



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ  
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ЛИПЕЦКИЙ  
ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Институт компьютерных наук  
Кафедра автоматизированных систем управления

Лабораторная работа №3  
по операционным системам

Студент      АС-21-1      \_\_\_\_\_      Станиславчук С. М.  
(подпись, дата)

Руководитель  
Доцент, к.п.н.      \_\_\_\_\_      Кургасов В. В.  
(подпись, дата)

Липецк 2024

## Содержание

2. Цель работы

3. Задание

4. Программа 1 (демон)

5. Программа 2 (корутина)

6. Вывод

## 2. Цель работы

Получение практических навыков по работе с подсистемой инициализации и управления службами.

### 3. Задание

Задание 1. Написать демон, представляющий собой программу для отдыха глаз. Демон будет показывать уведомление о начале отдыха раз в заданный промежуток времени (`repetition_period`) и уведомление об окончании отдыха через заданный промежуток времени (`relax_time`). Временные промежутки задаются случайным образом

Вариант 2: Стереокартинки

### Задание 2. Корутина

На диске лежат файлы, в них числа в строковом виде, в произвольном порядке, они разделены пробелами. Нужно отсортировать каждый файл и затем слить их в один большой.

То есть выполнить сортировку слиянием.

Реализовать задачу с использованием корутины.

Правила:

- Каждый файл надо сортировать в отдельной корутине. Эта часть задания нацелена на понимание кооперативного планирования задач.

- Файлы имеют кодировку ASCII. То есть это обычный текст, не unicode или что-то такое.

- Необходимо замерить время выполнения операций, используя `clock_gettime(CLOCK_MONOTONIC)`.

Ограничения:

- Глобальные переменные запрещены (кроме уже существующих).

- Для сортировки нельзя использовать встроенные функции типа `qsort()`, `system("sort ...")` и т.д.

- Сложность сортировки индивидуальных файлов должна быть  $O(N^2)$  (например, нельзя использовать сортировку пузырьком).

- Суммарное время работы всей программы ограничено. (Для теста можно использовать 6 файлов, каждый с 40к чисел, время не должно занимать больше одной секунды).

- Работа с файлами должна быть

- либо через числовые файловые дескрипторы и функции `open()` / `read()` / `write()` / `close()`,

- либо через `FILE*` дескрипторы и функции `fopen()` / `fscanf()` / `fprintf()` / `fclose()`. Нельзя

- использовать `std::iostream`, `std::ostream`, `std::istream` и прочий STL.

- корутины должны переключаться. Делать так называемые 'yield', 'илды'.

Послабления:

- Числа помещаются в 'int'.
- Можно полагать, что все файлы помещаются целиком в память, даже все одновременно.
- Финальный шаг - само слияние сортированных массивов – можно делать прямо в `main()` снаружи от корутин.

Советы:

- Вы можете найти больше информации о разных новых функциях при помощи консольной команды 'man'. Например, 'man read' (или 'man 2 read') напечатает документацию функции 'read()'.

'man strdup' расскажет больше про функцию 'strdup()'. Таким же образом можно искать другие встроенные функции.

- Шаги, которым можно следовать, если не знаете, с чего начать:
- Реализовать обычную сортировку одного файла. Без корутин, без множества файлов.

Просто прочитать и отсортировать один файл.

Протестируйте этот код.

- Расширьте свой код, чтобы теперь он сортировал много файлов через сортировку слиянием. Без корутин. Проверьте свой код на реальных тестах из задания. Когда он заработает, вы сможете сконцентрироваться на добавлении корутин, и не тратить время на отладку одновременно и корутин, и сортировки, и работы с файлами.

- Встройте в свой код корутины.

Предусмотреть в выводе: суммарное время работы программы, время работы и количество переключений каждой корутины. Учесть, что время работы корутины не включает ее время ожидания. То есть пока она спала во время `coro_yield()`. Вы должны остановить таймер работы корутины прямо перед `coro_yield()` и возобновить его сразу после.

