Отчёт по ИДЗ №4

Кормилицын Владимир Алексеевич, БПИ 226

Вариант 33

Отчёт о выполнении:

Ситуация, моделируемая в программе, может быть описана так:  
 Несколько человек (каждый человек соответствует одному потоку) кодируют куски текста. Текст разделён на равные куски, при этом есть общий “портфель” задач – число (общая переменная), равное последнему не обработанному номеру куска текста. Каждый человек берёт себе очередной номер куска, который начинает шифровать. По окончании шифрования человек кладёт зашифрованные данные в общую переменную (в первой реализации – вставка в отсортированный двусвязный список, во второй реализации – добавление в конец массива с заранее выделенным местом и сортировкой в конце). После этого человек берёт следующий номер куска текста для шифрования. Условие остановки – если номер превысил общее число кусков текста.

В первой реализации (папка version\_1) доступ к “портфелю” происходит за счёт 2 мьютексов – один для взятия номера и второй для вставки закодированного куска текста в двусвязный список. При этом для списка поддерживается инвариант: до вставки список является отсортированным, и после вставки в нужное место он остаётся отсортированным.

Во второй реализации (папка version\_2) доступ к “портфелю” осуществляется за счёт атомарных инструкций: счётчик номера куска текста для обработки и индекс свободного элемента в массиве, куда нужно поместить результат – переменные с типом atomic\_size\_t. Операция инкремента этих переменных делается атомарно, не требует блокировки потоков и не ломает работоспособность программы. Т. к. добавление в конец массива результатов кодировки делается не в отсортированном по номерам кусков порядке, то в конце, после кодирования всей строки, закодированные куски строки сортируются по номерам кусков.

Третья реализация (папка version\_3) сделана с использованием библиотеки openMP и основывается на реализации version 2. Более подробно она описана далее, после скриншотов работы version 1 и version 2.

В обоих реализациях кусок назван чанком (chunk), размера чанка – константа, в данной реализации она равна 128 символам. Т.к. в языке C переменные с квалификатором const формально не считаются константами этапа компиляции, то для использования переменной в качестве размера массива структуры encoded\_result\_t используется макрос. Остальные константы объявлены как переменные с квалификатором const.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

Для ввода данных можно использовать консоль или аргументы командной строки:

* + 1. Если при вызове исполняемого файла указать флаг -in, то следующая за ним строка будет проинтерпретирована как входная строка для шифрования. Если флаг -in не был указан, приложение попросит ввести строку из консоли.
    2. Если при вызове исполняемого файла указать флаг -fin, то следующая за ним строка будет проинтерпретирована как имя файла, из которого необходимо прочитать строку для шифрования
    3. Если флаги не были указаны, программа попросит пользователя ввести строку в консоль.
    4. Если указаны несколько флагов -in и/или -fin, то программа прочитает строку, основываясь на первом по порядку флаге, а на остальные флаги выведет предупреждение:

> No more than one input string can be passed for the encoding

С точки зрения архитектуры приложения (если уместно говорить про архитектуру такого небольшого приложения), первая и вторая версии программы состоят из нескольких файлов:

1. encoder\_config.h – заголовочный файл с конфигурацией приложения:

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

1. encoder\_types.h – файл со структурам, используемыми для выполнения задачи, а также аналогами конструкторов и деструкторов (init\_threads\_context, deinit\_threads\_context) и функциям для парсинга входных параметров и ввода строки от пользователя, при необходимости:

Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание

1. rand\_sleep.h – заголовочный файл с функцией rand\_sleep (объявлена отдельно, т.к. на Windows и Linux для “сна” используются разные функции, и в файле с помощбю директив препроцессор выбирается нужная реализация тела функции). Рандомное время сна, ограниченное сверху входным параметром, выбирается с помощью функции rand() (достаточно плохой, но быстрый генератор случайных чисел):

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

1. main.c – файл с реализацией приложения. В нём реализованы функции для создания потоков, сохранения результата кодирования строки и взятия нового номера чанка строки для кодировк:

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Приведённый выше код относится к version 1 (реализация с мьютексами). В version 2 код выглядит  
аналогично, однако вместо мьютексов и двусвязного списка атомарные переменные и массив.

