Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа интеллектуальных систем и суперкомпьютерных технологий

**КУРСОВАЯ РАБОТА**

**Работа с потоками в ОС Linux. Реализация алгоритмов сортировки для массива слияниями. Параллельный процесс сортировки по**

**возрастанию.**

по дисциплине «Операционные системы»

Выполнил

студент гр. 3530901/90201 <*подпись*> Фам Ба Нам

Руководитель

Старший преподаватель <*подпись*> В.Э. Шмаков

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2021 г.

Санкт-Петербург

2021

**ЗАДАНИЕ**

**НА ВЫПолнение курсовой работы**

студенту группы 3530901/90201 Дергачеву Максиму Владимировичу

*(номер группы) (фамилия, имя, отчество)*

***1. Тема работы:*** Реализация параллельной сортировки массива слияниями в ОС Linux

***2. Срок сдачи студентом законченной работы:***  10 декабря 2021

***3. Исходные данные к работе***: нет

***4. Содержание пояснительной записки*** (перечень подлежащих разработке вопросов): введение, основная часть (раскрывается структура основной части), заключение, список использованных источников, приложения.

Примерный объём пояснительной записки 12 страниц машинописного текста.

***5. Перечень графического материала*** (с указанием обязательных чертежей и плакатов): нет

***6. Консультанты*** нет

***7. Дата получения задания***: «01» октября 2021 г.

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.Э. Шмаков

*(подпись) (инициалы, фамилия)*

Задание принял к исполнению \_\_\_\_\_\_\_\_\_\_\_\_\_ Фам Ба Нам

*(подпись студента) (инициалы, фамилия)*

\_\_\_\_\_\_\_\_\_\_\_\_

*(дата)*

**Содержание**

[1. Введение 4](#_Toc89803825)

[2. Описание алгоритма и реализация 4](#_Toc89803826)

[3. Результаты работы и сравнение временных затрат 7](#_Toc89803827)

[Вывод 9](#_Toc89803828)

[Приложение 10](#_Toc89803829)

# **Введение**

Сортировка слиянием — алгоритм сортировки, который упорядочивает списки в определённом порядке. Эта сортировка — хороший пример использования принципа «разделяй и властвуй». Сначала задача разбивается на несколько подзадач меньшего размера. Затем эти задачи решаются с помощью рекурсивного вызова или непосредственно, если их размер достаточно мал. Наконец, их решения комбинируются, и получается решение исходной задачи.

В ходе работы будут рассмотрены однопоточная реализация сортировки слиянием, многопоточная реализация и оценка временных затрат этих сортировок.

# **Описание алгоритма и реализация**

В алгоритме сортировки слиянием выделяют три шага:

1. Сортируемый массив разбивается на две части примерно одинакового размера;
2. Каждая из получившихся частей сортируется отдельно, например — тем же самым алгоритмом;
3. Два упорядоченных массива половинного размера соединяются в один.

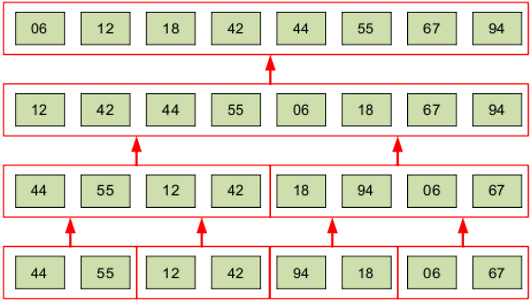


Рис. 1 Алгоритм сортировки

Алгоритм сортировки слиянием основан на идее, что два отсортированных списка можно слить в один отсортированный список за время, равное суммарной длине этих списков

Для этого сравним первые элементы данных списков. Тот элемент, который меньше, скопируем в конец результирующего списка (который первоначально пуст) и в этом списке перейдем к следующему элементу. Будем повторять этот процесс (выбираем из начала двух списков наименьший элемент, копируем его в результирующий список), пока один из исходных списков не кончится. После этого оставшиеся элементы (один из двух исходных списков будет непуст) скопируем в результирующий список.