#### 4. Программа 1.

Для написания daemon (в моем случае я использую ОС Arch Linux) потребуются следующие компоненты:

1. dbus – шина для возможности получения и отрисовки системных уведомлений
2. python – язык программирования, на котором будет написан демон
3. python-модули:
  1. plyer - для работы с уведомлениями
  2. PIL – для возможности открывать изображения

Код программы:

```
import time
import random
from plyer import notification
from PIL import Image

def show_images(images):
    for image in images:
        img = Image.open(image)
        img.show()

def eye_rest_demon(repetition_period, relax_time, images):
    while True:
        # Случайный выбор времени для начала отдыха
        start_rest_time = random.randint(1, repetition_period)
        # Случайный выбор времени для продолжительности отдыха
        duration_rest = random.randint(1, relax_time)

        # Ожидание начала отдыха
        time.sleep(start_rest_time)

        # Уведомление о начале отдыха
        notification.notify(
            title="Отдых для глаз",
            message="Пора отдохнуть от экрана! Время отдыха: {} мин.".format(duration_rest),
            timeout=10
        )

        # Показ изображений
        show_images(images)

        # Ожидание окончания отдыха
        time.sleep(duration_rest * 60)

        # Закрытие всех открытых изображений
        Image.close()

        # Уведомление об окончании отдыха
        notification.notify(
            title="Время вернуться к работе",
            message="Отдых закончен. Время вернуться к работе!",
            timeout=10
        )

if __name__ == "__main__":
    repetition_period = 60 # Периодичность проверки, в секундах
    relax_time = 2 # Продолжительность отдыха, в минутах
    images = ["image_1.png", "image_2.jpg", "image_3.jpg", "image_4.jpg", "image_5.jpg"]
    eye_rest_demon(repetition_period, relax_time, images)
```

Запустим daemon:

```
stanik@archlinux:~/programmer/++Programmer/3 курс 2/Операционные систем...  
[stanik@archlinux lr3]$ python daemon.py
```

Рисунок 1 - Запуск daemon

Спустя минуту (заданное для тестирования значение переменной) пришло уведомление в правом верхнем углу экрана (по умолчанию):

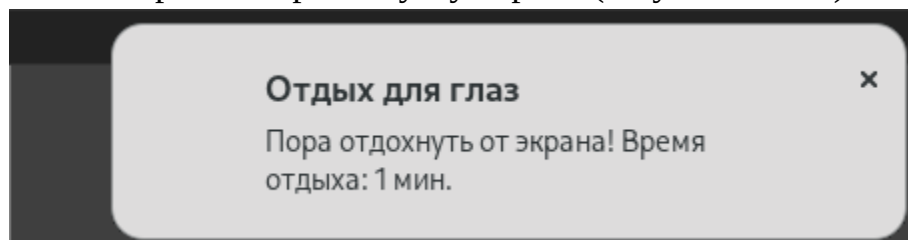


Рисунок 2 - Вызов и отрисовка уведомления, в данном случае с одноминутным отдыхом

Вывод изображений (отдых для глаз) в отдельном окне, в моем случае – Google Chrome

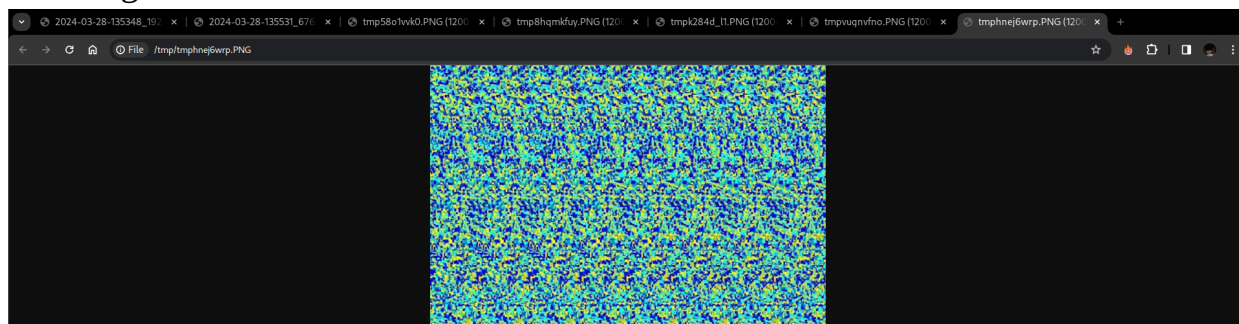


Рисунок 3 – Одно из открывшихся изображений (Стереокартинка)

Демон работает в бесконечном цикле и будет напоминать пользователю об отдыхе каждые  $\text{rand}(n, m)$  минут.

