Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №2 по курсу**

**«Операционные системы»**

**КАНАЛЫ И ПРОЦЕССЫ???**

Студент: Киреев Александр Константинович

Группа: М8О–206Б–20

Вариант: 4

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2020.

**Постановка задачи**

## Цель работы

Целью работы является 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.

Вариант 4:

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. Пользователь вводит команды вида: «число число число<endline>». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс производит деление первого числа, на последующие, а результат выводит в файл. 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.

**Общие сведения о программе**

Программа компилируется из файла main.c. Также используется заголовочные файлы: stdio.h, unistd.h, stdlib.h, stdbool.h, sys/types.h, sys/stat.h, fcntl.h. В программе используются следующие системные вызовы:

1. **pipe** ­­–– принимает массив из двух целых чисел, в случае успеха массив будет содержать два файловых дескриптора, которые будут использоваться для конвейера, первое число в массиве предназначено для чтения, второе для записи, а так же вернется 0. В случае неуспеха вернется -1.
2. **fork ––** создает новый процесс, который является копией родительского процесса, за исключением разных process ID и parent process ID. В случае успеха fork() возвращает 0 для ребенка, число больше 0 для родителя – child ID, в случае ошибки возвращает -1.
3. **close ––** принимает файловый дескриптор в качестве аргумента, удаляет файловый дескриптор из таблицы дескрипторов, в случае успеха вернет 0, в случае неуспеха вернет -1.
4. **open ––** создает или открывает файл, если он был создан. В качестве аргументов принимает путь до файла, режим доступа (запись, чтение и т.п.), модификатор доступа ( при создании можно указать права для файла ). Возвращает в случае успеха файловый дескриптор – положительное число, иначе возвращает -1.
5. **read ––** предназначена для чтения какого-то числа байт из файла, принимает в качестве аргументов файловый дескриптор, буфер, в который будут записаны данные и число байт. В случае успеха вернет число прочитанных байт, иначе -1.
6. **write ––** предназначена для записи какого-то числа байт в файл, принимает в качестве аргументов файловый дескриптор, буфер, из которого будут считаны данные для записи и число байт. В случае успеха вернет число записанных байт, иначе -1.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы pipe, fork.
2. Написать парсер для типа float, который бы превращал строковое представление числа в естественное.
3. Организовать считывание названия файла и троек чисел типа float.
4. Реализовать функции для процесса-родителя и процесса-ребенка.
5. Реализовать сообщение между процессами при помощи каналов.
6. Реализовать обработку системных ошибок согласно заданию.

**Основные файлы программы**

**main.c:**

// Киреев А.К., 206 группа

// 4 вариант

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <stdbool.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#define BUF\_SIZE 64

// считать имя файла с клавиатуры при помощи read

void read\_file\_name(char file\_name[BUF\_SIZE]) {

char ch;

int count = 0;

while (read(0, &ch, 1) > 0 && ch != '\n') {

file\_name[count++] = ch;

}

file\_name[count] = '\0';

}

// считать float с клавиатуры при помощи read

int read\_float(float\* n) {

char ch;

char buf[100];

int count = 0;

// проверка на еоф (read возвращает 0, если еоф)

int is\_eof = 1;

float sign = 1.0;

while ((is\_eof = read(0, &ch, 1)) > 0 && ch != ' ' && ch != '\n') {

if (ch == '-') {

sign \*= -1;

} else {

buf[count++] = ch;

}

}

buf[count] = '\0';

\*n = atof(buf) \* sign;

return (is\_eof > 0) ? 1 : 0;

}

// считать три float в строке через разделитель

int read\_3\_float(float\* num1, float\* num2, float\* num3) {

// проверка на еоф (если хоть одна функция считывания словила еоф)

int to\_return = 1;

to\_return \*= read\_float(num1);

if (to\_return == 0)

return to\_return;

to\_return \*= read\_float(num2);

if (to\_return == 0)

return to\_return;

to\_return \*= read\_float(num3);

return to\_return;

}

// закрывает канал

void close\_pipe(int pipe[2]) {

close(pipe[0]);

close(pipe[1]);

}

int parent\_proc(int pipe1[2], int pipe2[2]) {

close(pipe1[0]);

close(pipe2[1]);

float num1, num2, num3;

