МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФГБОУ ВО «АЛТАЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

ИНСТИТУТ ЦИФРОВЫХ ТЕХНОЛОГИЙ, ЭЛЕКТРОНИКИ И ФИЗИКИ (ИЦТЭФ)

КАФЕДРА ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ И ЭЛЕКТРОНИКИ (ВТиЭ)

**Отчет по лабораторной работе № 3**

по курсу “Системное программное обеспечение”

**Вариант №10**

Выполнил студент 506 гр.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д.В. Осипенко

Проверил: преп. кафедры ВТиЭ.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.В. Сеулеков

Лабораторная работа защищена

“\_\_\_” \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2023 г.

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Барнаул 2023

# Цель работы:

Получение навыков работы с Pipe и Threads в Linux.

# **Задачи:**

1. Напишите программу, в которой два процесса общаются через pipe, выполняя совместно обработку данных из лабораторной работы №1. Родительский процесс сначала читает информацию из входных файлов, производит необходимые вычисления, потом создает процесс потомок, передает ему через pipe результаты и ждёт окончания процесса-потомка. А процесс-потомок записывает результат в выходной файл.

2. Напишите программу, которая выполняет обработку данных согласно варианту из лабораторной работы №2, но делает это, используя две нити – одна читает информацию из входных файлов и обрабатывает ее, а другая записывает результат в файл.

# Задание:

1. Вариант 10(4) – Подсчитать количество пробельных символов (пробелов и табуляций) во всех входных файлах. Результат вывести в файл.

2. Вариант 10 – Найти среди входных файлов файл, содержащий минимальное количество пробельных символов (пробелов и табуляций), записать в выходной файл это количество и название этого файла. Если таких файлов несколько, записать все их имена.

# **Выполнение работы:**

В директорию input помещаем 3 файла:

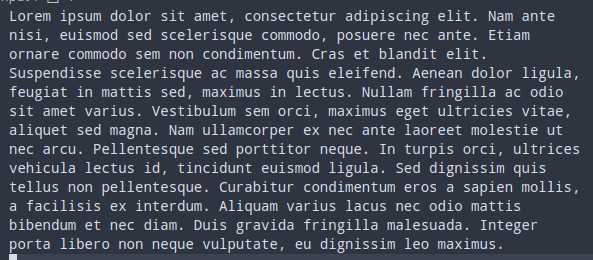


Рис 1.1 Содержимое файла 1.

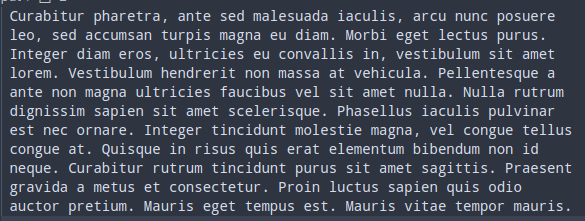


Рис 1.2 Содержимое файла 2.

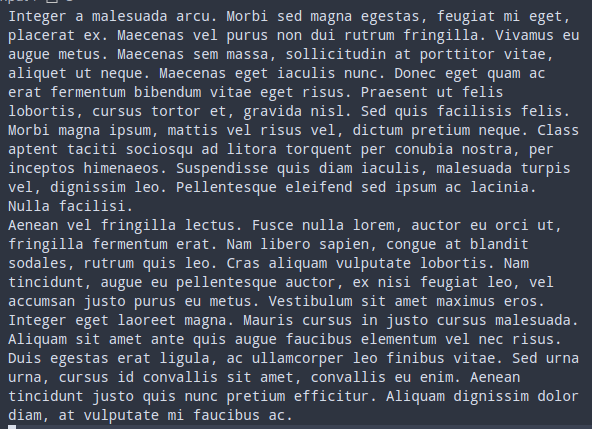


Рис 1.3 Содержимое файла 3.

Далее происходит разработка программы на языке ZIG. Для доступа к директориям, файлам и создания дочерних процессов используются функции из стандартной библиотеки, которые являются обертками системный вызовов определенной операционной системы.

