Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №2 по курсу**

**«Операционные системы»**

Студент: Орусский Вячеслав Русланович

Группа: М8О-206Б-21

Вариант: 20

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2023

Содержание

[Содержание 2](#_Toc133018203)

[**Репозиторий** 3](#_Toc133018204)

[**Постановка задачи** 3](#_Toc133018205)

[**Общие сведения о программе** 3](#_Toc133018206)

[**Общий метод и алгоритм решения** 4](#_Toc133018207)

[**Исходный код** 4](#_Toc133018208)

[Main.cpp: 4](#_Toc133018209)

[Slave1.cpp: 5](#_Toc133018210)

[Slave2.cpp: 6](#_Toc133018211)

[**Демонстрация работы программы** 6](#_Toc133018212)

[**Выводы** 7](#_Toc133018213)

**Репозиторий**

[Ссылка](https://github.com/RoKivals/OperatingSystems/tree/master/Lab%232)

**Постановка задачи**

**Цель работы**

Научиться создавать процессы, уметь работать с в дочернем и родительском процессах, а также обеспечить их взаимодействие (общение, передачу данных) с помощью канала данных «pipe».

**Задание**

Родительский процесс создает два дочерних процесса (slave1 и slave2). Для чтения дочерним процессам соответственно подаются на ввод текстовые файлы (file1 и file2). Родительский и дочерний процессы представлены разными программами.

Процессы slave1 и slave2 производят работу над строками (переворачивая каждую из них). Процессы пишут результаты своей работы в файлы (pipe1 и pipe2 соответственно)

Правило фильтрации: строки длины больше 10 символов отправляются в pipe2, иначе в pipe1.

**Общие сведения о программе**

Программа компилируется из файла main.cpp. Также подключаются файлы child1.cpp, child2.cpp через execlp в качестве отдельной программы. В родительский процесс подаем файлы file1.txt, file2.txt. В программе используются следующие системные вызовы:

1. pipe() – создает связь между памятью процессов
2. fork() – создает второй процесс
3. dup2() – копирует old\_file\_descriptor в new\_file\_descriptor.

**Общий метод и алгоритм решения**

В начале создаются два канала pipe, затем два дочерних процесса. Далее, в зависимости от того, какой длины строка нам попалась, мы её направляем на обработку одному из двух процессов. Родительский процесс отвечает за передачу данных дочерним через pipe для дальнейшей обработки.

В файлы pipe1.txt и pipe2.txt заносятся выходные данные.

**Исходный код**

Main.cpp:

**#include** <bits/stdc++.h>  
**#include** <unistd.h>  
**#include** <fcntl.h>  
**#include** <signal.h>  
**#include** <sys/wait.h>  
  
*//#define READ 0  
//#define WRITE 1*using namespace std;  
  
void wrt\_fd(int \*fd1, int \*fd2, *string* name\_file) {  
 *ifstream* file1(name\_file);  
 *string* line;  
 if (file1.is\_open()) {  
 while (getline(file1, line)) {  
 line *=* line *+* "\n";  
 int lineSize = line.length();  
 if (lineSize > 10) {  
 if (write(fd2[1], line.c\_str(), lineSize \* sizeof(char)) == -1) {  
 std::perror("Pipe2");  
 return;  
 }  
 } else {  
 if (write(fd1[1], line.c\_str(), lineSize \* sizeof(char)) == -1) {  
 std::perror("Pipe1");  
 return;  
 }  
 }  
 }  
 }  
 file1.close();  
}  
  
int main(int argc, char \*argv[]) {  
 *// Массивы указывающие на концы каналов* int fd1[2];  
 int fd2[2];  
  
 *// Создаём 2 пайпа* if (pipe(fd1) == -1) {  
 std::perror("pipe1");  
 return 1;  
 }  
  
 if (pipe(fd2) == -1) {  
 std::perror("pipe2");  
 return 2;  
 }  
  
 *// Создаём 1-ый дочерний процесс* int id = fork(), id2;  
 int flag = 0;  
  
