Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №2 по курсу**

**«Операционные системы»**

Группа: М80-207Б-20

Студент: Пономарев Н.В.

Преподаватель: Миронов Е.С.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: 03.10.21

Москва, 2021.

# Содержание

1. Постановка задачи.
2. Общие сведения о программе.
3. Общий метод и алгоритм решения.
4. Код программы.
5. Демонстрация работы программы.
6. Вывод.

## Постановка задачи

Составить и отладить программу на языке С++, родительский процесс которой считывает стандартный входной поток, отдает его дочернему процессу, который выполняет проверку строки на правило: «Строка начинается с большой буквы» и, если проверка пройдена успешно, то записывает строку в файл(имя файла также передается от родительского процесса), а в противном случае строки возвращаются в родительский процесс и выводятся в терминал.

## Общие сведения о программе

Программа состоит из одного файла main.cpp. В данном файле используются заголовочные файлы iostream, fstream, fcntl.h, unistd.h.

Программа использует следующие системные вызовы:

1. **read –** для чтения данных из входного потока.
2. **write** – для записи данных в файл или выходной поток.
3. **pipe –** для создания канала, через который процессы смогут обмениваться информацией.
4. **fork –** для создания дочернего процесса.
5. **close –** для закрытия выходного файла.
6. **dup2** – для перенаправления потока вывода.
7. **getpid** - для получения id процесса.

## Общий метод и алгоритм решения

* Проверить, задан ли файл для вывода как аргумент, создать дочерний процесс(с помощью **fork**) и **pipe**(для передачи данных из родительского процесса в дочерний) обработать возможные ошибки.
* Из родительского процесса: записать в **pipe** очередную строку для проверки: cчитать все символы, переданные во входной поток, и передать их через **pipe** в дочерний следующим образом: сначала вводится количество символов, а затем и сами эти символы. Будем продолжать такие действия до того момента, как не будет встречен символ конца ввода EOF.
* Из дочернего процесса: считывать из **pipe** число, показывающее сколько символов в строке, а затем и саму строку **s.** Далее выполняем проверку на наше правило. Если строка не прошла проверку, то мы по **pipe** передаем строку в родительский процесс, если же строка прошла проверку на правило, то полученная строка записывается в выходной файл с помощью функции **write**. Когда символы, переданные из родительского процесса закончатся, дочерний процесс закроет файл с помощью функции **close** и завершится с кодом выхода 0.
* После передачи символов входного потока в дочерний процесс, родительский процесс ожидает его завершения. После выполняется считывание строк, не прошедших правило, из **pipe** и выполняется их вывод в консоль..

