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: Стрыгин Д.Д.

Группа: М8О–306Б–19

Вариант: 16

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2021

**Постановка задачи**

**Цель работы**

Приобретение практических навыков в:

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

**Задание**

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.

Родительский процесс создает дочерний процесс. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child проверяет строки на валидность правилу. Если строка соответствует правилу, то она выводится в стандартный поток вывода дочернего процесса, иначе в pipe2 выводится информация об ошибке. Родительский процесс полученные от child ошибки выводит в стандартный поток вывода.

Правило проверки: строка должна оканчиваться на «.» или «;»

**Общие сведения о программе**

Программа компилируется из файла lab2\_nix.c. Также используется заголовочные файлы: unistd.h, stdio.h , stdlib.h, ctype.h. В программе используются следующие системные вызовы:

1. **fork** - создает копию текущего процесса, который является дочерним процессом для текущего процесса
2. **pipe** - создаёт однонаправленный канал данных, который можно использовать для взаимодействия между процессами.
3. **fflush** - если поток связан с файлом, открытым для записи, то вызов приводит к физической записи содержимого буфера в файл. Если же поток указывает на вводимый файл, то очищается входной буфер.
4. **open** - открывает файл, аргументом можно задать, открыть файл на запись, чтение.
5. **close** - закрывает файл.
6. **read** - читает количество байт(третий аргумент) из файла с файловым дескриптором(первый аргумент) в область памяти(второй агрумент).
7. **write -**  записывает в файл с файловым дескриптором(первый аргумент) из области памяти(второй аргумент) количество байт(третий аргумент).
8. **perror –** вывод сообщения об ошибке.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы fork, pipe, open, close, read, write.
2. Написать программу, которая будет работать с 3-мя процессами: один родительский и два дочерних, процессы связываются между собой при помощи pipe-ов.
3. Организовать работу с выделением памяти под строку неопределенной длины и запись длины в массив строки в качестве первого элемента для передачи между процессами через pipe.

**Основные файлы программы**

**lab2.c:**

#include <stdio.h>

#include <unistd.h>

#include <sys/wait.h>

#include <stdlib.h>

#include <stdbool.h>

#include <fcntl.h>

// Функция чтения строки и подсчёта её размера

int getStr(char \*\*str){

    int size = 0;    // Счётчик размера строки

    char a = '\0';    // Временное хранилище символа

    int strSize = 10;     // Начальное простраство строки

    (\*str) = malloc(sizeof(char) \* strSize);

    if((\*str) == NULL){     // Проверка на память

        write(STDERR\_FILENO, "memory error\n", sizeof(char) \* 14);

        exit(-1);

    }

    while(read(0, &a, 1) == 1){    // Делаем посимвольное прочтение со стандартного ввода

        if(a == '\n'){

            break;

        }

        (\*str)[size] = a;    // Присвоеваем каждый символ ячейке массива

        size++;

        if(size == strSize){     // Увеличиваем область памяти для строки

            strSize = strSize \* 3 / 2;

            (\*str) = realloc((\*str), sizeof(char) \* strSize);

            if((\*str) == NULL){

                write(STDERR\_FILENO, "memeory error\n", sizeof(char) \* 14);

                exit(-1);

            }

        }

    }

    if(size == 0){     // Проверка на ввод нуля элементов

        free((\*str));

        (\*str) = NULL;

        return -1;

    } else {

        (\*str)[size] = '\0'; // Символ окончания строки

        (\*str) = realloc((\*str), sizeof(char) \* (size + 1)); // Сокращаем область памяти до необходимой

        if((\*str) == NULL){

            write(STDERR\_FILENO, "memory error\n", sizeof(char) \* 14);

            exit(-1);

        }

    }

    return size;

}

//родительский процесс

void parent(int \*pipe1, int \*pipe2, int fd){

    close(pipe1[0]);   // Закрываем лишние дискрипторы чтения-записи

    close(pipe2[1]);

    char \*errorMess = "Error\n";

    while(1){

        char \*str;

        int size = getStr(&str);    // Получаем длину строки

        write(pipe1[1], &size, sizeof(int));

        if(size < 0){

            break;

        }

        if(size == 0){

            continue;

        }

        write(pipe1[1], str, sizeof(char) \* size);

        free(str);

        bool result;

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

        if(!result){

            write(fd, errorMess, sizeof(char) \* 6);

        }

    }

    wait(NULL);

    close(fd);

    close(pipe1[1]);

    close(pipe2[0]);

}

//дочерний процесс

void child(int \*pipe1, int \*pipe2, int fd){

    close(pipe1[1]);

    close(pipe2[0]);

    while(1){     //

        int size;

        bool result;

        char \*str;

        read(pipe1[0], &size, sizeof(int));

        if(size < 0){

            break;

        }

        str = malloc(sizeof(char) \* size);

        read(pipe1[0], str, sizeof(char) \* size);

        if((str[size - 1] != ';' && str[size - 1] != '.') || size == 0){

            result = false;

            write(pipe2[1], &result, sizeof(bool));

        } else {

            result = true;

            write(pipe2[1], &result, sizeof(bool));

            write(fd, str, sizeof(char) \* size);

            write(fd, "\n", sizeof(char));

        }

        free(str);

    }

    close(fd);

    close(pipe1[0]);

    close(pipe2[1]);

}

int main(){

    int fd1[2], fd2[2]; // Файловые дискрипоры

    int pid, fd;    // pid переменная для проверки и работы с процессами

    char \*filename = NULL;    // Указатель на область памяти файл

    while(!filename){     // Считываем в него строку(название)

        getStr(&filename);

    }

    fd = open(filename, O\_CREAT | O\_APPEND | O\_WRONLY, S\_IWUSR | S\_IRUSR); // Открываем файл

    if(pipe(fd1) == -1){    // Создаём каналы

        perror("pipe1 error");

    }

    if(pipe(fd2) == -1){

        perror("pipe2 error");

    }

    pid = fork();     // Создаём родительский и дочерний процессы

    if(pid == -1){

        perror("fork error");

    } else if(pid == 0){

        child(fd1, fd2, fd);

    } else {

        parent(fd1, fd2, fd);

    }

    free(filename);

    return 0;

}

**Пример работы**

den@DESKTOP-1B5EV3F:/mnt/c/Users/danst/OneDrive/Документы/GitHub/OC/lab2$ gcc -Wall -pedantic -Werror lab2.c

den@DESKTOP-1B5EV3F:/mnt/c/Users/danst/OneDrive/Документы/GitHub/OC/lab2$ ./a.out

test.txt

Hello world;

Hi master.

Linux is the one,

Windows my first

**test.txt**

Hello world;

Hi master.

Error

Error

**Вывод**

Существуют специальные системные вызовы(fork) для создания процессов, также существуют специальные каналы pipe, которые позволяют связать процессы и обмениваться данными при помощи этих pipe-ов. При использовании fork важно помнить, что фактически создается копию вашего текущего процесса и неправильная работа может привести к неожиданным результатам и последствиям, однако создание процессов очень удобно, когда вам нужно выполнять несколько действий параллельно. Также у каждого процесса есть свой id, по которому его можно определить. Также важно работать с чтением и записью из файла, помня что read, write возвращает количество успешно считанных/записанных байт и оно не обязательно равно тому значению, которое вы указали. Также важно не забывать, что открыв файл при помощи open, важно не забыть его потом закрыть.