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: Шандрюк Пётр Николаевич
Группа: М8О-208Б-20
Вариант: 11
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2021

Содержание

1. Репозиторий
1. Постановка задачи
1. Общие сведения о программе
1. Общий метод и алгоритм решения
1. Исходный код
1. Демонстрация работы программы
1. Выводы

Репозиторий

https://github.com/Peter1811/OS

Постановка задачи

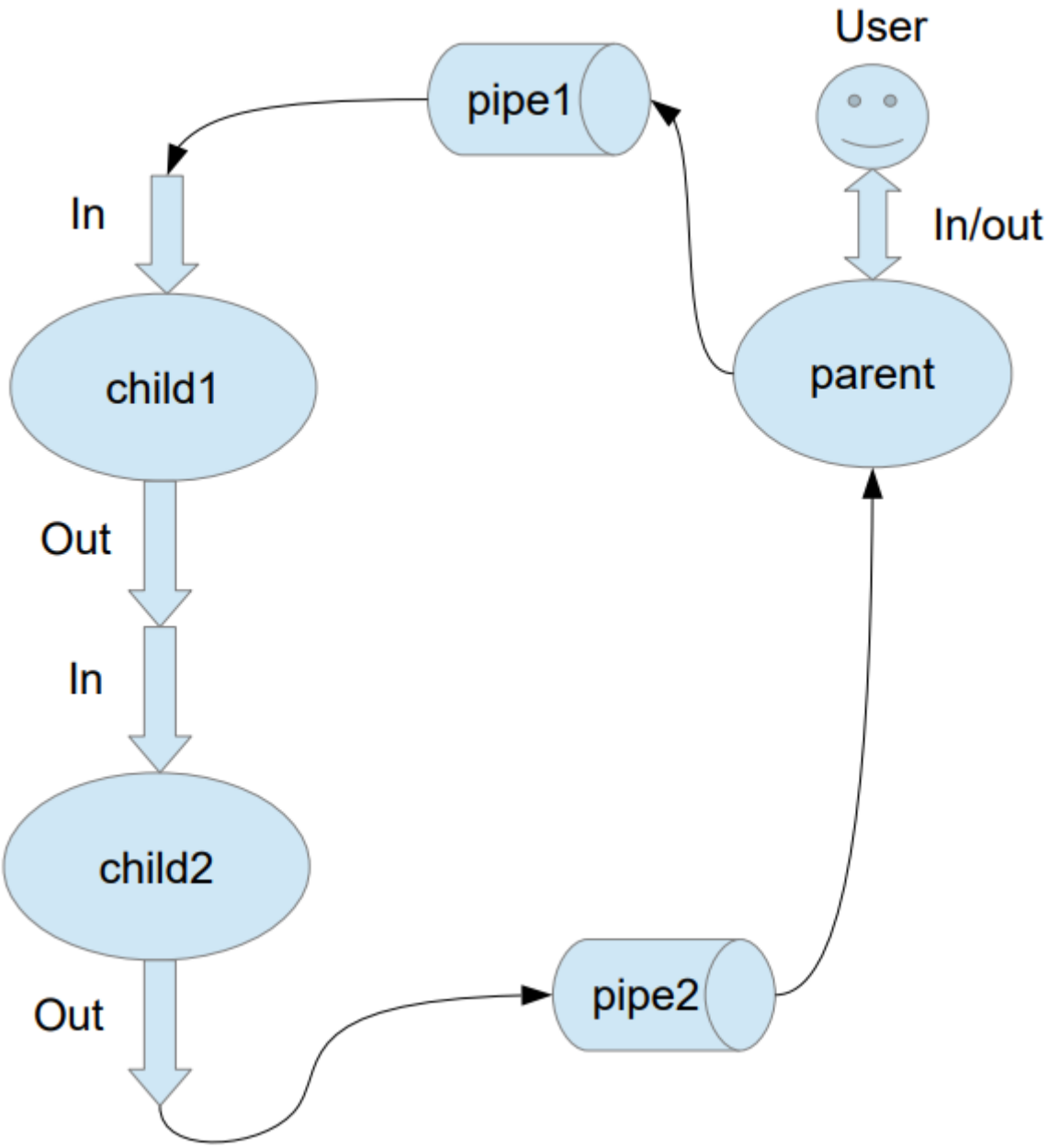
Цель работы

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.


User
pipe1
In
In/out
child1
parent
Out
In
child2
Out
pipe2

11 вариант) Child1 переводит строки в верхний регистр. Child2 превращает все пробельные символы в символ «_».

Общие сведения о программе

Программа компилируется из файла main2.cpp. Также используется библиотеки: unistd.h, algorithm , string, cctype. 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. **close** - закрывает файл.
5. **read** - читает количество байт(третий аргумент) из файла с файловым дескриптором(первый аргумент) в область памяти(второй агрумент).
6. **write -** записывает в файл с файловым дескриптором(первый аргумент) из области памяти(второй аргумент) количество байт(третий аргумент).
7. **perror –** вывод сообщения об ошибке.

Общий метод и алгоритм решения

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы fork, pipe, fflush, 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.

   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.

