**Введение**

С помощью данной лабораторной работы можно понять, каким образом работает планировщик потоков операционной системы, как работает бинарный поиск и когда следует его применять. Благодаря данной лабораторной работе можно рассмотреть, как реализуются и работают сортировки подсчётом, Шелла и пузырьком.   
При реализации данной лабораторной работы было изучено много нового и необходимого для дальнейшего программирования, также, был получен опыт реализации серьезных задач.

**Постановка задачи**

Реализовать программу, эмулирующую работу планировщика потоков операционной системы.

ОС распоряжается ресурсом ЦП и этот ресурс может быть выделен исполняемым сущностям – потокам. В ОС в каждый момент времени может быть несколько потоков ожидающих выполнения, поэтому возникает задача выбора очередного потока на исполнение. Прерывание выполнения потока не допускается. Ресурс ЦП представлен в виде непрерывного кванта времени, который может быть выделен потоку для его исполнения.

Алгоритм выбора очередного потока на исполнение будет состоять в выборе потока который имеет наиболее близкое значение к выделенному кванту времени ЦП.

Программа должна получить входные данные: количество потоков, время потока, номер потока и расположить данные в порядке их обработки ЦП. Вывод осуществляется в виде таблицы с полями:

Программа должна иметь интерфейс для реализации диалога с пользователем, который включает в себя:

* Возможность выбора количества потоков;
* Выбор способа задания времени потока:
* Ручной ввод;
* Генерация случайным образом;
* Чтение из файла;
* Выбор метода сортировки:
* Сортировка пузырьком;
* Сортировка Шелла (последовательность Хиббарда);
* Сортировка Подсчетом;
* Выбор способа задания времени ЦП:
* Ручной ввод;
* Генерация случайным образом;
* Чтение из файла;
* Если количество заданных потоков не превышает 21, то результат работы программы должен выводиться на экран, иначе в файл;
* Должна быть реализована проверка данных, введенных пользователем, на корректность. В случае ошибки необходимо известить об этом пользователя и предложить повторить ввод.
* Чтобы пользователь имел возможность сравнить эффективность сортировки разными алгоритмами, программа должна выводить на экран:
* Время выполнения процедуры сортировки;
* Число выполненных перестановок и сравнений;
* По окончанию выполнения сортировки, необходимо предложить пользователю применить другой алгоритм сортировки.

**Руководство пользователя**

Запуск программы осуществляется непосредственно с помощью исполняемого файла.

* После запуска программы, появится приветствие пользователя, и программа сразу же предложит на выбор 3 способа задания времени потоков операционной системы.



* Ввод осуществляется с клавиатуры. Допустим, в качестве примера, введем 1.



* Программа известит пользователя о подключенной процедуре и запросит ввести необходимое количество потоков.

Введем, например, количество потоков = 10.000.000.



* После выполнения указанной процедуры, программа известит пользователя о конце процедуры.

Далее, программа предложит отсортировать данные несколькими способами, какими – видно на изображении. Выберем 3 для примера.



После выполнения сортировки программа выведет на экран количество сравнений, перестановок и время, затраченное на выполнение. Как видите, время затраченное на сортировку 10 миллионов потоков = 147 мс, что представляет собой очень хороший результат.

*(Прим. Intel Core i5-4670K - 3.40Ghz)*

* Также, пользователю будет предложено применить другой алгоритм сортировки к исходным данным. Мы же выберем 2, и продолжим работу с программой.



* Программа предложит пользователю выбрать способ задания времени ЦП. Выберем 1. Программа начнет располагать потоки в порядке их выполнения ЦП, если количество потоков не превышает 21, то результат работы выведется на экран.

В нашем случае запись будет произведена в файл.



* Когда программа закончит выполнение алгоритма и произведет запись результата работы в файл, на экране появится время, затраченное на работу. Также, программа предложит пользователю продолжить работу с ней с самого начала.

**Руководство программиста**

**Описание структуры программы**

**Программа состоит из следующих файлов:**

* исполняемого файла (Лабораторная работа №1.exe);
* файла, содержащего время потоков (Input\_Stream\_LABA.txt);
* файла, содержащего время ЦП(Input\_CP\_LABA.txt);
* файла, в который сохраняется результат работы (Output\_LABA.txt).