// значение об окончании процесса, которое подается по второму каналу от ребенка к родителю

bool kill\_sig = false;

while (read\_3\_float(&num1, &num2, &num3)) {

write(pipe1[1], &num1, sizeof(float));

write(pipe1[1], &num2, sizeof(float));

write(pipe1[1], &num3, sizeof(float));

read(pipe2[0], &kill\_sig, sizeof(bool));

if (kill\_sig) {

close(pipe1[1]);

close(pipe2[0]);

exit(2);

}

}

close(pipe1[1]);

close(pipe2[0]);

exit(0);

}

int child\_proc(char\* file\_name, int pipe1[2], int pipe2[2]) {

close(pipe1[1]);

close(pipe2[0]);

float num1, num2, num3;

bool kill\_sig = false;

int fd = open(file\_name, O\_WRONLY | O\_CREAT, S\_IWRITE | S\_IREAD);

if (fd < 0) {

perror("Ubable to create file.");

close(pipe1[0]);

close(pipe2[1]);

exit(1);

}

while (read(pipe1[0], &num1, sizeof(float)) > 0 &&

read(pipe1[0], &num2, sizeof(float)) > 0 && read(pipe1[0], &num3, sizeof(float)) > 0) {

// если делим на ноль, то оправляем родителю сообщение прекратить работу

if (num2 \* num3 == 0.0) {

kill\_sig = true;

write(pipe2[1], &kill\_sig, sizeof(bool));

close(fd);

close(pipe1[0]);

close(pipe2[1]);

exit(2);

}

write(pipe2[1], &kill\_sig, sizeof(bool));

float res = num1 / (num2 \* num3);

// конвертируем флоат в строку для записи в файл

char float\_str[100];

sprintf(float\_str, "%f", res);

int count = 0;

while (float\_str[count] != '\0') {

write(fd, &float\_str[count++], 1);

}

char endl = '\n';

write(fd, &endl, 1);

}

close(fd);

close(pipe1[0]);

close(pipe2[1]);

exit(0);

}

int main() {

char file\_name[BUF\_SIZE];

read\_file\_name(file\_name);

int pipe1[2], pipe2[2];

if (pipe(pipe1) != 0) {

perror("Unable to create a pipe1.");

exit(1);

}

if (pipe(pipe2) != 0) {

perror("Unable to create a pipe2.");

close\_pipe(pipe1);

exit(1);

}

int id = fork();

if (id < 0) {

close\_pipe(pipe1);

close\_pipe(pipe2);

perror("Fork error.");

exit(1);

} else if (id == 0) {

child\_proc(file\_name, pipe1, pipe2);

} else {

parent\_proc(pipe1, pipe2);

}

close\_pipe(pipe1);

close\_pipe(pipe2);

exit(0);

}

**Пример работы**

MacBook-Air-K:test AK$ ./wrapper.sh

[info] [2020-10-01 20:50:09] Stage #1 Compiling...

gcc -pedantic -Wall -Werror lab2.c -o lab2

[info] [2020-10-01 20:50:13] Stage #2 Test generating...

[info] [2020-10-01 20:50:13] Stage #3 Checking...

MacBook-Air-K:test AK$ rm file\_\*

MacBook-Air-K:test AK$ rm test\_\*

MacBook-Air-K:test AK$ cd ../src

MacBook-Air-K:src AK$ ./lab2

StrangeFileName

12.4 2.0 2.0

3 1 1.0

123.456 12.33 14.0

1 1 1

0 9 9

12 3 0

MacBook-Air-K:src AK$ ls

Makefile StrangeFileName lab2 lab2.c

MacBook-Air-K:src AK$ cat StrangeFileName

3.100000

3.000000

0.715189

1.000000

0.000000

**Вывод**

Делая данную лабораторную работу, я познакомился с такими вещами, как процессы и каналы. Связанные процессы могут помочь повысить эффективность программы, решающей задачу, в которой есть множество независимых действий. Но основной задачей данной лабораторной работы я считаю работу с каналами. Концепция каналов очень важна, ведь сейчас набирает популярность многопоточность и программы, объединяющие множество процессов, в которых каналы играют ключевую роль . Каналы позволяют передавать информацию между процессами, что позволяет независимым процессам синхронизировать работу, решая общую задачу, с их помощью можно управлять работой процессов, передавая им различные сигналы и сообщения. Эти знания пригодятся мне в написании эффективных программ, использующих идею разделенных процессов.