Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №1 по курсу**

**«Операционные системы»**

Студент: Ибрагимов Далгат Магомедалиевич

Группа: М8О-208Б-22

Вариант: 14

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2023

**Содержание**

1. Репозиторий

2. Постановка задачи

3. Общие сведения о программе

4. Общий метод и алгоритм решения

5. Исходный код

6. Демонстрация работы программы

7. Выводы

**Репозиторий**

<https://github.com/L0ckR/OS_LABS>

**Постановка задачи**

Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы программа (основной процесс) должен создать для решение задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или каналы (pipe). Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

**Общие сведения о программе**

Программа состоит из двух частей – главная программа, родительский процесс, описанный в main.c, и программа дочернего прочесса, описанная в child.c. В программе используются следующие системныe вызовы:

1. fork()
2. execv()
3. pipe()
4. read()
5. write()
6. open()
7. close()
8. dup2()

**Общий метод и алгоритм решения**

Родительский процесс получает на вход имена выходных файлов и создает дочерние процессы, передавая им имена файлов в качестве аргументов командной строки. Перед этим родительский процесс создает два канала, по которым он будет передавать дочерним процессам строки, согласно правилу филтрации. Дочерние процессы удаляют из строк гласные и записывают результат в соответствующие файлы.

**Исходный код**

**========================== parent.h ==========================**

**#pragma once**

**#include <utils.hpp>**

**void ParentProcess(const char \* pathToChild1, const char \* pathToChild2, std::istream & streamIn, std::ostream & streamOut);**

**========================== utils.h ==========================**

**#pragma once**

**#include <sstream>**

**#include <algorithm>**

**#include <sys/wait.h>**

**#include <iostream>**

**#include <string>**

**#include <cstdlib>**

**#include <unistd.h>**

**#include <ext/stdio\_filebuf.h>**

**enum PipeEnd {**

**READ\_END,**

**WRITE\_END**

**};**

**void CreatePipe(int pipeFd[2]);**

**pid\_t CreateChild();**

**void Exec(const char \* pathToChild);**

**std::stringstream ReadFromPipe(int file\_descriptor);**

**========================== child1.c ==========================**

**#include <string>**

**#include <iostream>**

**#include <algorithm>**

**int main(){**

**std::string line;**

**while(std::getline(std::cin, line)){**

**transform(line.begin(), line.end(), line.begin(), ::tolower);**

**std::cout << line << std::endl;**

**}**

**exit(EXIT\_SUCCESS);**

**}**

**========================== child2.c ==========================**

**#include <string>**

**#include <iostream>**

**#include <algorithm>**

**bool BothAreSpaces(char lhs, char rhs){**

**return std::isspace(lhs) && std::isspace(rhs);**

**};**

**int main(){**

**std::string line;**

**while(std::getline(std::cin, line)){**

**auto new\_end = std::unique(line.begin(), line.end(), BothAreSpaces);**

**line.erase(new\_end, line.end());**

**std::cout << line << std::endl;**

**}**

**exit(EXIT\_SUCCESS);**

**}**

**========================== parent.c ==========================**

**), pid2 = -1;**

**if (pid == pid\_t(0)){**

**pid2 = CreateChild();**

**}**

**if (pid2 == pid\_t(0)){**

**//start of child1 process**

**close(child2Out[WRITE\_END]);**

**close(child1In[READ\_END]);**

**close(child1In[WRITE\_END]);**

**close(pipeBetween[READ\_END]);**

**if (dup2(child2Out[READ\_END], STDIN\_FILENO) == -1){**

**perror("dup2 error");**

**exit(EXIT\_FAILURE);**

**}**

**if (dup2(pipeBetween[WRITE\_END], STDOUT\_FILENO) == -1){**

**perror("dup2 error");**

**exit(EXIT\_FAILURE);**

**}**

**Exec(pathToChild1);**

**//end of child1 process**

**}**

**else if (pid2 > pid\_t(0)){**

**//start of child2 process**

**close(child2Out[WRITE\_END]);**

**close(child2Out[READ\_END]);**

**close(pipeBetween[WRITE\_END]);**

**close(child1In[READ\_END]);**

**if (dup2(pipeBetween[READ\_END], STDIN\_FILENO) == -1){**

**perror("dup2 error");**

**exit(EXIT\_FAILURE);**

**}**

**if (dup2(child1In[WRITE\_END], STDOUT\_FILENO) == -1){**

**perror("dup2 error");**

**exit(EXIT\_FAILURE);#include <parent.hpp>**

**void ParentProcess(const char \* pathToChild1, const char \* pathToChild2, std::istream & streamIn, std::ostream & streamOut ){**

