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: Дубровин Дмитрий
Группа: М80-207Б-21
Вариант: 1
Преподаватель: Миронов Евгений Сергеевич
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/1droozd1/os_labs

Постановка задачи

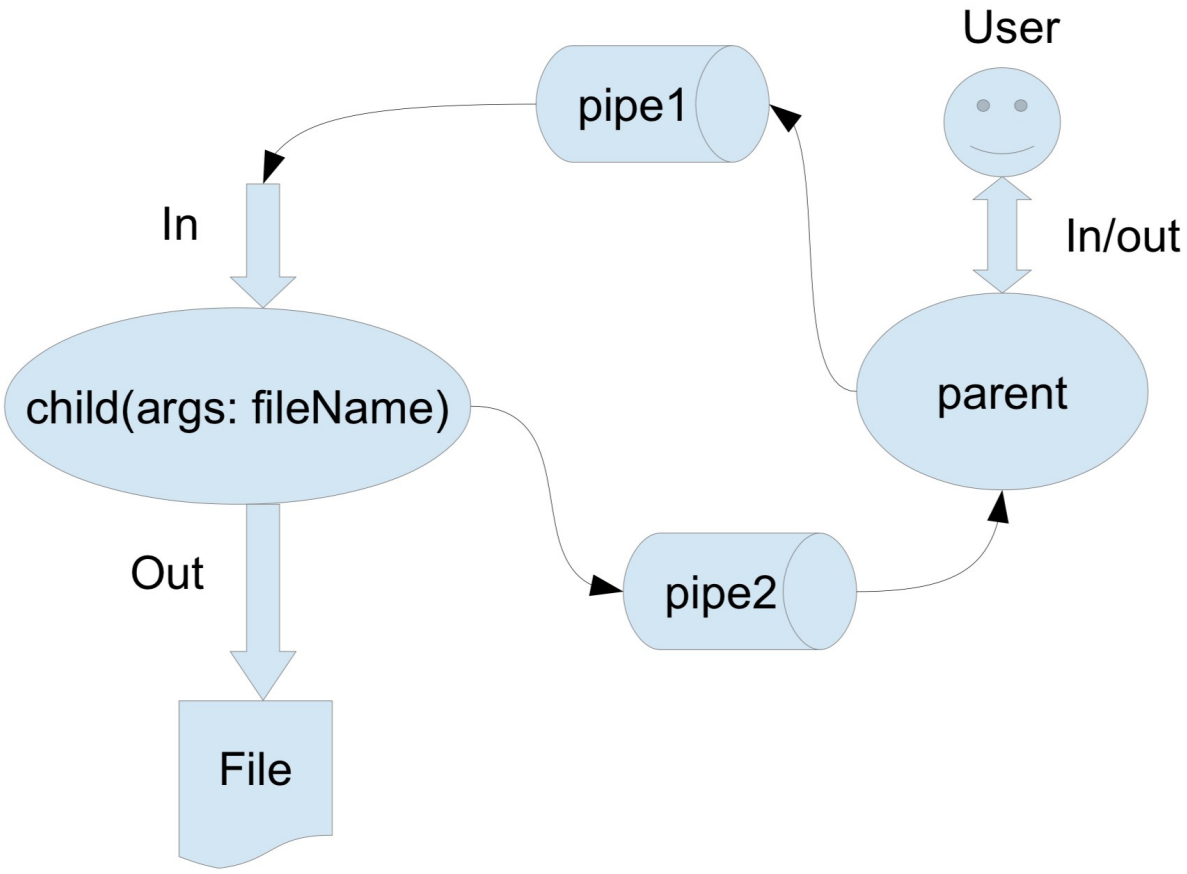
Цель работы

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 также обрабатывать системные ошибки, которые могут возникнуть в результате работы.



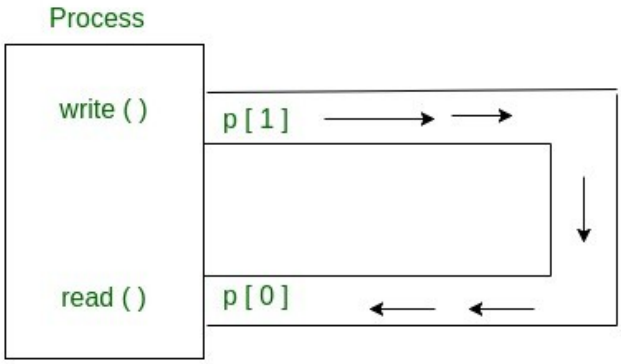
Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса пишет имя файла, которое будет передано при создании дочернего процесса. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс передает команды пользователя 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. Допускается просто открыть файл и писать туда, не перенаправляя стандартный поток вывода.

