Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №2 по курсу**

**«Операционные системы»**

**МЕЖПРОЦЕССОРНОЕ ВЗАИМОДЕЙСТВИЕ**

Студент: Садаков А. А.

Группа: М8О–206Б–19

Вариант: 15

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2020.

**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).

Родительский процесс создаёт 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 начинаться с заглавной буквы.

**Общие сведения о программе**

Программа компилируется из файла main.c. Также используется заголовочные файлы: unistd.h, stdio.h, stdlib.h, sys/types.h, sys/wait.h, fcntl.h, stdbool.h . В программе используются следующие системные вызовы:

1. **pipe** ­­–– принимает массив из двух целых чисел, в случае успеха массив будет содержать два файловых дескриптора, которые будут использоваться для конвейера, первое число в массиве предназначено для чтения, второе для записи, а также вернётся 0. В случае неуспеха вернётся -1.
2. **fork ––** создаёт новый процесс, который является копией родительского процесса, за исключением разных process ID и parent process ID. В случае успеха fork() возвращает 0 для ребёнка, число больше 0 для родителя – child ID, в случае ошибки возвращает -1.
3. **close ––** принимает файловый дескриптор в качестве аргумента, удаляет файловый дескриптор из таблицы дескрипторов, в случае успеха вернёт 0, в случае неуспеха вернёт -1.
4. **open ––** создаёт или открывает файл, если он был создан. В качестве аргументов принимает путь до файла, режим доступа (запись, чтение и т.п.), модификатор доступа ( при создании можно указать права для файла ). Возвращает в случае успеха файловый дескриптор – положительное число, иначе возвращает -1.
5. **read ––** предназначена для чтения какого-то числа байт из файла, принимает в качестве аргументов файловый дескриптор, буфер, в который будут записаны данные и число байт. В случае успеха вернёт число прочитанных байт, иначе -1.
6. **write ––** предназначена для записи какого-то числа байт в файл, принимает в качестве аргументов файловый дескриптор, буфер, из которого будут считаны данные для записи и число байт. В случае успеха вернёт число записанных байт, иначе -1.
7. **wait –** приостанавливает выполнение текущего процесса до тех пор, пока дочерний процесс не завершится, или до появления сигнала, который либо завершает текущий процесс, либо требует вызвать функцию-обработчик. Если дочерний процесс к моменту вызова функции уже завершился (так называемый "зомби", то функция немедленно возвращается.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить функции pipe, fork, wait, perror, open.
2. Написать функцию ввода строк и отправки дочернему процессу (parent)
3. Написать функцию получения строки от родителя, обработки и отправки результата в файл (child)
4. Организовать обмен информацией между процессами при помощи pipe.
5. Реализовать обработку ошибок.

**Основные файлы программы**

**main.c**

#include <stdio.h>

#include <string.h>

#include <unistd.h>

#include <sys/wait.h>

#include <stdlib.h>

#include <stdbool.h>

#include <fcntl.h>

//родительский процесс

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

    close(pipe1[0]);

    close(pipe2[1]);

    char\* errorMess = "ERROR\n";

    while(1){

        char str[100] = {'\0'};

        fgets(str, 100, stdin);

        int size = strlen(str);

        if (size == 0) {

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

            break;

        }

        if (size == 1) {

            if (str[0] == '\n') {

                continue;

            }

        }

        str[size - 1] = '\0';

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

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

        bool result;

        if (read(pipe2[0], &result, sizeof(bool)) != sizeof(bool)) {

            fprintf(stderr, "read error in parent process\n");

            close(fd);

            close(pipe1[1]);

            close(pipe2[0]);

            exit(-1);

        }

        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[100];

        if (read(pipe1[0], &size, sizeof(int)) != sizeof(int)) {

            fprintf(stderr, "read error in child process\n");

            close(fd);

            close(pipe1[0]);

            close(pipe2[1]);

            exit(-1);

        }

        if (size == 0) {

            break;

        }

        if (read(pipe1[0], str, sizeof(char) \* size) != sizeof(char) \* size) {

            fprintf(stderr, "read error in child process\n");

            close(fd);

            close(pipe1[0]);

            close(pipe2[1]);

            exit(-2);

        }

        if(str[0] < 'A' || str[0] > 'Z' || 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));

        }

    }

    close(fd);

    close(pipe1[0]);

    close(pipe2[1]);

}

int main(){

    char filename[30] = {'\0'};

    fgets(filename, 30, stdin);

    size\_t size = strlen(filename);

    if (size == 0 || (size == 1 && filename[0] == '\n')) {

        perror("file open error");

    }

    filename[size - 1] = '\0';

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

    if (fd == -1) {

        perror("open error\n");

    }

int fd1[2], fd2[2];

    int pid;

    if(pipe(fd1) == -1){

        perror("pipe1 error");

    }

    if(pipe(fd2) == -1){

        perror("pipe2 error");

    }

    pid = fork();

    if(pid == -1){

        perror("FORK");

    } else if(pid == 0){

        child(fd1, fd2, fd);

    } else {

        parent(fd1, fd2, fd);

    }

    return 0;

}

**Пример работы**

**Тест1** (проверка на открытие файла без прав).

aleksandr201@LAPTOP-M1VJISD6:~/OS/lab2$ ./a.out

FILE

file open error: Permission denied

aleksandr201@LAPTOP-M1VJISD6:~/OS/lab2$ ls FILE

FILE

aleksandr201@LAPTOP-M1VJISD6:~/OS/lab2$ ls -l FILE

---------- 1 aleksandr201 aleksandr201 6 Apr 21 21:05 FILE

**Тест2** (проверка на работоспособность).

aleksandr201@LAPTOP-M1VJISD6:~/OS/lab2$ cat test.txt

ans

Afaefaefe

aefef

aefe

AAAA

afe

sgrgrgr

A

aleksandr201@LAPTOP-M1VJISD6:~/OS/lab2$ ./a.out < test.txt

aleksandr201@LAPTOP-M1VJISD6:~/OS/lab2$ cat ans

Afaefaefe

ERROR

ERROR

AAAA

ERROR

ERROR

A

**Вывод**

Процессы позволяют нам выполнять несколько параллельных операций, даже при работе на одноядерном процессоре. Но для того, чтобы они выполняли задачу корректно, нужно синхронизировать их работу. Для этого хорошо подходят каналы. Они позволяют передавать информацию между процессами, что помогает синхронизировать их работу.