**int child1In[2];**

**CreatePipe(child1In);**

**int child2Out[2];**

**CreatePipe(child2Out);**

**int pipeBetween[2];**

**CreatePipe(pipeBetween);**

**pid\_t pid = CreateChild(**

**}**

**Exec(pathToChild2);**

**//end of child2 process**

**}**

**else{**

**close(child1In[WRITE\_END]);**

**close(child2Out[READ\_END]);**

**close(pipeBetween[WRITE\_END]);**

**close(pipeBetween[READ\_END]);**

**std::string line;**

**while(std::getline(streamIn, line)){**

**line += "\n";**

**write(child2Out[WRITE\_END], line.c\_str(), line.size());**

**}**

**close(child2Out[WRITE\_END]);**

**std::stringstream output = ReadFromPipe(child1In[READ\_END]);**

**while(std::getline(output, line)){**

**streamOut << line << std::endl;**

**}**

**close(child1In[READ\_END]);**

**//exit(EXIT\_SUCCESS); успешный выход из процесса будет осуществлятьcя в main.cpp**

**}**

**}**

**========================== utils.c ==========================**

**#include <utils.hpp>**

**pid\_t CreateChild(){**

**if(pid\_t pid = fork(); pid >= 0) {**

**return pid;**

**}**

**std::perror("Couldn't create child.");**

**exit(EXIT\_FAILURE);**

**}**

**void CreatePipe(int pipeFd[2]){**

**if( pipe(pipeFd) < 0 ) {**

**std::perror("Couldn't create pipe.");**

**exit(EXIT\_FAILURE);**

**}**

**}**

**void Exec(const char \* pathToChild){**

**if (execl(pathToChild, pathToChild, nullptr) == -1) {**

**perror("Failed to exec.");**

**exit(EXIT\_FAILURE);**

**}**

**}**

**std::stringstream ReadFromPipe (int file\_descriptor){**

**std::stringstream stream;**

**while (1){**

**unsigned char buf[256] = "";**

**ssize\_t state = read(file\_descriptor, &buf, 256UL);**

**if (state > 0){**

**stream << buf;**

**}else if (state == 0){**

**return stream;**

**}else{**

**perror("Read fail");**

**exit(EXIT\_FAILURE);**

**}**

**}**

**// ЗАПРЕТНАЯ РЕАЛИЗАЦИЯ**

**// \_\_gnu\_cxx::stdio\_filebuf<char> filebuf(file\_descriptor, std::ios\_base::in);**

**// std::stringstream stream;**

**// stream << &filebuf;**

**// return stream;**

**}**

**========================== main.c ==========================**

**#include "parent.hpp"**

**int main(void) {**

**// export PATH\_TO\_CHILD1="/home/lockr/projects/OS\_LABS/build/lab1/child1"**

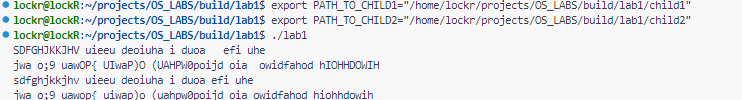
**// export PATH\_TO\_CHILD1="/home/lockr/projects/OS\_LABS/build/lab1/child1"**

**ParentProcess(getenv("PATH\_TO\_CHILD1"), getenv("PATH\_TO\_CHILD2"), std::cin, std::cout);**

**exit(EXIT\_SUCCESS);**

**}**

**Демонстрация работы программы**

****

**Выводы**

В ходе выполнения лабораторной работе я получил знания и навыки использования системных вызовов Linux при написании программ. Я узнал о системных вызовах fork, pipe, dup2 и научился их применять. Также были получены знания о структуре размещения процессов в памяти компьютера.