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: Юнусов Руслан Асифович
Группа: М8О-209Б-23
Вариант: 5
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2024

Содержание

Репозиторий..... 3
Постановка задачи..... 3
Общие сведения о программе.....4
Общий метод и алгоритм решения..... 5
Исходный код..... 5
Демонстрация работы программы..... 8
Выводы..... 9

Репозиторий

https://github.com/Rissochek/OSLabs/tree/main/lab1

Постановка задачи

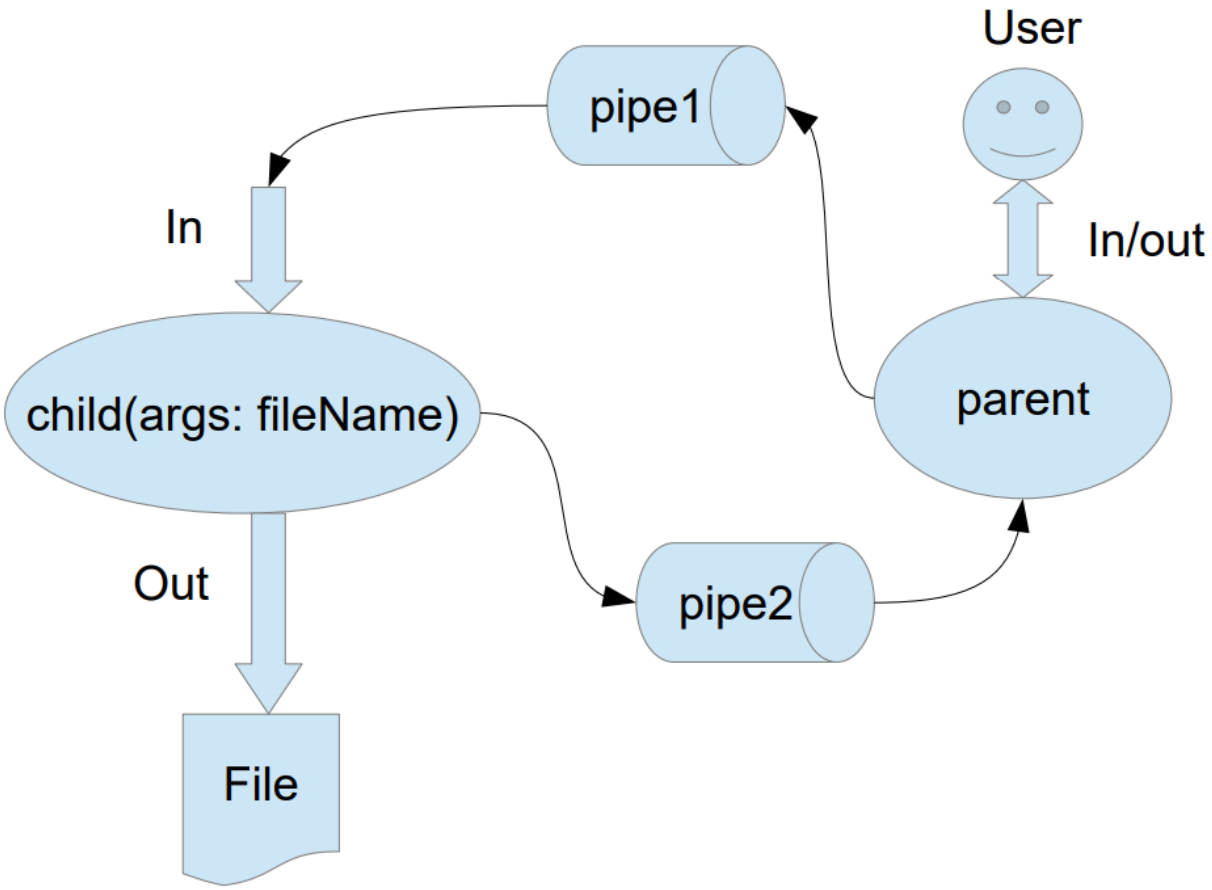
Цель работы

Приобретение практических навыков в:

1. Управление процессами в ОС
2. 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.



4 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 float. Количество чисел может быть произвольным.

Общие сведения о программе

Программа компилируется из файла main.c. Также используется заголовочные файлы: unistd.h, stdio.h , stdlib.h, ctype.h, wait.h, stdbool.h, unistd.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. **dup2 -** перенаправляет вывод родительского файла в дочерний файл, а также вывод дочернего файла в родительский
4. **execv -** запускает дочерний процесс из отдельного файла.
5. **close** - закрывает файл, а также файловые дескрипторы.
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.
9. **exit -** завершает выполнение программы.
10. **wait -** получает статус завершения дочернего процесса.

