Министерство науки и высшего образования

Московский Авиационный Институт
(национальный исследовательский университет)

ЛАБОРАТОРНАЯ РАБОТА №2
по курсе операционные системы I семестр 2022/2023

Студент: Катин Иван Вячеславович
Группа: М8О-210Б-21
Вариант: 9
Преподователь : Миронов Евгений Сергеевич
Оценка:
Дата:
Подпись:

Москва, 2022

Содержание

Постановка задачи **2**

Цель работы **2**

Алгоритм решения **2**

Код программы **3**

Тест кейсы **5**

Вывод **6**

Постановка задачи

Вариант 9.
В файле записаны команды вида:«число число число<endline>». Дочерний процесс производит деление первого числа команда, на последующие числа в команде, а результат выводит в стандартный поток вывода. 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.

Цель раобты

Приобритение практических навыков в:

- Управление процессами в ОС
- Обеспечение обменном данных между процессами посредством каналов

Алгоритм решения

Открываем переданный файл на чтение.Создаем **pipe** для того, чтобы передавать результат из дочернего процесса в родительский.
Создаем дочерний процесс, для дочернего процесса заменяем стандартный поток входа переданным файлом(с помощью **dup2**). Запускаем программу в аргументы передаем файловый дескриптор пайпа на запись. Читаем строку делим и проверяем деление на 0. Передаем результат в **pipe**.
Для родительского процесса. Просто читаем **pipe**, пока читается. Выводим в стандртный поток вывода.

Код программы

Родительский процесс

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

void handle_error(bool expr, char* msg) {
    if (expr) {
        write(fileno(stdout), msg, strlen(msg) * sizeof(char));
        exit(-1);
    }
}

void get_file_name(char* buffer){
    for (int i = 0; i < strlen(buffer); ++i) {
        if(buffer[i] == '\n') {
            buffer[i] = '\0';
            return;
        }
    }
}

int main() {
    int pipe1[2];
    handle_error((pipe(pipe1) == -1), "pipe error");
    char buffer[50];
    handle_error(read(fileno(stdin),buffer, sizeof(buffer)) <=0, "error reading form stdin");
    get_file_name(buffer);
    int file_descriptor = open(buffer, O_RDONLY);
    handle_error(file_descriptor == -1, "Can't open file");
    pid_t pid = fork();
    if(pid == 0){
        close(pipe1[0]);
        handle_error(dup2(file_descriptor, STDIN_FILENO) < 0, "error dub");
        char out[50];
        handle_error(sprintf(out, "%d", pipe1[1]) < 0, "error cast");
        handle_error(execl("child", out, NULL) < 0, "error process");
    } else{
        handle_error( (pid == -1 ), "process error");
        close(pipe1[1]);
        wait(0);
        float result;
        char answer[50];
        while ((read(pipe1[0], &result, sizeof(float))) > 0) {
            handle_error(result == -1, "div 0");
            sprintf(answer, "%f", result);
            handle_error(write(fileno(stdout), answer, strlen(answer)) == -1, "write error");
            handle_error(write(fileno(stdout), "\n", 1) == -1, "write error n");
        }
    }
    return 0;
}
```

Дочерний процесс

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdbool.h>


int main(int argc, char* argv[]){
    int file_descriptor = atoi(argv[0]);
    char character;
    float result;
    bool is_first_number = true;
    while ((read(fileno(stdin), &character, 1)) >0){
        char* buffer = malloc(sizeof(char) * 50);
        int k = 0;
        while(character != ' ' && character != '\n' && character != '\0'){
            buffer[k++] = character;
            if(read(fileno(stdin), &character, 1) <= 0) {
                character = EOF;
                break;
            }
        }
        if(is_first_number){
            is_first_number = false;
            result = strtof(buffer, &buffer);
        }
        else {
            float number = strtof(buffer, &buffer);
            if(number == 0){
                float error = -1;
                write(file_descriptor, &error, sizeof(float));
                exit(-1);
            }
            result /= number;
            if (character == '\n' || character == EOF){
                is_first_number = true;
                write(file_descriptor, &result, sizeof(float));
            }
        }
    }
    close(file_descriptor);
    return 0;
}
```

Тест кейсы

Данные файла

```
1 2 5
4 4 4
5 5 1
5 5 1
23123 32 313 12
23123 32 313 3
```

Результат

```
machine@Turing:~/Desktop/MAI/oos/lab2$ ./run
command
0.100000
0.250000
1.000000
1.000000
0.192384
0.769535
```

Вывод

В ходе лабороторной работы я познакомился с новыми системными вызовами, межпроцессным взаимодействие. **Pipe** удобен тем, что тебе не надо думать, о передаче данных одного процесса другому. Новый процесс я создавал с помощью **fork** и **execl**. С помощью этих системных вызовов удобно управлять процессами, которые выполняют свой участок кода. Однако, надо быть с осторожней с этими вызовами, так как можно по ошибке создать 'fork-бомбу' и допустить множество других ошибок межпроцессного взаимодействия.