, которое может использовать программа. Производится выделение памяти для массива потоков, для массива аргументов потоковой функции и для введенного пользователем текста и шаблона.

Далее в зависимости от введенного числа потоков введенная строка разбивается на перекрывающиеся интервалы с перекрытием размером в искомую подстроку минус один. Каждому потоку делегируется чать текста для поиска подстроки. Поиск происходит по наивному алгоритму, т.е. мы проходим по всей строке, сравнивая посимвольно шаблон со строкой. Потоки не смогут повлиять на работу друг друга, поэтому mutex не используется.

После обработки всего текста необходимо дождаться окончания работы всех активных потоков. После этого необходимо вывести минимальный индекс вхождения подстроки в текст среди найденных или указать, что такой строки в тексте не было найдено.

1. **Основные файлы программы**

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <time.h>

typedef struct ThreadArg {

    char \*str;

    char \*pattern;

    int strLen;

    int patternLen;

    int startIndex;

    int endIndex;

    int res;

} ThreadArg;

// Thread function - Naive algorithm for Pattern Searching

void\* Search(void\* argument) {

    ThreadArg\* arg = (ThreadArg\*) argument;

    char \*pattern = arg->pattern;

    char \*str = arg->str;

    int found = 1;

    if(arg->strLen < arg->patternLen) {

        printf("ERROR: pattern is larger than string.\n");

        exit(4);

    }

    // A loop to slide patttern[] one by one

    for (int i = arg->startIndex; i < arg->endIndex; ++i) {

        for (int j = 0; j < arg->patternLen; ++j) {

            if(i + j - arg->startIndex >= arg->strLen) {

                found = 0;

                break;

            }

            if (str[i+j] != pattern[j]) { //One character doesn't match

                found = 0;

                break;

            }

        }

        if(found) { // if pattern[0...(arg->patternLen)-1] = text[i, i+1, ...i+(arg->patternLen)-1]

            arg->res = i;

            pthread\_exit(NULL);

        }

        else {

            found = 1;

        }

    }

    arg->res = -1;

    pthread\_exit(NULL);

}

int main(int argc, char\* argv[]) {

    struct timespec start, finish;

    if (argc != 2) {

        printf("Syntax: ./main Number\_of\_threads\n");

        exit(1);

    }

    // Threads initialization

    int threadNumber = atoi(argv[1]);

    printf("Thread number: %d\n", threadNumber);

    pthread\_t\* threads = (pthread\_t\*)malloc(sizeof(pthread\_t) \* threadNumber);

    if (threads == NULL) {

        printf("ERROR: unable to allocate space for threads.\n");

        exit(3);

    }

    // Iput text and pattern initialization

    int input\_string\_len, input\_pattern\_len;

    printf("Enter string length: ");

    scanf(" %d", &input\_string\_len);

    printf("Enter pattern length: ");

    scanf(" %d", &input\_pattern\_len);

    char \*text = malloc(sizeof(char) \* (input\_string\_len+1));

    if (text == NULL) {

        printf("ERROR: unable to create an array for text.\n");

        exit(3);

    }

    char \*pattern = malloc(sizeof(char) \* (input\_pattern\_len+1));

    if (pattern == NULL) {

        printf("ERROR: unable to create an array for pattern.\n");

        exit(3);

    }

    printf("Enter string: ");

    scanf(" %s", text);

    printf("Enter pattern: ");

    scanf(" %s", pattern);

    // Array for arguments which will be passed into thread function

    ThreadArg\* args = (ThreadArg\*)malloc(sizeof(ThreadArg) \* threadNumber);

    if (args == NULL) {

        printf("ERROR: unable to create an array of arguments for threads\n");

        exit(3);

    }

    // Gets time of the start

    clock\_gettime(CLOCK\_MONOTONIC, &start);

    // Separating text for threads

    int step = input\_string\_len / threadNumber;

    for (int i = 0; i < threadNumber; ++i) {

        int curLen = step;

        if (i < threadNumber - 1) {

            curLen += input\_pattern\_len;

        }

        args[i].strLen = curLen;

        args[i].str = text;

        args[i].startIndex = step \* i;

        args[i].endIndex = args[i].startIndex + curLen;

        args[i].pattern = pattern;

        args[i].patternLen = input\_pattern\_len;

    }

    // Starting threads

    for (int i = 0; i < threadNumber; ++i) {

        if (pthread\_create(&threads[i], NULL, Search, &args[i]) != 0) {

            perror("ERROR: unable to create thread.\n");

            exit(2);

        }

    }

    // Waiting for all threads

    for (int i = 0; i < threadNumber; ++i){

        if (pthread\_join(threads[i], NULL) != 0) {

            perror("ERROR: unable to join threads.");

            exit(2);