Задание по варианту: пользователь вводит команды вида: «число число число<endline>». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс считает их сумму и выводит её в файл. Числа имеют тип int. Количество чисел может быть произвольным.

Общие сведения о программе

Программа компилируется из файла main.c. Также используется заголовочные файлы: stdio.h; stdlib.h; string.h; unistd.h, а также файл child_process, который принимает данные от main.c. В программе используются следующие системные вызовы:

1. pipe(fd[2]) - функция для открытия канала связи между двумя процессами, также стоит заметить, что она помещает дескрипторы файла для чтения и записи (соответственно) в fd[0] и fd[1].



2. fork() - создание нового процесса (процесса-потомка), почти идентичного процессу-родителю.
3. snprintf() - направляет данные в символьную строку str, в данной работе использую для образования строк.
4. execlp() - заменяет текущий образ процесса новым образом процесса (можно сказать, что она передает текующий процесс на испольнение другой программы).

5. fprintf() - печатает форматированные данные в поток (файл).
6. fgets() - считывает из потока данных строку пока не встретится символ "новая строка" или не закончится файл или пока длина считываемой строки не превысит ограничение аргумента n.
7. close() - закрывает связанный с дескриптором файл.
8. write() - записывает данный в указанный файл.
9. remove() - удаление файла в текущей директории с указанным имеменем.
10. getpid() - получение id текущего процесса.
11. fclose() - закрывает файл, который был ранее открыт fopen().

Общий метод и алгоритм решения

Для моей работы мне вполне достаточно одного канала, поэтому создаю его с помощью pipe, а также проверяю на наличие ошибок при создании. Далее идет создание дочернего процесса программы с помощью системного вызова fork(), также я проверяю id, которое было возвращено fork(). После чего с помощью утилиты execlp() необходимо передать исполнение дочернему процессу (child_process.c), но так как на вход данная утилита требует имя файла, исполняющего дочерний процесс, и переменные в строчном формате, приведу их нужному формату с помощью snprintf(). Получаю отправленные данные из родительского процесса, и удаляю файл (если такой есть в текущей директории) с аналогичным названием, что было передано ранее. Далее с помощью собственной функции обрабатываю строку и складываю полученные значения в переменную res. После чего создаю необходимый файл (с переданным названием) (а также проверяю на ошибки при создании файла) и записываю туда полученный результат. Не забываю закрыть потоки и созданный файл.

Исходный код

main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>


int main()
{
   pid_t id;
```

```
int pipe1[2];
char file_name[20];
char sequence_of_numbers[100];

// error: can't create pipe
if (pipe(pipe1) == -1) {
    printf("Unable to create pipe\n");
    return 1;
}

id = fork();
// error: can't create child process
if (id < 0) {
    perror("fork error");
    return -1;
}
// Child Process
else if (id == 0) {

    char pipe_1[20], pipe_2[20];
    snprintf(pipe_1, sizeof(pipe_1), "%d", pipe1[0]); //transform char to string format
    snprintf(pipe_2, sizeof(pipe_2), "%d", pipe1[1]);

    execlp ("./child_process", pipe_1, pipe_2, NULL);
    fprintf(stderr, "\nExec didn't work...\n");
    exit(1);
}
// Parent Process
else {
    printf("[Parent Process, id=%d]: Write name of file: ", getpid());
    fgets(file_name, 20, stdin);
    if (file_name[strlen(file_name) - 1] == '\n')
        file_name[strlen(file_name) - 1] = '\0';

    printf("[Parent Process, id=%d]: Write int numbers: ", getpid());
```

```
        fgets(sequence_of_numbers, 100, stdin);

        if (sequence_of_numbers[strlen(sequence_of_numbers) - 1] == '\n')
            sequence_of_numbers[strlen(sequence_of_numbers) - 1] = '\0';
        printf("[Parent Process, id=%d]: File name: %s\n\n", getpid(), file_name);



        close(pipe1[0]);

        write(pipe1[1], file_name, sizeof(file_name));
        write(pipe1[1], sequence_of_numbers, sizeof(sequence_of_numbers));