Сложность этого алгоритма равна O(nlog2 (n))

В результате была написана такая программа

|  |
| --- |
| Листинг 1. Обычная сортировка слиянием |
| def merge(\*args):  left, right = args[0] if len(args) == 1 else args  left\_length, right\_length = len(left), len(right)  left\_index, right\_index = 0, 0  merged = []  while left\_index < left\_length and right\_index < right\_length:  if left[left\_index] <= right[right\_index]:  merged.append(left[left\_index])  left\_index += 1  else:  merged.append(right[right\_index])  right\_index += 1  if left\_index == left\_length:  merged.extend(right[right\_index:])  else:  merged.extend(left[left\_index:])  return merged  def merge\_sort(data):  length = len(data)  if length <= 1:  return data  middle = int(length / 2)  left = merge\_sort(data[:middle])  right = merge\_sort(data[middle:])  return merge(left, right) |

Для параллельной реализации этого алгоритма необходимо в начале разбить исходный массив на столько частей, сколько потоков мы собираемся использовать. Далее каждая часть сортируется обыкновенной сортировкой слиянием. После того, как все части будут отсортированы, их необходимо слить вместе. Для этого используем те же потоки попарно сливая массивы, пока не останется единственный.

|  |
| --- |
| Листинг 2. Параллельная сортировка слиянием. |
| def merge\_sort\_parallel(data, processes):  pool = multiprocessing.Pool(processes=processes)  size = int(math.ceil(float(len(data)) / processes))  data = [data[i \* size:(i + 1) \* size] for i in range(processes)]  data = pool.map(merge\_sort, data)  while len(data) > 1:  extra = data.pop() if len(data) % 2 == 1 else None  data = [(data[i], data[i + 1]) for i in range(0, len(data), 2)]  data = pool.map(merge, data) + ([extra] if extra else [])  return data[0] |

# **Результаты работы и сравнение временных затрат**

В результате работы программ получилась такие данные:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | длина массива | | | | | | | | | суммарное время |
| 1000 | 5000 | 10000 | 50000 | 100000 | 500000 | 1000000 | 5000000 | 10000000 |
| количество потоков | 1 | 0.002 | 0.011 | 0.024 | 0.139 | 0.289 | 1.759 | 3.672 | 21.981 | 47.006 | 74.883 |
| 2 | 0.212 | 0.015 | 0.021 | 0.087 | 0.181 | 1.012 | 2.158 | 12.213 | 26.152 | 42.051 |
| 4 | 0.164 | 0.019 | 0.023 | 0.068 | 0.130 | 0.698 | 1.475 | 8.974 | 17.166 | 28.717 |
| 8 | 0.197 | 0.032 | 0.041 | 0.092 | 0.168 | 0.724 | 1.548 | 7.949 | 16.714 | 27.465 |

Таблица 1. Затраченное время на сортировку N-ого массива M-ым количеством потоков.

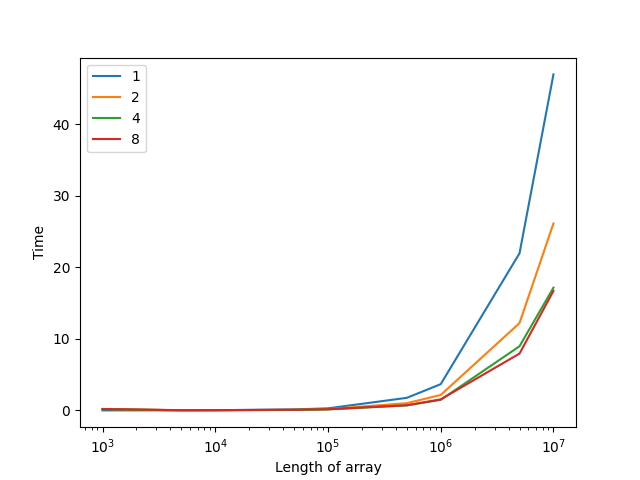


Рис. 2. Графическое представление таблицы 1.

Из результатов видно, что быстрее всего справились с сортировкой 8 потоков. Это объясняется тем, что при работе программы был использован процессор с 4-мя физическими ядрами и 8-ю логическими ядрами. Поскольку каждое логическое ядро может обрабатывать по потоку, становится понятно, почему лучший результат показали 8 потоков.

# **Вывод**

В данной работе были реализованы различные варианты алгоритма сортировки слиянием: однопоточный и многопоточный. Было проведено сравнение временных затрат на сортировку массивов различной длинны различным количеством потоков. Из результатов сравнения видно, что многопоточная реализация (с разумным количеством потоков) справляется лучше.