        }

    }

    // Gets minimal index of the patern occurence if it exists

    int min\_index = -1;

    for(int i = 0; i < threadNumber; ++i) {

        if((args[i].res != -1 && args[i].res < min\_index) || (min\_index == -1 && args[i].res != -1)) {

            min\_index = args[i].res;

        }

    }

    if (min\_index == -1) {

        printf("Pattern wasn't found!\n");

    }

    else {

        printf("Minimal index of the pattern occurence in the text: %d\n", min\_index);

    }

    // Gets finish time and prints overall time

    clock\_gettime(CLOCK\_MONOTONIC, &finish);

    double elapsed = (finish.tv\_sec - start.tv\_sec);

    elapsed += (finish.tv\_nsec - start.tv\_nsec) / 1000000000.0;

    printf("Time: %.4f seconds\n", elapsed);

    free(text);

    free(pattern);

    free(threads);

    free(args);

    return 0;

}

1. **Демонстрация работы программы**

**ivan@Laptop-IM:/mnt/c/Users/Иван/projects/os\_labs/os\_lab3$** cat Makefile

FLAGS = -std=c99 -pthread

СС = gcc

all: main

main: main.c

$(CC) $(FLAGS) main.c -o main

clean:

rm main

**ivan@Laptop-IM:/mnt/c/Users/Иван/projects/os\_labs/os\_lab3$** make

cc -std=c99 -pthread main.c -o main

**ivan@Laptop-IM:/mnt/c/Users/Иван/projects/os\_labs/os\_lab3$** ./main

Syntax: ./main Number\_of\_threads

**ivan@Laptop-IM:/mnt/c/Users/Иван/projects/os\_labs/os\_lab3$** ./main 1

Thread number: 1

Enter string length: 10

Enter pattern length: 2

Enter string: afagabagab

Enter pattern: ab

Minimal index of the pattern occurence in the text: 4

Time: 0.0018 seconds

**ivan@Laptop-IM:/mnt/c/Users/Иван/projects/os\_labs/os\_lab3$** ./main 1

Thread number: 1

Enter string length: 5

Enter pattern length: 10

Enter string: asdfg

Enter pattern: asdfghjklo

ERROR: pattern is larger than string.

**ivan@Laptop-IM:/mnt/c/Users/Иван/projects/os\_labs/os\_lab3$** ./main 1 < test\_02.txt

Thread number: 1

Enter string length: Enter pattern length: Enter string: Enter pattern: Minimal index of the pattern occurence in the text: 27639

Time: 0.0010 seconds

**ivan@Laptop-IM:/mnt/c/Users/Иван/projects/os\_labs/os\_lab3$** ./main 2 < test\_02.txt

Thread number: 2

Enter string length: Enter pattern length: Enter string: Enter pattern: Minimal index of the pattern occurence in the text: 27639

Time: 0.0020 seconds

**ivan@Laptop-IM:/mnt/c/Users/Иван/projects/os\_labs/os\_lab3$** ./main 3 < test\_02.txt

Thread number: 3

Enter string length: Enter pattern length: Enter string: Enter pattern: Minimal index of the pattern occurence in the text: 27639

Time: 0.0013 seconds

**ivan@Laptop-IM:/mnt/c/Users/Иван/projects/os\_labs/os\_lab3$** ./main 4 < test\_02.txt

Thread number: 4

Enter string length: Enter pattern length: Enter string: Enter pattern: Minimal index of the pattern occurence in the text: 27639

Time: 0.0018 seconds

**ivan@Laptop-IM:/mnt/c/Users/Иван/projects/os\_labs/os\_lab3$** ./main 5 < test\_02.txt

Thread number: 5

Enter string length: Enter pattern length: Enter string: Enter pattern: Minimal index of the pattern occurence in the text: 27639

Time: 0.0019 seconds

**ivan@Laptop-IM:/mnt/c/Users/Иван/projects/os\_labs/os\_lab3$** ./main 6 < test\_02.txt

Thread number: 6

Enter string length: Enter pattern length: Enter string: Enter pattern: Minimal index of the pattern occurence in the text: 27639

Time: 0.0023 seconds

**ivan@Laptop-IM:/mnt/c/Users/Иван/projects/os\_labs/os\_lab3$** ./main 7 < test\_02.txt

Thread number: 7

Enter string length: Enter pattern length: Enter string: Enter pattern: Minimal index of the pattern occurence in the text: 27639

Time: 0.0018 seconds

**ivan@Laptop-IM:/mnt/c/Users/Иван/projects/os\_labs/os\_lab3$** ./main 8 < test\_02.txt

Thread number: 8

Enter string length: Enter pattern length: Enter string: Enter pattern: Minimal index of the pattern occurence in the text: 27639

