Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №2 по курсу**

**«Операционные системы»**

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

Студент: Гаптулхаков Руслан Рамилевич

Группа: М80 – 208Б-19

Вариант: 9

Преподаватель: Миронов Евгений Сергеевич

Дата: 20.04.2021

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2020

1. **Постановка задачи**

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.

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.

1. **Общие сведения о программе**

Программа написана на языке Си в UNIX-подобной операционной системе (Ubuntu). В программе создается дочерний процесс child. Дочерний процесс связан с родительским при помощи отдельного канала pipe. Программа считывает числа и дилит каждое на следующее. Выводит в стандартный поток вывода.

1. **Общий метод и алгоритм решения**

При запуске программы пользователю предлагается ввести имя файла, откуда будут считыватсья данные.

После запуска программы создаются два канала fd1 и fd2, затем создаются два дочерних процесса.

Дочерний процесс закрывают ненужные каналы связи и перенаправляют свой стандартный вывод в созданный файл, а стандартный ввод – через соответствующий pipe. Затем он заменяют свой образ памяти и выполняют программу child, в которой он производит вычисления.

1. **Основные файлы программы**

**main.c**

#include <stdlib.h>

#include <unistd.h>

#include <stdio.h>

#include <fcntl.h>

#include<stdlib.h>

#include <sys/types.h>

#include <sys/wait.h>

#define SIZEBR 2000

int main()

{

    // Reading the name of file of input.

    char name[20];

    printf("Enter name of file:");

    scanf("%s", name);

    // Creating pipe

    int fd[2];

    if (pipe(fd) == -1){

        perror("Error of creating pipe");

        return -1;

    }

    // Creating file

    int file = open(name, O\_RDONLY  | O\_CREAT, 0777);

    if(file == -1){

        perror("Error of opening file!");

        return -1;

    }

    // Creating child process

    int id = fork();

    if (id == -1)

    {

        perror("fork error");

        return -1;

    }

    if (id == 0)

    { // Child process

        int file2 = dup2(file, STDIN\_FILENO);

        if(dup2(fd[1],STDOUT\_FILENO) == -1){

            perror("dup2 err");

            return -1;

        }

        close(file);

        close(fd[0]);

        int err = execlp("./child", "child", NULL);

        if(err == -1){

            perror("Eror of executing child programm");

            return -1;

        }

    }

    else

    { //Parent process

        close(fd[1]);

        float res;

        if(dup2(fd[0], STDIN\_FILENO) == -1){

            perror("dup2 err");

            return -1;

        };

        while(scanf("%f", &res) >  0){

            printf("%f ", res);

        }

        close(file);

        close(fd[0]);

        wait(NULL);

    }

    return 0;

}

**child.c**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <stdbool.h>

#define SIZEBR 2000

int main(){

    // dynamic string

    char\* str = (char\*)malloc(sizeof(char)\*SIZEBR);

    size\_t index = 0;

    char c;

    float res , data[2];

    bool fstvalue = 0;

    while(1){

        // Reading char

        if(read(STDIN\_FILENO, &c, sizeof(char)) == -1){

            perror("Err reading char");

            return -1;

        }

        // chech on

        if((c == ' ') || (c =='\n')){

            if(c == '\n'){

                if( data[1] == 0){

                    perror("Segmentation 0");

                    return -1;

                }

                res = data[0]/data[1];

                printf("%f ",res);

                break;

            }

            if (index > SIZEBR - 1){

                if((str = (char\*)realloc(str,sizeof(char)\*SIZEBR + 1)) == NULL){

                    perror("Err realloc");

                    return -1;

                }

            }

            if(!fstvalue){

                str[index] = '\0';

                data[0] = atof(str);

                fstvalue = 1;

                index = 0;

            }

            else{

                str[index] = '\0';

                data[1] = atof(str);

                // chech on segmentation 0

                if( data[1] == 0){

                    perror("Segmentation 0");

                    return -1;

                }

                res = data[0]/data[1];

                data[0] = data[1];

                printf("%f ",res);

                index = 0;

            }

        }

        else{

            // check on buffer overflow

            if (index > SIZEBR - 1){

                if((str = (char\*)realloc(str,sizeof(char)\*SIZEBR\*2)) == NULL){

                    perror("Err realloc");

                    return -1;

