МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 "Компьютерные науки и прикладная математика"
Кафедра 806 "Вычислительная математика и программирование"

Лабораторная работа №1

По курсу «Операционные системы»

Студент: Теребаев К. Д.
Группа: М8О-203Б-22
Преподаватель: Миронов Е. С.
Дата: _____
Оценка: _____
Подпись: _____

Москва, 2024

Цель работы

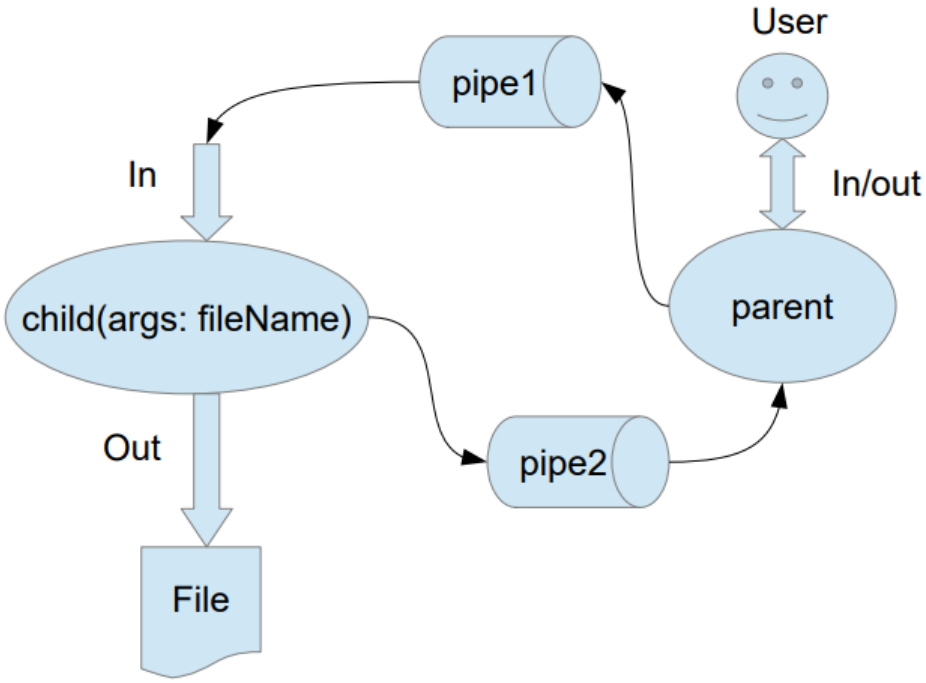
Приобретение практических навыков в:

- 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). Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

Группа вариантов 1



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.

2 вариант) Пользователь вводит команды вида: «число число число». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс считает их сумму и выводит её в файл. Числа имеют тип float. Количество чисел может быть произвольным.

Общие сведения о программе

1. write () – переписывает count байт из буфера в файл. Возвращает количество записанных байт или -1;
2. read () - считывает count байт из файла в буфер. Возвращает количество считаных байт (оно может быть меньше count) или -1;
3. pipe () - создаёт канал между двумя процессами. Создаёт и помещает в массив 2 файловых дескриптора для чтения и для записи. Возвращает 0 или -1;
4. open () - открывает или создаёт файл при необходимости. Возвращает дескриптор открытого файла или -1;
5. close () - закрывает файловый дескриптор, который больше не ссылается ни на один файл, возвращает 0 или -1;
6. fork () - порождается процесс-потомок. Весь код после fork () выполняется дважды, как в процессе-потомке, так и в процессе-родителе. Процесс-потомок и процесс-родитель получают разные коды возврата после вызова fork (). Процесс-родители возвращает идентификатор pid потомка или -1. Процесс-потомок возвращает 0 или -1;
7. dup2 () – переназначение файлового дескриптора, старый и новый файловые дескрипторы являются взаимозаменяемыми, указывают на одно и то же. Возвращает новый дескриптор или -1;
8. execl () - заменяет текущий образ процесса новым образом процесса. Новая программа наследует от вызывавшего процесса идентификатор и открытые файловые дескрипторы.