Что делает этот обычный скрипт демоном? - Лишь формат процесса. Демон – это точно такая же программа, она лишь выполняется в фоновом процессе. То есть, теперь, чтобы сделать скрипт демоном нужно либо:

запустить данную программу, используя ‘&’ или же прописать нужные параметры в конфиг файле systemd. Допустим, мы хотим чтобы этот демон тихо работал и включался каждый раз при входе в систему. Тогда делаем так: Перейдем в каталог `/etc/systemd/system/` и создаем файл расширением `.service`, например, `mydaemon.service` со следующими параметрами:

```
[Unit]
Description=My Daemon Service

[Service]
ExecStart=/usr/bin/mydaemon
Restart=always

[Install]
WantedBy=multi-user.target
```

Теперь наш скрипт стал демоном, который работает всегда в фоне и его совсем не нужно запускать вручную, так как он сам инициализируется и запускается в фоне при старте системы.

## Программа 2.

Реализую данный алгоритм в Unity, C#. В чем суть? Надо написать пару скриптов: один будет генерировать 6 текстовых файлов, которые будут хранить  $240\,000 / 6 \rightarrow$  по 40 000 случайных чисел; второй скрипт будет создавать корутины для каждого процесса сортировки файлов слиянием; третий же скрипт будет хранить используемый во втором метод сортировки.

### Скрипт, генерирующий файлы:

```
using System.Collections;
using System.Collections.Generic;
using System.IO;
using UnityEngine;

public class FileGenerator : MonoBehaviour
{
    private const uint numFiles = 6;
    private const int numNumbers = 40000;
    private const int maxNumberRoofRange = numNumbers * 100;

    private void Awake()
    {
        GenerateFilesRand();
    }

    private void GenerateFilesRand()
```



```

    {
        for (int i = 0; i < numFiles; i++)
        {
            string fileName = Path.Combine(Application.persistentDataPath, $"file{i+1}.txt");
            GenerateFileRand(fileName, numNumbers);
        }
    }

    private void GenerateFileRand(string fileName, int numNumbers)
    {
        using (StreamWriter writer = new StreamWriter(fileName))
        {
            for (int i = 0; i < numNumbers; i++)
            {
                writer.Write(UnityEngine.Random.Range(0, maxNumberRoofRange));
                if (i < numNumbers - 1)
                    writer.Write(" ");
            }
        }

        Debug.Log("File with random int's generated: " + fileName);
    }

    public void SaveOutputFile(int[] arr)
    {
        // Путь к файлу, куда будем сохранять массив
        string filePath = Path.Combine(Application.persistentDataPath, "output.txt");

        // Создаем новый файл или перезаписываем существующий
        using (StreamWriter writer = new StreamWriter(filePath))
        {
            // Записываем каждый элемент массива в файл
            foreach (int num in arr)
            {
                writer.WriteLine(num.ToString());
            }
        }

        Debug.Log("Array saved to file: " + filePath);
    }
}

```

## Скрипт, создающий сортирующие корутины, с таймерами:

```

using UnityEngine;
using System.Collections;
using System.IO;
using System;
using System.Diagnostics;
using System.Collections.Generic;
using static SortingAlgorithm;

public class ParallelFileSorting : MonoBehaviour
{
    private List<int[]> sortedArrays = new List<int[]>();
    private List<int[]> mergedArrays = new List<int[]>();

    private double totalSortingTime = 0.0;
    private double totalIndSortTime = 0.0;
    private int finishedCoroutines = 0;
    private readonly object lockObject = new object();

    [SerializeField] private FileGenerator fileGen;

    private void Awake()
    {
    }
}