**В программе реализованы следующие процедуры:**

* ***procedure*** *PROTECTION\_FOOL\_STREAM;*

- процедура проверки ввода количества потоков, введенных пользователем(защита от введения символов и т.п).

* ***procedure*** *PROTECTION\_FOOL\_MENU;*

- процедура проверки ввода в меню, выбор в котором представляется из 3 вариантов ответа.

* ***procedure*** *PROTECTION\_FOOL\_MENU\_1;*

**-** тоже самое, что PROTECTION\_FOOL\_MENU, но для меню, в котором только 2 варианта ответа.

* ***procedure*** *Input\_random;*

**-** генерация времени потоков случайным образом (random(255)).

* ***procedure*** *Input\_hand;*
* ввод времени потоков вручную.
* ***procedure*** *Input\_file;*

**-** ввод времени потоков из файла Input\_Stream\_LABA.txt.

* ***Procedure*** *Time\_CP\_Random;*

**-** генерация времени ЦП случайным образом.

* ***procedure*** *SORT\_bubble;*

**-** сортировка пузырьком.

* ***Procedure*** *SORT\_Shell;*

**-** сортировка Шелла (последовательность Хиббарда).

* ***Procedure*** *SORT\_Count;*

**-** сортировка подсчётом.

* *procedure Copy\_Arr;*

**-** процедура копирования массива (необходима для повторения сортировок над исходными данными).

* ***procedure*** *Output\_head;*

**-** вывод на экран «головы» таблицы, в которую будет записываться результат работы программы.

* ***Procedure*** *Output\_Body;*

**-** вывод на экран «тела» таблицы.

* ***procedure*** *Output\_End;*

**-** полоска, завершающая таблицу.

* ***Procedure*** *Output\_File\_Head;*

**-** то же самое что и *Output\_head* но, вывод происходит в файл.

* ***procedure*** *Output\_File\_LAB;*

**-** то же самое что *Output\_Body* но, вывод происходит в файл.

* ***procedure*** *Output\_File\_End;*

**-** то же самое что *Output\_End* но, вывод происходит в файл.

* ***procedure*** *Output\_Data\_Entered;*

**-** вывод исходного массива на экран (выполняется, если количество элементов в массиве не превышает 21).

* ***procedure*** *Tabl\_data\_Entered\_Head;*

**-** «голова» таблицы для вывода промежуточных данных .

* ***Procedure*** *Output\_Sort\_Data;*

**-** вывод сортированного массива на экран (выполняется, если количество элементов в массиве не превышает 21).

* ***Procedure*** *Binary\_Search;*

**-** процедура бинарного поиска.

* ***Procedure*** *Shift\_Array;*

**-** процедура сдвига массива, выполняется после бинарного поиска, и по сути «выкидывает» из массива найденный элемент.

**В программе можно выделить 5 блоков:**

* Процедуры ввода времени потоков 3 разными способами:

*Input\_random, Input\_hand, Input\_file.*

* Процедуры сортировки массивов 3 разными способами и вспомогательная процедура для них:

*SORT\_bubble, SORT\_Shell, SORT\_Count, Copy\_Arr.*

* Процедура бинарного поиска и вспомогательная процедура для нее:

*Binary\_Search,Shift\_Array.*

* Процедуры вывода данных на экран\в файл:

*Output\_head, Output\_Body, Output\_end, Output\_File\_Head, Output\_File\_LAB, Output\_File\_End,Tabl\_data\_Entered\_Head, Output\_Data\_Entered, Output\_Sort\_Data.*

* Процедуры защиты ввода:

*PROTECTION\_FOOL\_STREAM, PROTECTION\_FOOL\_MENU, PROTECTION\_FOOL\_MENU\_1.*

**Описание структур данных**

a,b: **array of** byte; – массивы для хранения времени потоков

a – массив в который записываются исходные данные, b – массив в котором происходит сортировка;

Number\_El: **array of** integer; – массив в котором хранятся номера потоков;

**i –** переменная в которую сохраняется количество введенных потоков;

**j** – переменная используемая в циклах for;

**z** – переменная используемая в процедуре сортировки пузырьком. Сохраняет значение b[j] а потом присваивает его значению b[j+1];