## Код программы

**main.cpp:**

**#include <iostream>**

**#include <fstream>**

**#include <fcntl.h>**

**#include "unistd.h"**

**int main(){**

**int fd1[2];**

**int fd2[2];**

**std::string name\_of\_file;**

**std::string s;**

**const char \*name\_of\_file\_c = name\_of\_file.c\_str();**

**std::cout << "Print input name of file: ";**

**std::cin >> name\_of\_file;**

**int f = open(name\_of\_file\_c, O\_WRONLY | O\_CREAT, 0777);**

**if(f == -1){**

**std::cout << "Error during creating file\n";**

**return 5;**

**}**

**if(pipe(fd1) == -1){**

**std::cout << "Error during creating pipe1\n";**

**return 1;**

**}**

**if(pipe(fd2) == -1){**

**std::cout << "Error during creating pipe2\n";**

**return 4;**

**}**

**int id = fork();**

**int size;**

**switch(id){**

**case -1:**

**std::cout << "Error during creating fork\n";**

**return 2;**

**break;**

**case 0:**

**printf("[%d] It's child\n", getpid());**

**dup2(f, 1);**

**close(fd1[1]);**

**close(fd2[0]);**

**while(read(fd1[0], &size, sizeof(int))){**

**s.resize(size);**

**read(fd1[0], s.data(), size \* sizeof(char));**

**if(65 <= int(s[0]) and int(s[0]) <= 90){**

**std::cout << s << "\n";**

**//f << s << "\n";**

**} else {**

**s = "Error in string " + s;**

**int size = s.length();**

**write(fd2[1], &size, sizeof(int));**

**write(fd2[1], s.data(), sizeof(char) \* size);**

**}**

**}**

**close(fd1[0]);**

**close(fd2[1]);**

**break;**

**default:**

**printf("[%d] It's parent. Child id: %d\n", getpid(), id);**

**close(fd1[0]);**

**close(fd2[1]);**

**while(std::cin >> s && s != "EOF"){**

**size = s.length();**

**write(fd1[1], &size, sizeof(int));**

**write(fd1[1], s.data(), (size \* sizeof(char)));**

**}**

**close(fd1[1]);**

**while(read(fd2[0], &size, sizeof(int))){**

**s.resize(size);**

**read(fd2[0], s.data(), (size \* sizeof(char)));**

**std::cout << s << "\n";**

**}**

**close(fd2[0]);**

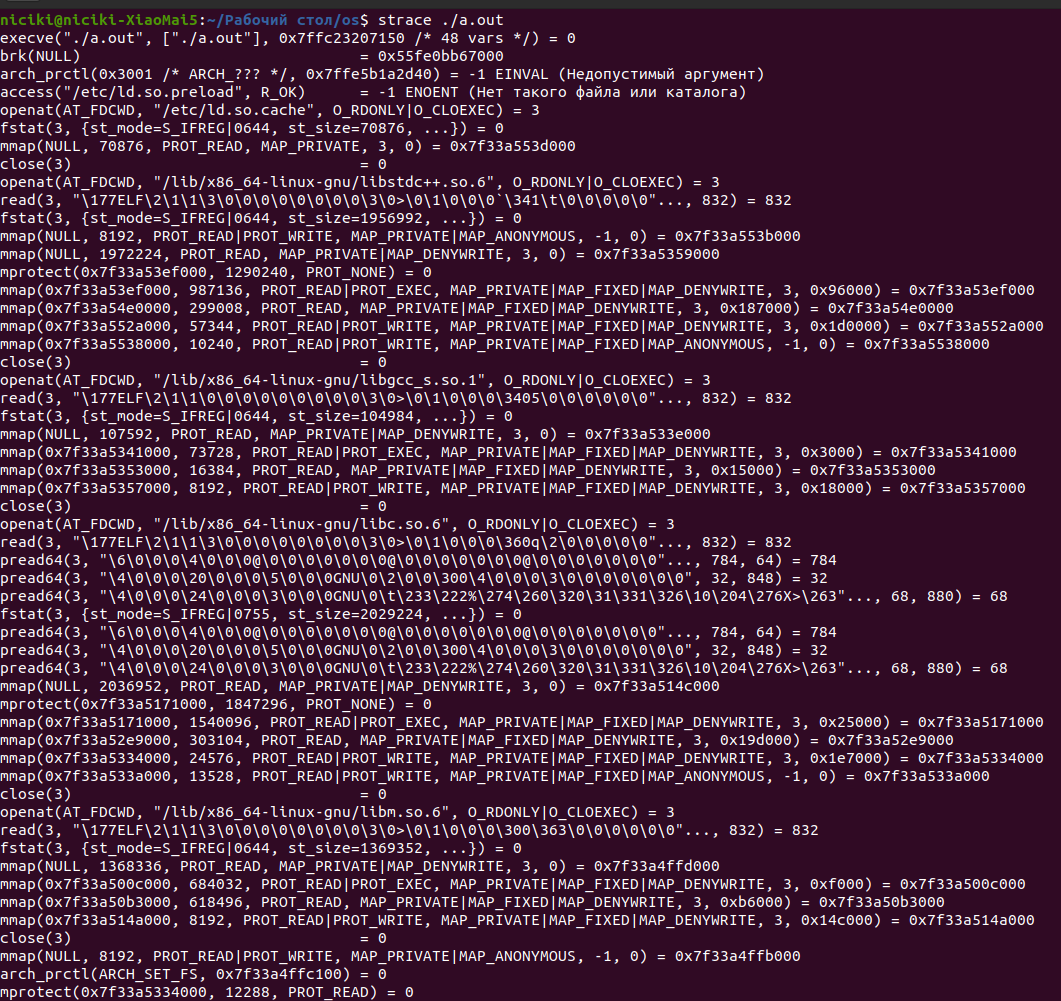