В качестве входных данных для сравнения программ используются файлы test\_in\_1.txt, test\_in\_2.txt и test\_in\_3.txt. Результаты работы потоков шифрования пишутся в консоль и указанные файлы test\_out\_1.txt, test\_out\_2.txt и test\_out\_3.txt. В каждом входном файле находится строка для шифрования, которая переводится в массив чисел, который записывается в выходной файл и выводится в консоль. Также на скриншотах показано, как ведёт себя программа, если данные ввести из консоли или через флаг -in

Скриншоты работы программы:

version 1 (с мьютексами):

Первый тестовый файл:

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Второй тестовый файл:

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Третий тестовый файл:

Изображение выглядит как снимок экрана, текст

Автоматически созданное описание

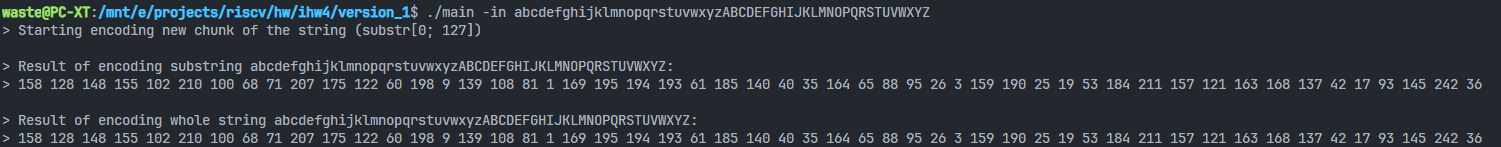
Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

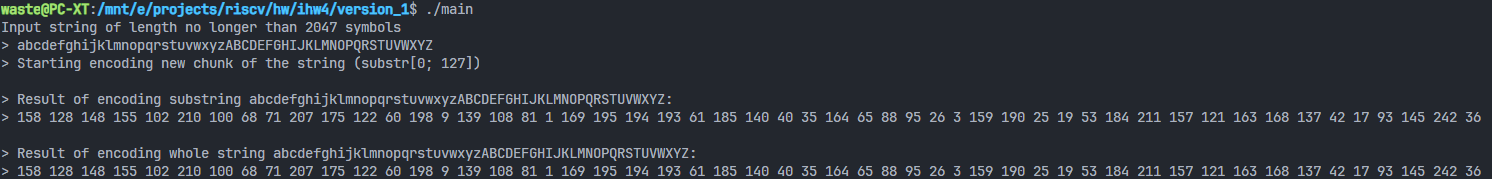
Изображение выглядит как одежда, шаблон, ткань

Автоматически созданное описание

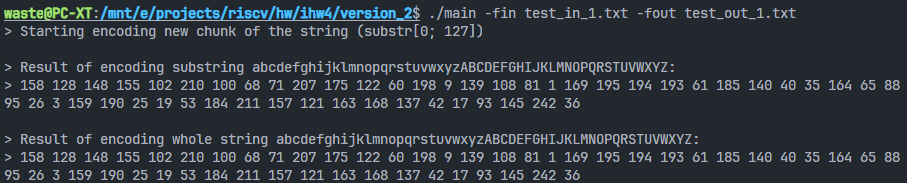
Ввод с флагом -in :



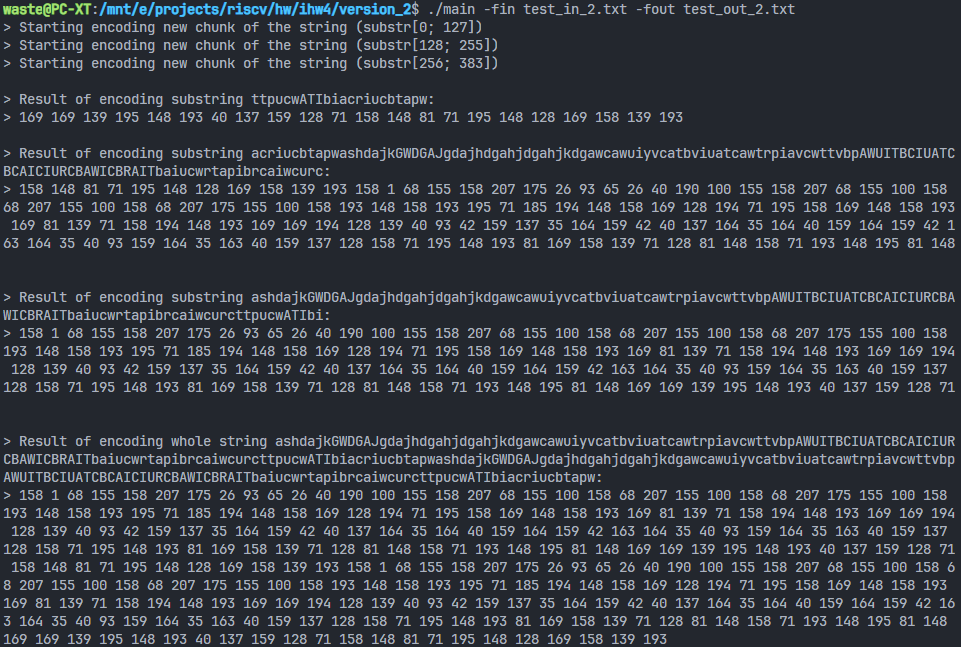
Ввод из консоли:



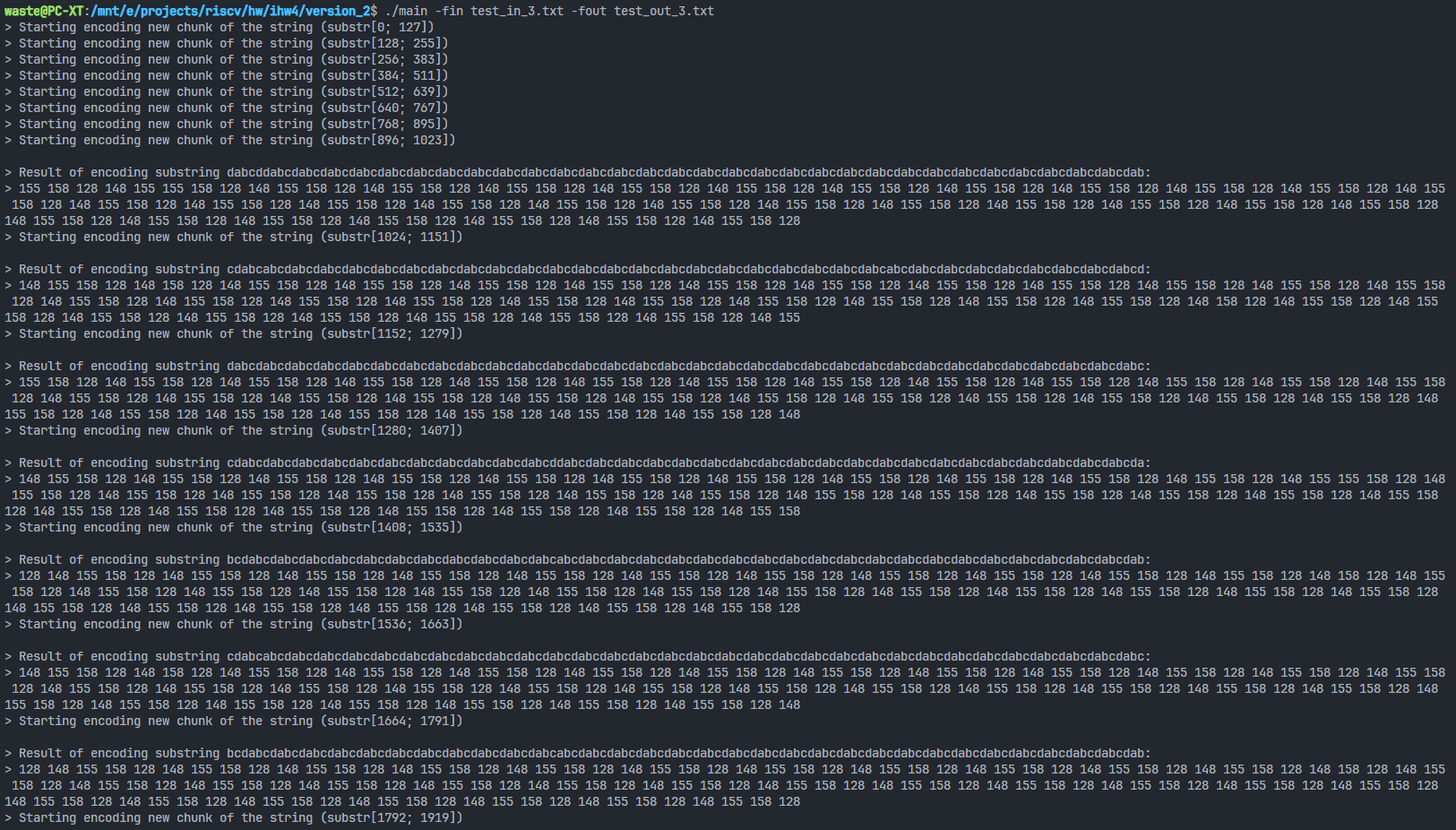
version 2 (с atomic переменными (формально, с \_Atomic переменными из stdatomic.h)):