**t1, t2** – переменные, используемые для измерения времени. Время выполнения высчитывается как t2 – t1;

**k** – переменная используемая в процедуре сортировки пузырьком. Используется для сохранения номера элемента. Сохраняет значение Number\_El [j] а потом присваивает его значению Number\_El [j+1];

**s** – переменная в которую сохраняется длинна массива. Используется в бинарном поиске как правая граница массива;

**Time\_CP** – переменная в которую сохраняется выделенный квант времени ЦП;

**Menu\_Input** – переменная используемая в меню выбора;

**mid, left, right** – переменные используемые в бинарном поиске. Left – левая граница поиска, Right – правая граница поиска, mid – середина выбранного участка поиска;

**Length\_Arr** – переменная в которую сохраняется длинна массива. Используется при чтении времени потоков из файла (определяет, массив какой длины нужен, чтобы все данные из файла в него уместились) Используется в циклах сортировки пузырьком, в которых принимает значение длины массива. С помощью этой переменной определяется, надо ли выводить результаты работы на экран (если ее значение не превышает 21, то результат работы выводится на экран).

**counter\_per, counter\_sr, counter\_iter : integer; -** счетчики перестановок, сравнений и итераций в сортировках.

**Number\_Inter, Ost\_Time\_CP :integer; -** счетчик итераций в алгоритме бинарного поиска, и переменная в которую сохраняется оставшийся квант времени ЦП после нахождения потока, время выполнение которого было наиболее близко к исходному кванту времени ЦП.

**str1, str2, str3, pr:string;**

**str1** –строка-полоска разделяющая таблицу вывода результата работы.

**str2** –чистая строка, в которой нет никаких данных, ее значение присваивается str3 после нахождения всех потоков которые могут быть выполнены за заданный квант времени ЦП и вывода данных на экран;

**str3 –** основная строка таблицы, выводящей результаты работы программы;

pr – переменная используемая для заполнения str3.

**n1:byte; -** переменная используемая в процедуре чтения данных из файла.

**F: Text;-** файл из которого происходит чтение времени потоков;

**Cp\_File:text; -** файл из которого происходит чтение кванта ЦП;

**Q: Text; -** файл, в который осуществляется вывод результата работы программы.

**Err\_1, Sort\_Repeat, Time\_CP\_GEN, Time\_CP\_HAND, Time\_CP\_FILE, CP, Output\_File, Ex\_Quit :boolean;**

**Err\_1 –** переменная определяющая произошла ли ошибка при вводе данных пользователем;

**Sort\_Repeat–** переменная определяющая нужно ли повторить сортировку с исходным массивом;

**Time\_CP\_GEN, Time\_CP\_HAND, Time\_CP\_FILE –** переменные определяющие то как должен генерироваться квант времени ЦП;

**Output\_File –** переменная определяющая должен ли быть осуществлен вывод данных в файл;

**Ex\_Quit –** переменная запрашивающая выход из программы;

**CP –** переменная используемая в бинарном поиске.

**Описание алгоритмов**

Программа работает по следующему алгоритму:

* Запрашивается способ задания времени потоков;
* Запрашивается количество потоков;
* Запрашивается метод сортировки;
* Запрашивается способ задания времени ЦП;
* Выполняется бинарный поиск;
* Вывод результатов работы программы.

**Сортировка пузырьком** *(см. Приложение – Сортировка пузырьком).*

Или **сортировка простыми обменами**. Принцип действий: массив обходится от начала до конца, и в нем меняются местами не отсортированные соседние элементы. В результате первого прохода на последнее место «всплывёт» максимальный элемент. Теперь снова обходим не отсортированную часть массива (от первого элемента до предпоследнего) и меняем по пути не отсортированных соседей. Второй по величине элемент окажется на предпоследнем месте. Продолжая в том же духе, будем обходить всё уменьшающуюся не отсортированную часть массива, перемещая найденные максимумы в конец.

