Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №2 по курсу**

**«Операционные системы»**

**Процессы операционных систем**

Студент: Белоносов Кирилл Алексеевич

Группа: М8О–208Б–21

Вариант: 13

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2022.

**Постановка задачи**

## Цель работы

Целью 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.

**Общие сведения о программе**

Программа компилируется из файла main.c. Также используется заголовочные файлы: stdio.h, stdbool.h, stdlib.h, string.h, unistd.h, errorlib.h. В программе используются следующие системные вызовы:

1. **pid\_t fork()** - создание дочернего процесса, возвращает -1 при ошибке создания дочернего процесса, 0 если процесс является дочерним, и pid если процесс является родительским.
2. **int execlp(const char \*file, const char \*arg, ...)** – заменяет текущий образ процесса новым образом процесса, c аргументами arg.
3. **pid\_t waitpid(pid\_t pid, int \*status, int options)** - Ожидание завершения дочернего процесса
4. **void exit(int status)** - завершения выполнения процесса и возвращение статуса
5. **int pipe(int pipefd[2])** - создание неименованного канала для передачи данных между процессами
6. **int dup2(int oldfd, int newfd)** - переназначение файлового дескриптора
7. **int close(int fd)** - закрыть файл

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы fork, pipe, execlp, dup2, waitpid.
2. Написать функцию oerror, для вывода сообщений об ошибках.
3. Написать программу child1 для перевода текста в нижний регистр
4. Написать программу child2 для замены пробельных символов на нижнее подчеркивание
5. Написать программу создающую два дочерних процесса заменяемые child1 и child2 и соединить каналами pipe1 и pipe2

**Основные файлы программы**

**main.c:**

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#include <sys/wait.h>

#include <string.h>

#include "errorlib.h"

int main () {

int fd1[2], fd2[2], fd3[2];

if (pipe(fd1) == -1)

{

oerror("can't create a pipe fd1:", -1);

}

if (pipe(fd2) == -1)

{

oerror("can't create a pipe fd2:", -2);

}

if (pipe(fd3) == -1)

{

oerror("can't create a pipe fd3:", -3);

}

int pid1, pid2;

if((pid1 = fork()) == 0) {

if (dup2(fd1[0], STDIN\_FILENO) == -1)

{

oerror("can't duplicate descriptor pipe 1:", -1);

}

if (dup2(fd3[1], STDOUT\_FILENO) == -1)

{

oerror("can't duplicate descriptor pipe 2:", -2);

}

if (close(fd1[1]) == -1)

{

oerror("can't close pipe 1 write", -3);

}

if (close(fd3[0]) == -1)

{

oerror("can't close pipe 3 read:", -4);

}

if (close(fd2[1]) == -1)

{

oerror("can't close pipe 3 read:", -5);

}

if (close(fd2[0]) == -1)

{

oerror("can't close pipe 3 read:", -6);

}

if(execlp("./child1", "child1", NULL) == -1) {

oerror("can't open file child1:", -7);

}

}

if(pid1 > 0 && (pid2 = fork()) == 0) {

if (dup2(fd2[1], STDOUT\_FILENO) == -1)

{

oerror("can't duplicate descriptor pipe 2:", -1);

}

if (dup2(fd3[0], STDIN\_FILENO) == -1)

{

oerror("can't duplicate descriptor pipe 3:", -2);

}

if (close(fd2[0]) == -1)

{

oerror("can't close pipe 2 read", -3);

}

if (close(fd3[1]) == -1)

{

oerror("can't close pipe 3 write:", -4);

}

if (close(fd1[1]) == -1)

{

oerror("can't close pipe 3 read:", -5);

}

if (close(fd1[0])== -1)

{

oerror("can't close pipe 3 read:", -6);

}

if(execlp("./child2", "child2", NULL) == -1) {

oerror("can't open file child2:", -7);

}

}

if(pid1 == -1 || pid2 == -1) {

fprintf(stderr, "Can't create a child process");

exit(-1);

}

if(pid1 == -1) {

oerror("can't create process child1:", -4);

}

if(pid2 == -1) {

oerror("can't create process child2:", -5);

}

if(pid1 != 0 && pid2 != 0) {

char c;

if(close(fd1[0]) == -1) {

oerror("can't close pipe 1 read:", -6);

}

if(close(fd2[1]) == -1) {

oerror("can't close pipe 1 read:", -7);

}

int err;

while(err = read(0, &c, 1) > 0) {

if(write(fd1[1], &c, 1) == -1) {

oerror("can't write in pipe 1:", -9);

}

if(read(fd2[0], &c, 1) == -1) {

oerror("can't read from pipe 2:", -10);

}

if(write(1, &c, 1) == -1) {

oerror("can't write in stdout:", -11);

}

}

if(err == -1) {

oerror("can't write in pipe 1:", -8);

}

close(fd1[1]);

close(fd2[0]);

close(fd3[1]);

close(fd3[0]);

waitpid(-1, NULL, 0);

}

return 0;

}

**child1.c**

#include<unistd.h>

#include<stdio.h>

#include<stdlib.h>

#include"errorlib.h"

int main() {

char c;

while(read(0, &c, 1) > 0) {

if(c >= 'A' && c <= 'Z') {

c = c + 32;

}

if(write(1, &c, 1) == -1) {

oerror("can't write the pipe 3 by child1", -1);

}

}

return 0;

}

**child2.c**

#include<unistd.h>

#include<stdio.h>

#include<stdlib.h>

#include"errorlib.h"

int main() {

char c;

while(read(0, &c, 1) > 0) {

if(c == ' ') {

c = '\_';

}

if(write(1, &c, 1) == -1) {

oerror("can't write the pipe 2 by child2", -1);

}

}

return 0;

}

**errorlib.c**

#include "errorlib.h"

int oerror(const char \* error, int id) {

write(STDERR\_FILENO, error, strlen(error));

exit(id);

}

**errorlib.h**

#ifndef ERRORLIB\_H

#define ERRORLIB\_H

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <unistd.h>

int oerror(const char \* error, int id);

#endif

**Пример работы**

**1.txt:**

ThE quick browNNN

f0x jumps .OvEr. the l@zy """dog"""

**2.txt:**

SwEEt dreams are made

of this

Who AM

$#$#$ <>><>>

I to disagree

I TRavel the WORLD\_\_\_\_and\_\_\_\_\_\_the seven seas

Everybody's looking FOR something

**Output:**

-------------------------test 1-------------------------

the\_quick\_brownnn\_\_\_\_\_\_\_\_

f0x\_jumps\_.over.\_the\_l@zy\_"""dog"""\n

-------------------------test 2-------------------------

sweet\_\_\_\_\_\_dreams\_\_\_\_\_\_are\_\_\_\_\_\_\_\_\_\_\_\_\_\_made\_

of\_this

\_\_\_\_\_\_\_\_who\_\_\_\_\_am

\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_$#$#$\_<>><>>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_i\_to\_\_\_\_\_disagree

i\_\_\_travel\_the\_\_\_\_world\_\_\_\_and\_\_\_\_\_\_the\_seven\_\_\_\_seas

\_\_\_\_\_\_\_\_everybody's\_\_\_\_\_\_\_looking\_for\_something

**Вывод**

В результате данной лабораторной работы были изучены основные методы работы с процессами в ОС linux. В данной работе я научился создавать процессы, работать с родительскими и дочерними процессами, передавать между ними данные с помощью каналов. Полученные знания пригодятся не только при выполнении дальнейших лабораторных работ, но и в дальнейшей работе.