Time: 0.0029 seconds

**ivan@Laptop-IM:/mnt/c/Users/Иван/projects/os\_labs/os\_lab3$** ./main 9 < test\_02.txt

Thread number: 9

Enter string length: Enter pattern length: Enter string: Enter pattern: Minimal index of the pattern occurence in the text: 27639

Time: 0.0030 seconds

**ivan@Laptop-IM:/mnt/c/Users/Иван/projects/os\_labs/os\_lab3$** ./main 100 < test\_02.txt

Thread number: 100

Enter string length: Enter pattern length: Enter string: Enter pattern: Minimal index of the pattern occurence in the text: 27639

Time: 0.0277 seconds

1. **Strace**

**ivan@Laptop-IM:/mnt/c/Users/Иван/projects/os\_labs/os\_lab3$** strace -f -e trace="%process,write" -o strace\_log.txt ./main 3

Thread number: 3

Enter string length: 10

Enter pattern length: 2

Enter string: adagosodos

Enter pattern: os

Minimal index of the pattern occurence in the text: 4

Time: 0.0075 seconds

**ivan@Laptop-IM:/mnt/c/Users/Иван/projects/os\_labs/os\_lab3$** cat strace\_log.txt

13025 execve("./main", ["./main", "3"], 0x7fffef7c9390 /\* 19 vars \*/) = 0

13025 arch\_prctl(ARCH\_SET\_FS, 0x7fc602c50740) = 0

13025 write(1, "Thread number: 3\n", 17) = 17

13025 write(1, "Enter string length: ", 21) = 21

13025 write(1, "Enter pattern length: ", 22) = 22

13025 write(1, "Enter string: ", 14) = 14

13025 write(1, "Enter pattern: ", 15) = 15

**13025 clone(child\_stack=0x7fc6023cffb0, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tidptr=0x7fc6023d09d0, tls=0x7fc6023d0700, child\_tidptr=0x7fc6023d09d0) = 13026**

**13025 clone(child\_stack=0x7fc601bbffb0, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tidptr=0x7fc601bc09d0, tls=0x7fc601bc0700, child\_tidptr=0x7fc601bc09d0) = 13027**

**13025 clone(child\_stack=0x7fc6013affb0, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tidptr=0x7fc6013b09d0, tls=0x7fc6013b0700, child\_tidptr=0x7fc6013b09d0) = 13028**

13026 exit(0) = ?

13027 exit(0 <unfinished ...>

13026 +++ exited with 0 +++

13027 <... exit resumed>) = ?

13027 +++ exited with 0 +++

13028 exit(0) = ?

13028 +++ exited with 0 +++

13025 write(1, "Minimal index of the pattern occ"..., 54) = 54

13025 write(1, "Time: 0.0075 seconds\n", 21) = 21

13025 exit\_group(0) = ?

13025 +++ exited with 0 +++

**ivan@Laptop-IM:/mnt/c/Users/Иван/projects/os\_labs/os\_lab3$** strace -f -e trace="%process,write" -o strace\_log1.txt ./main 2

Thread number: 2

Enter string length: 15

Enter pattern length: 3

Enter string: abdabfabhabcabr

Enter pattern: abc

Minimal index of the pattern occurence in the text: 9

Time: 0.0157 seconds

**ivan@Laptop-IM:/mnt/c/Users/Иван/projects/os\_labs/os\_lab3$** cat strace\_log1.txt

13034 execve("./main", ["./main", "2"], 0x7fffe7ec50e0 /\* 19 vars \*/) = 0

13034 arch\_prctl(ARCH\_SET\_FS, 0x7f6bad050740) = 0

13034 write(1, "Thread number: 2\n", 17) = 17

13034 write(1, "Enter string length: ", 21) = 21

13034 write(1, "Enter pattern length: ", 22) = 22

13034 write(1, "Enter string: ", 14) = 14

13034 write(1, "Enter pattern: ", 15) = 15

13034 clone(child\_stack=0x7f6bac7cffb0, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tidptr=0x7f6bac7d09d0, tls=0x7f6bac7d0700, child\_tidptr=0x7f6bac7d09d0) = 13035

13034 clone(child\_stack=0x7f6babfbffb0, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tidptr=0x7f6babfc09d0, tls=0x7f6babfc0700, child\_tidptr=0x7f6babfc09d0) = 13036

13035 exit(0 <unfinished ...>

13036 exit(0 <unfinished ...>

13035 <... exit resumed>) = ?

13036 <... exit resumed>) = ?

13035 +++ exited with 0 +++

13036 +++ exited with 0 +++

13034 write(1, "Minimal index of the pattern occ"..., 54) = 54

13034 write(1, "Time: 0.0157 seconds\n", 21) = 21

13034 exit\_group(0) = ?