**Сортировка Шелла*.*** *(см. Приложение – Сортировка Шелла).*

Сортировка Шелла – алгоритм сортировки, являющийся усовершенствованным вариантом сортировки вставками. Идея метода Шелла состоит в сравнении элементов, стоящих не только рядом, но и на определённом расстоянии **d** друг от друга. Иными словами – это сортировка вставками с предварительными «грубыми» проходами. В лабораторной работе осуществлена сортировка Шелла последовательностью Хиббарда (длина промежутков dмежду сравниваемыми элементами равна d[n]:=2\*d[n-1]+1) Получается последовательность вида 1,3,7,15…

**Сортировка подсчетом** *(см. Приложение – Сортировка подсчетом).*

Сортировка подсчётом – алгоритм сортировки, в котором используется диапазон чисел сортируемого массива (списка) для подсчёта совпадающих элементов. Применение сортировки подсчётом целесообразно лишь тогда, когда сортируемые числа имеют (или их можно отобразить в) диапазон возможных значений, который достаточно мал по сравнению с сортируемым множеством, например, миллион натуральных чисел меньших 1000.

Для сортировки необходимо создание дополнительного массива из нулей, длина которого задается максимальным числом встречающимся в массиве. В нашем случае длина вспомогательного массива arr\_sort\_countбудет равна 256.  
Алгоритм сортировки следующий:

* Последовательно пройти по массиву B и записать в Arr\_Sort\_Count[B[i]] количество чисел равных B[i];
* В цикле от I=1 до 255 пройти по массиву Arr\_Sort\_Count и записать число I в массив B arr\_sort\_count[i] раз.

**Бинарный поиск** *(см. Приложение – Бинарный поиск).*

Бинарный поиск – Алгоритм поиска элемента в отсортированном массиве.

* Сначала делим массив пополам и сравниваем элемент, который ищем с элементом в середине. Если они равны, то элемент, который мы ищем найден.
* Если же элемент, который мы ищем, меньше элемента в середине массива, то продолжим поиск в левой части массива (для массива сортированного по возрастанию), если элемент больше, то поиск следует продолжить в правой части.
* Далее, выбранную часть массива опять делим пополам и выполняем сравнение. Повторяем действия 2,3 до тех пор, пока искомый элемент не будет найден или пока не пройдем массив до конца. Если искомый элемент и элемент в массиве не совпали, выбирается первый элемент больше чем заданный.

**Заключение**

В программе реализован интерфейс осуществляющий общение с пользователем, программа умеет определять некорректный ввод и предлагать ввести данные заново. Также, реализована возможность выбора метода сортировки (сортировка пузырьком, сортировка Шелла (последовательность Хиббарда), сортировка подсчётом), способа задания времени потоков и кванта ЦП 3 разными способами (ручной ввод, генерация случайным образом, чтение из файла), реализован алгоритм бинарного поиска.

**Литература**

* Сортировка Шелла - [<https://ru.wikipedia.org/wiki/> Сортировка\_Шелла]
* Методы сортировки -[<http://habrahabr.ru/post/204968/>]
* Пузырьковая сортировка и все-все-все - [<http://habrahabr.ru/post/204600/>]
* Алгоритмы сортировки. Часть 1 - [<http://www.compdoc.ru/prog/pascal/algorithms_of_sorting_part1/>]
* Алгоритмы сортировки. Часть 2 - [<http://www.compdoc.ru/prog/pascal/algorithms_of_sorting_part2/>]
* Бинарный поиск -[<https://ru.wikipedia.org/wiki> Двоичный\_поиск]
* Я не могу написать бинарный поиск - [<http://habrahabr.ru/post/146228/>]
* Учебник по языку программирования Pascal -[<http://pas1.ru/pascaltextbook>]

**Приложения**

***Сортировка пузырьком***

**procedure** SORT\_bubble;

**var**

i, j: integer;

**begin**

counter\_iter := 0;

counter\_sr := 0;

counter\_per := 0;

Length\_Arr := High(b);

textcolor(10);

Writeln;

Writeln('<<Подключена процедура сортировки пузырьком>>');

Textcolor(7);

t1 := milliseconds;

**for** i := 0 **to** Length\_Arr - 1 **do**

**begin**

counter\_iter := counter\_iter + 1;

**for** j := 0 **to** (Length\_Arr - 1) - i **do**

**begin**

counter\_sr := counter\_sr + 1;