Исходный код

main2.cpp

```
#include <iostream>
#include <unistd.h>
#include <cstring>
#include <string>
#include <algorithm>
#include <cctype>


using namespace std;

int main(){

    int fd1[2]; // parent -> child_a
    int fd2[2]; // child_a -> child_b
    int fd3[2]; //child_b -> parent
    if ((pipe(fd1) == -1) or (pipe(fd2) == -1) or (pipe(fd3) == -1)) {
        cout << "Error with pipes" << endl;
```

```
        return 1;
    }

    int child_a = fork();

    if (child_a == -1) {
        cout << "Error with child_a fork" << endl;
        return 2;
    } else if (child_a == 0) { // Child A code
        close(fd1[1]);
        close(fd2[0]);
        close(fd3[0]);
        close(fd3[1]);
        int kol_a;
        if (read(fd1[0], &kol_a, sizeof(int)) == -1) {
            cout << "Error with reading in child_a" << endl;
            return 5;
        }
        if (write(fd2[1], &kol_a, sizeof(int)) == -1) {
            cout << "Error with writing in child_a" << endl;
            return 5;
        }
        for (int v = 0; v < kol_a; v++){

            int l;
            if (read(fd1[0], &l, sizeof(int)) == -1) {
                cout << "Error with reading child_a" << endl;
                return 5;
            }
```

```
            if (write(fd2[1], &l, sizeof(int)) == -1) {
                cout << "Error with writing child_a" << endl;
                return 6;
            }
            char *c = new char[l];
            if (read(fd1[0], c, sizeof(char) * l) == -1) {
                cout << "Error with reading child_a" << endl;
                return 5;
            }

            for (int i = 0; i < l; i++) {
                if (( c[i] >= 'a') and (c[i] <= 'z')) {
                    c[i] = c[i]-'a'+'A';
                }
            }
            if (write(fd2[1], c, sizeof(char) * l) == -1) {
                cout << "Error with writing child_a" << endl;
                return 6;
            }
            delete [] c;
        }

        close(fd1[0]);
        close(fd2[1]);

    } else {
        int child_b = fork();
        if (child_b == -1) {
            cout << "Error with child_b fork" << endl;
```

```
        return 3;
    } else if (child_b == 0) { //Child B code
        close(fd1[1]);
        close(fd1[0]);
        close(fd2[1]);
        close(fd3[0]);
        int kol_b;
        if (read(fd2[0], &kol_b, sizeof(int)) == -1) {
            cout << "Error with reading in child_b" << endl;
            return 7;
        }
        for (int gh = 0; gh < kol_b; gh++){
            int len_b;
            if (read(fd2[0], &len_b, sizeof(int)) == -1) {
                cout << "Error with reading in child_b" << endl;
                return 7;
            }
            if (write(fd3[1], &len_b, sizeof(int)) == -1) {
                cout << "Error with writing in child_b" << endl;
                return 8;
            }
            char *c_b = new char[len_b];
            if (read(fd2[0], c_b, sizeof(char) * len_b) == -1) {
                cout << "Error with reading in child_b" << endl;
                return 7;
            }
            for (int j = 0; j < len_b; j++) {
                if (c_b[j] == ' ') {
                    c_b[j] = '_';
```

```
            }
        }
        if (write(fd3[1], c_b, sizeof(char) * len_b) == -1) {
            cout << "Error with writing in child_b" << endl;
            return 8;
        }
        delete [] c_b;
    }
    close(fd2[0]);
    close(fd3[1]);


} else { //Parent code
    close(fd1[0]);
    close(fd2[1]);
    close(fd2[0]);
    close(fd3[1]);
    int k; cin >> k;

    if (write(fd1[1], &k, sizeof(int)) == -1) {
        cout << "Error with writing in parent" << endl;
        return 4;
    }

    for (int m = 0; m < k; m++) {
        string new_s;
        char c;
        while ((c = getchar()) != EOF){
            new_s += c;
```

```
    }
    int t = new_s.length();
    if (write(fd1[1], &t, sizeof(int)) == -1) {
        cout << "Error with writing in parent" << endl;
        return 4;
    }
    if (write(fd1[1], new_s.c_str(), sizeof(char) * t) == -1) {
        cout << "Error with writing in parent" << endl;
        return 4;
    }
}

for (int p = 0; p < k; p++) {
    int len_p;
    if (read(fd3[0], &len_p, sizeof(int)) == -1) {
        cout << "Error with writing in child_b";
        return 9;
    }
    char *c_par = new char[len_p];
    if (read(fd3[0], c_par, sizeof(char) * len_p) == -1) {
        cout << "Error with reading in parent" << endl;
        return 9;
    }

    for (int nn = 0; nn < len_p; nn++) {
        cout << c_par[nn];
    }

    delete [] c_par;
```

```
            }
            close(fd1[1]);
            close(fd3[0]);
        }

    }
    return 0;
}
```

Демонстрация работы программы

```
[Temi4@localhost ~]$ cd /mnt/c/users/peter/desktop/os
[Temi4@localhost 2_lab]$ g++ main2.cpp
[Temi4@localhost 2_lab]$ ./a.out
2
hello world
hahaha  fd

HELLO_WORLD
HAHAHA__FD
```

Выводы

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 канала, помня что read, write возвращает количество успешно считанных/записанных байт и оно не обязательно равно тому значению, которое вы указали. Также важно не забывать закрывать pipe после завершения работы.