13034 +++ exited with 0 +++

1. **Исследование ускорения и эффективности**

Для исследования ускорения и эффективности многопоточного поиска образца в строке наивным алгоритмом замерим время работы программы для строк следующих размеров:

1. 57625
2. 100000
3. 25138020

Время работы программы будет замеряться при помощи функции clock\_gettime(). Нужно учитывать, что время работы может варьироваться в небольших пределах из-за постоянной работы фоновых процессов. Результаты будут занесены в таблицы.

Таблица 1. Исследование поиска подстроки в тексте из 57625 символов

|  |  |  |  |
| --- | --- | --- | --- |
| Количество потоков (n) | Время работы программы (Tn), сек | Ускорение  (Sn = T1 / Tn) | Эффективность  (Xn = Sn / n) |
| 1 | 0.0011 | - | - |
| 2 | 0.0007 | 1,57 | 0,785 |
| 3 | 0.0009 | 1,22 | 0,407 |
| 4 | 0.0004 | 2,75 | 0,687 |
| 5 | 0.0005 | 2,2 | 0,44 |
| 6 | 0.0006 | 1,05 | 0,175 |
| 7 | 0.0005 | 1,83 | 0,261 |
| 8 | 0.0005 | 1,83 | 0,229 |
| 9 | 0.0006 | 1,05 | 0,117 |
| 10 | 0.0008 | 1,375 | 0,137 |

Таблица 2. Исследование поиска подстроки в тексте из 100000 символов

|  |  |  |  |
| --- | --- | --- | --- |
| Количество потоков (n) | Время работы программы (Tn), сек | Ускорение  (Sn = T1 / Tn) | Эффективность  (Xn = Sn / n) |
| 1 | 0.0834 | - | - |
| 2 | 0.0534 | 1,562 | 0,781 |
| 3 | 0.0567 | 1,47 | 0,49 |
| 4 | 0.0555 | 1,503 | 0,376 |
| 5 | 0.0578 | 1,443 | 0,289 |
| 6 | 0.0714 | 1,168 | 0,195 |
| 7 | 0.0884 | 0,943 | 0,135 |
| 8 | 0.0605 | 1,378 | 0,173 |
| 9 | 0.0668 | 1,248 | 0,138 |
| 10 | 0.0621 | 1,343 | 0,134 |

Таблица 3. Исследование поиска подстроки в тексте из 25138020 символов

|  |  |  |  |
| --- | --- | --- | --- |
| Количество потоков (n) | Время работы программы (Tn), сек | Ускорение  (Sn = T1 / Tn) | Эффективность  (Xn = Sn / n) |
| 1 | 0.9790 | - | - |
| 2 | 0.4924 | 1,988 | 0,994 |
| 3 | 0.5757 | 1,7 | 0,567 |
| 4 | 0.6264 | 1,562 | 0,39 |
| 5 | 0.6807 | 1,438 | 0,29 |
| 6 | 0.7338 | 1,334 | 0,22 |
| 7 | 0.7694 | 1,272 | 0,182 |
| 8 | 0.8565 | 1,143 | 0,143 |
| 9 | 0.8667 | 1,129 | 0,125 |
| 10 | 0.9251 | 1,058 | 0,106 |

По этим данным можно увидеть, что значительный выигрыш по времени мы имеем только при обработке действительно больших чисел. Оптимальнее всего использовать 2-3 потока. Таким образом можно ускорить время работы программы примерно в полтора раза. Средняя эффективность составляет 0,5. Если использовать большее количество потоков, то выигрыш от параллельной обработки будет перекрываться затратами на создание и регулирование потоков.\*

\*Ускорение должно наблюдаться при 4 потоках, так как вычислительная машина, на которой тестировалась данная программа имеет 4 ядра.

1. **Выводы**

В ходе данной лабораторной работы я познакомился с потоками на практике и увидел разницу между ними и процессами в программной реализации.

Создание потоков происходит быстрее, чем создание процессов, а все потоки используют одну и ту же область данных. Поэтому многопоточность – один из способов ускорить обработку каких-либо данных: выполнение однотипных, не зависящих друг от друга задач, можно поручить отдельным потокам, которые будут работать парарллельно.

Средствами языка Си можно совершать системные запросы на создание потока **pthread\_create()**, ожидания завершения потока **pthread\_join()**, а также использовать различные примитивы синхронизации (например, **pthread\_mutex\_lock()**/**pthread\_mutex\_unlock()**).

В данной лабораторной работе был реализован и исследован наивный алгоритм поиска образца в строке. В ходе исследования ускорения и эффективности было установлено, что при использовании двух-трёх потоков можно получить выигрыш по времени примерно в полтора раза, что значительно ускоряет обработку большого количества чисел. Но при использовании большего количества потоков ускорение не будет большим, так как операционной системе приходится тратить больше времени на выделение памяти под потоки и на их регулирование.