                }

                str[index++] = c;

            }

            else{

                str[index++] = c;

            }

        }

    }

    free(str);

    return 0;

}

1. **Демонстрация работы программы**

**rusya@DESKTOP-93JGKCU:/mnt/c/Users/rusya/Desktop/lab2\_2020 (2)$** ./main

Enter name of file:input2.txt

1.125000 1.142857 1.166667 1.200000 1.250000 1.333333 1.500000 1.000000

**rusya@DESKTOP-93JGKCU:/mnt/c/Users/rusya/Desktop/lab2\_2020 (2**)$ ./main

Enter name of file:input.txt

Segmentation 0: Success

1. 1.166667 1.200000 1.250000 1.333333 1.500000 2.000000

**rusya@DESKTOP-93JGKCU:/mnt/c/Users/rusya/Desktop/lab2\_2020 (2)$**

**rusya@DESKTOP-93JGKCU:/mnt/c/Users/rusya/Desktop/lab2\_2020 (2)$** strace -f -e trace="%process,read,write,dup2,pipe" -o strace\_log.txt ./main

422 execve("./main", ["./main"], 0x7ffcff5935b8 /\* 27 vars \*/) = 0

422 arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7ffe5f882070) = -1 EINVAL (Invalid argument)

422 read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\360q\2\0\0\0\0\0"..., 832) = 832

422 arch\_prctl(ARCH\_SET\_FS, 0x7fbba87ed540) = 0

422 write(1, "Enter name of file:", 19) = 19

422 read(0, "input2.txt\n", 1024) = 11

422 pipe([3, 4]) = 0

422 clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7fbba87ed810) = 423

423 dup2(5, 0) = 0

422 dup2(3, 0 <unfinished ...>

423 dup2(4, 1 <unfinished ...>

422 <... dup2 resumed>) = 0

423 <... dup2 resumed>) = 1

422 read(0, <unfinished ...>

423 execve("./child", ["child"], 0x7ffe5f882158 /\* 27 vars \*/) = 0

423 arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7ffdb37e92f0) = -1 EINVAL (Invalid argument)

423 read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\360q\2\0\0\0\0\0"..., 832) = 832

423 arch\_prctl(ARCH\_SET\_FS, 0x7fe110a63540) = 0

423 read(0, "9", 1) = 1

423 read(0, " ", 1) = 1

423 read(0, "8", 1) = 1

423 read(0, " ", 1) = 1

423 read(0, "7", 1) = 1

423 read(0, " ", 1) = 1

423 read(0, "6", 1) = 1

423 read(0, " ", 1) = 1

423 read(0, "5", 1) = 1

423 read(0, " ", 1) = 1

423 read(0, "4", 1) = 1

423 read(0, " ", 1) = 1

423 read(0, "3", 1) = 1

423 read(0, " ", 1) = 1

423 read(0, "2", 1) = 1

423 read(0, " ", 1) = 1

423 read(0, "1", 1) = 1

423 read(0, "\n", 1) = 1

423 write(1, "1.125000 1.142857 1.166667 1.200"..., 72 <unfinished ...>

422 <... read resumed>"1.125000 1.142857 1.166667 1.200"..., 1024) = 72

423 <... write resumed>) = 72

422 write(1, "1.125000 1.142857 1.166667 1.200"..., 72 <unfinished ...>

423 exit\_group(0 <unfinished ...>

422 <... write resumed>) = 72

423 <... exit\_group resumed>) = ?

422 read(0, "", 1024) = 0

423 +++ exited with 0 +++

422 --- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=423, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=0} ---

422 exit\_group(0) = ?

422 +++ exited with 0 +++

1. **Выводы**

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.

Внутри программы на языке Си можно создать дополнительный, т.н. дочерний процесс, который продолжит выполнение текущей программы параллельно с родительским процессом. Для этого используется функция fork, совершающая соответствующий системный вызов. Удобство в том, что при помощи ветвлений в коде программы можно отделить код родительского процесса от кода, предназначенного для ребёнка. А можно заставить ребёнка запустить другую программу. Для этого предназначено семейство функций exec\*. Обеспечить связь между процессами можно при помощи канала pipe, запрос на создание которого можно также совершить в языке Си.

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.