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: Е. А. Айрапетова

Группа: М8О–206Б–19

Вариант: 16

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2020.

**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.

Вариант:

Родительский процесс создает дочерний процесс. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Родительский и дочерний процесс должны быть представлены разными программами.

Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child проверяет строки на валидность правилу. Если строка соответствует правилу, то она выводится в стандартный поток вывода дочернего процесса, иначе в pipe2 выводится информация об ошибке. Родительский процесс полученные от child ошибки выводит в стандартный поток вывода.

Правило проверки: строка должна начинаться с заглавной буквы.

**Общие сведения о программе**

Программа компилируется из файла lab2.c. Также используется заголовочные файлы: stdio.h, unistd.h, sys/wait.h, stdlib.h, stdbool.h, fcntl.h. В программе используются следующие системные вызовы:

* pipe - создает канал и возвращает два файловых дескриптора, для взаимодействия по каналу;
* close – закрывает файловый дескриптор;
* fork – создает дочерний процесс и продолжает выполнение текущей программы в нем;

На вход программе поступет неограниченное количесво строк. Программа завершает свою работу при нажатии Ctrl+D.

Обрабатываются все возможные системные ошибки и в случае их возниконовения выводятся соответствующие сообщения.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Написать функцию для родительского процесса
2. Написать функцию для дочернего процесса
3. В функции main реализовать обработку входных данных, создание процессов с помощью fork() и передачу информации с помощью потоков между процессами с помощью системного вызова pipe().

**Основные файлы программы**

**lab2.c**

#include <stdio.h>

#include <string.h>

#include <unistd.h>

#include <sys/wait.h>

#include <stdlib.h>

#include <stdbool.h>

#include <fcntl.h>

//родительский процесс

void parent(int \*pipe1, int \*pipe2, int fd){

close(pipe1[0]);

close(pipe2[1]);

char\* errorMess = "ERROR\n";

while(1){

char str[100] = {'\0'};

fgets(str, 100, stdin);

int size = strlen(str);

if (size == 0) {

write(pipe1[1], &size, sizeof(int));

break;

}

if (size == 1) {

if (str[0] == '\n') {

continue;

}

}

str[size - 1] = '\0';

write(pipe1[1], &size, sizeof(int));

write(pipe1[1], str, sizeof(char) \* size);

bool result;

if (read(pipe2[0], &result, sizeof(bool)) != sizeof(bool)) {

fprintf(stderr, "read error in parent process\n");

close(fd);

close(pipe1[1]);

close(pipe2[0]);

exit(-1);

}

if(!result){

write(fd, errorMess, sizeof(char) \* 6);

}

}

wait(NULL);

close(fd);

close(pipe1[1]);

close(pipe2[0]);

}

//дочерний процесс

void child(int \*pipe1, int \*pipe2, int fd){

close(pipe1[1]);

close(pipe2[0]);

while (1) {

int size;

bool result;

char str[100];

if (read(pipe1[0], &size, sizeof(int)) != sizeof(int)) {

fprintf(stderr, "read error in child process\n");

close(fd);

close(pipe1[0]);

close(pipe2[1]);

exit(-1);

}

if (size == 0) {

break;

}

if (read(pipe1[0], str, sizeof(char) \* size) != sizeof(char) \* size) {

fprintf(stderr, "read error in child process\n");

close(fd);

close(pipe1[0]);

close(pipe2[1]);

exit(-2);

}

if(str[0] < 'A' || str[0] > 'Z' || size == 0){

result = false;

write(pipe2[1], &result, sizeof(bool));

} else {

result = true;

write(pipe2[1], &result, sizeof(bool));

write(fd, str, sizeof(char) \* size);

write(fd, "\n", sizeof(char));

}

}

close(fd);

close(pipe1[0]);

close(pipe2[1]);

}

int main(){

char filename[30] = {'\0'};

fgets(filename, 30, stdin);

size\_t size = strlen(filename);

if (size == 0 || (size == 1 && filename[0] == '\n')) {

fprintf(stderr, "incorrect file name\n");

exit(-1);

}

filename[size - 1] = '\0';

int fd = open(filename, O\_CREAT | O\_APPEND | O\_WRONLY, S\_IWUSR | S\_IRUSR);

int fd1[2], fd2[2];

int pid;

if(pipe(fd1) == -1){

perror("pipe1 error");

}

if(pipe(fd2) == -1){

perror("pipe2 error");

}

pid = fork();

if(pid == -1){

perror("FORK");

} else if(pid == 0){

child(fd1, fd2, fd);

} else {

parent(fd1, fd2, fd);

}

return 0;

}

**Пример работы**

jane@Evgenia:/mnt/c/Files/ОСи$ ./a.out

f.txt

String

Also string

wrong string

2 wrong string

jane@Evgenia:/mnt/c/Files/ОСи$ cat f.txt

String

Also string

ERROR

ERROR

jane@Evgenia:/mnt/c/Files/ОСи$ ./a.out

f.txt

jane@Evgenia:/mnt/c/Files/ОСи$ cat f.txt

jane@Evgenia:/mnt/c/Files/ОСи$

**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.