**return 0;**

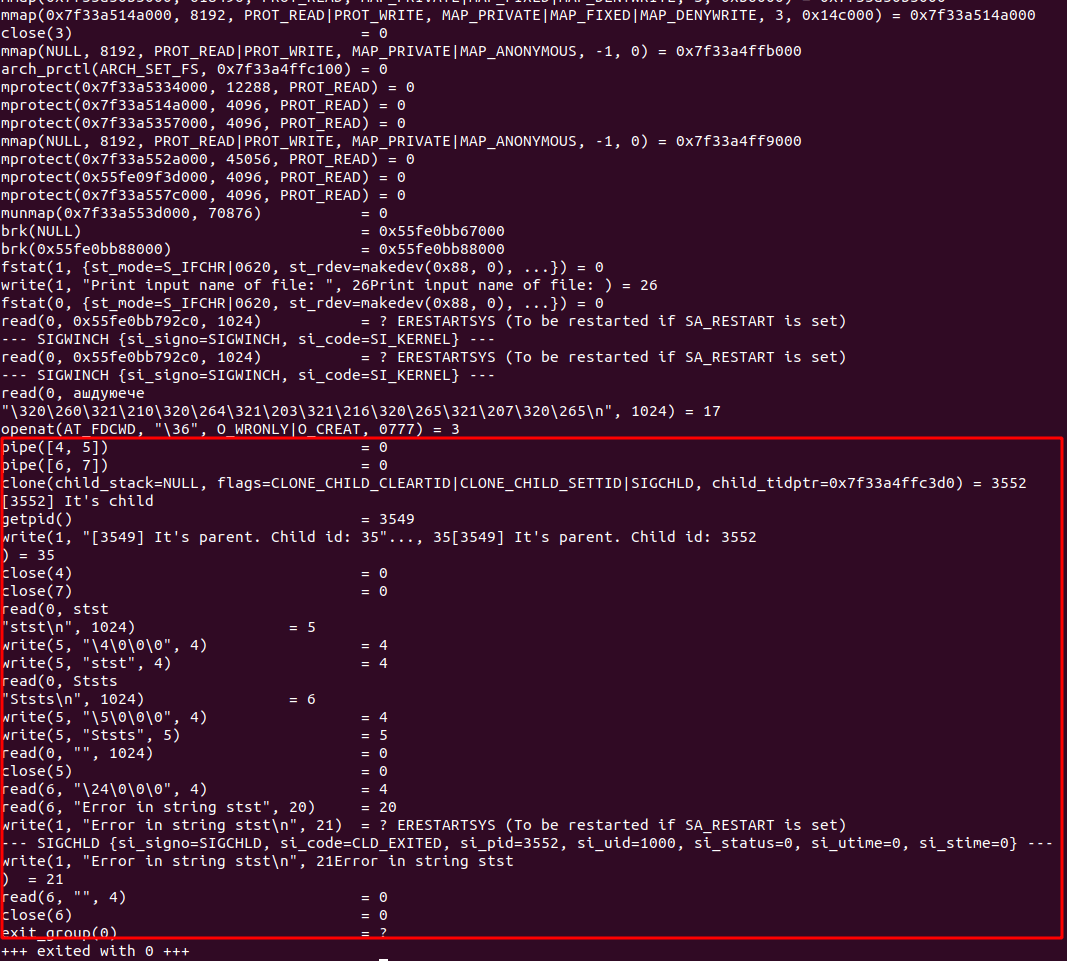
**close(f);**

**}**

**}**

## Использование утилиты strace





Листинг выполнения программы с использованием утилиты strace приведен в виде изображений. Создание неименнованных каналов (pipe), а также работа с ними отмечены красным прямоугольником на последнем слайде.

## Демонстрация работы программы

Print input name of file: test.txt

[3987] It's parent. Child id: 4012

[4012] It's child

String

string

mAmaa

nopoaASASasds

asSFAFAASFASfsafafa

Child

GOGOG

aasda

Aasda

**Output in terminal:**

Error in string string

Error in string mAmaa

Error in string nopoaASASasds

Error in string asSFAFAASFASfsafafa

Error in string aasda

**Output in file test.txt:**

String

Child

GOGOG

Aasda

## Вывод

Данная лабораторная работа оказалась полезной и интересной: она познакомила меня с понятием процесса в операционной системе, помогла мне разобраться с тем, как следует работать с неименованными каналами для межпроцессорного взаимодействия, научила переопределять потоки ввода/вывода. Хотя написанная программа реализуется проще в однопроцессорном режиме, а многопроцессорность, с точки зрения практичности, является бесполезной и ненужной, но ее написание позволило глубже разобраться в устройстве работы операционной системы, а использование утилиты strace показало обратную, более низкоуровневую сторону работы моей программы. Навыки работы с процессами, а также основы межпроцессорного взаимодействия безусловно пригодятся мне в будущем, ведь сложно придумать нетривиальную программу, которая работает в однопроцессорном режиме, где используются один и более неименнованных каналов.