```

```

private void Start()
{
    UnityEngine.Debug.Log("<b>Parallel Sorting</b>");
    StartCoroutine(SortFiles());
}

private IEnumerator SortFiles()
{
    string[] files = Directory.GetFiles(Application.persistentDataPath, "*.txt");
    List<Coroutine> coroutines = new List<Coroutine>();

    Stopwatch stopwatch = Stopwatch.StartNew();

    foreach (string file in files)
    {
        coroutines.Add(StartCoroutine(SortFileCoroutine(file)));
    }
    stopwatch.Stop();
    TimeSpan timeSpan = stopwatch.Elapsed;

    UnityEngine.Debug.Log($"Sorting files (with self-Coroutine for each file) took
    <b>{timeSpan.TotalMilliseconds}</b> ms");

    // Ждем завершения всех корутин
    foreach (Coroutine coroutine in coroutines)
    {
        yield return coroutine;
    }

    // Выполняем слияние отсортированных массивов
    mergedArrays.Add(sortedArrays[0]);
    for (int i = 1; i < sortedArrays.Count; i++)
    {
        int[] merged = Merge(mergedArrays[i-1], sortedArrays[i]);
        mergedArrays.Add(merged);
    }

    // Выводим результат слияния
    UnityEngine.Debug.Log("Merged Array: " + string.Join(", ", mergedArrays[mergedArrays.Count - 1]));
    fileGen.SaveOutputFile(mergedArrays[mergedArrays.Count - 1]);
    UnityEngine.Debug.Log("Merged Array written to output.txt");

    UnityEngine.Debug.Log("totalSortingTime = " + totalSortingTime);
    yield return null;
}

private IEnumerator SortFileCoroutine(string filePath)
{
    int[] array = ParseAndSort(filePath);
    sortedArrays.Add(array);

    // Если все корутины завершились, можно продолжить
    while (finishedCoroutines == sortedArrays.Capacity)
    {
        yield break;
    }
}

private int[] ParseAndSort(string filePath)
{
    // Загружаем данные из файла
    string data = File.ReadAllText(filePath);
    string[] stringNumbers = data.Split(' ');

    // Преобразуем данные в массив чисел
    List<int> numbersList = new List<int>();
    foreach (string str in stringNumbers)
    {
        if (int.TryParse(str, out int number))
        {
            numbersList.Add(number);
        }
        else

```

```

        {
            UnityEngine.Debug.LogWarning($"Unable to parse '{str}' as integer.");
        }
    }

    // Преобразуем список чисел в массив
    int[] array = numbersList.ToArray();

    // Засекаем время начала сортировки
    Stopwatch stopwatch = Stopwatch.StartNew();
    // Сортируем массив чисел
    SortingAlgorithm.MergeSort(array);
    // Засекаем время окончания сортировки
    stopwatch.Stop();

    TimeSpan timeSpan = stopwatch.Elapsed;
    UnityEngine.Debug.Log($"{filePath} Sorting took <b>{timeSpan.TotalMilliseconds}</b> ms");

    UnityEngine.Debug.Log("Array lenght = " + numbersList.Count);
    totalSortingTime += timeSpan.TotalMilliseconds;
    return array;
}
}

```

### Скрипт #3 содержит алгоритм сортировки:

```

public static class SortingAlgorithm
{
    public static void MergeSort(int[] array)
    {
        if (array == null || array.Length <= 1)
            return;

        MergeSortRecursive(array, 0, array.Length - 1);
    }

    public static int[] Merge(int[] arr1, int[] arr2)
    {
        int[] merged = new int[arr1.Length + arr2.Length];
        int i = 0, j = 0, k = 0;

        while (i < arr1.Length && j < arr2.Length)
        {
            if (arr1[i] < arr2[j])
            {
                merged[k] = arr1[i];
                i++;
            }
            else
            {
                merged[k] = arr2[j];
                j++;
            }
            k++;
        }

        while (i < arr1.Length)
        {
            merged[k] = arr1[i];
            i++;
            k++;
        }

        while (j < arr2.Length)
        {
            merged[k] = arr2[j];
            j++;
            k++;
        }
    }
}

