Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №2 по курсу**

**«Операционные системы»**

Студент: Шубин Г. С.

Группа: М8О–308Б–19

Вариант: 11

Преподаватель: Миронов Е. С.

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**

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.

Child1 переводит строки в верхний регистр. Child2 превращает все пробельные символы в символ «\_».

**Общие сведения о программе**

Программа компилируется из файла lab2.c. Также используется заголовочные файлы: unistd.h, stdio.h , stdlib.h, ctype.h. В программе используются следующие системные вызовы:

1. **fork** - создает копию текущего процесса, который является дочерним процессом для текущего процесса
2. **pipe** - создаёт однонаправленный канал данных, который можно использовать для взаимодействия между процессами.
3. **close** - закрывает файл.
4. **read** - читает количество байт (третий аргумент) из файла с файловым дескриптором (первый аргумент) в область памяти(второй аргумент).
5. **write -** записывает в файл с файловым дескриптором (первый аргумент) из области памяти(второй аргумент) количество байт(третий аргумент).
6. **perror –** вывод сообщения об ошибке.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы fork, pipe, close, read, write.
2. Написать программу, которая будет работать с 3 процессами: один родительский и два дочерних, процессы связываются между собой при помощи pipe.
3. Организовать работу с выделением памяти под строку неопределенной длины и запись длины в массив строки в качестве первого элемента для передачи между процессами через pipe.

**Листинг программы**

**lab2.c:**

#include "unistd.h"

#include "stdio.h"

#include "stdlib.h"

#include "ctype.h"

int main() {

    int fd\_0[2];

    int fd\_1[2];

    int fd\_2[2];

    pipe(fd\_0);

    pipe(fd\_1);

    pipe(fd\_2);

    int pid\_0 = 0;

    int pid\_1 = 0;

    if((pid\_0 = fork()) > 0) {

        if((pid\_1 = fork()) > 0) {

            char\* in = (char\*)malloc(2\*sizeof(char));

            in[0] = 0;

            char c;

            while((c = getchar()) != EOF) {

                in[0] += 1;

                in[in[0]] = c;

                in = (char\*)realloc(in, (in[0]+2)\*sizeof(char));

            }

            in[in[0]+1] = '\0';

            write(fd\_0[1], in, (in[0]+2)\*sizeof(char));

            char\* out = (char\*)malloc((in[0]+2)\*sizeof(char));

            for(int i = 0; i <= in[0]+1; ++i) {

                read(fd\_2[0], &out[i], sizeof(char));

            }

            for(int i = 1; i <= out[0]+1; ++i) {

                printf("%c", out[i]);

            }

            printf("\n");

            close(fd\_2[0]);

            close(fd\_0[1]);

            free(in);

            free(out);

        }

        else if(pid\_1 == 0) {

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

            read(fd\_1[0], &in[0], sizeof(char));

            in = (char\*)realloc(in, (in[0]+2)\*sizeof(char));

            char\* out = (char\*)malloc((in[0]+2)\*sizeof(char));

            for(int i = 1; i <= in[0]+1; ++i) {

                read(fd\_1[0], &in[i], sizeof(char));

            }

            for(int i = 1; i <= in[0]+1; ++i) {

                if(in[i] == ' ') {

                    out[i] = '\_';

                }

                else {

                    out[i] = in[i];

                }

            }

            out[0] = in[0];

            write(fd\_2[1], out, (out[0]+2)\*sizeof(char));

            close(fd\_2[1]);

            close(fd\_1[0]);

            free(in);

            free(out);

        }

        else {

            perror("fork error\n");

            exit(-1);

        }

    }

    else if (pid\_0 == 0) {

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

        read(fd\_0[0], &in[0], sizeof(char));

        in = (char\*)realloc(in, (in[0]+2)\*sizeof(char));

        char\* out = (char\*)malloc((in[0]+2)\*sizeof(char));

        for(int i = 1; i <= in[0]+1; ++i) {

            read(fd\_0[0], &in[i], sizeof(char));

        }

        for(int i = 1; i <= in[0]+1; ++i) {

            out[i] = toupper(in[i]);

        }

        out[0] = in[0];

        write(fd\_1[1], out, (out[0]+2)\*sizeof(char));

        close(fd\_0[0]);

        close(fd\_1[1]);

        free(in);

        free(out);

    }

    else {

        perror("fork error\n");

        exit(-1);

    }

    return 0;

}

**Пример работы**

gregory@DESKTOP-7L8SUG4:~/labsOS/lab2$ gcc lab2.c

gregory@DESKTOP-7L8SUG4:~/labsOS/lab2$ cat test.txt

text 3sp 2sP 4sp

5Sp sp 2sp

text TEXT text4sp

gregory@DESKTOP-7L8SUG4:~/labsOS/lab2$ ./a.out < test.txt

TEXT\_\_\_3SP\_\_2SP\_\_\_\_4SP

\_\_\_\_\_5SP\_SP\_\_2SP

TEXT\_TEXT\_TEXT4SP\_\_\_\_

**Вывод**

Управление процессами­­ — одна из главных задач операционной системы. Изучение распараллеливания процессов является фундаментальным знанием для понимания работы ОС.

В языке Си есть функционал, который позволит нам внутри нашей программы создать дополнительный, дочерний процесс с помощью функции fork. Этот процесс будет работать параллельно с родительским и будет являться его копией. Для этого в языке Си на Unix-подобных ОС используется библиотека unistd.h. Эта библиотека позволяет совершать системные вызовы, которые связаны с вводом/выводом, управлением файлами, каталогами, работой с процессами и запуском программ.