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: Ядров Артем Леонидович
Группа: М8О-208Б-20
Вариант: 12
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2021

Содержание

Репозиторий

https://github.com/Yadroff/OS/tree/master/2_lab

Постановка задачи

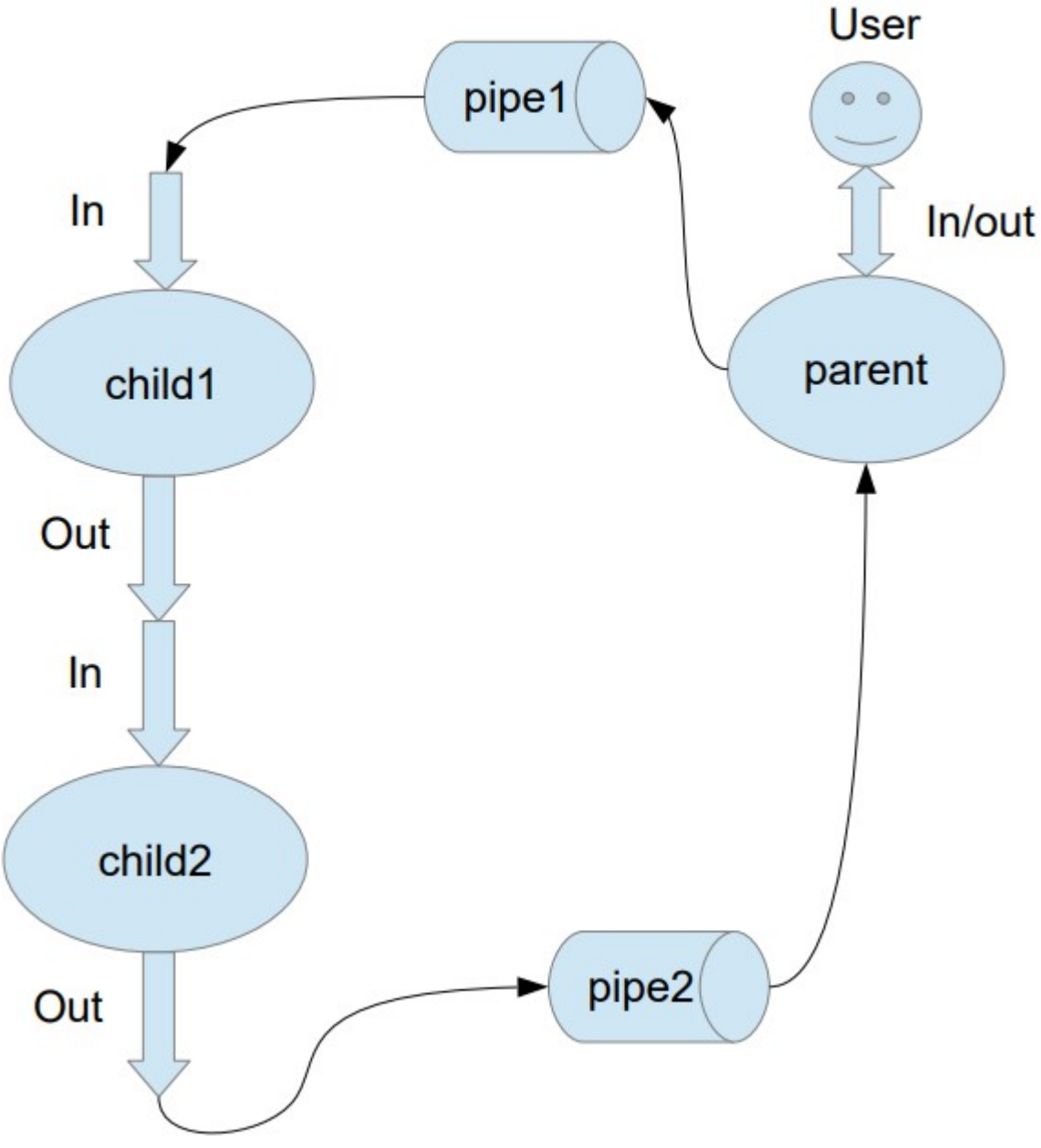
Цель работы

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.



12 вариант) Child1 переводит строки в верхний регистр. Child2 убирает все задвоенные пробелы.