        close(pipe1[0]);
        close(pipe1[1]);
    }
    return 0;
}
```

child_process.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

int sum_from_char(char *s)
{
    int i = 0, sum = 0, n = 0, flag1 = 0;
    int len = strlen(s);

    for (int i = 0; i < len; ++i) {
        if (s[i] >= '0' && s[i] <= '9' && flag1 == 0) {
            n = n * 10 + (s[i] - '0');
        } else if (flag1 == 1 && s[i] >= '0' && s[i] <= '9') {
            n = n * 10 - (s[i] - '0');
        } else if (s[i] == '-') {
```

```
            flag1 = 1;
        } else if (s[i] == ' ' && flag1 == 1) {
            flag1 = 0;
            sum += n;
            n = 0;
            continue;
        } else if (n) {
            sum += n;
            n = 0;
            flag1 = 0;
        }
    }
    return sum + n;
}

int main(int argc, char const *argv[])
{
    int pipe1[2];
    char read_file_name[20];
    char read_sequence_of_numbers[100];

    pipe1[0] = atoi(argv[0]);
    pipe1[1] = atoi(argv[1]);

    close(pipe1[1]);

    read(pipe1[0], &read_file_name, sizeof(read_file_name));
    read(pipe1[0], &read_sequence_of_numbers, sizeof(read_sequence_of_numbers));

    printf("[Child Process,  id=%d]: File name from the pipe: %s\n", getpid(), read_file_name);
    printf("[Child Process,  id=%d]: Numbers from the pipe: %s\n", getpid(),
read_sequence_of_numbers);


    remove(read_file_name); //if we have the same file
```

```
    int res = sum_from_char(read_sequence_of_numbers);

    FILE *write_res;
    if ((write_res = fopen(read_file_name, "w")) == NULL) {
        printf("Error: can't open file\n");
        exit(1);
    }

    fprintf(write_res, "%d", res);
    fclose(write_res);

    close(pipe1[0]);
    close(pipe1[1]);
    return 0;
}
```

CmakeLists.txt

```
cmake_minimum_required(VERSION 3.5 FATAL_ERROR)

project(lab_2)

add_executable(child_process child_process.c)
add_executable(main main.c)
```

Демонстрация работы программы

```
dr0ozd1@Dmitry-Nitro-AN515-45:~/Coding$ cd Lab_os/lab_2
dr0ozd1@Dmitry-Nitro-AN515-45:~/Coding/Lab_os/lab_2$ cmake src/
-- Configuring done
-- Generating done
-- Build files have been written to: /home/dr0ozd1/Coding/Lab_os/lab_2
dr0ozd1@Dmitry-Nitro-AN515-45:~/Coding/Lab_os/lab_2$ make
Consolidate compiler generated dependencies of target child_process
[ 50%] Built target child_process
Consolidate compiler generated dependencies of target main
```

```
[100%] Built target main
dr0ozd1@Dmitry-Nitro-AN515-45:~/Coding/Lab_os/lab_2$ ./main
[Parent Process, id=10853]: Write name of file: s
[Parent Process, id=10853]: Write int numbers: 1 2 3
[Parent Process, id=10853]: File name: s

[Child Process,  id=10854]: File name from the pipe: s
[Child Process,  id=10854]: Numbers from the pipe: 1 2 3
dr0ozd1@Dmitry-Nitro-AN515-45:~/Coding/Lab_os/lab_2$ ls -l
итого 388
-rwxrwxr-x 1 dr0ozd1 dr0ozd1  16432 окт  2 23:50 child_process
-rw-rw-r-- 1 dr0ozd1 dr0ozd1  13786 окт  2 23:50 CMakeCache.txt
drwxrwxr-x 6 dr0ozd1 dr0ozd1   4096 окт  3 21:39 CMakeFiles
-rw-rw-r-- 1 dr0ozd1 dr0ozd1   1640 окт  2 23:50 cmake_install.cmake
-rwxrwxr-x 1 dr0ozd1 dr0ozd1  16600 окт  2 23:50 main
-rw-rw-r-- 1 dr0ozd1 dr0ozd1   6419 окт  3 21:39 Makefile
-rw-rw-r-- 1 dr0ozd1 dr0ozd1 312921 окт  3 21:38 report.odt
-rw-rw-r-- 1 dr0ozd1 dr0ozd1      1 окт  3 21:40 s
drwxrwxr-x 2 dr0ozd1 dr0ozd1   4096 окт  2 23:48 src
dr0ozd1@Dmitry-Nitro-AN515-45:~/Coding/Lab_os/lab_2$ cat s
6dr0ozd1@Dmitry-Nitro-AN515-45:~/Coding/Lab_os/lab_2$
```

Выводы

В ходе проведения данной лабораторной я работы я познакомился с различными системными вызовами для управления процессами, а также научился передавать данные с помощью pipe().