Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №1 по курсу**

**«Операционные системы»**

Группа:М8О-215Б-23

Студент: Венгер Ирина Витальевна

Преподаватель: Миронов Е.С.

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

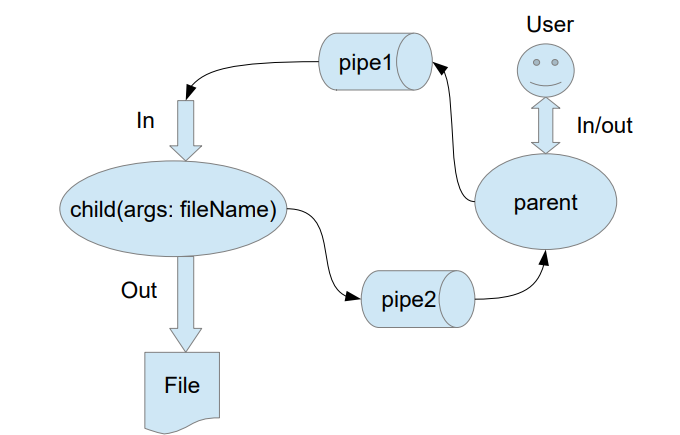
Дата: \_\_\_\_\_\_\_\_\_\_\_\_

Москва, 2024.

# 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.

3 вариант) Пользователь вводит команды вида: «число число число». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс производит деление первого числа, на последующие, а результат выводит в файл. 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 int. Количество чисел может быть произвольным.

## Общие сведения о программе

Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса пишет имя файла, которое будет передано при создании дочернего процесса. Родительский и дочерний процесс представлены разными программами main.c и child.c. Родительский процесс передает команды пользователя через pipe1, который связан с стандартным входным потоком дочернего процесса. Дочерний процесс при необходимости передает данные в родительский процесс через pipe2. Результаты своей работы дочерний процесс пишет в созданный им файл. Программа состоит из 2 файлов: main.c и child.c, в которых разделена логика родительского и дочернего процесса соответственно.

## Общий метод и алгоритм решения

Внутри main.c реализована функция main, в которой происходит создание двух pipe и форка от родительского процесса (создание дочернего процесса) при помощи. Pipe представляют собой массив из 3 элементов – файловых дескрипторов, файловые дескрипторы – неотрицательные числа, которые являются потоком ввода(0), вывода(1), ошибок(2). В данной лабораторной работе использовались только файловые дескрипторы ввода\вывода для упрощения логики программы. При помощи функции fork() мы создаём новый – дочерний процесс, а возвратным значением fork является process id(PID) – целое неотрицательное число. Если процесс создался, то возвращается 0, это значит, что мы находимся внутри ребёнка и мы запускаем child.c, перед этим закрыв конец записи у 1 пайпа и конец чтения у 2 пайпа. При помощи функции dup2() мы можем перенаправить pipe1 на стандартный ввод дочернего процесса. Аналогично у pipe2, но у стандартного вывода. Если же процесс успешно создался, то мы закрываем конец чтения у pipe1 и конец для записи у второго. Далее считываем имя файла, в который будет выведен результат. Если команда не “выход”, то отправляем её в дочерний процесс через pipe. Затем через pipe2 считываем результат и выводим его в род процессе. В конце работы закрываем оба pipe.

## Код программы

Код программы смотрите в приложении 1.

## Использование утилиты strace

Результаты strace смотрите в приложении 2.

По результатам strace видно, какие системные вызовы использует OS для реализации той концепции, которую мы реализовали. Nmap показывает, какие ячейки памяти выделяются в ходе выполнения программы. Функция clone является аналогом fork, только внутри ОС и создаёт новый процесс, который на стадии создания является таким же, как и его предок. Его PID=6458, при успешном создании процесса ему присвоился собственный ID. В ходе работы также можно в параллельном терминале убить этот процесс с помощью kill -9 (PID). Функции close используются при закрытии pipe в конце работы программы. Внутри ОС разные каналы pipe имеют разную нумерацию, чтобы не было путаницы у ОС, с каким pipe нужно работать.