```

```

        return merged;
    }

private static void MergeSortRecursive(int[] array, int left, int right)
{
    if (left < right)
    {
        int mid = (left + right) / 2;

        MergeSortRecursive(array, left, mid);
        MergeSortRecursive(array, mid + 1, right);

        Merge(array, left, mid, right);
    }
}

private static void Merge(int[] array, int left, int mid, int right)
{
    int n1 = mid - left + 1;
    int n2 = right - mid;

    int[] leftArray = new int[n1];
    int[] rightArray = new int[n2];

    for (int i = 0; i < n1; ++i)
        leftArray[i] = array[left + i];
    for (int j = 0; j < n2; ++j)
        rightArray[j] = array[mid + 1 + j];

    int p = 0, q = 0, k = left;
    while (p < n1 && q < n2)
    {
        if (leftArray[p] <= rightArray[q])
        {
            array[k] = leftArray[p];
            p++;
        }
        else
        {
            array[k] = rightArray[q];
            q++;
        }
        k++;
    }

    while (p < n1)
    {
        array[k] = leftArray[p];
        p++;
        k++;
    }

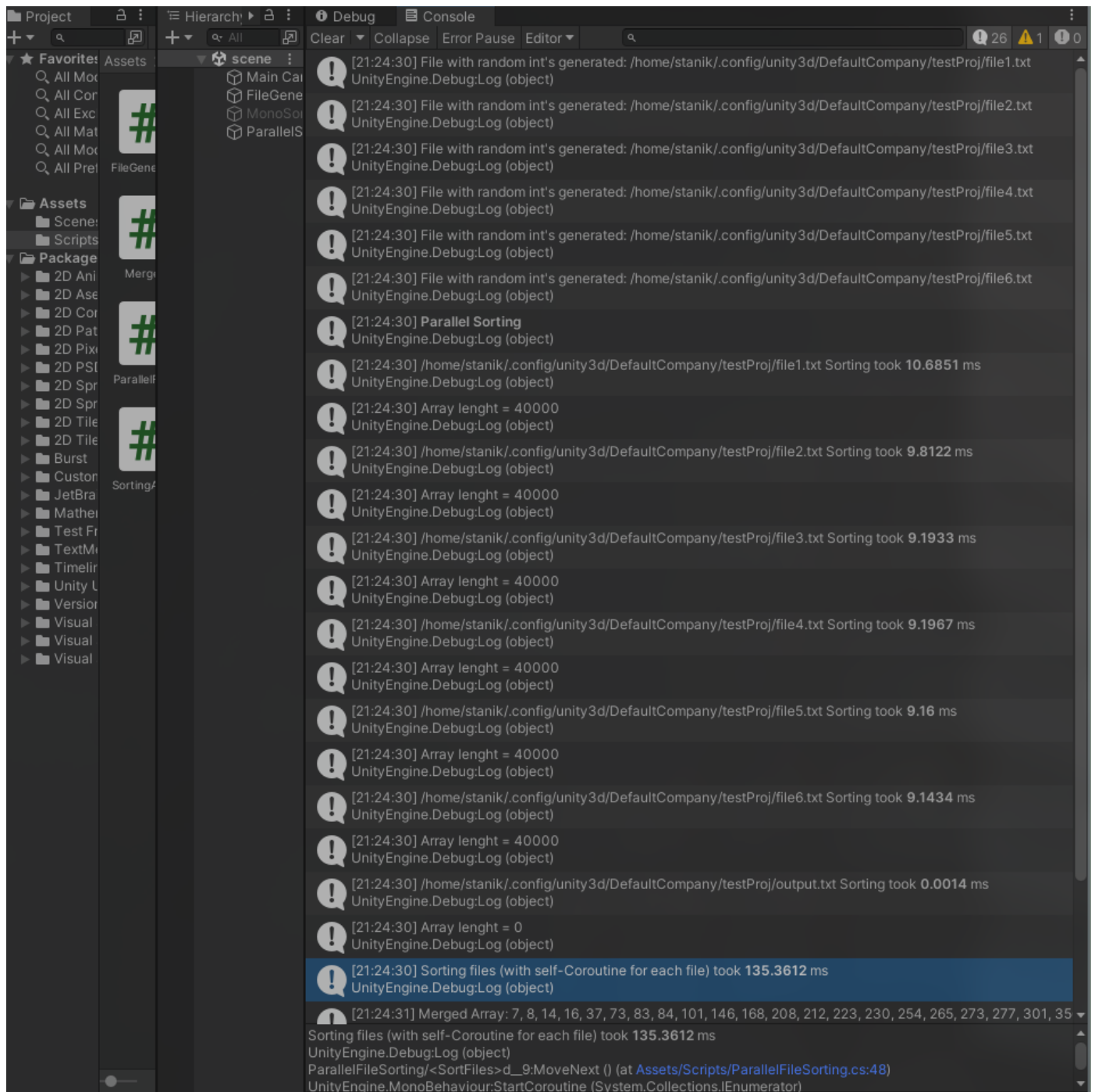
    while (q < n2)
    {
        array[k] = rightArray[q];
        q++;
        k++;
    }
}
}

