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: Куценко Б.Д.

Группа: М8О–207Б–20

Вариант: 9

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2021

**Постановка задачи**

**Цель работы**

Приобретение практических навыков в:

* Управление процессами в ОС
* Обеспечение обмена данных между процессами посредством каналов

**Задание**

Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль

родительского процесса пишет имя файла, которое будет передано при создании дочернего

процесса. Родительский процесс передает команды пользователя через pipe1, который связан с

стандартным входным потоком дочернего процесса. Дочерний процесс принеобходимости

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.

**Вариант задания**

В файле записаны команды вида:«число число число». 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.

**Общие сведения о программе**

Программа компилируется из файла lab2.c. . В программе используются следующие системные вызовы:

**fork** - системный вызов в Unix-подобных операционных системах, создающий новый процесс (потомок), который является практически полной копией процесса-родителя, выполняющего этот вызов.Между процессом-потомком, порождаемым вызовом fork(), и процессом-родителем существуют различия:

1) PID процесса-потомка отличен от PID процесса-родителя;

2) значению PPID процесса-потомка присваивается значение PID процесса-родителя;

3) процесс-потомок получает собственную таблицу файловых дескрипторов, являющуюся копией таблицы процесса-родителя на момент вызова fork(); это означает, что открытые файлы наследуются, но если процесс-потомок, например, закроет какой-либо файл, то это не повлияет на таблицу дескрипторов процесса-родителя;

3) для процесса-потомка очищаются все ожидающие доставки сигналы;

4) временная статистика выполнения процесса-потомка в таблицах ОС обнуляется;

5) блокировки памяти и записи, установленные в процессе-родителе, не наследуются.

**pipe** - создаёт однонаправленный канал данных, который можно использовать для взаимодействия между процессами.

**read** - Считывает данные из файла

**write -**  записывает в файл

**Код программы lab2.c**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <string.h>

#include <fcntl.h>

char\* get\_string() {

int len = 0;

int capacity = 1;

char \*s = (char\*) malloc(sizeof(char));

char c = getchar();

while (c != '\n') {

s[(len)++] = c;

if (len >= capacity) {

capacity \*= 2;

s = (char\*) realloc(s, capacity \* sizeof(char));

}

c = getchar();

}

s[len] = '\0';

return s;

}

void division(float numerator, float denominator, float \*res) {

\*res = numerator / denominator;

}

int main() {

int fd[2];

int fd2[2];

if(pipe(fd)) {

printf("ERROR: pipe was failed\n");

exit(EXIT\_FAILURE);

}

if(pipe(fd2)) {

printf("ERROR: pipe was failed\n");

exit(EXIT\_FAILURE);

}

pid\_t pid = fork();

if (pid < 0) {

printf("ERROR: fork was failed\n");

exit(EXIT\_FAILURE);

} else if (pid == 0) { // child

close(fd[1]);

close(fd2[0]);

printf("Child's id %d\n", getpid());

int strSize;

read(fd[0], &strSize, sizeof(strSize));

char\* receivedStr = malloc((strSize + 1) \* sizeof(char));

\*(receivedStr + strSize) = '\0';

read(fd[0], receivedStr, sizeof(char) \* strSize);

close(fd[0]);

int file = open(receivedStr, O\_RDONLY);

if (file == -1) {

printf("ERROR: file wasn't open. Check the name of file.\n");

exit(EXIT\_FAILURE);

}

printf("File(%s) was successfully opened.\n", receivedStr);

free(receivedStr);

dup2(file, STDIN\_FILENO);

close(file);

char str[20] = {'\0'};

dup2(fd2[1], STDOUT\_FILENO);

close(fd2[1]);

float numerator, denominator1, denominator2, res;

while (scanf("%f%f%f", &numerator, &denominator1, &denominator2) != EOF) {

if ((denominator1 == 0) || (denominator2 == 0)) exit(EXIT\_FAILURE);

division(numerator, denominator1, &res);

write(STDOUT\_FILENO, &res, sizeof(res));

division(numerator, denominator2, &res);

write(STDOUT\_FILENO, &res, sizeof(res));

}

close(STDOUT\_FILENO);

exit(0);

} else { // parent

close(fd[0]);

close(fd2[1]);

printf("Parent's id: %d, Child's id: %d\n", getpid(), pid);

printf("Enter the file's name: ");

int c = 0;

char\* filename = get\_string();

int number = strlen(filename);

if (write(fd[1], &number, sizeof(number)) == -1) {

printf("ERROR: cannot write to pipe\n");

exit(EXIT\_FAILURE);

}

if (write(fd[1], filename, sizeof(char) \* number) == -1) {

printf("ERROR: cannot write to pipe\n");

exit(EXIT\_FAILURE);

}

close(fd[1]);

free(filename);

int counter = 1;

float res1, res2;

while(1) {

if (read(fd2[0], &res1, sizeof(res1)) == 0) break;

read(fd2[0], &res2, sizeof(res2));

printf("%d: The result of devision: first %f, second %f\n", counter, res1, res2);

counter++;

}

close(fd2[0]);

}

**Листинг терминала**

sokol@DESKTOP-4ULOTQV:~/os/lab2/right\_variant$ ./a.out

Parent's id: 127, Child's id: 128

Child's id 128

Enter the file's name: test.txt

File(test.txt) was successfully opened.

1: The result of devision: first 0.500000, second 0.333333

2: The result of devision: first 3.000000, second 1.500000

3: The result of devision: first 2.000000, second 2.000000

4: The result of devision: first 1.666667, second 2.500000

5: The result of devision: first 0.000000, second 0.000000

6: The result of devision: first 1.000000, second 2.500000

7: The result of devision: first 1.000000, second 3.000000

8: The result of devision: first 0.250000, second 0.222222

sokol@DESKTOP-4ULOTQV:~/os/lab2/right\_variant$ ./a.out

Parent's id: 129, Child's id: 130

Child's id 130

Enter the file's name: ok.txt

ERROR: file wasn't open. Check the name of file.

**Вывод**

Управление процессами – одна из ключевых задач операционной системы. Обычно ОС сама создаёт необходимые для себя и для других программ процессы, но возникают ситуации, когда пользователю требуется вмешаться в работу системы.

Язык Си при подключении библиотеки unistd.h (для Unix-подобных ОС) обладает возможностью совершать системные вызовы, связанные с вводом/выводом данных, управлением файлами и каталогами и, что самое важное, управлением процессами.

В данной лабораторной работе я познакомился с такими понятиями как процесс, дочерний процесс, родительский процесс, пайп, потоки.

Также, я пользовался некторыми функциями языка Си, такими как: pipe, fork, open, close, read, write etc.

После выполнения данной работы я приобрел новые знания, которые позволят по-другому взглянуть на обычные программы.

Однако не только язык Си способен совершать системные вызовы, связанные с управлением процессами. Похожие библиотеки есть на многих других языках программирования, ведь современное программное обеспечение крайне редко состоит из одного процесса.