## Демонстрация работы программы

## (base) boopie@MacBook-Air-Irina src % ./main

## Введите имя файла: file.txt

## Введите числа через пробел или 'выход' для завершения : 100 10 5

## Результат деления: 2

## Введите числа через пробел или 'выход' для завершения : 90 8 7

## Результат деления: 1

## Введите числа через пробел или 'выход' для завершения : 70 90 3 0

## Результат деления: Деление на 0 запрещено. Выход из child.

Введите числа через пробел или 'выход' для завершения : выход

## Вывод

В данной лабораторной работе я познакомилась с процессами и операциями с ними. Научилась налаживать взаимодействие между родительским и дочерним процессами, а также передавать необходимые данные посредством pipe. Также в ходе лабораторной работы использовалась утилита strace, которая наглядно и вполне понятно показывает системные вызовы. Благодаря ней можно отслеживать действия процессов и улучшать понимание того, какой процесс с каким ресурсом взаимодействует. Полученные навыки могут сильно помочь в будущем при работе с крупными проектами, где процессов может быть крайне много.

## Приложения

Приложение 1 – код программы:

main.c

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <string.h>

#define MAX\_COMMAND\_LEN 256

int main() {

    int pipe1[2], pipe2[2];

    pid\_t pid;

    // Создаём  пайп 1 и пайп2

    if (pipe(pipe1) == -1 || pipe(pipe2) == -1) {

        perror("Ошибка");

        exit(1);

    }

    // форкаем и создаём дочерний процесс

    pid = fork();

    if (pid < 0) {

        perror("Ошибка создания процесса");

        exit(1);

    }

    if (pid == 0) {

        // Закрываем конец для записи у 1 пайпа, и закрываем конец чтения 2 пайпа

        close(pipe1[1]);

        close(pipe2[0]);

        // Перенаправляем pipe1[0] на стандартный ввод дочернего процесса

        dup2(pipe1[0], STDIN\_FILENO);

        close(pipe1[0]);

        // делаем то же самое, но для стандартного вывода дочернего процесса

        dup2(pipe2[1], STDOUT\_FILENO);

        close(pipe2[1]);

        // запускаем дочерний процесс

        execl("./child", "child", NULL);

        perror("Ошибка");

        exit(1);

    } else {

        //закрываем конец для чтения у 1 пайпа, и конец для записи у второго

        close(pipe1[0]);

        close(pipe2[1]);

        char filename[MAX\_COMMAND\_LEN];

        printf("Введите имя файла: ");

        fgets(filename, sizeof(filename), stdin);

        filename[strcspn(filename, "\n")] = 0;

        //отправляем имя файла, которое мы ввели через 1 пайп

        write(pipe1[1], filename, strlen(filename) + 1);

        char command[MAX\_COMMAND\_LEN];

        while (1) {

            printf("Введите числа через пробел или ‘выход’ для завершения: ");

            fgets(command, sizeof(command), stdin);

            // проверка на выход

            if (strstr(command, "выход") != NULL) {

                write(pipe1[1], "выход", strlen("выход") + 1);

                break;

            }

            //отправляем команду в дочерний проесс через первый пайп

            write(pipe1[1], command, strlen(command) + 1);

            // считываем результат из child через pipe2

            char result[MAX\_COMMAND\_LEN];

            read(pipe2[0], result, sizeof(result));

            printf("Результат деления: %s\n", result);

        }

        // ожидание завершения процесса child

        wait(NULL);

        close(pipe1[1]);

        close(pipe2[0]);

    }

    return 0;

}

сhild.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#define MAX\_COMMAND\_LEN 256

int main() {

    char filename[MAX\_COMMAND\_LEN];

    char command[MAX\_COMMAND\_LEN];

    //считываем имя файла из 1 пайпа

    read(STDIN\_FILENO, filename, sizeof(filename));

    FILE \*file = fopen(filename, "a");

    if (file == NULL) {

        perror("Ошибка открытия файла");

        exit(1);

    }

    while (1) {

        // считали команду через первый пайп

        read(STDIN\_FILENO, command, sizeof(command));

        // Проверка на exit

        if (strcmp(command, "выход") == 0) {

            printf("Выходим из child...\n");

            break;

        }

        // делим команду на числа, с которыми будем работать

        int num1, num2, result;

        char \*token = strtok(command, " ");

        num1 = atoi(token);

        result = num1;

        while ((token = strtok(NULL, " ")) != NULL) {

            num2 = atoi(token);

            if (num2 == 0) {

                printf("Деление на 0 запрещено. Выход из child.\n");

                fprintf(file, "Деление на 0 запрещено. Выход из child.\n");

                fclose(file);

                exit(1);

            }

            result /= num2;

        }

        fprintf(file, "Результат деления: %d\n", result);

        fflush(file);

        // отправка результата в родительский процесс

        char result\_str[MAX\_COMMAND\_LEN];

        sprintf(result\_str, "%d", result);

        write(STDOUT\_FILENO, result\_str, strlen(result\_str) + 1);

    }

    fclose(file);

    return 0;

}

Приложение 2 – результаты strace