```

Теперь протестируем программу.

Создадим файлы с названиями file\_n.txt и output.txt

Запускаем программу и смотрим лог.



Как видно из логов, на сортировку каждого файла, а также их слияние в один большой файл, размером 240 000 целочисленных значений, без учета времени генерации файлов с числами, заняло около 135 миллисекунд, что

является более чем успешным результатом. Таким образом, используя корутины, я задействовал сразу несколько потоков своего процессора, чтобы отсортировать конечный файл за максимально малое время.

Пример сгенерированного файла file\_1.txt:

```
file1.txt (~/config/unity3d/DefaultCompany/testProj) - VIM
1 2954555 564228 2777914 169852 3614195 2855441 1102262 2689604 300718 2847556 1013145 150888
7 3995463 309693 657292 1909132 3038417 2457319 3578658 2201159 3413518 42383 2422194 25912
31 3482372 3045787 455337 121958 2493802 2398515 2410478 2681003 2577499 3818973 2659509 75
35 1444372 2454634 1773466 236314 927 1067488 2078443 2142436 763165 3149527 2383111 247672
1 3901177 1987011 2958993 3092365 3698112 846917 2968595 1267361 3276737 1328348 3885694 23
51208 2570138 433626 2652697 3095147 1838342 3302899 769391 1294047 323274 2479748 829626 2
600883 3268534 3562792 3808752 884362 3066741 2765231 433207 3123260 3199386 1559191 145184
9 919460 1767518 698175 3340733 3232669 1174515 1514923 3618302 2838964 1088538 1363863 395
3187 769686 2225313 3044295 1761148 3070601 2538490 2038138 940467 3129602 3317778 3983080
545405 3646828 1454722 234084 1200331 2505791 320244 964791 3557338 472959 721357 3606321 2
285460 1398962 2270606 2114484 3043939 1203531 2376482 137521 3727018 2952958 3543962 26151
20 37528 1447075 863794 310766 3678792 748024 2280318 1574846 3953271 1005433 2350927 99475
2 3669230 3948823 3703468 3132736 1040162 551013 2798987 1196260 2675780 3329369 3319178 16
57618 493720 3559189 1199569 657084 3365100 254989 1206953 3844746 662829 520456 2773937 32
13123 650001 1166549 2827689 3136505 2020486 488377 2016461 1701809 1886808 1039143 406823
3782539 360011 222924 2929740 1328158 2273492 2267504 840805 3084541 471903 2247171 674849
3505703 2376100 3020619 3290046 3325958 819066 1553148 1092773 3453215 2463912 388035 1414
031 3824269 3448179 2193474 252746 1685550 2378040 2190329 3478675 2825180 1716495 2127976
2320138 72056 2459116 3913088 1206847 519565 3475967 1114828 466478 593110 763394 2019846 6
08947 694813 2490861 82545 2641274 2203 3703854 1011347 1849227 3151610 2067204 2060972 353
524 2719493 2613111 2862469 1014387 3085136 1493299 2421257 3475323 2912001 3696061 727939
3150583 3054265 2502643 2346934 1989568 1440751 1680313 3941239 127058 3982534 1357614 1796
057 2399477 1615534 2300933 1594356 685863 661669 777233 3357055 1132592 922843 918615 3224
904 2131149 228106 2915147 467199 3436970 725945 2550119 2745617 1461685 2995322 2014094 82
2540 3243444 226353 1034906 399021 1256012 702549 3395856 3272835 1439340 2028676 836723 13
