

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: Меркулов Ф. А.
Группа: М8О-207Б-21
Вариант: 9
Преподаватель: Черемисинов Максим Леонидович
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2022

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/WhatTheMUCK/OSi/tree/main/LR_1

Постановка задачи

Цель работы

Приобретение практических навыков в:

- Управление процессами в ОС
- Обеспечение обмена данных между процессами посредством каналов

Задание

Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия файла с таким именем на чтение. Стандартный поток ввода дочернего процесса переопределяется открытым файлом. Дочерний процесс читает команды из стандартного потока ввода. Стандартный поток вывода дочернего процесса перенаправляется в `pipe1`. Родительский процесс читает из `pipe1` и прочитанное выводит в свой стандартный поток вывода. Родительский и дочерний процесс должны быть представлены разными программами.

9 вариант) В файле записаны команды вида: «число число число<endline>». Дочерний процесс производит деление первого числа команда, на последующие числа в команде, а результат выводит в стандартный поток вывода. Если происходит деление на 0, то тогда дочерний и родительский процесс завершают свою работу. Проверка деления на 0 должна осуществляться на стороне дочернего процесса. Числа имеют тип `float`. Количество чисел может быть произвольным.

Общие сведения о программе

Программа компилируется из файла `main.c`. Также используются заголовочные файлы: `unistd.h`, `iostream`, `fcntl.h`, `sstream`.

В программе используются следующие системные вызовы:

1. `pipe()` - создает канал для чтения и записи.
2. `fork()` - создаёт новый процесс посредством копирования вызывающего процесса. Новый процесс считается дочерним процессом. Вызывающий процесс считается родительским процессом.
3. `execp()` - дублирует действия оболочки, относящиеся к поиску исполняемого файла. (семейство функций `exec` заменяет текущий образ процесса новым образом процесса.)
4. `read()` - пытается прочитать заданное число байт из файлового дескриптора в буфер.
5. `close()` - закрывает файловый дескриптор, который после этого не ссылается ни на один и файл и может быть использован повторно.
6. `dup2(oldfd, newfd)` - `dup2()` - закрывает файл, связанный с дескриптором `newfd` (если он был открыт) и записывает ссылку `oldfd` в `newfd`.

Общий метод и алгоритм решения

Чтение имени файла из консоли. Открытие файла с заданным именем на чтение и создание файлового дескриптора. Создание пайпа. Создание нового (дочернего) процесса. Передача дочернему процессу файлового дескриптора и пайпа. Считывание в дочернем процессе переданной информации и переопределение потока ввода на открытый файл и переопределение потока вывода на запись в пайп. Построчное считывание и выполнение команд вида :«число число ... число<endline>». Запись результатов команд на пайп с дополнительным разграничением “\n” между ответов. В родительском процессе считывание из пайпа переданной информации дочерним процессом и последующим вывод в стандартный поток вывода.

Исходный код

main.cpp

#include <unistd.h>

#include <iostream>

#include <fcntl.h>

using namespace std;

int main(){

```
    string name;
    cin >> name;
    int file = open(name.c_str(), O_RDONLY);
    int pipefd[2];
    int pip = pipe(pipefd);
    if (pip == -1){
        perror("pipe");
        exit(EXIT_FAILURE);
    }
    int child = fork();
    if (child == -1){
        perror("fork");
        exit(EXIT_FAILURE);
    } else if (child == 0){
        close(pipefd[0]);
```

```

        if (execlp("./child", to_string(file).c_str(),
to_string(pipefd[0]).c_str(), to_string(pipefd[1]).c_str(), NULL) == -1){
            perror("execlp");
            exit(EXIT_FAILURE);
        }
    } else {
        close(pipefd[1]);
        char ch;
        while (read(pipefd[0], &ch, sizeof(char)) > 0 && ch !=
'\0'){

            putchar(ch);
        }
        cout << "\n";
        close(pipefd[0]);
    }
    close(file);
    return 0;
}

```

child.cpp

```

#include <unistd.h>
#include <sstream>
#include <iostream>
#define STDIN 0
#define STDOUT 1
using namespace std;

int main(int argc, char *argv[]){
    int pipefd[2];
    pipefd[0] = atoi(argv[1]);
    pipefd[1] = atoi(argv[2]);
    dup2(atoi(argv[0]), STDIN);
    dup2(pipefd[1], STDOUT);
    close(pipefd[0]);
    float res, a;

```

```

bool flag = true;

string line;

while (getline(cin, line)){
    if (!flag){
        cout << res << "\n";

        flag = true;
    }

    stringstream ss(line);
    while(ss >> a){
        if (flag){
            flag = false;

            res = a;
        } else {
            if (a == 0){
                exit(0);
            }

            res /= a;
        }
    }

}

cout << res << "\0";

close(pipefd[1]);
}

```

Демонстрация работы программы

```

papik@papik-VirtualBox:~/OSlab2/src$ cat test1.txt
1000 2 5 10 1 1 1
15 7 2
123 1 2 3 4 5
papik@papik-VirtualBox:~/OSlab2/src$ ./a.out
test1.txt
10
1.07143
1.025
papik@papik-VirtualBox:~/OSlab2/src$ cat test2.txt
12300 1.23 10 12 1 1 1 1 1 1
10100 1.01 1.01 10
0.1 0.01 0.2 10
papik@papik-VirtualBox:~/OSlab2/src$ ./a.out
test2.txt
83.3333
990.099
5
papik@papik-VirtualBox:~/OSlab2/src$ cat test3.txt
12 10 7 6 5
120 1.00 2.00 3.00 4.00 5.00
13 1 2 3 0
12 1 1 1
13 1 1 1
14 1 1 1
papik@papik-VirtualBox:~/OSlab2/src$ ./a.out
test3.txt
0.00571429
1

```

Выводы

За время выполнения лабораторной работы я научился управлять процессами в ОС, а также разобрался с обеспечением обмена данных между процессов посредством каналов.