 *// Если мы в родительском процессе, то создаём второго ребёнка* if (id > 0) {  
 id2 = fork();  
 flag = 1;  
 }  
 if (id == -1 || id2 == -1) {  
 std::perror("fork");  
 return 5;  
 } else if (id2 != 0 && id != 0) {  
 *// Закрываем каналы для чтения* close(fd1[0]);  
 close(fd2[0]);  
 wrt\_fd(fd1, fd2, "./file1.txt");  
 wrt\_fd(fd1, fd2, "./file2.txt");  
 close(fd1[1]);  
 close(fd2[1]);  
 } else if (flag) {  
 close(fd1[1]);  
 close(fd1[1]);  
 close(fd2[0]);  
 dup2(fd1[0], **STDIN\_FILENO**);  
 execlp("./slave1", "slave1", **NULL**);  
 close(fd1[0]);  
 } else {  
 close(fd2[1]);  
 close(fd1[0]);  
 close(fd1[1]);  
 dup2(fd2[0], **STDIN\_FILENO**);  
 execlp("./slave2", "slave2", **NULL**);  
 close(fd2[0]);  
 }  
 return 0;  
}

Slave1.cpp:

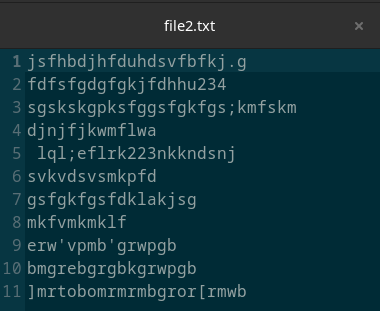
**#include**<bits/stdc++.h>  
**#include**<sys/wait.h>  
**#include**<unistd.h>  
using namespace std;  
  
void ReverseString(*string* &str) {  
 int n = str.length();  
 for (int i(0); i < n / 2; ++i) {  
 swap(str*[*i*]*, str*[*n - i - 1*]*);  
 }  
}  
  
int main(int argc, char \*argv[]) {  
 *string* s;  
  
 *// out - открытие на запись, trunc - очистка содержимого при открытии  
 ofstream* fout("./pipe1.txt", ios\_base::out *|* ios\_base::trunc);  
 while (getline(*cin*, s)) {  
 ReverseString(s);  
 fout *<<* s *+* "\r\n";  
 }  
 fout.close();  
 return 0;  
}

Slave2.cpp:

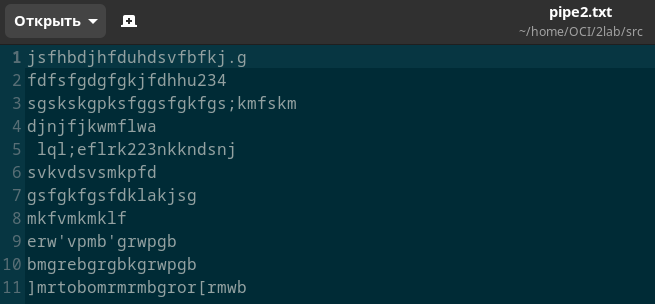
**#include**<bits/stdc++.h>  
**#include**<unistd.h>  
using namespace std;  
  
void ReverseString(string &str) {  
 int n = str.length();  
 for (int i(0); i < n / 2; ++i) {  
 swap(str[i], str[n - i - 1]);  
 }  
}  
  
int main(int argc, char \*argv[]) {  
 string s;  
 ofstream fout("./pipe2.txt", ios\_base::out | ios\_base::trunc);  
 while (getline(cin, s)) {  
 ReverseString(s);  
 fout << s + "\r\n";  
 }  
 fout.close();  
 return 0;  
}

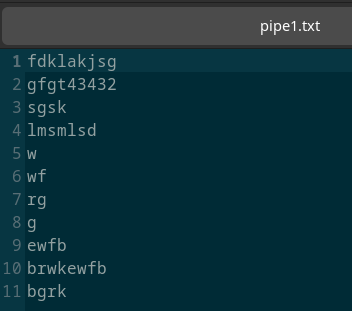
**Демонстрация работы программы**

Ввод**:**



Вывод**:**





**Выводы**

С помощью системных вызовов в ОС Linux с помощью ЯП C++ (Си) можно создавать программы, в которых работает несколько процессов, что крайне положительно может влиять на производительность, а также логику поведения программы. Для общения между процессами используется pipe, а не обычный файл, так как с обычным файлом могут возникать различные ошибки доступа и записи между процессами.