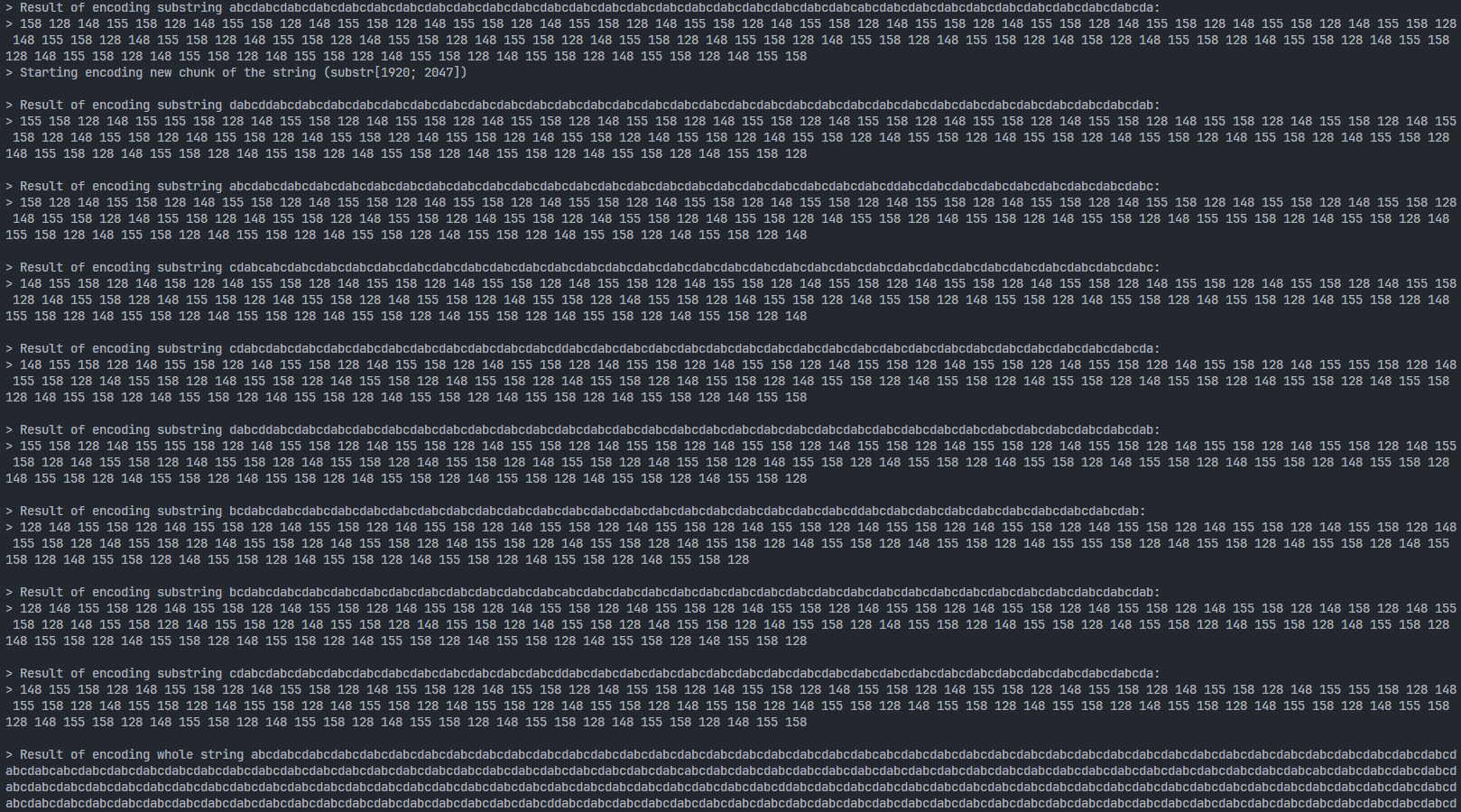
Первый тестовый файл:

Второй тестовый файл:



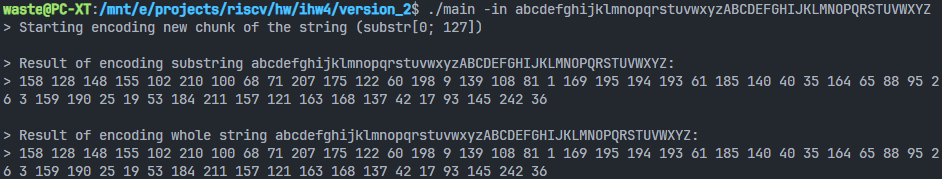
Третий тестовый файл:



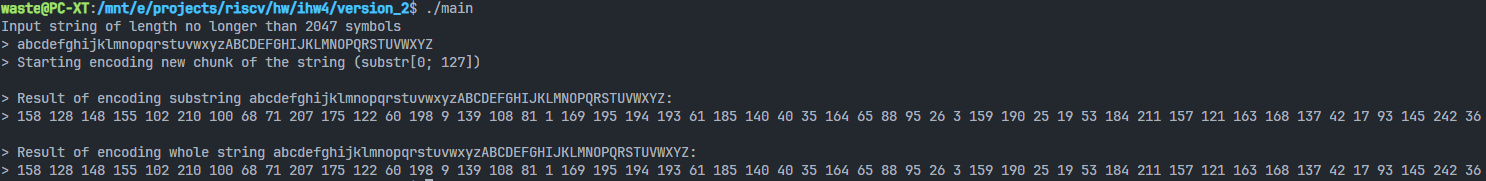




Ввод с флагом -in :



Ввод из консоли:



Программы, реализованные с помощью библиотеки pthread, компилировались с параметрами:

1. На Windows 10:
   1. Компилятор:  
      gcc version 13.2.0, built by msys2 project, environment mingw w64
   2. Версия языка: C17
   3. Флаги компиляции:  
      -std=c17 -g3 .\main.c -O2 -Wall -Wextra -Wpedantic -Wunused -Wconversion -Wshadow -Wnull-dereference -Wsign-conversion -Warith-conversion -Wint-conversion -Wcast-align=strict -Warray-bounds=2 -lpthread -o main.exe
2. На Windows 10 в WSL2 с подсистемой Ubuntu 20.04.2:
   1. Компилятор:  
      gcc version 9.4.0
   2. Версия языка: C17
   3. Флаги компиляции:  
      -std=c17 -g3 ./main.c -O2 -Wall -Wextra -Wpedantic -Wunused -Wconversion -Wshadow -Wnull-dereference -Wsign-conversion -Wint-conversion -Wcast-align=strict -Warray-bounds=2 -lpthread -o main

version 3 (с использованием OMP)

При реализации данной версии приложения за основу была взята version 2, с хранением зашифрованных чанков текст в массиве.

Функции парсинга входных аргументов и ввода строки не изменились, однако теперь потоки для шифрования чанков текста создаются не с помощью библиотеки pthread, а с помощью библиотеки openmp и прагмы omp parallel. Для доступа к счётчикам чанков и элементов массива используется прагма omp critical:

Изображение выглядит как текст, снимок экрана, программное обеспечение

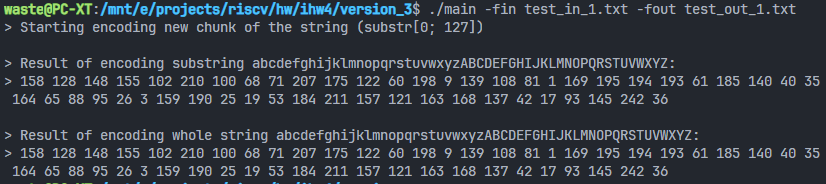
Автоматически созданное описание

Изображение выглядит как текст, снимок экрана

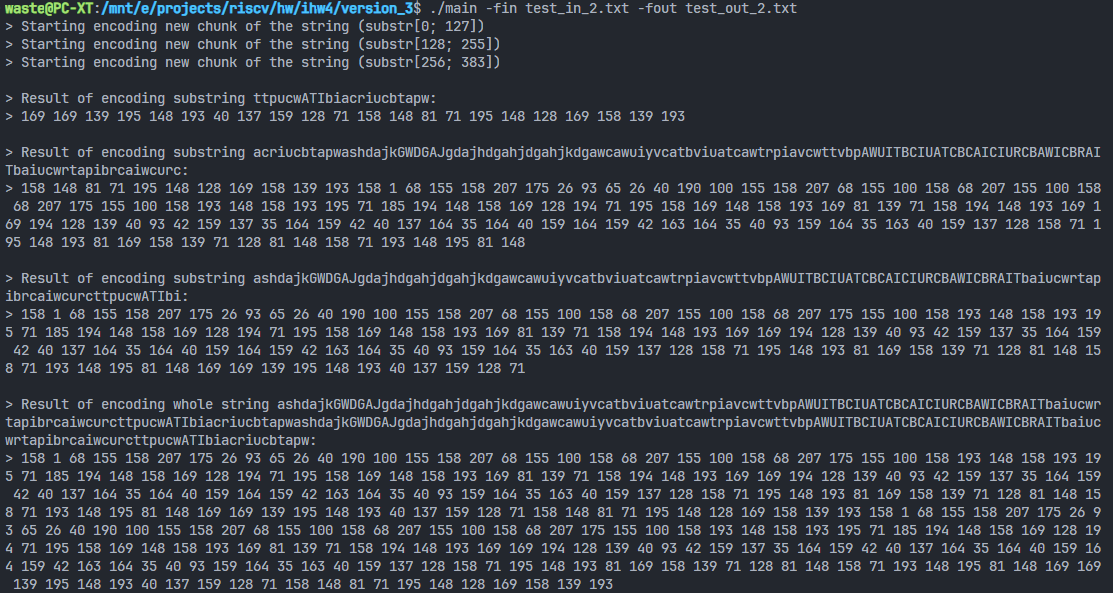
Автоматически созданное описание

Скриншоты работы программы:

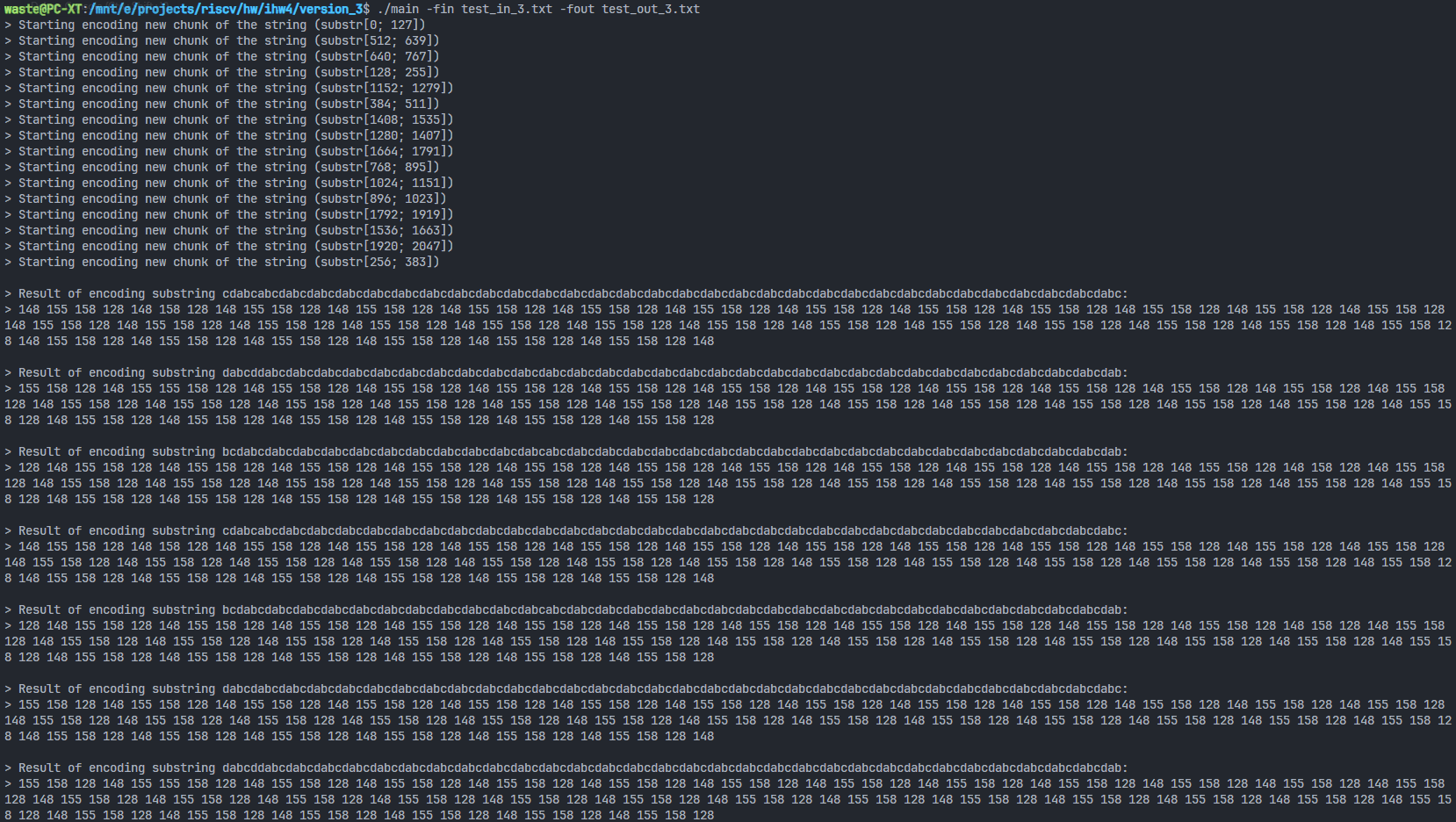
Первый тестовый файл:

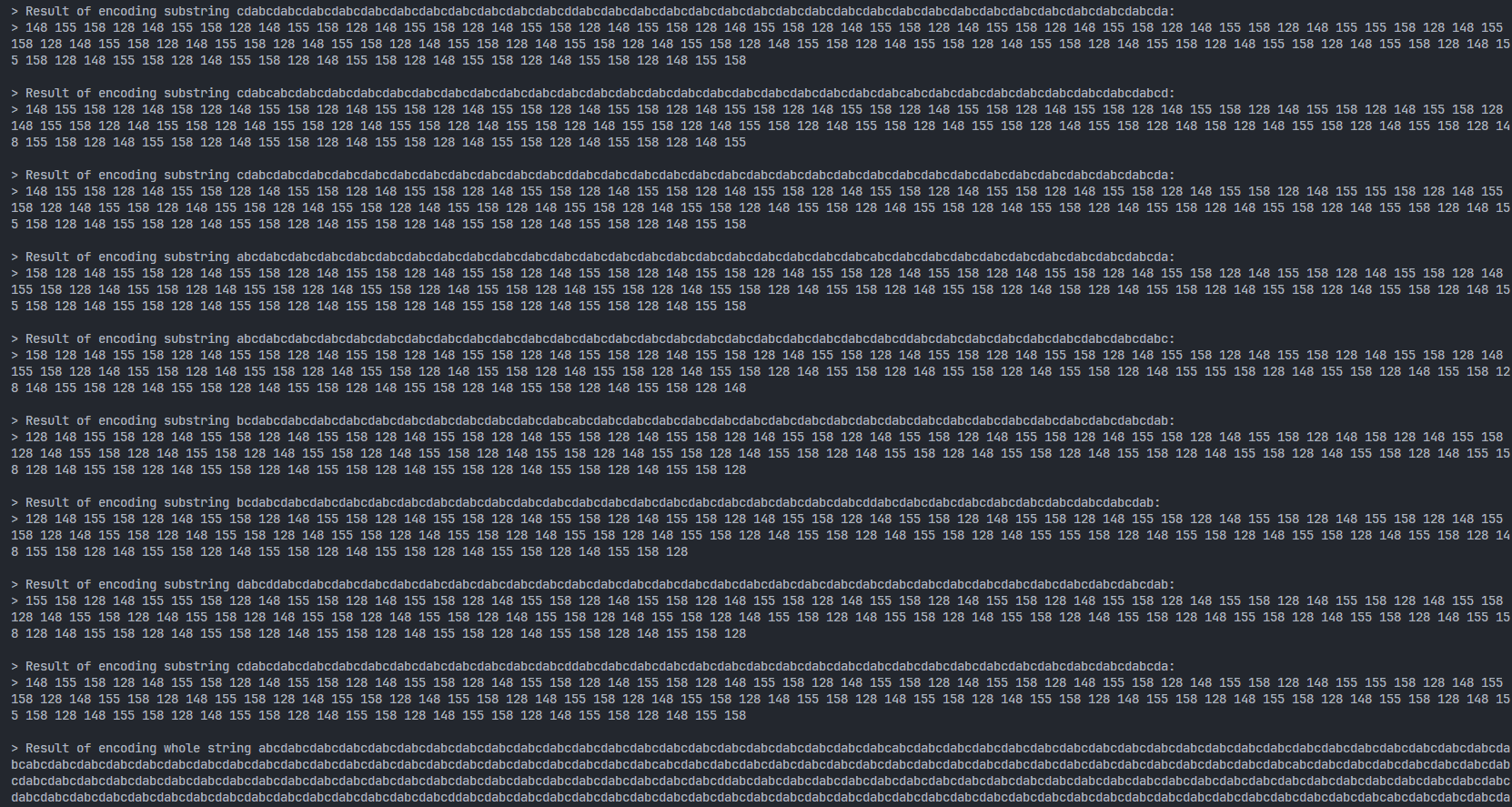


Второй тестовый файл:



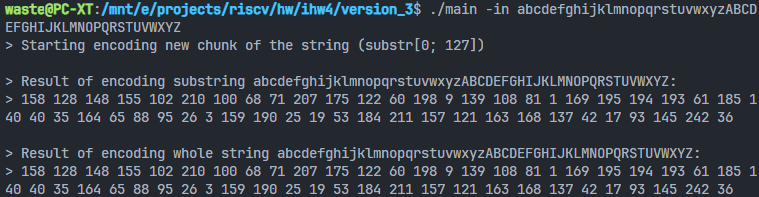
Третий тестовый файл:



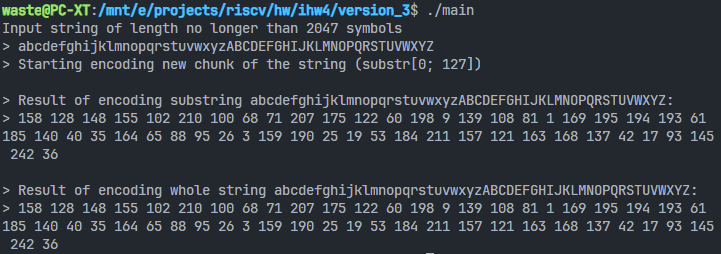




Ввод с флагом -in :



Ввод из консоли:



Как видно, результаты работы программы совпадают с результатами программ version 1 и version 2, реализованных с помощью библиотеки pthread.

Программа, реализованная с помощью библиотеки openMP, компилировалась с параметрами:

1. На Windows 10:
   1. Компилятор:  
      gcc version 13.2.0, built by msys2 project, environment mingw w64
   2. Версия языка: C17
   3. Флаги компиляции:  
      -std=c17 -g3 .\main.c -O2 -Wall -Wextra -Wpedantic -Wunused -Wconversion -Wshadow -Wnull-dereference -Wsign-conversion -Warith-conversion -Wint-conversion -Wcast-align=strict -Warray-bounds=2 -fopenmp -o main.exe
2. На Windows 10 в WSL2 с подсистемой Ubuntu 20.04.2:
   1. Компилятор:  
      gcc version 9.4.0
   2. Версия языка: C17
   3. Флаги компиляции:  
      -std=c17 -g3 ./main.c -O2 -Wall -Wextra -Wpedantic -Wunused -Wconversion -Wshadow -Wnull-dereference -Wsign-conversion -Wint-conversion -Wcast-align=strict -Warray-bounds=2 -fopenmp -o main