**if** b[j] > b[j + 1] **then**

**begin**

z := b[j];

b[j] := b[j + 1];

counter\_per := counter\_per + 1;

b[j + 1] := z;

k := Number\_El[j];

Number\_El[j] := Number\_El[j + 1];

Number\_El[j + 1] := k;

**end**;

**end**;

**end**;

textcolor(10);

Writeln('<<Конец процедуры... >>');

Textcolor(7);

t2 := milliseconds;

**end**;

***Сортировка Шелла (Последовательность Хиббарда)***

**procedure**SORT\_Shell;

**var**

d: **array** [0..29] **of** integer := (1, 3, 7, 15, 31, 63, 127, 255, 511, 1023, 2047, 4095, 8191, 16383, 32767, 65535, 131071, 262143, 524287, 1048575, 2097151, 4194303, 8388607, 16777215, 33554431, 67108863, 134217727, 268435455, 536870911, 1073741823);

g, w: integer;

c: boolean;

**begin**

counter\_iter := 0;

counter\_sr := 0;

counter\_per := 0;

t1 := 0;

t2 := 0;

textcolor(10);

Writeln;

Writeln('<< Подключена процедура сортировки Шелла >>');

Textcolor(7);

t1 := milliseconds;

**while** d[w + 1] <= Length(b) **do**

inc(w);

**repeat**

g := d[w];

i := g;

**repeat**

j := i - g;

**if** j + g >= Length(b) **then break**;

c := True;

**repeat**

counter\_sr := counter\_sr + 1;

**if** b[j] <= b[(j + g)] **then**

c := False

**else**

**begin**

z := b[j];

b[j] := b[j + g];

b[j + g] := z;

k := Number\_El[j];

Number\_El[j] := Number\_El[j + g];

counter\_per := counter\_per + 1;

Number\_El[j + g] := k;

**end**;

dec(j, g);

**until not** ((j >= 0) **and** C);

inc(i);

**until not** (i <= Length(b));

dec(w);

**until not** (w <> -1);

t2 := milliseconds;

textcolor(10);

Writeln('<<Конец процедуры... >>');

Textcolor(7);

**end**;

***Сортировка подсчётом***

**procedure** SORT\_Count;

**var**

arr\_sort\_count, place: **array of** byte;

i, j, k: integer;

**begin**

counter\_iter := 0;

counter\_sr := 0;

counter\_per := 0;

t1 := 0;

t2 := 0;

t1 := milliseconds;

textcolor(10);

Writeln;

Writeln('<< Подключена процедура сортировки подсчётом >>');

Textcolor(7);

setlength(arr\_sort\_count, 256);

setlength(place, 257);

setlength(Number\_El, High(b) + 1);

**for** i := 0 **to** High(b) **do**

inc(Arr\_sort\_count[b[i]]);

**for** i := 1 **to** 255 **do**

place[i] := place[i - 1] + Arr\_sort\_count[i];

**for** i := 0 **to** High(b) **do**

**begin**

**if** place[b[i]] <>0 **then** //Если будет = 0 вылезет Outside index

**begin**

Number\_El[place[b[i]] - 1] := i;

dec(place[b[i]]);

**end**

**end**;

**for** i := 1 **to** 255 **do**

**for** j := 1 **to** Arr\_sort\_count[i] **do**

**begin**

b[k] := i;

inc(k);

**end**;

textcolor(10);

Writeln('<<Конец процедуры... >>');

Textcolor(7);

t2 := milliseconds;

arr\_sort\_count := nil; //Очистка памяти

place := nil;

**end**;

***Бинарный поиск***

**procedure** Binary\_Search;

**begin**

cp := false;

**if** Time\_CP >= B[0] **then**

**begin**

cp := true;

left := 0; right := s;

**while** left <= right **do**

**begin**

mid := (left + right) **div** 2;

**if** Time\_CP < B[mid] **then**

right := mid - 1

**else**

**if** ((mid) <= S) **and** (Time\_CP >= B[mid]) **then**

left := mid + 1

**else**

**begin**

left := mid + 1;

right := mid;

**if** Time\_CP = B[mid] **then**

**begin**

left := mid + 1;

right := mid;

**end**;

**end**;

**if** time\_cp - b[mid] <0 **then**

mid := mid - 1;

**end**;

**end**;

**end**;