1. Далее приведен код родительской программы, выполняющей подсчет пробелов и табуляций во входных файлах, затем передающая результат в дочерний процесс:

const std = @import("std"); // импортируем стандартную библиотеку

// задаем псевдонимы для библиотек файловой системы и линукса

const fs = std.fs;

const linux = std.os.linux;

const cout = std.io.getStdOut().writer(); // псевдоним для потока вывода

pub fn main() !void {

var dir = try fs.cwd().openIterableDir("input", .{}); // открываем входную директорию

defer dir.close();

var it = dir.iterate();

// инициализируем дочерний процесс

var cp = std.process.Child.init(&.{"./lab3.2"}, std.heap.page\_allocator);

cp.stdin\_behavior = .Pipe; // устанавливаем входной поток дочернего процесса в режим pipe

while (try it.next()) |entry| { // проходим через входные файлы

var file = try dir.dir.openFile(entry.name, .{ .mode = .read\_only });

defer file.close();

var data = try file.readToEndAlloc(std.heap.page\_allocator, 65535); // читаем файл

var spaces = std.mem.count(u8, data, " "); // считаем кол-во пробелов

var tabs = std.mem.count(u8, data, "\t"); // считаем кол-во табуляций

var blancs = spaces + tabs; // складываем

try cp.spawn(); // загружаем дочерний процесс

try std.fmt.format(cp.stdin.?.writer(), "input/{s} -> {d}\n", .{ entry.name, blancs }); // передаем ему через поток результат вычислений

cp.stdin.?.close(); // закрываем поток

cp.stdin = null; // зануляем его, во избежания простоя дочернего процесса

\_ = try cp.wait(); // ждем окончания работы дочернего процесса

}

}

Затем приведен код дочернего процесса, считывающая из stdin данные, поступающие из родительского процесса, и выполняющая запись в выходной файл:

const std = @import("std");

const fs = std.fs;

const cout = std.io.getStdOut().writer();

const cin = std.io.getStdIn().reader();

pub fn main() !void {

// открываем выходной файл

var output = try fs.cwd().openFile("output/output", .{ .mode = .read\_write });

defer output.close();

// получаем его метаданные

var stat = try output.stat();

// устанавливаем указатель начала записи на конец файла

try output.seekTo(stat.size);

// считываем данные из stdin

var data = try cin.readAllAlloc(std.heap.page\_allocator, 65536);

// записываем их в выходной файл

try std.fmt.format(output.writer(), "{s}", .{data});

}

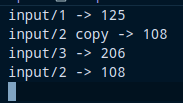


Рис. 1.5 Результат работы

2. Далее приведен код программы, который в первой нити осуществляет чтение и обработку входного файла, а во второй производит запись результата в выходной файл:

const std = @import("std");

const fs = std.fs;

const cout = std.io.getStdOut().writer();

const string = []const u8;

var min: i32 = -1; // переменная для минимального значения

// список имен, содержащих это число

var nameList = std.ArrayList(string).init(std.heap.page\_allocator);

pub fn main() !void {

// открываем входной файл

var out1 = try fs.cwd().openFile("output/output", .{ .mode = .read\_only });

defer out1.close();

// считываем его

var data = try out1.readToEndAlloc(std.heap.page\_allocator, 65536);

var it = std.mem.split(u8, data, "\n"); // разбиваем по строкам

while (it.next()) |row| {

if (row.len == 0) break;

var towIt = std.mem.split(u8, row, " -> "); // разбиваем строку на имя файла и число

var name = towIt.next().?; // записываем имя

var value = try std.fmt.parseInt(i32, towIt.next().?, 10); // записываем число

if (min == -1 or min > value) { // если число минимально

min = value; // задаем новой минимальное

nameList.clearAndFree(); // очищаем список имен

try nameList.append(name); // заносим имя

} else if (min == value) { // если мин и число совпали

try nameList.append(name); // просто заносим имя

}

}

var thread = try std.Thread.spawn(.{}, sub, .{}); // создаем вторую нить

thread.join(); // запускаем и ждем ее окончания

}

fn sub() !void {

// открываем выходной файл

var out2 = try fs.cwd().openFile("output/output2", .{ .mode = .write\_only });

defer out2.close();

// записываем имена

try std.fmt.format(cout, "{d} -> {s}\n", .{ min, nameList.items });

// Записываем минимальное число

try std.fmt.format(out2.writer(), "{d} -> {s}\n", .{ min, nameList.items });

}



Рис. 1.6 Результат работы

# Вывод:

В ходе выполнения лабораторной работы были получили навыки работы с Pipe и Threads.