88427 3470171 3922502 2545986 3550435 2577418 779765 591361 1050710 2479222 3193469 75023 4
86223 3546511 849574 3135048 3245702 3555626 3300858 2339440 2049113 1172203 1915127 107211
3 957443 3366750 3514267 3880302 2335014 940524 3233299 3100672 2600103 1541550 2542955 122
0804 3105614 2820726 347138 1271908 2004659 110717 1151332 3147750 1151553 1982727 2125400
3905616 556195 1452432 1749507 1527696 1860868 1109874 1022057 492947 156013 2776337 379702
1 1001330 355524 1564241 3960399 749005 670061 3307101 3151640 3657531 572684 1654717 13052
73 1395364 2334508 821623 1144545 2132695 197854 2634025 2428765 906927 3885460 1655872 317
3650 2406354 2309622 533807 2453764 2322175 3137053 3790375 2247008 777281 390838 161630 28
75958 1033876 2869267 1470441 2457395 3662420 484402 3083392 2846117 2002880 2461346 227615
4 365269 2464061 1439502 1838652 2117852 890748 3736260 1735858 2454786 1180769 3545653 269
2507 3722416 2094517 2609974 590400 731011 3386391 2324116 3998634 2183935 1163140 2164686
279292 1690482 2438580 2551815 2355129 2036125 2517615 1367486 2415010 3883132 3588964 2689
722 2139756 2397413 547142 399834 3126735 1118462 1938626 340976 1613183 3992989 3063443 36
73493 1423799 2609270 2601893 2152083 2277618 1070725 2259773 833697 3686616 427729 2044269
3683363 1537379 3459439 3724733 2890229 657911 735864 2186542 3884944 814159 3110780 13526
83 1190375 3813485 1478735 624168 3700004 3610304 3346013 1170863 2310478 396576 1327880 37
96038 3181410 2970800 1800549 2408753 3617890 1349742 3445898 2411359 488662 963777 363059
2832287 1287581 1226576 902197 3998060 3842412 1479257 1914460 3553496 2004939 1681875 9576
6 3479548 504820 2816857 1258820 3740365 2461627 3179933 187894 3090907 554894 3123696 2648
470 85543 2918658 1194591 3256767 795565 194621 145102 45898 1731745 1639413 2094194 369791
7 1870985 2420996 2578677 681505 201897 2408630 3384865 871326 1407459 1297114 3540923 2752
536 1431002 3698810 1430658 357468 2478804 1233767 2065238 3911088 3762603 3996079 3966474
748887 1025505 3272071 2445065 2593168 2817566 413700 1092915 1919183 1042959 100308 246738
8 1857961 1319917 3412429 2703258 2843665 2961547 2243230 3617758 2374103 1758664 2641981 3
177477 2122719 3507188 3771251 3093263 125100 1605899 2382403 638234 3624009 599601 1793061
1::1
```

Полученный в результате сортировки файл:

```
output.txt (~/.config/unity3d/DefaultCompany/testProj) - VIM
1 7
2 8
3 14
4 16
5 37
6 73
7 83
8 84
9 101
10 146
11 168
12 208
13 212
14 223
15 230
16 254
17 265
18 273
19 277
20 301
21 356
22 435
23 435
24 459
25 558
26 564
27 568
28 568
29 586
30 590
31 629
32 629
33 635
34 644
35 644
36 684
37 688
38 700
39 710
40 725
41 732
42 744
43 777
44 782
45 786
46 831
47 832
48 864
49 874
50 878

1::1
```

Вывод: в ходе выполненной работы создал демон-процесс и написал программу, параллельно сортирующую несколько файлов, после чего объединяет их в один.