Общие сведения о программе

Программа компилируется из файла main.cpp. 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. **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

main.cpp

```
#include <unistd.h>
#include <stdio.h>
#include <cctype>
#include <algorithm>

int main() {
    int err = 0;
    int fd[2];
    int fd_1[2];
    int fd_2[2];
    pipe(fd); // pipe между дочерними потоками
    pipe(fd_1); // pipe между родительским и первым
    pipe(fd_2); // pipe между родительским и вторым
    pid_t pid_1 = 0; // первый поток
    pid_t pid_2 = 0; // второй поток
    if ((pid_1 = fork()) > 0) { // создаем первый процесс
        if ((pid_2 = fork()) > 0) { //создаем второй процесс
            // Parent
            char *in = (char *) malloc(sizeof(char) * 2);
            in[0] = 0;
            char c;
            while ((c = getchar()) != EOF) {
                in[0] += 1;
                in[in[0]] = c;
```

```
            in = (char *) realloc(in, (in[0] + 2) * sizeof(char));
        }
        in[in[0]] = '\0';
        err = write(fd_1[1], in, (in[0] + 2) * sizeof(char)); // кидаем в pipe fd_1
        if (err == -1){
            printf("Write error\n");
            exit(-1);
        }
        char *out = (char *) malloc(sizeof(char));
        err = read(fd_2[0], &out[0], sizeof(char)); // вытаскиваем из pipe fd_2
        if (err == -1){
            printf("Read error\n");
            exit(-1);
        }
        out = (char *) realloc(out, (out[0] + 2) * sizeof(char));
        for (int i = 1; i < out[0] + 1; ++i) {
            err = read(fd_2[0], &out[i], sizeof(char));
            if (err == -1){
                printf("Read error\n");
                exit(-1);
            }
            printf("%c", out[i]);
        }
        printf("\n");
        close(fd_2[0]);
        close(fd_1[1]);
        free(in);
        free(out);
    } else if (pid_2 == 0) { //Child_2
```

```
fflush(stdin);
fflush(stdout);
char *in = (char *) malloc(sizeof(char));
err = read(fd[0], &in[0], sizeof(char)); // считываем из pipe fd
if (err == -1){
    printf("Read error\n");
    exit(-1);
}
in = (char *) realloc(in, (in[0] + 2) * sizeof(char));
for (int i = 1; i < in[0] + 1; i++) {
    err = read(fd[0], &in[i], sizeof(char));
    if (err == -1){
        printf("Read error\n");
        exit(-1);
    }
}
char *out = (char *) malloc(2 * sizeof(char));
out[0] = 0;
for (int i = 1; i < in[0]; i++) { // преобразование
    if (in[i] == ' ' && in[i + 1] == ' ') {
        i++;
        continue; //
    }
    out[0]++;
    out[out[0]] = in[i];
    out = (char *) realloc(out, (out[0] + 2) * sizeof(out));
}
out[0]++;
out[out[0]] = '\0';
```

```
        err = write(fd_2[1], out, (out[0] + 2) * (sizeof(char))); // кидаем в pipe fd_2
        if (err == -1){
            printf("Write error\n");
            exit(-1);
        }
        fflush(stdout);
        close(fd_2[1]); // закрываем вход и выход pipe'ов
        close(fd[0]);
        free(in);
        free(out);
    } else { //Parent
        printf("Fork error\n");
        exit(-1);
    }
} else if (pid_1 == 0) { //Child_1
    char *in = (char *) malloc(sizeof(char));
    err = read(fd_1[0], &in[0], sizeof(char));
    if (err == -1){
        printf("Read error\n");
        exit(-1);
    }
    in = (char *) realloc(in, (in[0] + 2) * sizeof(char));
    char *out = (char *) malloc((in[0] + 2) * sizeof(char));
    out[0] = in[0];
    for (int i = 1; i < in[0] + 1; i++) { // преобразование
        err = read(fd_1[0], &in[i], sizeof(char));
        if (err == -1){
            printf("Read error\n");
```

```
            exit(-1);
        }
        out[i] = toupper(in[i]);
    }
    err = write(fd[1], out, (out[0] + 2) * sizeof(char)); // в pipe между дочерними процессами
    if (err == -1){
        printf("Read error\n");
        exit(-1);
    }
    close(fd_1[0]);
    close(fd[1]);
    free(in);
    free(out);
  } else {
    printf("Fork error\n");
    exit(-1);
  }
  return 0;
}
```

Демонстрация работы программы

```
[Temi4@localhost ~]$ cd CLionProjects/OS/2_lab/
[Temi4@localhost 2_lab]$ cat test.txt
Hello   World!
I  am    learning OS
I love C++
And   you)))
[Temi4@localhost 2_lab]$ g++ main.cpp
[Temi4@localhost 2_lab]$ ./a.out < test.txt
HELLO WORLD!
IAMLEARNING OS
```

I LOVE C++
AND YOU)))

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.