Общий алгоритм решения

При помощи системного вызова pipe () создаем каналы для передачи данных между родительским процессом и будущим дочерним. Затем при помощи fork () создаем дочерний процесс. fork () возвращает идентификатор созданного процесса (pid): если он равен нулю, то значит, что выполняется дочерний процесс, если он равен -1, то значит, что возникла ошибка создания процесса, в другом случае он будет равен числу, которое больше нуля и которое хранит в себе идентификатор дочернего процесса (pid), следовательно в этот момент будет выполняться родительский процесс. В случае ошибки выводит сообщение об этом и заканчиваем работу программы. Если выполняется родительский процесс, то из потока ввода получаем данные и отправляем через pipe дочернему процессу. Если выполняется дочерний процесс, то через системный вызов execl () запускаем программу заранее дублировав с помощью системного вызова dup2 поток ввода в pipe1, где будут производиться вычисления. Там через pipe получаем данные и считаем сумму чисел. Далее выводим результат в поток ввода, так же дублировав его с файловым дескриптором файла.

Основные файлы программы

main.cpp

```
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <iostream>

using namespace std;

int main() {
    int pipe1[2], pipe2[2];
```

```
    if (pipe(pipe1) == -1 or pipe(pipe2) == -1) {
        cerr << "Pipe error" << endl;
        return 1;
    }
    string filename;
    getline(cin, filename);
    pid_t pid = fork();

    if (pid == -1) {
        cerr << "Fork error" << endl;
        return 1;
    } else if (pid == 0) {
        close(pipe1[1]);
        close(pipe2[0]);

        if (dup2(pipe1[0], fileno(stdin)) == -1 or dup2(pipe2[1], fileno(stdout))
== -1) {
            cerr << "Dup2 error" << endl;
            return 1;
        }

        if (execl("./child_process", "./child_process", filename.c_str(), NULL)
== -1) {
            cerr << "Exec error" << endl;
            return 1;
        }

        close(pipe1[0]);
        close(pipe2[1]);

    } else {
        close(pipe1[0]);
        close(pipe2[1]);

        string line;
        while(getline(cin, line)) {
            line += "\n";
            write(pipe1[1], line.c_str(), line.length());
        }

        close(pipe1[1]);
        close(pipe2[0]);
        wait(nullptr);
    }

    return 0;
}
```

child_process.cpp

```cpp
#include <unistd.h>
#include <fcntl.h>
#include <sstream>
#include <iostream>
#include <stdio.h>

using namespace std;

int main(int argc, char *argv[]) {
    int file_d = open(argv[1], O_CREAT | O_WRONLY | O_TRUNC, S_IRWXU);

    if (file_d == -1) {
        cerr << "Creating file error" << endl;
        return 1;
    }

    if (dup2(file_d, fileno(stdout)) == -1) {
        cerr << "Dup2 error" << endl;
        return 1;
    }
    string line;

    char c = ' ';
    float ans = 0.0, number = 0.0;
    while (scanf("%f%c", &number, &c) > 0) {
        ans += number;
        if (c == '\n') {
            cout << ans << endl;
            ans = 0.0;
        }
    }

    close(file_d);
    return 0;
}
```

Пример работы

Ввод

```
● lw1/build [main•] » ./main
  output.txt
  1 2 3 4 5 6 6 7
  1.5 1.5 1.5
```

Вывод (файл)

```
34
4.5
```

Вывод

В ходе лабораторной работы я узнал о таких системных процессах как: fork, pipe, execl, dup2. Мне кажется, что это достаточно полезные системные вызовы, особенно интересно было узнать про dup2, так как ранее пользовался freopen, а теперь узнал, что в основе лежит системный вызов dup2. fork и execl будут полезны для некой декомпозиции задачи, а также могут способствовать расширению программы: при их помощи задачу можно разбирать на микросервисы, так как работают они независимо друг от друга, что способствует отказоустойчивости, а при помощи pipe’ов можно передавать данные.