Общий метод и алгоритм решения

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы fork, pipe, execv, close, read, write, dup2.
2. Написать программу, которая будет работать с 2-мя процессами: один из них родительский и один дочерний, процессы связываются между собой при помощи pipe-ов.
3. Организовать работу с выделением памяти под строку неопределенной длины. Грамотно передать данные между процессами. Реализовать функцию проверки строки на наличие нулей, что противоречит правилам деления. Провести калькуляцию. Провести работу связанную с файлами и записать в файл результат вычислений.
4. Освободить всю выделенную память, а также проверить на наличие утечек при помощи специализированных программ.

Исходный код

main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/wait.h>
#include <unistd.h>
#include <ctype.h>
#include <stdbool.h>

int main(int argc, char *argv[])
{
   int    pipefd_from_parent_to_child[2];
```

```
int    pipefd_from_child_to_parent[2];
char   buf;
char  *input_data = malloc(sizeof(char) * 50);
char   filename[20];
size_t   size = 50;
int    counter = 0;
pid_t  cpid;

if (argc != 2) {
    fprintf(stderr, "Usage: %s <filename>\n", argv[0]);
    exit(EXIT_FAILURE);
}

if (pipe(pipefd_from_parent_to_child) == -1) {
    perror("pipe1");
    exit(EXIT_FAILURE);
}

if (pipe(pipefd_from_child_to_parent) == -1) {
    perror("pipe2");
    exit(EXIT_FAILURE);
}

cpid = fork();
if (cpid == -1) {
    perror("fork");
    exit(EXIT_FAILURE);
}
```

```
if (cpid == 0) {
    close(pipefd_from_parent_to_child[1]);
    close(pipefd_from_child_to_parent[0]);

    dup2(pipefd_from_parent_to_child[0], STDIN_FILENO);
    dup2(pipefd_from_child_to_parent[1], STDOUT_FILENO);

    close(pipefd_from_parent_to_child[0]);
    close(pipefd_from_child_to_parent[1]);
    char *args[] = {"./child", NULL};
    if (execv(args[0], args) == -1) {
        perror("execv");
        exit(EXIT_FAILURE);
    }
    close(pipefd_from_parent_to_child[0]);
    close(pipefd_from_child_to_parent[1]);
} else {
    close(pipefd_from_parent_to_child[0]);
    close(pipefd_from_child_to_parent[1]);
    char ch;
    while ((ch = getchar()) != EOF && ch != '\n'){
        if (counter < size){
            input_data[counter++] = ch;
        } else{
            size *= 2;
            char *buffer = realloc(input_data, size);
            if (buffer == NULL) {
                perror("realloc failed");
                exit(EXIT_FAILURE);
```

```
            }else{
                input_data = buffer;
                input_data[counter++] = ch;
            }
        }
    }
    input_data[counter] = '\0';
    write(pipefd_from_parent_to_child[1], argv[1], strlen(argv[1]));
    write(pipefd_from_parent_to_child[1], "|", 1);
    write(pipefd_from_parent_to_child[1], input_data, strlen(input_data));

    close(pipefd_from_parent_to_child[1]);
    close(pipefd_from_child_to_parent[0]);
    int status = 0;
    wait(&status);
    free(input_data);
    if (WIFEXITED(status) && WEXITSTATUS(status) == 0) {
        exit(EXIT_SUCCESS);
    } else {
        exit(EXIT_FAILURE);
    }
  }
}
```

Демонстрация работы программы

```
parol1@riss:~/Labs/OSLabs/lab1$ make build
gcc  -o parent main.c
gcc  -o child child_program.c
```

```
parol1@riss:~/Labs/OSLabs/lab1$ ./parent test.txt
2 2
parol1@riss:~/Labs/OSLabs/lab1$ cat test.txt
1.000000
```

Выводы

В ходе проделанной работы на Unix-подобной ОС я узнал много нового о работе процессов, а также смог реализовать программу, которая реализует работу, приведенную в задании моего варианта. В ходе выполнения задания я ознакомился с множеством ранее неизвестных понятий, таких как файловые дескрипторы, а также функций: fork, который создает дочерний процесс, связанные с родителем по pid; dup2, перенаправляющий потоки вывода и ввода; execv, запускающий выполнение программы.